Annie Foret · Glyn Morrill
Reinhard Muskens · Rainer Osswald
Sylvain Pogodalla (Eds.)

LNCS 9804

Formal Grammar

20th and 21st International Conferences
FG 2015, Barcelona, Spain, August 2015, Revised Selected Papers
FG 2016, Bozen, Italy, August 2016, Proceedings

Springer

Lecture Notes in Computer Science **9804**

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

FoLLI Publications on Logic, Language and Information
Subline of Lectures Notes in Computer Science

More information about this series at http://www.springer.com/series/7407

Annie Foret · Glyn Morrill
Reinhard Muskens · Rainer Osswald
Sylvain Pogodalla (Eds.)

Formal Grammar

20th and 21st International Conferences
FG 2015, Barcelona, Spain, August 2015, Revised Selected Papers
FG 2016, Bozen, Italy, August 2016, Proceedings

Springer

Editors
Annie Foret
IRISA
University of Rennes 1
Rennes
France

Glyn Morrill
Department of Computer Science
Universitat Politècnica de Catalunya
Barcelona
Spain

Reinhard Muskens
Tilburg University
Tilburg
The Netherlands

Rainer Osswald
Department of General Linguistics
Heinrich-Heine-University Düsseldorf
Düsseldorf
Germany

Sylvain Pogodalla
INRIA Nancy
Villers-lès-Nancy
France

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-662-53041-2 ISBN 978-3-662-53042-9 (eBook)
DOI 10.1007/978-3-662-53042-9

Library of Congress Control Number: 2016945939

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag GmbH Berlin Heidelberg

Preface

The Formal Grammar conference series (FG) provides a forum for the presentation of new and original research on formal grammar, mathematical linguistics, and the application of formal and mathematical methods to the study of natural language. Themes of interest include, but are not limited to:

- Formal and computational phonology, morphology, syntax, semantics, and pragmatics
- Model-theoretic and proof-theoretic methods in linguistics
- Logical aspects of linguistic structure
- Constraint-based and resource-sensitive approaches to grammar
- Learnability of formal grammar
- Integration of stochastic and symbolic models of grammar
- Foundational, methodological, and architectural issues in grammar and linguistics
- Mathematical foundations of statistical approaches to linguistic analysis

Previous FG meetings were held in Barcelona (1995), Prague (1996), Aix-en-Provence (1997), Saarbrücken (1998), Utrecht (1999), Helsinki (2001), Trento (2002), Vienna (2003), Nancy (2004), Edinburgh (2005), Malaga (2006), Dublin (2007), Hamburg (2008), Bordeaux (2009), Copenhagen (2010), Ljubljana (2011), Opole (2012), Düsseldorf (2013), and Tübingen (2014).

FG 2015, the 20th Conference on Formal Grammar, was held in Barcelona during August 8–9, 2015. The proceedings comprise two invited contributions, by Robin Cooper and Tim Fernando, and nine contributed papers selected from 13 submissions. The present volume includes the two invited papers and eight revised versions of the contributed papers.

FG 2016, the 21st Conference on Formal Grammar, was held in Bolzano-Bozen during August 20–21, 2016. The conference comprised two invited talks, by Frank Drewes and Mehrnoosh Sadrzadeh, and eight contributed papers selected from 11 submissions. The present volume includes the contributed papers.

We would like to thank the people who made the 20th and 21th FG conferences possible: the invited speakers, the members of the Program Committees, and the organizers of ESSLLI 2015 and ESSLLI 2016, with which the conferences were colocated.

June 2016

Annie Foret
Glyn Morrill
Reinhard Muskens
Rainer Osswald
Sylvain Pogodalla

FG 2015 Organization

Program Committee

Alexander Clark	King's College London, UK
Berthold Crysmann	CNRS - LLF (UMR 7110), Paris-Diderot, France
Denys Duchier	Université d'Orléans, France
Philippe de Groote	Inria Nancy – Grand Est, France
Nissim Francez	Technion - IIT, Israel
Laura Kallmeyer	Heinrich-Heine-Universität Düsseldorf, Germany
Makoto Kanazawa	National Institute of Informatics, Japan
Gregory Kobele	University of Chicago, USA
Robert Levine	Ohio State University, USA
Wolfgang Maier	Heinrich-Heine-Universität Düsseldorf, Germany
Stefan Müller	Freie Universität Berlin, Germany
Mark-Jan Nederhof	University of St Andrews, UK
Gerald Penn	University of Toronto, Canada
Christian Retoré	Université de Montpellier and LIRMM-CNRS, France
Manfred Sailer	Goethe University Frankfurt, Germany
Edward Stabler	UCLA and Nuance Communications, USA
Jesse Tseng	CNRS, France
Oriol Valentín	Universitat Politècnica de Catalunya, Spain

Standing Committee

Annie Foret	IRISA, University of Rennes 1, France
Glyn Morrill	Universitat Politècnica de Catalunya, Spain
Reinhard Muskens	Tilburg Center for Logic and Philosophy of Science, The Netherlands
Rainer Osswald	Heinrich-Heine-Universität Düsseldorf, Germany

FG 2016 Organization

Program Committee

Raffaella Bernardi	Free University of Bozen-Bolzano, Italy
Alexander Clark	King's College London, UK
Berthold Crysmann	CNRS - LLF (UMR 7110), Paris-Diderot, France
Philippe de Groote	Inria Nancy – Grand Est, France
Nissim Francez	Technion - IIT, Israel
Laura Kallmeyer	Heinrich-Heine-Universität Düsseldorf, Germany
Makoto Kanazawa	National Institute of Informatics, Japan
Gregory Kobele	University of Chicago, USA
Robert Levine	Ohio State University, USA
Wolfgang Maier	Heinrich-Heine-Universität Düsseldorf, Germany
Stefan Müller	Freie Universität Berlin, Germany
Mark-Jan Nederhof	University of St Andrews, UK
Christian Retoré	Université de Montpellier and LIRMM-CNRS, France
Mehrnoosh Sadrzadeh	Queen Mary University of London, UK
Manfred Sailer	Goethe University Frankfurt, Germany
Edward Stabler	UCLA and Nuance Communications, USA
Jesse Tseng	CNRS, France
Oriol Valentín	Universitat Politècnica de Catalunya, Spain

Standing Committee

Annie Foret	IRISA, University of Rennes 1, France
Reinhard Muskens	Tilburg Center for Logic and Philosophy of Science, The Netherlands
Rainer Osswald	Heinrich-Heine-Universität Düsseldorf, Germany
Sylvain Pogodalla	LORIA/Inria Nancy – Grand Est, France

Contents

Formal Grammar 2015:
Invited Papers

Frames as Records

Robin Cooper(✉)

Department of Philosophy, Linguistics and Theory of Science,
University of Gothenburg, Box 200, 405 30 Göteborg, Sweden
cooper@ling.gu.se
http://www.ling.gu.se/∼cooper

Abstract. We suggest a way of formalizing frames using records in type theory. We propose an analysis of frames as records which model situations (including events) and we suggest that frame types (record types) are important in both the analysis of the Partee puzzle concerning rising temperatures and prices and in the analysis of quantification which involves counting events rather than individuals likes passengers or ships passing through a lock.

Our original inspiration for frames comes from the work of [13,14] and work on FrameNet (https://framenet.icsi.berkeley.edu). An important aspect of our approach to frames, which differs from the Fillmorean approach, is that we treat them as first class objects. That is, they can be arguments to predicates and can be quantified over. The proposal that we have made for solving the Partee puzzle is closely related to the work of [22,23] whose inspiration is from the work of [1–3] rather than Fillmore.

Keywords: Frames · Type theory · Record types · Events · Situations

1 Introduction

In this paper[1] we will suggest a way of formalizing frames using records in type theory and apply this to two phenomena: the "Partee puzzle" concerning rising temperatures and prices for which Montague [24] used individual concepts (functions from possible worlds and times to individuals) and the problem of apparent quantification over events rather than individuals in sentences like (1) discussed by Krifka [18] among others.

(1) Four thousand ships passed through the lock

Our leading idea is to model frames as records and the *roles* in frames (or *frame elements* in the terminology of FrameNet) as fields in records.

[1] An expanded version of this paper with a more detailed formal development is available as Chap. 5 of a book draft [7]. This book draft also gives a general introduction to TTR (the type theory with records that we are using here). For a published introduction see [6].

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 3–18, 2016.
DOI: 10.1007/978-3-662-53042-9_1

Records are in turn what we use to model situations so frames and situations in our view turn out to be the same. Given that we are working in a type theory which makes a clear distinction between types and the objects which belong to those types it is a little unclear whether what we call frame should be a record or a record type. We need both and we will talk of frames (records) and frame types (record types). For example, when we look up the frame Ambient_temperature (https://framenet2.icsi.berkeley.edu/fnReports/data/frameIndex.xml?frame=Ambient_temperature) in FrameNet we will take that to be an informal description of a frame type which can be instantiated by the kinds of situations which are described in the examples there. In our terms we can characterize a type corresponding to a very stripped down version of FrameNet's Ambient_temperature which is sufficient for us to make the argument we wish to make. This is the type *AmbTempFrame* defined in (2).[2]

$$(2) \quad \begin{bmatrix} \mathrm{x} & : \mathit{Real} \\ \mathrm{loc} & : \mathit{Loc} \\ \mathrm{e} & : \mathrm{temp}(\mathrm{loc}, \mathrm{x}) \end{bmatrix}$$

A record, r, will be of this type just in case in contains three fields with the labels 'x', 'loc' and 'e' and the objects in these fields are of the types *Real*, *Loc* and temp(r.loc,r.x), that is, a type whose witness is a proof-object (a situation) which shows that the real number in the 'x'-field of r (r.x) is the temperature at the location r.loc. r may contain more fields than those required by the type. A record may not contain more than one field with a given label.

In order to characterize temperature changes we will introduce a notion of scale relating to frames. A scale is a function which maps frames (situations) to a real number. Thus a scale for ambient temperature will be of the type (3a) and the obvious function to choose of that type is the function in (3b) which maps any ambient temperature frame to the real number in its 'x'-field.

(3) a. $(\mathit{AmbTempFrame} \rightarrow \mathit{Real})$
b. $\lambda r{:}\mathit{AmbTempFrame} \; . \; r.\mathrm{x}$

Let us call (3b) ζ_{temp}. As a first approximation we can take an event of a temperature rise to be a string[3] of two temperature frames, $r_1^\frown r_2$, where $\zeta_{\mathrm{temp}}(r_1) < \zeta_{\mathrm{temp}}(r_2)$. Using a notation where T^n is the type of strings of length n each of whose members are of type T and where for a given string, s, $s[0]$ is the first member of s, $s[1]$ the second and so on, a first approximation to the type of temperature rises could be (4).

$$(4) \quad \begin{bmatrix} \mathrm{e} & : \mathit{AmbTempFrame}^2 \\ \mathrm{c_{rise}} & : \zeta_{\mathrm{temp}}(\mathrm{e}[0]) < \zeta_{\mathrm{temp}}(\mathrm{e}[1]) \end{bmatrix}$$

[2] Our treatment of the Partee puzzle here represents an improvement over the proposal presented in Cooper [6] in that it allows a more general treatment of the type of rising events.

[3] The idea of events as strings is taken from an important series of papers by Fernando, [8–12] among others.

In the c_{rise}-field of (4) we are using $<$ as an infix notation for a predicate 'less-than' with arity $\langle Real, Real\rangle$ which obeys the constraint in (5).

(5) less-than(n, m) is non-empty ("true") iff $n < m$

A more general type for temperature rises is given by (6) where we abstract away from the particular temperature scale used by introducing a field for the scale into the record type. This, for example, allows for an event to be a temperature rise independent of whether it is measured on the Fahrenheit or Celsius scales.

$$(6) \quad \begin{bmatrix} \text{scale} : (AmbTempFrame \rightarrow Real) \\ \text{e} \quad\;\; : AmbTempFrame^2 \\ c_{rise} \;\; : \text{scale(e[0])} < \text{scale(e[1])} \end{bmatrix}$$

This type, though, is now too general to count as the type of temperature rising events. To be of this type, it is enough for there to be some scale on which the rise condition holds and the scale is allowed to be any arbitrary function from temperature frames to real numbers. Of course, it is possible to find some arbitrary function which will meet the rise condition even if the temperature is actually going down. For example, consider a function which returns the number on the Celsius scale but with the sign (plus or minus) reversed making temperatures above 0 to be below 0 and *vice versa*. There are two ways we can approach this problem. One is to make the type in the scale-field a subtype of $(AmbTempFrame \rightarrow Real)$ which limits the scale to be one of a number of standardly accepted scales. This may be an obvious solution in the case of temperature where it is straightforward to identify the commonly used scales. However, scales are much more generally used in linguistic meaning and people create new scales depending on the situation at hand. This makes it difficult to specify the nature of the relevant scales in advance and we therefore prefer our second way of approaching this problem.

The second way is to parametrize the type of temperature rising events. By this we mean using a dependent type which maps a record providing a scale to a record type modelling the type of temperature rising events according to that scale. The function in (7) is a dependent type which is related in an obvious way to the record type in (6).

$$(7) \quad \lambda r{:}\left[\text{scale}{:}(AmbTempFrame \rightarrow Real)\right] . \begin{bmatrix} \text{e} \quad\;\; : AmbTempFrame^2 \\ c_{rise} \;\; : r.\text{scale(e[0])} < r.\text{scale(e[1])} \end{bmatrix}$$

According to (6) an event will be a temperature rise if there is some scale according to which the appropriate relation holds between the temperatures of the two stages of the event which we are comparing. According to (7) on the other hand, there is no absolute type of a temperature rise. We can only say whether an event is a temperature rise with respect to some scale or other. If we choose some

non-standard scale like the one that reverses plus and minus temperatures as we suggested above then what we normally call a fall in temperature will in fact be a rise in temperature *according to that scale*. You are in principle allowed to choose whatever scale you like, though if you are using the type in a communicative situation you had better make clear to your interlocutor what scale you are using and perhaps also why you are using this scale as opposed to one of the standardly accepted ones. The dependent types introduce a presupposition-like component to communicative situations. We are assuming the existence of some scale in the context.

Why do we characterize the domain of the function in (7) in terms of records containing a scale rather than just scales as in (8)?

(8) $\lambda\sigma{:}(\mathit{AmbTempFrame} \rightarrow \mathit{Real}).\begin{bmatrix} \mathrm{e} & : \mathit{AmbTempFrame}^2 \\ \mathrm{c_{rise}} & : \sigma(\mathrm{e}[0]) < \sigma(\mathrm{e}[1]) \end{bmatrix}$

The intuitive reason is that we want to think of the arguments to such functions as being contexts, that is situations (frames) modelled as records. The scale will normally be only one of many informational components which can be provided by the context and the use of a record type allows for there to be more components present. In practical terms of developing an analysis it is useful to use a record type to characterize the domain even if we have only isolated one parameter since if further analysis should show that additional parameters are relevant this will mean that we can add fields to the domain type thereby restricting the domain of the function rather than giving it a radically different type.

And indeed in this case we will now show that there is at least one more relevant parameter that needs to be taken account of before we have anything like a reasonable account of the type of temperature rise events. In (2) we specified that an ambient temperature frame relates a real number ("the temperature") to a spatial location. And now we are saying that a temperature rise is a string of two such frames where the temperature is higher in the second frame. But we have not said anything about how the locations in the two frames should be related. For example, suppose I have a string of two temperature frames where the location in the first is London and the location in the second is Marrakesh. Does that constitute a rise in temperature (assuming that the temperature in the second frame is higher than the one in the first)? Certainly not a temperature rise in London, nor in Marrakesh. If you want to talk about a temperature rise in a particular location then both frames have to have that location and we need a way of expressing that restriction. Of course, you can talk about temperature rises which take place as you move from one place to another and which therefore seem to involve distinct locations. However, it seems that even in these cases something has to be kept constant between the two frames. One might analyse it in terms of a constant path to which both locations have to belong or as a constant relative location such as the place where a particular person (or car, or airplane) is. You cannot just pick two arbitrary temperature

frames without holding something constant which ties them together. We will deal here with the simple case where the location is kept constant.[4] We will say that the background information for judging an event as a temperature rise has to include not only a scale but also a location which is held constant in the two frames. This is expressed in (9).

$$(9) \quad \lambda r{:}\begin{bmatrix}\text{fix}{:}\big[\text{loc}{:}\mathit{Loc}\big]\\ \text{scale}{:}(\mathit{AmbTempFrame} \rightarrow \mathit{Real})\end{bmatrix}.\ \begin{bmatrix}\text{e} & : (\mathit{AmbTempFrame} \wedge\!\!\!\cdot\ \big[\text{loc}{=}r.\text{fix.loc}{:}\mathit{Loc}\big])^2\\ \text{c}_{\text{rise}} & : r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$$

Here we have introduced two new pieces of notation. The symbol '$\wedge\!\!\!\cdot$' represents that two record types are to be *merged*, an operation which corresponds to unification of feature structures. Essentially, $T_1 \wedge\!\!\!\cdot\ T_2$ is a record type T_3 such that for any record r, $r : T_3$ iff $r : T_1$ and $r : T_2$. We also introduce a *manifest field* $\big[\text{loc}{=}r.\text{fix.loc}{:}\mathit{Loc}\big]$. A manifest field $\big[\ell{=}a{:}T\big]$ is a convenient notation for $\big[\ell{:}T_a\big]$ where T_a is a *singleton type* such that $b : T_a$ iff $a : T$ and $b = a$. Thus a manifest field in a type specifies what the value in the corresponding record must be. Precise definitions of these concepts can be found in [6,7]. Here we give an example of the merge of *AmbTempFrame* (spelled out in (10a)) and $\big[\text{loc}{=}r.\text{fix.loc}{:}\mathit{Loc}\big]$ which is identical to (10b).

$$(10) \quad \text{a.} \quad \begin{bmatrix}\text{x} & : \mathit{Real}\\ \text{loc} & : \mathit{Loc}\\ \text{e} & : \text{temp(loc, x)}\end{bmatrix} \wedge\!\!\!\cdot\ \big[\text{loc}{=}r.\text{fix.loc}{:}\mathit{Loc}\big]$$

$$\text{b.} \quad \begin{bmatrix}\text{x} & : \mathit{Real}\\ \text{loc}{=}r.\text{fix.loc} & : \mathit{Loc}\\ \text{e} & : \text{temp(loc, x)}\end{bmatrix}$$

The 'fix'-field in the domain type of (9) is required to be a record which provides a location. One reason for making the 'fix'-field a record rather than simply a location is that we will soon see an example where more than one parameter needs to be fixed. It will also help us ultimately in characterizing a general type for a rising event (not just a rise in temperature) if we can refer to the type in the 'fix'-field as *Rec* ("record") rather than to list a disjunction of all the various types of the parameters that can be held constant in different cases.

The temperature rise event itself is now required to be a string of two frames which belong to a subtype of *AmbTempFrame*, namely where the 'loc'-field has been made manifest and is specified to have the value specified for 'loc' in the 'fix'-field. Here we are using the record in the 'fix'-field of the argument to the function to partially specify the type *AmbTempFrame* by fixing values for some

[4] Although in astronomical terms, of course, even a location like London is a relative location, that is, where London is according to the rotation of the earth and its orbit around the sun. Thus the simple cases are not really different from the cases apparently involving paths.

of its fields. One can think of the 'fix'-record as playing the role of a partial assignment of values to fields in the type. To emphasize this important role and to facilitate making general statements without having to name the particular fields involved, we shall introduce an operation which maps a record type, T, and a record, r to the result of specifying T with r, which we will notate as $T \parallel r$. (11) provides an abstract example of how it works.

$$(11)\quad \begin{bmatrix}\ell_1{:}T_1\\ \ell_2{:}T_2\\ \ell_3{:}T_3\end{bmatrix} \parallel \begin{bmatrix}\ell_2{=}a\\ \ell_3{=}b\\ \ell_4{=}c\end{bmatrix} = \begin{bmatrix}\ell_1{:}T_1\\ \ell_2{=}a{:}T_2\\ \ell_3{=}b{:}T_3\end{bmatrix}$$
$$\text{provided that } a : T_2 \text{ and } b : T_3$$

In a case where for example $a : T_2$ but not $b : T_3$ we would have (12).

$$(12)\quad \begin{bmatrix}\ell_1{:}T_1\\ \ell_2{:}T_2\\ \ell_3{:}T_3\end{bmatrix} \parallel \begin{bmatrix}\ell_2{=}a\\ \ell_3{=}b\\ \ell_4{=}c\end{bmatrix} = \begin{bmatrix}\ell_1{:}T_1\\ \ell_2{=}a{:}T_2\\ \ell_3{:}T_3\end{bmatrix}$$

Using this notation we can now rewrite (9) as (13).

$$(13)\quad \lambda r{:}\begin{bmatrix}\text{fix}{:}\big[\text{loc}{:}\mathit{Loc}\big]\\ \text{scale}{:}(\mathit{AmbTempFrame} \rightarrow \mathit{Real})\end{bmatrix}.\ \begin{bmatrix}\text{e} & : (\mathit{AmbTempFrame} \| r.\text{fix})^2\\ \text{c}_{\text{rise}} & : r.\text{scale}(\text{e}[0]) < r.\text{scale}(\text{e}[1])\end{bmatrix}$$

This is still a very simple theory of what a temperature rise event may be but it will be sufficient for our current purposes. We move on now to price rise events. We will take (14) to be the type of price frames, *PriceFrame*.

$$(14)\quad \begin{bmatrix}\text{x} & : \mathit{Real}\\ \text{loc} & : \mathit{Loc}\\ \text{commodity} & : \mathit{Ind}\\ \text{e} & : \text{price(commodity, loc, x)}\end{bmatrix}$$

The fields represented here are based on a much stripped down version of the FrameNet frame `Commerce_scenario` where our 'commdodity'-field corresponds to the frame element called 'goods' and the 'x'-field corresponds to the frame element 'money'. A price rise is a string of two price frames where the value in the 'x'-field is higher in the second. Here, as in the case of a temperature rise, we need to keep the location constant. It does not make sense to say that a price rise has taken place if we compare a price in Marrakesh with a price in London, even though the price in London may be higher. In the case of price we also need to keep the commodity constant, something that does not figure at all in ambient temperature. We cannot say that a price rise has taken place if we have the price of tomatoes in the first frame and the price of oranges in the second

frame. Thus, following the model of (13), we can characterize the dependent type of price rises as (15).

$$(15)\quad \lambda r{:}\begin{bmatrix}\text{fix}{:}\begin{bmatrix}\text{loc}{:}\mathit{Loc}\\ \text{commodity}{:}\mathit{Ind}\end{bmatrix}\\ \text{scale}{:}(\mathit{PriceFrame}\rightarrow\mathit{Real})\end{bmatrix}.\ \begin{bmatrix}\text{e} & :(\mathit{PriceFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : r.\text{scale}(\text{e}[0])<r.\text{scale}(\text{e}[1])\end{bmatrix}$$

Finally we consider a third kind of rising event discussed in [6] based on the example in (16).

(16) As they get to deck, they see the Inquisitor, calling out to a Titan in the seas. The giant Titan rises through the waves, shrieking at the Inquisitor.

http://en.wikipedia.org/wiki/Risen_(video_game)
accessed 4th February, 2010

Here what needs to be kept constant in the rising event is the Titan. What needs to change between the two frames in the event is the height of the location of the Titan. Thus in this example the location is *not* kept constant. In order to analyze this we can use location frames of the type *LocFrame* as given in (17).

$$(17)\quad \begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{loc} & :\mathit{Loc}\\ \text{e} & :\text{at}(\text{x},\text{loc})\end{bmatrix}$$

The dependent type for a rise in location event is (18).

$$(18)\quad \lambda r{:}\begin{bmatrix}\text{fix}{:}\begin{bmatrix}\text{x}{:}\mathit{Ind}\end{bmatrix}\\ \text{scale}{:}(\mathit{LocFrame}\rightarrow\mathit{Real})\end{bmatrix}.\ \begin{bmatrix}\text{e} & :(\mathit{LocFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : r.\text{scale}(\text{e}[0])<r.\text{scale}(\text{e}[1])\end{bmatrix}$$

Here the obvious scale function does not simply return the value of a field in the location frame. What is needed is a scale based on the height of the location. One way to do this would be to characterize the type of locations, *Loc*, as the type of points in three-dimensional Euclidean space. That is, we consider *Loc* to be an abbreviation for (19).

$$(19)\quad \begin{bmatrix}\text{x-coord} & :\mathit{Real}\\ \text{y-coord} & :\mathit{Real}\\ \text{z-coord} & :\mathit{Real}\end{bmatrix}$$

Each of the fields in (19) corresponds to a coordinate in Euclidean space. A more adequate treatment would be to consider locations as regions in Euclidean space but we will not pursue that here. Treating *Loc* as (19) means that we

can characterize the scale function as returning the height of the location in the location frame, as in (20).

(20) $\lambda r{:}\mathit{LocFrame}\ .\ r.\text{loc.z-coord}$

If we wish to restrict the dependent type of rising events to vertical rises we can fix the x and y-coordinates of the location as in (21).

$$(21)\quad \lambda r{:}\begin{bmatrix}\text{fix:}\begin{bmatrix}\text{x:}\mathit{Ind}\\ \text{loc:}\begin{bmatrix}\text{x-coord:}\mathit{Real}\\ \text{y-coord:}\mathit{Real}\end{bmatrix}\end{bmatrix}\\ \text{scale:}(\mathit{LocFrame}\rightarrow\mathit{Real})\end{bmatrix}\ .\ \begin{bmatrix}\text{e} & : (\mathit{LocFrame}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : r.\text{scale(e[0])} < r.\text{scale(e[1])}\end{bmatrix}$$

We have now characterized three kinds of rising events. In [5,6] we argued that there is in principle no limit to the different kinds of rising events which can be referred to in natural language and that new types are created on the fly as the need arises. The formulation in those works did not allow us to express what all these particular meanings have in common. We were only able to say that the various meanings seem to have some kind of family resemblance. Now that we have abstracted out scales and parameters to be fixed we have an opportunity to formulate something more general. There are two things that vary across the different dependent types that we have characterized for risings. One is the frame type being considered and the other is the type of the record which contains the parameters held constant in the rising event. If we abstract over both of these we have a characterization of rising events in general. This is given in (22).

$$(22)\quad \lambda r{:}\begin{bmatrix}\text{frame\_type:}\mathit{RecType}\\ \text{fix\_type:}\mathit{RecType}\\ \text{fix:fix\_type}\\ \text{scale:}(\text{frame\_type}\rightarrow\mathit{Real})\end{bmatrix}\ .\ \begin{bmatrix}\text{e} & : (r.\text{frame\_type}\|r.\text{fix})^2\\ \text{c}_{\text{rise}} & : r.\text{scale(e[0])} < r.\text{scale(e[1])}\end{bmatrix}$$

(22) is so general (virtually everything of content has been parametrized) that it may be hard to see it as being used in the characterization of the meaning of *rise*. What seems important for characterizing the meanings of *rise* that a speaker has acquired is precisely the collection of frame types, and associated fix types and scales which an agent has developed through experience. (22) seems to be relevant to a kind of meta-meaning which specifies what kind of contents can be associated with the word *rise*. In this sense it seems related to the notion of *meaning potential*, a term which has its origins in the work of Halliday [16] where meanings are spoken of informally as being "created by the social system" and characterized as "integrated systems of meaning potential" (p. 199). The notion is much discussed in more recent literature, for example,

Linell [19], where meaning potential is discussed in the following terms: "Lexical meaning potentials are (partly) open meaning resources, where actual meanings can only emerge in specific, situated interactions" (p. 330).

2 Individual vs. Frame Level Nouns

Perhaps the most recent discussion of the Partee puzzle is that of Löbner [23]. As we will see, his proposal is closely related to our own. The puzzle is one that Barbara Partee raised while sitting in on an early presentation of the material that led to [24]. In its simplest form it is that (23c) should follow from (23a, b) given some otherwise apparently harmless assumptions.

(23) a. The temperature is rising
b. The temperature is ninety
c. Ninety is rising

Clearly, our intuitions are that (23c) does not follow from (23a, b).

A central aspect of our analysis of the Partee puzzle is that the contents of common nouns are functions that take frames, that is records, as arguments. We make a distinction between individual level predicates like 'dog' whose arity is $\langle Ind \rangle$ and frame level predicates like 'temperature' whose arity is $\langle Rec \rangle$. The content associated with an utterance event of type "dog" would be (24a). This is contrasted with the content for an utterance of type "temperature" given in (24b).

(24) a. $\lambda r{:}\left[\mathrm{x}{:}Ind\right] \,.\, \left[\mathrm{e} : \mathrm{dog}(r.\mathrm{x})\right]$
b. $\lambda r{:}\left[\mathrm{x}{:}Rec\right] \,.\, \left[\mathrm{e} : \mathrm{temperature}(r.\mathrm{x})\right]$

We make an exactly similar distinction between individual level and frame level verb phrases. In (25) we present contents which can be associated with utterances of type "run" and "rise" respectively.

(25) a. $\lambda r{:}\left[\mathrm{x}{:}Ind\right] \,.\, \left[\mathrm{e} : \mathrm{run}(r.\mathrm{x})\right]$
b. $\lambda r{:}\left[\mathrm{x}{:}Rec\right] \,.\, \left[\mathrm{e} : \mathrm{rise}(r.\mathrm{x})\right]$

When we predicate the content of *rise*, that is, (25b), of a temperature frame that has no consequence that the real number in the 'x'-field of the temperature frame also rises.

We have made a distinction between individual level nouns like *dog* and frame level nouns like *temperature*, differentiating their contents as in (24) and motivating the distinction with the Partee puzzle. Now consider (26).

(26) a. The dog is nine
b. The dog is getting older/aging
c. Nine is getting older/aging

We have the same intuitions about (26) as we do about the original temperature puzzle. We cannot conclude (26c) from (26a, b). Does this mean that *dog* is a frame level noun after all? Certainly, if we think of frames as being like entries in relational databases it would be natural to think of age (or information allowing us to compute age such as date of birth) as being a natural field in a dog-frame.

Our strategy to deal with this will be to say that contents of individual level nouns can be coerced to frame level contents, whereas the contents of frame level nouns cannot be coerced "down" to individual level contents. Thus in addition to (24a), repeated as (27a) we have (27b).

(27) a. $\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right].\ \left[\mathrm{e} : \mathrm{dog}(r.\mathrm{x})\right]$
b. $\lambda r{:}\left[\mathrm{x}{:}\mathit{Rec}\right].\ \left[\mathrm{e} : \mathrm{dog\_frame}(r.\mathrm{x})\right]$

The predicate 'dog_frame' is related to the predicate 'dog' by the constraint in (28).

(28) dog_frame(r) is non-empty implies r: $\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{dog(x)}\end{bmatrix}$

There are several different kinds of dog frames with additional information about a dog which an agent may acquire or focus on. Here we will consider just frames which contain a field labelled 'age' as specified in (29).

(29) If r: $\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{e}{:}\mathrm{dog(x)}\\ \mathrm{age}{:}\mathit{Real}\\ \mathrm{c_{age}}{:}\mathrm{age\_of(x,age)}\end{bmatrix}$ then dog_frame(r) is non-empty

An age scale, ζ_{age}, for individuals can then be defined as the function in (30).

(30) $\zeta_{\mathrm{age}} = \lambda r{:}\begin{bmatrix}\mathrm{x}{:}\mathit{Ind}\\ \mathrm{age}{:}\mathit{Real}\\ \mathrm{c_{age}}{:}\mathrm{age\_of(x,age)}\end{bmatrix}.\ r.\mathrm{age}$

We can think of the sentence *the dog is nine* as involving two coercions: one coercing the content of *dog* to a frame level property and the other coercing the content of *be* to a function which when applied to a number will return a frame level property depending on an available scale. Such coercions do not appear to be universally available in languages. For example, in German it is preferable to say *die Temperature ist 35 Grad* "the temperature is 35 degrees" rather than *#die Temperatur ist 35* "the temperature is 35". Similarly *der Hund ist neun Jahre alt* "the dog is nine years old" is preferred over *#der Hund ist neun* "the dog is nine".

3 Passengers and Ships

Gupta [15] points out examples such as (31).

(31) a. National Airlines served at least two million passengers in 1975
b. Every passenger is a person
c. National Airlines served at least two million persons in 1975

His claim is that we cannot conclude (31c) from (31a, b). There is a reading of (31a) where what is being counted is not passengers as individual people but passenger events, events of people taking flights, where possibly the same people are involved in several flights. Gupta claims that it is the only reading that this sentence has. While it is certainly the preferred reading for this sentence (say, in the context of National Airlines' annual report or advertizing campaign), I think the sentence also has a reading where individuals are being counted. Consider (32).

(32) National Airlines served at least two million passengers in 1975. Each one of them signed the petition.

While (32) could mean that a number of passengers signed the petition several times our knowledge that people normally only sign a given petition once makes a reading where there are two million distinct individuals involved more likely. Similarly, while (31c) seems to prefer the individual reading where there are two million distinct individuals it is not impossible to get an event reading here. [18] makes a similar point. Gupta's analysis of such examples involves individual concepts and is therefore reminiscent of the functional concepts used by [20,21] to analyze the Partee puzzle.

Carlson [4] makes a similar point about Gupta's examples in that nouns which appear to normally point to individual related readings can in the right context get the event related readings. One of his examples is a traffic engineer's report as in (33).

(33) Of the 1,000 cars using Elm St. over the past 49 h, only 12 cars made noise in excess of EPA recommended limits.

It is easy to interpret this in terms of 1,000 and 12 car events rather than individual cars. Carlson's suggestion is to use his notion of *individual stage*, what he describes intuitively as "things-at-a-time". Krifka [18] remarks that "Carlson's notion of a stage serves basically to reconstruct events". While this is not literally correct, the intuition is nevertheless right. Carlson was writing at a time when times and time intervals were used to attempt to capture phenomena that in more modern semantics would be analyzed in terms of events or situations. Thus Carlson's notion of stage is related to a frame-theoretic approach which associates an individual with an event.

Consider the noun *passenger*. It would be natural to assume that passengers are associated with journey events. FrameNet[5] does not have an entry for *passenger*. The closest relevant frame appears to be TRAVEL which has frame elements

[5] As of 13th May 2015.

for traveller, source, goal, path, direction, mode of transport, among others. The FrameNet lexical entry for *journey* is associated with this frame. Let us take the type *TravelFrame* to be the stripped down version of the travel frame type in (34a). Then we could take the type *PassengerFrame* to be (34b).

(34) a. $\begin{bmatrix} \text{traveller} & : \mathit{Ind} \\ \text{source} & : \mathit{Loc} \\ \text{goal} & : \mathit{Loc} \end{bmatrix}$

b. $\begin{bmatrix} \text{x} & : \mathit{Ind} \\ \text{e} & : \text{passenger(x)} \\ \text{journey} & : \mathit{TravelFrame} \\ \text{c}_{\text{travel}} & : \text{take\_journey(x, journey)} \end{bmatrix}$

A natural constraint to place on the predicate 'take_journey' is that in (35).

(35) If a:*Ind* and e:*TravelFrame*, then the type take_journey(a, e) is non-empty just in case e.traveller = a.

Let us suppose that the basic lexical entry for *passenger* provides the content (36).

(36) $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right] \,.\, \left[\text{e:passenger}(r.\text{x})\right]$

We can coerce this lexical item to (37).

(37) $\lambda r{:}\left[\text{x}{:}\mathit{Rec}\right] \,.\, \left[\text{e:passenger\_frame}(r.\text{x})\right]$

This means that the non-parametric content is a property of frames. An agent who has the frame type *PassengerFrame* available as a resource can use it to restrict the domain of the property. This produces (38).

(38) $\lambda r{:}\left[\text{x}{:}\mathit{PassengerFrame}\right] \,.\, \left[\text{e:passenger\_frame}(r.\text{x})\right]$

This means that the non-parametric content will now be a property of passenger frames of type *PassengerFrame*. This introduces not only a passenger but also a journey, an event in which in which the passenger is the traveller.

It seems that we have now done something which Krifka [18] explicitly warned us against. At the end of his discussion of Carlson's analysis he comes to the conclusion that it is wrong to look for an explanation of event-related readings of these sentences in terms of a noun ambiguity. One of Krifka's examples is (39) (which gives the title to his paper).

(39) Four thousand ships passed through the lock

This can either mean that four thousand distinct ships passed through the lock or that the there were four thousand ship-passing-through-the-lock events a number

of which might have involved the same ships. The problem he sees is that if we treat *ship* as being ambiguous between denoting individual ships or ship stages in Carlson's sense then there will be too many stages which pass through the lock. For example, suppose that a particular ship passes through the lock twice. This gives us two stages of the ship which pass through the lock. But then, Krifka claims, there will be a third stage, the sum of the first two, which also passes through the lock. It is not clear to me that this is an insuperable problem for the stage analysis. We need to count stages that pass through the lock exactly once. Let us see how the frame analysis fares.

We will start with a singular example in order to avoid the additional problems offered by the plural. Consider (40).

(40) Every passenger gets a hot meal

Suppose that an airline has this as part of its advertizing campaign. Smith, a frequent traveller, takes a flight with the airline and as expected gets a hot meal. A few weeks later she takes another flight with the same airline and does not get a hot meal. She sues the airline for false advertizing. At the hearing, her lawyer argues, citing Gupta [15], that the advertizing campaign claims that every passenger gets a hot meal on every flight they take. The lawyer for the airline company argues, citing Krifka [18], that the sentence in question is ambiguous between an individual and an event reading, that the airline had intended the individual reading and thus the requirements of the advertizing campaign had been met by the meal that Smith was served on the first flight. Smith's lawyer then calls an expert witness, a linguist who quickly crowdsources a survey of native speakers' interpretations of the sentence in the context of the campaign and discovers that there is an overwhelming preference for the meal-on-every-flight reading. (The small percentage of respondents who preferred the individual reading over the event reading gave their occupation as professional logician.) Smith wins the case and receives an additional hot meal.

What is important for us at the moment is the fact that there is an event reading of this sentence. We use the coerced content associated with *passenger* in (38).

In order to simplify matters let us treat *gets a hot meal* as if it were an intransitive verb corresponding to a single predicate 'get_a_hot_meal'. This is a predicate whose arity is $\langle Ind \rangle$. It is individuals, not frames (situations), that get hot meals. Thus the content of *gets a hot meal* will be (41).

(41) $\lambda r{:}\left[\mathrm{x}{:}Ind\right] \,.\, \left[\mathrm{e}{:}\mathrm{get\_a\_hot\_meal}(r.\mathrm{x})\right]$

We need a coercion which will obtain a frame level intransitive verb to match the frame level noun. The new content for *get_a_hot_meal* will be (42).

(42) $\lambda r{:}\left[\mathrm{x}{:}Rec\right] \,.\, \left[\mathrm{e}{:}\mathrm{get\_a\_hot\_meal\_frame}(r.\mathrm{x})\right]$

Recall that if p is a predicate of individuals then p_frame is a predicate of frames that contain an individual of which p holds. This means that an argument,

r, to 'get_a_hot_meal_frame' which makes the type 'get_a_hot_meal_frame(r)' non-empty will be of type (43).

(43) $\begin{bmatrix} \text{x} : \textit{Ind} \\ \text{e} : \text{get\_a\_hot\_meal(x)} \end{bmatrix}$

Thus intuitively the 'every' relation holding between the two frame-level coerced individual properties corresponding to *passenger* and *get_a_hot_meal* will mean "every frame (situation) containing an individual in the 'x'-field who is a passenger taking a journey will be a frame where the individual in the 'x'-field gets a hot meal". Or, more formally, (44).

(44) every r of type $\begin{bmatrix} \text{x} & : \textit{Ind} \\ \text{e} & : \text{passenger(x)} \\ \text{journey} & : \textit{TravelFrame} \\ \text{c}_{\text{travel}} & : \text{take\_journey(x, journey)} \end{bmatrix}$ is of type $\begin{bmatrix} \text{x} : \textit{Ind} \\ \text{e} : \text{get\_a\_hot\_meal(x)} \end{bmatrix}$

This means that every frame of type *PassengerFrame* will be of type (45).

(45) $\begin{bmatrix} \text{x} & : \textit{Ind} \\ \text{e} & : \text{passenger(x)} \wedge \text{get\_a\_hot\_meal(x)} \\ \text{journey} & : \textit{TravelFrame} \\ \text{c}_{\text{travel}} & : \text{take\_journey(x, journey)} \end{bmatrix}$

Thus even though we have coerced to a frame-level reading it is still the passengers (i.e. individuals) in the frames who are getting the hot meal not the situation which is the frame.

4 Conclusion

In this paper we have proposed an analysis of frames as records which model situations (including events) and we have suggested that frame types (record types) are important in both the analysis of the Partee puzzle concerning rising temperatures and prices and in the analysis of quantification which involves counting events rather than individuals likes passengers or ships passing through a lock.

Our original inspiration for frames comes from the work of [13,14] and work on FrameNet (https://framenet.icsi.berkeley.edu). An important aspect of our approach to frames is that we treat them as first class objects. That is, they can be arguments to predicates and can be quantified over. While this is important, it is not surprising once we decide that frames are in fact situations (here modelled by records) or situation types (here modelled by record types). The distinction between frames and frame types is not made in the literature deriving from

Fillmore's work but it seems to be an important distinction to draw if we wish to apply the notion of frame to the kind of examples we have discussed in this chapter.

The proposal that we have made for solving the Partee puzzle is closely related to the work of Löbner [22,23] whose inspiration is from the work of Barsalou [1–3] rather than Fillmore. Barsalou's approach embedded in a theory of cognition based on perception and a conception of cognition as dynamic, that is, a system in a constant state of flux [25], seems much in agreement with what we are proposing. Barsalou's [3] characterization of basic frame properties constituting a frame as: "(1) predicates, (2) attribute-value bindings, (3) constraints, and (4) recursion" seem to have a strong family resemblance with our record types. Our proposal for incorporating frames into natural language semantics is, however, different from Löbner's in that he sees the introduction of a psychological approach based on frames as a reason to abandon a formal semantic approach whereas we see type theory as a way of combining the insights we have gained from model theoretic semantics with a psychologically oriented approach.

Our approach to frames has much in common with that of Kallmeyer and Osswald [17] who use feature structures to characterize their semantic domain. We have purposely used record types in a way that makes them correspond both to feature structures and discourse representation structures which allows us to relate our approach to more traditional model theoretic semantics at the same time as being able to merge record types corresponding to unification in feature-based systems. However, our record types are included in a richer system of types including function types facilitates a treatment of quantification and binding which is not available in a system which treats feature structures as a semantic domain.[6]

References

1. Barsalou, L.W.: Cognitive Psychology. An Overview for Cognitive Scientists. Lawrence Erlbaum Associates, Hillsdale (1992)
2. Barsalou, L.W.: Frames, concepts, and conceptual fields. In: Lehrer, A., Kittay, E.F. (eds.) Frames, Fields, and Contrasts: New Essays in Semanticand Lexical Organization, pp. 21–74. Lawrence Erlbaum Associates, Hillsdale (1992)
3. Barsalou, L.W.: Perceptual symbol systems. Behav. Brain Sci. **22**, 577–660 (1999)
4. Carlson, G.N.: Generic terms and generic sentences. J. Philos. Logic **11**, 145–181 (1982)
5. Cooper, R.: Frames in formal semantics. In: Loftsson, H., Rögnvaldsson, E., Helgadóttir, S. (eds.) IceTAL 2010. LNCS, vol. 6233, pp. 103–114. Springer, Heidelberg (2010)
6. Cooper, R.: Type theory and semantics in flux. In: Kempson, R., Asher, N., Fernando, T. (eds.) Philosophy of Linguistics. Handbook of the Philosophy of Science, vol. 14, pp. 271–323. Elsevier, Amsterdam (2012). General editors: Gabbay, D.M., Thagard, P., Woods, J

[6] It is possible to code up a notation for quantification in feature structures but that is not the same as giving a semantics for it.

7. Cooper, R.: Type theory and language: from perception to linguistic communication (in prep). https://sites.google.com/site/typetheorywithrecords/drafts
8. Fernando, T.: A finite-state approach to events in natural language semantics. J. Logic Comput. **14**(1), 79–92 (2004)
9. Fernando, T.: Situations as strings. Electron. Notes Theoret. Comput. Sci. **165**, 23–36 (2006)
10. Fernando, T.: Finite-state descriptions for temporal semantics. In: Bunt, H., Muskens, R. (eds.) Comput. Meaning. Studies in Linguistics and Philosophy, vol. 83, pp. 347–368. Springer, Heidelberg (2008)
11. Fernando, T.: Situations in LTL as strings. Inf. Comput. **207**(10), 980–999 (2009)
12. Fernando, T.: Constructing situations and time. J. Philos. Logic **40**, 371–396 (2011)
13. Fillmore, C.J.: Frame semantics. Linguistics in the Morning Calm, pp. 111–137. Hanshin Publishing Co., Seoul (1982)
14. Fillmore, C.J.: Frames and the semantics of understanding. Quaderni di Semantica **6**(2), 222–254 (1985)
15. Gupta, A.: The Logic of Common Nouns: An Investigation in Quantified Model Logic. Yale University Press, New Haven (1980)
16. Halliday, M.A.K.: Text as semantic choice in social contexts. In: van Dijk, T., Petöfi, J. (eds.) Grammars and Descriptions, pp. 176–225. Walter de Gruyter, Berlin (1977)
17. Kallmeyer, L., Osswald, R.: Syntax-driven semantic frame composition in lexicalized tree adjoining grammars. J. Lang. Model. **1**(2), 267–330 (2013)
18. Krifka, M.: Four thousand ships passed through the lock: object-induced measure functions on events. Linguist. Philos. **13**, 487–520 (1990)
19. Linell, P.: Rethinking Language, Mind, and World Dialogically: Interactional and Contextual Theories of Human Sense-Making. Advances in Cultural Psychology: Constructing Human Development. Information Age Publishing Inc., Charlotte (2009)
20. Löbner, S.: Intensionale Verben und Funktionalbegriffe. Untersuchung zur Syntax und Semantik von wechseln und den vergleichbaren Verben des Deutschen. Narr, Tübingen (1979)
21. Löbner, S.: Intensional verbs and functional concepts: more on the "rising temperature" problem. Linguist. Inq. **12**(3), 471–477 (1981)
22. Löbner, S.: Evidence for frames from human language. In: Gamerschlag, T., Gerland, D., Petersen, W., Osswald, R. (eds.) Frames and Concept Types, Studies in Linguistics and Philosophy, vol. 94, pp. 23–68. Springer, Heidelberg (2014)
23. Löbner, S.: Functional Concepts and Frames (in prep). http://semanticsarchive.net/Archive/jI1NGEwO/Loebner_Functional_Concepts_and_Frames.pdf
24. Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics, pp. 247–270. D. Reidel Publishing Company, Dordrecht (1973)
25. Prinz, J.J., Barsalou, L.W.: Steering a course for embodied representation. In: Dietrich, E., Markman, A.B. (eds.) Cognitive Dynamics: Conceptual and Representational Change in Humans and Machines, pp. 51–77. Psychology Press, Hove (2014). Previously published in 2000 by Lawrence Erlbaum

Types from Frames as Finite Automata

Tim Fernando(✉)

Trinity College Dublin, Dublin, Ireland
tim.fernando@cs.tcd.ie

Abstract. An approach to frame semantics is built on a conception of frames as finite automata, observed through the strings they accept. An institution (in the sense of Goguen and Burstall) is formed where these strings can be refined or coarsened to picture processes at various bounded granularities, with transitions given by Brzozowski derivatives.

Keywords: Frame · Finite automaton · Trace · Derivative · Institution

1 Introduction

A proposal for frame semantics recently put forward in Muskens (2013) analyzes a frame of the sort studied by Barsalou (1999); Löbner (2014), and Petersen and Osswald (2014) (not to forget Fillmore 1982) as a *fact* in a *data lattice* $\langle \mathfrak{F}, \circ, 0 \rangle$ with zero $0 \in \mathfrak{F}$ and meet $\circ : (\mathfrak{F} \times \mathfrak{F}) \to \mathfrak{F}$ (Veltman 1985). A frame such as

is analyzed, relative to any three entities e, x and y, as the $\circ$-combination

$$smash\, e \circ \text{AGENT}\, ex \circ John\, x \circ \text{THEME}\, ey \circ window\, y$$

of five facts, $smash\, e$, $\text{AGENT}\, ex$, $John\, x$, $\text{THEME}\, ey$, and $window\, y$. In general, any fact $g \in \mathfrak{F}$ induces a function $[g]$ from facts to one of three truth values, $\mathbf{t}$, $\mathbf{f}$ and $\mathbf{n}$, such that for all $f \in \mathfrak{F} - \{0\}$,

$$[g](f) \quad = \quad \begin{cases} \mathbf{t} & \text{if } f \circ g = f \text{ (i.e., } f \text{ incorporates } g\text{)} \\ \mathbf{f} & \text{if } f \circ g = 0 \text{ (i.e., } f \text{ and } g \text{ are incompatible)} \\ \mathbf{n} & \text{otherwise.} \end{cases}$$

Functions such as $[g]$ from $\mathfrak{F}$ to $\{\mathbf{t}, \mathbf{f}, \mathbf{n}\}$ are what Muskens calls *propositions*, breaking from possible worlds semantics in replacing possible worlds with facts, and adding a third truth value, $\mathbf{n}$, for a gap between truth and falsity. Sentences are interpreted as propositions, assembled compositionally from an interpretation of words as λ-abstracts of propositions, as in

$$\mathsf{smash} \quad = \quad \lambda y x \lambda f \exists e.[smash\, e \circ \text{AGENT}\, ex \circ \text{THEME}\, ey] f. \tag{1}$$

Muskens attaches significance to the separation of facts from propositions. Identifying frames with facts, he declares

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 19–40, 2016.
DOI: 10.1007/978-3-662-53042-9_2

> I reject the idea (defended in Barsalou (1999), who explicitly discusses frame representations of negation, disjunction, and universal quantification) that all natural language meaning can profitably be represented with the help of frames

(page 176).

One way to evaluate Muskens' proposal is by comparing it with others. An alternative to existentially quantifying the event e in (1) is λ-abstraction, as in the analysis of sortal frames in Petersen and Osswald (2014), with

$$\lambda e.\ \mathit{smash}'(e) \wedge \mathit{animate}'(\textsc{agent}'(e)) \wedge \mathit{concrete}'(\textsc{theme}'(e)) \tag{2}$$

for the typed feature structure (a), or, to bring the example closer to (1),

$$\lambda e.\ \mathit{smash}'(e) \wedge \mathit{all}'(\textsc{agent}'(e)) \wedge \mathit{all}'(\textsc{theme}'(e)) \tag{3}$$

for the typed feature structure (b) over a vacuous all-encompassing type all.[1]

(a) $\begin{bmatrix} \mathit{smash} & \\ \textsc{agent} & \mathit{animate} \\ \textsc{theme} & \mathit{concrete} \end{bmatrix}$ (b) $\begin{bmatrix} \mathit{smash} & \\ \textsc{agent} & \mathit{all} \\ \textsc{theme} & \mathit{all} \end{bmatrix}$ (c) $\begin{bmatrix} \mathit{smash} \\ \textsc{agent} \\ \textsc{theme} \end{bmatrix}$

It is understood in both (2) and (3) that e is in the domain of the partial functions $\textsc{agent}'$ and $\textsc{theme}'$, making the terms $\textsc{agent}'(e)$ and $\textsc{theme}'(e)$ well-defined. We will simplify (b) shortly to (c), but before dropping all, let us use it to illustrate how to express the definedness presuppositions in (2) and (3) under the approach of Cooper (2012). To model a context, Cooper uses a record such as (d), which is of type (e) assuming all encompasses all.

(d) $\begin{bmatrix} \textsc{agent} = x \\ \textsc{theme} = y \end{bmatrix}$ (e) $\begin{bmatrix} \textsc{agent} : \mathit{all} \\ \textsc{theme} : \mathit{all} \end{bmatrix}$ (f) $\begin{bmatrix} p_1 : \mathit{smash}(r) \\ p_2 : \mathit{animate}(r.\textsc{agent}) \\ p_3 : \mathit{concrete}(r.\textsc{theme}) \end{bmatrix}$

Now, if bg is the record type (e), and φ is the type (f) dependent on a record r of type bg, we can form the function

$$(\lambda r : bg)\ \varphi \tag{4}$$

mapping a record r of type bg to the record type φ. (4) serves as Cooper's meaning function with

- domain bg (for background) encoding the definedness presuppositions, and
- record type φ replacing what a Montagovian would have as a truth value.

Compared to the prefix λyx in Muskens' (1), the prefix $(\lambda r : bg)$ in (4) provides not just the parameters x and y but the information that they are the agent and theme components of a record r, around which abstraction is centralized in accordance with the methodological assumption

[1] This introductory section presupposes some familiarity with the literature, but is followed by sections that proceed in a more careful manner, without relying on a full understanding of the Introduction.

(†) components are extracted from a single node associated with the frame.

The number of components may be open-ended, as argued famously for events in Davidson (1967)

Jones did it slowly, deliberately, in the bathroom with a knife, at midnight

(page 81). Under pressure from multiple components, the assumption (†) is relaxed for non-sortal frames in Löbner (2014) and Petersen and Osswald (2014), with the latter resorting to ϵ-terms (page 249) and ι-terms (page 252). It is, however, possible to maintain (†) by adding attributes that extend the central node to incorporate the required components (making ϵ- and ι-terms unnecessary). At stake in upholding (†) is a record type approach, a finite-state fragment of which is the subject of the present paper.

In Cooper (2012), record types are part of a rich system TTR of types with function spaces going well beyond finite-state methods. Viewing frames as finite automata — bottom-dwellers in the Chomsky hierarchy — certainly leads us back to Muskens' contention that frames cannot capture all natural language meaning. But while any *single* finite automaton is bound to fall short, much can be done with many automata. Or so the present paper argues. Very briefly, the idea is to reduce the matrices (a)–(c) to the sets (a)′–(c)′ of strings over the alphabet $\{smash, \textsc{agent}, \textsc{theme}, animate, concrete, all\}$, and to represent the typing in (g) by the matrix (h) that is reduced to the language (h)′ over an expansion of the alphabet with symbols a_x and a_y for x and y respectively.

(a)′ $\{smash,\ \textsc{agent}\ animate,\ \textsc{theme}\ concrete\}$
(b)′ $\{smash,\ \textsc{agent}\ all,\ \textsc{theme}\ all\}$
(c)′ $\{smash,\ \textsc{agent},\ \textsc{theme}\}$

(g) $\begin{bmatrix} smash \\ \textsc{agent} = x \\ \textsc{theme} = y \end{bmatrix} : \begin{bmatrix} smash \\ \textsc{agent} : animate \\ \textsc{theme} : concrete \end{bmatrix}$ (h) $\begin{bmatrix} smash & \\ \textsc{agent} & \begin{bmatrix} animate \\ a_x \end{bmatrix} \\ \textsc{theme} & \begin{bmatrix} concrete \\ a_y \end{bmatrix} \end{bmatrix}$

(h)′ $\{smash,\ \textsc{agent}\ a_x,\ \textsc{theme}\ a_y\} \cup \{agent\ animate,\ \textsc{theme}\ concrete\}$

To interpret the strings in the sets (a)′, (b)′, (c)′ and (h)′, we assume every symbol a in the string alphabet Σ is interpreted as a (partial) function $[\![a]\!]$, which we extend to strings $s \in \Sigma^*$, setting $[\![\epsilon]\!]$ to the identity, and $[\![sa]\!]$ to the sequential composition $\lambda x.[\![a]\!]([\![s]\!](x))$. Then in place of the λ-expressions (1) to (4), a language L is interpreted as the intersection

$$\bigcap_{s \in L} domain([\![s]\!]) \tag{5}$$

of the domains of the interpretations of strings in L. It is customary to present the interpretations $[\![a]\!]$ model-theoretically (as in the case of the interpretation $\textsc{agent}'$ of $\textsc{agent}$ in (2)), making the interpretations $[\![s]\!]$ and (5) model-theoretic.

But as will become clear below, the functions $[\![\alpha]\!]$ can also be construed as the α-labeled transitions in a finite automaton. The ultimate objective of the present work is to link frames to the finite-state perspective on events in Fernando (2015) (the slogan behind the bias for finite-state methods being *less is more*), as well as to more wide ranging themes of "semantics in flux" in Cooper (2012), and "natural languages as collections of resources" in Cooper and Ranta (2008).

The intersection (5) approximates the image $\{r : bg \mid \varphi\}$ of Cooper's meaning function $(\lambda r : bg)\varphi$ but falls short of maintaining the careful separation that $(\lambda r : bg)\varphi$ makes between the presuppositions bg and the dependent record type φ. That separation is recreated below within what is called an *institution* in Goguen and Burstall (1992), with bg formulated as a signature and φ as a sentence of that signature. Clarifying this formulation is largely what the remainder of the present paper is about, which consists of three sections, plus a conclusion. The point of departure of Sect. 2 is the determinism of a frame — the property that for every state (or node) q and every label a on an arc (or edge), there is at most one arc from q labeled a. Based on determinism and building on Hennessy and Milner (1985), Sect. 2 reduces a state q to a set of strings of labels (i.e., a language). This reduction is tested against states as types and states as particulars in Sect. 3. To ensure the languages serving as states are accepted by finite automata (i.e., regular languages), Sect. 4 works with various finite sets Σ of labels. The sets Σ are paired with record types for signatures, around which approximations are structured following Goguen and Burstall (1992).

One further word about the scope of the present work before proceeding. Functions and λ's are commonly taken for granted in a compositional syntax/semantics interface, yet another significant proposal for which is detailed in Kallmeyer and Osswald (2013) using Lexicalized Tree Adjoining Grammar (distinct from frames with a first-order formulation in Sects. 3.3.3–3.3.4 there compatible with (5) above[2]). The present paper steers clear of any choice of a syntactic formalism, making no claim of completeness in focusing (modestly) on types from frames as finite automata.

2 Deterministic Systems and Languages

Fix a (possibly infinite) set A of labels. An A-*deterministic system* is a partial function $\delta : Q \times \mathsf{A} \rightharpoonup Q$ from pairs $(q, a) \in Q \times \mathsf{A}$ to elements of Q, called states (of which there may or may not be infinitely many). Let ϵ be the null string (of length 0) and for any state $q \in Q$, let $\delta_q : \mathsf{A}^* \rightharpoonup Q$ be the partial Q-valued function from strings over the alphabet A that repeatedly applies δ starting at q; more precisely, δ_q is the $\subseteq$-least set P of pairs such that

(i) $(\epsilon, q) \in P$, and
(ii) $(sa, \delta(q', a)) \in P$ whenever $(s, q') \in P$ and $(q', a) \in \mathit{domain}(\delta)$.

[2] The compatibility here becomes obvious if the moves described in footnote 5 of page 281 in Kallmeyer and Osswald (2013) are made, and a root added with attributes to the multiple base nodes.

For example,

$$aa' \in \mathit{domain}(\delta_q) \iff (q,a) \in \mathit{domain}(\delta) \text{ and } a' \in \mathit{domain}(\delta_{\delta(q,a)}).$$

The partial functions δ_q determine

$$\mathit{transitions}\, q \xrightarrow{s} \delta_q(s) \text{ whenever } s \in \mathit{domain}(\delta_q)$$

which we can also read as

$$\mathit{s\text{-}components}\, \delta_q(s) \text{ of } q, \text{ for all } s \in \mathit{domain}(\delta_q).$$

The labels in A may correspondingly be regarded as acts or as attributes. In either case, there is, we will see, a useful sense in which the language $\mathit{domain}(\delta_q)$ over A holds just about all the A-deterministic system δ has to say about q. An element of $\mathit{domain}(\delta_q)$ is called a *trace of* q (*from* δ), and henceforth, we write $\mathit{trace}_\delta(q)$ interchangably with $\mathit{domain}(\delta_q)$.

2.1 Satisfaction and Traces

Given a set A of labels, the set Φ_A of (A-modal) formulas φ is generated

$$\varphi \ ::= \ \top \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle a \rangle \varphi$$

from a tautology $\top$, negation $\neg$, conjunction $\wedge$, and modal operators $\langle a \rangle$ with labels $a \in \mathsf{A}$ (Hennessy and Milner 1985). We interpret a formula $\varphi \in \Phi_\mathsf{A}$ relative to an A-deterministic system $\delta : Q \times \mathsf{A} \rightharpoonup Q$ and state $q \in Q$ via a satisfaction relation $\models$ in the usual way, with (keeping δ implicit in the background)

$$q \models \top,$$

'not' $\neg$

$$q \models \neg\varphi \iff \text{not } q \models \varphi,$$

'and' $\wedge$

$$q \models \varphi \wedge \varphi' \iff q \models \varphi \text{ and } q \models \varphi'$$

and the accessibility relation $\{(q, \delta_q(a)) \mid q \in Q \text{ and } a \in \mathit{domain}(\delta_q)\}$ for $\langle a \rangle$

$$q \models \langle a \rangle \varphi \iff a \in \mathit{domain}(\delta_q) \text{ and } \delta_q(a) \models \varphi$$

It is not difficult to see that the set

$$\Phi_\mathsf{A}(q) \ := \ \{\varphi \in \Phi_\mathsf{A} \mid q \models \varphi\}$$

of formulas $\models$-satisfied by q depends precisely on $\mathit{domain}(\delta_q)$. That is, recalling that $\mathit{trace}_\delta(q)$ is $\mathit{domain}(\delta_q)$, the following conditions, (a) and (b), are equivalent for all states $q, q' \in Q$.

(a) $trace_\delta(q) = trace_\delta(q')$
(b) $\Phi_{\mathsf{A}}(q) = \Phi_{\mathsf{A}}(q')$

Let us write $q \sim q'$ if (a), or equivalently (b), holds,[3] and pronounce $\sim$ *trace equivalence*.

2.2 Identity of Indiscernibles and states as Languages

Identity of indiscernibles (also known as Leibniz's law, and mentioned in Osswald 1999, invoking Quine) can be understood against the set A of attributes as the requirement on δ that distinct pairs q, q' of states (in Q) *not* be trace equivalent

$$q \neq q' \implies q \not\sim q'.$$

Basing discernibility on formulas $\varphi \in \Phi_{\mathsf{A}}$, we say φ *differentiates* q *from* q' if $q \models \varphi$ but not $q' \models \varphi$. It follows that

$$\varphi \text{ differentiates } q \text{ from } q' \iff \neg\varphi \text{ differentiates } q' \text{ from } q$$

and

$$q \sim q' \iff \textit{no} \text{ formula in } \Phi_{\mathsf{A}} \text{ differentiates } q \text{ from } q'.$$

We can replace formulas by attributes and make differentiation symmetric, by agreeing that a label a *differentiates* q *from* q' if (exactly) one of the following holds

(i) $a \in trace_\delta(q) - trace_\delta(q')$
(ii) $a \in trace_\delta(q') - trace_\delta(q)$
(iii) $a \in trace_\delta(q) \cap trace_\delta(q')$ and $\Phi_{\mathsf{A}}(\delta_q(a)) \neq \Phi_{\mathsf{A}}(\delta_{q'}(a))$

In the case of (i) and (ii), we can see $q \not\sim q'$ already at a, whereas (iii) digs deeper. Two other equivalent ways to say a differentiates q from q' are (a) and (b) below.

(a) a is a prefix of a string in the symmetric difference of trace sets

$$(trace_\delta(q) \cup trace_\delta(q')) - (trace_\delta(q) \cap trace_\delta(q'))$$

(b) there exists $\varphi \in \Phi_{\mathsf{A}}$ such that the formula $\langle a \rangle \varphi$ differentiates either q from q' or q' from q

The notion of an attribute $a \in \mathsf{A}$ differentiating q from q' generalizes straightforwardly to a string $a_1 \cdots a_n \in \mathsf{A}^+$ differentiating q from q'.

In fact, if we reduce a state q to the language $trace_\delta(q)$, the notions of differentiation above link up smoothly with derivatives of languages (Brzozowski 1964;

[3] Readers familiar with bisimulations will note that $\sim$ is the largest bisimulation (determinism being an extreme form of image-finiteness; Hennessy and Milner 1985).

Conway 1971; Rutten 1998, among others). Given a language L and a string s, the *s-derivative of* L is the set

$$L_s \;:=\; \{s' \mid ss' \in L\}$$

obtained from strings in L that begin with s, by stripping s off. Observe that for all $q \in Q$ and $s \in \mathit{trace}_\delta(q)$, if $L = \mathit{trace}_\delta(q)$ then the s-derivative of L corresponds to the s-component $\delta_q(s)$ of q

$$L_s \;=\; \mathit{trace}_\delta(\delta_q(s))$$

and L decomposes into its components

$$L \;=\; \epsilon + \sum_{a \in \mathsf{A}} aL_a. \tag{6}$$

The fact that ϵ belongs to $\mathit{trace}_\delta(q)$ reflects prefix-closure. More precisely, a language L is said to be *prefix-closed* if $s \in L$ whenever $sa \in L$. That is, L is prefix-closed iff $\mathit{prefix}(L) \subseteq L$, where the set $\mathit{prefix}(L)$ of prefixes in L

$$\mathit{prefix}(L) \;:=\; \{s \mid L_s \neq \emptyset\}$$

consists of all strings that induce non-empty derivatives. For any non-empty prefix-closed language L, we can form a deterministic system δ over the set

$$\{L_s \mid s \in L\}$$

of s-derivatives of L, for $s \in L$, including ϵ for $L_\epsilon = L = \mathit{trace}_\delta(L)$, where $\mathit{domain}(\delta)$ is defined to be $\{(L_s, a) \mid sa \in L\}$ with

$$\delta(L_s, a) \;:=\; L_{sa} \;\text{ whenever } sa \in L.$$

But what about languages that are not prefix-closed? Without the assumption that L is prefix-closed, we must adjust Eq. (6) to

$$L \;=\; o(L) + \sum_{a \in \mathsf{A}} aL_a$$

with ϵ replaced by $\emptyset$ in case $\epsilon \notin L$, using

$$o(L) \;:=\; \begin{cases} \epsilon & \text{if } \epsilon \in L \\ \emptyset & \text{otherwise} \end{cases}$$

(called the *constant part* or *output* of L in Conway 1971, page 41). Now, the chain of equivalences

$$a_1 \cdots a_n \in L \iff a_2 \cdots a_n \in L_{a_1} \iff \cdots \iff \epsilon \in L_{a_1 \cdots a_n}$$

means that L is accepted by the automaton with

(i) all s-derivatives of L as states (whether or not $s \in \mathit{prefix}(L)$)

$$Q \;:=\; \{L_s \mid s \in \mathsf{A}^*\}$$

(ii) s-derivatives L_s for $s \in L$ as *final* (accepting) states
(iii) transitions forming a total function $Q \times \mathsf{A} \to Q$ mapping (L_s, a) to L_{sa}

and initial state $L_\epsilon = L$ (e.g., Rutten 1998).

An alternative approach out of prefix closure (from deterministic systems) is to define for any label $a \in \mathsf{A}$ and language $L \subseteq \mathsf{A}^*$, the *a-coderivative of* L to be the set

$$_aL \;:=\; \{s \mid sa \in L\}$$

of strings that, with a attached at the end, belong to L. Observe that the a-coderivative of a prefix-closed language is not necessarily prefix-closed (in contrast to s-derivatives). Furthermore,

Fact 1. *Every language is the coderivative of a prefix-closed language.*

Fact 1 is easy to establish: given a language L, attach a symbol a not occurring in L to the end of L, and form $\mathit{prefix}(La)$ before taking the a-coderivative

$$_a\mathit{prefix}(La) \;=\; L.$$

An a-coderivative effectively builds in a notion of final state (lacking in a deterministic system δ) around not $o(L)$ but $o(L_a)$, checking if ϵ is in L_a, rather than L (i.e., $a \in L$, rather than $\epsilon \in L$). The idea of capturing a type of state through a label (such as a for a-coderivatives) is developed further next.

3 From Attribute Values to Types and Particulars

An A-deterministic system $\delta : Q \times \mathsf{A} \rightharpoonup Q$ assigns each state $q \in Q$ a set

$$\hat{\delta}(q) \;:=\; \{(a, \delta(q, a)) \mid a \in \mathsf{A} \cap \mathit{trace}_\delta(q)\}$$

of attribute value pairs (a, q') with values q' that can themselves be thought as sets $\hat{\delta}(q')$ of attribute values pairs. Much the same points in Sect. 2 could be made appealing to the modal logic(s) of attribute value structures (Blackburn 1993) instead of Hennessy and Milner (1985). But is the reduction of q to its trace set, $\mathit{trace}_\delta(q)$, compatible with intuitions about attribute value structures? Let us call a state q *δ-null* if $\hat{\delta}(q) = \emptyset$; i.e., $\mathit{trace}_\delta(q) = \{\epsilon\}$. Reducing a δ-null state q to $\mathit{trace}_\delta(q) = \{\epsilon\}$ lumps all δ-null states into one — which is problematic if distinct atomic values are treated as δ-null (Blackburn 1993). But what if equality with a fixed value were to count as a discerning attribute? Let us define a state q to be *δ-marked* if there is a label $a_q \in \mathsf{A}$ such that for all $q' \in Q$,

$$a_q \in \mathit{trace}_\delta(q') \iff q = q'.$$

Clearly, a δ-marked state q is trace equivalent only to itself. If a state q is *not* δ-marked, we can introduce a fresh label a_q (not in A) and form the $(\mathsf{A} \cup \{a_q\})$-deterministic system

$$\delta[q] \;:=\; \delta \cup \{(q, a_q, q)\}$$

with q $\delta[q]$-marked. To avoid infinite trace sets $trace_{\delta[q]}(q) \supseteq {a_q}^*$ (from loops (q, a_q, q)), we can instead fix a δ-null (or fresh) state $\surd$ and mark q in

$$\delta[q, \surd] \;:=\; \delta \cup \{(q, a_q, \surd)\}.$$

Marking a state is an extreme way to impose Leibniz's law. A more moderate alternative described below introduces types over states, adding constraints to pick out particulars.

3.1 Type-Attribute Specifications and Containment

To differentiate states in a set Q through subsets $Q_t \subseteq Q$ given by types $t \in T$ is to define a binary relation $v \subseteq Q \times T$ (known in Kripke semantics as a *T-valuation*) such that for all $q \in Q$ and $t \in T$

$$v(q, t) \iff q \in Q_t.$$

We can incorporate v into $\delta : Q \times \mathsf{A} \rightharpoonup Q$ by adding a fresh attribute a_t, for each $t \in T$, to A for the expanded attribute set

$$\mathsf{A}_T \;:=\; \mathsf{A} \cup \{a_t \mid t \in T\}$$

and forming either the A_T-deterministic system

$$\delta[v] \;:=\; \delta \cup \{(q, a_t, q) \mid t \in T \text{ and } v(q, t)\}$$

or, given a δ-null state $\surd$ outside $\bigcup_{t \in T} Q_t$, the A_T-deterministic system

$$\delta[v, \surd] \;:=\; \delta \cup \{(q, a_t, \surd) \mid t \in T \text{ and } v(q, t)\}.$$

In practice, a type t may be defined from other types, as in (a) below.

$$\text{(a)}\quad t = \begin{bmatrix} \mathit{smash} & : \top \\ \textsc{agent} & : \mathit{animate} \\ \textsc{theme} & : \mathit{concrete} \end{bmatrix} \qquad\qquad \text{(b)}\quad e = \begin{bmatrix} \mathit{smash} & = x \\ \textsc{agent} & = y \\ \textsc{theme} & = z \end{bmatrix}$$

The record e given by (b) is an instance of t just in case x, y and z are instances of the types $\top$, *animate* and *concrete* respectively

$$e : t \iff x : \top \text{ and } y : \mathit{animate} \text{ and } z : \mathit{concrete}.$$

It is natural to analyze t in (a) as the modal formula φ_t with three conjuncts

$$\varphi_t \;=\; \langle \mathit{smash} \rangle \top \wedge \langle \textsc{agent} \rangle \langle \mathit{animate} \rangle \top \wedge \langle \textsc{theme} \rangle \langle \mathit{concrete} \rangle \top$$

(implicitly analyzing $\top$, *animate* and *concrete* as $\top$, $\langle animate\rangle\top$ and $\langle animate\rangle\top$ respectively) and to associate e with the trace set

$$q(e) \;=\; \epsilon \;+\; smash\ q(x) \;+\; \textsc{agent}\ q(y) \;+\; \textsc{theme}\ q(z)$$

(given trace sets $q(x), q(y)$ and $q(z)$ for x, y and z respectively) for the reduction

$$\begin{aligned} e : t \;&\Longleftrightarrow\; q(e) \models \varphi_t \\ &\Longleftrightarrow\; animate \in q(y) \text{ and } concrete \in q(z). \end{aligned}$$

We can rewrite (a) as the A'-deterministic system

$$\tau' \;:=\; \{(t, smash, \top),\ (t, \textsc{agent}, animate),\ (t, \textsc{theme}, concrete)\}$$

over the attribute set

$$\mathsf{A}' \;:=\; \{smash, \textsc{agent}, \textsc{theme}\}$$

and state set

$$T' \;:=\; \{t, \top, animate, concrete\}$$

given by types. To apply τ' to a deterministic system δ with states given by tokens of types, the following notion will prove useful. A $(T, \mathsf{A}, \surd)$-*specification* is an A_T-deterministic system $\tau : T \times \mathsf{A}_T \rightharpoonup T$ where $\surd \in T$ is τ-null and each $t \in T - \{\surd\}$ is τ-marked, with for all $t' \in T$,

$$(t', a_t) \in domain(\tau) \;\Longleftrightarrow\; t = t'$$

(the intuition being to express a type t as the modal formula $\langle a_t\rangle\top$). For τ' given above, we can form the $(T' \cup \{\surd\}, \mathsf{A}', \surd)$-specification

$$\tau' \;\cup\; \{(x, a_x, \surd) \mid x \in T'\}.$$

Let us agree that a set Ψ of modal formulas is *true in* δ if every formula in Ψ is satisfied, relative to δ, by every δ-state. The content of a $(T, \mathsf{A}, \surd)$-specification τ is given by the set

$$\begin{aligned} spec(\tau) \;:=\; &\{\langle a_t\rangle\top \supset \langle a\rangle\top \mid (t, a, \surd) \in \tau\} \cup \\ &\{\langle a_t\rangle\top \supset \langle a\rangle\langle a_{t'}\rangle\top \mid (t, a, t') \in \tau \text{ and } t' \neq \surd\} \end{aligned}$$

of formulas true in an A_T-deterministic system δ precisely if for every $(t, a, t') \in \tau$ and $q \in Q_t$,

$$a \in trace_\delta(q) \;\text{ and }\; \delta(q, a) \in Q_{t'}$$

where for every $t \in T - \{\surd\}$, Q_t is $\{q \in Q \mid a_t \in trace_\delta(q)\}$ and $Q_\surd = Q$. We can express membership in a trace set

$$s \in trace_\delta(q) \;\Longleftrightarrow\; q \models_\delta \langle s\rangle\top$$

through formulas $\langle s\rangle\varphi$ defined by induction on s:

$$\langle\epsilon\rangle\varphi \;:=\; \varphi \;\text{ and }\; \langle as\rangle\varphi \;:=\; \langle a\rangle\langle s\rangle\varphi$$

so that for $a_1a_2\cdots a_n \in \mathsf{A}^n$,

$$\langle a_1a_2\cdots a_n\rangle\varphi \;=\; \langle a_1\rangle\langle a_2\rangle\cdots\langle a_n\rangle\varphi.$$

Given a language L, let us say q *δ-contains* L if $L \subseteq \mathit{trace}_\delta(q)$.

Fact 2. *For any $(T, \mathsf{A}, \surd)$-specification τ, $\mathit{spec}(\tau)$ is true in an A_T-deterministic system $\delta : Q\times\mathsf{A}_T \rightharpoonup Q$ iff for every $t \in T-\{\surd\}$ and $q \in Q$, q δ-contains $\mathit{trace}_\tau(t)$ whenever q δ-contains $\{a_t\}$.*

Under certain assumptions, $\mathit{trace}_\tau(t)$ is finite. More specifically, let $T_0 = \emptyset$ and for any integer $n \geq 0$,

$$T_{n+1} \;:=\; \{t \in T \mid \tau \cap\; (\{t\} \times \mathsf{A} \times T) \subseteq T \times \mathsf{A} \times T_n\}$$

(making $T_1 = \{\surd\}$). For each $t \in T_n$, $\mathit{trace}_\tau(t)$ is finite provided

($\star$) for all $t \in T$, $\{a \in \mathsf{A} \mid (t, a) \in \mathit{domain}(\tau)\}$ is finite.

Although ($\star$) and $T \subseteq \bigcup_n T_n$ can be expected of record types in Cooper (2012), notice that if $(t, a, t) \in \tau$ then $t \notin \bigcup_n T_n$ and $a^* \subseteq \mathit{trace}_\tau(t)$.

3.2 Terminal Attributes, $\Diamond$ and Subtypes

Fact 2 reduces a $(T, \mathsf{A}, \surd)$-specification τ to its trace sets, $\mathit{trace}_\tau(t)$ for $t \in T - \{\surd\}$, unwinding $\mathit{spec}(\tau)$ to

$$\langle a_t\rangle\top \;\supset \bigwedge_{s\in \mathit{trace}_\tau(t)} \langle s\rangle\top \tag{7}$$

for $t \in T - \{\surd\}$ (where the conjunction might be infinite). The converse of (7) follows from $a_t \in \mathit{trace}_\tau(t)$. (7) has the form

$$\varphi_t \;\supset\; \varphi_{t[\tau]}$$

with antecedent $\langle a_t\rangle\top$ construed as a formula φ_t representing t, and consequent $\bigwedge_{s\in \mathit{trace}_\tau(t)}\langle s\rangle\top$ as a formula $\varphi_{t[\tau]}$ representing τ's conception of t. The attribute a_t often has the following property. Given an A-deterministic system δ, let us say an attribute $a \in \mathsf{A}$ is *δ-terminal* if the set

$$\mathit{Tml}_\mathsf{A}(a) \;:=\; \{\neg\langle sab\rangle\top \mid s \in \mathsf{A}^* \text{ and } b \in \mathsf{A}\}$$

of formulas is true in δ — i.e., for every δ-state q, string $s \in \mathsf{A}^*$ and attribute $b \in \mathsf{A}$, $sab \notin \mathit{trace}_\delta(q)$. It is natural to associate a frame such as

with an A-deterministic system δ in which labels on nodes (i.e., *John*, *smash* and *window*) are δ-terminal, while labels on arcs (AGENT, THEME) are not.

Next, we quantify away the strings s mentioned in $Tml_{\mathsf{A}}(a)$ for a useful modal operator $\Diamond$. Given an A-deterministic system $\delta : Q \times \mathsf{A} \rightharpoonup Q$, and states $q, q' \in Q$, we say q' is a *δ-component of* q and write $q \rightsquigarrow_\delta q'$, if q' is $\delta_q(s)$ for some string $s \in \mathsf{A}^*$. With the relation $\rightsquigarrow_\delta$, we extend satisfaction $\models$ to formulas $\Diamond\varphi$

$$q \models \Diamond\varphi \iff (\exists q')\ q \rightsquigarrow_\delta q' \text{ and } q' \models \varphi$$

making $\Diamond\varphi$ essentially the infinitary disjunction $\bigvee_{s\in\mathsf{A}^*}\langle s\rangle\varphi$ and its dual $\Box\varphi := \neg\Diamond\neg\varphi$ the infinitary conjunction $\bigwedge_{s\in\mathsf{A}^*}[s]\varphi$ (where $[s]\varphi := \neg\langle s\rangle\neg\varphi$). The extension preserves invariance under trace equivalence $\sim$

$$\text{whenever } q \sim q' \text{ and } q \models \Diamond\varphi,\ q' \models \Diamond\varphi.$$

That is, for the purpose of $\models$, we can reduce a state q to its trace set $trace_\delta(q)$. Accordingly, we collect all non-empty prefix-closed subsets of A^* in

$$Mod(\mathsf{A}) := \{prefix(L \cup \{\epsilon\}) \mid L \subseteq \mathsf{A}^*\}$$

(where "Mod" is for models) with the understanding that the transitions δ between $q, q' \in Mod(\mathsf{A})$ are given by derivatives

$$\delta(q, a) = q' \iff q_a = q'.$$

Given a set Ψ of modal formulas over A, we form the subset of $Mod(\mathsf{A})$ satisfying every formula of Ψ

$$Mod_{\mathsf{A}}(\Psi) := \{q \in Mod(\mathsf{A}) \mid (\forall\varphi \in \Psi)\ q \models \varphi\}$$

and say Ψ is A-*equivalent to* a set Ψ' of modal formulas over A if they have the same models

$$\Psi \equiv_{\mathsf{A}} \Psi' \iff Mod_{\mathsf{A}}(\Psi) = Mod_{\mathsf{A}}(\Psi').$$

Ψ can be strengthened to

$$\begin{aligned}\Box_{\mathsf{A}}(\Psi) &:= \{\neg\langle s\rangle\neg\varphi \mid s \in \mathsf{A}^* \text{ and } \varphi \in \Psi\}\\ &\equiv_{\mathsf{A}} \{\Box\varphi \mid \varphi \in \Psi\}\end{aligned}$$

requiring that every formula in Ψ holds in all components. Clearly,

$$Tml_{\mathsf{A}}(a) \equiv_{\mathsf{A}} \Box_{\mathsf{A}}(\{\neg\langle ab\rangle\top \mid b \in \mathsf{A}\}).$$

Given representations φ_t and φ_u of types t and u, we can express the subtyping $t \sqsubseteq u$ as the set

$$\text{`}t \sqsubseteq u\text{'} \;:=\; \{\neg\langle s\rangle(\varphi_t \wedge \neg\varphi_u) \mid s \in \mathsf{A}^*\}$$

of modal formulas denying the existence of components of type t but not u. Then '$t \sqsubseteq u$' requires $\varphi_t \supset \varphi_u$ of all components

$$\text{`}t \sqsubseteq u\text{'} \;\equiv_{\mathsf{A}}\; \Box(\varphi_t \supset \varphi_u)$$

bringing us back to the implication (7) above of the form $\varphi_t \supset \varphi_{\tau[t]}$. Inasmuch as an attribute a is represented by the formula $\langle a\rangle\top$, we can speak of a being contained in an attribute b, $a \sqsubseteq b$, when asserting

$$\Box(\langle a\rangle\top \supset \langle b\rangle\top).$$

3.3 Typings and Particulars

We can reduce a typing $p : t$ to a subtyping through the equivalence

$$p : t \;\iff\; \{p\} \sqsubseteq t$$

assuming we can make sense of the singleton type $\{p\}$. Given an A-deterministic system δ and a δ-state q, let us call a formula φ an (A, q)-*nominal* if q satisfies the set

$$\begin{aligned} \mathit{Nom}_{\mathsf{A}}(\varphi) \;&:=\; \{\neg(\langle s'\rangle(\varphi \wedge \langle s\rangle\top) \;\wedge\; \langle s''\rangle(\varphi \wedge \neg\langle s\rangle\top)) \mid s, s', s'' \in \mathsf{A}^*\} \\ &\equiv_{\mathsf{A}} \{\neg(\Diamond(\varphi \wedge \langle s\rangle\top) \;\wedge\; \Diamond(\varphi \wedge \neg\langle s\rangle\top)) \mid s \in \mathsf{A}^*\} \end{aligned}$$

of formulas that together say any two δ-components that δ-satisfy φ δ-contain the same languages over A. We can rephrase $\mathit{Nom}_{\mathsf{A}}(\varphi)$ as implications

$$\begin{aligned} \mathit{Nom}_{\mathsf{A}}(\varphi) \;&\equiv_{\mathsf{A}} \{\Diamond(\varphi \wedge \langle s\rangle\top) \supset \Box(\varphi \supset \langle s\rangle\top) \mid s \in \mathsf{A}^*\} \\ &\equiv_{\mathsf{A}} \{\Diamond(\varphi \wedge \psi) \supset \Box(\varphi \supset \psi) \mid \psi \in \Phi_{\mathsf{A}}\} \end{aligned}$$

making δ-components that δ-satisfy φ Φ_{A}-indistinguishable.[4]

Fact 3. *Let δ be an A-deterministic system, q be a δ-state and φ be an (A, q)-nominal.*

(i) For all δ-components q' and q'' of q,

$$q' \models \varphi \text{ and } q'' \models \varphi \;\; \textit{implies} \;\; q' \sim q''.$$

[4] $\mathit{Nom}_{\mathsf{A}}(\varphi)$ does the work of the scheme (Nom_N) for nominals given in Blackburn (1993), just as $\mathit{Tml}_{\mathsf{A}}(a)$ is analogous to the scheme (*Term*) there for instantiating atomic information at terminal nodes.

(ii) There is a δ-component of q that δ-satisfies φ and ψ

$$q \models \Diamond(\varphi \wedge \psi)$$

iff every δ-component of q satisfies $\varphi \supset \psi$ and some δ-component of q δ-satisfies φ

$$q \models \Box(\varphi \supset \psi) \ \wedge \ \Diamond\varphi.$$

Now, an (A, q)-*particular* is an (A, q)-nominal φ such that q has a δ-component that δ-satisfies φ — i.e., in addition to all formulas in $Nom_{\mathsf{A}}(\varphi)$, q satisfies $\Diamond\varphi$. We say (A, q) *verifies* $p : t$ if

$$\varphi_{\{p\}} \text{ is an } (\mathsf{A}, q)\text{-particular and } q \models \Box(\varphi_{\{p\}} \supset \varphi_t).$$

Part (ii) of Fact 3 says this is equivalent to $\varphi_{\{p\}}$ being an (A, q)-nominal and q having a δ-component that δ-satisfies $\varphi_{\{p\}} \wedge \varphi_t$. Note that if q is δ-marked by a_q then $\langle a_q\rangle\top$ is an (A, q')-nominal for all δ-states q'. The weaker notion of an (A, q)-nominal φ has the advantage over $\langle a_q\rangle\top$ that the set $Nom_{\mathsf{A}}(\varphi)$ of modal formulas allows the modal formulas satisfied by a state δ-satisfying $\varphi_{\{p\}}$ to vary with δ.[5] We can use the set $Nom_{\mathsf{A}}(\varphi) \cup \{\Diamond\varphi\}$ to lift the notion of an (A, q)-particular to that of a (A, Ψ)-*particular*, given a set Ψ of modal formulas, requiring that $Nom_{\mathsf{A}}(\varphi) \cup \{\Diamond\varphi\}$ follow from Ψ in that

$$Mod_{\mathsf{A}}(\Psi) \ \subseteq \ Mod_{\mathsf{A}}(Nom_{\mathsf{A}}(\varphi) \cup \{\Diamond\varphi\}).$$

Having described how a particular can be understood as a formula (or attribute a via $\langle a\rangle\top$), it is perhaps worth emphasizing that we need *not* understand particulars as formulas (or attributes). In the present context, particulars are first and foremost δ-states in an A-deterministic system δ. For any δ-state q, we can speak of types to which q belongs with*out* introducing an attribute that (relative to an extension of δ) represents q. What is important is that we build into A all the structure in particulars we wish to analyze.

4 Finite Approximations and Signatures

Throughout this section, we shall work with a set A of labels that is large enough so that we may assume trace equivalence $\sim$ is equality. Identity of indiscernibles in an A-deterministic system δ reduces a state q to $trace_\delta(q)$, allowing us to identify a state with a prefix-closed, non-empty subset of A^*. As huge as A can be, we can form the set $Fin(\mathsf{A})$ of finite subsets of A and use the equality

$$\mathsf{A}^* \ = \ \bigcup_{\Sigma \in Fin(\mathsf{A})} \Sigma^*$$

[5] By contrast, for a language q over A, all formulas in the set

$$\{\Box(\langle a_q\rangle\top \supset (\langle s\rangle\top \wedge \neg\langle s'\rangle\top)) \mid s \in q \text{ and } s' \in \mathsf{A}^* - q\}$$

must be satisfied for a_q to mark q.

to approach a language $L \subseteq \mathsf{A}^*$ via its intersections $L \cap \Sigma^*$ (for $\Sigma \in \mathit{Fin}(\mathsf{A})$)

$$L \;=\; \bigcup_{\Sigma \in \mathit{Fin}(\mathsf{A})} (L \cap \Sigma^*)$$

at the cost of bounding discernibility to Σ. For $X \in \mathit{Fin}(\mathsf{A}) \cup \{\mathsf{A}\}$, we define an *X-state* to be a non-empty, prefix-closed subset q of X^* on which transitions are given by derivatives

$$\delta(q, a) = q_a \quad \text{for } a \in \mathsf{A} \cap q$$

making $\delta_q(s) = q_s$ for $s \in q$. Henceforth, we put the notation δ aside, but track the parameter X, which we set to a finite subset Σ of A to pick out what is of particular interest.

Two subsections make up the present section. The first illustrates how the set A of labels can explode; the second how, after this explosion, to keep a grip on satisfaction $\models$ of formulas. We associate the formulas with sets $X \in \mathit{Fin}(\mathsf{A}) \cup \{\mathsf{A}\}$, defining $\mathit{sen}(X)$ to be the set of formulas generated from $a \in X$ and $Y \subseteq X$ according to

$$\varphi \;::=\; \top \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle a \rangle \varphi \mid \Box_Y \varphi.$$

We interpret these formulas over A-states q, treating $\top$, $\neg$ and $\wedge$ as usual, and setting

$$q \models \langle a \rangle \varphi \quad \Longleftrightarrow \quad a \in q \text{ and } q_a \models \varphi$$

which generalizes to strings $s \in X^*$

$$q \models \langle s \rangle \varphi \quad \Longleftrightarrow \quad s \in q \text{ and } q_s \models \varphi$$

(recalling that $\langle a_1 \rangle \cdots \langle a_n \rangle \varphi$ is $\langle a_1 \rangle \cdots \langle a_n \rangle \varphi$). As for $\Box_Y$, we put

$$q \models \Box_Y \varphi \quad \Longleftrightarrow \quad (\forall s \in q \cap Y^*)\; q_s \models \varphi$$

relativizing $\Box$ to Y for reasons that will become clear below. When $\Box$ appears with no subscript, it is understood to carry the subscript A; i.e., $\Box$ is $\Box_{\mathsf{A}}$. Also, for $s \in \mathsf{A}^*$, we will often shorten the formula $\langle s \rangle \top$ to s, writing, for example, $\neg a$ for $\neg \langle a \rangle \top$.

4.1 Labels Refining Partitions

For any $\Sigma \in \mathit{Fin}(\mathsf{A})$ and Σ-state q, we can formulate the Myhill-Nerode equivalence $\approx_q$ between strings $s, s' \in \Sigma^*$ with the same extensions in q

$$s \approx_q s' \quad \Longleftrightarrow \quad (\forall w \in \Sigma^*)\; (sw \in q \iff s'w \in q)$$

(e.g., Hopcroft and Ullman 1979) in terms of derivatives

$$s \approx_q s' \quad \Longleftrightarrow \quad q_s = q_{s'}$$

from which it follows that q is regular iff its set

$$\{q_s \mid s \in \Sigma^*\}$$

of derivatives is finite. There are strict bounds on what we can discern with Σ and Σ-states. For example, the regular language

$$q' \;=\; \textsc{father}^* + \textsc{father}^* man$$

over $\Sigma' = \{\textsc{father}, man$ is a model of

$$man \;\wedge\; \Box(man \;\supset\; \langle \textsc{father} \rangle man)$$

(i.e., q' is a token of the record type man given by $\mathrm{eq}(man) = \{(\textsc{father}, man)\}$), with derivatives

$$q'_s \;=\; \begin{cases} q' & \text{if } s \in \textsc{father}^* \\ \{\epsilon\} & \text{if } s \in \textsc{father}^* man \\ \emptyset & \text{otherwise} \end{cases}$$

so that according to the definitions from the previous section (with δ given by derivatives of q'), the Σ'-state $\{\epsilon\}$ is null, and the label man is terminal. The Σ'-state q' does not differentiate between distinct tokens of man, although it is easy enough to introduce additional labels and formulas

$$\Box\,(man \;\supset\; (John \;\vee Peter \vee Other_{man:John,Peter})) \tag{8}$$

with

$$\Box\,(John \;\supset\; \neg\langle \textsc{father} \rangle John)$$

unless say, $John$ could apply to more than one particular, as suggested by

$$\Box\,(John \;\supset\; (John1 \;\vee John2 \vee Other_{John:1,2})).$$

Generalizing line (8) above, we can refine a label a to a finite partition from a set $\Sigma \in Fin(\mathsf{A})$, asserting

$$Partition_a(\Sigma) \;\;:=\;\; \Box\,(a \supset Uniq(\Sigma))$$

where $Uniq(\Sigma)$ picks out exactly one label in Σ

$$Uniq(\Sigma) \;\;:=\;\; \bigvee_{a \in \Sigma} (a \wedge \bigwedge_{a' \in \Sigma - \{a\}} \neg a')).$$

Co-occurrence restrictions on a set Σ of alternatives

$$Alt(\Sigma) \;\;:=\;\; \Box \bigwedge_{a \in \Sigma} \bigwedge_{a' \in \Sigma - \{a\}} \neg(a \wedge a')$$

(declaring any pair to be incompatible) is equivalent to the conjunction

$$\bigwedge_{a \in \Sigma} Partition_a(\Sigma).$$

And if the labels in Σ are understood to specify components, we might say a label marks a Σ-atom if it rules out a'-components for $a' \in \Sigma$

$$\begin{aligned} \mathit{Atom}_\Sigma(a) &:= \Box(a \supset \bigwedge_{a' \in \Sigma} \neg a') \\ &\iff \mathit{Partition}_a(\Sigma \cup \{a\}). \end{aligned}$$

That said, we arrived at $\mathit{Partition}_a(\Sigma)$ from *man* above through the example $\Sigma = \{\mathit{John},\ \mathit{Peter},\ \mathit{Other}_{man:John,Peter}\}$ of labels that differentiate between tokens of *man* rather than (as in the case of the record label FATHER or AGENT or THEME) specifying components. We can extend the example

$$\mathit{Partition}_{man}(\{\mathit{John}, \mathit{Peter}, \mathit{Other}_{man:John,Peter}\})$$

through a function $f : T \to \mathit{Fin}(\mathsf{A})$ from some finite set T of labels (representing types) such that for $a \in T$, $f(a)$ outlines a partition of a (just as $\{\mathit{John},\ \mathit{Peter},\ \mathit{Other}_{man:John,Peter}\}$ does for *man*). An *f-token* is then an A-state q such that

$$q \models \bigwedge_{a \in T} \mathit{Partition}_a(f(a))$$

making (as it were) a choice from $f(a)$, for each $a \in T$.

4.2 Reducts for Satisfaction

Given a string $s \in \mathsf{A}^*$ and a set $\Sigma \in \mathit{Fin}(\mathsf{A})$, the longest prefix of s that belongs to Σ^* is computed by the function $\pi_\Sigma : \mathsf{A}^* \to \Sigma^*$ defined by $\pi_\Sigma(\epsilon) := \epsilon$ and

$$\pi_\Sigma(as) := \begin{cases} a\,\pi_\Sigma(s) & \text{if } a \in \Sigma \\ \epsilon & \text{otherwise.} \end{cases}$$

The *Σ-reduct of* a language $q \subseteq \mathsf{A}^*$ is the image of q under π_Σ

$$q \restriction \Sigma := \{\pi_\Sigma(s) \mid s \in q\}\ .$$

If q is an A-state, then its Σ-reduct, $q \restriction \Sigma$, is a Σ-state and is just the intersection $q \cap \Sigma^*$ with Σ^*. Σ-reducts are interesting because satisfaction $\models$ of formulas in $\mathit{sen}(\Sigma)$ can be reduced to them.

Fact 4. *For every $\Sigma \in \mathit{Fin}(\mathsf{A})$, $\varphi \in \mathit{sen}(\Sigma)$ and A-state q,*

$$q \models \varphi \iff q \restriction \Sigma \models \varphi$$

and if, moreover, $s \in q \restriction \Sigma$, then

$$q \models \langle s \rangle \varphi \iff (q \restriction \Sigma)_s \models \varphi. \tag{9}$$

Fact 4 is proved by a routine induction on $\varphi \in sen(\Sigma)$. Were $sen(\Sigma)$ closed under $\Box = \Box_{\mathsf{A}}$, Fact 4 would fail for infinite A; hence, the relativized operators $\Box_Y$ for $Y \subseteq \Sigma$ in $sen(\Sigma)$.

There is structure lurking around Fact 4 that is most conveniently described in category-theoretic terms. For $\Sigma \in Fin(\mathsf{A})$, let $Q(\Sigma)$ be the category with

- Σ-states q as objects.
- pairs (q, s) such that $s \in q$ as morphisms from q to q_s, composing by concatenating strings

$$(q, s)\ ;\ (q_s, s')\ :=\ (q, ss')$$

with identities (q, ϵ).

To turn Q into a functor from $Fin(\mathsf{A})^{op}$ (with morphisms (Σ', Σ) such that $\Sigma \subseteq \Sigma' \in Fin(\mathsf{A})$) to the category **Cat** of small categories, we map a $Fin(\mathsf{A})^{op}$-morphism (Σ', Σ) to the functor $Q(\Sigma', \Sigma) : Q(\Sigma') \to Q(\Sigma)$ sending a Σ'-state q' to the Σ-state $q' \restriction \Sigma$, and the $Q(\Sigma')$-morphism (q', s') to the $Q(\Sigma)$-morphism $(q' \restriction \Sigma, \pi_\Sigma(s'))$. The *Grothendieck construction for* Q is the category $\int Q$ where

- objects are pairs (Σ, q) such that $\Sigma \in Fin(\mathsf{A})$ and q is a Σ-state.
- morphisms from (Σ', q') to (Σ, q) are pairs $((\Sigma', \Sigma), (q'', s))$ of $Fin(\mathsf{A})^{op}$-morphisms (Σ', Σ) and $Q(\Sigma)$-morphisms (q'', s) such that $q'' = q' \restriction \Sigma$ and $q = q''_s$.

(e.g., Tarlecki et al. 1991). $\int Q$ integrates the different categories $Q(\Sigma)$ (for $\Sigma \in Fin(\mathsf{A})$), lifting a $Q(\Sigma)$-morphism (q, s) to a $(\int Q)$-morphism from (Σ', q') to (Σ, q_s) whenever $\Sigma \subseteq \Sigma'$ and $q' \restriction \Sigma = q$.

Given a small category C, let us write $|C|$ for the set of objects of C. Thus, for $\Sigma \in Fin(\mathsf{A})$, $|Q(\Sigma)|$ is the set

$$|Q(\Sigma)|\ =\ \{q \subseteq \Sigma^* \mid q \neq \emptyset \text{ and } q \text{ is prefix}-\text{closed}\}$$

of Σ-states. Next, for $(\Sigma, q) \in |\int Q|$, let $Mod(\Sigma, q)$ be the full subcategory of $Q(\Sigma)$ with objects required to have q as a subset

$$|Mod(\Sigma, q)|\ :=\ \{q' \in |Q(\Sigma)| \mid q \subseteq q'\}.$$

That is, $|Mod(\Sigma, q)|$ is the set of Σ-states q' such that for all $s \in q$, $q' \models \langle s \rangle \top$. The intuition is that q is a form of record typing over Σ that allows us to simplify clauses such as

$$q' \models \langle s \rangle \varphi \iff s \in q' \text{ and } {q'}_s \models \varphi \tag{10}$$

when $s \in q \subseteq q'$. The second conjunct in the righthand side of (10), $q'_s \models \varphi$, presupposes the first conjunct, $s \in q'$. We can lift that presupposition out of (10) by asserting that whenever $s \in q$ and $q \subseteq q'$,

$$q' \models \langle s \rangle \varphi \iff q'_s \models \varphi.$$

This comes close to the equivalence (9) in Fact 4, except that Σ-reducts are missing. These reducts appear once we vary Σ, and step from $Q(\Sigma)$ to $\int Q$. Taking this step, we turn the categories $Mod(\Sigma, q)$ to a functor Mod from $\int Q$ to **Cat**, mapping a $\int Q$-morphism $\sigma = ((\Sigma', \Sigma), (q' \restriction \Sigma, s))$ from (Σ', q') to (Σ, q) to the functor

$$Mod(\sigma) : Mod(\Sigma', q') \to Mod(\Sigma, q)$$

sending $q'' \in |Mod(\Sigma', q')|$ to the s-derivative of its Σ-reduct, $(q'' \restriction \Sigma)_s$, and a $Mod(\Sigma', q')$-morphism (q'', s') to the $Mod(\Sigma, q)$-morphism $(q'' \restriction \Sigma, \pi_\Sigma(s'))$.

The syntactic counterpart of $Q(\Sigma)$ is $sen(\Sigma)$, which we turn into a functor sen matching Mod. A basic insight from Goguen and Burstall (1992) informing the present approach is the importance of a category **Sign** of *signatures* which the functor sen maps to the category **Set** of sets (and functions) and which Mod maps contravariantly to **Cat**. The definition of Mod above suggests that $\mathbf{Sign}^{op}$ is $\int Q$.[6] A $\int Q$-morphism from $\int Q$-objects (Σ', q') to (Σ, q) is determined uniquely by a string $s \in q' \restriction \Sigma$ such that

$$q = (q' \restriction \Sigma)_s \text{ and } \Sigma \subseteq \Sigma'. \tag{11}$$

Let $(\Sigma, q) \xrightarrow{s} (\Sigma', q')$ abbreviate the conjunction (11), which holds precisely if $((\Sigma', \Sigma), (q' \restriction \Sigma, s))$ is a $\int Q$-morphism from (Σ', q') to (Σ, q). Now for $(\Sigma, q) \in |\int Q|$, let

$$sen(\Sigma, q) \;=\; sen(\Sigma)$$

(ignoring q), and when $(\Sigma, q) \xrightarrow{s} (\Sigma', q')$, let

$$sen(\sigma) : sen(\Sigma) \to sen(\Sigma')$$

send $\varphi \in sen(\Sigma)$ to $\langle s \rangle \varphi \in sen(\Sigma')$. To see that an *institution* arises from restricting $\models$ to $|Mod(\Sigma, q)| \times sen(\Sigma)$, for $(\Sigma, q) \in |\int Q|$, it remains to check the *satisfaction condition*:

$$\text{whenever } (\Sigma, q) \xrightarrow{s} (\Sigma', q') \text{ and } q'' \in |Mod(\Sigma', q')| \text{ and } \varphi \in sen(\Sigma),$$

$$q'' \models \langle s \rangle \varphi \iff (q'' \restriction \Sigma)_s \models \varphi.$$

This follows from Fact 4 above, as s must be in $q' \restriction \Sigma$ and thus also in $q'' \restriction \Sigma$.

[6] That said, we might refine **Sign**, requiring of a signature (Σ, q) that q be a regular language. For this, it suffices to replace $\int Q$ by $\int R$ where $R : Fin(\mathsf{A})^{op} \to \mathbf{Cat}$ is the subfunctor of Q such that $R(\Sigma)$ is the full subcategory of $Q(\Sigma)$ with objects regular languages. We can make this refinement with*out* requiring that Σ-states in $Mod(\Sigma, q)$ be regular, forming $Mod(\Sigma, q)$ from Q (not R).

5 Conclusion

A process perspective is presented in Sect. 2 that positions frames to the left of a satisfaction predicate $\models$ for Hennessy-Milner logic over a set A of labels (or, from the perspective of Blackburn 1993, attributes). A is allowed to become arbitrarily large so that under identity of indiscernibles relative to A, a frame can be identified with a non-empty prefix-closed language over A. This identification is tried out in Sect. 3 on frames as types and particulars. A handle on A is provided by its finite subsets Σ, which are paired with languages $q \subseteq \Sigma^*$ for signatures (Σ, q), along which to reduce satisfaction to Σ-reducts and/or s-derivatives, for $s \in q$ (Fact 4). This plays out themes (mentioned in the introduction) of "semantics in flux" and "natural languages as collections of resources" (from Cooper and Ranta) in that, oversimplifying somewhat, s-derivatives specify transitions, while Σ-reducts pick out resources to use. The prominence of transitions (labeled by A) here contrasts strikingly with a finite-state approach to events (most recently described in Fernando 2015), where a single string (representing a timeline) appears to the left of a satisfaction predicate.[7] A Σ-state q to the left of $\models$ above offers a choice of strings, any number of which might be combined with other strings from other Σ'-states over different alphabets Σ'. A combination can be encoded as a string describing a timeline of resources used. This type/token distinction between languages and strings to the left of satisfaction has a twist; the languages are conceptually prior to the strings representing timelines, as nonexistent computer programs cannot run. Indeed, a profusion of alphabets Σ and Σ-states compete to make, in some form or other, a timeline that has itself a bounded signature (of a different kind). The processes through which a temporal realm is pieced together from bits of various frames call out for investigation.

While much remains to be done, let us be clear about what is offered above. A frame is structured according to strings of labels, allowing the set Σ of labels to vary over finite sets. That variation is tracked by a signature (Σ, q) picking out non-empty prefix-closed languages over Σ that contain the set q of strings over Σ. For example, Cooper's meaning function

$$\left(\lambda r : \begin{bmatrix} \text{AGENT} : \mathit{all} \\ \text{THEME} : \mathit{all} \end{bmatrix}\right) \begin{bmatrix} p_1 : \mathit{smash}(r) \\ p_2 : \mathit{animate}(r.\text{AGENT}) \\ p_3 : \mathit{concrete}(r.\text{THEME}) \end{bmatrix}$$

is approximated by the signature (Σ, q) as the language L, where

$$\begin{aligned} \Sigma &= \{\text{AGENT},\ \text{THEME},\ \mathit{smash},\ \mathit{animate},\ \mathit{concrete}\} \\ q &= \{\text{AGENT},\ \text{THEME},\ \epsilon\} \\ L &= q \cup \{\mathit{smash},\ \text{AGENT}\ \mathit{animate},\ \text{THEME}\ \mathit{concrete}\}. \end{aligned}$$

Further constraints can be imposed through formulas built with Boolean connectives and modal operators dependent on Σ — for instance, $\mathit{Nom}(\langle a \rangle \top)$. The

[7] This is formulated as an institution in Fernando (2014).

possibility of expanding Σ to a larger set makes the notion of identity as Σ-indiscernibility open-ended, and Σ a bounded but refinable level of granularity. A measure of satisfaction is taken in a finite-state calculus with, as Conway (1971) puts it, Taylor series

$$L \;=\; o(L) + \sum_{a \in \Sigma} aL_a$$

(from derivatives L_a), and a Grothendieck signature

$$\textbf{Sign} \;=\; \int Q.$$

Acknowledgements. My thanks to Robin Cooper for discussions, to Glyn Morrill for help with presenting this paper at Formal Grammar 2015, and to two referees for comments and questions.

References

Barsalou, L.: Perceptual symbol systems. Behav. Brain Sci. **22**, 577–660 (1999)

Blackburn, P.: Modal logic and attribute value structures. In: de Rijke, M. (ed.) Diamonds and Defaults. Synthese Library, vol. 229, pp. 19–65. Springer, Netherlands (1993)

Brzozowski, J.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)

Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, London (1971)

Cooper, R.: Type theory and semantics in flux. In: Philosophy of Linguistics, pp. 271–323. North-Holland (2012)

Cooper, R., Ranta, A.: Natural languages as collections of resources. In: Language in Flux: Dialogue Coordination, Language Variation, Change and Evolution, pp. 109–120. College Publications, London (2008)

Davidson, D.: The logical form of action sentences. In: The Logic of Decision and Action, pp. 81–95. University of Pittsburgh Press (1967)

Fernando, T.: Incremental semantic scales by strings. In: Proceedings of EACL 2014 Workshop on Type Theory and Natural Language Semantics, pp. 63–71. ACL (2014)

Fernando, T.: The semantics of tense and aspect: a finite-state perspective. In: Lappin, S., Fox, C. (eds.) The Handbook of Contemporary Semantic Theory, 2nd edn. Wiley, New York (2015)

Fillmore, C.: Frame semantics. In: Linguistics in the Morning Calm, pp. 111–137. Hanshin Publishing Co., Seoul (1982)

Goguen, J., Burstall, R.: Institutions: abstract model theory for specification and programming. J. ACM **39**(1), 95–146 (1992)

Hennessy, M., Milner, R.: Algebraic laws for non-determinism and concurrency. J. ACM **32**(1), 137–161 (1985)

Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)

Kallmeyer, L., Osswald, R.: Syntax-driven semantic frame composition in lexicalized tree adjoining grammars. J. Lang. Model. **1**(2), 267–330 (2013)

Löbner, S.: Evidence for frames from human language. In: Gamerschlag, T., Gerland, D., Osswald, R., Petersen, W. (eds.) Frames and Concept Types. Studies in Linguistics and Philosophy, vol. 94, pp. 23–67. Springer, Switzerland (2014)

Muskens, R.: Data semantics and linguistic semantics. In: The Dynamic, Inquisitive, and Visionary Life of $\phi, ?\phi$, and $\Diamond\phi$: A Festschrift for Jeroen Groenendijk, Martin Stokhof, and Frank Veltman, pp. 175–183. Amsterdam (2013)

Osswald, R.: Semantics for attribute-value theories. In: Proceedings of Twelfth Amsterdam Colloquium, pp. 199–204. Amsterdam (1999)

Petersen, W., Osswald, T.: Concept composition in frames: focusing on genitive constructions. In: Gamerschlag, T., Gerland, D., Osswald, R., Petersen, W. (eds.) Frames and Concept Types. Studies in Linguistics and Philosophy, vol. 94, pp. 243–266. Springer, Switzerland (2014)

Rutten, J.J.M.M.: Automata and coinduction (an exercise in coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)

Tarlecki, A., Burstall, R., Goguen, J.: Some fundamental algebraic tools for the semantics of computation: Part 3. indexed categories. Theoret. Comput. Sci. **91**(2), 239–264 (1991)

Veltman, F.: Logics for conditionals. Ph.D. dissertation, University of Amsterdam (1985)

Formal Grammar 2015: Contributed Papers

Cyclic Multiplicative-Additive Proof Nets of Linear Logic with an Application to Language Parsing

Vito Michele Abrusci and Roberto Maieli(✉)

Department of Mathematics and Physics, Roma Tre University,
Largo San Leonardo Murialdo, 1, 00146 Rome, Italy
{abrusci,maieli}@uniroma3.it

Abstract. This paper concerns a logical approach to natural language parsing based on proof nets (PNs), i.e. de-sequentialized proofs, of linear logic (LL). It first provides a syntax for proof structures (PSs) of the cyclic multiplicative and additive fragment of linear logic (CyMALL). A PS is an oriented graph, weighted by boolean monomial weights, whose conclusions Γ are endowed with a cyclic order σ. Roughly, a PS π with conclusions $\sigma(\Gamma)$ is correct (so, it is a proof net), if any slice $\varphi(\pi)$, obtained by a boolean valuation φ of π, is a multiplicative (CyMLL) PN with conclusions $\sigma(\Gamma_r)$, where Γ_r is an additive resolution of Γ, i.e. a choice of an additive subformula for each formula of Γ. The correctness criterion for CyMLL PNs can be considered as the non-commutative counterpart of the famous Danos-Regnier (DR) criterion for PNs of the pure multiplicative fragment (MLL) of LL. The main intuition relies on the fact that any DR-switching (i.e. any correction or test graph for a given PN) can be naturally viewed as a seaweed, that is, a rootless planar tree inducing a cyclic order on the conclusions of the given PN. Unlike the most part of current syntaxes for non-commutative PNs, our syntax allows a sequentialization for the full class of CyMALL PNs, without requiring these latter to be cut-free.

One of the main contributions of this paper is to provide a characterization of CyMALL PNs for the additive Lambek Calculus and thus a geometrical (non inductive) way to parse sentences containing words with syntactical ambiguity (i.e., with polymorphic type).

1 Introduction

Proof nets (PNs) are one of the most innovative inventions of linear logic (LL, [7]): they are used to represent demonstrations in a geometric (i.e., non inductive) way, abstracting away from the technical bureaucracy of sequential proofs. Proof nets quotient classes of derivations that are equivalent up to some irrelevant permutations of inference rules instances.

In this spirit, we present a syntax for PNs of the cyclic multiplicative and additive fragment of linear logic (CyMALL, Sect. 1.1). This syntax, like the original one of Girard [8], is based on weighted (by boolean monomials) proof structures with explicit binary contraction links (Sect. 2). The conclusions Γ (i.e.,

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 43–59, 2016.
DOI: 10.1007/978-3-662-53042-9_3

a sequence of formula occurrences) of any PS are endowed with a cyclic order σ on Γ. Naively, a CyMALL PS π with conclusions $\sigma(\Gamma)$ is correct if, for any slice $\varphi(\pi)$, obtained by a boolean valuation φ of π, there exists an additive resolution (i.e., a multiplicative restriction of $\varphi(\pi)$) that is a CyMLL PN with conclusion $\sigma(\Gamma_r)$, where Γ_r is an additive resolution of Γ (i.e. a choice of an additive sub-formula for each formula of Γ). In turn, the correctness criterion for CyMLL PNs can be considered as the non-commutative counterpart of the famous Danos-Regnier (DR) criterion for proof nets of linear logic (see [5,6]). The main intuition relies on the fact that any *DR-switching for a PS* (i.e. any correction or test graph, obtained by mutilating one premise of each disjunction ∇-link) can be naturally viewed as a rootless planar tree, called a *seaweed*, inducing a *cyclic ternary relation* on the conclusions of the given proof structure.

Unlike some previous syntaxes for non-commutative logic, like e.g., [3,13], this new syntax admits a *sequentialization* (i.e., a correspondence with sequential proofs) for the full class of CyMLL PNs including those ones with cuts. Actually, the presence of cut links is "rather tricky" in the non-commutative case, since cut links are not equivalent, from a topological point of view, to tensor links (like in the commutative MLL case): indeed, tensor links make new conclusions appear that may disrupt the original (i.e., in presence of cut links) order of conclusions.

CyMALL PNs satisfy a simple (lazy) convergent cut-elimination procedure (Sect. 2.2) in Laurent-Maieli's style [12]: our strategy relies on the notion of *dependency graph of an eigen weight p* (Definition 9), that is, the smallest $\&_p$-box that must be duplicated in a commutative $\&_p/C$-cut reduction step [14]. Moreover, cut-reduction preserves PNs sequentialization (Sect. 2.3).

CyMALL can be considered as a conservative classical extension of Lambek Calculus (LC, see [1,11,17]) one of the ancestors of LL. The LC represents the first attempt of the so called *parsing as deduction*, i.e., parsing of natural language by means of a logical system. Following [4], in LC, parsing is interpreted as type checking in the form of theorem proving of Gentzen sequents. Types (i.e. propositional formulas) are associated to words in the lexicon; when a string $w_1...w_n$ is tested for grammaticality, the types $t_1, ..., t_n$ associated with the words are retrieved from the lexicon and then parsing reduces to proving the derivability of a one-sided sequent of the form $\vdash t_n^{\perp}, ..., t_1^{\perp}, s$, where s is the type associated with sentences. Moreover, forcing constraints on the Exchange rule, by e.g. allowing only *cyclic permutations* over sequents of formulas, gives the required computational control needed to view theorem proving as parsing in Lambek Categorial Grammar style. Anyway, LC parsing presents some syntactical ambiguity problems; actually, there may be:

1. (*non canonical proofs*) more than one cut-free proof for the same sequent;
2. (*lexical polymorphism*) more than one type associated with a single word.

Now, proof nets are commonly considered an elegant solution to the first problem of representing canonical proofs; in this sense, in Sect. 3, we give an *embedding* of extended MALL Lambek Calculus into Cyclic MALL PNs. Concerning the second problem, in Sect. 4, we propose a parsing approach based on CyMALL PNs that could be considered a step towards a proof-theoretical solution to

the problem of lexical polymorphism. Technically, CyMALL proof nets allow to manage formulas superposition (types polymorphism) by means of the additive &-links, or dually, $\oplus$-links. By means of Lambek CyMALL PNs, we propose the parsing of some sentences, suggested by [18], which make use of polymorphic words; naively, when a word has two possible formulas A and B assigned, then we can combine (or super-pose) these into a single additive formula $A\&B$.

1.1 The Cyclic MALL Fragment of Linear Logic

We briefly recall the necessary background of the *Cyclic MALL fragment* of LL, denoted *CyMALL*, without units (see [1]). We arbitrarily assume literals $a, a^\perp, b, b^\perp, ...$ with a polarity: *positive* (+) for atoms, $a, b, ...$ and *negative* (−) for their duals $a^\perp, b^\perp ....$ A *formula* is built from literals by means of the two groups of connectives: *negative*, ∇ ("par") and & ("with") and *positive*, $\otimes$ ("tensor") and $\oplus$ ("plus"). For these connectives we have the following De Morgan laws: $(A \otimes B)^\perp = B^\perp \nabla A^\perp$, $(A \nabla B)^\perp = B^\perp \otimes A^\perp$, $(A\&B)^\perp = B^\perp \oplus A^\perp$, $(A \oplus B)^\perp = B^\perp \& A^\perp$. A CyMALL (resp., CyMLL) proof is any derivation tree built by the following (resp., by only *identities and multiplicative*) inference rules where sequents Γ, Δ are sequences of formula occurrences endowed with a *total cyclic order* (or *cyclic permutation*).

identities:
$$\frac{}{\vdash A, A^\perp}\ axiom \qquad\qquad \frac{\vdash \Gamma, A \qquad A^\perp \Delta}{\vdash \Gamma, \Delta}\ cut$$

multiplicatives:
$$\frac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\ \otimes \qquad\qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \nabla B}\ \nabla$$

additives:
$$\frac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A\&B}\ \& \qquad\qquad \frac{\vdash \Gamma, A_i}{\vdash \Gamma, A_1 \oplus_i A_2}\ \oplus_{i=1,2}$$

A *total cyclic order* can be thought as follows; consider a set of points of an oriented circle; the orientation induces a total order on these points as follows: if a, b and c are three distinct points, then b is either between a and c ($a < b < c$) or between c and a ($c < b < a$). Moreover, $a < b < c$ is equivalent to $b < c < a$ or $c < a < b$.

Definition 1 (total cyclic order). *A* total cyclic order *is a pair* (X, σ) *where* X *is a set and* σ *is a ternary relation over* X *satisfying the following properties:*

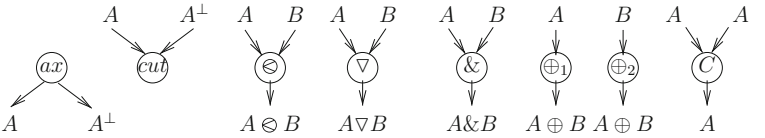
1. $\forall a, b, c \in X, \sigma(a, b, c) \rightarrow \sigma(b, c, a)$ (cyclic),
2. $\forall a, b \in X, \neg\sigma(a, a, b)$ (anti-reflexive),
3. $\forall a, b, c, d \in X, \sigma(a, b, c) \wedge \sigma(c, d, a) \rightarrow \sigma(b, c, d)$ (transitive),
4. $\forall a, b, c \in X, \sigma(a, b, c) \vee \sigma(c, b, a)$ (total).

Negative (or *asynchronous*) connectives correspond to *true determinism* in the way we apply bottom-up their corresponding inference rules. In particular, observe that Γ must appear as the same context (with the same order) in both premises of the &-rule. Positive (or *synchronous*) connectives correspond to *true*

non-determinism in the way we apply, bottom-up, their corresponding rules; there is no deterministic way to split, bottom up, the context (Γ, Δ) in the $\otimes$-rule; similarly, there not exist a deterministic way to select, bottom up, $\oplus_1$ or $\oplus_2$-rule.

2 Cyclic MALL Proof Structures

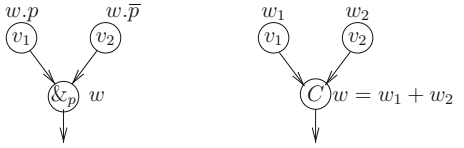
Definition 2 (CyMALL proof structure). *A* CyMALL proof structure (PS) *is an oriented graph π whose edges (resp., nodes) are labeled by formulas (resp., by connectives) of CyMALL and built by juxtaposing the following special graphs, called* links*, in which incident (resp., emergent) edges are called* premises *(resp.,* conclusions*):*



In a PS each premise (resp., conclusion) of a link must be conclusion (resp., premise) of exactly (resp., at most) one link. We call conclusion of a PS *any emergent edge that is not premises of any link. We call* CyMLL proof structure*, any PS built by only means of axioms, cut and multiplicative links ($\otimes, \nabla$).*

Definition 3 (Girard CyMALL proof structure). *A* Girard proof structure (GPS) *is a PS with* weights *associated as follows (a* weights assignment*):*

1. *first we associate a* boolean variable*, called* eigen weight *p, to each &-node (eigen weights are supposed to be different);*
2. *then we associate a* weight*, a product of (negation of) boolean variables $(p, \overline{p}, q, \overline{q}...)$ to each node, with the constraint that two nodes have the same weight if they have a common edge, except when the edge is the premise of a & or C-node; in these cases we do like below:*



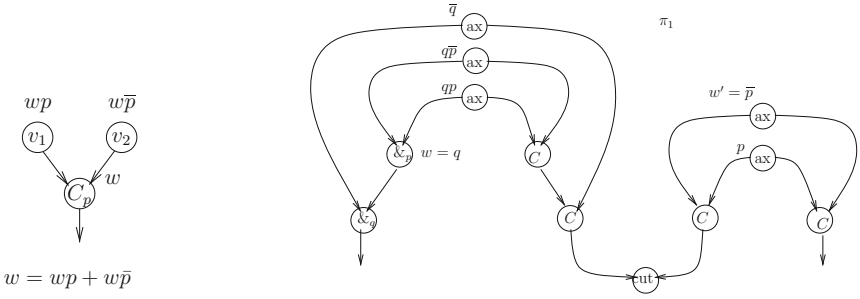
if p does not occur in w with $w_1.w_2 = 0$

3. *a* conclusion node *has weight* 1*;*
4. *if w is the weight of a &-node, with eigen weight p, and w' is a weight depending on p and appearing in the proof structure then $w' \leq w$.*

A weight w *depends on* the eigen weight p if p or $\bar{p}$ occurs in w. A node L with weight w *depends on* the eigen weight p if w depends on p or L is a C-node and one of the weights just above it depends on p.

Remark 1. Observe that:

1. since weights associated to a PS are products (*monomials*) of the Boolean algebra generated by the eigen weights associated to a proof structure, then, for each weight w associated to a binary contraction node, there exists a unique eigen weight p that *splits* w into $w_1 = wp$ *and* $w_2 = w\overline{p}$. We sometimes index a C-link with its *toggling variable* p, see the next left hand side picture;
2. the graph π_1 (the next r.h.s. picture) is not a GPS since it violates condition 4 of Definition 3; indeed, if $w = q$ is the weight of the $\&_p$-link and $w' = \overline{p}$ is a weight depending on p and appearing in the proof-structure then $\overline{p} \not\leq q$.



2.1 Correctness

Definition 4 (slices, switchings, resolutions). *Let π be a CyMALL GPS.*

- *A* valuation *φ of π is a function from the set of all weights of π into $\{0, 1\}$.*
- *Fixed a valuation φ for π, the* slice *$\varphi(\pi)$ is the graph obtained from π by keeping only those nodes with weight* 1 *together with their incident edges.*
- *Fixed a slice $\varphi(\pi)$ a* multiplicative switching *S for π is the non-oriented graph $S^m_\varphi(\pi)$ built on the nodes and edges of $\varphi(\pi)$ with the modification that for each ∇-node we take only one premise (*left/right switch*).*
- *Fixed a slice $\varphi(\pi)$ an* additive switching*, denoted $S^a_\varphi(\pi)$ is a multiplicative switching $S^m_\varphi(\pi)$ for π, in which for each $\&_p$-node we erase the (unique) premise in $\varphi(\pi)$ and we add an oriented edge, called* jump*, from the $\&_p$-node to an link L whose weight depends on the eigen weight p.*
- *An* additive resolution *$\varphi_r(\pi)$ for a slice $\varphi(\pi)$ is the graph obtained by replacing in $\varphi(\pi)$ each unary link L (a link that, possibly, after the valuation has a single premise) by a single edge that is the (unique) premise of L. In particular, each conclusion of $\varphi_r(\pi)$ will be labeled by a multiplicative (CyMLL) formula.*

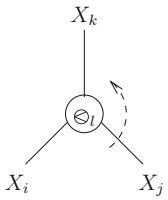
We call *additive resolution of a CyMALL sequent Γ* what remains of Γ after deleting one of the two sub-formulas in each additive (sub)formula of Γ.

In the following we characterize, by a *correctness criterion*, those CyMALL GPSs corresponding to proofs. This correctness criterion (Definition 7) is defined in terms of the correctness of CyMLL PSs (Definition 6). There exist several syntaxes for CyMLL proof nets; here we adopt the syntax of [2] inspired to [13].

Definition 5 (seaweeds). *Assume π is a CyMLL PS with conclusions Γ and assume $S(\pi)$ is an acyclic and connected multiplicative switching for π; $S(\pi)$ is*

the rootless planar tree whose nodes are labeled by $\Subset$-nodes, and whose leaves $X_1, ..., X_n$ (with $\Gamma \subseteq X_1, ..., X_n$) are the terminal (pending) edges of $S(\pi)$; $S(\pi)$ is a ternary relation, called a seaweed, *with* support $X_1, ..., X_n$*; an ordered triple (X_i, X_j, X_k) belongs to the seaweed $S(\pi)$ iff:*

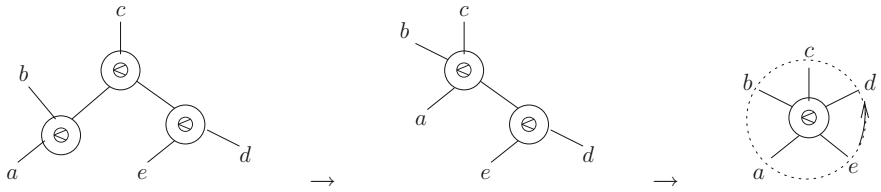
- *the intersection of the three paths X_iX_j, X_jX_k and X_kX_i is the node $\Subset_l$;*
- *the three paths $X_i\Subset_l, X_j\Subset_l$ and $X_k\Subset_l$ are in this cyclic order while moving anti-clockwise around the $\Subset_l$-node as below.*



If A is an edge of the seaweed $S(\pi)$, then $S_i(\pi) \downarrow^A$ is the restriction of the seaweed $S(\pi)$*, that is, the sub-graph of $S(\pi)$ obtained as follows:*

1. *disconnect the graph below (w.r.t. the orientation of π) the edge A.*
2. *delete the graph not containing A.*

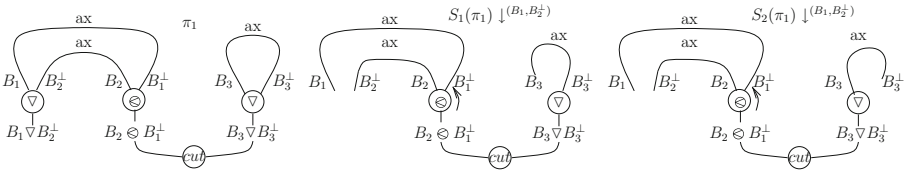
Fact 1 (seaweeds as cyclic orders). *Any seaweed $S(\pi)$ can be viewed as a cyclic total order (Definition 1) on its support $X_1, ..., X_n$; in other words, if a triple $(X_i, X_j, X_k) \in S(\pi)$, then $X_i < X_j < X_k$ are in cyclic order. Intuitively, we may contract a seaweed (by associating the $\Subset$-nodes) until it collapses into single n-ary $\Subset$-node with n pending edges (its support), like in the example below.*



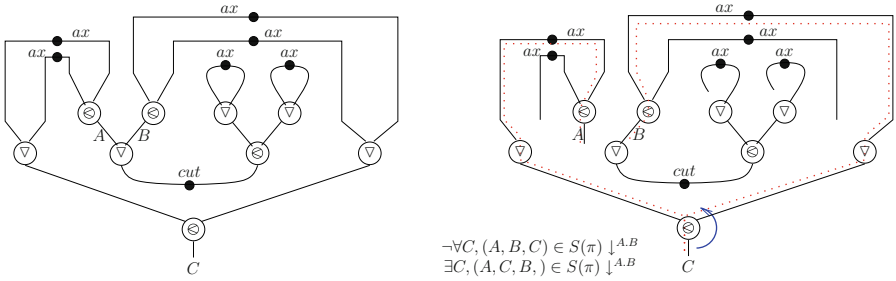
Definition 6 (CyMLL proof net). *A CyMLL PS π is* correct*, i.e. it is a CyMLL proof net (PN), iff:*

1. *π is a standard MLL PN, that is, any switching $S(\pi)$ is a connected and acyclic graph (therefore, $S(\pi)$ is a seaweed);*
2. *for any ∇-link $\frac{A \quad B}{A \nabla B}$ the triple (A, B, C) must occur in this cyclic order in any seaweed $S(\pi)$ restricted to A, B, i.e., $(A, B, C) \in S(\pi) \downarrow^{(A,B)}$, for all pending leaves C (if any) in the support of the restricted seaweed.*

Example 1 (CyMLL PSs). We give below an instance of CyMLL PN π_1 with its two restricted seaweeds, $S_1(\pi_1) \downarrow^{(B_1, B_2^\perp)}$ and $S_2(\pi_1) \downarrow^{(B_1, B_2^\perp)}$, both satisfying condition 2 of Definition 6.



Mellies's Counter-Example. Observe that, unlike what happens in the commutative MLL case, the presence of cut links is "quite tricky" in the non-commutative case, since cut links are not equivalent, from a topological point of view, to tensor links: these latter make appear new conclusions that may disrupt the original (i.e., in presence of cut links) order of conclusions. In particular, the *Mellies's proof structure*[1] below (see page 224 of [17]) is not correct according to our correctness criterion since there exists a $\frac{A \quad B}{A \nabla B}$ link and a switching $S(\pi)$ s.t. $\neg\forall C, (A, B, C) \in S(\pi) \downarrow^{(A,B)}$, contradicting condition 2 of Definition 6: following the crossing dotted lines in the next r.h.s. figure, you can easily verify $\exists C$ pending s.t. $(A, C, B) \in S(\pi) \downarrow^{(A,B)}$.

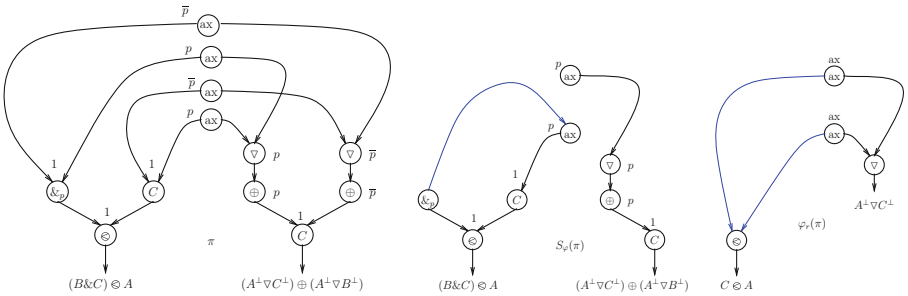


Definition 7 (CyMALL proof net). *We call* correct *(or* proof net, GPN*) any CyMALL GPS π s.t., its conclusions Γ are endowed with a cyclic order $\sigma(\Gamma)$ and for any valuation φ of π:*

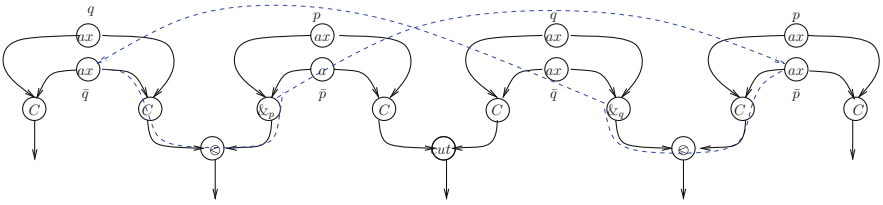
1. *each additive switching $S_\varphi(\pi)$ is an acyclic and connected graph (ACC);*
2. *there exists an additive resolution $\varphi_r(\pi)$ for $\varphi(\pi)$ that is a CyMLL PN with cyclic order conclusions $\sigma(\Gamma_r)$, where Γ_r is an additive resolution of Γ.*

Example 2 (CyMALL GPSs). Observe that the following proof structure π, on the left hand side, is not correct: actually, fixed a valuation φ s.t. $\varphi(p) = 1$, there exists an additive switching $S_\varphi(\pi)$ (with a jump) that is not ACC (see the center side figure). Nevertheless, any slice $\varphi(\pi)$ is ACC; for each slice $\varphi(\pi)$ there exists indeed an additive resolution $\varphi_r(\pi)$ that is a CyMLL PN like that one, on the rightmost hand side, with conclusions $C \otimes A, A^\perp \nabla C^\perp$. Observe, $\Gamma_r = (C \otimes A, A^\perp \nabla C^\perp)$ is an additive resolution of the conclusion of π, $\Gamma = (B \& C) \otimes A, (A^\perp \nabla C^\perp) \oplus (A^\perp \nabla B^\perp)$.

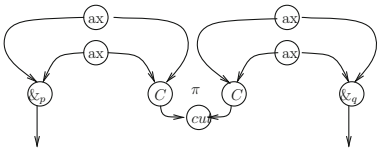
[1] This PS is considered as "a measure of the satisfiability degree" of correctness criteria of non-commutative logic: any "good" criterion should recognize this PS as uncorrect.



Similarly, the proof structure below is not correct: you can easily get an additive switching with a cycle like that one in (blue) dashed line.



Finally, the proof structure below is correct.

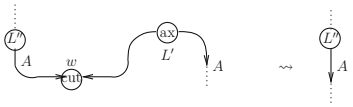


There currently exist other syntaxes for MALL PNs like the recent one by Hughes–van Glabbeek [10]. Unlike the Girard's one, this new syntax only works with "uniform proof structures", i.e., proof structures with only η-expanded axioms and with contraction links only immediately below the axiom links.

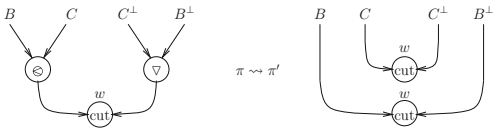
2.2 Cut Reduction

Definition 8 (ready cut reduction). *Let L be a cut link in a proof net π whose premises A and $A^{\perp}$ are conclusions of, resp., links L' and L'' with both of these different from contraction C. Then we define the result π' (called,* reductum*) of reducing a* ready cut *in π (called,* redex*), as follows:*

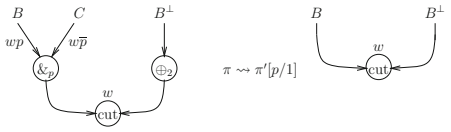
Ax*-cut: if L' (resp., L'') is an axiom link then π' is obtained by removing in π both formulas $A, A^{\perp}$ (as well as L) and giving to L'' (resp., to L') the other conclusion of L' (resp., L'') as new conclusion:*



($\parr/\otimes$)-cut: *if L' is a $\parr$-link with remises B and C and L'' is a $\otimes$-link with premises $C^\perp$ and $B^\perp$, then π' is obtained by removing in π both formulas A and $A^\perp$ as well as the cut link L together with L' and L'' and by adding two new cut links with, resp., premises B, $B^\perp$ and $C, C^\perp$, as follows:*



($\&/\oplus$)-cut: *if L' is a $\&_p$-link with premises B and C and L'' is a $\oplus_2$-link (resp., a $\oplus_1$-link) with premise $B^\perp$ (resp., $C^\perp$), then π' is obtained in three steps: first remove in π both formulas A, $A^\perp$ as well as the cut link L with L' and L'', then replace the eigen weight p by 1 (resp., p by 0) and keep only those links (vertexes and edges) that still have non-zero weight; finally we add a cut between B and $B^\perp$ (resp., between C and $C^\perp$) as below.*



Theorem 1 (stability of GPN under ready cut reduction). *Assume π is a GPN that reduces to π' in one step of ready cut reduction, then π' is a GPN.*

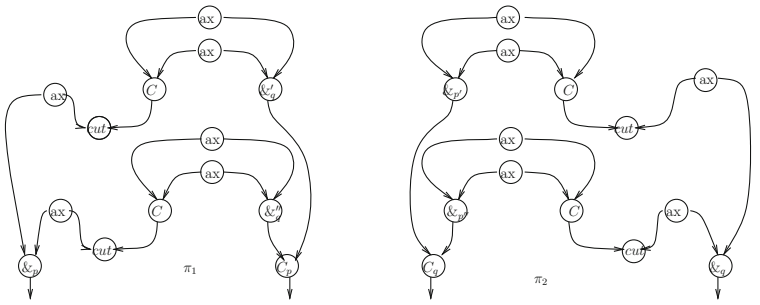
Proof. Stability of condition 1 of Definition 6 and condition 1 of Definition 7, under ready cut reduction, follows as a consequence of the next graph theoretical property (see pages 250–251 of [9]):

Property 1 (Euler-Poicaré invariance). Given a graph $\mathcal{G}$, then $(\sharp CC - \sharp Cy) = (\sharp V - \sharp E)$, where $\sharp CC$, $\sharp Cy$, $\sharp V$ and $\sharp E$ denotes the number of, respectively, connected components, cycles, vertexes and edges of $\mathcal{G}$.

Meanwhile, stability of condition 2 of Definition 6 (resp., condition 2 of Definition 7) follows simply by calculation.

The confluence problem - Reducing a cut involving a contraction link as (at least) one of its premises may lead to different reductum, depending on which sub-graph of the redex we decide to duplicate. For instance, as illustrated below, reducing the commutative cut of the last proof net of Example 2 leads either to π_1 or to π_2 (in the next picture), depending on which additive box, $\&_q$ or $\&_p$, we decide to duplicate. There is no a-priori way to make π_1 and π_2 "equal". Girard, in [8], did not give a solution for this problem which has been later provided by Laurent and Maieli in [12]. Here we present an original lazy commutative cut reduction that simplifies the latter: technically, our reduction relies on the notion of *dependency graph* (Definition 9), i.e. the smallest &-box needed

for duplication (see [14]). This cut reduction procedure preserves the notion of GPN (Theorems 1 and 2) and it is strong normalizing (Theorems 3 and 4).

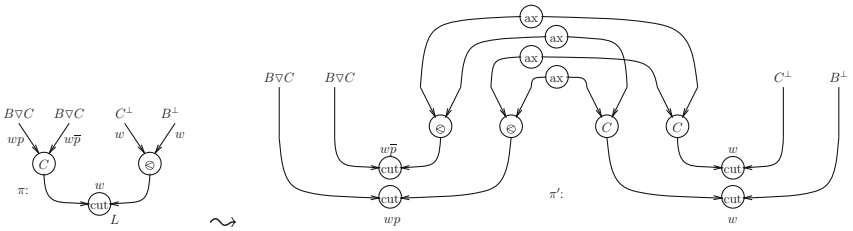


Definition 9 (empire and spreading). *Assume a proof structure π, an eigen weight p and a weight w, then:*

- *the* dependency graph *of p (w.r.t. π), denoted $\mathcal{E}_p$, is the (possibly disconnected) subgraph of π made by all links depending on p;*
- *the* spreading *of w over π, denoted by $w.[\pi]$, is the product of w for π, i.e., π where we replaced each weight v with the product of weights vw.*

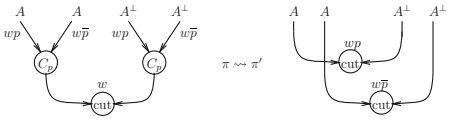
Definition 10 (commutative cut reduction). *Let L be cut link in a proof net π whose premises A and $A^\perp$ are the respective conclusions of links L' and L'' s. t. at least one of them is a contraction link C. Then we define the result π' (*reductum*) of reducing this* commutative *cut L in π (*redex*), as follows:*

*$(C/\parr)$***-cut:** *if L' is a C-link and L'' is a $\parr$-link, then π reduces in one $(C/\parr)$ step to π' (the (C/∇) step is analogous) as follows:*

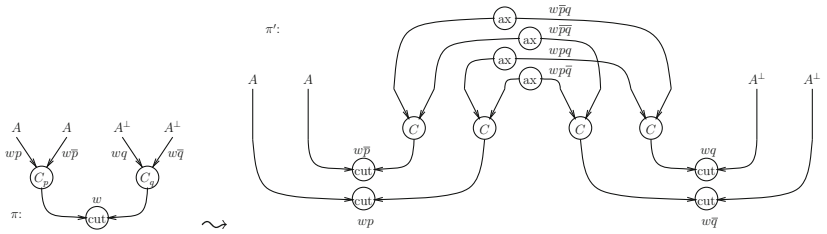


*(C/C)***-cut:** *if both L' and L'' are C-links, then there are two cases:*

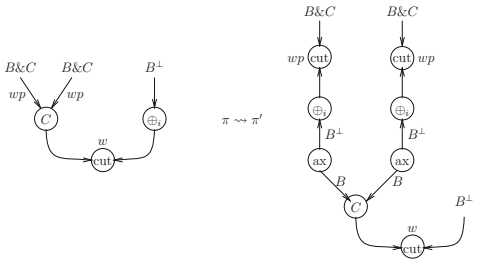
1. *either the weight w of both L' and L'' splits on the same p variable, then π reduces in one (C_p/C_p) step to π' as follows*



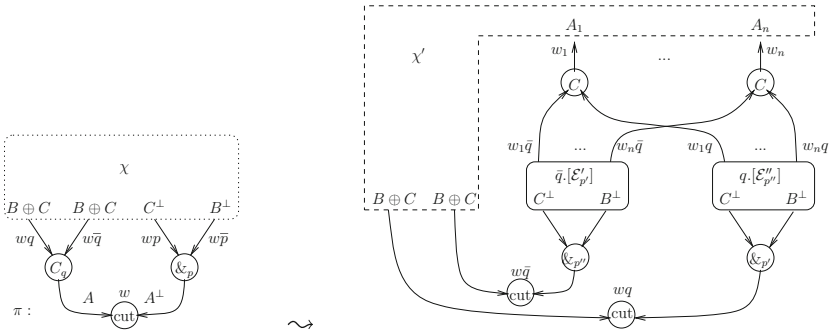
2. *or the weight w of L' (resp., w of L'') splits on p (resp., on q) then π reduces in one (C_p/C_q) step to π' as follows*



$(C/\oplus_i)$**-cut:** *if L' is a C-link and L'' a $\oplus_{i=1,2}$-link, then π reduces in one $(C/\oplus)$ step to π' as follows*



$(C/\&)$**-cut:** *if L' is a C-link and L'' a $\&_p$-link, then π reduces in one $(C/\&)$ step to π' as follows*



with the assumptions that graphs $\bar{q}.[\mathcal{E}'_{p'}]$ and $q.[\mathcal{E}''_{p''}]$ are obtained as follows:

1. *we take two copies, $\mathcal{E}'_p$ and $\mathcal{E}''_p$, of the dependency graph $\mathcal{E}_p$ of p;*
2. *we replace in $\mathcal{E}'_p$ (resp., in $\mathcal{E}''_p$) p with a new variable p' (resp., p'');*
3. *we spread $\bar{q}$ (resp., q) over $\mathcal{E}'_{p'}$ (resp., over $\mathcal{E}''_{p''}$).*

Technical details of proofs of Theorems 2, 3 and 4 can be found in [14].

Theorem 2 (stability). *If π is a GPN that reduces to π' in one step of commutative cut reduction then π' is a GPN too.*

Theorem 3 (termination). *We can always reduces a proof net π into a proof net π' that is cut-free, by iterating the reduction steps of Definitions 8 and 10.*

Theorem 4 (confluence). *Assume π is a proof net s.t. it reduces in one step α to π' ($\pi \rightsquigarrow_\alpha \pi'$) and it reduces in an other step β to π'' ($\pi \rightsquigarrow_\beta \pi''$); then, there exists a proof net σ such that both π' reduces, in a certain number of steps, to σ ($\pi' \rightsquigarrow^* \sigma$) and π'' reduces, in a certain number of steps, to σ ($\pi'' \rightsquigarrow^* \sigma$).*

2.3 Sequentialization

There exists a correspondence, called *sequentialization* (Theorem 5), between PNs and sequential proofs.

Lemma 1 (splitting). *Let π be a CyMLL PN with at least a $\otimes$-link or cut-link and with conclusions Γ not containing any terminal ∇-link (so, we say π is in* splitting condition*); then, there must exist a $\otimes$-link $\frac{A \quad B}{A \otimes B}$ (resp., a cut-link $\frac{A \quad A^{\perp}}{}$) that splits π in two CyMLL PNs, π_A and π_B (resp., π_A and $\pi_{A^{\perp}}$).*

Proof. Consequence of the Splitting Lemma for commutative MLL PNs [7].

Lemma 2 (PN cyclic order conclusions). *Let π be a CyMLL PN with conclusions Γ, then all seaweeds $S_i(\pi) \downarrow^{\Gamma}$, restricted to Γ, induce the same cyclic order σ on Γ, denoted $\sigma(\Gamma)$ and called* the (cyclic) order of the conclusions of π.

Proof. By induction on the size $\langle \sharp V, \sharp E \rangle$[2] of π.

Corollary 1 (stability of PN order conclusions under cut reduction). *If π, with conclusions $\sigma(\Gamma)$, reduces in one step of cut reduction to π', then also π' has conclusions $\sigma(\Gamma)$.*

Theorem 5 (sequentialization of CyMLL PNs). *Any CyMLL PN with conclusions $\sigma(\Gamma)$ can be* sequentialized *into a CyMLL sequent proof with the same cyclic order conclusions $\sigma(\Gamma)$.*

Proof. By induction on the size of the given proof net π via Lemmas 1 and 2.

Theorem 6 (sequentialization of CyMALL PNs). *A CyMALL GPN with conclusions $\sigma(\Gamma)$ can be* sequentialized *into a CyMALL sequent proof with the same cyclic conclusions $\sigma(\Gamma)$ and vice-versa (*de-sequentialization*).*

Proof. There are two parts.

Sequentialization-part. Any CyMALL proof net π can be sequentialized into a proof Π, by induction on the number of &-links. The base of induction corresponds to the sequentialization of the CyMLL proof nets (Theorem 5). The induction step follows by the sequentialization of standard MALL PNs (see [8]) where the only novelty is to show that: if a PN π contains a terminal &-link L, then π can be *toggled*[3] at L in two PNs preserving conditions 1 and 2 of Definition 7.

De-sequentialization-part. Any CyMALL proof Π of $\sigma(\Gamma)$ can be de-sequentialized into a PN π of $\sigma(\Gamma)$, by induction of the height of Π derivation.

[2] Number of vertexes and number of edges.

[3] We say that a terminal $\&_p$-link of a GPN π is *toggling* when the restriction of π w.r.t. p and the restriction of π w.r.t. $\bar{p}$ are both correct GPSs. We call the *restriction of π w.r.t. p* (resp., *w..r.t. $\bar{p}$*) what remains of π when we replace p with 1 (resp., $\bar{p}$ with 1) and keep only those vertexes and edges whose weights are still non-zero (see [8]).

Unlike most part of correctness criteria for non-commutative proof nets, like [3,13], our syntax enjoys a sequentialization for the full class of CyMLL PNs (with possible cuts). Observe that, *Mellies's counter-example* (Example 1) represents a non-sequentializable proof structure that becomes correct (therefore, sequentializable) after cut reduction.

3 Embedding Lambek Calculus into CyMALL PNs

Definition 11 (Lambek formulas and sequents of CyMALL). *Assume A and S are, respectively, a formula and a sequent of CyMALL.*

1. *A is a* pure Lambek formula (pLF) *if it is a CyMLL formula recursively built according to this grammar: $A := \text{positive atoms} \mid A \otimes A \mid A^{\perp} \triangledown A \mid A \triangledown A^{\perp}$.*
2. *A is an* additive Lambek formula (aLF or simply LF) *if it is a CyMALL formula recursively built according this grammar: $A := pLF \mid A \& A \mid A \oplus A$.*
3. *S is a* Lambek sequent of CyMALL *iff $S = (\Gamma^{\perp}, A)$, where A is a non-void LF and $\Gamma^{\perp}$ is a possibly empty finite sequence of negations of LFs (i.e., Γ is a possibly empty sequence of LFs and $\Gamma^{\perp}$ is obtained by taking the negation of each formula in Γ).*
4. *A* Lambek proof *is any derivation built by means of the CyMALL inference rules whose premise(s) and conclusions are CyMALL Lambek sequents.*

Definition 12 (Lambek proof net). *We call* Lambek CyMALL proof net *(resp.,* pure Lambek CyMLL proof net*) any CyMALL PN (resp., CyMLL PN) whose edges are labeled by LFs (resp. pLFs) or negation of LFs (resp., pLFs) and whose conclusions form a Lambek sequent.*

Corollary 2. *Any Lambek CyMALL proof net π is stable under cut reduction, i.e., if π reduces in one step to π', then π' is a Lambek CyMALL proof net too.*

Proof. Consequence of Theorems 1 and 2. Trivially, each reduction step preserves the property that each edge of the reductum is labeled by a Lambek formula or the negation of a Lambek formula.

Theorem 7 (de-sequentialization of Lambek CyMALL proofs). *Any proof of a CyMALL Lambek sequent $\vdash \sigma(\Gamma^{\perp}, A)$ can be de-sequentialized into a Lambek CyMALL PN with conclusions $\sigma(\Gamma^{\perp}, A)$.*

Proof. By induction on the height of the given sequent proof (similarly to the de-sequentialization part of Theorem 6).

Theorem 8 (sequentialization of Lambek CyMALL PNs). *Any Lambek CyMALL PN of $\sigma(\Gamma^{\perp}, A)$ can be sequentialized into a Lambek CyMALL proof of the sequent $\vdash \sigma(\Gamma^{\perp}, A)$.*

Proof. Sequentialization follows by induction on the number &-links of the given PN. The base of induction is given by next Theorem 9. The induction step, simply follows by Theorem 6.

Theorem 9 (sequentialization of pure Lambek CyMLL PNs). *Any Lambek CyMLL PN of* $\sigma(\Gamma^{\perp}, A)$ *can be sequentialized into a Lambek CyMLL proof of* $\vdash \sigma(\Gamma^{\perp}, A)$.

Proof. See details in [2].

4 Language Parsing with Lambek CyMALL PNs

In order to show how powerful PNs are, in this section we adapt to our syntax, some linguistics (typing) examples suggested by Richard Moot in his PhD thesis [18]. We use s, np and n as the types expressing, respectively, a *sentence*, a *noun phrase* and a *common noun*. According to the "parsing as deduction style", when a string $w_1...w_n$ is tested for grammaticality, the types $t_1, ..., t_n$ associated with the words are retrieved from the lexicon and then parsing reduces to proving the derivability of a two-sided sequent of the form $t_1, ..., t_n \vdash s$. Recall that proving a two sided Lambek derivation $t_1, ..., t_n \vdash s$ is equivalent to prove the one-sided sequent $\vdash t_n^{\perp}, ...t_1^{\perp}, s$ where $t_i^{\perp}$ is the dual (i.e., the linear negation) of each type t_i. Therefore, any phrase or sentence should be written like in a mirror (observing the opposite natural direction).

Assume the following lexicon, where *linear implication* $\multimap$ (resp., $\circ\!\!-$) is traditionally used for expressing types in two-sided sequent parsing style:

1. Vito $= np$;
2. Sollozzo $= np$;
3. him $= (s \circ\!\!- np) \multimap s \;=\; (s \nabla np^{\perp})^{\perp} \nabla s \;=\; (np \parr s^{\perp}) \nabla s$;
4. trusts $= (np \multimap s) \circ\!\!- np \;=\; (np^{\perp} \nabla s) \nabla np^{\perp}$.

Cases of lexical ambiguity follow to words with several possible formulas A and B assigned it. For example, a verb like "*to believe*" can express a relation between two persons, np's in our interpretation, or between a person and a statement, interpreted as s, like in these examples:

(1) *Sollozzo believes Vito*; (2) *Sollozzo believes Vito trusts him.*

We can express this polymorphism by two lexical assignments as follows:

5. believes $= (np \multimap s) \circ\!\!- np \;=\; (np^{\perp} \nabla s) \nabla np^{\perp}$;
6. believes $= (np \multimap s) \circ\!\!- s \;=\; (np^{\perp} \nabla s) \nabla s^{\perp}$.

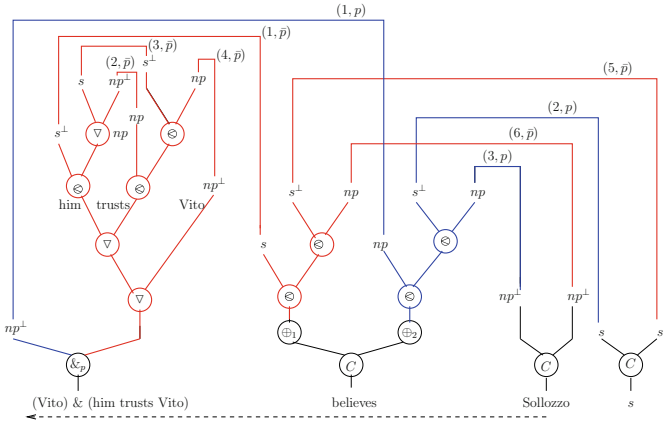
Typically, additives are used to capture cases of lexical ambiguity. When a word has two possible formulas A and B assigned it, we can combine these into a single additive formula $A \& B$ (resp., $A \oplus B$). Thus, we can collapse assignments 5 and 6 into the following single additive assignment:

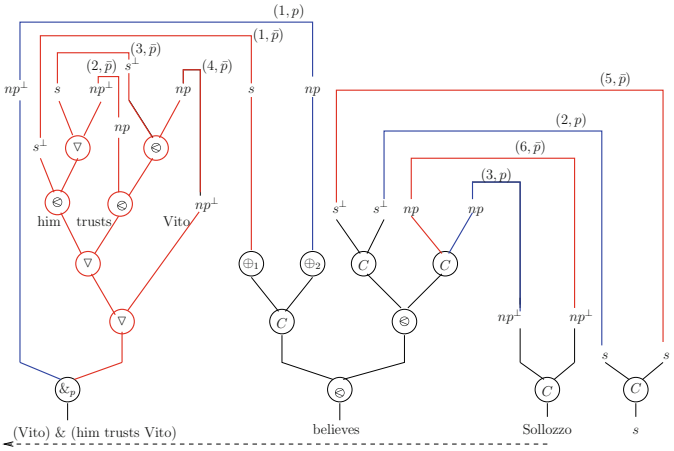
7. believes $= ((np \multimap s) \circ\!\!- np) \& ((np \multimap s) \circ\!\!- s) \;=\; ((np^{\perp} \nabla s) \nabla np^{\perp}) \& ((np^{\perp} \nabla s) \nabla s^{\perp})$.

Equivalently, via distributivity of negative connectives, we could also move the additive "inside" and generate a more compact lexical entry, in which the two assignments share their identical initial parts (see also [19] on type polymorphism):

8. believes $= ((np \multimap s) \circ\!\!- (s \oplus nps)) = ((np^\perp \triangledown s) \triangledown (np^\perp \& s^\perp))$.

Using that, we can then move lexical ambiguity into proof nets. In the following we give two equivalent Lambek PNs as parsing of the additive superposition of sentences (1) and (2); the first (resp., the second) PN makes use of the lexical entry 7 (resp., the lexical entry 8).





Observe that, for each slice of each proof net above, $\varphi_p(\pi)$ and $\varphi_{\bar{p}}(\pi)$, there exists an additive resolution that is a CyMLL PN with the same sequentialization:

$$\varphi_p(\pi) \;\Rightarrow\; \Pi_1 : \quad \dfrac{\dfrac{}{np^\perp, np}\, ax_1^p \qquad \dfrac{\dfrac{}{s^\perp, s}\, ax_2^p \qquad \dfrac{}{np, np^\perp}\, ax_3^p}{s^\perp \otimes np, np^\perp, s}\,\otimes}{np^\perp, np \otimes (s^\perp \otimes np), np^\perp, s}\,\otimes$$

$$\varphi_{\bar{p}}(\pi) \;\Rightarrow\; \Pi_2 : \quad \dfrac{\dfrac{\dfrac{\dfrac{}{s^\perp, s}\, ax_1^{\bar{p}} \qquad \dfrac{\dfrac{s, np^\perp, np \otimes (s^\perp \otimes np), np^\perp}{s \nabla np^\perp, np \otimes (s^\perp \otimes np), np^\perp}\,\nabla}{s^\perp \otimes (s \nabla np^\perp), np \otimes (s^\perp \otimes np), np^\perp, s}\,\otimes}{\dfrac{s^\perp \otimes (s \nabla np^\perp), np \otimes (s^\perp \otimes np), np^\perp, s \otimes (s^\perp \otimes np), np^\perp, s}{((s^\perp \otimes (s \nabla np^\perp)) \nabla (np \otimes (s^\perp \otimes np))) \nabla np^\perp, s \otimes (s^\perp \otimes np), np^\perp, s}\,\nabla^2} \qquad \dfrac{\dfrac{}{s^\perp, s}\, ax_5^{\bar{p}} \qquad \dfrac{}{np, np^\perp}\, ax_6^{\bar{p}}}{s^\perp \otimes np, np^\perp, s}\,\otimes}{}\,\otimes \qquad \dfrac{\dfrac{}{np^\perp, np}\, ax_2^{\bar{p}} \qquad \dfrac{\dfrac{}{s^\perp, s}\, ax_3^{\bar{p}} \qquad \dfrac{}{np, np^\perp}\, ax_4^{\bar{p}}}{s^\perp \otimes np, np^\perp, s}\,\otimes}{}\,\otimes$$

5 Conclusions

As future work we aim at investigating the how topological correctness criteria based on graph rewriting (or retraction) of MALL proof structures [6,15] may be used for (linguistic) parsing. Naively, retraction criteria allow to switch from the paradigm of "parsing as deduction" to the paradigm of "parsing as rewriting" (see, e.g., [16]). Moreover, retraction could also be a useful computational tool for studying the complexity class of the CyMALL correctness criterion.

Acknowledgements. We thank the anonymous reviewers, Michael Moortgat and Richard Moot for their useful comments and suggestions. This work was partially supported by the PRIN Project *Logical Methods of Information Management*.

References

1. Abrusci, V.M.: Classical conservative extensions of Lambek calculus. Stud. Logica. **71**(3), 277–314 (2002)
2. Abrusci, V.M., Maieli, R.: Cyclic multiplicative proof nets of linear logic with an application to language parsing. In: de Paiva, V., de Queiroz, R., Moss, L.S., Leivant, D., de Oliveira, A. (eds.) WoLLIC 2015. LNCS, vol. 9160, pp. 53–68. Springer, Heidelberg (2015)
3. Abrusci, V.M., Ruet, P.: Non-commutative logic I: the multiplicative fragment. Ann. Pure Appl. Logic **101**(1), 29–64 (2000)
4. Andreoli, J.-M., Pareschi, R.: From Lambek calculus to word-based parsing. In: Proceedings of Substructural Logic and Categorial Grammar Workshop, Munchen (1991)
5. Danos, V., Regnier, L.: The structure of multiplicatives. Arch. Math. Logic **28**, 181–203 (1989)
6. Danos, V.: La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalment du λ-calcul). Ph.D. thesis, Paris (1990)
7. Girard, J.-Y.: Linear logic. Theoret. Comput. Sci. **50**, 1–102 (1987)
8. Girard, J.-Y.: Proof-nets: the parallel syntax for proof theory. In: Logic and Algebra. Marcel Dekker (1996)
9. Girard, J.-Y.: Le point aveugle. Cours de Logique, vol. I, Vers la Perfection. Ed. Hermann, Paris (2006)
10. Hughes, D., van Glabbeek, R.: Proof nets for unit-free multiplicative-additive linear logic. In: Proceedings of IEEE LICS (2003)
11. Lambek, J.: The mathematics of sentence structure. Amer. Math. Monthly **65**, 154–170 (1958)

12. Laurent, L., Maieli, R.: Cut elimination for monomial MALL proof nets. In: Proceedings of IEEE LICS, Pittsburgh, USA, pp 486–497 (2008)
13. Maieli, R.: A new correctness criterion for multiplicative non-commutative proof-nets. Arch. Math. Logic **42**, 205–220 (2003). Springer-Verlag
14. Maieli, R.: Cut elimination for monomial proof nets of the purely multiplicative and additive fragment of linear logic. IAC-CNR Report, no. 140 (2/2008). HAL Id: hal-01153910. https://hal.archives-ouvertes.fr/hal-01153910
15. Maieli, R.: Retractile proof nets of the purely multiplicative and additive fragment of linear logic. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 363–377. Springer, Heidelberg (2007)
16. Maieli, R.: Construction of retractile proof structures. In: Dowek, G. (ed.) RTA-TLCA 2014. LNCS, vol. 8560, pp. 319–333. Springer, Heidelberg (2014)
17. Moot, R., Retoré, C.: A logic for categorial grammars: Lambek's syntactic calculus. In: Moot, R., Retoré, C. (eds.) The Logic of Categorial Grammars. LNCS, vol. 6850, pp. 23–63. Springer, Heidelberg (2012)
18. Moot, R.: Proof nets for linguistic analysis. Ph.D. thesis. Utrecht University (2002)
19. Morrill, G.: Additive operators for polymorphism. In: Categorial Grammar: Logical Syntax, Semantics and Processing. Oxford University Press (2011)

Algebraic Governance and Symmetry in Dependency Grammars

Carles Cardó(✉)

Universitat Politècnica de Catalunya, Barcelona, Spain
cardocarles@gmail.com

Abstract. We propose a purely algebraic approach to governance structure in dependency grammars aiming to capture all linguistic dependencies (such as morphological, lexico-semantic, etc.) in monoidal patterns. This provides a clear perspective and allows us to define grammars declaratively through classical projective structures. Using algebraic concepts the model is going to suggest some symmetries among languages.

Keywords: Algebraic governance · Dependency grammars · Dependency structure · Governance · Symmetry · Projective structure

1 Preliminaries: Dependency Structures

Our model is based on an algebraic characterization of dependency trees (Sect. 2) and on an intuitive working hypothesis (Sect. 3). It consists in two objects: a *manifold* and a *linearization*. For reasons of space we can only address in this paper the construction of manifolds (Sect. 4). We show linearizations for specific cases but we do not develop the general mechanism. Fortunately manifolds and linearization are independent modules in our formalism. In Sect. 5 we implement some classical formal languages which encode certain cross-serial phenomena. In Sect. 6 we sketch how to build a manifold for a natural language. Using basic algebraic concepts, the model is going to suggest some symmetries of languages (Sect. 7) which will help us to understand our analysis. In the last section (Sect. 8) we show another equivalent interpretation of the presented results.

To connect our approach with the literature we now define some basic concepts in dependency grammars.

A *dependency structure* is a triplet $(D, \trianglelefteq, \leq)$, where $\trianglelefteq$ and $\leq$ are partial orders. The first relationship forms a tree (i.e. the Hasse diagram is a tree), the second relation forms a chain (i.e. the relationship is a total order and its Hasse diagram is a chain). The tree relation $\trianglelefteq$ is called *governance*; the total order $\leq$ is called *precedence*. To visualize a dependency structure we could draw both Hasse diagrams over the same set D, but it is more convenient to picture the Hasse diagram of the structures $(D, \trianglelefteq)$, $(D, \leq)$ separately and connect them with dashed lines, as in Figs. 1(a) and (b). These lines are usually called *projection lines*.

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 60–76, 2016.
DOI: 10.1007/978-3-662-53042-9_4

The set D can be understood as a set of words, but since we frequently find sentences with repeated words we remember that we are really dealing with word tokens. A dependency structure with words is a structure $(D, \trianglelefteq, \leq, f)$ such that $(D, \trianglelefteq, \leq)$ is a dependency structure and f is a mapping $f : D \longrightarrow \Sigma$, where Σ is a finite vocabulary.[1]

A dependency grammar is a finite description of a set $\mathbf{G}$ of dependency structures with words. The language of a dependency grammar is the same set of dependency structures without the governance relation, i.e. only the chains. If we want the language in Σ^* we can define the set:

$$\{f(x_1)\cdots f(x_n) \in \Sigma^* \,|\, \{x_1, \ldots, x_n\} = D,\ x_1 \leq \cdots \leq x_n \text{ and } (D, \trianglelefteq, \leq, f) \in \mathbf{G}\}.$$

There are several frameworks providing such description, for example *Lexicalized Grammars*, *Tree Adjoining Grammar*, *Regular Dependency Grammars* or *eXtensible Dependency Grammar*. See [3,4].

The literature on dependency grammars focuses on explaining the relationship between governance and precedence. Since from the point of view of natural languages there are a lot of inadequate dependency structures some constraints must be imposed. The main one is called *projectivity*. A dependency structure is called projective iff each subtree of the governance structure is an interval on the precedence structure, as in Fig. 1(a) and (c). Graphically classical projective structures are planar graphs (trees of governance) that can be embedded in the plane and be geometrically projected on a line (chain of precedence) without crossing lines of projection, see Fig. 1(a). Figure 1(b) shows a non-projective structure. This is a geometrical characterization of projectivity, although there are others.

It is thought that projective structures cover over 75–80 % of sentences of natural language [4]. To explain the rest there are two options. Either we must assume a less intuitive analysis of the sentences, or we must loosen the sense of projectivity. For this reason other projections have been proposed, such as *weak non-projective structures* or *well-nested structures* [6].

2 Syntagmata and Manifolds

We make a few comments on notation. Given a finite set, A, we denote A^* the *free monoid generated by* A. The *length* of an $x \in A^*$ is denoted by $|x|$. We are going to use free monoids ζ^* and Σ^*. 1 is the identity of ζ^* and 0, the identity of Σ^*. We will use the product notation for both. Given two monoids Γ, Γ', the Cartesian product $\Gamma \times \Gamma'$ is again a monoid, called the direct sum, with its operation defined componentwise. We abbreviate $\mathbb{N}_+ = \mathbb{N} \cup \{0\}$ and $\Sigma_+ = \Sigma \cup \{0\}$. We omit the brackets and write $\Sigma_+^n = (\Sigma_+)^n = \Sigma_+ \times \cdots \times \Sigma_+$ n times. Equally $\zeta^{*n} = (\zeta^*)^n$. Given a set $A \subseteq \zeta^{*n}$, we denote by A^* the minimal submonoid in ζ^{*n} containing A. Thus the star operator is a generalization of

[1] There are other definitions of dependency structures, such as those encoding structure as a graph or through *Robinson's axioms*, [2], but the above is more concise.

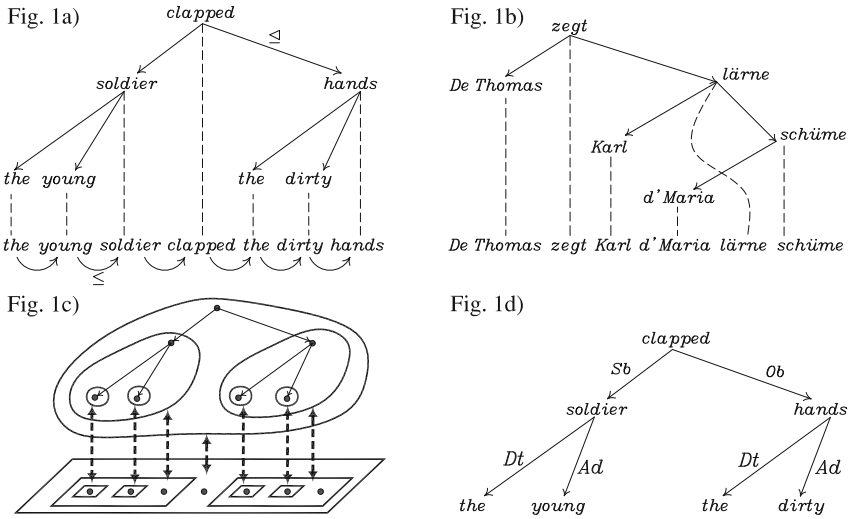


Fig. 1. (a) dependency structure. (b) non-projective dependency structure. (c) projective structure relating subtrees to intervals. (d) graphical representation of a syntagma.

the Kleene star for languages to monoids of several components. Algebraically $G^* = \langle G \rangle$ is the submonoid generated by G. We say that a submonoid G^* is *finitely generated* iff G is a finite set.

Definition 1. *Let ζ be a finite set of syntactic functions, and Σ a finite set of words. We call a* syntagma *a mapping $S : \zeta^* \longrightarrow \Sigma_+$ such that*

(i) S has finite order, i.e. $|\{x \in \zeta^* \,|\, S(x) \neq 0\}| < \infty$.
(ii) S is non-elliptic, i.e. $S(x) = 0 \Longrightarrow S(yx) = 0, \ \forall x, y \in \zeta^*$.[2]

We call the elements of ζ^* loci. *We call the* support *of S the non-null loci, $Spt(S) = \{x \in \zeta^* \,|\, S(x) \neq 0\}$. We call the* order *the cardinality of the support, $|S| = |\{x \,|\, S(x) \neq 0\}|$. A locus x is a* leaf *in S iff $S(x) \neq 0$, but $S(yx) = 0$ for all $y \neq 1, y \in \zeta^*$. We call the* depth *of S the number $d(S) = \max\{|x| \mid x \in \zeta^*$, such that x is a leaf in $S\}$.*

Some of the syntactic functions that we are going to use are: *Sb*, Subject; *Ob*, Object; *In*, indirect object; *At*, Attribute (the argument of copulative verbs); *Aj*, Adjunct to indicate mode, place or time; *Dt*, Determiner; *Ad*, Adjective; *Nc*, Noun Complement (similar to *Ad*, but it introduces a new nominal syntagma which is announced by a preposition); *Ip*, Introduction of a Preposition. Nevertheless the names are not important: syntactic functions play roles only in relation to each other.

[2] An ellipsis occurs when a locus is null but there is underneath a non-null locus; otherwise the locus is definitively null. A class of *elliptic* syntagmata is also useful, but in order to simplify things we are not using them except in the last example.

The locus $Dt \cdot Sb$, say, can be read as *the determiner of the subject*. The set of loci ζ^* can be interpreted as a set of addresses. Words can be repeated in a sentence but not loci, so invoking a word can be made unequivocally through the loci. The main benefit of syntagmata is in translating linguistic descriptions into algebraic descriptions.

Example 1. Let the syntactic functions be $\zeta = \{Sb, Ob, Dt, Ad\}$ and let the vocabulary be $\Sigma = \{\text{the, soldier, hands, clapped, dirty}\}$. Let S be the mapping $S : \zeta^* \longrightarrow \Sigma_+$ given by: $S(1) = \text{clapped}$, $S(Sb) = \text{soldier}$, $S(Ob) = \text{hands}$, $S(Dt \cdot Sb) = \text{the}$, $S(Ad \cdot Sb) = \text{young}$, $S(Dt \cdot Ob) = \text{the}$, $S(Ad \cdot Ob) = \text{dirty}$, and $S(x) = 0$, otherwise. Then S is a syntagma with order $|S| = 7$ and depth $d(S) = 2$. The leaves are the loci $Dt \cdot Sb$, $Ad \cdot Sb$, $Dt \cdot Ob$ and $Ad \cdot Ob$.

The next proposition connects syntagmata with governance structures.

Proposition 1. *Given a syntagma S we call* algebraic governance *the relationship $(Spt(S), \trianglelefteq)$ such that $x \trianglelefteq y \iff \exists \varphi \in \zeta^*$ such that $x = \varphi \cdot y$. Algebraic governance is a governance relationship. So $(Spt(S), \trianglelefteq, S)$ is governance with words.*

Proof. On the one hand we see that $\trianglelefteq$ is a partial order as follows. Reflexivity: we have $x \trianglelefteq x, \forall x \in Spt(S)$, because $x = 1 \cdot x$. Antisymmetry: let x, y be in $Spt(S)$ such that $x \trianglelefteq y$ and $y \trianglelefteq x$, iff $x = x' \cdot y$ and $y = y' \cdot x$ for some $x', y' \in \zeta^*$. So $x = x' \cdot (y' \cdot x) = (x' \cdot y') \cdot x$. But free monoids enjoy the cancellation laws, therefore $1 = x' \cdot y'$, and then $x' = y' = 1$. So $x = y$. Transitivity: let x, y, z be in $Spt(S)$ such that $x \trianglelefteq y$ and $y \trianglelefteq z$, iff $x = x' \cdot y$ and $y = y' \cdot z$, then we compose both: $x = x' \cdot (y' \cdot z) \iff x = (x' \cdot y') \cdot z \iff x \trianglelefteq z$.

On the other hand we must prove that its Hasse diagram is a tree. This can be done by induction on the depth of adding new leaves to the syntagma. Finally we see that the mapping $S : \zeta^* \longrightarrow \Sigma_+$ can be restricted to the mapping $S : Spt(S) \longrightarrow \Sigma$, because the elements in $Spt(S)$ cannot be null, so we have the governance structure with words: $(Spt(S), \trianglelefteq, S)$. □

To represent a syntagma graphically we can use the Cayley digraph of a binary operation to represent the support, and then substitute the nodes according to the mapping S. The Cayley digraph of a free monoid ζ^* is the directed digraph (V, E) with vertexes $V = Spt(S)$ and edges $E = \{(x, \lambda x) \,|\, x \in \zeta^*, \lambda \in \zeta\}$; since the monoid is free the digraph is always tree-shaped. Then we label every edge $(x, \lambda x)$ with the function λ. Finally we just have to label the vertexes according the map S. See Fig. 1(d). Note that the Hasse diagram of the algebraic governance coincides with the Cayley digraph without labels of edges, so a locus is the labeling of a path from root to node.

Some frameworks in dependency grammars require another kind of underlying non-tree-shaped structure, such as directed acyclic graphs or trees with repeated syntactic functions on a same level like XDG [3], or Topological Grammar, [5].

To accommodate this there is a natural generalization of our syntagmata based on automata.[3]

We notate $\mathbf{Synt}_{\zeta,\Sigma}$ the set of all syntagmata with ζ and Σ fixed. A manifold is a subset $W \subseteq \mathbf{Synt}_{\zeta,\Sigma}$. Although we do not go into details of linearization here, if we have a manifold and a procedure to assign a chain to each syntagma, the *linearization*, then we will have a set of dependency structures with words, i.e. a grammar and consequently a language. Thus our grammars would consist of a pair $\mathbf{G} = (W, \Pi)$, where W is a manifold and $\Pi \subseteq W \times \Sigma^*$ is a relationship which relates syntagmata to strings. The language of the grammar will be $L(\mathbf{G}) = \Pi(W) = \{x \in \Sigma^* \,|\, (S, x) \in \Pi, S \in W\}$. Now there are two crucial questions. First, how can the manifolds be established? Second, how can Π be established? This paper tries to answer the first question. Regarding the second question we are going to define the chains through the figures; but this should not create problems. In other words, we do not show a general mechanism to establish Π, but we will do it for each specific case.

3 Grouping Agreement: Linguistic Basis

In order to understand the central proposal we are going to present some linguistic cases and extract a heuristic conclusion.

3.1 Verb Inflection in English

We are interested in grouping the several agreement instances of the words in a sentence. The most visible match is when two words are morphologically matched. Some languages are more profuse in this respect, for example Romance languages, however even English manifests some examples. In English when the subject of a sentence is the third person singular and the main verb is present tense and not modal, the verb goes with a -s at the end. For example:

(1) John$_{Sb}$ often eats$_1$ meat.

We have underlined the morphologically matched words and subscripted the loci. The analysis is given in Fig. 2(a); the agreeing words are linked by a curve dashed line in this and the following examples.[4] Since the main verb is always allocated the locus 1 and the subject is in Sb we can represent this match as an ordered pair $(1, Sb)$. At the moment we are interested in the loci involved, not in the rule or condition. Now we can take the new sentence:

(2) Mary says John$_{Sb \cdot Ob}$ often eats$_{Ob}$ meat.

[3] For the purposes of this article it may be assumed that there are no repeated syntactic functions at the same level. In work in progress we relax this condition but there is not space to present the details here, and they are not relevant to the contribution of the present article.

[4] Agreements (dashed lines) are not dependencies (arrows), but loci can be used to describe them.

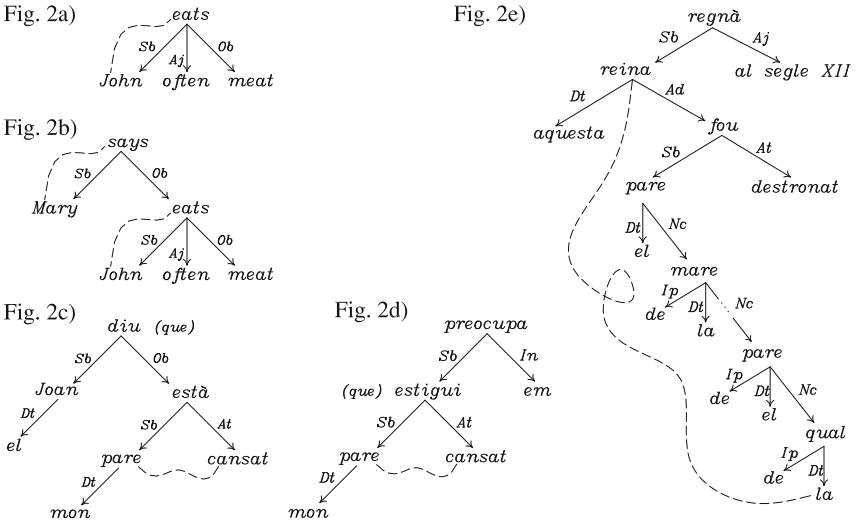


Fig. 2. (a) and (b) agreement of subject and verb in English. (c) and (d) agreement of subject and attribute in Catalan. (e) agreement in pied-piping in Catalan.

There are two matched pairs: a new pair $(1, Sb)$ and the old which is now in a deeper position $(Ob, Sb \cdot Ob)$. The sentence has an analysis as in Fig. 2(b). The agreement phenomenon iterates in deeper sentences: *Paul says Mary says John often eats meat*, *Peter says Paul says Mary says John often eats meat*, ... In general we can describe the set of the loci of all these agreements as: $Agr = \{(Ob^n, Sb \cdot Ob^n) \,|\, n \in \mathbb{N}_+\}$.

3.2 Match of Subject and Attribute

We can consider other languages and other types of matching. In some Romance languages the number and gender of the attribute of the copula must be equal to the subject.

(3) Mon pare$_{Sb}$ està cansat$_{At}$. / Les meves germanes$_{Sb}$ estan cansades$_{At}$.
My father is exhausted. / the my sisters are exhausted.
'My father is exhausted. / My sisters are exhausted.'

These examples are in Catalan, however the same occurs in Spanish, French or Italian. The matched pair is (Sb, At). We also consider other examples in a deeper position:

(4) a. El Joan diu que mon pare$_{Sb \cdot Ob}$ està cansat$_{At \cdot Ob}$.
The John says that my father is exhausted.
'John says my father is exhausted'.

b. Que mon $\underline{\text{pare}}_{Sb\cdot Sb}$ estigui $\underline{\text{cansat}}_{At\cdot Sb}$ em preocupa.
that my $\underline{\text{father}}$ is $\underline{\text{exhausted}}$ me worries.
'The fact of my father being exhausted is worrying'.

The analyses are given in Figs. 2(c) and (d). The first agreement is $(Sb \cdot Ob, At \cdot Ob)$, but the second is $(Sb \cdot Sb, At \cdot Ob)$. This can be interpreted as that the set of agreements is $Agr = \{(Sb \cdot x, At \cdot x) \,|\, x \in \zeta^*\}$, in other words, this match would hold everywhere.

3.3 Pied-Piping in Some Romance Languages

We consider another phenomenon called *pied-piping* consisting in the embedding of a filler such as a relative pronoun within accompanying material from the extraction site. Recall that we are only interested in the morphological matches, not in the semantical operation of the anaphora. In Catalan we must say:

(5) Aquesta $\underline{\text{reina}}_{Sb}$, el pare del pare ... del pare de
This $\underline{\text{queen}}$ the father of-the father ... of-the father of
$\underline{\text{la}}_{Dt\cdot Nc\cdot Nc^n\cdot Sb\cdot Ad\cdot Sb}$ qual fou destronat, regnà al segle XII.
$\underline{\text{the}}$ who was dethroned, reigned in century XII
'This queen the father of the father ... of the father of whom was dethroned reigned in XII century'.

The underlined words must agree in number and gender; see Fig. 2(e). In English there is not any match and we cannot underline any word because the pronoun is not preceded by any article. The proof of this agreement is that in Catalan we cannot say: **Aquesta reina, el pare del pare ... del pare de el qual fou destronat, regnà al segle XII.* Now the agreement set is $Agr = \{(Sb, Dt \cdot Nc \cdot Nc^n \cdot Sb \cdot Ad \cdot Sb) \,|\, n \in \mathbb{N}_+\}$. This is structurally a distinct type of construction because the monoid is growing right in the middle of certain loci. But this phenomenon can occur anywhere, for example in the object:

(6) El poble no acceptà mai aquesta $\underline{\text{reina}}_{Ob}$ el pare del
The populace not accepted never this $\underline{\text{queen}}$ the father of-the
pare ... del pare de $\underline{\text{la}}_{Dt\cdot Nc\cdot Nc^n\cdot Sb\cdot Ad\cdot Ob}$ qual fou destronat.
father ... of-the father of $\underline{\text{the}}$ who was dethroned.
'The populace never accepted this queen the father of the father ... of the father of whom was dethroned'.

So the agreements are more general:

$$Agr = \{(x, Dt \cdot Nc \cdot Nc^n \cdot Sb \cdot Ad \cdot x) \,|\, n \in \mathbb{N}_+, x \in \zeta^*\}.$$

3.4 A Working Hypothesis

We can summarize all these agreements using subsets of the monoid $\zeta^* \times \zeta^*$. In the first case we have: $Agr = \{(Ob^n, Sb \cdot Ob^n) \,|\, n \in \mathbb{N}_+\} = \varphi \cdot \Gamma$, where

$\varphi = (1, Sb)$, $\Gamma = (Ob, Ob)^*$. In the second case we also have: $Agr = \{(Sb \cdot x, At \cdot x) \,|\, x \in \zeta^*\} = \varphi \cdot \Gamma$, where $\varphi = (Sb, At)$, $\Gamma = \{(x, x) \,|\, x \in \zeta^*\}$. Finally in the third case: $Agr = \{(x, Dt \cdot Nc \cdot Nc^n \cdot Sb \cdot Ad \cdot x) \,|\, n \in \mathbb{N}_+, x \in \zeta^*\} = \varphi \cdot \Gamma \cdot \psi \cdot \Gamma'$, where $\varphi = (1, Dt \cdot Nc)$, $\Gamma = (1, Nc)^*$, $\psi = (1, Sb \cdot Ad)$ and $\Gamma' = \{(x, x) \,|\, x \in \zeta^*\}$. From the examples seen, and many others that could be given, we can extract a hypothesis. Let φ_i be some elements in ζ^* and let Γ_i be some submonoids of the monoid ζ^{*n}:

MONOIDAL HYPOTHESIS. *For any natural language the set of the places where agreements occur can be described as:*

$$Agr = \prod_{i=1}^{k} \varphi_i \cdot \Gamma_i \subseteq \zeta^{*n}.$$

Note that some submonoids Γ_i and constants φ_i can be trivial, thus we have a lot of patterns like: $\varphi\Gamma$, $\Gamma\varphi$, $\Gamma\Gamma'\varphi$, $\varphi\Gamma\psi\Gamma'$, ...

Even though the agreements shown were binary the monoidal hypothesis supports several arities, but the most common are 1, 2 and 3. For example: that *the subject Sb is always a noun* is itself an agreement of arity 1 which must hold generally in all loci of a syntagma, therefore the pattern must be $Sb \cdot \zeta^*$.

We have been talking about morphological agreement but there is no reason not to talk about other kinds. When a verb like *to drink* selects a "drinkable" object it is producing a semantic "agreement" $(1, Ob)$, and so forth.

4 Rules, Patterns and Syntactic Manifolds

4.1 Valuations, Rules and Satisfiability

We are going to formalize the concept of *pattern*, which goes preceded by the concept of *rule*. A rule is a linking condition while a pattern tells us where the rule must be respected.

A *valuation* of arity n is a mapping $B : \Sigma_+^n \longrightarrow \{0, 1\}$, where $0, 1$ are the truth values. Some valuations that we will frequently use are the following: the *characteristic function* defined as $(\in \sigma) : \Sigma_+^n \longrightarrow \{0, 1\}$, $(\in \sigma)(x) = 1 \iff x \in \sigma$; the *equality* defined as $(\approx) : \Sigma_+^2 \longrightarrow \{0, 1\}$, $(\approx)(x, y) = 1 \iff x = y$; or its curried restriction, $(\approx a)(x) = (\approx)(x, a)$. We will use also the notations: $x \in \sigma$, $x \approx y$ and $x \approx a$.

Definition 2. *Given $(\varphi_1, \ldots, \varphi_n) \in \zeta^{*n}$ and given a valuation B of arity n, we call the ordered pair $(B, (\varphi_1, \ldots, \varphi_n))$ a* rule. *Given a syntagma $S : \zeta^* \longrightarrow \Sigma_+$, and a rule, $R = (B, (\varphi_1, \ldots, \varphi_n))$, we say that S* satisfies *R, and we write, S sat R iff*

$$B(S(\varphi_1), \ldots, S(\varphi_n)) = 1.$$

Example 2. Let $\mathbf{Det} \subset \Sigma$ be the set of all determiners in English and let $Dt \in \zeta$ be a syntactic function. We want Dt to always take an element from **Det**. The rule $(\in \mathbf{Det}, (Dt))$, or informally $Dt \in \mathbf{Det}$, says exactly that.

Due to the boolean nature of valuations we can define new valuations from others. Let a pair of valuations be $B : \Sigma_+^n \longrightarrow \{0,1\}$, $B' : \Sigma_+^m \longrightarrow \{0,1\}$; then we can define:

$$B \wedge B' : \Sigma_+^{n+m} \longrightarrow \{0,1\}$$

$$(x_1, \ldots, x_n, y_1, \ldots, y_m) \longmapsto B(x_1, \ldots, x_n) \wedge B'(y_1, \ldots, y_n).$$

In the same way we can define: $B \vee B'$, $B \to B'$, $\neg B$, and so forth. If we have two rules: $R = (B, (\varphi_1, \ldots, \varphi_n))$ and $R' = (B', (\varphi'_1, \ldots, \varphi'_n))$ we can define: $R \wedge R' = (B \wedge B', (\varphi_1, \ldots, \varphi_n, \varphi'_1, \ldots, \varphi'_n))$, $R \vee R' = (B \vee B', (\varphi_1, \ldots, \varphi_n, \varphi'_1, \ldots, \varphi'_n))$, $R \to R' = (B \to B', (\varphi_1, \ldots, \varphi_n, \varphi'_1, \ldots, \varphi'_n))$, $\neg R = (\neg B, \varphi_1, \ldots, \varphi_n)$. Composing new valuations and rules does not increase our descriptive power but it makes reading easier.

Example 3. A transitive verb is a verb which always takes an object (e.g. the following sentence is ungrammatical: **John likes ∅*). Let $\mathbf{Trans} \subset \Sigma$ be the set of transitive verbs and let Ob be a syntactic function. Consider the valuation $B(x,y) = \big(\in \mathbf{Trans}\big)(x) \to \big(\not\approx 0\big)(y) = \big(x \in \mathbf{Trans} \to y \not\approx 0\big)$. The following rule asserts that the object cannot be null if a verb is transitive: $\big(B, (1, Ob)\big)$; or by a mild abuse of notation: $1 \in \mathbf{Trans} \to Ob \not\approx 0$.

4.2 Patterns and Manifolds

First of all we fix some easy set notation. As is usual in algebra, if $x \in X$ and $Y, Y' \subseteq X$, where X is a set with a binary operation $\cdot : X \times X \longrightarrow X$, then $x \cdot Y = \{x\} \cdot Y = \{x \cdot y \,|\, y \in Y\}$ and $Y \cdot Y' = \{y \cdot y' \,|\, y \in Y, y' \in Y'\}$.

Definition 3. *We say $\Gamma \subseteq \zeta^{*n}$ is a* (monoidal) pattern *of arity n iff it can be written as:*

$$\Gamma = \prod_{i=1}^{k} \varphi_i \cdot \Gamma_i,$$

where $\varphi_i \in \zeta^{*n}$ and each Γ_i is a finitely generated submonoid of ζ^{*n}.[5] *We call the number k the* length of the pattern.

From the definition we see that every submonoid of ζ^{*n} is itself a monoidal pattern. Some simple examples are:

Example 4. The set $\{(1,1,1)\}$ is a monoidal pattern of arity 3. We call such a pattern trivial. The set $\{(\varphi_1, \varphi_2)\}$ is a monoidal pattern with arity 2 and of only one element. We call such patterns constants. The set $\{(\alpha\beta^i\gamma\alpha^j, \gamma^2\alpha^i) \,|\, i,j \in \mathbb{N}_+\}$ is a monoidal pattern of arity 2 because it can be written as $(\alpha, \gamma^2) \cdot (\beta, \alpha)^* \cdot (\gamma, 1) \cdot (\alpha, 1)^*$. However the set $\{(\alpha^i\beta^j, \beta^j\alpha^i) \,|\, i,j \in \mathbb{N}_+\}$ is not a monoidal pattern because the set in itself is not a monoid nor a constant and it cannot be decomposed into a succession of products of submonoids and constants. On the other hand, the set $\{(\alpha^i\beta^j, \alpha^i\beta^j) \,|\, i,j \in \mathbb{N}^+\}$ can be written as: $(\alpha,\alpha)^* \cdot (\beta,\beta)^*$, and this shows that it is a pattern.

[5] Recall that some of the φ_i and Γ_i can be trivial.

We notate $H_{\alpha_1,\ldots,\alpha_t} = \{\alpha_1,\ldots,\alpha_t\}^*$, where $\alpha_1,\ldots,\alpha_t \in \zeta$. We also define, given a pattern Γ, the nth homogeneous power as the set $\Gamma^{n)} = \{(x,\ldots,x)_n \,|\, x \in \Gamma\}$, where the subscript means there are n copies of x in the vector. We call monoids such as $H^{n)}_{\alpha_1,\ldots,\alpha_t}$ homogeneous submonoids. So the last pattern from Example 4 can be written more comfortably as $(\alpha,\alpha)^* \cdot (\beta,\beta)^* = H^{2)}_{\alpha} H^{2)}_{\beta}$. We say that a pattern is homogeneous iff all its submonoids are homogeneous.

Figure 3 depicts the Cayley graph of the free monoid $\zeta^* = \{\alpha,\beta\}^*$. After the constant patterns, the simplest patterns are those of the form $\varphi\Gamma$, and they can be understood as follows: we fix some loci defined by the components of the constant φ, then we geometrically translate them around the syntagma according to Γ as in Fig. 3(b) which is depicting a pattern of arity 3, $(1,\alpha,\beta)H^{3)}_{\beta} = (1,\alpha,\beta)(\beta,\beta,\beta)^*$. If $\Gamma = \zeta^{*n)}$ then for each subsyntagma under φ we have a copy of φ. However when we multiply on the left side, $\Gamma\varphi$, the constant is reproduced in parallel as in Fig. 3(a) which is depicting a pattern of arity 2, $H^{2)}_{\alpha}(1,\beta) = (\alpha,\alpha)^*(1,\beta)$. More complex patterns composing these actions can also be visualized.

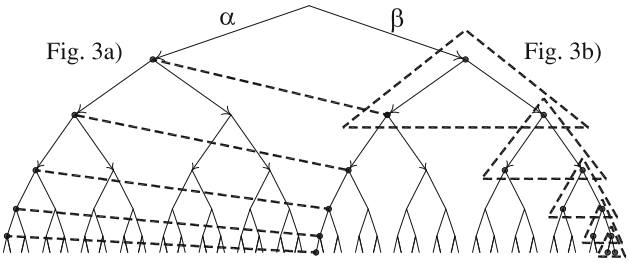


Fig. 3. (a) the pattern $(\alpha,\alpha)^*(1,\beta)$. (b) the pattern $(1,\alpha,\beta)(\beta,\beta,\beta)^*$.

Definition 4. *Let Γ be a monoidal pattern and B a valuation, both with the same arity. We call the pair (B,Γ) a* pattern rule *which defines a set of rules denoted by $\binom{B}{\Gamma} = \{(B,\varphi) \,|\, \varphi \in \Gamma\}$.*

Definition 5. *Given a pattern rule (B,Γ) we define the* simple syntactic manifold $\mathbf{Synt}_{\Sigma,\zeta}\binom{B}{\Gamma} = \{S \in \mathbf{Synt}_{\Sigma,\zeta} \mid S \,sat\, R \;\forall R \in \binom{B}{\Gamma}\}$. *Given a number of pattern rules we define the* syntactical manifold*:*

$$\mathbf{Synt}_{\Sigma,\zeta}\begin{pmatrix} B_1 \cdots B_n \\ \Gamma_1 \cdots \Gamma_n \end{pmatrix} = \mathbf{Synt}_{\Sigma,\zeta}\begin{pmatrix} B_1 \\ \Gamma_1 \end{pmatrix} \cap \cdots \cap \mathbf{Synt}_{\Sigma,\zeta}\begin{pmatrix} B_n \\ \Gamma_n \end{pmatrix}.$$

When the sets ζ, Σ are understood we just write: **Synt**. From the definitions it immediately follows that if V,W are syntactic manifolds, then $V \cap W$ is a syntactic manifold, so syntactic manifolds form a semilattice.

5 Five Classical Examples

We show five classical formal language examples, presented in two groups. The monoidal patterns will highlight a symmetry between these languages.

5.1 Squares Language vs. Copy Language and Mirror Language

We fix a vocabulary, for instance $\Sigma^* = \{a, b, c\}$. Consider the languages $L_{\text{squa}} = \{x_1^2 \cdots x_n^2 \,|\, x_1, \ldots, x_n \in \Sigma, n \in \mathbb{N}_+\}$, and $L_{\text{copy}} = \{xx \,|\, x \in \Sigma^*\}$. The first is a context free, indeed regular, language. It is a mathematical idealization of a chain of subordinate clauses in many languages such as English. E.g.:

(7) ... that John^a saw^a Peter^b help^b Mary^c read^c.

The second is not context free and it represents the typical chain of subordinate clauses of Dutch:[6]

(8) ... dat Jan^a Piet^b Marie^c zag^a helpen^b lezen^c.
... that Jan Piet Marie saw help read.
'... that J. saw P. help M. read'.

Consider the squares language L_{squa}. We take the syntactic manifold with $\zeta = \{\alpha, \beta\}$ and $\Sigma = \{a, b, c\}$ as follows. Let there be the valuation $x \approx 0$ and $x \approx y$ (we write simply $\approx$). Then we have the manifold:

$$W_{\text{squa}} = \mathbf{Synt}\begin{pmatrix} \approx 0 & \approx 0 \\ \alpha^2 H_\beta & \beta\alpha H_\beta \end{pmatrix} \cap \mathbf{Synt}\begin{pmatrix} \approx \\ (1,\alpha)H_\beta^{2)} \end{pmatrix}.$$

We separate this manifold in two parts because the second is giving important information about the syntagmata. The first part can be considered superfluous; it is just saying that all loci not invoked by the second part will be null.

If we take the projective linearization Π_{squa} as in Fig. 4(a) for the manifold W_{squa}, we will obtain the language $\Pi_{\text{squa}}(W_{\text{squa}}) = L_{\text{squa}}$.

For the copy language we define the manifold:[7]

$$W_{\text{copy}} = \mathbf{Synt}\begin{pmatrix} \approx 0 & \approx 0 \\ \alpha H_\beta \alpha & \beta H_\beta \alpha \end{pmatrix} \cap \mathbf{Synt}\begin{pmatrix} \approx \\ H_\beta^{2)}(1,\alpha) \end{pmatrix}.$$

If we take the projective linearization Π_{copy} as in Fig. 4(b) for the manifold W_{copy}, we will obtain the language $\Pi_{\text{copy}}(W_{\text{copy}}) = L_{\text{copy}}$.

Note that both nonsuperfluous parts are very similar; the valuations are equal and the only difference lies on the laterality of patterns: $(1,\alpha)H_\beta^{2)}$ in the first language and $H_\beta^{2)}(1,\alpha)$ in the second.

[6] This also occurs in Swiss-German, [7].

[7] Our analysis of Dutch dependencies uses a parallel arrangement of the functions β which shares distant similarities with [1], but there the framework was constituency grammars.

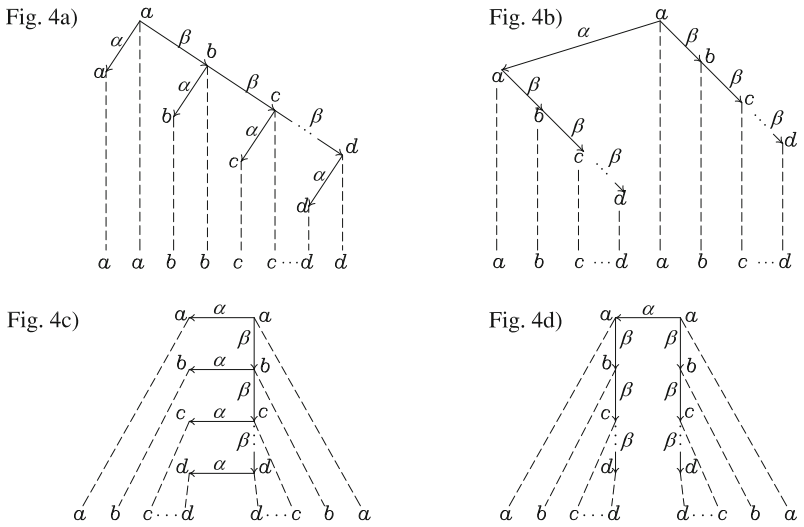


Fig. 4. (a) syntagmata for the squares language. (b) syntagmata for the copy language. (c) syntagmata for the mirror language. (d) alternative syntagmata for the mirror language.

Closely related to both the above languages we consider the mirror language defined as $L_{\text{mirr}} = \{xx^R \mid x \in \Sigma^*\}$, where x^R is the reversed word. This language captures the nested dependencies in German, with sentences like:

(9) ... dass Jan^a Piet^b Marie^c lesen^c helfen^b sah^a.
... that Jan Piet Marie read help saw.
'... that J. saw P. help M. read'.

Curiously these sentences can be projectively obtained from both manifolds W_{squa} and W_{copy} as can be checked in Fig. 4(c) and (d): if we define Π_1 as in Fig. 4(c) and Π_2 as in Fig. 4(d) we will have $\Pi_1(W_{\text{squa}}) = L_{\text{mirror}} = \Pi_2(W_{\text{copy}})$.

5.2 Language of Multiple *abc* vs. Respectively *abc*

Now we are going to consider the languages $L_{\text{mult}} = \{(abc)^n \mid n \in \mathbb{N}_+\}$ and $L_{\text{resp}} = \{a^n b^n c^n \mid n \in \mathbb{N}_+\}$. The first is a context free, indeed regular language, but not the second. The first corresponds to simple coordination, (10a), while the second is a mathematical idealization of the *respectively* construction (10b):

(10) a. Jean^a seems German^b but he is French^c, Pietro^a seems Russian^b but he is Italian^c and Peter^a seems Belgian^b, but he is English^c.
b. Jean^a, Pietro^a and Peter^a seem respectively German^b, Russian^b and Belgian^b, but they are French^c, Italian^c and English^c.

Let there be the sets $\zeta = \{\alpha, \beta, \gamma\}$, $\Sigma = \{a, b, c\}$, and let there be the valuations:

$$B(x, y, z) = (x \in \{0, a\}) \wedge (y \in \{0, b\}) \wedge (z \in \{0, c\}),$$

$$B'(x,y,z) = \big(x \approx 0 \wedge y \approx 0 \wedge z \approx 0\big) \vee \big(x \not\approx 0 \wedge y \not\approx 0 \wedge z \not\approx 0\big).$$

We take the manifold

$$W_{\text{mult}} = \mathbf{Synt}\begin{pmatrix} \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 \\ \alpha\alpha H_\beta & \beta\alpha H_\beta & \gamma\alpha H_\beta & \alpha\gamma H_\beta & \beta\gamma H_\beta & \gamma\gamma H_\beta \end{pmatrix} \cap \mathbf{Synt}\begin{pmatrix} B & B' \\ (\alpha,1,\gamma)H_\beta^{3)} & (\alpha,1,\gamma)H_\beta^{3)} \end{pmatrix}.$$

The projective linearization $\mathit{\Pi}_{\text{mult}}$ as in Fig. 5(a) for the manifold W_{mult} yields the language $\mathit{\Pi}_{\text{mult}}(W_{\text{mult}}) = L_{\text{mult}}$.

Finally we take the manifold with exactly the same valuations B, B' but with some specific changes in the patterns:

$$W_{\text{resp}} = \mathbf{Synt}\begin{pmatrix} \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 & \approx 0 \\ \alpha H_\beta \alpha & \alpha H_\beta \gamma & \alpha H_\beta & \gamma H_\beta & \alpha H_\beta \gamma & \gamma H_\beta \gamma \end{pmatrix} \cap \mathbf{Synt}\begin{pmatrix} B & B' \\ H_\beta^{3)}(\alpha,1,\gamma) & H_\beta^{3)}(\alpha,1,\gamma) \end{pmatrix}.$$

The projective linearization $\mathit{\Pi}_{\text{resp}}$ as in Fig. 5(b) for the manifold W_{resp} yields the language $\mathit{\Pi}_{\text{resp}}(W_{\text{resp}}) = L_{\text{resp}}$. Again note that both nonsuperfluous parts are very similar; the valuations are equal and the only difference lies in the laterality: $(\alpha,1,\gamma)H_\beta^{3)}$ and $H_\beta^{3)}(\alpha,1,\gamma)$.

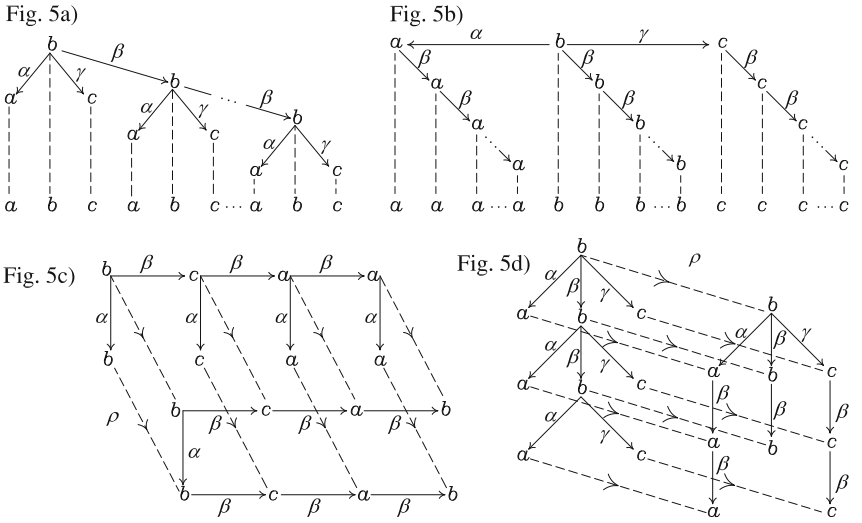


Fig. 5. (a) syntagmata for the language of multiple *abc*. (b) syntagmata for the respectively *abc* language. (c) symmetry between the squares language and the copy language. (d) symmetry between the multiple *abc* language and the respectively *abc* language.

6 How to Build Natural Languages

Due to the boolean nature of valuations we are able to implement a wide variety of rules. If we want to explain a natural language we can build a number of manifolds capturing different phenomena. We can begin with a manifold to describe for each word what kind of words or categories it can govern. For example, a noun can govern a determiner and an adjective, but not an adverb or verb. With another manifold we can define for each function what functions can follow, such as the rule for transitive verbs in Example 3.[8] We can define still another manifold to describe agreements. Then we have to think where these rules must hold, i.e. we must think of the patterns. For all rules of this kind the pattern will have the shape $\varphi\Gamma$. In fact these kind of patterns seems to be related to context free languages. Fortunately in order to add new phenomena, like *pied-piping*, *cross-serial dependencies* or others, we can build other manifolds and intersect with the existing ones. Possibly new phenomena demand new kinds of patterns. For example we already saw that pied-piping in Catalan had a pattern like $\varphi\Gamma\varphi'\Gamma'$.

7 Symmetric Syntagmata, Manifolds and Languages

To analyze sentences with cross-serial dependencies from (8) and (10) we can use the underlying logic of the simplified languages L_{copy} and L_{resp}. See Figs. 6(b) and (d). Nevertheless, this solution could be considered inadequate by the general reader for at least two reasons. First, it assumes different languages to have the same analysis (equal syntagmata up to words). Second, intuitively it is thought that governance must be coherent with semantical roles. We have violated both principles. On the one hand, we would have supposed different analyses for English and Dutch, or for respectively constructions and ordinary coordination; see again all the analyses in Fig. 6(a), (b), (c) and (d). On the other hand, in the Dutch analysis the function *Ob* should follow a verb, not a noun. We need to address these two points. Even though our analysis seems *ex professo* chosen to be comfortably projective,[9] there exists an easy and closer relationship between both analysis: an algebraic permutation of syntactic functions. We now look at this.

As is usual in algebra a morphism is a mapping preserving structures. In our case a morphism between syntagmata $S : \zeta^* \longrightarrow \Sigma_+$ and $S : \zeta'^* \longrightarrow \Sigma'_+$ is a pair of mappings $f : \zeta^* \longrightarrow \zeta'^*$, $g : \Sigma \longrightarrow \Sigma'$ (which may be partial) such that $g(0) = 0$ and they make commutative the diagram, $S' \circ f = g \circ S$:

$$\begin{array}{ccc} Spt(S) & \xrightarrow{f} & Spt(S') \\ {\scriptstyle S}\downarrow & & \downarrow{\scriptstyle S'} \\ \Sigma & \xrightarrow{g} & \Sigma' \end{array}$$

[8] Using the Tesnèrian denomination, this constitutes defining the *valence* of each word.
[9] A similar strategy can be found in Topological Dependency Grammar, [5].

Definition 6. *Let ρ be a permutation of subscripts $\{1,\dots,m\}$, and consider a disjoint decomposition $\zeta = \zeta_1 \cup \cdots \cup \zeta_m$. We call a pair (f,g) a* symmetry *when f is a permutation of submonoids, $f : \zeta_1^* \cdots \zeta_m^* \longrightarrow \zeta_{\rho(1)}^* \cdots \zeta_{\rho(m)}^*$, $f(x_1 \cdots x_m) = x_{\rho(1)} \cdots x_{\rho(m)}$. By a mild abuse of notation we simply write $\rho = (f,g)$.*

Proposition 2. *Using the same notation, given two manifolds W and W' we have that for each syntagma $S \in W$, $Spt(S) \subseteq \zeta_1^* \cdots \zeta_m^* \subseteq \zeta^*$, there exists a unique syntagma $S' \in W'$, $Spt(S') \subseteq \zeta_{\rho(1)}^* \cdots \zeta_{\rho(m)}^* \subseteq \zeta^*$, such that the symmetry becomes a morphism of syntagmata.*

Proof. For (f,g) to be a morphism, we require that $S' \circ f = g \circ S$. Note that f is bijective and well defined since the decomposition of ζ^* is disjoint, so given S we put $S' = g \circ S \circ f^{-1}$, and in this way S' is uniquely defined. Clearly if $Spt(S) \subseteq \zeta_1^* \cdots \zeta_m^*$ then $Spt(S') \subseteq \zeta_{\rho(1)}^* \cdots \zeta_{\rho(m)}^*$. □

Automatically a symmetry induces a mapping of manifolds $\tilde{\rho} : W \longrightarrow W'$, $S \mapsto S'$. When this occurs we say that the languages $L = \Pi(W)$ and $L' = \Pi'(W')$ are symmetric, and we write $L \bot L'$.

Example 5. We consider the symmetry, $f : \{\alpha\}^*\{\beta\}^* \longrightarrow \{\beta\}^*\{\alpha\}^*$; see Fig. 5(c). The pair $\rho = (f, Id)$ defines a bijection between manifolds $W_{\text{squa}} \longrightarrow W_{\text{copy}}$. So we have two symmetric languages $L_{\text{squa}} \bot L_{\text{copy}}$. Similary if we consider the symmetry, $f' : \{\alpha,\gamma\}^*\{\beta\}^* \longrightarrow \{\beta\}^*\{\alpha,\gamma\}^*$, see Fig. 5(d), we will have that $L_{\text{mult}} \bot L_{\text{resp}}$. Any language is symmetric with itself, because the pair (Id, Id) is trivially a symmetry. However the mirror language enjoys nontrivial selfsymmetry with the first pair (f, Id).

Symmetries seem to establish a close correspondence between context free languages and some non-context free languages. Regarding the question of the suitability of analysis, the fact is that in our model we do not have equal syntagmata, but isomorphic or symmetric syntagmata. Let us now see this effect in natural languages:

Example 6. Following the Figs. 6(a) and (b), we consider the symmetry from English to Dutch: $f : \{Sb\}^* \cdot \{Ob\}^* \longrightarrow \{Ob\}^* \cdot \{Sb\}^*$. Let g be the vocabulary mapping defined as: $g : \Sigma_{\text{English}} \longrightarrow \Sigma_{\text{Dutch}}$, $g(\text{says}) = \text{zegt}$, $g(\text{saw}) = \text{zag}$, $g(\text{help}) = \text{helpen}$, ... This suggests that subordinated clauses in English and Dutch are symmetric.

Example 7. Finally let us show a slightly more complex situation where we need elliptic syntagmata. Consider the ordinary coordination and the respectively construction:

(11) a. The young boy is English, the fat man German, and the blond woman Dutch.

b. The young boy, the fat man and the blond woman are respectively English, German and Dutch.

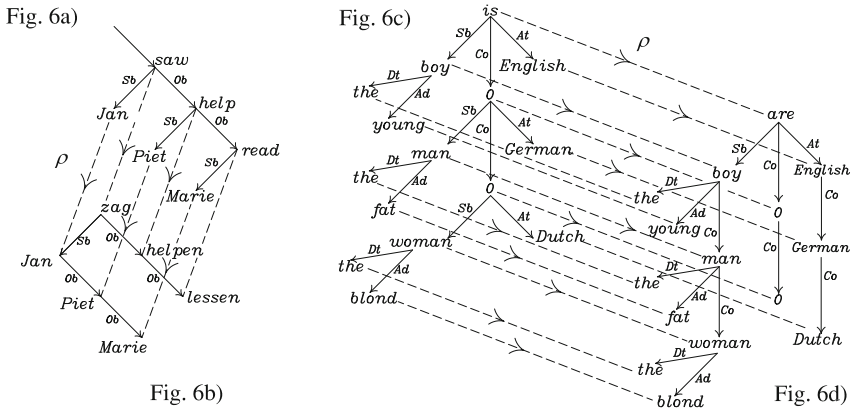


Fig. 6. (a) and (b) symmetry between subordinated clauses of English and Dutch. (c) and (d) symmetry between the ordinary coordinated construction and the respectively construction in English.

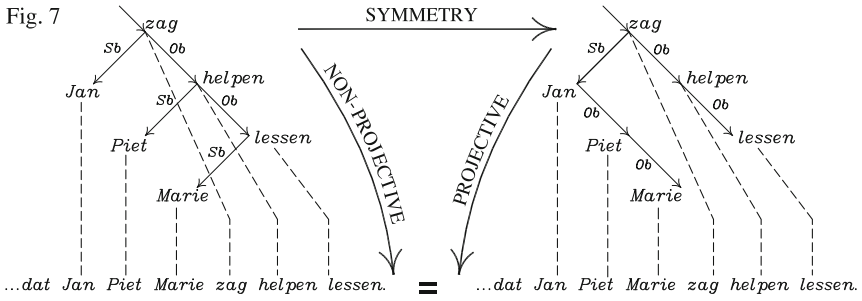


Fig. 7. Factorization of a non-projective linearization as a symmetry and a projective linearization.

See the dashed lines linking Figs. 6(c) and (d). Now the symmetry follows by commutation of the coordination function Co to the subject and object: $f : \{Dt, Ad\}^* \cdot \{Sb, Ob\}^* \cdot \{Co\}^* \longrightarrow \{Dt, Ad\}^* \cdot \{Co\}^* \cdot \{Sb, Ob\}^*$, while the vocabulary does not change anything except $g(\text{is}) = \text{are}$.

8 Conclusions

Initially in Sect. 1 we commented that there are some situations that seem not to accept projective structures. In such a circumstance we have two options: either we change the governance structure or we loosen the sense of projectivity. It is usually thought that changing the governance structure is more difficult than trying other projections. However patterns and symmetries allow us to explore the former solution.

Notwithstanding, there is another interpretation of the above results. If we want to insist on the usual analyses of dependencies, then we will have to linearize

non-projectively some trees, say $\Pi_{\text{No-Pr}}$. However then the symmetries tell us that the relationship $\Pi_{\text{No-Pr}}$ in some (!) cases can be factorized as $\Pi_{\text{No-Pr}} = \Pi_{\text{Pr}} \circ \tilde{\rho}$, where Π_{Pr} is a projective linearization, as in Fig. 7. That is:

$$\text{Non-projectivity} = \text{Symmetry} + \text{Projectivity}.$$

Acknowledgements. Thanks to *Formal Grammar* reviewing for many suggestions to improve this paper, and to Glyn Morill and Oriol Valentín for encouragement, advice and support. All errors are my own.

References

1. Bresnan, J., Kaplan, R.M., Peters, S., Zaenen, A.: Cross-serial dependencies in Dutch. In: Savitch, W.J., Bach, E., Marsh, W., Safran-Naveh, G. (eds.) The Formal Complexity of Natural Language, pp. 286–319. Springer, Netherlands (1987)
2. Debusmann, R.: An introduction to dependency grammar. Hausarbeit fur das Hauptseminar Dependenzgrammatik SoSe, pp. 1–16 (2000)
3. Debusmann, R., Duchier, D., Kruijff, G.J.M : Extensible dependency grammar: a new methodology. In: Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar, pp. 70–76 (2004)
4. Debusmann, R., Kuhlmann, M.: Dependency grammar: classification and exploration. In: Crocker, M.W., Siekmann, J. (eds.) Resource-Adaptive Cognitive Processes, pp. 365–388. Springer, Heidelberg (2010)
5. Duchier, D., Debusmann, R.: Topological dependency trees: a constraint-based account of linear precedence. In: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, pp. 180–187 (2001)
6. Kuhlmann, M.: Mildly non-projective dependency grammar. Comput. Linguist. **39**(2), 355–387 (2013)
7. Shieber, S.M.: Evidence against the context-freeness of natural language. In: Kulas, J., Fetzer, J.H., Rankin, T.L. (eds.) Philosophy, Language, and Artificial Intelligence. tudies in Cognitive Systems, vol. 2, pp. 79–89. Springer, Heidelberg (1987)

On the Mild Context-Sensitivity of k-Tree Wrapping Grammar

Laura Kallmeyer(✉)

Heinrich-Heine-Universität Düsseldorf, Düsseldorf, Germany
kallmeyer@phil.hhu.de

Abstract. Tree Wrapping Grammar (TWG) has been proposed in the context of formalizing the syntactic inventory of Role and Reference Grammar (RRG). It is close to Tree Adjoining Grammar (TAG) while capturing predicate-argument dependencies in a more appropriate way and being able to deal with discontinuous constituents in a more general way. This paper is concerned with the formal properties of TWG. More particularly, it considers k-TWG, a constrained form of TWG. We show that for every k-TWG, a simple Context-Free Tree Grammar (CFTG) of rank k can be constructed, which is in turn equivalent to a well-nested Linear Context-Free Rewriting System (LCFRS) of fan-out $k+1$. This shows that, when formalizing a grammar theory such as RRG, which is based on thorough and broad empirical research, we obtain a grammar formalism that is mildly context-sensitive.

Keywords: Tree rewriting grammars · Role and Reference Grammar · Simple context-free tree grammar · Mild context-sensitivity

1 Introduction

Tree Wrapping Grammar (TWG) [6] has been introduced in the context of formalizing the syntactic inventory of Role and Reference Grammar (RRG) [14,15]. A TWG consists of elementary trees, very much in the spirit of Tree Adjoining Grammar (TAG) [5], and from these elementary trees, larger trees are obtained by the operations of *(wrapping) substitution* and *sister adjunction*. (Wrapping) substitutions are supposed to add syntactic arguments while sister adjunction is used to add modifiers and functional operators. A discontinuous argument can be split via the wrapping substitution operation: the argument tree has a specific split node v. When adding such an argument to a predicate tree, the lower part (rooted in v) fills an argument slot via substitution while the upper ends up above the root of the target predicate tree.

[6] adopts the rather flat syntactic structure from RRG with categories CORE, CLAUSE and SENTENCE. A sample RRG-inspired TWG derivation is shown in Fig. 1. This example involves substitutions of the arguments *John* and *Mary* into the *hates* tree and of *Bill* into *claims*, sister adjunction of *definitely* into *hates* (sister adjunction simply adds a new daughter to a node where

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 77–93, 2016.
DOI: 10.1007/978-3-662-53042-9_5

the root of the adjoining tree has to match the target node), and wrapping substitution of the *hates* tree around the *claims* tree.

It was shown in [6] that, in contrast to TAG, the TWG operations enable us to add even sentential arguments with long-distance extractions by a substitution operation. In this, TWG is close to other formalisms in the context of TAG-related grammar research that have been proposed in order to obtain derivation structures that reflect dependencies in a more appropriate way than it is done by TAG [1,10,11].

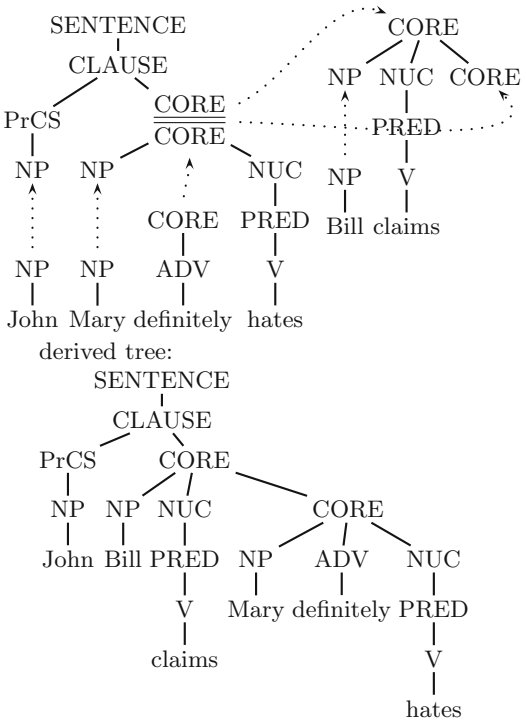


Fig. 1. RRG-style TWG derivation for *John Bill claims Mary definitely hates*

The focus of this paper is on the formal properties of TWG. After an introduction to TWG and in particular to k-TWG, a restricted form of TWG, we show how to construct an equivalent simple context-free tree grammar (CFTG) for a given k-TWG.

2 Tree Wrapping Grammar

The following introduction to TWG is largely taken from [6], except for the declarative definition of the notion of k-TWG, based on properties of the derivation trees decorated with wrapping substitution markings.

Borrowing from the TAG terminology, trees that can be added by substitution are called *initial* trees. In addition, we need *adjunct trees* for modeling modifiers and functional elements in RRG. These trees are added by sister adjunction. We distinguish **l**eft-adjoining, **r**ight-adjoining and unrestricted adjunct trees, resp. called *l-adjunct*, *r-adjunct* and *d-adjunct* trees. (The latter can add a **d**aughter at any position.)

Definition 1 (Tree Wrapping Grammar). *A* Tree Wrapping Grammar *(TWG) is a tuple* $G = \langle N, T, I, A_D, A_L, A_R, C\rangle$ *where*

(a) N, T *are disjoint alphabets of non-terminal and terminal symbols.*
(b) I, A_D, A_L *and* A_R *are disjoint finite sets of ordered labeled trees such that*
  - *each non-leaf in a tree is labeled by some element from* $N \cup N^2$*,*
  - *there is at most one node with a label from* N^2*,*
  - *leaves have labels from* $N \cup T$*, and*
  - *the root of each tree in* $A_D \cup A_L \cup A_R$ *has exactly one daughter.*

(c) $C \subseteq N$*.*

A non-terminal leaf is called a substitution node*, and the labels from* N^2 *are called* split categories*.*

Every tree in I *is called an* initial *tree, every tree in* $A_D \cup A_L \cup A_R$ *an* adjunct *tree and every tree in* $I \cup A_D \cup A_L \cup A_R$ *an* elementary tree*.*

As we will see later, C is the set of non-terminals that can occur on a *wrapping spine* (i.e., between root and substitution site of the target tree of a wrapping substitution).

There are two TWG composition operations (see Fig. 2):

1. *Standard/Wrapping substitution:* a substitution node v in a tree γ gets replaced with a subtree α' of an initial tree α. If $\alpha' \neq \alpha$, then the root node v' of α' must be labeled with a split category $\langle X, Y\rangle$ such that the root of γ is labeled X and v is labeled Y. α is then split at v' and wraps around γ, i.e., the upper part of α ends up above the root of γ while α' fills the substitution slot. In this case, we call the operation a *wrapping substitution*. Otherwise ($\alpha = \alpha'$), we have a *standard substitution* and the root of α (i.e., v') must have the same label as v.
2. *Sister adjunction:* an adjunct tree β with root category X is added to a node v of γ with label X. The root r_β of β is identified with v and the (unique) daughter of r_β is added as a new daughter to v. Furthermore, if $\beta \in A_L$ (resp. $\beta \in A_R$), then the new daughter must be a leftmost (resp. rightmost) daughter.

A slightly different form of tree wrapping is proposed in [9] for RRG, leading to a flatter structure. One can consider a split node as a very special dominance edge (with specific constraints on how to fill it). In our definition, we would then have for a split node with categories X, Y a dominance edge between a node labeled X and a node labeled Y such that the X-node does not have any other

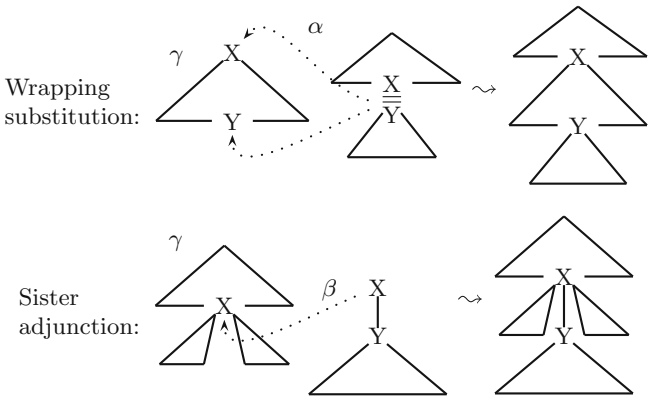


Fig. 2. Operations in TWG

daughters. In the flatter version of wrapping [9], the X-node can have other daughters that end up being sisters of the target tree of the wrapping that fills this dominance edge (see Fig. 3).

It is easy to see that this form of wrapping can be transformed into the one used in this paper, simply by replacing the dominance edge with an immediate dominance edge and splitting the lower node with top and bottom categories X and Y respecitvely. As a result, we obtain trees that are slightly less flat than the ones from [9] and that, if we keep track of which edges have been added in this transformation, can be easily transformed back to the original flatter form. Therefore, without losing anything concerning the desired linguistic structures, for formal properties and parsing considerations we can work with the tree wrapping definition presented here.

Every elementary tree in a TWG G is a derived tree wrt. G, and every tree we can obtain with our composition operations from derived trees in G is again a derived tree. Wrapping substitutions require that, in the target tree, all categories on the path from the root to the substitution node (the *wrapping spine*) are in C. A further constraint is that wrapping substitution can target only initial trees, i.e., we cannot wrap a tree around an adjunct tree. Note that, in contrast to [6], we do not impose a limit on the number of wrapping substitutions

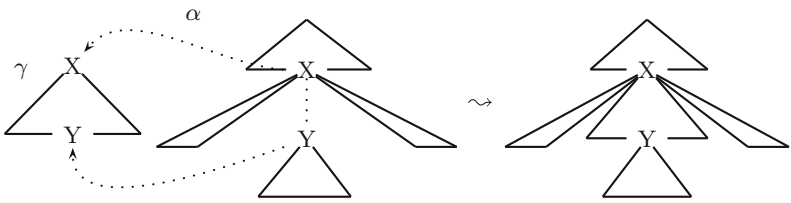


Fig. 3. Wrapping from [9]

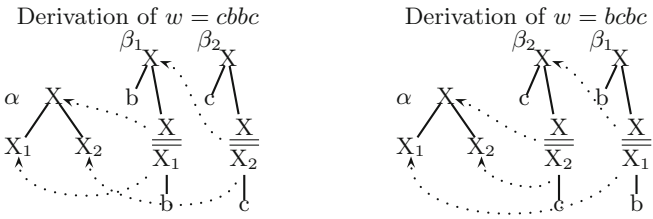


Fig. 4. Sample derivations: wrapping substitutions at sister nodes

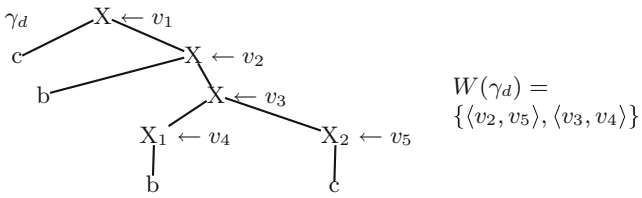


Fig. 5. Decorated derived tree arising from the first derivation in Fig. 4

stretching across a node in our definition of a TWG derived trees. This constraint comes later with the definition of k-TWG.

So far, wrapping can occur several times at the same elementary tree. Or, to put it differently, a node can be on several wrapping spines that are not nested. See Fig. 4 for an example. In the two derivations, the root node of α is part of the two wrapping spines. In other words, both wrappings stretch across the root node of α. This has implications for the generative capacity and also for the parsing complexity.

We now want to restrict the wrapping substitutions in a derivation concerning the number of times a node can be part of a wrapping spine. To this end, we first introduce some decoration for the derived trees in the TWG given a specific derivation: Let γ_d be a tree derived in a TWG with a fixed derivation (there can be several derivations per derived tree and, consequently, several decorations). We define the wrapping decoration of γ_d as the following set of node pairs $W(\gamma_d)$: In every wrapping substitution step of the derivation in question with r and v being the root node r and the substitution node v of the target of the wrapping substitution, $\langle r, v\rangle \in W(\gamma_d)$. Nothing else is in $W(\gamma_d)$. We call every $v_\perp$ such that there exists a $v_\top$ with $\langle v_\top, v_\perp\rangle \in W(\gamma_d)$ a $\perp$ *node* in γ_d. We call a derived tree with such a decoration a *decorated* derived tree. An example is given in Fig. 5.

Once we have that, we can identify for a node v in such a decorated derived tree γ all wrapping substitution sites ($\perp$ nodes) where the wrapping substitution stretches across v.

Definition 2 (Gap set, wrapping degree). *Let $\gamma = \langle V, E, \prec, r, l\rangle$ be a decorated TWG derived tree with decoration $W(\gamma)$, and let $v \in V$.*

1. A set $V_\perp \subset V$ of $\perp$ nodes is a gap set *with respect to v if*

   (a) *for every pair* $\langle v_\top, v_\bot\rangle \in W(\gamma)$ *with* $v_\bot \in V_\bot$*, it holds that* $v_\top$ *dominates* v *and* v *strictly dominates* $v_\bot$*, and*
   (b) *for every pair* $\langle v_\top, v_\bot\rangle \in W(\gamma)$ *with* $v_\bot \in V_\bot$*, there is no pair* $\langle v'_\top, v'_\bot\rangle \in W(\gamma)$*,* $\langle v'_\top, v'_\bot\rangle \neq \langle v_\top, v_\bot\rangle$ *with* $v_\top$ *dominating* $v'_\top$*,* $v'_\top$ *dominating* v*,* v *strictly dominating* $v'_\bot$*, and* $v'_\bot$ *dominating* $v_\bot$*.*
2. *We then define the* wrapping degree *of* v *as the cardinality of its gap set.*
3. *The* wrapping degree *of a decorated derived tree is the maximal wrapping degree of its nodes, and the* wrapping degree *of a derived tree* γ_d *in the TWG* G *is the minimal wrapping degree of any decorated derived tree with respect to* G *that yields* γ_d*.*

In the example in Fig. 5, we have for instance a gap set $\{v_4, v_5\}$ for the node v_3. The wrapping degree of this derived tree is 2.

Now we can define the tree language for a TWG in general and in the case where wrapping degrees are limited by a $k \geq 0$:

Definition 3 (Language of a TWG). *Let* G *be a TWG.*

- *A* saturated *derived tree is a derived tree without substitution nodes and without split categories.*
- *The* tree language *of* G *is* $L_T(G) = \{\gamma \mid \gamma$ *is a saturated derived initial tree in* $G\}$*.*
- *The* string language *of* G *is the set of yields of trees in* $L_T(G)$*.*
- *The* k-tree language *of* G *is* $L^k_T(G) = \{\gamma \mid \gamma$ *is a saturated derived initial tree in* G *with a wrapping degree* $\leq k\}$*.*
- *The* k-string language *of* G *is the set of yields of trees in* $L^k_T(G)$*.*

Some TWGs are such that the maximal wrapping degree is limited, given the form of the elementary trees. But this is not always the case. In the following, we call a TWG a k-TWG if we impose k as a limit for the wrapping degree of derived trees, i.e., for a k-TWG, we consider the k-tree language of the grammar as its tree language.

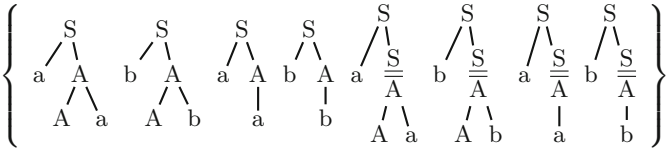
As an example, Fig. 6 gives a TWG for the copy language. Here, all substitution nodes must be filled by wrapping substitutions since there are no trees with root label A, and the nodes with the split categories are always the middle nodes. The grammar is such that only derived trees with a wrapping degree 1 are possible.

In order to facilitate the construction of an equivalent simple CFTG for a given k-TWG, we show the following normal form lemma:

Lemma 1. *For every* k*-TWG* $G = \langle N, T, I, A_D, A_L, A_R, C\rangle$*, there is exists a weakly equivalent* k*-TWG* $G' = \langle N', T, I', \emptyset, \emptyset, \emptyset, C\rangle$*, i.e., a* k*-TWG without adjunct trees.*

The construction idea is again rather simple. For every daughter position, we precompile the possibility to add something by sister adjunction in the following way: We start with $I_{temp} = I$ and $I' = \emptyset$ and we set $A_l = A_L \cup A_D$ and $A_r = A_R \cup A_D$.

$G = \langle \{\mathrm{S}, \mathrm{A}\}, \{a, b\}, I, \emptyset, \emptyset, \{\mathrm{S}, \mathrm{A}\} \rangle$ with $I =$



Sample derivation of *baabaa*:

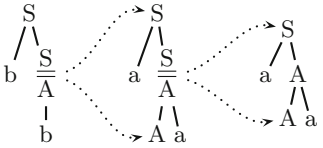


Fig. 6. TWG for the copy language $\{ww \mid w \in \{a, b\}^+\}$

1. For every adjunct tree β, we add a subscript l (r or d respectively) to the root label of β if β is in A_l (resp. in A_r or in A_D). The resulting tree is added to I_{temp}.
2. For every $\gamma \in I_{temp}$: For every node v in γ that is not the root of a former adjunct tree: If v has i daughters, then we pick one combination $i_1, \ldots, i_k$ ($k \geq 0$) with $0 \leq i_1 < \ldots < i_k \leq i$ of positions between daughters. We then add new daughters to v at all these positions, labeled with the non-terminal of v and a subscript l for position 0, r for position i and d otherwise. The result is added to I'. This is repeated until I' does not change any more, i.e., all possible combinations of daughter positions for the nodes in γ have been taken into account.
3. For every $\gamma \in I'$ that is a former adjunct tree: Add
   - a tree γ_l to I' that consists of γ with an additional leftmost daughter of the root having the same label as the root and a subscript l in case the subscript of the root is l, d otherwise.
   - a tree γ_r to I' that consists of γ with an additional rightmost daughter of the root having the same label as the root and a subscript r in case the subscript of the root is r, d otherwise.
   - a tree γ_{lr} to I' that consists of γ with two additional daughters of the root, one leftmost and one rightmost daughter such that these daughters have the same label as the root and the following subscripts: the leftmost has a subscript l in case the subscript of the root is l, d otherwise. The rightmost has a subscript r in case the subscript of the root is r, d otherwise.

An example of this construction can be found in Fig. 7.

The equivalence of the original grammar and the constructed one is obvious. The latter requires the same number of wrapping substitutions as the original one and has the same string language for the same k. But it generates different derived trees since in cases of multiple adjunctions between two nodes we obtain a binary structure.

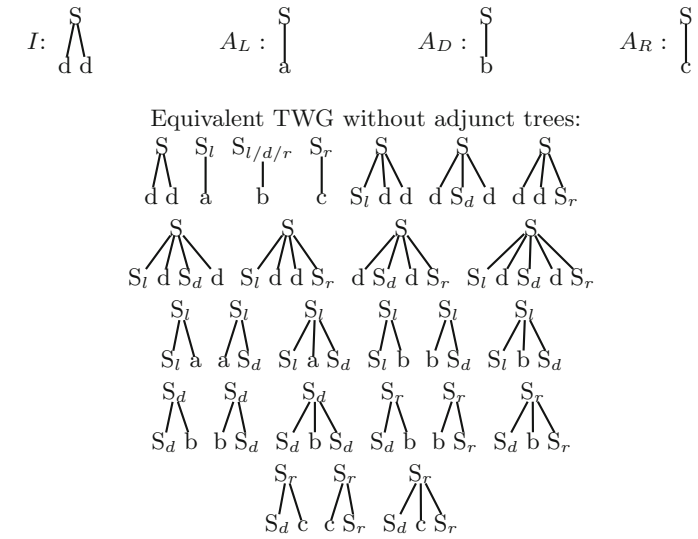


Fig. 7. Sample elimination of adjunct trees

3 Relation to Context-Free Tree Gramamrs

We now show that for every k-TWG one can construct an equivalent simple context-free tree grammar of rank k. This, in turn, is weakly equivalent to well-nested $(k+1)$-LCFRS (see [13,16] for the definition of LCFRS and [3,7] for well-nested LCFRS).

Without loss of generality, we assume the k-TWG to be without adjunct trees.

3.1 Context-Free Tree Grammars

The following introduction to context-free tree grammars is taken from [8].

A ranked alphabet is a union $\Delta = \bigcup_{r \in \mathbb{N}} \Delta^{(r)}$ of disjoint sets of symbols. If $f \in \Delta^{(r)}$, r is the *rank* of f.

A tree over a ranked alphabet Δ is a labeled ordered tree where each node with n daughters is labeled by some $f \in \Delta^{(n)}$. We use the term representation of trees. The set $\mathcal{T}_\Delta$ of trees over Δ is defined as follows: 1. If $f \in \Delta^{(0)}$, then $f \in \mathcal{T}_\Delta$. 2. If $f \in \Delta^{(n)}$ and $t_1, \ldots, t_n \in \mathcal{T}_\Delta (n \geq 1)$, then $(f t_1 \ldots t_n) \in \mathcal{T}_\Delta$.

If Σ is an (unranked) alphabet and Δ a ranked alphabet ($\Sigma \cap \Delta = \emptyset$), let $\mathcal{T}_{\Sigma,\Delta}$ be the set of trees such that whenever a node is labeled by some $f \in \Delta$, then the number of its children is equal to the rank of f.

For a set $X = \{x_1, \ldots, x_n\}$ of variables, $\mathcal{T}_\Delta(X)$ denotes the set of trees over $\Delta \cup X$ where members of X all have rank 0. Such a tree t containing the variables X is often written $t[x_1, \ldots, x_n]$. If $t[x_1, \ldots, x_n] \in \mathcal{T}_\Delta(X)$ and $t_1, \ldots, t_n \in \mathcal{T}_\Delta$, then $t[t_1, \ldots, t_n]$ denotes the result of substituting $t_1, \ldots, t_n$ for

$x_1, \ldots, x_n$, respectively, in $t[x_1, \ldots, x_n]$. An element $t[x_1, \ldots, x_n] \in \mathcal{T}_\Delta(X)$ is an n-context over Δ if for each $i = 1, \ldots, n$, x_i occurs exactly once in $t[x_1, \ldots, x_n]$.

Definition 4 (Context-free tree grammar). *A* context-free tree grammar *(CFTG) [2,12] is a quadruple $G = \langle N, \Sigma, P, S \rangle$, where*

1. *N is a ranked alphabet of non-terminals,*
2. *Σ an unranked alphabet of terminals,*
3. *$S \in N$ is of rank 0, and*
4. *P is a finite set of productions of the form*

$$Ax_1 \ldots x_n \to t[x_1, \ldots, x_n]$$

where $A \in N^{(n)}$ and $t[x_1, \ldots, x_n] \in \mathcal{T}_{\Sigma,N}(\{x_1, \ldots, x_n\})$.

The rank *of G is* $\max\{r \mid N^{(r)} \neq \emptyset\}$.

For every $s, s' \in \mathcal{T}_{\Sigma,N}$, $s \Rightarrow_G s'$ is defined to hold if and only if there is a 1-context $c[x_1] \in \mathcal{T}_{\Sigma,N}(\{x_1\})$, a production $Bx_1 ... x_n \to t[x_1, ..., x_n]$ in P, and trees $t_1, ..., t_n \in \mathcal{T}_{\Sigma,N}$ such that $s = c[Bt_1 \ldots t_n]$, $s' = c[t[t_1, \ldots, t_n]]$.

The relation $\Rightarrow_G^*$ is defined as the reflexive transitive closure of$\Rightarrow_G$. The tree language $L(G)$ generated by a CFTG G is defined as $\{t \in \mathcal{T}_\Sigma \mid S \Rightarrow_G^* t\}$. The string language is the set of yields of the trees in $L(G)$.

A CFTG is said to be simple if all right-hand sides of productions in the grammar are n-contexts, in other words, they contain exactly one occurrence of each of their n variables.

3.2 k-TWG and Simple CFTG

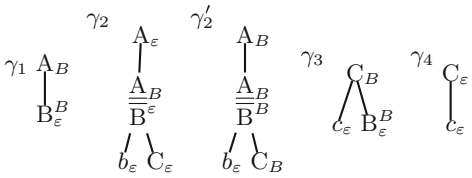
In the following, we will show that for each k-TWG, an equivalent simple context-free tree grammar of rank k can be constructed.

Let us explain the construction while going through the simple example in Fig. 9. The CFTG non-terminals have the form $[A, A_1 A_2 \ldots A_n]$ with $n \leq k$,

CFTG for $\{w^3 \mid w \in \{a, b\}^+\}$:
$N^0 = \{\overline{S}\}, N^{(3)} = \{\overline{X}\}$, $\Sigma = \{a, b, A\}$, S the start symbol.
P contains the following productions:
$\overline{S} \to \overline{X}aaa \mid \overline{X}bbb$
$\overline{X}x_1x_2x_2 \to \overline{X}(Aax_1)(Aa_2)(Aax_3) \mid \overline{X}(Abx_1)(Abx_2)(Abx_3) \mid Ax_1x_2x_3$
Sample derivation for the string *abaabaaba*:
$\overline{S} \Rightarrow \overline{X}aaa \Rightarrow \overline{X}(Aba)(Aba)(Aba)$
$\Rightarrow \overline{X}(Aa(Aba))(Aa(Aba))(Aa(Aba))$
$\Rightarrow A(Aa(Aba))(Aa(Aba))(Aa(Aba))$

Fig. 8. Simple CFTG for the double copy language

TWG for $\{(bc)^n \mid n \geq 1\} \cup \{c\}$:



Equivalent simple CFTG:
$N^{(0)} = \{S, [A], [C]\}, N^{(1)} = \{[A, B], [C, B]\}$, start symbol S, productions:
$S \to [A]$, $S \to [C]$
γ_1: $[A, B]x_1 \to Ax_1$
γ_2: $[A] \to A([A, B](Bb[C]))$
γ_2': $[A, B]x_1 \to A([A, B](Bb([C, B]x_1)))$
γ_3: $[C, B]x_1 \to Ccx_1$
γ_4: $[C] \to Cc$

Fig. 9. Sample TWG and equivalent simple CFTG

$A \in N$ and $A_i \in N$ for $1 \leq i \leq n$ where the intuition is the A is the root category of the tree this nonterminal expands to and $A_1 A_2 \dots A_n$ are the categories of pending gaps from wrappings that stretch across this tree. In other words, in the final decorated derived tree, they are the categories of the gap set nodes of this root, in linear order. Note that, since we assume a k-TWG, there cannot be more than k such categories. The gap trees that are to be inserted in the gap nodes (which are substitution nodes) are the arguments of this non-terminal. In other words, such a non-terminal tells us that we have to find an A-tree and there are pending lower parts of split trees with categories $A_1, \dots, A_n$, which have to be inserted into that A-tree. For instance, the category $[A, B]$ in our example expands to A-trees that need a B-substitution node at some point (maybe after some further substitution), in order to insert the B-gap tree to which this category applies. The γ_1-rule in the TWG for instance encodes that one way to find such a tree is to create an A-node with a single daughter, where this daughter is the pending B-tree.

The construction does not go directly from an elementary TWG tree to a single production. Instead, it yields a single production for each possible decoration of the TWG tree with category sequences corresponding to possible gap set node labels in linear order that can arise within a derivation. An example where we have more than one possibility for a single TWG tree is the tree γ_2 in Fig. 9 where the γ_2 and γ_2' indicate the two cases. Accordingly, there are two productions. One $[A]$-production where $[A]$ is of rank 0. This is the case where nothing else is wrapped around γ_2 and, consequently, there is no pending gap at its root node. The second production (the γ_2' case) is the possibility to have something wrapped around the γ_2 tree. In this case, the gap category of the outer

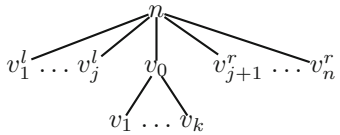
tree is B, and this gap must be placed somewhere below the C node, hence the non-terminal $[C, B]$ with the pending gap as argument for this node.

In order to keep track of these sequences of gap labels, we first define possible mappings f_1, f_2 for every elementary tree γ that assign to every node x in γ either a single sequence $f_1(x) = f_2(x)$ of non-terminals (= gap node labels) or, if the node is a split node or a substitution node that is used for a wrapping substitution, a pair of two possibly different such subsequences $\langle f_1(x), f_2(x)\rangle$. Intuitively, a split node starts a new gap, which is then filled by the lower part of the split node. Any gaps in the tree below the gap are accessible at the mother node of the split node.

Figure 9 gives the assignments f_1 and f_2 for each node as a super- and a subscript. In cases where $f_1 = f_2$, there is just one subscript, while for $f_1 \neq f_2$ (split nodes and wrapping substitution nodes), we have both. For γ_2, we have two possible assignments. The mapping of γ_1 tells us that this tree is used in a wrapping configuration where a split tree with some lower category B is wrapped around it. Furthermore, according to the B-node annotation, this leaf is filled by the wrapping substitution ($f_2 = \varepsilon$). The first assignment for γ_2 tells us that this tree is used without wrapping anything around it. At its split node, we wrap it around something that has to contain a B-gap ($f_1 = B$), which will be filled by the lower part of the split tree, therefore at that point, no more gaps are pending ($f_2 = \varepsilon$). In contrast to this, the second γ_2 is used in a wrapping configuration where a split tree with some lower category B is wrapped around it ($f_1 = f_2 = B$ at the root). The B gap arising from the split node in the middle is filled by the lower B node. However, the overall B gap is still pending, therefore we have $f_2 = B$ at the split node. This pending gap is not inserted at the substitution node (category C), instead, the information about the B-gap is passed ($f_1 = f_2 = B$). It can be inserted in γ_3. There the substitution node has $f_1 = B$ (which means that we need a B-substitution node to be filled with some pending B-tree) and $f_2 = \varepsilon$ (signifying that this is the substitution node we were looking for, no more pending gaps below).

The definition of these assignments is such that we guess the pending gap categories for the leaves, we guess whether a substitution node is used for wrapping, and for split nodes, we guess the pending gap categories that arise out of the tree that this node is wrapped around. The rest is calculated in a bottom-up way as follows:

- For a substitution node v with category A: either v is not used for a wrapping substitution and we have $f_1(v) = f_2(v) = A_1 \dots A_i$ $(0 \leq i)$ or v is used for a wrapping substitution and we have $f_1(v) = A$ and $f_2(v) = \varepsilon$.
- $f_2(v_0) = f_1(v_1) \dots f_1(v_j)$ for every node v_0 with $v_1, \dots v_j$ being all daughters of v_0 in linear precedence order such that none of the daughters is a split node.
- For every split node v_0 with top label X and bottom label Y with $v_1, \dots v_k$ being all daughters of v_0 in linear precedence order, n being the mother of v_0 and $v_1^l, \dots, v_j^l$ and $v_{j+1}^r, \dots, v_n^r$ being the sisters of v_0 to the left and right in linear precedence order:



$f_2(v_0) = f_1(v_1) \dots f_1(v_k)$ and there are $B_1, \dots, B_j \in N$ such that
$f_1(v_0) = B_1 \dots B_i Y B_{i+1} \dots B_j$ and
$f_2(n) = f_1(v_1^l) \dots f_1(v_j^l) B_1 \dots B_i f_2(v_0) B_{i+1} \dots B_j f_1(v_{j+1}^r) \dots f_1(v_n^r)$.
We call the Y in this step the split category.

- For every node v that is neither a split node nor a non-terminal leaf, we have $f_1(v) = f_2(v)$.
- For every leaf v with a terminal label, we have $f_1(v) = f_2(v) = \varepsilon$.
- The length of the assigned sequences is limited to k.
- For every node v with a non-terminal label from $N \backslash C$ (C is the set of categories allowed on wrapping spines), it holds that $f_1 = f_2 = \varepsilon$.

Instead of using the original TWG, we can also use the trees with annotations f_1, f_2 in TWG derivations. For these derivations, let us make the following assumptions: The conditions for wrapping are that the f_1 value of the split node must be the f_1 value of the root of the target tree while the bottom category of the split node must be the f_1 of the target substitution node and the f_2 of this substitution node must be ε. The annotation of the root of the target tree remains while the annotation of the substitution node is the f_2 value of the split node. Furthermore, annotations of substitution nodes that are not used for wrapping have to be equal to the ones of the root of the tree that substitutes in.

An example of such a TWG derivation can be found in Fig. 10.

For these TWG derivations, the following lemma holds:

Lemma 2. *With this annotated TWG we obtain exactly the set of derived trees of the original TWG including for each node in a derived tree, obtained with a specific derivation, a decoration with the labels of the nodes from its gap set in linear precedence order.*

This lemma holds since all possible combinations of pending gaps below substitution nodes and to the left and right of split nodes are considered in the f_1, f_2 annotations. Furthermore, gaps are passed upwards. The only way to get rid of a gap in the f_1, f_2 value of a root node is to wrap a tree filling this gap around it.

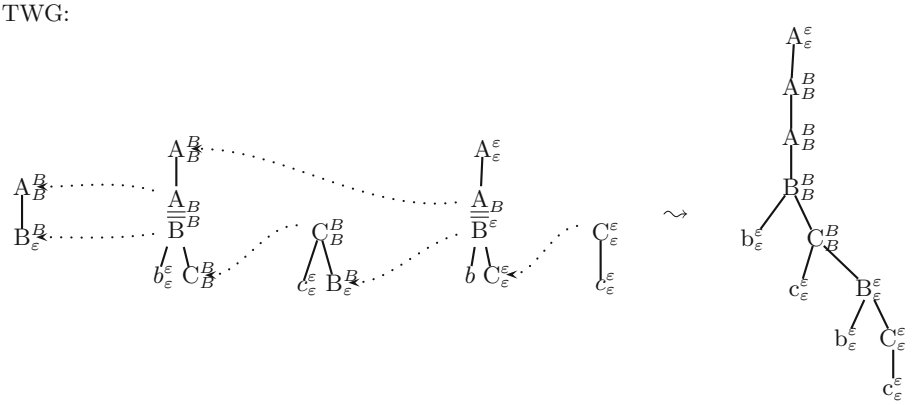
Given the gap assignment definition, we can now specify the set of productions in our CFTG that we obtain for each elementary tree.

1. For every non-terminal category X in our TWG, we add a production

$$S \rightarrow [X]$$

which is used for derived trees with root category X.

Sample derivations of $w = bcbc$:
TWG:



Corresponding CFTG derivation:
$S \Rightarrow [A] \Rightarrow A([A,B](Bb[C])) \Rightarrow A([A,B](Bb(Cc)))$
$\Rightarrow A(A([A,B](Bb[C,B](Bb(Cc))))) \Rightarrow A(A(A(Bb[C,B](Bb(Cc)))))$
$\Rightarrow A(A(A(Bb(Cc(Bb(Cc))))))$

Fig. 10. Sample derivations in the grammars from Fig. 9

2. For every tree γ in the k-TWG with root r and root category A and for every assignment $f = \langle f_1, f_2 \rangle$ for γ as defined above, we have productions

$$[A, f_1(r)]y_1 \dots y_{|f_1(r)|} \to \tau(\gamma, f)$$

where $\tau(\beta, f)$ for any subtree β of a TWG tree with gap assignment f is defined as follows:

- If β has only a single node v with non-terminal category B and $f_1(v) = f_2(v)$, then $\tau(\beta, f) = [B, f_1(v)]x_1 \dots x_{|f_1(v)|}$.[1]
- If β has only a single node v with non-terminal category B and $f_1(v) = A \in N$, $f_2 = \varepsilon$, then $\tau(\beta, f) = x_1$.
- If the root v of β is not a split node, its root category is A, and if $\beta_1 \dots, \beta_n$ are the daughter trees of v, then $\tau(\beta, f) = (A\tau(\beta_1, f) \dots \tau(\beta_n, f))$.
- If the root v of β is a split node with top category A and bottom category B, and if $\beta_1 \dots, \beta_n$ are the daughter trees of v, then $\tau(\beta, f) = ([A, f_1(v)]x_1 \dots x_i(B\tau(\beta_1, f) \dots \tau(\beta_n, f))x_{i+1} \dots x_j)$. where $f_1(v) = A_1 \dots A_i B A_{i+1} \dots A_j$ and B is the split category from the construction of f.

The variables $y_1, \dots, y_{|f_1(r)|}$ in the lefthand side of the production are exactly the ones from the righthand side in linear precedence order.

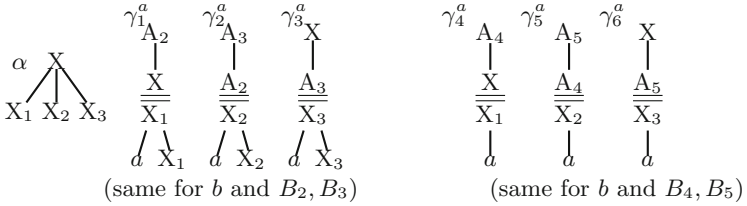
3. These are all the productions in the CFTG.

[1] We assume that fresh variables are used each time a new variable is needed.

An example of this construction can be found in Fig. 9, and a sample derivation in the TWG and the corresponding CFTG is given in Fig. 10. The TWG derivation involves two wrappings of γ_2 around γ_1, the first (inner one) with an additional substitution of γ_3 into the C substitution node, the second, outer one with a substitution of γ_4 into this slot. The corresponding CFTG derivation starts by expanding $[A]$ to the tree corresponding to the outer wrapping of γ_2, with a non-terminal $[C]$ for γ_4. Inside the resulting tree, we have a non-terminal $[A, B]$ of rank 1 whose argument is the tree $Bb(Cc)$) which has to fill a B-gap. This is then expanded to a γ_2 tree that assumes that there is a B-gap below its C-substitution node. This second use of γ_2 creates again the request for an A-tree with a B-gap (non-terminal $[A, B]$), which is now filled by γ_1, and, below its substitution node, it needs a C-tree with a B-substitution node (non-terminal $[C, B]$) which can then be filled by the pending B-tree $Bb(Cc)$) from the outer use of γ_2. Such a tree is provided by γ_3.

As a further example consider the TWG and corresponding CFTG in Fig. 11.

TWG for the double copy language $\{w^3 \mid w \in \{a, b\}^+\}$:



Equivalent simple CFTG:
Start symbol S, productions:

$S \to [X]$
γ_6^a: $[X] \to X([A_5, X_3](X_3 a))$
γ_5^a: $[A_5, X_3]x_1 \to A_5([A_4, X_2X_3](X_2 a)x_1)$
γ_4^a: $[A_4, X_2X_3]x_1x_2 \to A_4([X, X_1X_2X_3](X_1 a)x_1x_2)$
γ_3^a: $[X, X_1X_2X_3]x_1x_2x_3 \to X([A_3, X_1X_2X_3]x_1x_2(X_3 a x_3))$
γ_2^a: $[A_3, X_1X_2X_3]x_1x_2x_3 \to A_3([A_2, X_1X_2X_3]x_1(X_2 a x_2)x_3)$
γ_1^a: $[A_2, X_1X_2X_3]x_1x_2x_3 \to A_2([X, X_1X_2X_3](X_1 a x_1)x_2x_3)$
(same with b and B_2, B_3, B_4, B_5)
α: $[X, X_1X_2X_3]x_1x_2x_3 \to Xx_1x_2x_3$

Fig. 11. Sample 3-TWG and equivalent simple CFTG

The crucial part of the construction is actually the definition of the f_1, f_2 gap category annotations. Once we have this, the following holds:

Lemma 3. *There is a derived tree γ in the TWG with pending gap category sequence annotations $f = \langle f_1, f_2 \rangle$ (written $\langle \gamma, f \rangle$) as described above iff there is a corresponding derivation in the CFTG.*

Here, "corresponding derivation" means the following: if γ has gap sequence annotations f_1, f_2 and a root node v with node label A, then the correponding derivation is of the form $[A, f_1(v)]y_1 \dots y_{|f_1(v)|} \Rightarrow_G^* \tau(\gamma, f)$ where $\tau(\gamma, f)$ is as defined above in the construction for the case of elementary trees.

We can show this by an induction over the derivation structure, proving that

- The claim holds for elementary trees. This follows immediately from the construction.
- We assume that the claim holds for $\langle \gamma, f \rangle$ and $\langle \alpha, f_\alpha \rangle$ with corresponding CFTG derivations $[A_\gamma, gaps_\gamma]\boldsymbol{x}_\gamma \Rightarrow_G^* \tau(\gamma, f)$ and $[A_\alpha, gaps_\alpha]\boldsymbol{x}_\alpha \Rightarrow_G^* \tau(\alpha, f_\alpha)$. Then:
  $\langle \gamma', f' \rangle$ can be derived from $\langle \gamma, f \rangle$ in the TWG via substitution of $\langle \alpha, f_\alpha \rangle$ into one of the non-termial leaves
  $\Leftrightarrow$ this non-terminal leaf in $\langle \gamma', f' \rangle$ has category A_α and gap sequence $gaps_\alpha$
  $\Leftrightarrow$ there is a corresponding non-terminal $[A_\alpha, gaps_\alpha]$ in $\tau(\gamma, f)$ that can be expanded using the derivation $[A_\alpha, gaps_\alpha]\boldsymbol{x} \Rightarrow_G^* \tau(\alpha, f_\alpha)$ (induction assumption)
  $\Leftrightarrow$ $[A_\gamma, gaps_\gamma]\boldsymbol{x} \Rightarrow_G^* \tau(\gamma', f')$ where, in this derivation, we have one part $[A_\gamma, gaps_\gamma]\boldsymbol{x}_\gamma \Rightarrow_G^* \tau(\gamma, f)$ and a second part consisting of an application of $[A_\alpha, gaps_\alpha]\boldsymbol{x}_\alpha \Rightarrow_G^* \tau(\alpha, f_\alpha)$.
- We assume that the claim holds for $\langle \gamma, f \rangle$ and $\langle \beta, f_\beta \rangle$ with corresponding CFTG derivations $[A_\gamma, gaps_\gamma]\boldsymbol{x}_\gamma \Rightarrow_G^* \tau(\gamma, f)$ and $[A_\beta, gaps_\beta]\boldsymbol{x}_\beta \Rightarrow_G^* \tau(\beta, f_\beta)$. Then:
  $\langle \gamma', f' \rangle$ is derived from $\langle \gamma, f_\gamma \rangle$ in the TWG by wrapping $\langle \beta, f_\beta \rangle$ around $\langle \gamma, f_\gamma \rangle$
  $\Leftrightarrow$ there is a split node in $\tau(\beta, f_\beta)$ with category $\langle A_\gamma, Y \rangle$ and with an f_1 value $gaps_\gamma$ and there is a non-terminal leaf in $\langle \gamma, f_\gamma \rangle$ with category Y and with $f_1 = Y$ and $f_2 = \varepsilon$
  $\Leftrightarrow$ there is a non-terminal $[A_\gamma, gaps_\gamma]$ corresponding to the split node in $\tau(\beta, f_\beta)$ that can be expanded by the derivation $[A_\gamma, gaps_\gamma]\boldsymbol{x}_\gamma \Rightarrow_G^* \tau(\gamma, f)$
  $\Leftrightarrow$ there is a derivation $[A_\beta, gaps_\beta]\boldsymbol{x}_\beta \Rightarrow_G^* \tau(\gamma', f')$ in the CFTG consisting of $[A_\beta, gaps_\beta]\boldsymbol{x}_\beta \Rightarrow_G^* \tau(\beta, f_\beta)$ and then an application of $[A_\gamma, gaps_\gamma]\boldsymbol{x}_\gamma \Rightarrow_G^* \tau(\gamma, f)$.

With this lemma, we obtain the following theorem:

Theorem 1. *For every k-TWG there is an equivalent simple CFTG of rank k.*

As a consequence, we obtain that the languages of k-TWGs are in the class of well-nested linear context-free rewriting languages[2] and therefore mildly context-sensitive [16]. This term, introduced by [4], characterizes formalisms beyond CFG that can describe cross-serial dependencies, that are polynomially parsable and that generate languages of constant growth. Joshi's conjecture is that mildly

[2] Note that the fact that we can construct an equivalent well-nested LCFRS for a k-TWG does not mean that k-TWG (for some fixed k) cannot deal with ill-nested dependencies. The structures described by the LCFRS do not correspond to the dependency structures obtained from TWG derivations. The latter are determined only by the fillings of substitution slots.

context-sensitive grammar formalisms describe the appropriate grammar class for dealing with natural languages.

4 Conclusion

We have shown that k-TWG is a mildly context-sensitive grammar formalism, more particular, it falls into the class of simple context-free tree languages of rank k/well-nested $(k+1)$-LCFRS. This is an interesting result, considering that TWG arose out of an attempt to formalize the syntactic inventory of RRG, a grammar theory that emerged from broad empirical linguistic studies. Therefore the formal results in this paper support in a convincing way Joshi's conjecture about the mild context-sensitivity of natural languages.

References

1. Chiang, D., Scheffler, T.: Flexible composition and delayed tree-locality. In: TAG+9 Proceedings of the Ninth International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+9), Tübingen, pp. 17–24, June 2008
2. Engelfriet, J., Schmidt, E.M.: IO and OI. J. Comput. Syst. Sci. **15**(3), 328–353 (1977)
3. Gómez-Rodríguez, C., Kuhlmann, M., Satta, G.: Efficient parsing of well-nested linear context-free rewriting systems. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, California, June 2010, pp. 276–284. Association for Computational Linguistics (2010). http://www.aclweb.org/anthology/N10-1035
4. Joshi, A.K.: Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions? In: Dowty, D., Karttunen, L., Zwicky, A. (eds.) Natural Language Parsing, pp. 206–250. Cambridge University Press, New York (1985)
5. Joshi, A.K., Schabes, Y.: Tree-adjoning grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, pp. 69–123. Springer, heidelberg (1997)
6. Kallmeyer, L., Osswald, R., Van Valin Jr., R.D.: Tree wrapping for Role and Reference Grammar. In: Morrill, G., Nederhof, M.-J. (eds.) Formal Grammar 2012 and 2013. LNCS, vol. 8036, pp. 175–190. Springer, Heidelberg (2013)
7. Kanazawa, M.: The pumping lemma for well-nested multiple context-free languages. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 312–325. Springer, Heidelberg (2009)
8. Kanazawa, M.: Multidimensional trees and a Chomsky-Schützenberger-Weir representation theorem for simple context-free tree grammars. J. Logic Comput. (2014). doi:10.1093/logcom/exu043
9. Osswald, R., Kallmeyer, L.: Towards a formalization of Role and Reference Grammar. In: Proceedings of the 2013 Conference on Role and Reference Grammar (to appear)
10. Rambow, O., Vijay-Shanker, K., Weir, D.: D-Tree grammars. In: Proceedings of ACL (1995)
11. Rambow, O., Vijay-Shanker, K., Weir, D.: D-Tree substitution grammars. In: Computational Linguistics (2001)

12. Rounds, W.C.: Mappings and grammars on trees. Math. Syst. Theor. **4**(3), 257–287 (1970)
13. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theor. Comput. Sci. **88**(2), 191–229 (1991)
14. Van Valin Jr., R.D.: Exploring the Syntax-Semantics Interface. Cambridge University Press, Cambridge (2005)
15. Van Valin Jr., R.D., Foley, W.A.: Role and reference grammar. In: Moravcsik, E.A., Wirth, J.R. (eds.) Current Approaches to Syntax, Syntax and Semantics, vol. 13, pp. 329–352. Academic Press, New York (1980)
16. Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: Proceedings of ACL, Stanford (1987)

Distributional Learning and Context/Substructure Enumerability in Nonlinear Tree Grammars

Makoto Kanazawa[1(✉)] and Ryo Yoshinaka[2,3P]

[1] Principles of Informatics Research Division, National Institute of Informatics and SOKENDAI, Tokyo, Japan
kanazawa@nii.ac.jp
[2] Graduate School of Informatics, Kyoto University, Kyoto, Japan
[3] Graduate School of Information Sciences, Tohoku University, Sendai, Japan

Abstract. We study tree-generating almost linear second-order ACGs that admit bounded nonlinearity either on the context side or on the substructure side, and give distributional learning algorithms for them.

1 Introduction

Originally developed for efficient learning of context-free languages [3,13], the method of *distributional learning* under the paradigm of *identification in the limit from positive data and membership queries* has been successfully applied to a number of more complex grammatical formalisms that derive objects (strings, trees, λ-terms, etc.) through local sets of *derivation trees* [9,12,14]. In these formalisms, a subtree s of a complete derivation tree $t = c[s]$ contributes a certain "substructure" $S = \phi(s)$ which is contained in the whole derived object $T = \phi(t)$, and the remaining part $c[\,]$ of the derivation tree contributes a function $C = \phi(c[\,])$ that maps S to $T = C(S)$. We can think of C as a "context" that surrounds S in T. Fixing a class $\mathbb{G}$ of grammars fixes the set $\mathbb{S}$ of possible substructures and the set $\mathbb{C}$ of possible contexts that may be contributed by parts of possible derivation trees. Each language L generated by a grammar in $\mathbb{G}$ acts as an arbiter that decides which context $C \in \mathbb{C}$ should "accept" which substructure $S \in \mathbb{S}$ (i.e., whether $C(S) \in L$).

Distributional learning algorithms come in two broad varieties. In the *primal* approach, the learner first extracts all substructures and all contexts that are contained in the input data, which is a finite set of elements of the target language L_*. The learner then collects all subsets of the extracted substructures whose cardinality does not exceed a certain fixed bound m. These subsets are used as nonterminal symbols of the hypothesized grammar. Out of all possible grammar rules that can be written using these nonterminals, the learner lists those that use operations that may be involved in the generation of the objects in the input data. In the final step of the algorithm, the learner tries to validate each of these rules with the membership oracle, which answers a query "$C(S) \in L_*$?" in constant time. If a rule has a set $\mathbf{S}$ of substructures on the left-hand side and

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 94–111, 2016.
DOI: 10.1007/978-3-662-53042-9_6

sets $\mathbf{S}_1, \ldots, \mathbf{S}_r$ on the right-hand side, and the grammatical operation associated with the rule is f, then the learner determines whether the following implication holds for all contexts C extracted from the input data:

$$C(S) \in L_* \text{ for all } S \in \mathbf{S} \text{ implies}$$
$$C(f(S_1, \ldots, S_n)) \in L_* \text{ for all } S_1 \in \mathbf{S}_1, \ldots, S_n \in \mathbf{S}_n. \quad (1)$$

The grammar conjectured by the learner includes only those rules that pass this test.

The idea of the rule validation is the following: It is dictated that the elements of the nonterminal $\mathbf{S}$ together *characterize* the set of all substructures that can be derived from $\mathbf{S}$ by the hypothesized grammar in the sense that every context $C \in \mathbb{C}$ that accepts all elements of $\mathbf{S}$ must accept all substructures derived from $\mathbf{S}$. Thus, only those rules that are consistent with this requirement are allowed in the hypothesized grammar. A remarkable property of the algorithm is that it successfully learns the language of every grammar in the given class $\mathbb{G}$ that has the *m-finite kernel property* in the sense that each nonterminal is characterized by a set of substructures of cardinality up to m.

In the *dual* approach to distributional learning, the role of contexts and substructures is switched. The learner uses as nonterminals subsets of the contexts extracted from the input data with cardinality $\leq m$, and uses the extracted substructures to validate candidate rules. The algorithm learns those languages that have a grammar with the *m-finite context property* in the sense that each nonterminal is characterized by a set of contexts of cardinality $\leq m$.

Whether each of these algorithms runs in polynomial time in the size of the input data D depends on several factors that are all determined by the grammar class $\mathbb{G}$. The foremost among them is the enumeration of the two sets

$$\mathbb{S}|_D = \{\, S \in \mathbb{S} \mid C(S) \in D \text{ for some } C \in \mathbb{C} \,\},$$
$$\mathbb{C}|_D = \{\, C \in \mathbb{C} \mid C(S) \in D \text{ for some } S \in \mathbb{S} \,\}.$$

There are two possible difficulties in enumerating each of these sets in polynomial time. First, the sheer number of elements of the set may be super-polynomial, in which case explicit enumeration of the set is not possible in polynomial time. Second, recognizing which substructure/context belongs to the set may be computationally costly. The second problem, even when it arises, can often be dealt with by replacing the set in question by a more easily recognizable superset without disrupting the working of the algorithm. The first problem is the more pressing one.

With all *linear* grammar formalisms to which distributional learning has been applied, neither of these two difficulties arise. When these formalisms are extended to allow nonlinearity in grammatical operations, however, the problem of super-polynomial cardinality hits hard. Thus, with *parallel multiple context-free grammars*, the nonlinear extension of *multiple context-free grammars* (successfully dealt with in [12]), the set $\mathbb{C}$ becomes a much larger set, even though $\mathbb{S}$ stays exactly the same. As a result, the cardinality of $\mathbb{C}|_D$ is no longer bounded by a polynomial. The situation with *IO context-free grammars*, the nonlinear

extension of the *simple context-free tree grammars* (treated in [9]), is even worse. Both of the sets $\mathbb{S}|_D$ and $\mathbb{C}|_D$ become super-polynomial in cardinality.

When only one of the two sets $\mathbb{S}|_D$ and $\mathbb{C}|_D$ is of super-polynomial cardinality, as is the case with PMCFGs, however, there is a way out of this plight [4]. The solution is to restrict the offending set by a certain property, parametrized by a natural number, so that its cardinality will be polynomial. The parametrized restriction leads to an increasing chain of subsets inside $\mathbb{S}$ or $\mathbb{C}$. In the case of PMCFGs, we get $\mathbb{C}_1 \subset \mathbb{C}_2 \subset \mathbb{C}_3 \subset \cdots \subset \mathbb{C} = \bigcup_k \mathbb{C}_k$, where $\mathbb{C}_k$ is the set of all possible contexts that satisfy the property with respect to the parameter k. The actual property used by [4] was a measure of nonlinearity of the context ("k-copying"), but this specific choice is not crucial for the correct working of the algorithm, as long as $\mathbb{C}_k|_D$ can be enumerated in polynomial time. The learning algorithm now has two parameters, m and k: the former is a bound on the cardinality of sets of contexts the learner uses as nonterminals as before, and the latter is a restriction on the kind of context allowed in these sets. The class of languages successfully learned by the algorithm includes the languages of all grammars in the target class that have the (k, m)-*finite context-property* in the sense that each nonterminal is characterized by a subset of $\mathbb{C}_k$ of cardinality $\leq m$.

This algorithm does not learn the class of all grammars with the m-finite context property, but a proper subset of it. Nevertheless, the parametrized restriction has a certain sense of naturalness, and the resulting learnable class properly extends the corresponding linear class, so the weaker result is interesting in its own right.

In this paper, we explore the connection between distributional learning and context/substructure enumerability in the general setting of *almost linear second-order abstract categorial grammars* generating trees [5–7] ("almost linear ACGs" for short). This class of grammars properly extends IO context-free tree grammars and is equivalent in tree generating power to *tree-valued attribute grammars* [1]. In fact, the expressive power of typed lambda calculus makes it possible to faithfully encode most known tree grammars within almost linear ACGs.

Like IO context-free tree grammars and unlike PMCFGs, almost linear ACGs in general do not allow polynomial-time enumerability either on the context side or on the substructure side. Only very special grammars do, and an interesting subclass of them consists of those grammars that allow only a bounded degree of nonlinearity in the contexts (or in the substructures). It is easily decidable whether a given ACG satisfies each of these properties. We show that both of the resulting classes of grammars indeed allow a kind of efficient distributional learning similar to that for PMCFGs.

2 Typed Lambda Terms and Almost Linear ACGs

2.1 Types and Typed Lambda Terms

We assume familiarity with the notion of a *simply typed* λ*-term* (à la Church) over a *higher-order signature* $\Sigma = (A_\Sigma, C_\Sigma, \tau_\Sigma)$, where A_Σ is the set of *atomic*

types, C_Σ is the set of constants, and τ_Σ is a function from C_Σ to types over A_Σ. We use standard abbreviations: $\alpha_1 \to \cdots \to \alpha_n \to p$ means $\alpha_1 \to (\cdots \to (\alpha_n \to p) \dots)$, and $\lambda x_1^{\alpha_1} \dots x_n^{\alpha_n}.MN_1 \dots N_m$ is short for $\lambda x_1^{\alpha_1}.\dots.\lambda x_n^{\alpha_n}.((\dots(MN_1)\dots)N_m)$. The *arity* of $\alpha = \alpha_1 \to \cdots \to \alpha_n \to p$ with $p \in A_\Sigma$ is $\mathrm{arity}(\alpha) = n$. We write $\beta^n \to p$ for the type $\beta \to \cdots \to \beta \to p$ of arity n.

We take for granted such notions as β- and η-*reduction*, β-*normal form*, and *linear* λ-*terms*. We write $\twoheadrightarrow_\beta$ and $\twoheadrightarrow_\eta$ for the relations of β- and η-reduction between λ-terms. Every typed λ-term has a β-normal form, unique up to renaming of bound variables, which we write as $|M|_\beta$.

The set $\mathrm{LNF}_X^\alpha(\Sigma)$ of λ-terms of type α in η-*long* β-*normal form* (with free variables from X) is defined inductively as follows:

- If $x^{\alpha_1 \to \cdots \to \alpha_n \to p} \in X$, $M_1 \in \mathrm{LNF}_X^{\alpha_1}(\Sigma), \dots, M_n \in \mathrm{LNF}_X^{\alpha_n}(\Sigma)$, and $p \in A_\Sigma$, then $x^{\alpha_1 \to \cdots \to \alpha_n \to p} M_1 \dots M_n \in \mathrm{LNF}_X^p(\Sigma)$.
- If $c \in C_\Sigma$, $\tau_\Sigma(c) = \alpha_1 \to \cdots \to \alpha_n \to p$, $p \in A_\Sigma$, and $M_1 \in \mathrm{LNF}_X^{\alpha_1}(\Sigma), \dots, M_n \in \mathrm{LNF}_X^{\alpha_n}(\Sigma)$, then $cM_1 \dots M_n \in \mathrm{LNF}_X^p(\Sigma)$.
- If $M \in \mathrm{LNF}_{X \cup \{x^\alpha\}}^\beta(\Sigma)$, then $\lambda x^\alpha.M \in \mathrm{LNF}_X^{\alpha \to \beta}(\Sigma)$.

We often suppress the superscript and/or subscript in $\mathrm{LNF}_X^\alpha(\Sigma)$. Note that $\mathrm{LNF}_\varnothing^\alpha(\Sigma)$ denotes the set of *closed* λ-terms of type α in η-long β-normal form. We note that if $M \in \mathrm{LNF}^{\alpha \to \beta}(\Sigma)$ and $N \in \mathrm{LNF}^\alpha(\Sigma)$, then $|MN|_\beta \in \mathrm{LNF}^\alpha(\Sigma)$.

Henceforth, we often suppress the type superscript on variables. This is just for brevity; each variable in a typed λ-term comes with a fixed type.

We use strings over $\{0, 1\}$ to refer to positions inside a λ-term or a type. We write ε for the empty string, and write $u \le v$ to mean u is a prefix of v. When $u = u'0^i$, we refer to u' as $u0^{-i}$.

The *shape* of a type α, written $[\alpha]$, is defined by

$$[p] = \{\varepsilon\} \text{ if } p \text{ is atomic}, \qquad [\alpha \to \beta] = \{\varepsilon\} \cup \{\, 1u \mid u \in [\alpha] \,\} \cup \{\, 0u \mid u \in [\beta] \,\}.$$

The elements of $[\alpha]$ are the *positions* of α. A position u is *positive* if its parity (i.e., the number of 1s in u modulo 2) is 0, and *negative* if its parity is 1. We write $[\alpha]^+$ and $[\alpha]^-$ for the set of positive and negative positions of α, respectively. A position u of α is a *subpremise* if $u = u'1$ for some u'. Such an occurrence is a *positive* (resp. *negative*) *subpremise* if it is positive (resp. negative). We write $[\alpha]^+_{\mathrm{sp}}$ (resp. $[\alpha]^-_{\mathrm{sp}}$) for the set of positive (resp. negative) subpremises of $[\alpha]$.

If $u \in [\alpha]$, the *subtype* of α occurring at u, written α/u, is defined by

$$\alpha/\varepsilon = \alpha, \quad (\alpha \to \beta)/0u = \beta/u, \quad (\alpha \to \beta)/1u = \alpha/u.$$

If $\alpha/u = \beta$, we say that β *occurs* at position u in α.

Given a λ-term M, the *shape* of M, written $[M]$, is defined by

$$\begin{aligned} [M] &= \{\varepsilon\} \quad \text{if } M \text{ is a variable or a constant,} \\ [MN] &= \{\varepsilon\} \cup \{\, 0u \mid u \in [M] \,\} \cup \{\, 1u \mid u \in [N] \,\}, \\ [\lambda x.M] &= \{\varepsilon\} \cup \{\, 0u \mid u \in [M] \,\}. \end{aligned}$$

The elements of $[M]$ are the *positions* of M.

If $u \in [M]$, the *subterm* of M occurring at u, written M/u, is defined by

$$M/\varepsilon = M, \quad (MN)/0u = M/u, \quad (MN)/1u = N/u, \quad (\lambda x.M)/0u = M/u.$$

When $N = M/u$, we sometimes call u an *occurrence of* N (in M).

When $v \in [M]$ but $v0 \notin [M]$, M/v is a variable or a constant. For each $u \in [M]$, we refer to the unique occurrence of a variable or constant in $[M]$ of the form $u0^k$ as the *head* of u (in M); we also call the variable or constant occurring at the head of u the *head* of M/u.

A position $v \in [M]$ *binds* a position $u \in [M]$ if M/u is a variable x and v is the longest prefix of u such that M/v is a λ-abstract of the form $\lambda x.N$. When v binds u in M, we write $v = b_M(u)$. When every occurrence in M of a λ-abstract is the binder of some position, M is called a *λI-term*.

Let $M \in \mathrm{LNF}^{\alpha}_{\mathscr{O}}(\Sigma)$. Note that an occurrence $v \in [M]$ of a variable or a constant of type β with $\mathrm{arity}(\beta) = n$ is always accompanied by n arguments, so that $v0^{-i}$ is defined for all $i \leq n$. The set of *replaceable* occurrences [2] of bound variables in M and the negative subpremise $\mathrm{nsp}_M(u)$ of α associated with such an occurrence u, are defined as follows:[1]

(i) If $b_M(u) = 0^{j-1}$ for some $j \geq 1$ (i.e., $b_M(u)$ is the jth of the leading λs of M), then u is replaceable and $\mathrm{nsp}_M(u) = 0^{j-1}1$.
(ii) If $b_M(u) = v0^{-i}10^{j-1}$ for some replaceable v and $i, j \geq 1$ (i.e., $b_M(u)$ is the jth of the leading λs of the ith argument of v), then u is replaceable and $\mathrm{nsp}_M(u) = \mathrm{nsp}_M(v)0^{i-1}10^{j-1}1$.

It is easy to see that the following conditions always hold:

– If u is a replaceable occurrence of a bound variable x^{β}, then $\beta = \alpha/\mathrm{nsp}_M(u)$.
– If M is a λI-term (in addition to belonging to $\mathrm{LNF}^{\alpha}_{\mathscr{O}}(\Sigma)$), then for every $v \in [\alpha]^{-}_{\mathrm{sp}}$, there exists a $u \in [M]$ such that $\mathrm{nsp}_M(u) = v$.

Example 1. Let

$$M = \lambda y_1^o y_2^{o\to(o\to(o\to o)\to o)\to o}.y_2(fy_1a)(\lambda y_3^o y_4^{o\to o}.f(y_4(fy_3y_1))(y_4(fy_3y_1))).$$

Then $M \in \mathrm{LNF}^{\alpha}_{\mathscr{O}}(\Delta)$, where Δ contains constants f, a of type $o \to o \to o$ and o, respectively, and

$$\alpha = \overset{1}{o} \to (o \to (\overbrace{\overset{01011}{o} \to \underbrace{(\underbrace{o \to o}_{01})}_{0101}}^{010101} \to o) \to o) \to o.$$

[1] A definition equivalent to $\mathrm{nsp}_M(u)$ for untyped λ-terms is in [2] (*access path*). The correspondence between these paths and negative subpremises for typed linear λ-terms is in [10].

- The bound variable y_1^o occurs in M at three positions, 000101, 001000111, 00100111, whose binder is ε. These positions are associated with the negative subpremise 1 in α.
- The bound variable $y_2^{o\to(o\to(o\to o)\to o)\to o}$ occurs in M at one position, 0000, whose binder is 0. This position is associated with the subpremise 01 in α.
- The bound variable y_3^o occurs in M at two positions, 0010001101 and 001001101, whose binder is 001. These positions are associated with the negative subpremise 0101.
- The bound variable $y_4^{o\to o}$ occurs in M at two positions, 00100010 and 0010010, whose binder is 0010. These positions are associated with the negative subpremise 010101.

2.2 Almost Linear Lambda Terms over a Tree Signature

Now we are going to assume that Δ is a tree signature; i.e., every constant of Δ is of type $o^r \to o$ for some $r \geq 0$, where o is the only atomic type of Δ. For a closed $M \in \mathrm{LNF}^\alpha_\varnothing(\Delta)$, every occurrence of a bound variable in M is replaceable.

A *tree* is an element of $\mathrm{LNF}^o_\varnothing(\Delta)$. A closed λ-term $M \in \mathrm{LNF}^{o^r\to o}_\varnothing(\Delta)$ is called a *tree context*. We say that a tree context $M = \lambda x_1 \dots x_r.N$ *matches* a tree T if there are trees $T_1, \dots, T_r$ such that $(\lambda x_1 \dots x_r.N)T_1 \dots T_r \twoheadrightarrow_\beta T$. We say that M is *contained* in T if it matches a subtree of T.

The notion of an *almost linear λ-term* was introduced by Kanazawa [5,7]. Briefly, a closed typed λ-term is almost linear if every occurrence of a λ-abstract $\lambda x^\alpha.N$ in it binds a unique occurrence of x^α, unless α is atomic, in which case it may bind more than one occurrence of x^α. Almost linear λ-terms share many of the properties of linear λ-terms; see [5–8] for details.

Almost linear λ-terms are typically not β-normal. For instance, $\lambda y^{o\to o}.(\lambda x^o.fxx)(yc)$, where f and c are constants of type $o \to o \to o$ and o, respectively, is almost linear, but its β-normal form, $\lambda y^{o\to o}.f(yc)(yc)$, is not. In this paper, we choose to deal with the η-long β-normal forms of almost linear λ-terms directly, rather than through their almost linear β-expanded forms.

We write $\mathrm{AL}^\alpha(\Delta)$ for the set of *closed* λ-terms in $\mathrm{LNF}^\alpha_\varnothing(\Delta)$ that β-expand to an almost linear λ-term. (The superscript is often omitted.) The following lemma, which we do not prove here, may be taken as the definition of $\mathrm{AL}^\alpha(\Delta)$ (see [7,8] for relevant properties of almost linear λ-terms):

Lemma 1. *Let M be a closed λI-term in $\mathrm{LNF}^\alpha_\varnothing(\Delta)$. Then $M \in \mathrm{AL}^\alpha(\Delta)$ if and only if the following conditions hold for all bound variable occurrences $u, v \in [M]$ such that $\mathrm{nsp}_M(u) = \mathrm{nsp}_M(v)$, where $n = \mathrm{arity}(\alpha/\mathrm{nsp}_M(u))$:*

(i) $\{\, w \mid u0^{-n}w \in [M] \,\} = \{\, w \mid v0^{-n}w \in [M] \,\}$.
(ii) *If $M/u0^{-n}w$ is a constant, then $M/u0^{-n}w = M/v0^{-n}w$.*
(iii) *If $M/u0^{-n}w$ is a variable, then $M/v0^{-n}w$ is also a variable and $\mathrm{nsp}_M(u0^{-n}w) = \mathrm{nsp}_M(v0^{-n}w)$.*

We call $M \in \mathrm{AL}^\alpha(\Delta)$ a *canonical writing* if for all bound variable occurrences u, v of M, $\mathrm{nsp}_M(u) = \mathrm{nsp}_M(v)$ implies $M/u = M/v$ and vice versa. For example,

$\lambda y_1^{(o\to o)\to o} y_2^{(o\to o)\to o}.f(y_1(\lambda z_1^o.z_1))(y_1(\lambda z_1^o.z_1))(y_2(\lambda z_2^o.z_2))$ is a canonical writing, whereas neither $\lambda y_1^{(o\to o)\to o} y_2^{(o\to o)\to o}.f(y_1(\lambda z_1^o.z_1))(y_1(\lambda z_2^o.z_2))(y_2(\lambda z_3^o.z_3))$ nor $\lambda y_1^{(o\to o)\to o} y_2^{(o\to o)\to o}.f(y_1(\lambda z_1^o.z_1))(y_1(\lambda z_1^o.z_1))(y_2(\lambda z_1^o.z_1))$ is.

Lemma 2. *For every $M \in \mathrm{AL}^\alpha(\Delta)$, there exists a canonical writing $M' \in \mathrm{AL}^\alpha(\Delta)$ such that $M' \equiv_\alpha M$.*

A *pure* λ-term is a λ-term that contains no constant. We write AL^α for the subset of $\mathrm{AL}^\alpha(\Delta)$ consisting of pure λ-terms. An important property of $\mathrm{AL}^\alpha(\Delta)$ that we heavily rely on in what follows is that every $M \in \mathrm{AL}^\alpha(\Delta)$ can be expressed in a unique way as an application $M^\circ M_1^\bullet \dots M_l^\bullet$ of a *pure* λ-term M° to a list of tree contexts $M_1^\bullet, \dots, M_l^\bullet$. We call the former the *container* of M and the latter its *stored tree contexts*. These λ-terms satisfy the following conditions:

1. $l \le |[\alpha]^+_{\mathrm{sp}}| + 1$,
2. $M_i^\bullet \in \mathrm{AL}^{o^{r_i}\to o}(\Delta)$ for some $r_i \le |[\alpha]^-_{\mathrm{sp}}|$ for each $i = 1, \dots, l$,
3. $M^\circ \in \mathrm{AL}^{(o^{r_1}\to o)\to\dots\to(o^{r_l}\to o)\to\alpha}$,
4. $M^\circ M_1^\bullet \dots M_l^\bullet \twoheadrightarrow_\beta M$.

The formal definition of this separation of $M \in \mathrm{AL}^\alpha(\Delta)$ into its container and stored tree contexts is rather complex, but the intuitive idea is quite simple. The stored tree contexts of M are the maximal tree contexts that can be discerned in the input λ-term.

Example 2. Consider the λ-term M of type $\alpha = o\to(o\to(o\to(o\to o)\to o)\to o)\to o$ in Example 1. This λ-term belongs to $\mathrm{AL}^\alpha(\Delta)$. Its container and stored tree contexts are:

$$M^\circ = \lambda z_1^{o\to o} z_2^{o\to o} z_3^{o\to o\to o} y_1^o y_2^{o\to(o\to(o\to o)\to o)\to o}.y_2(z_1 y_1)(\lambda y_3^o y_4^{o\to o}.z_2(y_4(z_3 y_3 y_1)),$$
$$M_1^\bullet = \lambda x_1.f x_1 a, \quad M_2^\bullet = \lambda x_1.f x_1 x_1, \quad M_3^\bullet = \lambda x_1 x_2.f x_1 x_2.$$

Here is the formal definition. Let $M \in \mathrm{AL}^\alpha(\Delta)$. We assume that M is canonical. Then $|[\alpha]^-_{\mathrm{sp}}|$ is exactly the number of distinct bound variables in M. Let $s_1, \dots, s_k$ list the elements of $[\alpha]^-_{\mathrm{sp}}$ in lexicographic order. Let $y_1, \dots, y_k$ be the corresponding list of bound variables in M, and let $n_i = \mathrm{arity}(\alpha/s_i)$ for each $i = 1, \dots, k$. Note that

$$\sum_{i=1}^k n_i \le |[\alpha]^+_{\mathrm{sp}}|.$$

The canonicity of M implies that every occurrence of y_i in M is accompanied by the exact same list of arguments $N_{i,1}, \dots, N_{i,n_i}$. The type of $N_{i,j}$ is $\alpha/s_i 0^{j-1}1$.

Let $x_1, \dots, x_k$ be fresh variables of type o. For each subterm N of M of type o, define $N^\blacktriangle$ by

$$(cT_1 \dots T_n)^\blacktriangle = cT_1^\blacktriangle \dots T_n^\blacktriangle, \quad (y_i N_{i,1} \dots N_{i,n_i})^\blacktriangle = x_i.$$

Let M' be the maximal subterm of M of atomic type; in other words, M' is the result of stripping M of its leading λs. Likewise, let $N'_{i,j}$ be the maximal subterm of $N_{i,j}$ of atomic type. Let $(M_1, \dots, M_l)$ be the sublist of

$$(M', N'_{1,1}, \dots, N'_{1,n_1}, \dots, N'_{k,1}, \dots, N'_{k,n_k})$$

consisting of the λ-terms whose head is a constant. (This list will contain duplicates if there exist i_1, j_1, i_2, j_2 such that $(i_1, j_1) \neq (i_2, j_2)$, $N'_{i_1,j_1} = N'_{i_2,j_2}$, and the head of this λ-term is a constant.) For each $i = 1, \ldots, l$, let $x_{m_{i,1}}, \ldots, x_{m_{i,r_i}}$ list the variables in $M_i^{\blacktriangle}$, in the order of their first appearances in $M_i^{\blacktriangle}$. Define

$$M_i^{\bullet} = \lambda x_{m_{i,1}} \ldots x_{m_{i,r_i}}.M_i^{\blacktriangle}, \quad \overrightarrow{M^{\bullet}} = (M_1^{\bullet}, \ldots, M_l^{\bullet}).$$

These are the stored tree contexts of M.

In order to define the container M°, we first define $N^{\triangle}$ by induction for each subterm N of M that is either (i) some M_i, (ii) a λ-term of atomic type whose head is a variable, or (iii) a λ-abstract. Let $z_1, \ldots, z_l$ be fresh variables of type $o^{r_1} \to o, \ldots, o^{r_l} \to o$, respectively[2].

$$M_i^{\triangle} = z_i(y_{m_{i,1}} N_{m_{i,1},1} \ldots N_{m_{i,1},n_{m_{i,1}}})^{\triangle} \ldots (y_{m_{i,r_i}} N_{m_{i,r_i},1} \ldots N_{m_{i,r_i},n_{m_{i,r_i}}})^{\triangle},$$
$$(y_i N_{i,1} \ldots N_{i,n_i})^{\triangle} = y_i N_{i,1}^{\triangle} \ldots N_{i,n_i}^{\triangle},$$
$$(\lambda y_i.N)^{\triangle} = \lambda y_i.N^{\triangle}.$$

Finally, define

$$M^{\circ} = \lambda z_1 \ldots z_l.M^{\triangle}.$$

Lemma 3. *$M^{\circ}, \overrightarrow{M^{\bullet}}$ satisfy the required conditions.*

Lemma 4. *Let $N \in \mathrm{AL}^{\alpha_1 \to \cdots \to \alpha_n \to \beta}(\Delta), M_i \in \mathrm{AL}^{\alpha_i}(\Delta)$ $(i = 1, \ldots, n)$, and $P = |NM_1 \ldots M_n|_{\beta} \in \mathrm{AL}^{\beta}(\Delta)$. Suppose*

$$\overrightarrow{M_i^{\bullet}} = ((M_i)_1^{\bullet}, \ldots, (M_i)_{l_i}^{\bullet}), \quad (M_i)_j^{\bullet} \in \mathrm{AL}^{o^{r_{i,j}} \to o}(\Delta), \qquad \overrightarrow{P^{\bullet}} = (P_1^{\bullet}, \ldots, P_m^{\bullet}).$$

For $i = 1, \ldots, n$ and $j = 1, \ldots, l_i$, let $c_{i,j}$ be a fresh constant of type $o^{r_{i,j}} \to o$. Let Δ' be the tree signature that extends Δ with the $c_{i,j}$, and let

$$Q = |N((M_1)^{\circ} c_{1,1} \ldots c_{1,l_1}) \ldots ((M_n)^{\circ} c_{n,1} \ldots c_{n,l_n})|_{\beta}.$$

We can compute the container and stored tree contexts of $Q \in \mathrm{AL}^{\beta}(\Delta')$ with respect to Δ'. Then we have

$$P^{\circ} = Q^{\circ}, \quad P_i^{\bullet} = |(Q_i^{\bullet})[c_{i,j} := (M_i)_j^{\bullet}]|_{\beta},$$

where $[c_{i,j} := (M_i)_j^{\bullet}]$ denotes the substitution of $(M_i)_j^{\bullet}$ for each $c_{i,j}$.

Definition 1. Let $M \in \mathrm{AL}^{\alpha}(\Delta)$.

(i) The *unlimited profile* of M is $\mathrm{prof}_{\infty}(M) = (M^{\circ}, w_1, \ldots, w_l)$, where l is the length of $\overrightarrow{M^{\bullet}} = (M_1^{\bullet}, \ldots, M_l^{\bullet})$ and for each i, w_i is the r_i-tuple of positive integers whose jth component is the number of occurrences of the jth bound variable in $M_i^{\bullet}$.

[2] When $M_i = M_j$ for some distinct i, j, the definition of $M_i^{\triangle}$ in fact depends on the subscript i.

(ii) For $k \geq 1$, the *k-threshold profile* of M, written $\mathrm{prof}_k(M)$, is just like its unlimited profile except that any number greater than k is replaced by ∞.

The *type* of the (unlimited or k-threshold) profile of M is α.

Example 3. The unlimited profile of the λ-term M from Example 1 is $\mathrm{prof}(M) = (M^\circ, (1), (2), (1,1))$. Its 1-threshold profile is $\mathrm{prof}_1(M) = (M^\circ, (1), (\infty), (1,1))$, and its k-threshold profile for $k \geq 2$ is the same as its unlimited profile.

Lemma 5. *For each $k \geq 1$ and type α, there are only finitely many k-threshold profiles of type α.*

We say that a k-threshold profile $(M^\circ, w_1, \ldots, w_l)$ is *k-bounded* if $w_i \in \{1, \ldots, k\}^{r_i}$ for $i = 1, \ldots, l$. A λ-term $M \in \mathrm{AL}(\Delta)$ that has a k-bounded profile is called *k-bounded*. We write $\mathrm{AL}_k^\alpha(\Delta)$ for the set of all k-bounded λ-terms in $\mathrm{AL}^\alpha(\Delta)$.

Note that $M \in \mathrm{AL}(\Delta)$ is linear if and only if it is 1-bounded and has a linear container.

Lemma 6. *Let $N \in \mathrm{AL}^{\alpha_1 \to \cdots \to \alpha_n \to \beta}(\Delta)$, and $M_i, M_i' \in \mathrm{AL}^{\alpha_i}(\Delta)$ for each $i = 1, \ldots, n$. Suppose that for each $i = 1, \ldots, n$, $\mathrm{prof}_k(M_i) = \mathrm{prof}_k(M_i')$. Then $\mathrm{prof}_k(|NM_1 \ldots M_n|_\beta) = \mathrm{prof}_k(|NM_1' \ldots M_n'|_\beta)$.*

The above lemma justifies the notation $N\pi_1 \ldots \pi_n$ for $\mathrm{prof}_k(|NM_1 \ldots M_n|_\beta)$ with $\mathrm{prof}_k(M_i) = \pi_i$, when k is understood from context. When $N = \lambda x_1 \ldots x_n.Q$, we may also write $Q[x_1 := \pi_1, \ldots, x_n := \pi_n]$ for $N\pi_1 \ldots \pi_n$. In this way, we can freely write profiles in expressions that look like λ-terms, like $\lambda x.\pi_1(Mx\pi_2)$.

Lemma 7. *Given a λ-term $N \in \mathrm{AL}^{\alpha_1 \to \cdots \to \alpha_n \to \beta}(\Delta)$ and k-threshold profiles $\pi_1, \ldots, \pi_n$ of type $\alpha_1, \ldots, \alpha_n$, respectively, the k-threshold profile $N\pi_1 \ldots \pi_n$ can be computed in polynomial time.*

In what follows, we often speak of "profiles" to mean k-threshold profiles, letting the context determine the value of k.

2.3 Almost Linear Second-Order ACGs on Trees

A (tree-generating) *almost linear second-order ACG* $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$ consists of a second-order signature Σ (*abstract vocabulary*), a tree signature Δ (*object vocabulary*), a set $\mathscr{I} \subseteq A_\Sigma$ of *distinguished types*, and a *higher-order homomorphism* $\mathcal{H}$ that maps each atomic type $p \in A_\Sigma$ to a type $\mathcal{H}(p)$ over A_Δ and each constant $c \in C_\Sigma$ to its *object realization* $\mathcal{H}(c) \in \mathrm{AL}^{\mathcal{H}(\tau_\Delta(c))}(\Delta)$. It is required that the image of $\mathscr{I}$ under $\mathcal{H}$ is $\{o\}$. That Σ is second-order means that for every $c \in C_\Sigma$, its type $\tau_\Sigma(c)$ is of the form $p_1 \to \cdots \to p_n \to q$; thus, any λ-term in $\mathrm{LNF}_\varnothing^p(\Sigma)$ for $p \in A_\Sigma$ has the form of a tree. A closed *abstract term* $P \in \mathrm{LNF}_\varnothing^\alpha(\Sigma)$ is homomorphically mapped by $\mathcal{H}$ to its object realization $|\mathcal{H}(P)|_\beta \in \mathrm{AL}^{\mathcal{H}(\alpha)}(\Delta)$. For $p \in A_\Sigma$, we write $\mathcal{S}(\mathscr{G}, p)$ for $\{\, |\mathcal{H}(P)|_\beta \mid P \in \mathrm{LNF}_\varnothing^p(\Sigma) \,\}$ and

$\mathcal{C}(\mathscr{G}, p)$ for $\{\, |\mathcal{H}(Q)|_\beta \mid Q$ is a closed linear λ-term in $\mathrm{LNF}^{p\to s}_{\varnothing}(\Sigma)$ for some $s \in \mathscr{I}\,\}$. The elements of these sets are *substructures* and *contexts* of $\mathscr{G}$, respectively. The tree language generated by $\mathscr{G}$ is $\mathcal{O}(\mathscr{G}) = \bigcup_{s\in\mathscr{I}} \mathcal{S}(\mathscr{G}, s)$.

An abstract constant $c \in C_\Sigma$ together with its type $\tau(c)$ and its object realization $\mathcal{H}(c)$ corresponds to a rule in more traditional grammar formalisms. An abstract atomic type $p \in A_\Sigma$ corresponds to a nonterminal. We say that $\mathscr{G}$ is *rule-k-bounded* if $\mathcal{H}(c)$ is k-bounded for every abstract constant $c \in C_\Sigma$.

Definition 2. Let $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$ be a tree-generating almost linear second-order ACG.

(i) We say that $\mathscr{G}$ is *substructure-k-bounded* if $\mathcal{S}(\mathscr{G}, p) \subseteq \mathrm{AL}_k^{\mathcal{H}(p)}(\Delta)$ for all atomic types $p \in A_\Sigma$.

(ii) We say that $\mathscr{G}$ is *context-k-bounded* if $\mathcal{C}(\mathscr{G}, p) \subseteq \mathrm{AL}_k^{\mathcal{H}(p)\to o}(\Delta)$ for all atomic types $p \in A_\Sigma$.

The set of possible k-threshold profiles of elements of $\mathcal{S}(\mathscr{G}, p)$ or $\mathcal{C}(\mathscr{G}, p)$ can easily be computed thanks to Lemmas 5 and 6, so substructure-k-boundedness and context-k-boundedness are both decidable properties of almost linear second-order ACGs. Conversely, one can design a substructure-k-bounded almost linear ACG by first assigning to each $p \in A_\Sigma$ a possible profile set Π_p consisting of profiles of type $\mathcal{H}(p)$; then, as the realization $\mathcal{H}(c)$ of a constant c of type $p_1 \to \cdots \to p_n \to q$, we admit only λ-terms in $\mathrm{AL}_k^{\mathcal{H}(p_1\to\cdots\to p_n\to q)}(\Delta)$ that satisfy

$$\mathcal{H}(c)\Pi_{p_1} \dots \Pi_{p_n} \subseteq \Pi_q, \tag{2}$$

where $M\Pi_1 \dots \Pi_n = \{\, M\pi_1 \dots \pi_n \mid \pi_i \in \Pi_i \ (i = 1, \dots, n)\,\}$. To construct a context-k-bounded almost linear ACG, we need to assign a possible context profile set Ξ_p in addition to Π_p to each $p \in A_\Sigma$. The realization $\mathcal{H}(c)$ must satisfy

$$\lambda x.\Xi_q(\mathcal{H}(c)\Pi_{p_1} \dots \Pi_{p_{i-1}} x \Pi_{p_{i+1}} \dots \Pi_{p_n}) \subseteq \Xi_{p_i} \tag{3}$$

for all $i = 1, \dots, n$, in addition to (2). Note that (2) and (3) are "local" properties of rules of ACGs. Instead of Definition 2, one may take this local constraint as a definition of substrucure/context-k-bounded almost linear ACGs.

Example 4. Let $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$, where $A_\Sigma = \{p_1, p_2, s\}$, $C_\Sigma = \{a, b, c_1, c_2, d_1, d_2\}$, $\tau_\Sigma(a) = p_1 \to s$, $\tau_\Sigma(b) = p_2 \to p_1$, $\tau_\Sigma(c_i) = p_i \to p_i$, $\tau_\Sigma(d_i) = p_i$, $A_\Delta = \{o\}$, $C_\Delta = \{e, f\}$, $\tau_\Delta(f) = o \to o \to o$, $\tau_\Delta(e) = o$, $\mathscr{I} = \{s\}$, $\mathcal{H}(p_i) = (o \to o) \to o \to o$, $\mathcal{H}(s) = o$ and

$$\begin{aligned}
\mathcal{H}(a) &= \lambda x^{(o\to o)\to o\to o}.x(\lambda z^o.z)e,\\
\mathcal{H}(b) &= \lambda x^{(o\to o)\to o\to o} y^{o\to o} z^o.x(\lambda w^o.y(fww))z,\\
\mathcal{H}(c_i) &= \lambda x^{(o\to o)\to o\to o} y^{o\to o} z^o.x(\lambda w^o.yw)(fzz),\\
\mathcal{H}(d_i) &= \lambda y^{o\to o} z^o.y(fzz)\,.
\end{aligned}$$

This grammar is rule-2-bounded and generates the set of perfect binary trees of height ≥ 1. We have, for example, $\mathcal{H}(b(c_2 d_2)) \in \mathcal{S}(\mathscr{G}, p_1)$ and $\mathcal{H}(\lambda x^{p_2}.a(c_1(b(c_2 x)))) \in \mathcal{C}(\mathscr{G}, p_2)$, and

$$|\mathcal{H}(b(c_2 d_2))|_\beta = \lambda y^{o\to o} z^o.y(f(f(fzz)(fzz))(f(fzz)(fzz))),$$
$$|\mathcal{H}(\lambda x^{p_2}.a(c_1(b(c_2 x))))|_\beta = \lambda x^{(o\to o)\to o\to o}.x(\lambda z.fzz)(f(fee)(fee)).$$

One can see

$$\mathrm{prof}_\infty(\mathcal{S}(\mathscr{G}, p_1)) = \mathrm{prof}_\infty(\mathcal{S}(\mathscr{G}, p_2)) = \{\, (\lambda {z_1}^{o\to o} y^{o\to o} w^o.y(z_1 w), (2^n)) \mid n \geq 1 \,\},$$

and

$$\mathrm{prof}_\infty(\mathcal{C}(\mathscr{G}, p_1)) = \{(\lambda {z_1}^o x^{(o\to o)\to o\to o}.x(\lambda w^o.w) z_1, ())\},$$
$$\mathrm{prof}_\infty(\mathcal{C}(\mathscr{G}, p_2)) = \{(\lambda {z_1}^o x^{(o\to o)\to o\to o}.x(\lambda w^o.w) z_1, ()),$$
$$(\lambda {z_1}^{o\to o} {z_2}^o x^{(o\to o)\to o\to o}.x(\lambda w^o.z_1 w) z_2, (2), ())\}.$$

The grammar is context-2-bounded, but not substructure-k-bounded for any k. If a new constant a' of type $p_1 \to s$ with $\mathcal{H}(a') = \lambda x^{(o\to o)\to o\to o}.x(\lambda z^o.fzz)e$ is added to $\mathscr{G}$, the grammar is not context-2-bounded any more, since $|\mathcal{H}(\lambda x^{p_2}.a'(bx))|_\beta = \lambda x^{(o\to o)\to o\to o}.x(\lambda z^o.f(fzz)(fzz))e \in \mathcal{C}(\mathscr{G}, p_2)$.

3 Extraction of Tree Contexts from Trees

We say that $M \in \mathrm{AL}^\alpha(\Delta)$ is *contained* in a tree T if there is an $N \in \mathrm{AL}^{\alpha\to o}(\Delta)$ such that $NM \twoheadrightarrow_\beta T$. The problem of extracting λ-terms in $\mathrm{AL}^\alpha(\Delta)$ contained in a given tree reduces to the problem of extracting tree contexts from trees.

Explicitly enumerating all tree contexts of type $o^r \to o$ is clearly intractable. A perfect binary tree with n leaves (labeled by the same constant) contains more than 2^n tree contexts of type $o \to o$.

It is easy to explicitly enumerate all tree contexts of type $o^r \to o$ that are *k-copying* in the sense that each bound variable occurs at most k times. (Just pick at most $rk + 1$ nodes to determine such a tree context.) Hence it is easy to explicitly enumerate all $M \in \mathrm{AL}^\alpha_k(\Delta)$ whose stored tree contexts (which are all k-copying) are contained in a given tree. (Recall that there is a fixed finite set of candidate containers for each α.) Not all these λ-terms are themselves contained in T, but it is harmless and simpler to list them all than to enumerate exactly those λ-terms $M \in \mathrm{AL}^\alpha_k(\Delta)$ for which there is an $N \in \mathrm{AL}^{\alpha\to o}(\Delta)$ (which may not be k-bounded) such that $MN \twoheadrightarrow_\beta T$.

We consider distributional learners for tree-generating almost linear second-order ACGs who are capable of extracting k-copying tree contexts from trees. Such a learner conjectures rule-k-bounded almost linear ACGs, and use only k-bounded substructures and k-bounded contexts in order to form hypotheses.

4 Distributional Learning of One-Side k-bounded ACGs

We present two distributional learning algorithms, a primal one for the context-k-bounded almost linear ACGs, and a dual one for the substructure-k-bounded almost linear ACGs.

In distributional learning, we often have to fix certain parameters that restrict the class $\mathbb{G}$ of grammars available to the learner as possible hypotheses, in order to make the universal membership problem solvable in polynomial time. This is necessary since the learner needs to check whether the previous conjecture generates all the positive examples received so far, including the current one. In the case of almost linear ACGs, the parameters are the maximal arity n of the type of abstract constants and the finite set Ω of the possible object images of abstract atomic types. When these parameters are fixed, the universal membership problem "$T \in \mathcal{O}(\mathscr{G})$?" is in P [7].

In addition to these two parameters, we also fix a positive integer k so that any hypothesized grammar is rule-k-bounded, for the reason explained in the previous section. The hypothesis space for our learners is thus determined by three parameters, Ω, n, k. We write $\mathbb{G}(\Omega, n, k)$ for the class of grammars determined by these parameters.

In what follows, we often use sets of profiles or λ-terms inside expressions that look like λ-terms, as we did in (2) and (3) in Sect. 2.3.

4.1 Learning Context-k-bounded ACGs with the Finite Kernel Property

For $\mathbf{T} \subseteq \mathrm{LNF}^o_{\varnothing}(\Delta)$ and $\mathbf{R} \subseteq \mathrm{AL}^{\alpha}(\Delta)$, we define the *$k$-bounded context set of* $\mathbf{R}$ *with respect to* $\mathbf{T}$ by

$$\mathrm{Con}_k(\mathbf{T}|\mathbf{R}) = \{\, Q \in \mathrm{AL}_k^{\alpha\to o}(\Delta) \mid |QR|_\beta \in \mathbf{T} \text{ for all } R \in \mathbf{R} \,\}.$$

Definition 3. A context-k-bounded ACG $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$ is said to have the *profile-insensitive* (k,m)*-finite kernel property* if for every abstract atomic type $p \in A_\Sigma$, there is a nonempty set $\mathbf{S}_p \subseteq \mathcal{S}(\mathscr{G}, p) \cap \mathrm{AL}_k^{\mathcal{H}(p)}(\Delta)$ such that $|\mathbf{S}_p| \leq m$ and

$$\mathrm{Con}_k(\mathcal{O}(\mathscr{G})|\mathbf{S}_p) = \mathrm{Con}_k(\mathcal{O}(\mathscr{G})|\mathcal{S}(\mathscr{G}, p)).$$

This may be thought of as a primal analogue of the notion of (k,m)-FCP in [4] for the present case. It turns out, however, designing a distributional learning algorithm targeting grammars satisfying this definition is neither elegant nor quite as straightforward as existing distributional algorithms. One reason is that simply validating hypothesized rules against k-bounded contexts (see (1) in Sect. 1) does not produce a context-k-bounded grammar. Recall that to construct a context-k-bounded grammar, we must fix an assignment of an admissible substructure profile set $\mathit{\Pi}_p$ and an admissible context profile set $\mathit{\Xi}_p$ to each atomic type p which restricts the object realizations of abstract constants of each type.

We let our learning algorithm use such an assignment together with finite sets of k-bounded substructures in constructing grammar rules, and make the validation of rules sensitive to the context profile set assigned to the "left-hand side" nonterminal. This naturally leads to the following definition:

Definition 4. A context-k-bounded ACG $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$ is said to have the *profile-sensitive* (k,m)*-finite kernel property* ((k,m)-FKP$_{\mathrm{prof}}$) if for every abstract atomic type $p \in A_\Sigma$, there is a nonempty set $\mathbf{S}_p \subseteq \mathcal{S}(\mathscr{G}, p) \cap \mathrm{AL}_k^{\mathcal{H}(p)}(\Delta)$ such that $|\mathbf{S}_p| \leq m$ and

$$\mathrm{Con}_k(\mathcal{O}(\mathscr{G})|\mathbf{S}_p) \cap \mathrm{prof}_k^{-1}(\Xi) = \mathrm{Con}_k(\mathcal{O}(\mathscr{G})|\mathcal{S}(\mathscr{G}, p)) \cap \mathrm{prof}_k^{-1}(\Xi)\,, \tag{4}$$

where $\Xi = \mathrm{prof}_k(\mathcal{C}(\mathscr{G}, p))$. Such a set $\mathbf{S}_p$ is called a *characterizing substructure set of* p.

Clearly, if a context-k-bounded grammar satisfies Definition 3, then it satisfies the (k,m)-FKP$_{\mathrm{prof}}$, so the class of grammars with (k,m)-FKP$_{\mathrm{prof}}$ is broader than the class given by Definition 3. The notion of (k,m)-FKP$_{\mathrm{prof}}$ is also monotone in k in the sense that (4) implies

$$\mathrm{Con}_{k+1}(\mathcal{O}(\mathscr{G})|\mathbf{S}_p) \cap \mathrm{prof}_{k+1}^{-1}(\Xi') = \mathrm{Con}_{k+1}(\mathcal{O}(\mathscr{G})|\mathcal{S}(\mathscr{G}, p)) \cap \mathrm{prof}_{k+1}^{-1}(\Xi')\,,$$

where $\Xi' = \mathrm{prof}_{k+1}(\mathcal{C}(\mathscr{G}, p)) = \mathrm{prof}_k(\mathcal{C}(\mathscr{G}, p))$, as long as $\mathscr{G}$ is context-k-bounded. This means that as we increase the parameter k, the class of grammars satisfying (k,m)-FKP$_{\mathrm{prof}}$ monotonically increases. This is another advantage of Definition 4 over Definition 3.

The polynomial enumerability of the k-bounded λ-terms makes an efficient primal distributional learner possible for the class of context-k-bounded grammars in $\mathbb{G}(\Omega, n, k)$ with the (k,m)-FKP$_{\mathrm{prof}}$.

Algorithm. Hereafter we fix a learning target $\mathbf{T}_* \subseteq \mathrm{LNF}^o_\varnothing(\Delta)$ which is generated by $\mathscr{G}_* = (\Sigma, \Delta, \mathcal{H}, \mathscr{I}) \in \mathbb{G}(\Omega, n, k)$ with the (k,m)-FKP$_{\mathrm{prof}}$. We write $\mathbf{S}^{[\Xi]} = \mathrm{Con}_k(\mathbf{T}_*|\mathbf{S}) \cap \mathrm{prof}_k^{-1}(\Xi)$ for a k-bounded profile set Ξ.

For a tree $T \in \mathrm{LNF}^o_\varnothing(\Delta)$, let $\mathrm{Ext}_k^\alpha(T) = \{\, M \in \mathrm{AL}_k^\alpha(\Delta) \mid \overrightarrow{M^\bullet}$ are contained in $T \,\}$. Define

$$\mathrm{Sub}_k^\Omega(\mathbf{D}) = \bigcup\{\, \mathrm{Ext}_k^\alpha(T) \mid T \in \mathbf{D},\, \alpha \in \Omega \,\},$$

$$\mathrm{Glue}_k^{\Omega,n}(\mathbf{D}) = \bigcup\{\, \mathrm{Ext}_k^{\alpha_1 \to \cdots \to \alpha_j \to \alpha_0}(T) \mid T \in \mathbf{D},\, \alpha_i \in \Omega \text{ for } i = 1, \ldots, j \text{ and } j \leq n \,\},$$

$$\mathrm{Con}_k^\Omega(\mathbf{D}) = \bigcup\{\, \mathrm{Ext}_k^{\alpha \to o}(T) \mid T \in \mathbf{D},\, \alpha \in \Omega \,\}.$$

It is easy to see that $\mathcal{H}(c) \in \mathrm{Glue}_k^{\Omega,n}(\mathbf{T}_*)$ for all $c \in C_\Sigma$.

Our learner (Algorithm 1) constructs a context-k-bounded ACG $\hat{\mathscr{G}} = \mathcal{G}(\mathbf{K}, \mathbf{B}, \mathbf{F}) = (\Gamma, \Delta, \mathcal{J}, \mathscr{J})$ from three sets $\mathbf{K} \subseteq \mathrm{Sub}_k^\Omega(\mathbf{D})$, $\mathbf{B} \subseteq \mathrm{Glue}_k^{\Omega,n}(\mathbf{D})$ and $\mathbf{F} \subseteq \mathrm{Con}_k^\Omega(\mathbf{D})$, where $\mathbf{D}$ is a finite set of positive examples given to the

Algorithm 1. Learning ACGs in $\mathbb{G}(\Omega, n, k)$ with the (k, m)-FKP$_{\text{prof}}$.

Data: A positive presentation $T_1, T_2, \ldots$ of $\mathbf{T}_*$; membership oracle on $\mathbf{T}_*$;
Result: A sequence of ACGs $\mathscr{G}_1, \mathscr{G}_2, \ldots$;
let $\mathbf{D} := \mathbf{K} := \mathbf{B} := \mathbf{F} := \varnothing$; $\hat{\mathscr{G}} := \mathcal{G}(\mathbf{K}, \mathbf{B}, \mathbf{F})$;
for $i = 1, 2, \ldots$ **do**
  let $\mathbf{D} := \mathbf{D} \cup \{T_i\}$; $\mathbf{F} := \mathrm{Con}_k^{\Omega}(\mathbf{D})$;
  if $\mathbf{D} \not\subseteq \mathcal{O}(\hat{\mathscr{G}})$ **then**
    let $\mathbf{B} := \mathrm{Glue}_k^{\Omega,n}(\mathbf{D})$;
    let $\mathbf{K} := \mathrm{Sub}_k^{\Omega}(\mathbf{D})$;
  end if
  output $\hat{\mathscr{G}} = \mathcal{G}(\mathbf{K}, \mathbf{B}, \mathbf{F})$ as $\mathscr{G}_i$;
end for

learner. As with previous primal learning algorithms, whenever we get a positive example that is not generated by our current conjecture, we expand $\mathbf{K}$ and $\mathbf{B}$, while in order to suppress incorrect rules, we keep expanding $\mathbf{F}$.

Each abstract atomic type of our grammar is a triple of a subset of $\mathbf{K}$, a k-threshold profile set, and a k-bounded profile set:

$$A_\Gamma = \{\, [\![\mathbf{S}, \Pi, \Xi]\!] \mid \mathbf{S} \subseteq \mathbf{K} \cap \mathrm{prof}_k^{-1}(\Pi) \text{ with } 1 \le |\mathbf{S}| \le m, \text{ where for some } \alpha \in \Omega, \\ \Pi \text{ is a set of } k\text{-threshold profiles of type } \alpha \text{ and} \\ \Xi \text{ is a set of } k\text{-bounded profiles of type } \alpha \to o \,\}.$$

We have $|A_\Gamma| \le 2^{2\ell}|\mathbf{K}|^m$, where ℓ is the total number of profiles of relevant types, which is a constant.

The set of distinguished types is defined as

$$\mathscr{J} = \{\, [\![\, \mathbf{S},\, \{(\lambda z^o.z)\},\, \{(\lambda y^o.y)\} \,]\!] \in A_\Gamma \mid \mathbf{S} \subseteq \mathbf{T}_* \,\},$$

which is determined by membership queries. Define $\mathcal{J}([\![\mathbf{S}, \Pi, \Xi]\!])$ to be the type of the profiles in Π.

We have an abstract constant $d \in C_\Gamma$ such that

$$\tau_\Gamma(d) = [\![\mathbf{S}_1, \Pi_1, \Xi_1]\!] \to \cdots \to [\![\mathbf{S}_j, \Pi_j, \Xi_j]\!] \to [\![\mathbf{S}_0, \Pi_0, \Xi_0]\!] \text{ with } j \le n\,,$$
$$\mathcal{J}(d) = R \in \mathbf{B}\,,$$

if

- $R\Pi_1 \ldots \Pi_j \subseteq \Pi_0$,
- $\lambda x.\Xi_0(R\Pi_1 \ldots \Pi_{i-1} x \Pi_{i+1} \ldots \Pi_j) \subseteq \Xi_i$ for $i = 1, \ldots, j$,
- $|Q(RS_1 \ldots S_j)|_\beta \in \mathbf{T}_*$ for all $Q \in \mathbf{S}_0^{[\Xi_0]} \cap \mathbf{F}$ and $S_i \in \mathbf{S}_i$ for $i = 1, \ldots, j$.

The last condition is checked with the aid of the membership oracle.

Lemma 8. *We have* $\mathrm{prof}_k(N) \in \Pi$ *for all* $N \in \mathcal{S}(\mathscr{G}, [\![\mathbf{S}, \Pi, \Xi]\!])$, *and* $\mathrm{prof}_k(M) \in \Xi$ *for all* $M \in \mathcal{C}(\mathscr{G}, [\![\mathbf{S}, \Pi, \Xi]\!])$. *The grammar* $\mathcal{G}(\mathbf{K}, \mathbf{B}, \mathbf{F})$ *is context-*k*-bounded.*

Lemma 9.
If $\mathbf{K} \subseteq \mathbf{K}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})) \subseteq \mathcal{O}(\mathcal{G}(\mathbf{K}',\mathbf{B},\mathbf{F}))$.
If $\mathbf{B} \subseteq \mathbf{B}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})) \subseteq \mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B}',\mathbf{F}))$.
If $\mathbf{F} \subseteq \mathbf{F}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})) \supseteq \mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F}'))$.

Lemma 10. *Let* $\mathbf{S}_p$ *be a characterizing set of each atomic type* $p \in A_\Sigma$ *of the target grammar* $\mathscr{G}_*$*. Then* $\mathbf{S}_p \subseteq \mathrm{Sub}_k^\Omega(\mathbf{T}_*)$*. Moreover, if* $\mathbf{S}_p \subseteq \mathbf{K}$ *for all* $p \in A_\Sigma$ *and* $\mathcal{H}(c) \in \mathbf{B}$ *for all* $c \in C_\Sigma$*, then* $\mathbf{T}_* \subseteq \mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F}))$ *for any* $\mathbf{F}$.

We say that an abstract constant d of type $[\![\mathbf{S}_1, \Pi_1, \Xi_1]\!] \to \cdots \to [\![\mathbf{S}_j, \Pi_j, \Xi_j]\!] \to [\![\mathbf{S}_0, \Pi_0, \Xi_0]\!]$ is *invalid* if $|Q(\mathcal{J}(c)S_1 \dots S_j)|_\beta \notin \mathbf{T}_*$ for some $Q \in \mathbf{S}_0^{[\Xi_0]}$ and $S_i \in \mathbf{S}_i$.

Lemma 11. *For every* $\mathbf{K}$ *and* $\mathbf{B}$*, there is a finite set* $\mathbf{F} \subseteq \mathrm{Con}_k^\Omega(\mathbf{T}_*)$ *of cardinality* $|\mathbf{B}||A_\Gamma|^{n+1}$ *such that* $\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})$ *has no invalid constant.*

Lemma 12. *If* $\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})$ *has no invalid constant, then* $\mathcal{O}(\mathcal{G}(\mathbf{K},\mathbf{B},\mathbf{F})) \subseteq \mathbf{T}_*$.

Theorem 1. *Algorithm 1 successfully learns all grammars in* $\mathbb{G}(\Omega, n, k)$ *with the* (k,m)-$\mathrm{FKP}_{\mathrm{prof}}$.

We remark on the efficiency of our algorithm. It is easy to see that the description sizes of $\mathbf{K}$ and $\mathbf{B}$ are polynomially bounded by that of $\mathbf{D}$, and so is that of Γ. We need at most a polynomial number of membership queries to construct a grammar. Thus Algorithm 1 updates its conjecture in polynomial time in $\|\mathbf{D}\|$. Moreover, we do not need too much data. To make $\mathbf{K}$ and $\mathbf{B}$ satisfy the condition of Lemma 10, $m|A_\Sigma|+|C_\Sigma|$ examples are enough. To remove invalid constants, polynomially many contexts are enough by Lemma 11.

4.2 Learning Substructure-k-bounded ACGs with the Finite Context Property

For sets $\mathbf{T} \subseteq \mathrm{LNF}_\varnothing^o(\Delta)$ and $\mathbf{Q} \subseteq \mathrm{AL}_k^{\alpha\to o}(\Delta)$, we define the *$k$-bounded substructure set of* $\mathbf{Q}$ *with respect to* $\mathbf{T}$ by

$$\mathrm{Sub}_k(\mathbf{T}|\mathbf{Q}) = \{\, R \in \mathrm{AL}_k^\alpha(\Delta) \mid |QR|_\beta \in \mathbf{T} \text{ for all } Q \in \mathbf{Q} \,\}.$$

Again, we target grammars that satisfy a property sensitive to profile sets assigned to nonterminals:

Definition 5. A substructure-k-bounded ACG $\mathscr{G} = (\Sigma, \Delta, \mathcal{H}, \mathscr{I})$ is said to have *the profile-sensitive (k,m)-finite context property* ((k,m)-$\mathrm{FCP}_{\mathrm{prof}}$) if for every abstract atomic type $p \in A_\Sigma$, there is a nonempty set $\mathbf{Q}_p \subseteq \mathcal{C}(\mathscr{G}, p) \cap \mathrm{AL}_k^{\mathcal{H}(p)\to o}(\Delta)$ of k-bounded λ-terms such that $|\mathbf{Q}_p| \leq m$ and

$$\mathrm{Sub}_k(\mathcal{O}(\mathscr{G})|\mathbf{Q}_p) \cap \mathrm{prof}_k^{-1}(\Pi) = \mathcal{S}(\mathscr{G}, p)\,,$$

where $\Pi = \mathrm{prof}(\mathcal{S}(\mathscr{G}, p))$. We call $\mathbf{Q}_p$ a *characterizing context set* of p.

Algorithm. Our dual learner turns out to be considerably simpler than its primal cousin. While the primal learner uses two profile sets, the dual learner assigns just a single profile to each nonterminal. This corresponds to the fact that the context-profiles play no role in constructing a structure-k-bounded grammar and that the (k,m)-$\mathrm{FCP}_{\mathrm{prof}}$ is preserved under the normalization which converts a grammar into an equivalent one $\mathscr{G}'$ where $\mathrm{prof}_k(\mathcal{S}(\mathscr{G}',p))$ is a singleton for all abstract atomic types p of $\mathscr{G}'$, where it is not necessarily the case for the (k,m)-$\mathrm{FKP}_{\mathrm{prof}}$.

Hereafter we fix a learning target $\mathbf{T}_* \subseteq \mathrm{LNF}^o_\varnothing(\Delta)$ which is generated by $\mathscr{G}_* = (\Sigma,\Delta,\mathcal{H},\mathscr{I}) \in \mathbb{G}(\Omega,n,k)$ with the (k,m)-$\mathrm{FCP}_{\mathrm{prof}}$. We write $\mathbf{Q}^{[\pi]} = \mathrm{Sub}_k(\mathbf{T}_*|\mathbf{Q}) \cap \mathrm{prof}_k^{-1}(\pi)$ for a k-bounded profile π.

Our learner (Algorithm 2) constructs a context-k-bounded ACG $\hat{\mathscr{G}} = \mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K}) = (\Gamma,\Delta,\mathcal{J},\mathscr{J})$ from three sets $\mathbf{F} \subseteq \mathrm{Con}^\Omega_k(\mathbf{D})$, $\mathbf{B} \subseteq \mathrm{Glue}^{\Omega,n}_k(\mathbf{D})$, and $\mathbf{K} \subseteq \mathrm{Sub}^\Omega_k(\mathbf{D})$, where $\mathbf{D}$ is a finite set of positive examples.

Algorithm 2. Learning ACGs in $\mathbb{G}(\Omega,n,k)$ with (k,m)-$\mathrm{FCP}_{\mathrm{prof}}$

Data: A positive presentation $T_1,T_2,\dots$ of $\mathbf{T}_*$; membership oracle on $\mathbf{T}_*$;
Result: A sequence of ACGs $\mathscr{G}_1,\mathscr{G}_2,\dots$;
let $\mathbf{D} := \mathbf{F} := \mathbf{B} := \mathbf{K} := \varnothing$; $\hat{\mathscr{G}} := \mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})$;
for $i = 1,2,\dots$ **do**
  let $\mathbf{D} := \mathbf{D} \cup \{T_i\}$; $\mathbf{K} := \mathrm{Sub}^\Omega_k(\mathbf{D})$;
  if $\mathbf{D} \not\subseteq \mathcal{O}(\hat{\mathscr{G}})$ **then**
    let $\mathbf{B} := \mathrm{Glue}^{\Omega,n}_k(\mathbf{D})$;
    let $\mathbf{F} := \mathrm{Con}^\Omega_k(\mathbf{D})$;
  end if
  output $\hat{\mathscr{G}} = \mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})$ as $\mathscr{G}_i$;
end for

Each abstract atomic type of our grammar is a pair of a finite subset of $\mathbf{F} \cap \mathrm{AL}^\alpha_k(\Delta)$ of cardinality at most m and a profile π whose type is α:

$$A_\Gamma = \{\, [\![\mathbf{Q},\pi]\!] \mid \pi \text{ is a } k\text{-bounded profile of type } \alpha \in \Omega,\ \mathbf{Q} \subseteq \mathbf{F} \cap \mathrm{AL}^{\alpha\to o}_k(\Delta) \text{ and } 1 \le |\mathbf{Q}| \le m \,\}.$$

We have $|A_\Gamma| \le |\mathbf{F}|^m \ell$ for ℓ the number of possible profiles. We have only one distinguished type:

$$\mathscr{J} = \{\, [\![\{\lambda y.y\},\ (\lambda z^o.z)]\!] \,\}.$$

We define $\mathcal{J}([\![\mathbf{Q},\pi]\!])$ to be the type of π.

We have an abstract constant $c \in C_\Gamma$ such that

$$\tau_\Gamma(c) = [\![\mathbf{Q}_1,\pi_1]\!] \to \cdots \to [\![\mathbf{Q}_j,\pi_j]\!] \to [\![\mathbf{Q}_0,\pi_0]\!] \text{ with } j \le n\,, \quad \mathcal{J}(c) = P \in \mathbf{B}\,,$$

if

– $\pi_0 = P\pi_1 \dots \pi_j$,

– $|Q(PS_1 \dots S_j)|_\beta \in \mathbf{T}_*$ for all $Q \in \mathbf{Q}_0$ and $S_i \in \mathbf{Q}_i^{[\pi_i]} \cap \mathbf{K}$.

The second clause is checked with the aid of the membership oracle. By the construction, $\mathrm{prof}(|\mathcal{J}(M)|_\beta) \in \pi$ for every $M \in \mathrm{LNF}_\varnothing^{[\![\mathbf{Q},\pi]\!]}(\Gamma)$. Thus the grammar $\hat{\mathscr{G}}$ is substructure-k-bounded.

Lemma 13.
If $\mathbf{F} \subseteq \mathbf{F}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})) \subseteq \mathcal{O}(\mathcal{G}(\mathbf{F}',\mathbf{B},\mathbf{K}))$.
If $\mathbf{B} \subseteq \mathbf{B}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})) \subseteq \mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B}',\mathbf{K}))$.
If $\mathbf{K} \subseteq \mathbf{K}'$*, then* $\mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})) \supseteq \mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K}'))$.

Lemma 14. *Let* $\mathbf{Q}_p$ *be a characterizing set of each atomic type* $p \in A_\Sigma$ *of the target grammar* $\mathscr{G}_*$*. Then* $\mathbf{Q}_p \subseteq \mathrm{Con}_k^\Omega(\mathbf{T}_*)$*. Moreover, if* $\mathbf{Q}_p \subseteq \mathbf{F}$ *for all* $p \in A_\Sigma$ *and* $\mathcal{H}(c) \in \mathbf{B}$ *for all* $c \in C_\Sigma$*, then* $\mathbf{T}_* \subseteq \mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K}))$ *for any* $\mathbf{K}$.

We say that an abstract constant c of type $[\![\mathbf{Q}_1,\pi_1]\!] \to \dots \to [\![\mathbf{Q}_j,\pi_j]\!] \to [\![\mathbf{Q}_0,\pi_0]\!]$ is *invalid* if $|Q(\mathcal{J}(c)S_1 \dots S_j)|_\beta \notin \mathbf{T}_*$ for some $S_i \in \mathbf{Q}_i^{[\pi_i]}$ and $Q \in \mathbf{Q}_0$.

Lemma 15. *For every* $\mathbf{F}$ *and* $\mathbf{B}$*, there is a finite set* $\mathbf{K} \subseteq \mathrm{Sub}_k^\Omega(\mathbf{T}_*)$ *of cardinality* $n|\mathbf{B}||A_\Gamma|^{n+1}$ *such that* $\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})$ *has no invalid constant.*

Lemma 16. *If* $\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})$ *has no invalid constant, then* $\mathcal{O}(\mathcal{G}(\mathbf{F},\mathbf{B},\mathbf{K})) \subseteq \mathbf{T}_*$.

Theorem 2. *Algorithm 2 successfully learns all grammars in* $\mathbb{G}(\Omega,n,k)$ *with the* (k,m)-$\mathrm{FCP}_{\mathrm{prof}}$.

A remark similar to the one on the efficiency of Algorithm 1 applies to Algorithm 2.

Acknowledgement. This work was supported in part by MEXT/JSPS KAKENHI (24106010, 26330013) and NII joint research project "Algorithmic Learning of Nonlinear Formalisms Based on Distributional Learning".

References

1. Bloem, R., Engelfriet, J.: A comparison of tree transductions defined by monadic second order logic and by attribute grammars. J. Comput. Syst. Sci. **61**, 1–50 (2000)
2. Böhm, C., Coppo, M., Dezani-Ciancaglini, M.: Termination tests inside λ-calculus. In: Salomaa, A., Steinby, M. (eds.) Automata, Languages and Programming. LNCS, vol. 52, pp. 95–110. Springer, Heidelberg (1977)
3. Clark, A.: Learning context free grammars with the syntactic concept lattice. In: Sempere and García [11], pp. 38–51
4. Clark, A., Yoshinaka, R.: Distributional learning of parallel multiple context-free grammars. Mach. Learn. **96**(1–2), 5–31 (2014). doi:10.1007/s10994-013-5403-2
5. Kanazawa, M.: Parsing and generation as datalog queries. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Prague, Czech Republic, pp. 176–183 (2007)

6. Kanazawa, M.: A lambda calculus characterization of MSO definable tree transductions (abstract). Bull. Symbolic Logic **15**(2), 250–251 (2009)
7. Kanazawa, M.: Parsing and generation as datalog query evaluation. IfColog J. Logics Their Appl. (to appear). http://research.nii.ac.jp/~kanazawa/publications/pagadqe.pdf
8. Kanazawa, M.: Almost affine lambda terms. In: Indrzejczak, A., Kaczmarek, J., Zawidzki, M. (eds.) Trends in Logic XIII, pp. 131–148. Łódź University Press, Łódź (2014)
9. Kasprzik, A., Yoshinaka, R.: Distributional learning of simple context-free tree grammars. In: Kivinen, J., Szepesvári, C., Ukkonen, E., Zeugmann, T. (eds.) ALT 2011. LNCS, vol. 6925, pp. 398–412. Springer, Heidelberg (2011)
10. Salvati, S.: Encoding second order string ACG with deterministic tree walking transducers. In: Wintner, S. (ed.) Proceedings of FG 2006: The 11th conference on Formal Grammar. FG Online Proceedings, pp. 143–156. CSLI Publications, Stanford (2007)
11. Sempere, J.M., García, P. (eds.): Grammatical Inference: Theoretical Results and Applications. Proceedings of 10th International Colloquium, ICGI 2010, Valencia, Spain, 13–16 September 2010. LNCS. Springer, Heidelberg (2010)
12. Yoshinaka, R.: Polynomial-time identification of multiple context-free languages from positive data and membership queries. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 230–244. Springer, Heidelberg (2010)
13. Yoshinaka, R.: Towards dual approaches for learning context-free grammars based on syntactic concept lattices. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 429–440. Springer, Heidelberg (2011)
14. Yoshinaka, R., Kanazawa, M.: Distributional learning of abstract categorial grammars. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 251–266. Springer, Heidelberg (2011)

Between the Event Calculus and Finite State Temporality

Derek Kelleher, Tim Fernando(✉), and Carl Vogel

Trinity College Dublin, Dublin 2, Ireland
kellehdt@tcd.ie, {Tim.Fernando,vogel}@cs.tcd.ie

Abstract. Event Calculus formulas dealing with instantaneous and continuous change are translated into regular languages interpreted relative to finite models. It is shown that a model over the real line for a restricted class of these Event Calculus formulas (relevant for natural language semantics) can be transformed into a finite partition of the real line, satisfying the regular languages. Van Lambalgen and Hamm's treatment of type coercion is reduced to changes in the alphabet from which the strings are formed.

Keywords: Event Calculus · Finite State Temporality · Type coercion

1 Introduction

An important application of the Event Calculus (EC), originally developed by Kowalski and Sergot [14], has been Lambalgen and Hamm's treatment of Event Semantics [15] (hereafter referred to as VLH). The EC concentrates on change in time-dependent properties, known as fluents, formalizing both instantaneous change (due to punctual events), and continuous change (under some force). The underlying model is taken to be a continuum, the real line, with a predicate (HoldsAt) interpreting fluents relative to points (real numbers). This is in contrast to the common practice since Bennett and Partee [3] of evaluating temporal propositions over intervals.

Another approach to event semantics, which uses finite-state methods, is Finite State Temporality (FST). In recent years FST has been shown to be applicable to a diverse range of linguistic phenomena, such as Dowty's Aktionsart [9]. Strings are taken from an alphabet made up of subsets of a finite set of fluents (Σ), shown visually as boxes containing the fluents. Intuitively, fluents in the same box hold at the same time, and fluents in subsequent boxes hold of subsequent times. For example, the event of John going from his home to the library could be represented as the string:

at(john,home)		at(john,library)	(1)

the empty box above symbolising that there is an interval of time between John being at home, and being at the library, with no commitment made to what

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 112–129, 2016.
DOI: 10.1007/978-3-662-53042-9_7

occurred during this interval. A fuller representation could add to that box fluents representing John's journey to the library, and split it into various boxes representing various stages of this journey.

It should be noted that while both the EC and FST talk of "fluents", there are some crucial differences between the two formalisms in their interpretation of this notion. In the EC, a fluent is a "time-dependent property" in that a fluent holding of an instant indicates that property holds of that instant. For example, if the fluent "building" holds of time t, then there is some building going on at time t. Events have a different status, being the entities that initiate or terminate the fluents (cause them to become true or false), and temporal instants form another ontological category. In FST, "fluents" correspond to EC-fluents, events, temporal instants, and complex entities built from these. These entities are differentiated by their various properties. For example, FST-fluents corresponding to events are not homogeneous, they hold only of particular intervals, and not any subintervals of that interval, whereas FST-fluents corresponding to EC-fluents are homogeneous.

Arising from the differences between these two formalisms is the question of whether the whole continuum is needed, or whether some finite representation will suffice for natural language semantics. Specifically, is anything lost by translating models for EC predicates that deal with instantaneous and continuous change, with the real line as the domain, into models for FST representations of these predicates, where the domain is a finite partition of the real line. The FST representations of EC predicates will be languages whose various strings represent different temporal granularities consistent with the EC predicates. A major advantage of this approach is that entailment can be expressed in terms of set-theoretic inclusions between languages [7], which is a decidable question for regular languages. Restricting our representations to regular languages ensures the computability of these entailments.

One application of the EC to event semantics is representing type coercion. Since Vendler [19], it is common to assign verb phrases or eventualities to certain categories such as "achievement" (punctual with consequent state), or "activity" (protracted with no associated end point). Under certain circumstances, however, eventualities typically associated with one category can behave as if in another category. For example, "sneeze" is usually considered to be a "point" (punctual with no consequent state). However, when it occurs with the progressive ("John was sneezing"), it can no longer be viewed as a punctual occurrence and is coerced into an activity (becoming a process of multiple sneezes by iteration).

Taking the concepts of instantaneous and continuous change from the EC, the possibility of representing type coercion in FST will be explored. The main idea is to associate coercion with a "change in perspective", implemented through a change in the alphabet from which strings are drawn.

Section 2 gives a brief introduction to Finite State Temporality. Section 3 will show how models for the EC-predicates can be translated into models for the FST-languages, and vice-versa. It also will show that if an EC-model satisfies an EC-predicate, the translated FST-model will satisfy the translated

FST-language, and vice-versa. Section 4 will address type coercion, and will show how this can be implemented in FST as changes in Σ.

2 Finite State Temporality

Fix a finite set Φ of fluents. The alphabet from which strings are drawn is made up of subsets of Φ:

$$\Sigma = 2^{\Phi} \tag{2}$$

For example, if $\Phi = \{\mathrm{f}, \mathrm{g}\}$, then $\Sigma = \{\{\}, \{\mathrm{f}\}, \{\mathrm{g}\}, \{\mathrm{f}, \mathrm{g}\}\}$. To emphasise that the elements of this alphabet are being used as symbols, and to improve readability, the traditional curly brackets are replaced with boxes enclosing the fluents, with the empty set becoming $\boxed{\ }$.

A typical string over this alphabet might be $\boxed{\mathrm{f, g}}\boxed{\ }\boxed{\mathrm{f}}$ (instead of $\{\mathrm{f}, \mathrm{g}\}\{\}\{\mathrm{f}\}$), while a typical language might look something like $\boxed{\mathrm{f, g}}^*$, consisting of the strings ϵ, $\boxed{\mathrm{f, g}}$, $\boxed{\mathrm{f, g}}\boxed{\mathrm{f, g}}$, $\boxed{\mathrm{f, g}}\boxed{\mathrm{f, g}}\boxed{\mathrm{f, g}}$, etc.

Fernando [9] shows how a set of temporal points can be partitioned into a finite set of intervals, with a satisfaction relation holding between intervals and fluents. Since the EC takes the positive real line ($\mathbb{R}_+$) as its domain, for the purposes of this paper it is natural to take this as the set of temporal points to be partitioned. A model for FST consists of a segmentation of $\mathbb{R}_+$, and an interpretation function, $[\![]\!]_{FST}$, mapping a fluent to the set of intervals which satisfy it.

A segmentation is a sequence $\mathbb{I} = \mathrm{I}_1 \ldots \mathrm{I}_n$ of intervals such that:

$$\mathrm{I}_i \text{ m } \mathrm{I}_{i+1} \text{ for } 1 \leq i < n \tag{3}$$

Here, "m" is the relation "meets", defined by Allen [1] as: A meets B if and only if A precedes B, there is no point between A and B, and A and B do not overlap. $\mathbb{I}$ is a segmentation of I (in symbols: $\mathbb{I} \nearrow \mathrm{I}$) if and only if $\mathrm{I} = \bigcup_{i=1}^{n} \mathrm{I}_i$.

While there are various ways to partition the real line, one particular form of segmentation is especially useful where the EC is concerned:

$$[0, b_1](a_2, b_2] \ldots (a_{n-1}, b_{n-1}](a_n, \infty), \text{ where } \forall i \;\; \mathrm{a}_{i+1} = b_i \tag{4}$$

So $[0, \mathrm{b}_1]$ includes every point between 0 and b_1, including both 0 and b_1 (closed at both ends), whereas $(\mathrm{a}_i, \mathrm{b}_i]$ includes every point between a_i and b_i, including b_i, but not a_i (open at the beginning, closed at the end).

The reason for choosing this form of segmentation (where I_1 is closed at both ends, but $\mathrm{I}_2, \ldots, \mathrm{I}_n$ are open at the beginning and closed at the end) is as follows: In FST, one use of fluents is to represent states (though they can also represent events or "times", see Sect. 2.1). The interpretation of these stative fluents will be a set of intervals from the segmentation. Intuitively, if an interval I is in the interpretation of a fluent f, then f holds across the interval I.

Many temporal reasoning problems specify initial conditions: states or properties which hold at the beginning of time (EC's Initially predicate). These fluents hold from time-point 0 onwards, which requires an interval which includes 0 in their interpretation. This interval will have the form $[0, s]$ for some $s \in \mathbb{R}_+$.

Other states are "brought into being" by events. The assumption in EC (and common in the literature [2,11]) is that these states hold after the event, not during it. So, if an event happens at time point a_i and brings a state f into being, then an interval in the interpretation of f will not include a_i, but will include the points after it, giving an interval that is open at the beginning.

If an event at b_i causes state f to cease holding, the effect will be seen after b_i, meaning that f still holds at b_i but not at any point after. So b_i will be in an interval that is in the interpretation of f, and this interval will be closed at the end. Any fluent caused to hold by an event, and subsequently caused to stop holding by another event will therefore have an interval that is open at the beginning and closed at the end in its interpretation (with inertia causing it to hold at all points within the interval).

A satisfaction relation $\models_{FST}$ is defined between fluents and intervals:

$$I \models_{FST} f \iff I \in [\![f]\!]_{FST} \tag{5}$$

This can be extended to a relation holding between segmentations and strings:

$$I_1 \dots I_n \models_{FST} \alpha_1 \dots \alpha_n \iff (\forall f \in \alpha_i)\ I_i \models_{FST} f, \text{ for } 1 \leq i \leq n \tag{6}$$

and further extended to a relation holding between segmentations and languages:

$$I_1 \dots I_n \models_{FST} L \iff (\exists s \in L)\ I_1 \dots I_n \models_{FST} s \tag{7}$$

2.1 States, Events, and Times

As noted in the introduction, the EC formalises two notions of change, instantaneous change due to punctual events, and continuous change due to some force. These distinctions may seem more at home in the field of physics than linguistics but, as will be seen later, linguistic "eventualities" can be represented as complex sequences of change, both instantaneous and continuous.

Before continuing, some comments must be made on differences between the EC's and FST's ontologies, and how they are related. In the EC, a distinction is drawn between "fluents", which are time-dependent properties that "hold" of time-points, and events, which are punctual entities that "happen" at time-points, causing an instantaneous change by "initiating" (causing to hold), or "terminating" (causing to cease to hold) fluents. Temporal points make up a third category, represented by the positive real numbers (and 0).

In FST, the word "fluent" encompasses these three categories, though their fluents have different properties. Stative fluents represent states/properties. They are homogeneous. A fluent is homogeneous if for all intervals I and I′ such that $I \cup I'$ is an interval:

$$\mathrm{I} \models_{FST} \ \mathrm{f} \text{ and } \mathrm{I}' \models_{FST} \ \mathrm{f} \iff \mathrm{I} \cup \mathrm{I}' \models_{FST} \ \mathrm{f} \tag{8}$$

Essentially, if a stative fluent holds over an interval, and that interval is split into two intervals, the fluent will hold over both intervals. If John slept from 2pm to 4pm, then John slept from 2pm to 3pm, and John slept from 3pm to 4pm (and vice-versa).

Event fluents represent events. Depending on the event they may be homogeneous or "whole". Fernando [9] defines a fluent as being whole if for all intervals I and I′ such that I ∪ I′ is an interval:

$$\mathrm{I} \models_{FST} \ \mathrm{f} \text{ and } \mathrm{I}' \models_{FST} \ \mathrm{f} \text{ implies } \mathrm{I} = \mathrm{I}' \tag{9}$$

The EC makes use of an explicit timeline, necessitating the use of "temporal fluents" in FST to represent this timeline. For every possible interval (including points), there is a "temporal fluent" which marks this interval, i.e. every interval has a fluent whose interpretation is that interval. For simplicity of presentation, we use the same symbol for the fluent as we use for the interval or the point that it marks. It should be noted that the interval (a, b] is a complex symbol built from brackets, a comma, and numbers/variables, while the fluent "(a, b]" is an atomic symbol. This allows statements of the form $\{t\} \models t$ to be used.

3 Translating Language and Models

VLH (p.41) sketch an "intuitively appealing class of models of EC" with domain $\mathbb{R}_+$. The interpretation of fluents is given by the model, and is taken to be a set of intervals of the form $(\mathrm{a}, \mathrm{b}]$ $(\mathrm{a}, \mathrm{b} \in \mathbb{R}_+)$, or $[0, \mathrm{a}]$ $(\mathrm{a} \in \mathbb{R}_+)$. The intervals are, in a sense, "maximal". They are the longest contiguous stretches for which f holds. If an interval is in the interpretation of a fluent f, f will hold for time-point in that interval, but the time-point itself will not be in the interpretation of f.

It cannot be assumed that all models of the EC give rise to finite segmentations of the real line. VLH give the example of an initiating event for a fluent f happening a time t iff $\mathrm{t} \in \mathbb{Q}$, with terminating events happening at a time t′ iff $\mathrm{t}' \in \mathbb{R} - \mathbb{Q}$. This will lead to a fluent varying infinitely between holding and not holding over any interval, a situation which rules out a finite segmentation.

One necessary condition for there to be a finite segmentation is that each fluent is "alternation bounded" [8]. Intuitively this rules out the above case of a fluent varying infinitely between holding and not holding over some interval. We say a fluent is alternation bounded if the set of points or intervals in its extension (U) is alternation bounded. To define when a set of points U taken from a linear order $(T, <)$ is alternation bounded, the following notions are useful.

Given a subset U of T, a *segmentation of* U is a sequence $\mathbb{I} = I_1 \cdots I_n$ of intervals such that $U = \bigcup_{i=1}^{n} I_i$ and $I_i < I_{i+1}$ for $1 \leq i < n$.

A subset I of T is *U-homogeneous* if

$$(\exists t \in I)\ t \in U \iff (\forall t \in I)\ t \in U$$

— i.e.,

$$I \subseteq U \text{ or } I \cap U = \emptyset.$$

A *U-segmentation* is a segmentation $I_1 \cdots I_n$ of T such that for all i between 1 and n, I_i is U-homogeneous and when $i < n$,

$$I_i \subseteq U \iff I_{i+1} \cap U = \emptyset.$$

Given a subset U of T and a positive integer $n > 0$, an (n, U)-*alternation* is a string $t_1 t_2 \cdots t_n \in T$ of length n such that for $1 \leq i < n$, $t_i < t_{i+1}$ and

$$t_i \in U \iff t_{i+1} \notin U.$$

U is *alternation bounded* (a.b.) if for some positive integer n, there is no (n, U)-alternation. It will also be useful to define the equivalence relation

$$\begin{aligned} t \sim_U t' &\iff \text{there is an } (n,U)\text{-alternation with both } t \text{ and } t' \text{ only if } t = t' \\ &\iff [t, t'] \cup [t', t] \text{ is } U\text{-homogeneous} \end{aligned}$$

Lemma For any subset U of T,

$$\text{there is a } U\text{-segmentation} \iff U \text{ is a.b.}$$

For $\Rightarrow$, a U-segmentation of length n implies there can be no $(n+1, U)$-alternation. Conversely, let n be the largest positive integer for which there is an (n, U)-alternation. Let $t_1 \cdots t_n$ be an (n, U)-alternation, and define $I_1 \cdots I_n$ by

$$I_i := \{t \in T \mid t_i \sim_U t\}$$

Then $I_1 \cdots I_n$ is a U-segmentation.

As well as the traditional axioms of the EC, VLH include formulas in the EC, together known as a scenario, which further constrain the possible models. A formula in the scenario with the following form:

$$S(t) \Longrightarrow Happens(e, t) \tag{10}$$

where S(t) is a first-order formula built from:

1. literals of the form (¬)HoldsAt(f, t)
2. equalities between fluent terms, and between event terms
3. formulas in the language of the structure $(\mathbb{R}, <; +, \times, 0, 1)$

can cause a fluent to vary infinitely, as can be seen in the below example:

$$\begin{aligned} &HoldsAt(f, t) \Longrightarrow Happens(e_1, t + 1) \\ &\neg HoldsAt(f, t) \Longrightarrow Happens(e_2, t + 1) \\ &Initiates(e_2, f, t) \\ &Terminates(e_1, f, t) \\ &HoldsAt(f, 0) \end{aligned} \tag{11}$$

Of course, in this particular example, the fluent can only vary infinitely over an interval stretching to infinity, and while arguments can be made that this is an unrealistic condition for natural language semantics, the formulation of the EC does allow it.

Due to the inertial axioms, when a fluent is initiated, it holds until it is terminated. An initiating event has no effect on a fluent that already holds, and similarly, a terminating event has no effect on a fluent that does not hold. It follows that an infinite variation of the holding of a fluent over an interval can only arise if there is an infinite sequence of alternating initiating and terminating events.

A further necessary condition for there to exist a finite segmentation of an EC-model is that we are dealing with only a finite number of these alternation bounded fluents. Each of these fluents defines a finite segmentation, and taking a finite number of these fluents together and intersecting the intervals in their segmentations gives a finite segmentation. It will be seen in Sect. 4 on type coercion that eventualities are defined using a finite number of fluents, and type coercion involves adding or removing a finite number of fluents, allowing us to only deal with finite segmentations. Hereafter, when we refer to an "EC-model" it is understood we are discussing models consistent with these conditions:

1. The inferences of interest involve only a finite number of fluents.
2. Each fluent is alternation bounded.

An FST-model can be formed from an EC-model as follows:

1. For each fluent f in EC, form $[\![f]\!]_{EC \to FST} : I \in [\![f]\!]_{EC}$ implies $(\forall I' \subseteq I) I' \in [\![f]\!]_{EC \to FST}$
2. Choose a segmentation $\mathbb{I}$ of $\mathbb{R}_+$
3. Form $[\![f]\!]_{FST}$ by relativizing $[\![f]\!]_{EC \to FST}$ to $\mathbb{I} = I_1 \ldots I_n : I_i \in [\![f]\!]_{FST}$ iff $I_i \in [\![f]\!]_{EC \to FST}$. Only those intervals that are part of the segmentation, $\mathbb{I}$, can be in $[\![f]\!]_{FST}$.

An EC-model can be formed from an FST-model as follows:

1. Suppose $\mathbb{I} \nearrow \mathbb{R}_+$. Find a sequence $I_i \ldots I_j$ in $\mathbb{I}$ such that for all $(i \leq q \leq j)$ $I_q \in [\![f]\!]_{FST}, I_{i-1} \notin [\![f]\!]_{FST}$ and $I_{j+1} \notin [\![f]\!]_{FST}$. This sequence will be the longest unbroken stretch for which f holds.
2. Put $I = \bigcup_{r=i}^{j} I_r$ in $[\![f]\!]_{EC}$.

3.1 Initially

In the EC, Initially(f) signifies that the fluent f "was true at the beginning of time" (VLH p.38). It can be translated as follows:

$$\mathrm{L}_{\mathrm{Initially(f)}} = \langle \trianglerighteq \rangle \boxed{\mathrm{f}}\boxed{\phantom{\mathrm{f}}}^{*} \tag{12}$$

Every string in this language includes the fluent f in its first box/symbol. Understanding the above equation requires two definitions:

$$\mathrm{s} \in \langle \mathrm{R} \rangle L \iff (\exists \mathrm{s}' \in L)\ \mathrm{s}\ \mathrm{R}\ \mathrm{s}' \tag{13}$$

where R is some relation between strings. Every string in ⟨R⟩L bears the relation R to some string in L.

$\trianglerighteq$ is the relation "subsumes". If s and s′ are strings, where $\mathrm{s} = \alpha_1 \ldots \alpha_n$ and $\mathrm{s}' = \beta_1 \ldots \beta_k$ then:

$$s \trianglerighteq s' \text{iff n = k, and for every i, } \alpha_i \supseteq \beta_i \tag{14}$$

So every symbol of s contains all the fluents (information) of the corresponding symbol of s′, and possibly more. Fernando [6] shows that this relation can be computed using finite-state methods.

According to VLH (p.42), a model for Initially(f) will have, in its interpretation of f, an interval that begins at 0:

$$Initially := \{f | (\exists s > 0)[0, s] \in [\![f]\!]_{EC}\} \tag{15}$$

In FST, a model for $\mathrm{L}_{\mathrm{Initially(f)}}$ is a segmentation of $\mathbb{R}_+$, where the first interval is in the interpretation of f:

$$I_1 \ldots I_n \nearrow \mathbb{R}_+ \text{ and } I_1 \in [\![f]\!]_{FST} \tag{16}$$

EC → FST: If there is an EC-model for Initially(f) then, by (15), there is some $\mathrm{s} \in \mathbb{R}$ where $[0, \mathrm{s}] \in [\![f]\!]_{EC}$. By construction of an FST-model, [0, s] and all its subintervals are in $[\![f]\!]_{EC \to FST}$. Any segmentation where $\mathrm{I}_1 = [0, \mathrm{r}]$, with $\mathrm{r} \leq \mathrm{s}$, will have $\mathrm{I}_1 \in [\![f]\!]_{FST}$, satisfying (16).

FST → EC: For an FST-model of $\mathrm{L}_{\mathrm{Initially(f)}}$ where $I_1 \ldots I_n \nearrow \mathbb{R}_+$ and $I_1 \in [\![f]\!]_{FST}$ there exists some j with $1 < \mathrm{j} \leq \mathrm{n}$ such that $I_j \notin [\![f]\!]_{FST}$. By construction of an EC-model, $I = \bigcup_{r=1}^{j-1} I_r \in [\![f]\!]_{EC}$. Since I_1 is included in I, I begins at 0 and therefore is of the form [0, s] ($\mathrm{s} \in \mathbb{R}$), which fulfills condition (15).

3.2 Instantaneous Change: Happens and Initiates

When dealing with events and their effects on fluents, some new fluents will prove helpful:

$$I \models_{FST}\ )_e \iff (\forall J \text{ such that } I \text{ m } J) I \not\models_{FST}\ f \text{ and } J \models_{FST} f \tag{17}$$

$$(a, b] \models_{FST}\ \langle end \rangle t \iff \{b\} \models_{FST} t \tag{18}$$

$$(a, b] \models_{FST}\ \langle start \rangle t \iff \{a\} \models_{FST} t \tag{19}$$

The fluent $)_e$ is used to mark the end of events. In the EC, events are punctual: they occur at points (interpreted as real numbers). In FST, events are allowed

to occur over intervals. The reason for this will become clear when type coercion is dealt with. If $\langle end \rangle t$ holds of an interval, then that interval ends at the time point t. If $\langle start \rangle t$ holds of an interval, then that interval starts after the time point t.

The EC treats change due to an event as instantaneous. A fluent initiated by a punctual event holds AFTER, but not AT, the time at which the event occurs. When moving from punctual events to events that happen over intervals, a choice must be made as to when the change occurs. In keeping with the EC, FST formalizes the change as happening after the event ends. Only initiating events are dealt with below, the definitions for terminating events are broadly similar. Note that the fluents f and $)_e$ are linked in that this particular event initiates the fluent f, other events initiate other fluents.

Happens(e, t), which signifies that event e occurs at time t, can be translated as follows:

$$\mathrm{L}_{\mathrm{Happens(e,t)}} = \langle \sqsupseteq \rangle \boxed{(_e \; \boxed{\quad}^* \; \boxed{\langle end \rangle t,)_e}} \tag{20}$$

If $\mathrm{s} = \alpha_1 \ldots \alpha_n, \mathrm{s} \sqsupseteq \mathrm{s}'$ iff there is some substring r of s, $\alpha_i \ldots \alpha_j$, and $\mathrm{r} \trianglerighteq \mathrm{s}'$. $(_e$ represents the start of the event. It cannot be translated from the EC as the EC treats events as punctual. Its purpose in FST is to allow punctual events to be "stretched" and given an internal structure. It causes no problem in translations as $)_e$ is the necessary fluent to represent instantaneous change.

VLH (p.42) define the extension of Happens as follows:

$$Happens := \{(e,t) | (\exists f)(f,t) \in e\} \tag{21}$$

In an EC-model, the event e will either initiate or terminate some fluent f. If it is an initiating event then there must be some $\mathrm{s} \in \mathbb{R}$ for which $(\mathrm{t}, \mathrm{s}] \in [\![f]\!]_{EC}$.

In FST, a model for $\mathrm{L}_{\mathrm{Happens(e,t)}}$ will be a segmentation $I_1 \ldots I_n \nearrow \mathbb{R}_+$ with the following conditions:

$$(\exists I_i, I_j) I_i < I_j \text{ and } I_i \models_{FST} \; (_e \text{ and } I_j \models_{FST})_e \text{ and } I_j \models_{FST} \; \langle end \rangle t \tag{22}$$

Note that $<$ is the relation "precedes". For intervals I and J, I precedes J if its end point is before J's start point.

Using the convention $\{t\} \models_{FST} \; t$, and the definition of $\langle end \rangle t$, then $I_j = (a_j, \mathrm{t}]$ for some $a_j \in \mathbb{R}$. From the definition of $)_e$, $(a_j, \mathrm{t}] \not\models_{FST}$ f and $(\mathrm{t}, b_{j+1}] \models_{FST}$ f for some $b_{j+1} \in \mathbb{R}$.

EC $\rightarrow$ FST: From above ($\exists$ s) $(\mathrm{t}, \mathrm{s}] \in [\![f]\!]_{EC}$. Because the intervals in the interpretation of fluents are maximal, there must be an interval (at least a point) that meets (t, s] which is not in the interpretation of f. Therefore, $(\mathrm{r}, \mathrm{t}] \notin [\![f]\!]_{EC}$ for some $\mathrm{r} \in \mathbb{R}$. Constructing an FST-model from this, (t, s] and all its subintervals are in $[\![f]\!]_{EC \rightarrow FST}$, and there is some $\mathrm{q} \in \mathbb{R}$ for which (q, t] and all its subintervals are not in $[\![f]\!]_{EC \rightarrow FST}$. An example segmentation would be $[0, \mathrm{b}_1] \ldots (\mathrm{q}, \mathrm{t}] (\mathrm{t}, \mathrm{s}] \ldots (\mathrm{a}_n, \infty)$. $[\![f]\!]_{EC \rightarrow FST}$ relativized to this will have (q, t]

$\notin [\![f]\!]_{FST}$, $(t, s] \in [\![f]\!]_{FST}$ satisfying the conditions above with a_j = q, and b_{j+1} = s. (Again there will be many segmentations of $\mathbb{R}_+$ that satisfy this language).

FST → EC: For an FST-model of $L_{\text{Happens(e,t)}}$ where $I_1 \ldots I_n \nearrow \mathbb{R}_+$ and $I_j \notin [\![f]\!]_{FST}$ and $I_{j+1} \in [\![f]\!]_{FST}$, there exists some r with j+1<r≤n such that $I_r \notin [\![f]\!]_{FST}$. By construction of an EC-model $I = \bigcup_{d=j+1}^{r-1} I_d \in [\![f]\!]_{EC}$. Since I_{j+1} is the first interval in I, I is of the form (t, s], satisfying the condition above for an EC-model.

Initiates(e, f, t) can be translated as follows:

$$L_{\text{Initiates(e,f,t)}} = \boxed{\langle end\rangle t,)_e \,\boxed{}} \Longrightarrow \boxed{\boxed{\text{f}}} \tag{23}$$

This is encoded as a constraint on strings. The constraint L ⟹ L′ is the set of strings s, such that every stretch of s that subsumes L also subsumes L′. The notion of "stretched" is formalized as follows: s′ is a factor of s if s = us′v for some (possibly empty) strings u and v.

$$s \in L \Longrightarrow L' \iff \text{for every factor } s' \text{ of } s, s' \trianglerighteq L \text{ implies } s' \trianglerighteq L' \tag{24}$$

The constraint for $L_{\text{Initiates(e,f,t)}}$ says that every string in this language that has a box containing $\langle end\rangle t$ and $)_e$, must contain f in the following box.

An EC-model for Initiates(e, f, t) must have the following condition: $(\exists s \in \mathbb{R})(t, s] \in [\![f]\!]_{EC}$.

In FST, a model for $L_{\text{Initiates(e,f,t)}}$ will be a segmentation $I_1 \ldots I_n \nearrow \mathbb{R}_+$ with the following conditions:

$$(\forall I_i, I_j)[I_i \text{ m } I_j \text{ and } I_i \models_{FST})_e \text{ and } I_i \models_{FST} \langle end\rangle t] \longrightarrow I_j \models_{FST} f \tag{25}$$

EC → FST: From above (∃ s) (t, s]∈ $[\![f]\!]_{EC}$. For the same reason as above for the Happens predicate, (t, s] and all its subintervals are in $[\![f]\!]_{EC \to FST}$, and there is some q ∈ ℝ for which (q, t] and all its subintervals are not in $[\![f]\!]_{EC \to FST}$. Taking as an example segmentation, $[0, b_1]\ldots(q, t](t, s]\ldots(a_n,\infty)$, the interval (q, t] meets (t, s], it satisfies $\langle end\rangle t$, it also satisfies $)_e$ (by definition of $)_e$). Therefore it meets the conditions of the antecedent of (25), and since it is given that (t, s] ∈ $[\![f]\!]_{EC}$, and therefore in $[\![f]\!]_{FST}$, the consequent is also true, fulfilling the conditions for an FST-model.

FST → EC: Take an FST-model of $L_{\text{Initiates(e,f,t)}}$ where $I_1 \ldots I_n \nearrow \mathbb{R}_+$ and (25) holds. Given that there is only one interval I_i in this segmentation which satisfies $\langle end\rangle t$, and only one interval I_j that it meets, we can reduce (25) to:

$$(a_i, t] \models_{FST})_e \longrightarrow (t, b_j] \models_{FST} f \tag{26}$$

By definition of $)_e$, $(a_i, t] \not\models_{FST}$ f. (26) then becomes:

$$(a_i, t] \not\models_{FST} \ \ f \longrightarrow (t, b_j] \models_{FST} \ \ f \tag{27}$$

there exists some r with j+1<r≤n such that $I_r \notin [\![f]\!]_{FST}$. By construction of an EC-model, $I = \bigcup_{d=j}^{r-1} I_d \in [\![f]\!]_{EC}$. Since I_{j+1} is the first interval in I, I is of the form (t, s], satisfying the condition above for an EC-model.

3.3 Continuous Change: Trajectory

Trajectory$(f_1, t, f_2, t+d)$ signifies that if fluent f_1 holds between t and t + d, then fluent f_2 will hold at t + d. In VLH, f_2 is generally a real-valued function describing continuous change, so the Trajectory predicate describes continuous change as long as fluent f_1 holds. It can be translated as follows:

$$\mathrm{L}_{\mathrm{Trajectory(f_1,t,f_2,t+d)}} = \boxed{\langle start\rangle t, f_1}\boxed{f_1}^{*}\boxed{\langle end\rangle t+d, f_1} \Longrightarrow \boxed{\ }^{*}\boxed{\langle end\rangle f_2} \tag{28}$$

While VLH does not give a model for Trajectory, it is not hard to construct an intuitive model consistent with the other predicates:

$$\begin{aligned} Trajectory := \{(f_1, t, f_2, t+d) | (\exists a, b \in \mathbb{R}_+)\ a \le t \le t+d \le b \text{ and} \\ (a, b] \in [\![f_1]\!]_{EC} \longrightarrow (\exists I \in [\![f_2]\!]_{EC}) t+d \in I\} \end{aligned} \tag{29}$$

A FST-model for $\mathrm{L}_{\mathrm{Trajectory(f_1,t,f_2,t+d)}}$ will be a segmentation of $\mathbb{R}_+$ with the following conditions:

$$(\exists I_i = (t, b_i], I_j = (a_j, t+d])(\forall i \le p \le j) I_p \models_{FST} \ \ f_1 \longrightarrow \{t+d\} \models_{FST} \ \ f_2 \tag{30}$$

EC → FST: If the antecedent of (30) is true, then the sequence $I_1 \ldots I_j$ is part of an unbroken stretch over which f holds. Let the start and end points of this stretch be a and b from (29). Therefore, (a,b] ∈ $[\![f]\!]_{EC}$, fulfilling the antecedent of the conditional contained in (29). Now from the consequent of (29), there is an interval I ∈ $[\![f_2]\!]_{EC}$, so all its subintervals, notably {t + d}, are in $[\![f_2]\!]_{EC \to FST}$. By definition of $\langle end\rangle t + d$, $(a_j, t + d] \in [\![\langle end\rangle f_2]\!]_{EC \to FST}$. Relativized to the segmentation given, $(a_j, t + d] \in [\![\langle end\rangle f_2]\!]_{FST}$. Therefore {t + d}$\models_{FST}$ f_2.

FST → EC: For an FST-model of $\mathrm{L}_{\mathrm{Trajectory(f_1,t,f_2,t+d)}}$ where $I_1 \ldots I_n \nearrow \mathbb{R}_+$ and, supposing the antecedent contained in (29) is true, $(t, b_i], (a_j, t + d]$ and all the intervals between them are in $[\![f_1]\!]_{FST}$ (by construction of an FST-model. Therefore, the consequent of (30) is true, so {t + d}$\models$ f_2, so {t + d} ∈ $[\![f_2]\!]_{FST}$. {t + d} is either a maximal interval for which f_2 holds or is a subinterval of that maximal interval. Either way, there is some I ∈ $[\![f_2]\!]_{EC}$ with t+d ∈ I.

While VLH (p.45) require a real-valued function of time in place of f_2, and while this can be implemented in FST as long as only a finite number of values from this function are used as fluents, it is not clear that natural language

semantics needs the precision of this real-valued function to address continuous change. As will be seen in the next section, when dealing with type coercions that involve adding elements to an event structure, it is not always clear in advance what form these elements will have. An alternative in FST to the above representation of continuous change is as follows:

$$\boxed{\mathrm{f}_1} \Longrightarrow \boxed{\mathrm{f}_{2\uparrow}} \tag{31}$$

The above essentially says that if f_1 is in a box, then $\mathrm{f}_{2\uparrow}$ is in the same box, or as long as f_1 holds, f_2 is increasing, or moving along some trajectory. For type coercion, it will be useful to have a fluent that holds when the end of the trajectory is reached, $\mathrm{f}_{2,MAX}$.

These fluents can be interpreted model-theoretically, relative to an underlying function, representing trajectory or path taken, in the model. This is in line with the approach of Kennedy and Levin [13], who propose that the semantics of degree achievements (such as "the soup cooled") rely on a "difference function", a function that measures the amount an object changes along a scalar dimension as a result of participating in an event. Certain functions describing change will have an end-point or maximum. This may be contextually given, as Fernando [10] assumes, or may be a natural feature of the scale against which change is measured. Closed scales (such as "smooth") have a natural maximum, complete smoothness in this case [18].

3.4 Scenarios

The EC relates instantaneous and continuous change to natural language semantics through scenarios, which "state the specific causal relationships holding in a given situation" (VLH, p.43). Only the elements of the a scenario dealing with change will be dealt with here.

A scenario is a conjunct of statements of the form:

1. Initially(f)
2. $\mathrm{S(t)} \rightarrow \mathrm{Initiates(e, f, t)}$
3. $\mathrm{S(t)} \rightarrow \mathrm{Terminates(e, f, t)}$
4. $\mathrm{S(t)} \rightarrow \mathrm{Happens(e, t)}$
5. $\mathrm{S}(f_1, f_2, t, d) \rightarrow \mathrm{Trajectory}(\mathrm{f}_1, \mathrm{t}, \mathrm{f}_2, \mathrm{d})$

where S(t) is a conjunction of statements of the form HoldsAt(f, t). Element 5 is referred to as the "dynamics", relating fluent f_1 (viewed as a force), to the change in f_2 (viewed as a changing, partial object).

As with EC-predicates, each element of an EC-scenario can be translated into FST. If S(t) is not a part of an element (for instance if one element of the scenario is Initiates(e, f, t), as opposed to $\mathrm{S(t)} \rightarrow \mathrm{Initiates(e, f, t)}$), then the equivalent element of the FST-scenario is the language given above as equivalent. If S(t) is part of an element then the equivalent will be a constraint relating fluents holding at a certain time to the language given above as equivalent. For example, $\mathrm{HoldsAt(f, t)} \rightarrow \mathrm{Happens(e, t)}$ can be translated as follows:

$$\boxed{}^{*}\boxed{\langle end\rangle\ \mathrm{t}, \langle end\rangle\ \mathrm{f}} \Longrightarrow \boxed{}^{*}\boxed{(_e}\boxed{}^{*}\boxed{)_e} \tag{32}$$

For the temporal fluents occurring in a string to have the correct ordering, one further set of constraints is needed. In an FST-model, every interval is ordered relative to every other interval by the "precedes" relation ($<$). A scenario will contain the set of constraints:

$$\forall I, I' \text{ in an FST-model such that } I < I' : \boxed{\mathrm{I'}}\boxed{}^{*}\boxed{\mathrm{I}} \Longrightarrow \emptyset \tag{33}$$

The above constraint says that if, in an FST-model, an interval I precedes an interval I′, then any string with the fluent I′ preceding I will not be in the language.

4 Type Coercion

Having developed the tools necessary to represent instantaneous and continuous change in FST, these are now applied to the natural language semantics problem of type coercion.

4.1 Eventualities vs. Events

The EC treats "events" as punctual occurrences that cause an instantaneous change (a fluent becomes initiated or terminated). However, most occurrences described as events have a more complex structure. Moens and Steedman [16] (hereafter referred to as M&S) have proposed a three-part event structure, called a "nucleus", which consists of a preparatory process, a culmination, and a consequent state. They note that all of these elements can be compound. The culmination "reaching the top of Mt. Everest" may have a number of processes such as climbing, eating lunch, etc. as part of its preparatory process, and there may be many consequent states.

Due to this, it is important to note that the discussion of coercion must have a quite general character. If adding a consequent state to an event structure, the question arises: what or which consequent state? For this reason, non-specific fluents such as f_3, g_1 etc. will often be used to describe "sailent' states or activities that have been added to an event structure.

The EC implements a similar event structure to that of M&S which they call "eventualities", having the following form: $(\mathrm{f}_1, \mathrm{f}_2, \mathrm{e}, \mathrm{f}_3)$ where f_1 is an activity, f_2 is a fluent representing a partial object which changes under the influence of the activity, e is a culminating event, and f_3 is a consequent state.

Different approaches have used different terminologies for what are, essentially, the same categories. What Vendler [19] called activities, achievements, and accomplishments, M&S call processes, culminations, and culminated processes respectively.

4.2 Activity $\rightsquigarrow$ Accomplishments

VLH (p.171) discuss the case of "additive coercion", where a scenario is elaborated or added to. Alternative views of this type of coercion are given by Pulman [17] and M&S. Pulman sees this as supplementing a process with a consequent state (Pulman does not include culminating events in his ontology). The activity of "swimming" would have a salient consequent state added, perhaps as general as "has swum". M&S propose that the process is bundled into a point, with a new preparatory process and consequent state added. They cite the example "has John worked in the garden?" as only making sense if John working in the garden was part of some plan, with the preparatory process being whatever preparation was involved to work in the garden, the culmination being the working in the garden viewed as a point, and the consequent state perhaps being "has worked in the garden".

Both these views can be accounted for in FST. In the first, the activity f_1 is augmented with a fluent representing the change in the partial object (swimming augmented with the trajectory of the swim), a culminating event (finishing the swim), and a consequent state (has swum). In alphabet terms:

$$\Sigma \rightsquigarrow \Sigma' = \Sigma \cup \{f_{2\uparrow}, f_{2,MAX}, (_e,)_e, f_3\} \tag{34}$$

The scenario will also have to be augmented with general constraints of the form:

$$\boxed{\mathrm{f}_1} \Longrightarrow \boxed{f_{2\uparrow}} \tag{35}$$

$$\boxed{\mathrm{f}_{2,MAX}} \Longrightarrow \boxed{)_e} \tag{36}$$

$$\boxed{)_e}\boxed{\ } \Longrightarrow \boxed{\ }\boxed{f_3} \tag{37}$$

For M&S's coercion to be implemented, f_1, the original activity and internal structure of some event, is deleted, and fluents representing its start and end points are added, turning it into a point with no internal structure. A new preparatory process, g_1 is added, and a new consequent state, g_3. In alphabet terms:

$$\Sigma \rightsquigarrow \Sigma' = \Sigma \cup \{g_1, g_{2\uparrow}, g_{2,MAX}, (_e,)_e, g_3\} - \{f_1\} \tag{38}$$

The scenario will be elaborated as follows:

$$\boxed{\mathrm{g}_1} \Longrightarrow \boxed{g_{2\uparrow}} \tag{39}$$

$$\boxed{\mathrm{g}_{2,MAX}} \Longrightarrow \boxed{)_e} \tag{40}$$

$$\boxed{)_e}\boxed{\ } \Longrightarrow \boxed{\ }\boxed{g_3} \tag{41}$$

4.3 Achievements ⇝ Accomplishments

As pointed out in M&S, the progressive needs a process (activity), or culminated process (accomplishment) as "input", so what to make of the following:

John was reaching the top (42)

"Reach the top" is usually seen as a culmination (achievement), punctual with an associated consequent state. The progressive coerces this culmination into a culminated process by adding a preparatory process, and focussing on this.

For VLH (p.172), the achievement $(\text{-}, \text{-}, \mathrm{e}_1, \mathrm{f}_3)$, where e_1 would be the culmination (reaching the top), and f_3 would be the consequent state of this (perhaps being at the top), would be coerced into an accomplishment $(\mathrm{g}_1, \mathrm{g}_2, \mathrm{e}_2, \mathrm{g}_3)$, where g_1 and g_2 are "unknown parameters", depending on what preparatory process is being associated with the culmination. Presumably the culminations and consequent states of "reach the top" viewed as an achievement and as an accomplishment are the same. The addition of the fluents representing the preparatory process and changing partial object means a dynamics must be added to the scenario to relate these fluents.

To account for the addition of a preparatory process, and changing object in FST, the fluents marking the beginning and ending of events $(_e$ and $)_e$ are associated with a fluent prog_e which signifies that the event in question is ongoing:

$$I \models prog_e \Longleftrightarrow I \models (_e \text{ or } I \models)_e \text{ or } [(\exists I_i, I_j) I_i < I < I_j \text{ and} I_i \models (_e \text{ and } I_j \models)_e]$$

This fluent corresponds to g_1 from the EC. Its addition to the alphabet of FST will lead to a finer-grained segmentation, effectively giving the event, previously viewed as punctual, an internal structure. In alphabetic terms:

$$\Sigma \rightsquigarrow \Sigma' = \Sigma \cup \{prog_e, g_{2\uparrow}, g_{2,MAX}, (_d,)_d, g_3\} - \{f_3\} \tag{43}$$

4.4 Accomplishments ⇝ Activities

M&S discuss the case of a culminated process being coerced into a process in the presence of the progressive:

Roger was running a mile (44)

For this to happen, the culmination and consequent state must be "stripped off", leaving the preparatory process.

In VLH (p.172), this process is described as subtractive coercion, essentially removing the last two elements from the event-nucleus, and removing statements relating to the culmination and consequent state from the scenario. The same can be achieved in FST by removing those constraints from the scenario that "cause" the culminating event to happen at the maximum point in the trajectory

of the run, and relate the consequent state to the occurrence of the culminating event. In alphabetic terms:

$$\Sigma \rightsquigarrow \Sigma' = \Sigma - \{(_e,)_e, f_3\} \tag{45}$$

Another way for this coercion to occur is if the direct object is a mass noun. This would lead to a lack of end point in the trajectory, perhaps even a lack of trajectory at all.

4.5 Points $\rightsquigarrow$ Activities

As noted in M&S, the progressive requires a process as "input", yet interpretations can be given for:

Harry was sneezing (46)

M&S see this as being coerced by iteration, referring to a number of sneezes, rather than one sneeze.

Both Comrie [4] and Pulman [17] provide another interpretation for this coercion, where what was viewed as a point is stretched into a process, having internal structure.

VLH (p.175) deal with the first type, viewing it as a change from (-, -, e, -) to (f_1, -, -, -) or (f_1, f_2, -, -). While the EC needs to coerce an event into an activity fluent (a method for this is provided in [12]), FST does not face this ontological problem.

Events in FST can be represented by a multitude of fluents, providing different "perspectives" on events. Up to now $(_e)_e$ have been used to mark the start and end points of events, where we were not concerned with their internal or overall structure. Not representing the internal structure leads to the events being viewed as points. A fluent representing internal structure can be introduced, and as a fluent with a model-theoretic definition, there are no ontological obstacles to doing this.

To represent a point-like event being "stretched' the fluent prog_e, defined above, is added. This represents the event in progress as a homogeneous, stative fluent.

To represent iteration, a fluent iter_e is defined, which holds of any interval if two or more events happen within that interval. It can be thought of as a homogeneous process. While a particular subinterval may not contain a sneeze, it is considered part of the process of iteratively sneezing.

$$I \models_{FST} iter_e \iff (\exists I_i < I_j < I_k < I_l) \text{ s.t. } \begin{cases} I_i \models_{FST} (_e \\ \text{and } I_j \models_{FST})_e \\ \text{and } I_k \models_{FST} (_e \\ \text{and } I_l \models_{FST})_e \end{cases} \tag{47}$$

This coercion can be achieved by adding either prog_e or iter_e to the alphabet:

$$\Sigma \rightsquigarrow \Sigma' = \Sigma \cup \{prog_e\} \tag{48}$$

$$\Sigma \rightsquigarrow \Sigma' = \Sigma \cup \{iter_e\} \tag{49}$$

4.6 States ⇝ Activities

Croft [5] gives the following example of a state being coerced into a process:

She is resembling her mother more and more every day (50)

Though the acceptability of this type of coercion varies, VLH (p.173) give an account where a fluent representing the state "resembles her mother" is coerced into an activity fluent. They argue against coercing the state into the changing partial object, as might be expected given that increasing resemblance over time could be viewed as a trajectory.

Since a state is homogeneous, there are no problems treating it as an activity in FST. Activities and states are both represented by homogeneous fluents.

The activity fluent, which previously represented a state, must be related to a changing, partial object (represented by f_2) by some constraint set, leading to the crucial difference between "resembles" and "is resembling more and more".

5 Conclusion

The Event Calculus differs from other temporal representations by directly formalizing change, both instantaneous change as a result of events, and continuous change as a result of some constant force. Section 3 described how EC-predicates can be implemented as regular languages in FST. Furthermore, it was shown how a model, with a finite segmentation of the real number line as domain, for these FST-languages, could be formed from a model (with certain conditions excluding those models for which no finite segmentation exists), with the real number line as domain, for the equivalent EC-predicates. This shows that if a natural language semantics problem can be described in terms of change (or at least the kind of change that the EC formalizes), then the full real number line is not needed to model this problem.

Section 4 discusses type coercion, which the EC has been applied to. It is shown how various types of coercion, implemented in the EC as changes in scenario, or transformations of type (from fluents to events), can be implemented in FST. Type coercions in FST can be viewed as changes in "focus" or "perspective", formalized as changes in the alphabet from which strings are drawn.

Acknowledgement. This research is enhanced by support from Science Foundation Ireland through the CNGL Programme (Grant 12/CE/I2267) in the ADAPT Centre (www.adaptcentre.ie) at Trinity College Dublin. The ADAPT Centre for Digital Content Technology is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund.

References

1. Allen, J.F.: An interval-based representation of temporal knowledge. IJCAI **1**, 221–226 (1981)
2. Allen, J.F., Ferguson, G.: Actions and events in interval temporal logic. In: Stock, O. (ed.) Spatial and Temporal Reasoning, pp. 205–245. Springer, Dordrecht (1997)
3. Bennett, M., Partee, B.H.: Toward the logic of tense and aspect in English. Wiley Online Library (1978)
4. Comrie, B.: Aspect: An Introduction to the Study of Verbal Aspect and Related Problems. Cambridge University Press, Cambridge (1976)
5. Croft, W.: The structure of events. In: Tomasello, M. (ed.) The New Psychology of Language, Chap. 3. Lawrence Erlbaum Associates, Mahwah (1998)
6. Fernando, T.: Finite-state temporal projection. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 230–241. Springer, Heidelberg (2006)
7. Fernando, T.: Temporal propositions as regular languages. In: 6th International Workshop on Finite-State Methods and Natural Language Processing, pp. 132–48 (2008)
8. Fernando, T.: Partitions representing change homogeneously. In: Aloni, M., Franke, M., Roelofsen, F. (eds.) A festschrift for Jeroen Groenendijk, Martin Stokhof, and Frank Veltman, pp. 91–95. Onbekend (2013)
9. Fernando, T.: Segmenting temporal intervals for tense and aspect. In: The 13th Meeting on the Mathematics of Language, p. 30 (2013)
10. Fernando, T.: Incremental semantic scales by strings. In: EACL 2014, p. 63 (2014)
11. Halpern, J.Y., Shoham, Y.: A propositional modal logic of time intervals. J. ACM (JACM) **38**(4), 935–962 (1991)
12. Hamm, F., van Lambalgen, M.: Nominalization, the progressive and event calculus. Linguist. Philos. **26**, 381–458 (2003)
13. Kennedy, C., Levin, B.: Measure of change: the adjectival core of degree achievements. In: McNally, L., Kennedy, C. (eds.) Adjectives and Adverbs: Syntax, Semantics and Discourse, pp. 156–182. Oxford University Press, Oxford (2008)
14. Kowalski, R., Sergot, M.: A logic-based calculus of events. In: Schmidt, J.W., Thanos, C. (eds.) Foundations of Knowledge Base Management, pp. 23–55. Springer, Heidelberg (1989)
15. van Lambalgen, M., Hamm, F.: The Proper Treatment of Events. Wiley, New York (2005)
16. Moens, M., Steedman, M.: Temporal ontology and temporal reference. Comput. Linguist. **14**(2), 15–28 (1988)
17. Pulman, S.G.: Aspectual shift as type coercion. Trans. Philol. Soc. **95**(2), 279–317 (1997)
18. Solt, S.: Measurement scales in natural language. Lang. Linguist. Compass **9**(1), 14–32 (2015)
19. Vendler, Z.: Verbs and times. The Philos. Rev. **66**, 143–160 (1957)

A Modal Representation of Graded Medical Statements

Hans-Ulrich Krieger[1(✉)] and Stefan Schulz[2]

[1] German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany
krieger@dfki.de
[2] Institute of Medical Informatics, Medical University of Graz, Graz, Austria
stefan.schulz@medunigraz.at

Abstract. Medical natural language statements uttered by physicians are usually *graded*, i.e., are associated with a degree of uncertainty about the validity of a medical assessment. This uncertainty is often expressed through specific *verbs*, *adverbs*, or *adjectives* in natural language. In this paper, we look into a *representation* of such graded statements by presenting a simple non-normal *modal logic* which comes with a set of modal operators, directly associated with the words indicating the uncertainty and interpreted through *confidence intervals* in the model theory. We complement the model theory by a set of RDFS-/OWL 2 RL-like entailment (*if-then*) rules, acting on the syntactic representation of modalized statements. Our interest in such a formalization is related to the use of OWL as the *de facto* standard in (medical) ontologies today and its weakness to represent and reason about assertional knowledge that is uncertain or that changes over time. The approach is not restricted to medical statements, but is applicable to other graded statements as well.

1 Introduction and Background

Medical natural language statements uttered by physicians or other health professionals and found in medical examination letters are usually *graded*, i.e., are associated with a degree of uncertainty about the validity of a medical assessment. This uncertainty is often expressed through specific verbs, adverbs, or adjectives in natural language (which we will call *gradation words*). E.g., *Dr. X suspects that Y suffers from Hepatitis* or *The patient probably has Hepatitis* or *(The diagnosis of) Hepatitis is confirmed*.

In this paper, we look into a representation of such graded statements by presenting a simple non-standard modal logic which comes with a small set of partially-ordered modal operators, directly associated with the words indicating the uncertainty and interpreted through confidence intervals in the model theory. The approach currently only addresses modalized propositional formulae in negation normal form which can be seen as a canonical representation of natural language sentences of the above form (a kind of a *controlled natural language*).

Our interest in such a formalization is related to the use of OWL in our projects as the *de facto* standard for (medical) ontologies today and its weakness

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 130–146, 2016.
DOI: 10.1007/978-3-662-53042-9_8

to represent and reason about assertional knowledge that is uncertain [15] or that changes over time [12]. There are two principled ways to address such a restriction: *either* by sticking with the existing formalism (viz., OWL) and trying to find an encoding that still enables some useful forms of reasoning [15]; *or* by deviating from a defined standard in order to arrive at an easier, intuitive, and less error-prone representation [12].

Here, we follow the latter avenue, but employ and extend the standard entailment rules from [6,18] for positive binary relation instances in RDFS and OWL towards modalized n-ary relation instances, including negation. These entailment rules talk about, e.g., subsumption, class membership, or transitivity, and have been found useful in many applications. The proposed solution has been implemented in *HFC* [13], a forward chaining engine that builds Herbrand models which are compatible with the open-world view underlying OWL. The approach presented in this paper is clearly not restricted to medical statements, but is applicable to other graded statements as well (including *trust*), e.g., technical diagnosis (*The engine is probably overheated*) or more general in everyday conversation (*I'm pretty sure that X has signed a contract with Y*) which can be seen as the common case (contrary to true *universal* statements).

2 Graded Medical Statements: OWL vs. Modalized Representation

We note here that our initial modal operators were inspired by the *qualitative information parts* of diagnostic statements from [15] shown in Fig. 1, but we might have chosen other operators, capturing the meaning of the gradation words used in the examples at the beginning of Sect. 1 (e.g., *probably*).

Fig. 1. Vague schematic mappings of the qualitative information parts *excluded* (E), *unlikely* (U), *not excluded* (N), *likely* (L), and *confirmed* (C) to *confidence intervals*, as used in this paper. Figure taken from [15].

These qualitative parts were used in statements about, e.g., liver inflammation with varying levels of detail. From this, we want to infer that, e.g., **if** *Hepatitis is confirmed* **then** *Hepatitis is likely* but **not** *Hepatitis is unlikely*. And **if** *Viral Hepatitis B is confirmed*, **then** both *Viral Hepatitis is confirmed* **and** *Hepatitis is confirmed* (generalization). Things "turn around" when we look

at the adjectival modifiers *excluded* and *unlikely*: **if** *Hepatitis is excluded* **then** *Hepatitis is unlikely*, but **not** *Hepatitis is not excluded*. Furthermore, **if** *Hepatitis is excluded*, **then** both *Viral Hepatitis is excluded* **and** *Viral Hepatitis B is excluded* (specialization). The set of *plausible* entailments for this kind of graded reasoning is depicted in Fig. 2.

		Being said to have hepatitis (H) / viral hepatitis (vH) / viral hepatitis B (vHB) is...														
Precondition:		confirmed			likely			not excluded			unlikely			excluded		
Entailment:		H	vH	vHB	H	vH	vHB	H	vH	vHB	H	vH	vHB	H	vH	vHB
confirmed	H	x	x	x												
	vH		x	x												
	vHB			x												
likely	H	x	x	x	x	x	x									
	vH		x	x		x	x									
	vHB			x			x									
not excluded	H	x	x	x	x	x	x	x	x	x						
	vH		x	x		x	x		x	x						
	vHB			x			x			x						
unlikely	H										x			x		
	vH										x	x		x	x	
	vHB										x	x	x	x	x	x
excluded	H													x		
	vH													x	x	
	vHB													x	x	x

Fig. 2. Statements about liver inflammation with varying levels of detail: *Viral Hepatitis B* (vHB) implies *Viral Hepatitis* (vH) which implies *Hepatitis* (H). The matrix depicts entailments considered plausible, based on the inferences that follow from Fig. 1. Hepatitis and its subclasses can be easily replaced by other medical situations/diseases. Figure taken from [15].

[15] consider five encodings (one outside the expressivity of OWL), from which only two were able to fully reproduce the inferences from Fig. 2. Let us quickly look on approach 1, called *existential restriction*, before we informally present its modal counterpart (we will use abstract description logic syntax here [2]):

$$\mathsf{HepatitisSituation} \equiv \mathsf{ClinicalSituation} \sqcap \exists \mathsf{hasCondition.Hepatitis}$$

% *Hepatitis subclass hierarchy*

$$\mathsf{ViralHepatitisB} \sqsubseteq \mathsf{ViralHepatitis} \sqsubseteq \mathsf{Hepatitis}$$

% *vagueness via two subclass hierarchies*

$$\mathsf{IsConfirmed} \sqsubseteq \mathsf{IsLikely} \sqsubseteq \mathsf{IsNotExcluded} \qquad \mathsf{IsExcluded} \sqsubseteq \mathsf{IsUnlikely}$$

% *a diagnostic statement about Hepatitis*

$$\mathsf{BeingSaidToHaveHepatitisIsConfirmed} \equiv \mathsf{DiagnosticStatement} \sqcap \forall \mathsf{hasCertainty.IsConfirmed} \sqcap \exists \mathsf{isAboutSituation.HepatitisSituation}$$

Standard OWL reasoning under this representation then ensures that, for instance,

$$\mathsf{BeingSaidToHaveHepatitsIsConfirmed} \sqsubseteq \mathsf{BeingSaidToHaveHepatitisIsLikely}$$

is the case, exactly one of the plausible inferences from Fig. 2.

The encodings in [15] were quite cumbersome as the primary interest was to stay within the limits of the underlying calculus (OWL). Besides coming up with complex encodings, only minor forms of reasoning were possible, viz., subsumption reasoning. These disadvantages are a result of two conscious decisions: OWL only provides unary and binary relations (concepts and roles) and comes up with a (mostly) fixed set of entailment/tableaux rules.

In our approach, however, the *qualitative information parts* from Fig. 1 are first class citizens of the object language (the modal operators) and *diagnostic statements* from the Hepatitis use case are expressed through the binary property $\mathsf{suffersForm}$ between p (patients, people) and d (diseases, diagnoses). The plausible inferences are then simply a *byproduct* of the *instantiation* of the entailment rule schemas (G) from Sect. 5.1, and $(\mathsf{S1})$ and $(\mathsf{S0})$ from Sect. 5.2 for property $\mathsf{suffersForm}$ (the rule variables are universally quantified; $\top$ = *universal truth*; C = *confirmed*; L = *likely*), e.g.,

$$(\mathsf{S1}) \top\mathsf{ViralHepatitisB}(d) \wedge \mathsf{ViralHepatitisB} \sqsubseteq \mathsf{ViralHepatitis} \rightarrow \top\mathsf{ViralHepatitis}(d)$$
$$(\mathsf{G}) C\mathsf{suffersFrom}(p, d) \rightarrow L\mathsf{suffersFrom}(p, d)$$

Two things are worth to be mentioned here. *Firstly*, not only OWL-like properties (binary relations) can be graded, such as $C\mathsf{suffersFrom}(p, d)$ (= *it is confirmed that p suffers from d*), but also class membership (unary relations), e.g., $C\mathsf{ViralHepatitisB}(d)$ (= *it is confirmed that d is Viral Hepatitis B*). However, as the original OWL example above is unable to make use of any modals, we employ a special modal $\top$ here:$\top\mathsf{ViralHepatitisB}(d)$. *Secondly*, modal operators are only applied to assertional knowledge, involving individuals (the ABox in OWL)—neither axioms about classes (TBox) nor properties (RBox) are being affected by modals, as they are supposed to express universal truth.

3 Confidence of Statements and Confidence Intervals

We address the *confidence* of an asserted medical statement [15] through *graded* modalities applied to propositional formulae: E (*excluded*), U (*unlikely*), N (*not excluded*), L (*likely*), and C (*confirmed*). For various (technical) reasons, we add a *wildcard* modality? (*unknown*), a complementary *failure* modality! (*error*), plus two further modalities to syntactically state definite truth and falsity: $\top$ (*true*) and $\bot$ (*false*). Let $\triangle$ now denotes the set of all modalities:

$$\triangle = \{?, !, \top, \bot, E, U, N, L, C\}$$

A *measure function*

$$\mu : \triangle \mapsto [0, 1] \times [0, 1]$$

is a mapping which returns the associated *confidence interval* $[l, h]$ for a modality from $\triangle$ ($l \leq h$). We presuppose that[1]

- $\mu(?) = [0, 1]$
- $\mu(!) = \emptyset$
- $\mu(\top) = [1, 1]$
- $\mu(\bot) = [0, 0]$

In addition, we define two disjoint subsets of $\triangle$, called

- $\mathsf{1} = \{\top, C, L, N\}$
- $\mathsf{0} = \{\bot, E, U\}$

and again make a presupposition: the confidence intervals for modals from $\mathsf{1}$ *end* in 1, whereas the confidence intervals for $\mathsf{0}$ modals always *start* with 0. It is worth noting that we do *not* make use of μ in the syntax of the modal language (for which we employ the modalities from $\triangle$), but in the semantics when dealing with the satisfaction relation of the model theory (see Sect. 4).

We have talked about *confidence intervals* now several times without saying what we actually mean by this. Suppose that a physician says that it is *confirmed* ($= C$) that patient p suffers from disease d, given a set of recognized symptoms $S = \{s_1, \ldots, s_k\}$: $C\mathit{suffersFrom}(p, d)$.

Assuming that a different patient p' shows the same symptoms S (and only S, and perhaps further symptoms which are, however, *independent* from S), we would assume that the same doctor would diagnose $C\mathit{suffersFrom}(p', d)$.

Even an other, but similar trained physician is supposed to grade the two patients *similarly*. This similarity which originates from patients showing the same symptoms and from physicians being taught at the same medical school is addressed by confidence *intervals* and not through a *single* (posterior) probability, as there are still variations in diagnostic capacity and daily mental state of the physician. By using intervals (instead of single values), we can usually reach a consensus among people upon the *meaning* of gradation words, even though the low/high values of the confidence interval for, e.g., *confirmed* might depend on the context.

Being a bit more theoretic, we define a *confidence interval* as follows. Assume a *Bernoulli experiment* [11] that involves a large set of n patients P sharing the same symptoms S. W.r.t. our example, we would like to know whether $\mathit{suffersFrom}(p, d)$ or $\neg\mathit{suffersFrom}(p, d)$ is the case for every patient $p \in P$, sharing S. Given a Bernoulli trials sequence $\boldsymbol{X} = \langle x_1, \ldots, x_n \rangle$ with indicator random variables $x_i \in \{0, 1\}$ for a patient sequence $\langle p_1, \ldots, p_n \rangle$, we can *approximate* the *expected value* E for *suffersFrom* being *true*, given disease d and background symptoms S by the *arithmetic mean* A:

$$\mathrm{E}[\boldsymbol{X}] \approx \mathrm{A}[\boldsymbol{X}] = \frac{\sum_{i=1}^{n} x_i}{n}$$

Due to the *law of large numbers*, we expect that if the number of elements in a trials sequence goes to infinity, the arithmetic mean will coincide with the expected value:

$$\mathrm{E}[\boldsymbol{X}] = \lim_{n \to \infty} \frac{\sum_{i=1}^{n} x_i}{n}$$

[1] Recall that an interval is a set of real numbers, together with a total ordering relation (e.g., $\leq$) over the elements, thus $\emptyset$ is a perfect, although degraded interval.

Clearly, the arithmetic mean for each new *finite* trials sequence is different, but we can try to *locate* the expected value within an interval around the arithmetic mean:

$$\mathrm{E}[\boldsymbol{X}] \in [\mathrm{A}[\boldsymbol{X}] - \epsilon_1, \mathrm{A}[\boldsymbol{X}] + \epsilon_2]$$

For the moment, we assume $\epsilon_1 = \epsilon_2$, so that $\mathrm{A}[\boldsymbol{X}]$ is in the center of this interval which we will call from now on *confidence interval*.

Coming back to our example and assuming $\mu(C) = [0.9, 1]$, $C\mathit{suffersFrom}(p, d)$ can be read as being true in 95 % of all cases *known* to the physician, involving patients p potentially having disease d and sharing the same prior symptoms (evidence) $s_1, \ldots, s_k$:

$$\frac{\sum_{p \in P} \mathrm{Prob}(\mathit{suffersFrom}(p, d) | s_1, \ldots, s_k)}{n} \approx 0.95$$

The variance of $\pm 5\,\%$ is related to varying diagnostic capabilities between (comparative) physicians, daily mental form, undiscovered important symptoms or examinations which have not been carried out (e.g., lab values), or perhaps even the physical stature of the patient which unconsciously affects the final diagnosis, etc., as elaborated above. Thus the individual modals from $\triangle$ express (via μ) different forms of the physician's confidence, depending on the set of already acquired symptoms as (potential) explanations for a specific disease.

4 Model Theory and Negation Normal Form

Let $\mathcal{C}$ denote the set of constants that serve as the arguments of a relation instance. In order to define basic n-ary propositional formulae (ground atoms, propositional letters), let $p(\boldsymbol{c})$ abbreviates $p(c_1, \ldots, c_n)$, for some $c_1, \ldots, c_n \in \mathcal{C}$, given $\mathit{length}(\boldsymbol{c}) = n$. In case the number of arguments do not matter, we sometimes simply write p, instead of, e.g., $p(c, d)$ or $p(\boldsymbol{c})$. As before, we assume $\triangle = \{?, !, \top, \bot, E, U, N, L, C\}$. We inductively define the set of *well-formed formulae* ϕ of our modal language as follows:

$$\phi ::= p(\boldsymbol{c}) \mid \neg\phi \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \triangle\phi$$

4.1 Simplification and Normal Form

We now syntactically *simplify* the set of well-formed formulae ϕ by restricting the uses of *negation* and *modalities* to the level of propositional letters p and call the resulting language Λ:

$$\begin{aligned} \pi &::= p(\boldsymbol{c}) \mid \neg p(\boldsymbol{c}) \\ \phi &::= \pi \mid \triangle\pi \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \end{aligned}$$

To do so, we need the notion of a *complement* modal δ^{C} for every $\delta \in \triangle$, where

$$\mu(\delta^{\mathsf{C}}) := \mu(\delta)^{\mathsf{C}} = \mu(?) \setminus \mu(\delta) = [0, 1] \setminus \mu(\delta)$$

I.e., $\mu(\delta^{\mathsf{C}})$ is defined as the complementary interval of $\mu(\delta)$ (within the bounds of $[0,1]$, of course). For example, E and N (*excluded, not excluded*) or ? and ! (*unknown, error*) are already existing complementary modals. We also require *mirror* modals δ^{M} for every $\delta \in \triangle$ whose confidence interval $\mu(\delta^{\mathsf{M}})$ is derived by "mirroring" $\mu(\delta)$ to the opposite site of the confidence interval, either to the left or to the right:

$$\textbf{if } \mu(\delta) = [l,h] \textbf{ then } \mu(\delta^{\mathsf{M}}) := [1-h, 1-l]$$

For example, E and C (*excluded, confirmed*) or $\top$ and $\bot$ (*top, bottom*) are mirror modals. In order to transform ϕ into its *negation normal form*, we need to apply simplification rules a finite number of times (until rules are no longer applicable). We depict those rules by using the $\vdash$ relation, read as *formula* $\vdash$ *simplified formula*:

1. $?\phi \vdash \epsilon$ % *?ϕ is not informative at all, but its existence should alarm us*
2. $\neg\neg\phi \vdash \phi$
3. $\neg(\phi \wedge \phi') \vdash \neg\phi \vee \neg\phi'$
4. $\neg(\phi \vee \phi') \vdash \neg\phi \wedge \neg\phi'$
5. $\neg\triangle\phi \vdash \triangle^{\mathsf{C}}\phi$ (example: $\neg E\phi = N\phi$)
6. $\triangle\neg\phi \vdash \triangle^{\mathsf{M}}\phi$ (example: $E\neg\phi = C\phi$)

Clearly, the mirror modals δ^{M} are not necessary as long as we explicitly allow for negated statements, and thus case 6 can, in principle, be dropped.

What is the result of simplifying $\triangle(\phi \wedge \phi')$ and $\triangle(\phi \vee \phi')$? Let us start with the former case and consider as an example the statement about an engine that *a mechanical failure m and an electrical failure e is confirmed*: $C(m \wedge e)$. It seems plausible to simplify this expression to $Cm \wedge Ce$. Commonsense tells us furthermore that neither Em nor Ee is compatible with this description.

Now consider the "opposite" statement $E(m \wedge e)$ which must *not* be rewritten to $Em \wedge Ee$, as *either Cm or Ce* is well *compatible* with $E(m \wedge e)$. Instead, we rewrite this kind of "negated" statement as $Em \vee Ee$, and this works fine with either Cm or Ce.

In order to address the other modal operators, we generalize these plausible inferences by making a distinction between $\mathsf{0}$ and $\mathsf{1}$ modals (see Sect. 3):

7a. $\mathsf{0}(\phi \wedge \phi') \vdash \mathsf{0}\phi \vee \mathsf{0}\phi'$
7b. $\mathsf{1}(\phi \wedge \phi') \vdash \mathsf{1}\phi \wedge \mathsf{1}\phi'$

Now let us consider disjunction inside the scope of a modal operator. As we do allow for the full set of Boolean operators, we are allowed to deduce

8. $\triangle(\phi \vee \phi') \vdash \triangle(\neg(\neg(\phi \vee \phi'))) \vdash \triangle(\neg(\neg\phi \wedge \neg\phi')) \vdash \triangle^{\mathsf{M}}(\neg\phi \wedge \neg\phi')$

This is, again, a conjunction, so we apply schemas 7a and 7b, giving us

8a. $\mathsf{0}(\phi \vee \phi') \vdash \mathsf{0}^{\mathsf{M}}(\neg\phi \wedge \neg\phi') \vdash \mathsf{1}(\neg\phi \wedge \neg\phi') \vdash \mathsf{1}\neg\phi \wedge \mathsf{1}\neg\phi' \vdash \mathsf{1}^{\mathsf{M}}\phi \wedge \mathsf{1}^{\mathsf{M}}\phi' \vdash \mathsf{0}\phi \wedge \mathsf{0}\phi'$
8b. $\mathsf{1}(\phi \vee \phi') \vdash \mathsf{1}^{\mathsf{M}}(\neg\phi \wedge \neg\phi') \vdash \mathsf{0}(\neg\phi \wedge \neg\phi') \vdash \mathsf{0}\neg\phi \vee \mathsf{0}\neg\phi' \vdash \mathsf{0}^{\mathsf{M}}\phi \vee \mathsf{0}^{\mathsf{M}}\phi' \vdash \mathsf{1}\phi \vee \mathsf{1}\phi'$

Note how the modals from $\mathsf{0}$ in 7a and 8a act as a kind of *negation* to turn the logical operators into their counterparts, similar to de Morgan's law.

4.2 Model Theory

In the following, we extend the standard definition of modal (Kripke) frames and models [3] for the *graded* modal operators from $\triangle$ by employing the measure function μ and focussing on the minimal definition for ϕ in Λ. A *frame* $\mathcal{F}$ for the probabilistic modal language Λ is a pair

$$\mathcal{F} = \langle \mathcal{W}, \triangle \rangle$$

where $\mathcal{W}$ is a non-empty set of *worlds* (or *situations, states, points, vertices*) and $\triangle$ a family of binary relations over $\mathcal{W} \times \mathcal{W}$, called *accessibility relations*. Note that we have overloaded $\triangle$ (and each $\delta \in \triangle$) in that it refers to the modals used in the syntax of Λ, but also to depict the binary relations, connecting worlds.

A *model* $\mathcal{M}$ for the probabilistic modal language Λ is a triple

$$\mathcal{M} = \langle \mathcal{F}, \mathcal{V}, \mu \rangle$$

such that $\mathcal{F}$ is a *frame*, $\mathcal{V}$ a *valuation*, assigning each proposition ϕ a subset of $\mathcal{W}$, viz., the set of worlds in which ϕ holds, and μ a mapping, returning the confidence interval for a given modality from $\triangle$. Note that we only require a definition for μ in $\mathcal{M}$ (the model, but *not* in the frame), as $\mathcal{F}$ represent the relational structure without interpreting the edge labeling (the modal names) of the graph.

The *satisfaction relation* $\models$, given a model $\mathcal{M}$ and a specific world w is inductively defined over the set of well-formed formulae of Λ in *negation normal form* (remember $\pi ::= p(\boldsymbol{c}) \mid \neg p(\boldsymbol{c})$):

1. $\mathcal{M}, w \models p(\boldsymbol{c})$ **iff** $w \in \mathcal{V}(p(\boldsymbol{c}))$ **and** $w \notin \mathcal{V}(\neg p(\boldsymbol{c}))$
2. $\mathcal{M}, w \models \neg p(\boldsymbol{c})$ **iff** $w \in \mathcal{V}(\neg p(\boldsymbol{c}))$ **and** $w \notin \mathcal{V}(p(\boldsymbol{c}))$
3. $\mathcal{M}, w \models \phi \wedge \phi'$ **iff** $\mathcal{M}, w \models \phi$ **and** $\mathcal{M}, w \models \phi'$
4. $\mathcal{M}, w \models \phi \vee \phi'$ **iff** $\mathcal{M}, w \models \phi$ **or** $\mathcal{M}, w \models \phi'$
5. **for all** $\delta \in \triangle$: $\mathcal{M}, w \models \delta\pi$ **iff** $\frac{\#\{u \mid (w,u) \in \delta \textbf{ and } \mathcal{M}, u \models \pi\}}{\#\{u \mid (w,u) \in \delta' \textbf{ and } \delta' \in \triangle\}} \in \mu(\delta)$

The last case of the satisfaction relation addresses the modals: for a world w, we look for the successor states u that are directly reachable via δ and in which π holds, and divide the number of such states by the number of all worlds that are directly reachable from w. This number between 0 and 1 must lie in the confidence interval $\mu(\delta)$ of δ in order to satisfy $\delta\pi$, given $\mathcal{M}, w$.

It is worth noting that the satisfaction relation above differs in its handling of $\mathcal{M}, w \models \neg p(\boldsymbol{c})$, as negation is *not* interpreted through the *absence* of $p(\boldsymbol{c})$ ($\mathcal{M}, w \not\models p(\boldsymbol{c})$), but through the *existence* of $\neg p(\boldsymbol{c})$. This treatment addresses the *open-world* nature in OWL and the evolvement of a (medical) domain over time.

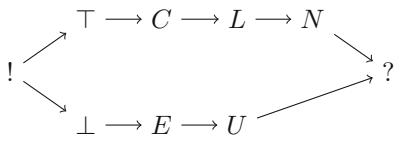
We also note that the definition of the satisfaction relation for modalities (last clause) is related to the *possibility operators* $M_k \cdot$ ($= \Diamond^{\geq k} \cdot$; $k \in \mathbb{N}$) [4] and *counting modalities* $\cdot \geq n$ [1], used in modal logic characterizations of *description logics* with *cardinality* restrictions.

4.3 Well-Behaved Frames

As we will see later, it is handy to assume that the graded modals are arranged in a kind of hierarchy—the more we move "upwards" in the hierarchy, the more a statement in the scope of a modal becomes *uncertain*. In order to address this, we slightly extend the notion of a *frame* by a third component $\preceq \subseteq \triangle \times \triangle$, a partial order between modalities:

$$\mathcal{F} = \langle \mathcal{W}, \triangle, \preceq \rangle$$

Let us consider the following modal hierarchy that we build from the set $\triangle$ of already introduced modals:



This graphical representation is just a compact way to specify a set of 33 binary relation instances over $\triangle$, such as, e.g., $\top \preceq \top$, $\top \preceq N$, $C \preceq N$, $\bot \preceq ?$, or $! \preceq ?$. The above mentioned form of uncertainty is expressed by the measure function μ in that the associated confidence intervals become larger:

$$\textbf{if } \delta \preceq \delta' \textbf{ then } \mu(\delta) \subseteq \mu(\delta')$$

In order to arrive at a proper and intuitive model-theoretic semantics which mirrors intuitions such as **if** ϕ *is confirmed* $(C\phi)$ **then** ϕ *is likely* $(L\phi)$, we will focus here on *well-behaved* frames $\mathcal{F}$ which enforce the existence of edges in $\mathcal{W}$, given $\preceq$ and $\delta, \delta^{\uparrow} \in \triangle$:

$$\textbf{if } (w,u) \in \delta \textbf{ and } \delta \preceq \delta^{\uparrow} \textbf{ then } (w,u) \in \delta^{\uparrow}$$

However, by imposing this constraint, we also need to adapt the last case of the satisfiability relation:

5. **for all** $\delta \in \triangle$: $\mathcal{M}, w \models \delta\pi$ **iff** $\frac{\#\{u \mid (w,u) \in \delta^{\uparrow}, \delta \preceq \delta^{\uparrow}, \textbf{ and } \mathcal{M}, u \models \pi\}}{\#\{u \mid (w,u) \in \delta' \textbf{ and } \delta' \in \triangle\}} \in \mu(\delta)$

Not only are we scanning for edges (w,u) labeled with δ and for successor states u of w in which π holds in the denominator (original definition), but also take into account edges marked with more general modals $\delta^{\uparrow}$, s.t. $\delta^{\uparrow} \succeq \delta$. This mechanism implements a kind of *built-in model completion* that is not necessary in ordinary modal logics as they deal with only a *single* relation (viz., unlabeled arcs) that connects elements from $\mathcal{W}$ and the two modals $\Diamond$ and $\Box$ are defined in the usual dual way: $\Box\phi \equiv \neg\Diamond\neg\phi$.

5 Entailment Rules

This section addresses a restricted subset of entailment rules which will unveil new (or implicit) knowledge from graded medical statements. Recall that these kind of statements (in negation normal form) are a consequence of the application of simplification rules as depicted in Sect. 4.1. Thus, we assume a *pre-processing step* here that "massages" more complex statements that arise from a representation of graded (medical) statements in *natural language*. The entailments which we will present in a moment can either be *directly* implemented in a tuple-based reasoner, such as *HFC*, or in triple-based engines (e.g., Jena, OWLIM) which need to *reify* the medical statements in order to be compliant with the RDF triple model.

5.1 Modal Entailments

The entailments presented in this section deal with *plausible* inference centered around modals $\delta, \delta' \in \triangle$, some of them partly addressed in [15] in a pure OWL setting. We use the implication sign $\rightarrow$ to depict the entailment rules:

$$lhs \rightarrow rhs$$

which act as *completion* (or *materialization*) rules the way as described in, e.g., [6,18], and used in today's semantic repositories. We sometimes even use the bi-conditional $\leftrightarrow$ to address that the LHS and the RHS are semantically equivalent, but will indicate the direction that should be used in a practical setting. As before, we define $\pi ::= p(\boldsymbol{c}) \mid \neg p(\boldsymbol{c})$.

We furthermore assume that for every modal $\delta \in \triangle$, a *complement* modal δ^{C} and a *mirror* modal δ^{M} exist (see Sect. 4.1).

Lift

$$(\mathsf{L}) \quad \pi \leftrightarrow \top\pi$$

This rule interprets propositional statements as special modal formulae. It might be dropped and can be seen as a pre-processing step. We have used it in the Hepatitis example above. Usage: left-to-right direction.

Generalize

$$(\mathsf{G}) \quad \delta\pi \wedge \delta \preceq \delta' \rightarrow \delta'\pi$$

This rule schema can be instantiated in various ways, using the modal hierarchy from Sect. 4.3; e.g., $\top\pi \rightarrow C\pi$, $C\pi \rightarrow L\pi$, or $E\pi \rightarrow U\pi$. It has been used in the Hepatitis example.

Complement

$$(\mathsf{C}) \quad \neg\delta\pi \leftrightarrow \delta^{\mathsf{C}}\pi$$

In principle, (C) is not needed in case the statement is already in negation normal form. This schema might be useful for natural language paraphrasing (explanation). Given $\triangle$, there are two possible instantiations, viz., $E\pi \leftrightarrow \neg N\pi$ and $N\pi \leftrightarrow \neg E\pi$ (note: $\mu(E) \cup \mu(N) = [0,1]$).

Mirror

$$(\mathsf{M}) \quad \delta\neg\pi \leftrightarrow \delta^{\mathsf{M}}\pi$$

Again, (M) is in principle not needed as long as the modal proposition is in negation normal form, since we do allow for negated propositional statements $\neg p(\boldsymbol{c})$. This schema might be useful for natural language paraphrasing (explanation). For $\triangle$, there are six possible instantiations, viz., $E\pi \leftrightarrow C\neg\pi$, $C\pi \leftrightarrow E\neg\pi$, $L\pi \leftrightarrow U\neg\pi$, $U\pi \leftrightarrow L\neg\pi$, $\top\pi \leftrightarrow \bot\neg\pi$, and $\bot\pi \leftrightarrow \top\neg\pi$.

Uncertainty

$$(\mathsf{U}) \quad \delta\pi \wedge \neg\delta\pi \leftrightarrow \delta\pi \wedge \delta^{\mathsf{C}}\pi \leftrightarrow ?\pi$$

The *co-occurrence* of $\delta\pi$ and $\neg\delta\pi$ does *not* imply logical *inconsistency* (propositional case: $\pi \wedge \neg\pi$), but leads to complete *uncertainty* about the validity of π. Remember that $\mu(?) = \mu(\delta) \cup \mu(\delta^{\mathsf{C}}) = [0,1]$ (usage: left-to-right direction):

$$\begin{array}{cccccc} & 0 & & & & 1 \\ \mu : & | & \text{—}\delta^{\mathsf{C}}\text{—} & | & \text{——}\delta\text{——} & | \\ & & \pi & & \pi & \end{array}$$

Negation

$$(\mathsf{N}) \quad \delta(\pi \wedge \neg\pi) \leftrightarrow \delta\pi \wedge \delta\neg\pi \leftrightarrow \delta\pi \wedge \delta^{\mathsf{M}}\pi \leftrightarrow \delta^{\mathsf{M}}\neg\pi \wedge \delta^{\mathsf{M}}\pi \leftrightarrow \delta^{\mathsf{M}}(\pi \wedge \neg\pi)$$

(N) shows that $\delta(\pi \wedge \neg\pi)$ can be formulated equivalently using the mirror modal:

$$\begin{array}{cccccc} & 0 & & & & 1 \\ \mu : & | & \text{—}\delta^{\mathsf{M}}\text{—} & | & \text{——}|\text{—}\,\delta\,\text{—} & | \\ & & \pi \wedge \neg\pi & & \pi \wedge \neg\pi & \end{array}$$

In general, (N) is *not* the modal counterpart of the *law of non-contradiction*, as $\pi \wedge \neg\pi$ is usually afflicted by vagueness, meaning that from $\delta(\pi \wedge \neg\pi)$, we can *not* infer that $\pi \wedge \neg\pi$ is the case for the concrete example in question (recall the intention behind the confidence intervals; see Sect. 3). There is one notable exception, involving the $\top$ and $\bot$ modals. This is formulated by the next entailment rule.

Error

$$(\mathsf{E}) \quad \top(\pi \wedge \neg\pi) \leftrightarrow \bot(\pi \wedge \neg\pi) \rightarrow !(\pi \wedge \neg\pi)$$

(E) *is* the modal counterpart of the *law of non-contradiction* (recall: $\top = \bot^{\mathsf{M}}$ and $\bot = \top^{\mathsf{M}}$). For this reason and by definition, the *error* (or *failure*) modal ! from Sect. 3 comes into play here. The modal ! can serve as a hint to either stop a computation the first time it occurs or to continue reasoning, but to syntactically memorize the ground atoms (viz., π and $\neg\pi$) which have led to an inconsistency. Usage: left-to-right direction.

5.2 Subsumption Entailments

As before, we define two subsets of $\triangle$, called $\mathsf{1} = \{\top, C, L, N\}$ and $\mathsf{0} = \{\bot, E, U\}$, thus 1 and 0 effectively become

$$\mathsf{1} = \{\top, C, L, N, U^{\mathsf{C}}\} \qquad \mathsf{0} = \{\bot, U, E, C^{\mathsf{C}}, L^{\mathsf{C}}, N^{\mathsf{M}}\}$$

due to the use of complement modals δ^{C} and mirror modals δ^{M} for every base modal $\delta \in \triangle$ and by assuming that $E = N^{\mathsf{C}}$, $E = C^{\mathsf{M}}$, $U = L^{\mathsf{M}}$, and $\bot = \top^{\mathsf{M}}$, together with the four "opposite" cases.

Now let $\sqsubseteq$ abbreviate relation subsumption as known from description logics and realized in OWL through `rdfs:subClassOf` (class subsumption) and `rdfs:subPropertyOf` (property subsumption). Given these remarks, we define two further very practical and plausible modal entailments which can be seen as the modal extension of the entailment rules (rdfs9) (for classes) and (rdfs7) (for properties) in RDFS; see [6].

$$(\mathsf{S1}) \quad \mathsf{1}p(\boldsymbol{c}) \wedge p \sqsubseteq q \rightarrow \mathsf{1}q(\boldsymbol{c}) \qquad (\mathsf{S0}) \quad \mathsf{0}q(\boldsymbol{c}) \wedge p \sqsubseteq q \rightarrow \mathsf{0}p(\boldsymbol{c})$$

Note how the use of p and q switches in the antecedent and the consequent, even though $p \sqsubseteq q$ holds in both cases. Note further that propositional statements π are restricted to the positive case $p(\boldsymbol{c})$ and $q(\boldsymbol{c})$, as their negation in the antecedent will not lead to any valid entailments. Here are four *instantiations* of (S0) and (S1) (remember, $C \in \mathsf{1}$ and $E \in \mathsf{0}$):

$$C\mathsf{ViralHepatitisB}(x) \wedge \mathsf{ViralHepatitisB} \sqsubseteq \mathsf{ViralHepatitis} \rightarrow C\mathsf{ViralHepatitis}(x)$$
$$E\mathsf{Hepatitis}(x) \wedge \mathsf{ViralHepatitis} \sqsubseteq \mathsf{Hepatitis} \rightarrow E\mathsf{ViralHepatitis}(x)$$
$$C\mathsf{deeplyEnclosedIn}(x, y) \wedge \mathsf{deeplyEnclosedIn} \sqsubseteq \mathsf{containedIn} \rightarrow C\mathsf{containedIn}(x, y)$$
$$E\mathsf{containedIn}(x, y) \wedge \mathsf{superficiallyLocatedIn} \sqsubseteq \mathsf{containedIn}$$
$$\rightarrow E\mathsf{superficiallyLocatedIn}(x, y)$$

5.3 Extended RDFS and OWL Entailments

In this section, we will consider some of the entailment rules for RDFS [6] and a restricted subset of OWL [18]. Remember that modals only head literals π, neither TBox nor RBox axioms. Concerning the original entailment rules, we will distinguish *four principal cases* to which the extended rules belong (we will only consider the unary and binary case here as used in description logics/OWL):

1. TBox and RBox axiom schemas will not undergo a modal extension;
2. rules get extended in the antecedent;
3. rules take over the modal from the antecedent to the consequent;
4. rules aggregate several modals from the antecedent in the consequent.

We will illustrate the individual cases in the following subsections with examples by using a kind of description logic syntax. Clearly, the set of extended entailments depicted here is *not complete*.

Case-1 Rules: No Modals. Entailment rule rdfs11 from [6] deals with class subsumption: $\mathsf{C} \sqsubseteq \mathsf{D} \wedge \mathsf{D} \sqsubseteq \mathsf{E} \rightarrow \mathsf{C} \sqsubseteq \mathsf{E}$. As this is a terminological axiom schema, the rule stays *constant* in the modal domain. Example:

$$\mathsf{ViralHepatitisB} \sqsubseteq \mathsf{ViralHepatitis} \wedge \mathsf{ViralHepatitis} \sqsubseteq \mathsf{Hepatitis}$$
$$\rightarrow \mathsf{ViralHepatitisB} \sqsubseteq \mathsf{Hepatitis}$$

Case-2 Rules: Modals on LHS, No or $\top$ Modals on RHS. The following original rule rdfs3 from [6] imposes a range restriction on objects of binary ABox relation instances: $\forall \mathsf{P}.\mathsf{C} \wedge \mathsf{P}(\mathsf{x},\mathsf{y}) \rightarrow \mathsf{C}(\mathsf{y})$.

The extended version (which we call Mrdfs3) needs to address the proposition in the antecedent, but must not change the consequent (even though we always use the $\top$ modality here for typing; see Sect. 2):

$$(\mathsf{Mrdfs3}) \quad \forall \mathsf{P}.\mathsf{C} \wedge \delta\mathsf{P}(\mathsf{x},\mathsf{y}) \rightarrow \top\mathsf{C}(\mathsf{y})$$

Example: $\forall \mathsf{suffersFrom}.\mathsf{Disease} \wedge L\mathsf{suffersFrom}(x, y) \rightarrow \top\mathsf{Disease}(y)$

Case-3 Rules: Keeping LHS Modals on RHS. Inverse properties switch their arguments [18]: $\mathsf{P} \equiv \mathsf{Q}^- \wedge \mathsf{P}(\mathsf{x},\mathsf{y}) \rightarrow \mathsf{Q}(\mathsf{y},\mathsf{x})$.

The extended version of rdfp8 simply keeps the modal operator:

$$(\mathsf{Mrdfp8}) \quad \mathsf{P} \equiv \mathsf{Q}^- \wedge \delta\mathsf{P}(\mathsf{x},\mathsf{y}) \rightarrow \delta\mathsf{Q}(\mathsf{y},\mathsf{x})$$

Example: $\mathsf{containedIn} \equiv \mathsf{contains}^- \wedge C\mathsf{containedIn}(x, y) \rightarrow C\mathsf{contains}(y, x)$

Case-4 Rules: Aggregating LHS Modals on RHS. Now comes the most interesting case of modalized RDFS/OWL entailment rules that offers several possibilities on a varying scale between *skeptical* and *credulous* entailments, depending on the degree of uncertainty, as expressed by the measuring function μ of the modal operator. Consider the original rule rdfp4 from [18] for transitive properties P: $\mathsf{P}^+ \sqsubseteq \mathsf{P} \wedge \mathsf{P}(\mathsf{x},\mathsf{y}) \wedge \mathsf{P}(\mathsf{y},\mathsf{z}) \rightarrow \mathsf{P}(\mathsf{x},\mathsf{z})$.

How does the modal on the RHS of the extended rule look like, depending on the two LHS modals? There are several possibilities. By operating directly on the *modal hierarchy*, we are allowed to talk about, e.g., the *least upper bound* or the *greatest lower bound* of δ and δ'. When taking the associated *confidence intervals* into account, we might even play with the low and high number of

the intervals, say, by applying the *arithmetic mean* or simply by *multiplying* the corresponding numbers.

Let us first consider the general rule from which more specialized versions can be derived, simply by instantiating the combination operator $\odot$:

$$(\mathsf{Mrdfp4}) \quad \mathsf{P}^{+} \sqsubseteq \mathsf{P} \wedge \delta\mathsf{P}(\mathsf{x},\mathsf{y}) \wedge \delta'\mathsf{P}(\mathsf{y},\mathsf{z}) \rightarrow (\delta \odot \delta')\mathsf{P}(\mathsf{x},\mathsf{z})$$

Here is an instantiation of Mrdfp4 dealing with the transitive relation contains from above: $C\mathsf{contains}(x,y) \wedge L\mathsf{contains}(y,z) \rightarrow (C \odot L)\mathsf{contains}(x,z)$.

What is the result of $C \odot L$ here? It depends. Probably both on the application domain and the epistemic commitment one is willing to accept about the "meaning" of gradation words/modal operators. To enforce that $\odot$ is at least both *commutative* and *associative* is probably a good idea, making the sequence of modal clauses order-independent.

5.4 Custom Entailments

Custom entailments are inference rules that are not derived from universal non-modalized RDFS and OWL entailment rules (Sect. 5.3), but have been formulated to capture the domain knowledge of experts (e.g., physicians). Here is an example. Consider that Hepatitis B is an infectious disease:

$$\mathsf{ViralHepatitisB} \sqsubseteq \mathsf{InfectiousDisease} \sqsubseteq \mathsf{Disease}$$

and note that there exist vaccines against it. Assume that the liver l of patient p quite hurts (modal C), but p has been definitely vaccinated (modal $\top$) against Hepatitis B before:

$$C\mathsf{hasPain}(p,l) \wedge \top\mathsf{vaccinatedAgainst}(p,\mathsf{ViralHepatitisB})$$

Given that p received a vaccination, the following custom rule will *not* fire (x and y below are now universally-quantified variables; z an existentially-quantified RHS-only variable):

$$\top\mathsf{Patient}(x) \wedge \top\mathsf{Liver}(y) \wedge C\mathsf{hasPain}(x,y) \wedge U\mathsf{vaccinatedAgainst}(x,\mathsf{ViralHepatitisB})$$
$$\rightarrow N\mathsf{ViralHepatitisB}(z) \wedge N\mathsf{suffersFrom}(x,z)$$

Now assume another person p' that is pretty sure (s)he was never vaccinated:

$$E\mathsf{vaccinatedAgainst}(p',\mathsf{ViralHepatitisB})$$

Given the above custom rule, we are allowed to infer that (h instantiation of z)

$$N\mathsf{ViralHepatitisB}(h) \wedge N\mathsf{suffersFrom}(p',h)$$

The subclass axiom from above thus assigns

$$N\mathsf{InfectiousDisease}(h)$$

so that we can query for patients for whom an infectious disease is *not unlikely*, in order to initiate appropriate methods (e.g., further medical investigations).

6 Related Approaches and Remarks

It is worth noting to state that this paper is interested in the representation of and reasoning with *uncertain assertional* knowledge, and neither in dealing with *vagueness* found in natural language (*very small*), nor in handling *defaults* and *exceptions* in *terminological* knowledge (*penguins can't fly*).

To the best of our knowledge, the modal logic presented in this paper uses for the first time modal operators for expressing the degree of (un)certainty of propositions. These modal operators are interpreted in the model theory through confidence intervals, by using a measure function μ. From a model point of view, our modal operators are related to *counting modalities* $\Diamond^{\geq k}$ [1,4]—however, we do *not* require a *fixed* number $k \in \mathbb{N}$ of reachable successor states (*absolute* frequency), but instead *divide* the number of worlds v reached through label $\delta \in \triangle$ by the number of all reachable worlds, given current state w, yielding $0 \leq p \leq 1$. This fraction then is further constrained by requiring $p \in \mu(\delta)$ (*relative* frequency), as defined in case 5. of the satisfaction relation in Sects. 4.2 and 4.3.

As [20] precisely put it: "*... what axioms and rules must be added to the propositional calculus to create a usable system of modal logic is a matter of philosophical opinion, often driven by the theorems one wishes to prove ...*". Clearly, the logic Λ is *no* exception and its design is driven by commonsense knowledge and plausible inferences, we try to capture.

Our modal logic can be regarded as an instance of the *normal* modal logic $\mathbf{K} := (N) + (K)$ when identifying the basic modal operator $\Box$ with the modal $\top$ (and *only* with $\top$) and by enforcing the *well-behaved* frame condition from Sect. 4.3. Given $\Box \equiv \top$, Λ then includes the *necessitation rule* (N) $p \rightarrow \top p$ and the *distribution axiom* (K) $\top(p \rightarrow q) \rightarrow (\top p \rightarrow \top q)$ where p, q being special theorems in Λ, viz., positive and negative propositional letters.

(N) can be seen as a special case of (L), the *Lift* modal entailment (left-to-right direction) from Sect. 5.1. (K) can be proven in Λ by choosing $\top \in \underline{1}$ in simplification rule *8b* (Sect. 4.1) and by instantiating (G), the *Generalize* modal entailment (Sect. 5.1), together with the application of the tautology $(p \rightarrow q) \Leftrightarrow (\neg p \vee q)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\top(p \rightarrow q) \rightarrow (\top p \rightarrow \top q)}{\top(\neg p \vee q) \rightarrow (\neg \top p \vee \top q)}}{(\top \neg p \vee \top q) \rightarrow (\neg \top p \vee \top q)}}{\top \neg p \rightarrow \neg \top p}}{\bot p \rightarrow \top^{\mathsf{C}} p}$$

The final simplification at which we arrive is valid, since $\bot \preceq \top^{\mathsf{C}}$:

$$\mu(\bot) = [0,0] \subseteq [0,1) = \mu(\top^{\mathsf{C}})$$

Again, through (L) (right-to-left direction), Λ also incorporates the *reflexivity axiom* (T) $\top p \rightarrow p$ making Λ (at least) an instance of the system $\mathbf{T}$. However, this investigation is in a certain sense *useless* as it does *not* address the other

modals: almost always, neither (N), (K), nor (T) hold for modals from $\triangle$. Thus, we can *not* view Λ as an instance of a *poly-modal* logic.

Several approaches to representing and reasoning with uncertainty have been investigated in *Artificial Intelligence* (see [5,14] for two comprehensive overviews). Very less so has been researched in the *Description Logic* community, and little or nothing of this research has find its way into implemented systems. [7,8] consider *uncertainty* in $\mathcal{ALC}$ concept hierarchies, plus concept typing of individuals (unary relations) in different ways (probability values vs. intervals; conditional probabilities in TBox vs. ABox). They do not address uncertain binary (or even n-ary) relations. [19] investigates *vagueness* in $\mathcal{ALC}$ concept descriptions to address statements, such as *the patient's temperature is high, but also for determining membership degree (38.5 °C)*. This is achieved through *membership manipulators* which are functions, returning a truth value between 0 and 1, thus deviating from a two-valued logic. [17] defines a *fuzzy* extension of $\mathcal{ALC}$, based on Zadeh's *fuzzy logic*. As in [19], the truth value of an assertion is replaced by a membership value from $[0, 1]$. $\mathcal{ALC}$ assertions α in [17] are made fuzzy by writing, e.g., $\langle \alpha \geq n \rangle$, thus taking a single truth value from $[0, 1]$. An even more expressive description logic, Fuzzy OWL, based on OWL DL, is investigated in [16].

Our work might be viewed as a modalized version of a restricted fragment of *Subjective Logic* [9,10], a probabilistic logic that can be seen as an extension of Dempster-Shafer belief theory. Subjective Logic addresses subjective believes by requiring numerical values for *believe* b, *disbelieve* d, and *uncertainty* u, called (subjective) opinions. For each proposition, it is required that $b + d + u = 1$. The translation from modals δ to $\langle b, d, u \rangle$ is determined by the length of the confidence interval $\mu(\delta) = [l, h]$ and its starting/ending numbers, viz., $u := h - l$, $b := l$, and $d := 1 - h$.

Acknowledgements. The research described in this paper has been co-funded by the Horizon 2020 Framework Programme of the European Union within the project PAL (Personal Assistant for healthy Lifestyle) under Grant agreement no. 643783. The authors have profited from discussions with our colleagues Miroslav Janíček and Bernd Kiefer and would like to thank the three reviewers for their suggestions.

References

1. Areces, C., Hoffmann, G., Denis, A.: Modal logics with counting. In: Dawar, A., de Queiroz, R. (eds.) WoLLIC 2010. LNCS, vol. 6188, pp. 98–109. Springer, Heidelberg (2010)
2. Baader, F.: Description logic terminology. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.) The Description Logic Handbook, pp. 495–505. Cambridge University Press, Cambridge (2003)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge (2001)
4. Fine, K.: In so many possible worlds. Notre Dame J. Formal Logic **13**(4), 516–520 (1972)

5. Halpern, J.Y.: Reasoning About Uncertainty. MIT Press, Cambridge (2003)
6. Hayes, P.: RDF semantics. Technical report, W3C (2004)
7. Heinsohn, J.: ALCP - Ein hybrider Ansatz zur Modellierung von Unsicherheit in terminologischen Logiken. Ph.D. thesis, Universität des Saarlandes, June 1993 (in German)
8. Jaeger, M.: Probabilistic reasoning in terminological logics. In: Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 305–316 (1994)
9. Jøsang, A.: Artificial reasoning with subjective logic. In: Proceedings of the 2nd Australian Workshop on Commonsense Reasoning (1997)
10. Jøsang, A.: A logic for uncertain probabilities. Int. J. Uncertainty, Fuzzyness Knowl. Based Syst. **9**(3), 279–311 (2001)
11. Krengel, U.: Einführung in die Wahrscheinlichkeitstheorie und Statistik, 7th edn. Vieweg (2003) (in German)
12. Krieger, H.U.: A temporal extension of the Hayes/ter Horst entailment rules and an alternative to W3C's n-ary relations. In: Proceedings of the 7th International Conference on Formal Ontology in Information Systems (FOIS), pp. 323–336 (2012)
13. Krieger, H.U.: An efficient implementation of equivalence relations in OWL via rule and query rewriting. In: Proceedings of the 7th IEEE International Conference on Semantic Computing (ICSC), pp. 260–263 (2013)
14. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco (1988)
15. Schulz, S., Martínez-Costa, C., Karlsson, D., Cornet, R., Brochhausen, M., Rector, A.: An ontological analysis of reference in health record statements. In: Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS 2014) (2014)
16. Stoilos, G., Stamou, G.B., Tzouvaras, V., Pan, J.Z., Horrocks, I.: Fuzzy OWL: uncertainty and the semantic web. In: Proceedings of the OWLED 2005 Workshop on OWL: Experiences and Directions (2005)
17. Straccia, U.: Reasoning within fuzzy description logics. J. Artif. Intell. Res. **14**, 147–176 (2001)
18. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. J. Web Seman. **3**, 79–115 (2005)
19. Tresp, C.B., Molitor, R.: A description logic for vague knowledge. In: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI), pp. 361–365 (1998)
20. Wikipedia: Modal logic – Wikipedia, The Free Encyclopedia (2015). https://en.wikipedia.org/wiki/Modal_logic, Accessed on 19 June 2015

Models for the Displacement Calculus

Oriol Valentín(✉)

Universitat Politècnica de Catalunya, Barcelona, Spain
ovalentin@cs.upc.edu

Abstract. The displacement calculus **D** is a conservative extension of the Lambek calculus **L1** (with empty antecedents allowed in sequents). **L1** can be said to be the logic of concatenation, while **D** can be said to be the logic of concatenation and intercalation. In many senses, it can be claimed that **D** mimics **L1** in that the proof theory, generative capacity and complexity of the former calculus are natural extensions of the latter calculus. In this paper, we strengthen this claim. We present the appropriate classes of models for **D** and prove some completeness results; strikingly, we see that these results and proofs are natural extensions of the corresponding ones for **L1**.

1 Introduction

The displacement calculus **D** is a quite well-studied extension of the Lambek calculus **L1** (with empty antecedents allowed in sequents). In many papers (see [9,11,12]), **D** has proved to provide elegant accounts of a variety of linguistic phenomena of English, and of Dutch, namely a processing interpretation of the so-called Dutch cross-serial dependencies.

The hypersequent format **hD**[1] of displacement calculus is a pure sequent calculus free of structural rules which subsumes the sequent calculus for **L1**. The Cut elimination algorithm for **hD** provided in [12] mimics the one of Lambek's [5] syntactic calculus (with some minor differences concerning the possibility of empty antecedents). Like **L1**, **D** enjoys some nice properties such as the subformula property, decidablity, the finite reading property and the focalisation property [7].

Like **L1**, **D** is known to be NP-complete [6]. Concerning (weak) generative capacity, **D** recognises the class of well-nested multiple context-free languages [13]. In this respect, the result on generative capacity generalises the result that states that **L1** recognises the class of context-free languages. One point of divergence in terms of generative capacity is that **D** recognises the class of the permutation closures of context-free languages [10]. Finally, it is important to note that a Pentus-like upper bound theorem for **D** is not known.

In this paper we present natural classes of models for **D**. Several strong completeness results are proved, in particular strong completeness w.r.t. the class

Research partially supported by SGR2014-890 (MACDA) of the Generalitat de Catalunya, and MINECO project APCOM (TIN2014-57226-P).

[1] Not to be confused with the hypersequents of Avron [1].

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 147–163, 2016.
DOI: 10.1007/978-3-662-53042-9_9

of residuated displacement algebras (a natural extension of residuated monoids). Powerset frames for **L1** are of interest from the linguistic point of view because of their relation to language models. Powerset residuated displacement algebras over displacement algebras are given, which generalise the powerset residuated monoids over monoids, as well as over free monoids. Strong completeness results for the so-called implicative fragment of **D**, which is very relevant linguistically, is proved in the spirit of Buszkowski [2], but the construction is more subtle.

The structure of the paper is as follows. In Sect. 2 we present the basic proof-theoretic tools (useful for the construction of canonical models) which we shall employ for the study of **D** from a semantic point of view. In Sect. 3 we provide the proof of two strong completeness of what we call the *implicative* fragment w.r.t. powerset DAs over standard DAs (with a countably infinite set of generators) and L-models respectively.

2 The Categorical Calculus cD and the Hypersequent Calculus hD

D is model-theoretically motivated, and the key to its conception is the use of many-sorted universal algebra [3], namely ω-sorted universal algebra. Here, we assume a version of many-sorted algebra such that the sort domains of an ω-sorted algebra $\mathcal{A}$ are non-empty. With this condition we avoid some pathologies which arise in a naïve version of many-sorted universal algebra (cf. [3,4]). Some definitions are needed. Let $\mathcal{M} = (|\mathcal{M}|, +, 0, 1)$ be a free monoid where 1 is a distinguished element of the set of generators X of $\mathcal{M}$. We call such an algebra a *separated monoid*. Given an element $a \in |\mathcal{M}|$, we can associate to it a number, called its *sort* as follows:

(1) $\begin{array}{ll} s(1) & = 1 \\ s(a) & = 0 \text{ if } a \in X \text{ and } a \neq 1 \\ s(w_1 + w_2) & = s(w_1) + s(w_2) \end{array}$

This induction is well-defined since $\mathcal{M}$ is free and 1 is a (distinguished) generator; the sort function $s(\cdot)$ in a separated monoid simply counts the number of separators an element contains.

Definition 1 (Sort Domains). Where $\mathcal{M} = (|\mathcal{M}|, +, 0, 1)$ is a separated monoid, the *sort domains* $|\mathcal{M}|_i$ of sort i are defined as follows:

$$|\mathcal{M}|_i = \{a \in |\mathcal{M}| : s(a) = i\}, i \geq 0$$

It is readily seen that for every $i, j \geq 0, |\mathcal{M}|_i \cap |\mathcal{M}|_j = \emptyset$ iff $i \neq j$.

Definition 2 (Standard Displacement Algebra). The *standard displacement algebra* (or standard DA) defined by a separated monoid $(|\mathcal{M}|, +, 0, 1)$ is the ω-sorted algebra with the ω-sorted signature $\Sigma_D = (+, \{\times_i\}_{i>0}, 0, 1)$ with sort functionality $((i, j \to i + j)_{i,j\geq 0}, (i, j \to i + j - 1)_{i>0, j\geq 0}, 0, 1)$:

$$(\{|\mathcal{M}|_i\}_{i\geq 0}, +, \{\times_i\}_{i>0}, 0, 1)$$

where:

operation	which is
$+ : \lvert\mathcal{M}\rvert_i \times \lvert\mathcal{M}\rvert_j \to \lvert\mathcal{M}\rvert_{i+j}$	as in the separated monoid
$\times_k : \lvert\mathcal{M}\rvert_i \times \lvert\mathcal{M}\rvert_j \to \lvert\mathcal{M}\rvert_{i+j-1}$	$\times_k(s,t)$: the result of replacing the k-th separator in s by t

The sorted types of **D**, which we will interpret residuating w.r.t. the sorted operations in Definition 2, are defined by mutual recursion in Fig. 1. We let $\mathbf{Tp} = \bigcup_{i\geq 0} \mathbf{Tp}_i$. A subset B of $\lvert\mathcal{M}\rvert$ is called a *same-sort* subset iff there exists an $i \in \omega$ such that for every $a \in B$, $s(a) = i$. **D** types are to be interpreted as same-sort subsets of $\lvert\mathcal{M}\rvert$. I.e. every inhabitant of $[\![A]\!]$ has the same sort. The intuitive semantic interpretation of the connectives is shown in Fig. 2; this interpretation is called the *standard interpretation*. Observe that for any type $A \in \mathbf{Tp}$, the interpretation of A, i.e. $[\![A]\!]$, is contained in $M_{s(A)}$, where the sort map $s(\cdot)$ for the set **Tp**, is such that

$$
\begin{array}{lll}
(2) & s(p) = i & \text{for p} \in \mathbf{Pr}_i \\
& s(I) = 0 & \\
& s(J) = 1 & \\
& s(A \bullet B) = s(A) + s(B) & \\
& s(A \backslash B) = s(B) - s(A) & \\
& s(B / A) = s(B) - s(A) & \\
& s(A \odot_k B) = s(A) + s(B) - 1 & \\
& s(A \downarrow_k B) = s(B) - s(A) + 1 & \\
& s(B \uparrow_k A) = s(B) - s(A) + 1 &
\end{array}
$$

$$
\begin{array}{rll}
\mathbf{Tp}_i ::= \mathbf{Pr}_i & & \text{where } \mathbf{Pr}_i \text{ is the set of atomic types of sort } i \\
\\
\mathbf{Tp}_0 ::= I & & \text{Continuous unit} \\
\mathbf{Tp}_1 ::= J & & \text{Discontinuous unit} \\
\\
\mathbf{Tp}_{i+j} ::= \mathbf{Tp}_i \bullet \mathbf{Tp}_j & & \text{continuous product} \\
\mathbf{Tp}_j ::= \mathbf{Tp}_i \backslash \mathbf{Tp}_{i+j} & & \text{under} \\
\mathbf{Tp}_i ::= \mathbf{Tp}_{i+j} / \mathbf{Tp}_j & & \text{over} \\
\\
\mathbf{Tp}_{i+j} ::= \mathbf{Tp}_{i+1} \odot_k \mathbf{Tp}_j & & \text{discontinuous product} \\
\mathbf{Tp}_j ::= \mathbf{Tp}_{i+1} \downarrow_k \mathbf{Tp}_{i+j} & & \text{extract} \\
\mathbf{Tp}_{i+1} ::= \mathbf{Tp}_{i+j} \uparrow_k \mathbf{Tp}_j & & \text{infix}
\end{array}
$$

Fig. 1. The sorted types of **D**

$$
\begin{array}{rll}
[\![I]\!] = \{0\} & & \text{continuous unit} \\
[\![J]\!] = \{1\} & & \text{discontinuous unit} \\
\\
[\![A \bullet B]\!] = \{s_1 + s_2 |\ s_1 \in [\![A]\!] \ \&\ s_2 \in [\![B]\!]\} & & \text{product} \\
[\![A \backslash C]\!] = \{s_2 |\ \forall s_1 \in [\![A]\!], s_1 + s_2 \in [\![C]\!]\} & & \text{under} \\
[\![C / B]\!] = \{s_1 |\ \forall s_2 \in [\![B]\!], s_1 + s_2 \in [\![C]\!]\} & & \text{over} \\
\\
[\![A \odot_k B]\!] = \{\times_k(s_1, s_2) |\ s_1 \in [\![A]\!] \ \&\ s_2 \in [\![B]\!]\} & k > 0 & \text{discontinuous product} \\
[\![A \downarrow_k C]\!] = \{s_2 |\ \forall s_1 \in [\![A]\!], \times_k(s_1, s_2) \in [\![C]\!]\} & k > 0 & \text{infix} \\
[\![C \uparrow_k B]\!] = \{s_1 |\ \forall s_2 \in [\![B]\!], \times_k(s_1, s_2) \in [\![C]\!]\} & k > 0 & \text{extract}
\end{array}
$$

Fig. 2. Standard semantic interpretation of **D** types

2.1 D and its Categorical Presentation cD

In [14] **D** is presented as a categorical calculus:

(3) $A \to A$ Axiom
$A \bullet B \to C$ iff $A \to C/B$ iff $B \to A \backslash C$ $\quad Res_{cont}$
$A \odot_i B \to C$ iff $A \to C \uparrow_i B$ iff $B \to A \downarrow_i C$ $\quad Res_{disc}$

$A \bullet I \leftrightarrow A \leftrightarrow I \bullet A \quad A \odot_i J \leftrightarrow A \leftrightarrow J \odot_1 A$
$(A \bullet B) \bullet C \leftrightarrow A \bullet (B \bullet C)$ Continuous associativity
$A \odot_i (B \odot_j C) \leftrightarrow (A \odot_i B) \odot_{i+j-1} C$ Discontinuous associativity
$(A \odot_i B) \odot_j C \leftrightarrow A \odot_i (B \odot_{j-i+1} C)$, if $i \leq j \leq 1 + s(B) - 1$

$(A \odot_i B) \odot_j C \leftrightarrow (A \odot_{j-s(B)+1} C) \odot_i B$, if $j > i + s(B) - 1$ Mixed permutation
$(A \odot_i C) \odot_j B \leftrightarrow (A \odot_j B) \odot_{i+s(B)-1} C$, if $j < i$

$(A \bullet B) \odot_i C \leftrightarrow (A \odot_i C) \bullet B$, if $1 \leq i \leq S(A)$ Mixed associativity
$(A \bullet B) \odot_i C \leftrightarrow A \bullet (B \odot_{i-s(C)} C)$, if $s(A) + 1 \leq i \leq s(A) + s(B)$
From $A \to B$ and $B \to C$ we have $A \to C$ Transitivity

In Fig. 3 we find the axiomatisation of the class of DAs $\mathcal{DA}$. Just as in the case of **L1**, the natural class of algebras is the class of residuated monoids $\mathcal{RM}$, in the case of **D**, the natural class of algebras is the class of residuated displacement algebras (residuated DAs) $\mathcal{RD}$.

One can restrict the definition of the sorted types. Let **C** be a subset of the connectives considered in the definition of types in Fig. 1. We define **Tp**[**C**] as the least set of sorted types generated by **Pr** and the set of connectives **C**. If the context is clear, we will write **Tp** instead of **Tp**[**C**].

Let us define the formal definition of a model. A model $\mathcal{M} = (\mathcal{A}, v)$ comprises a (residuated) Σ_D-algebra and a ω-sorted mapping $v : \mathbf{Pr} \to \mathbf{Tp}[\mathbf{C}]$ called a valuation. The mapping $\widehat{v}$ is the unique function which extends v and which is such that $\widehat{v}(A * B) = \widehat{v}(A) * \widehat{v}(B)$ (if $*$ is a binary connective of **C**) and $\widehat{v}(*A) = *\widehat{v}(A)$ (if $*$ is a unary connective of **C**). Finally, a 0-ary connective is mapped into the corresponding unit of $|\mathcal{A}|$. Needless to say, the mappings v and $\widehat{v}$ preserve the sorting regime.

Continuous associativity
$x + (y + z) \approx (x + y) + z$

Discontinuous associativity
$x \times_i (y \times_j z) \approx (x \times_i y) \times_{i+j-1} z$
$(x \times_i y) \times_j z \approx x \times_i (y \times_{j-i+1} z)$ if $i \leq j \leq 1 + s(y) - 1$

Mixed permutation
$(x \times_i y) \times_j z \approx (x \times_{j-\,S(y)\,+1} z) \times_i y$ if $j > i + s(y) - 1$
$(x \times_i z) \times_j y \approx (x \times_j y) \times_{i+S(y)-1} z$ if $j < i$

Mixed associativity
$(x + y) \times_i z \approx (x \times_i z) + y$ if $1 \leq i \leq s(x)$
$(x + y) \times_i z \approx x + (y \times_{i-s(x)} z)$ if $x + 1 \leq i \leq s(x) + s(y)$

Continuous unit and discontinuous unit
$0 + x \approx x \approx x + 0$ and $1 \times_1 x \approx x \approx x \times_i 1$

Fig. 3. Axiomatisation of $\mathcal{DA}$

Let us see that **D** (with all the connectives) is strongly complete w.r.t. $\mathcal{RD}$. Soundness is trivial because we are considering the categorical calculus **cD**. For completeness, we can define the well-known Lindenbaum-Tarski construction to see that **cD** is strongly complete w.r.t. $\mathcal{RD}$. The canonical model is $(\mathcal{L}, v)$ where $\mathcal{L}$ is $(\mathbf{Tp}/\theta, \circ, (\circ_i)_{i>0}, \backslash\backslash, //, (\downarrow\downarrow_i)_{i>0}(\uparrow\uparrow_i)_{i>0}, \mathbb{I}, \mathbb{J}; \leq)$ where the interpretation of the new symbols is as expected. Let θ_R be the equivalence relation on **Tp** defined as follows: $A\theta_R B$ iff $R \vdash_{\mathbf{cD}} A \to B$ and $R \vdash_{\mathbf{cD}} B \to A$, where R is a set of non-logical axioms. Using the usual tonicity properties for the connectives of **Tp**, one proves that θ_R is a congruence. Where A is a type, $\overline{A}$ is an element of $\mathbf{Tp}/\theta_R$, i.e. **Tp** modulo θ_R. We define $\overline{A} \leq \overline{B}$ iff $R \vdash_{\mathbf{cD}} A \to B$. We define the valuation v as $v(p) = \overline{p}$ (p is a primitive type). We have that for every type A, $\widehat{v}(A) = \overline{A}$. Finally, one has that $(\mathcal{L}, v) \models A \to B$ iff $R \vdash_{\mathbf{cD}} A \to B$. From this, we infer the following theorem:

Theorem 1. *The calculus* **cD** *is strongly complete w.r.t.* $\mathcal{RD}$.

Since $\mathcal{DA}$ is a variety[2] (see Fig. 3), it is closed by subalgebras, direct products and homomorphic images, which give additional DAs.

We have other interesting examples of DAs, for instance the *powerset DA over* $\mathcal{A} = (|\mathcal{A}|, +, \{\times_i\}_{i>0}, 0, 1)$, which we denote $\mathcal{P}(A)$. We have:

(4) $\mathcal{P}(A) = (|\mathcal{P}(A)|, \cdot, \{\circ_i\}_{i>0}, \mathbb{I}, \mathbb{J})$

[2] The term *equational class* is sometimes used in the literature.

The notation of the carrier set of $\mathcal{P}(A)$ presupposes that its members are same-sort subsets; notice that $\emptyset$ vacuously satisfies the *same-sort* condition. Where A, B and C denote same-sort subsets of $|\mathcal{A}|$, the operations $\mathbb{I}$, $\mathbb{J}$, $\cdot$ and $\circ_i$ are defined as follows:

$$\begin{array}{lll} (5) & \mathbb{I} & = \{0\} \\ & \mathbb{J} & = \{1\} \\ & A \cdot B & = \{a + b : a \in A \text{ and } b \in B\} \\ & A \circ_i B & = \{a \times_i b : a \in A \text{ and } b \in B\} \end{array}$$

It is readily seen that for every $\mathcal{A}, \mathcal{P}(A)$ is in fact a DA. Notice that every sort domain $|\mathcal{P}(A)|_i$ is a collection of same-sort subsets, that the sort domains of $\mathcal{P}(A)$ are non-empty, but no longer satisfy that $|\mathcal{P}(A)|_i \cap |\mathcal{P}(A)|_j = \emptyset$ iff $i \neq j$, since the empty set $\emptyset \in |\mathcal{P}(A)|_i$ for every $i \geq 0$. A residuated powerset displacement algebra over a displacement algebra $\mathcal{P}(A)$ is the following:

$$(6)\quad \mathcal{P}(A) = (|\mathcal{P}(A)|, \cdot, \backslash\backslash, //, \{\circ_i\}_{i>0}, , \{\uparrow\uparrow_i\}_{i>0}, \{\downarrow\downarrow_i\}_{i>0}, \mathbb{I}, \mathbb{J}; \subseteq)$$

where $\backslash\backslash$, $//$, $\uparrow\uparrow_i$ and $\downarrow\downarrow_i$ are defined as follows:

$$\begin{array}{lll} (7) & A\backslash\backslash B & = \{d : \text{ for every } a \in A,\ a + d \in B\} \\ & B//A & = \{d : \text{ for every } a \in A,\ d + a \in B\} \\ & B \uparrow\uparrow_i A & = \{d : \text{ for every } a \in A,\ d \times_i a \in B\} \\ & A \downarrow\downarrow_i B & = \{d : \text{ for every } a \in A,\ a \times_i d \in B\} \end{array}$$

The class of powerset residuated DAs over a DA is denoted $\mathcal{PRDD}$. The class of powerset residuated DAs over a standard DA is denoted $\mathcal{PRSD}$. Finally, the subclass of $\mathcal{PRSD}$ which is formed by powerset residuated algebras over finitely-generated standard DA are known simply as *L-models*.

Every standard DA $\mathcal{A}$ has two remarkable properties, namely the property that sort domains $|\mathcal{A}|_i$ (for $i > 0$) can be defined in terms of $|\mathcal{A}|_0$, and the property that every element a of a sort domain $|\mathcal{A}|_i$ is decomposed uniquely around the separator 1:

$$(8)\quad \text{(S1) For } i > 0,\ |\mathcal{A}|_i = \underbrace{|\mathcal{A}|_0 \circ \{1\} \cdots \{1\} \circ |\mathcal{A}|_0}_{(i-1)\ 1's}$$

(S2) For $i > 0$, if $a_0 + 1 + \cdots + 1 + a_i = b_0 + 1 + \cdots + 1 + b_i$ then $a_k = b_k$ for $0 \leq k \leq i$

Standard DAs, as their name suggests, are particular cases of (general) DAs:

Lemma 1. *The class of standard DAs is a subclass of the class of DAs.*[3]

Proof. We define a useful notation which will help us to prove the lemma. Where $\mathcal{A} = (|\mathcal{A}|, +, (\times_i)_{i>0}, 0, 1)$ is a standard DA, let a be an arbitrary element of sort $s(a)$. We associate to every $a \in |\mathcal{A}|$ a sequence of elements $a_0, \cdots, a_{s(A)}$. We have the following vectorial notation:

$$(9)\quad \overrightarrow{a}_i^j = \begin{cases} a_i, \text{ if } i = j \\ \overrightarrow{a}_i^{j-1} + 1 + a_j, \text{ if } j - i > 0 \end{cases}$$

[3] Later we see that the inclusion is proper.

Since $\mathcal{A}$ is a standard DA, the a_i associated to a given $\overrightarrow{a}$ are unique (by freeness of the underlying monoid). We have that $a = \overrightarrow{a}_0^{s(A)}$, and we write $\overrightarrow{a}$ in place of $\overrightarrow{a}_0^{s(A)}$. Consider arbitrary elements $\overrightarrow{a}$, $\overrightarrow{b}$ and $\overrightarrow{c}$ of $|\mathcal{A}|$:

- Continuous associativity is obvious.
- Discontinuous associativity. Let i, j be such that $i \leq j \leq i + s(\overrightarrow{a}) - 1$:

$$\overrightarrow{b} \times_j \overrightarrow{c} = \overrightarrow{b}_0^{i-1} + \overrightarrow{c} + \overrightarrow{b}_i^{s(b)}, \text{ therefore:}$$
$$\overrightarrow{a} \times_i (\overrightarrow{b} \times_j \overrightarrow{c}) = \boxed{\overrightarrow{a}_0^{i-1} + \overrightarrow{b}_0^{j-1} + \overrightarrow{c} + \overrightarrow{b}_j^{s(b)} + \overrightarrow{a}_i^{s(a)}} \qquad (*)$$

On the other hand, we have that:

$$\overrightarrow{a} \times_i \overrightarrow{b} = \overrightarrow{a}_0^{i-1} + \overrightarrow{b} + \overrightarrow{a}_i^{s(\overrightarrow{a})} = \overrightarrow{a}_0^{i-1} + \overrightarrow{b}_0^{j-1} + \underbrace{1}_{(i+j-1)\text{-th } separator} + \overrightarrow{b}_j^{s(\overrightarrow{b})} + \overrightarrow{a}_i^{s(\overrightarrow{a})}$$

It follows that:

$$(\overrightarrow{a} \times_i \overrightarrow{b}) \times_{i+j-1} \overrightarrow{c} = \boxed{\overrightarrow{a}_0^{i-1} + \overrightarrow{b}_0^{j-1} + \overrightarrow{c} + \overrightarrow{b}_j^{s(\overrightarrow{b})} + \overrightarrow{a}_i^{s(\overrightarrow{a})}} \qquad (**)$$

By comparing the right hand side of (*) and (**), we have therefore:

$$\overrightarrow{a} \times_i (\overrightarrow{b} \times_j \overrightarrow{c}) = (\overrightarrow{a} \times_i \overrightarrow{b}) \times_{i+j-1} \overrightarrow{c}$$

- Mixed Permutation. Consider $(\overrightarrow{a} \times_i \overrightarrow{b}) \times_j \overrightarrow{c}$ and suppose that $i + s(\overrightarrow{b}) - 1 < j$:

$$\overrightarrow{a} \times_i \overrightarrow{b} = \underbrace{\overrightarrow{a}_0^{i-1} + \overrightarrow{b} + \overrightarrow{a}_i^{j-s(\overrightarrow{b})}}_{j-s(\overrightarrow{b})+s(\overrightarrow{b})-1=j-1 \text{ separators}} + 1 + \overrightarrow{a}_{j-s(\overrightarrow{b})+1}^{s(\overrightarrow{a})}$$

It follows that:

$$(\overrightarrow{a} \times_i \overrightarrow{b}) \times_j \overrightarrow{c} = \boxed{\overrightarrow{a}_0^{i-1} + \overrightarrow{b} + \overrightarrow{a}_i^{j-s(\overrightarrow{b})} + \overrightarrow{c} + \overrightarrow{a}_{j-s(\overrightarrow{b})+1}^{s(\overrightarrow{a})}} \qquad (***)$$

Since $i + s(\overrightarrow{b}) - 1 < j$, then $i < j - s(\overrightarrow{b}) + 1$. Then we have that:

$$\overrightarrow{a} \times_{j-s(\overrightarrow{b})+1} \overrightarrow{c} = \overrightarrow{a}_0^{i-1} + 1 + \overrightarrow{a}_i^{j-s(\overrightarrow{b})} + \overrightarrow{c} + \overrightarrow{a}_{j-s(\overrightarrow{b})+1}^{s(\overrightarrow{a})}$$

It follows that:

$$(\overrightarrow{a} \times_{j-s(\overrightarrow{b})+1} \overrightarrow{c}) \times_i \overrightarrow{b} = \boxed{\overrightarrow{a}_0^{i-1} + \overrightarrow{b} + \overrightarrow{a}_i^{j-s(\overrightarrow{b})} + \overrightarrow{c} + \overrightarrow{a}_{j-s(\overrightarrow{b})+1}^{s(\overrightarrow{a})}} \qquad (****)$$

By comparing the right hand side of (***) and (****), we have therefore:

$$(\overrightarrow{a} \times_i \overrightarrow{b}) \times_j \overrightarrow{c} = (\overrightarrow{a} \times_{j-s(\overrightarrow{b})+1} \overrightarrow{c}) \times_i \overrightarrow{b}$$

- Mixed associativity. There are two cases: $i \leq s(\overrightarrow{a})$ or $i > s(\overrightarrow{a})$. Considering the first one, this is true for:

$$(\overrightarrow{a}+\overrightarrow{b})\times\overrightarrow{c} = (\overrightarrow{a}_0^{i-1}+1+\overrightarrow{a}_i^{s(\overrightarrow{a})})\times_i\overrightarrow{c} = \overrightarrow{a}_0^{i-1}+\overrightarrow{c}+\overrightarrow{a}_i^{s(\overrightarrow{a})}+\overrightarrow{b} = (\overrightarrow{a}\times_i\overrightarrow{c})+\overrightarrow{b}$$

  The other case corresponding to $i > s(\overrightarrow{a})$ is completely similar.
- The case corresponding to the units is completely trivial. □

2.2 The Hypersequent Calculus hD

We will now consider the *string-based* hypersequent syntax from [8]. The reason for using the prefix *hyper* in the term *sequent* is that the data-structure used in hypersequent antecedents is quite nonstandard. A fundamental tool to build the data-structure of a sequent calculus for **D** is the notion of *type-segment*. For any type of sort 0 $seg(A) = \{A\}$. If $s(A) > 0$ then $seg(A) = \{\sqrt[0]{A}, \cdots, \sqrt[s(A)]{A}\}$. We call $seg(A)$ the set of type-segments of A. If **C** is a set of connectives, we can now define the set of type-segments corresponding to the set $\mathbf{Tp}[\mathbf{C}]$ of types generated by the connectives **C** as $\mathbf{seg}[\mathbf{C}] = \bigcup_{A\in\mathbf{Tp}[\mathbf{C}]}\mathrm{seg(A)}$. Type segments of sort 0 are types. But, type segments of sort greater than 0 are no longer types. Strings of type segments can form meaningful logical material like the set of configurations, which we now define. Where **C** is a set of connectives the *configurations* $\mathcal{O}[\mathbf{C}]$ are defined in BNF unambiguously by mutual recursion as follows, where Λ is the empty string and 1 is the metalinguistic separator:

$$\begin{array}{ll}(10) & \mathcal{O}[\mathbf{C}] ::= \Lambda\\ & \mathcal{O}[\mathbf{C}] ::= 1, \mathcal{O}[\mathbf{C}]\\ & \mathcal{O}[\mathbf{C}] ::= A, \mathcal{O}[\mathbf{C}] \quad \text{for } s(A) = 0\\ & \mathcal{O}[\mathbf{C}] ::= \sqrt[0]{A}, \mathcal{O}[\mathbf{C}], \sqrt[1]{A}, \cdots, \sqrt[s(A)-1]{A}, \mathcal{O}[\mathbf{C}], \sqrt[s(A)]{A}_{s(A)}, \mathcal{O}[\mathbf{C}] \quad \text{for } s(A) > 0\end{array}$$

The intuitive semantic interpretation of the last clause from (10) consists of elements $\alpha_0+\beta_1+\alpha_1+\cdots+\alpha_{n-1}+\beta_n+\alpha_n$ where $\alpha_0+1+\alpha_1+\cdots+\alpha_{n-1}+1+\alpha_n \in [\![A]\!]$ and $\beta_1, \cdots, \beta_n$ are the interpretations of the intercalated configurations.

If the context is clear we will write $\mathcal{O}$ for $\mathcal{O}[\mathbf{C}]$, and likewise **Tp**, and **seg**.

The syntax in which $\mathcal{O}$ has been defined is called *string-based hypersequent syntax*. An equivalent syntax for $\mathcal{O}$ is called *tree-based hypersequent syntax*, which was defined in [9,12]. For proof-search and human readability, the tree-based notation is more convenient than the string-based notation, but for semantic purposes, the string-based notation turns out to be very useful since the canonical model construction considered in Sect. 3 relies on the set of type-segments.

In string-based notation the *figure* $\overrightarrow{A}$ of a type A is defined as follows:

$$(11)\quad \overrightarrow{A} = \begin{cases} A & \text{if } s(A) = 0\\ \sqrt[0]{A}, 1, \sqrt[1]{A}, \cdots, \sqrt[s(A)-1]{A}, 1, \sqrt[s(A)]{A} & \text{if } s(A) > 0\end{cases}$$

The sort of a configuration is the number of metalinguistic separators it contains. We have $\mathcal{O} = \bigcup_{i\geq 0}\mathcal{O}_i$, where $\mathcal{O}_i$ is the set of configurations of sort i. We define a more general notion of configuration, namely preconfiguration. If V denotes $\mathbf{seg}[\mathbf{C}] \cup \{1\}$, a preconfiguration Δ is simply a word of V^*. Obviously,

we have that $\mathcal{O} \subsetneq V^*$. A preconfiguration Δ is proper iff $\Delta \notin \mathcal{O}$. As in the case of configurations, preconfigurations have a sort.

Where Γ and Φ are configurations and the sort of Γ is at least 1, $\Gamma|_k\Phi$ $(k > 0)$ signifies the configuration which is the result of replacing the k-th separator in Γ by Φ. The notation $\Delta\langle\Gamma\rangle$, which we call a configuration with a distinguished configuration Γ abbreviates the following configuration: $\Delta_0, \Gamma_0, \Delta_1, \cdots, \Delta_{s(\Gamma)}, \Gamma_{s(\Gamma)}, \Delta_{s(\Gamma)+1}$, where $\Delta_i \in \mathcal{O}$ but Δ_0 and $\Delta_{s(\Gamma)+1}$ are possibly proper preconfigurations. When a type-occurrence A in a configuration is written without vectorial notation, that means that the sort of A is 0. However, when one writes the metanotation for configurations $\Delta\langle\overrightarrow{A}\rangle$, this does not mean that the sort of A is necessarily greater than 0.

$$\overrightarrow{A} \Rightarrow A \text{ if } A \text{ is primitive}$$

$$\frac{\Delta\langle\Lambda\rangle \Rightarrow A}{\Delta\langle I\rangle \Rightarrow A}\, IL \qquad \frac{}{\Lambda \Rightarrow I}\, IR$$

$$\frac{\Delta\langle 1\rangle \Rightarrow A}{\Delta\langle\overrightarrow{J}\rangle \Rightarrow A}\, JL \qquad \frac{}{1 \Rightarrow J}\, JR$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\overrightarrow{B/A}, \Gamma\rangle \Rightarrow C}\, /L \qquad \frac{\Delta, \overrightarrow{A} \Rightarrow B}{\Delta \Rightarrow B/A}\, /R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\Gamma, \overrightarrow{A\backslash B}\rangle \Rightarrow C}\, \backslash L \qquad \frac{\overrightarrow{A}, \Delta \Rightarrow B}{\Delta \Rightarrow A\backslash B}\, \backslash R$$

$$\frac{\Delta\langle\overrightarrow{A}, \overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\overrightarrow{A \bullet B}\rangle \Rightarrow C}\, \bullet L \qquad \frac{\Delta \Rightarrow A \qquad \Gamma \Rightarrow B}{\Delta, \Gamma \Rightarrow A \bullet B}\, \bullet R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\overrightarrow{B\uparrow_i A}|_i\Gamma\rangle \Rightarrow C}\, \uparrow_i L \qquad \frac{\Delta|_i\overrightarrow{A} \Rightarrow B}{\Delta \Rightarrow B\uparrow_i A}\, \uparrow_i R$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\Gamma|_i\overrightarrow{A\downarrow_i B}\rangle \Rightarrow C}\, \downarrow_i L \qquad \frac{\overrightarrow{A}|_i\Delta \Rightarrow B}{\Delta \Rightarrow A\downarrow_i B}\, \downarrow_i R$$

$$\frac{\Delta\langle\overrightarrow{A}|_i\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\overrightarrow{A \odot_i B}\rangle \Rightarrow C}\, \odot_i L \qquad \frac{\Delta \Rightarrow A \qquad \Gamma \Rightarrow B}{\Delta|_i\Gamma \Rightarrow A \odot_i B}\, \odot_i R$$

Fig. 4. Hypersequent calculus for D

A *hypersequent* $\Gamma \Rightarrow A$ comprises an antecedent configuration in string-based notation of sort i and a succedent type A of sort i. The hypersequent calculus for

D is as shown in Fig. 4. The following lemma is useful for the strong completeness results of Sect. 3:

Lemma 2. *Recall that $\mathcal{O}$ is a subset of $V^* = (\mathbf{seg}[C] \cup \{1\})^*$. We have that:*

(i) $\mathcal{O}$ is closed by concatenation and intercalation.
(ii) If $\Delta \in V^*$, $\Gamma \in \mathcal{O}$, and $\Delta, \Gamma \in \mathcal{O}$, then $\Delta \in \mathcal{O}$. Similarly, if we have $\Gamma, \Delta \in \mathcal{O}$ instead of $\Delta, \Gamma \in \mathcal{O}$. Finally, If $\Delta \in V^*$, $\Gamma \in \mathcal{O}$, and $\Delta|_i \Gamma \in \mathcal{O}$, then $\Delta \in \mathcal{O}$.

Proof. Propositions (i) and (ii) are both proved via the BNF derivations of (10). The details of the proof are rather tedious but not difficult.

What is the connection between the calculi **cD** and **hD**? In [14] a (faithful) embedding translation is proved. Let Δ denote a configuration. We define its *type-equivalent* $\Delta^\bullet$, which is a type which has the same algebraic meaning as Δ. Via the BNF formulation of $\mathcal{O}[\mathbf{C}]$ in (10) one defines recursively $\Delta^\bullet$ as follows:

$$
\begin{array}{l}
\Lambda^\bullet = I \\
(1, \Gamma)^\bullet = \mathbf{J} \bullet \Gamma^\bullet \\
(A, \Gamma)^\bullet = A \bullet \Gamma^\bullet, \text{ if } s(A) = 0 \\
(\sqrt[0]{A}, \Delta_1, \cdots, \sqrt[s(A)-1]{A}, \Delta_{s(A)}, \sqrt[s(A)]{A}_{s(A)}, \Delta_{s(A+1)})^\bullet = \\
((\cdots (A \odot_1 \Delta_1^\bullet) \cdots) \odot_{1+s(\Delta_1)+\cdots+s(\Delta_{s(A)})} \Delta_{s(A)}^\bullet) \bullet \Delta_{s(A)+1}^\bullet, \text{ if } s(A) > 0
\end{array}
$$

The semantic interpretation of a configuration Δ (for a given valuation v) is $\widehat{v}(\Delta) = \widehat{v}(\Delta^\bullet)$. The embedding translation is as follows. For any $\Delta \in \mathcal{O}, \mathbf{cD} \vdash \Delta^\bullet \to A$ iff $\mathbf{hD} \vdash \Delta \Rightarrow A$.

2.3 Some Special DAs

The standard DA $\mathcal{S}$, induced by the separated monoid with generator set $V = \mathbf{seg} \cup \{1\}$, plays an important role. The interpretation of the signature Σ_D in $|\mathcal{S}|$ is:

(12) $\mathcal{S} = (V^*, +, \{|_i\}_{i>0}, \Lambda, 1)$

Here, $+$ denotes concatenation, and $\{|_i\}_{i>0}$ i-th intercalation. We have seen in Sect. 2 that $\mathcal{O}$ is closed by concatenation $+$ and intercalation $|_i$, $i > 0$, i.e. $\mathcal{C} = (\mathcal{O}, +, (|_i)_{i>0}, \Lambda, 1)$ is a Σ_D-subalgebra of the standard DA $\mathcal{S}$. Since $\mathcal{DA}$ is a variety,[4] $\mathcal{C}$ is a (general) DA, concretely a nonstandard DA. To see that $\mathcal{C}$ cannot be standard we notice that the sort domains of $\mathcal{C}$ are not separated by $\{1\}$. Recall that $|\mathcal{C}| = \bigcup_{i \geq 0} \mathcal{O}_i$ ($|\mathcal{C}|_i = \mathcal{O}_i$, for every $i \geq 0$). We have that:

(13) For $i > 0$, $|\mathcal{C}|_i \neq \underbrace{\mathcal{O}_0 \circ \cdots \circ \mathcal{O}_0}_{i \text{ times}}$

[4] We recall that varieties are closed by subalgebras, homomorphic images, and direct products.

Because, for example, let us take $\overrightarrow{p\uparrow_1 p} = \sqrt[0]{p\uparrow_1 p}, 1, \sqrt[1]{p\uparrow_1 p}$, where $p \in \mathbf{Pr}_0$. The type $p\uparrow_1 p$ has sort 1, but clearly neither $\sqrt[0]{p\uparrow_1 p}$ nor $\sqrt[1]{p\uparrow_1 p}$ are members of $\mathcal{O}_0$. In fact, we have the proper inclusion:

(14) For $i > 0$, $\underbrace{\mathcal{O}_0 \circ \cdots \circ \mathcal{O}_0}_{i \text{ times}} \subsetneq |\mathcal{C}|_i$

It follows that the class of standard DAs is a *proper* subclass of the class of general DAs.

2.4 Synthetic Connectives and the Implicative Fragment

From a logical point of view, synthetic connectives abbreviate formulas in sequent systems. They form new connectives with left and right sequent rules. Using a linear logic slogan, synthetic connectives help to eliminate some *bureaucracy* in Cut-free proofs and in the (syntactic) Cut-elimination algorithms (see [14]). We consider here a set of synthetic connectives which are of linguistic interest:

- The binary non-deterministic implications $\Uparrow$, and $\Downarrow$.
- The unary connectives $\triangleleft^{-1}$, $\triangleright^{-1}$ and $(^{\smile k})_{k>0}$, which are called respectively left projection, right projection, and split.

Together with the binary deterministic implications $\backslash$, $/$, $(\uparrow_i)_{i>0}$, $(\downarrow_i)_{i>0}$, these constitute what we call *implicative* connectives. These connectives are incorporated in the recursive definitions of **Tp**, **seg**, and $\mathcal{O}$. We denote this implicative fragment as $\mathbf{D}[\rightarrow]$. We write also $\mathbf{Tp}[\rightarrow]$, $\mathbf{seg}[\rightarrow]$, and $\mathcal{O}[\rightarrow]$, although, as usual, if the context is clear we will avoid writing $[\rightarrow]$. The intuitive semantic interpretation of the implicative connectives can be found in Fig. 5. Figures 6 and 7 correspond to their hypersequent rules.

Besides the usual continuous and discontinuous implications, the nondeterministic discontinous implications are used to account for particle shift nondeterminism where the object can be intercalated between the verb and the particle, or after the particle. For a particle verb like *call*+1+*up* we can give the lexical assignment $\triangleleft^{-1}(^{\smile 1}(N\backslash S)\Uparrow N)$. Projections can be used to account for the cross-serial dependencies of Dutch. The split connective can be used for parentheticals like *fortunately* with the type assignment $^{\smile 1}S\downarrow_1 S$.

$$
\begin{array}{rcll}
[\![\triangleleft^{-1}A]\!] & = & [\![A]\!]//\mathbb{J} & \text{left projection} \\
[\![\triangleright^{-1}A]\!] & = & \mathbb{J}\backslash\backslash[\![A]\!] & \text{right projection} \\
[\![^{\smile i}A]\!] & = & [\![A]\!]\uparrow_i\mathbb{I} & i\text{-th split} \\
[\![B\Uparrow A]\!] & = & [\![B\uparrow_1 A]\!] \cap \cdots \cap [\![B\uparrow_{s(B)-s(A)+1} A]\!] & \text{nondeterministic extract} \\
[\![A\Downarrow B]\!] & = & [\![A\downarrow_1 B]\!] \cap \cdots \cap [\![A\downarrow_{s(B)-s(A)+1} B]\!] & \text{nondeterministic infix}
\end{array}
$$

Fig. 5. Semantic interpretation in standard DAs for the set of synthetic connectives

$$\frac{\Gamma\langle\overrightarrow{A}\rangle \Rightarrow B}{\Gamma\langle\overrightarrow{\triangleleft^{-1}A}, 1\rangle \Rightarrow B}\,\triangleleft^{-1}L \qquad \frac{\Gamma, 1 \Rightarrow A}{\Gamma \Rightarrow \triangleleft^{-1}A}\,\triangleleft^{-1}R$$

$$\frac{\Gamma\langle\overrightarrow{A}\rangle \Rightarrow B}{\Gamma\langle 1, \overrightarrow{\triangleright^{-1}A}\rangle \Rightarrow B}\,\triangleright^{-1}L \qquad \frac{1, \Gamma \Rightarrow A}{\Gamma \Rightarrow \triangleright^{-1}A}\,\triangleright^{-1}R$$

$$\frac{\Delta\langle\overrightarrow{B}\rangle \Rightarrow C}{\Delta\langle\overrightarrow{\backsim_i B}|_i\Lambda\rangle \Rightarrow C}\,\backsim_i L \qquad \frac{\Delta|_i\Lambda \Rightarrow B}{\Delta \Rightarrow \backsim_i B}\,\backsim_i R$$

Fig. 6. Hypersequent rules for synthetic unary implicative connectives

$$\frac{\Delta \Rightarrow A \qquad \Gamma\langle\overrightarrow{B}\rangle \Rightarrow C}{\Gamma\langle\overrightarrow{B \Uparrow A}|_i\Gamma\rangle \Rightarrow C}\,\Uparrow L \qquad \frac{\Delta|_1\overrightarrow{A} \Rightarrow B \quad \cdots \quad \Delta|_d\overrightarrow{A} \Rightarrow B}{\Delta \Rightarrow B \Uparrow A}\,\Uparrow R$$

$$\frac{\Delta \Rightarrow A \qquad \Gamma\langle\overrightarrow{B}\rangle \Rightarrow C}{\Gamma\langle\Gamma|_i\overrightarrow{A \Downarrow B}\rangle \Rightarrow C}\,\Downarrow L \qquad \frac{\overrightarrow{A}|_1\Delta \Rightarrow B \quad \cdots \quad \overrightarrow{A}|_a\Delta \Rightarrow B}{\Delta \Rightarrow A \Downarrow B}\,\Downarrow R$$

Fig. 7. Hypersequent calculus rules for nondeterministic synthetic connectives

3 Strong Completeness of the Implicative Fragment w.r.t. L-Models

In this section we prove two strong completeness theorems in relation to the implicative fragment. In order to prove them, we demonstrate first strong completeness of $\mathbf{hD}[\rightarrow]$ w.r.t. powerset residuated DAs over standard DAs with a countable set of generators.

Let $V = \mathbf{seg}[\rightarrow] \cup \{1\}$. Clearly, V is countably infinite since $\mathbf{seg}[\rightarrow]$ is the countable union $\bigcup_i \mathbf{seg}[\rightarrow]_i$, where each $\mathbf{seg}[\rightarrow]_i$ is also countably infinite. Let us consider the standard DA $\mathcal{S}$ (from (12)), induced by the (countably) infinite set of generators V:

$$\mathcal{S} = (V^*, +, \{|_i\}_{i>0}, \Lambda, 1)$$

We define some notation:

Definition 3. *For any type $C \in \mathbf{Tp}[\rightarrow]$ and set R of non-logical axioms:*

$$[C]_R = \{\Delta : \Delta \in \mathcal{O} \text{ and } R \vdash \Delta \Rightarrow C\}$$

In practice, when the set of hypersequents R is clear from the context, we simply write $[C]$ instead of $[C]_R$.

Lemma 3 (Truth Lemma). Let $\mathcal{P}(S)$ be the powerset residuated DA over the standard DA $\mathcal{S}$ from (12). Let v_R be the following valuation on the powerset $\mathcal{P}(S)$:

$$\text{For every } p \in \mathbf{Pr},\ v_R(p) = [p]_R$$

Let $\mathcal{M} = (\mathcal{P}(S), v_R)$ be called as usual the canonical model. The following equality holds:

$$\text{For every } C \in \mathbf{Tp}[\rightarrow],\ \widehat{v}_R(C) = [C]_R$$

Proof. We proceed by induction on the structure of type C; we will write $\widehat{v}$ instead of $\widehat{v_R}$, and $[\cdot]$ instead of $[\cdot]_R$; we will say that an element $\Delta \in \widehat{v}(A)$ is *correct*[5] iff $\Delta \in \mathcal{O}[\mathbf{C}]$.

- C is primitive. True by definition.

- $C = B{\uparrow_i}A$. Let us see:

$$[B{\uparrow_i}A] \subseteq \widehat{v}(B{\uparrow_i}A)$$

Let Δ be such that $R \vdash \Delta \Rightarrow B{\uparrow_i}A$. Let $\Gamma_A \in \widehat{v}(A)$. By induction hypothesis (i.h.), $\widehat{v}(A) = [A]$. Hence, $R \vdash \Gamma_A \Rightarrow A$ We have:

$$\frac{\Delta \Rightarrow B{\uparrow_i}A \qquad \overrightarrow{B{\uparrow_i}A}|_i\Gamma_A \Rightarrow B}{\Delta|_i\Gamma_A \Rightarrow B}\,Cut$$

By i.h., $\widehat{v}(B) = [B]$. It follows that $\Delta|_i\Gamma_A \in \widehat{v}(B)$, hence $\Delta \in \widehat{v}(B{\uparrow_i}A)$. Whence, $[B{\uparrow_i}A] \subseteq \widehat{v}(B{\uparrow_i}A)$.
Conversely, let us see:

$$\widehat{v}(B{\uparrow_i}A) \subseteq [B{\uparrow_i}A]$$

Let $\Delta \in \widehat{v}(B{\uparrow_i}A)$. By i.h. $\widehat{v}(A) = [A]$. For any type A, we have eta-expansion, i.e. $\overrightarrow{A} \Rightarrow A$[6]. Hence, $\overrightarrow{A} \in \widehat{v}(A)$. We have that $\Delta|_i\overrightarrow{A} \in \widehat{v}(B)$. By i.h., $\Delta|_i\overrightarrow{A} \Rightarrow B$. Since $\overrightarrow{A}$ is correct, and by i.h. $\Delta|_i\overrightarrow{A}$ is correct, by Lemma 2, Δ is correct. By applying the $\uparrow_i$ right rule to the provable hypersequent $\Delta|_i\overrightarrow{A} \Rightarrow B$ we get:

$$\Delta \Rightarrow B{\uparrow_i}A$$

This ends the case of $B{\uparrow_i}A$.

- $C = A{\downarrow_i}B$. Completely similar to case $B{\uparrow_i}A$.

- $C = B/A$ or $A\backslash B$. Similar to the disconcontuous case.
- Nondeterministic connectives. Consider the case $C = B{\Uparrow}A$.

$$[B{\Uparrow}A] \subseteq \widehat{v}(B{\Uparrow}A)$$

[5] Recall that a priori $\Delta \in |\mathcal{S}|$, which is equal to $(\mathbf{Seg}[\rightarrow] \cup \{1\})^*$.
[6] By simple induction on the structure of types.

Let $\Gamma_A \in \widehat{v}(A)$. By i.h., $\Gamma_A \Rightarrow A$. Let $\Delta \Rightarrow B{\Uparrow}A$. By $s(B) - s(A) + 1$ applications of $\Uparrow$ left rule, we have

$$\frac{\Gamma_A \Rightarrow A \qquad \overrightarrow{B} \Rightarrow B, \text{ by eta-expansion}}{\overrightarrow{B{\Uparrow}A}|_i\Gamma_A \Rightarrow B, \text{ for } i = 1, \cdots, s(B) - s(A) + 1} {\Uparrow}L$$

By $s(B) - s(A) + 1$ Cut applications with $\Delta \Rightarrow B{\Uparrow}A$, we get:

$$\Delta|_i\Gamma_A \Rightarrow B$$

Hence, for $i = 1, \cdots s(B)-s(A)+1$, by i.h. $\Delta|_i\Gamma_A \in \widehat{v}(B)$. Hence, $\Delta \in \widehat{v}(B{\Uparrow}A)$. Conversely, let us see:

$$\widehat{v}(B{\Uparrow}A) \subseteq [B{\Uparrow}A]$$

By i.h., we see that $\overrightarrow{A} \in \widehat{v}(A)$. Let $\Delta \in \widehat{v}(B{\Uparrow}A)$. This means that for every $i = 1, \cdots, s(B) - s(A) + 1\Delta|_i\overrightarrow{A} \in \widehat{v}(B)$. By i.h., $\Delta|_i\overrightarrow{A} \Rightarrow B$. By a similar reasoning to the deterministic case $C = B{\uparrow_i}A$, we see that Δ is correct. We have that:

$$\frac{\Delta|_1\overrightarrow{A} \Rightarrow B \qquad \cdots \qquad \Delta|_{s(B)-s(A)+1}\overrightarrow{A} \Rightarrow B}{\Delta \Rightarrow B{\Uparrow}A} {\Uparrow}R$$

- $C = A{\Downarrow}B$ is completely similar to the previous one.

- $C = {\lhd^{-1}}A$. Let us see:

$$[{\lhd^{-1}}A] \subseteq \widehat{v}({\lhd^{-1}}A)$$

Let $\Delta \in [{\lhd^{-1}}A]$. Hence, $\Delta \Rightarrow {\lhd^{-1}}A$. We have that:

$$\frac{\Delta \Rightarrow {\lhd^{-1}}A \qquad \dfrac{\overrightarrow{A} \Rightarrow A}{\overrightarrow{{\lhd^{-1}}A}, 1 \Rightarrow A}\,{\lhd^{-1}}L}{\Delta, 1 \Rightarrow A}\,Cut$$

By i.h., $\Delta, 1 \in \widehat{v}(A)$. Hence, $\Delta \in \widehat{v}({\lhd^{-1}}A)$.
Conversely, let us see:

$$\widehat{v}({\lhd^{-1}}A) \subseteq [{\lhd^{-1}}A]$$

Let $\Delta \in \widehat{v}({\lhd^{-1}}A)$. By definition, $\Delta, 1 \in \widehat{v}(A)$. By i.h., $\Delta, 1 \Rightarrow A$, and by Lemma 2, Δ is correct. By application of ${\lhd^{-1}}$ right rule, we get:

$$\Delta \Rightarrow {\lhd^{-1}}A$$

This proves the converse.

- $C = {\rhd^{-1}}A$ is completely similar to the previous one.

- $C = {}^{\check{}_k}A$. Let us see:

$$[{}^{\check{}_k}A] \subseteq \widehat{v}({}^{\check{}_k}A)$$

Let $\Delta \Rightarrow {}^{\check{}_k}A$. We have that:

$$\dfrac{\Delta \Rightarrow {}^{\check{}_i}A \qquad \dfrac{\overrightarrow{A} \Rightarrow A}{\overrightarrow{{}^{\check{}_i}A}|_k\Lambda \Rightarrow A}\,{}^{\check{}_k}L}{\Delta|_k\Lambda \Rightarrow A}\,Cut$$

By i.h., $\Delta \in \widehat{v}({}^{\check{}_k}A)$.
Conversely, let us see that:

$$\widehat{v}({}^{\check{}_k}A) \subseteq [{}^{\check{}_k}A]$$

Let $\Delta \in \widehat{v}({}^{\check{}_k}A)$. By definition, $\Delta|_k\Lambda \in \widehat{v}(A)$. By i.h. and Lemma 2, Δ is correct and $\Delta|_k\Lambda \Rightarrow A$. By application of the ${}^{\check{}_k}$ right rule:

$$\Delta \Rightarrow {}^{\check{}_k}A$$

Hence, $\Delta \in [{}^{\check{}_k}A]$. □

By induction on the structure of $\mathcal{O}$, see (10), one proves the following lemma:

Lemma 4 (Identity lemma). For any $\Delta \in \mathcal{O}$, $\Delta \in \widehat{v}(\Delta)$.

Let $(A_i)_{i=1,\cdots,n}$ be the sequence of type-occurrences in a configuration Δ. Let $\Delta\begin{pmatrix}\Gamma_1 \cdots \Gamma_n\\ A_1 \cdots A_n\end{pmatrix}$ be the result of replacing every type-occurrence A_i with Γ_i. Recall that we have fixed a set of hypersequents R. We have the lemma:

Lemma 5. $\mathcal{M} = (\mathcal{P}(S), v) \models R$.

Proof. Let $(\Delta \Rightarrow A) \in R$. For every type-occurrence A_i in Δ (we suppose that the sequence of type occurrences in Δ is $(A_i)_{i=1,\cdots,n}$), we have by the Truth Lemma that $\widehat{v}(A_i) = [A_i]_R$. For any $\Gamma_i \in \widehat{v}(A_i)$, we have by the Truth Lemma that $R \vdash \Gamma_i \Rightarrow A_i$. Since $(\Delta \Rightarrow A) \in R$, we have then that $R \vdash \Delta \Rightarrow A$. By n applications of the Cut rule with the premises Γ_i we get from $R \vdash \Delta \Rightarrow A$ that $R \vdash \Delta\begin{pmatrix}\Gamma_1 \cdots \Gamma_n\\ A_1 \cdots A_n\end{pmatrix} \Rightarrow A$. We have that $\widehat{v}(\Delta) = \{\Delta\begin{pmatrix}\Gamma_1 \cdots \Gamma_n\\ A_1 \cdots A_n\end{pmatrix} : \Gamma_i \in \widehat{v}(A_i)\}$. Since, we have $R \vdash \Delta\begin{pmatrix}\Gamma_1 \cdots \Gamma_n\\ A_1 \cdots A_n\end{pmatrix} \Rightarrow A$, again by the Truth Lemma, $\Delta\begin{pmatrix}\Gamma_1 \cdots \Gamma_n\\ A_1 \cdots A_n\end{pmatrix} \in \widehat{v}(A)$. We have then that $\widehat{v}(\Delta) \subseteq \widehat{v}(A)$. We are done. □

Theorem 2. $\mathbf{D}[\rightarrow]$ *is strongly complete w.r.t. the class* $\mathcal{PRSD}$.

Proof. Suppose $\mathcal{PRSD}(R) \models \Delta \Rightarrow A$. Hence, in particular this is true of the canonical model $\mathcal{M}$. Since $\Delta \in \widehat{v}(\Delta)$, it follows that $\Delta \in \widehat{v}(A)$. By the Truth Lemma, $\widehat{v}(A) = [A]_R$. Hence, $R \vdash \Delta \Rightarrow A$. We are done. □

We shall also prove strong completeness w.r.t. L-models over the set of connectives $\Sigma[\rightarrow]-\mathbf{split}$, where $\mathbf{split} = \{\breve{}^k : k > 0\}$. Since the canonical model $\mathcal{S}$ is countably infinite, $|\mathcal{S}|$ is in bijection with a set $V_1 = (a_i)_{i>0} \cup \{1\}$ via a mapping Φ. Let $\mathcal{A}$ be the standard DA associated to V_1. Φ extends to an isomorphism of standard DAs between $\mathcal{S}$ and $\mathcal{A}$, and then induces an isomorphism $\bar{\Phi}$ of residuated powerset DAs over standard DAs. Let $\mathcal{B}$ be a standard DA generated by the finite set of generators $V_2 = \{a, b, 1\}$. We have that $|\mathcal{A}| = V_1^*$, and $|\mathcal{B}| = V_2^*$. Let ρ be the following injective mapping from V_1 into V_2^*:

(15) $\rho(1) = 1 \quad \rho(a_i) = a + b^i + a$

The mapping ρ extends recursively to the morphism of standard DAs. Clearly ρ is injective by freeness[7] of the underlying free monoids $|\mathcal{A}|$ and $|\mathcal{B}|$. The mapping ρ is a monomorphism of DAs which induces a monomorphism $\bar{\rho}$ of residuated powerset DAs over DAs. Let A, B and C range over subsets of $|\mathcal{A}|$ such that they are non-empty and different from $\{\epsilon\}$. Since ρ is injective, so is $\bar{\rho}$. The following equalities hold:

$$\begin{array}{llll} \bar{\rho}(A \cdot B) = \bar{\rho}(A) \cdot \bar{\rho}(B) & \bar{\rho}(A \circ_i B) = \bar{\rho}(A) \circ_i \bar{\rho}(B) & \bar{\rho}(A/\!/\mathbb{J}) = \bar{\rho}(A)/\!/\bar{\rho}(\mathbb{J}) \\ \bar{\rho}(A\backslash\backslash B) = \bar{\rho}(A)\backslash\backslash\bar{\rho}(B) & \bar{\rho}(B/\!/A) = \bar{\rho}(B)/\!/\bar{\rho}(A) & \bar{\rho}(\mathbb{J}\backslash\backslash A) = \mathbb{J}\backslash\backslash\bar{\rho}(A) \\ \bar{\rho}(A \between_i B) = \bar{\rho}(A) \between_i \bar{\rho}(B) & \bar{\rho}(B \curlyvee_i A) = \bar{\rho}(B) \curlyvee_i \bar{\rho}(A) & \end{array} \tag{16}$$

The equalities (16) are due to the fact that (1) $\bar{\rho}$ is a monomorphism of DAs, (2) we can apply cancellation, and (3) the subsets considered are non-empty and different from $\{\epsilon\}$. Since $\bar{\rho}$ is injective, arbitrary families of (same-sort) subsets satisfy $\bar{\rho}(\bigcap_{i\in I} X_i) = \bigcap_{i\in I} \bar{\rho}(X_i)$. Moreover, using (16) one proves:

$$\begin{array}{l} \bar{\rho}\Big(\bigcap_{i=1}^{s(B)-s(A)+1} B \curlyvee_i A\Big) = \bigcap_{i=1}^{s(B)-s(A)+1} \bar{\rho}(B) \curlyvee_i \bar{\rho}(A) \\ \bar{\rho}\Big(\bigcap_{i=1}^{s(B)-s(A)+1} A \between_i B\Big) = \bigcap_{i=1}^{s(B)-s(A)+1} \bar{\rho}(A) \between_i \bar{\rho}(A) \end{array} \tag{17}$$

Recall that v is the valuation of the canonical model $\mathcal{P}(S)$. Consider the following composition of mappings: $\mathrm{Pr} \xrightarrow{v} \mathcal{P}(S) \xrightarrow{\bar{\Phi}} \mathcal{P}(A) \xrightarrow{\bar{\rho}} \mathcal{P}(B)$. Put $w = \bar{\rho} \circ \bar{\Phi} \circ v$. We have that $\widehat{w} = \bar{\rho} \circ \bar{\Phi} \circ \widehat{v}$. In order to prove the last equality we have to see that $\bar{\rho} \circ \bar{\Phi} \circ \widehat{v}$ is a monorphism of DAs.[8] For example, if A and B are types, one has:

$$\begin{array}{l} (\bar{\rho} \circ \bar{\Phi} \circ \widehat{v})(B\!\uparrow_k\! A) = \bar{\rho}(\bar{\Phi}(\widehat{v}(B\!\uparrow_k\! A)) = \bar{\rho}(\bar{\Phi}(\widehat{v}(B) \curlyvee_k \widehat{v}(A)) = \\ \bar{\rho}(\bar{\Phi}(\widehat{v}(B)) \curlyvee_k \widehat{v}(A)), \bar{\Phi} \text{ is an isomorphism of DAs} = \\ \bar{\rho}(\bar{\Phi}(\widehat{v}(B))) \curlyvee_k \bar{\rho}(\bar{\Phi}(\widehat{v}(A))), \bar{\rho} \text{ satisfies (16)} \end{array}$$

Similar computations give the desired equalities for the remaining considered implicative connectives.[9] Given a set of non-logical axioms R, $R \vdash_{\mathbf{hD}} \Delta \Rightarrow A$ iff $\widehat{v}(\Delta) \subseteq \widehat{v}(A)$ (we write $\widehat{v}$ instead of $\widehat{v_R}$) iff $(\bar{\Phi} \circ \widehat{v})(\Delta) \subseteq (\bar{\Phi} \circ \widehat{v})(A)$ iff $(\bar{\rho} \circ \bar{\Phi} \circ \widehat{v})(\Delta) \subseteq (\bar{\rho} \circ \bar{\Phi} \circ \widehat{v})(A)$. We have proved:

[7] Since the underlying structures are free monoids we can apply left/right cancellation.
[8] There is a unique morphism of DAs extending w.
[9] Including also projection connectives.

Theorem 3. $\mathbf{D}[\Sigma_{\rightarrow}\text{−}\mathbf{split}]$ *is strongly complete w.r.t. L-models.*

Corollary 1. $\mathbf{D}[\Sigma_{\rightarrow}\text{−}\mathbf{split}]$ *is strongly complete w.r.t. powerset residuated DAs overs standard DAs with* 3 *generators.*

References

1. Avron, A.: Hypersequents, logical consequence and intermediate logic form concurrency. Ann. Math. Stat. Artif. Intell. **4**, 225–248 (1991)
2. Buszkowski, W.: Completeness results for Lambek syntactic calculus. Zeitschrift für mathematische Logik und Grundlagen der Mathematik **32**, 13–28 (1986)
3. Goguen, J.A., Meseguer, J.: Completeness of many-sorted equational logic. Houston J. Math. **11**(3), 307–334 (1985)
4. Lalement, R.: Logique, réduction, résolution. études et recherches en informatique. Masson, Paris (1990)
5. Lambek, J.: The mathematics of sentence structure. Am. Math. Monthly **65**, 154–170 (1958). Reprinted in Buszkowski, W., Marciszewski, W., Benthem, J. (eds.) Categorial Grammar, Linguistic & Literary. Studies in Eastern Europe, vol. 25. John Benjamins, Amsterdam, pp. 153–172 (1988)
6. Moot, R.: Extended Lambek calculi and first-order linear logic. In: Casadio, C., Coecke, B., Moortgat, M., Scott, P. (eds.) Categories and Types in Logic, Language, and Physics. LNCS, vol. 8222, pp. 297–330. Springer, Heidelberg (2014)
7. Morrill, G., Valentín, O.: Spurious ambiguity and focalisation. Manuscript submitted
8. Morrill, G., Fadda, M., Valentín, O.: Nondeterministic discontinuous Lambek calculus. In: Geertzen, J., Thijsse, E., Bunt, H., Schiffrin, A. (eds.) Proceedings of the Seventh International Workshop on Computational Semantics, IWCS 2007, pp. 129–141. Tilburg University (2007)
9. Morrill, G., Valentín, O.: Displacement calculus. Linguist. Anal. **36**(1–4), 167–192 (2010). Special issue Festschrift for Joachim Lambek. http://arxiv.org/abs/1004.4181
10. Morrill, G., Valentín, O.: On calculus of displacement. In: Bangalore, S., Frank, R., Romero, M. (eds.) TAG+10: Proceedings of the 10th International Workshop on Tree Adjoining Grammars and Related Formalisms, pp. 45–52. Linguistics Department, Yale University, New Haven (2010)
11. Morrill, G., Valentín, O., Fadda, M.: Dutch grammar and processing: a case study in TLG. In: Bosch, P., Gabelaia, D., Lang, J. (eds.) TbiLLC 2007. LNCS, vol. 5422, pp. 272–286. Springer, Heidelberg (2009)
12. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. J. Logic Lang. Inform. **20**(1), 1–48 (2011). doi:10.1007/s10849-010-9129-2
13. Sorokin, A.: Normal forms for multiple context-free languages and displacement Lambek grammars. In: Artemov, S., Nerode, A. (eds.) LFCS 2013. LNCS, vol. 7734, pp. 319–334. Springer, Heidelberg (2013)
14. Valentín, O.: Theory of Discontinuous Lambek Calculus. Ph.D. thesis, Universitat Autònoma de Barcelona, Barcelona (2012)

On Some Extensions of Syntactic Concept Lattices: Completeness and Finiteness Results

Christian Wurm(✉)

Universität Düsseldorf, Düsseldorf, Germany
cwurm@phil.hhu.de

Abstract. We provide some additional completeness results for the full Lambek calculus and syntactic concept lattices, where the underlying structure is extended to tuples of arbitrary finite and infinite size. Whereas this answers an open question for finite tuples, infinite tuples have not been considered yet. Nonetheless, they have a number of interesting properties which we establish in this paper, such as a particular class of languages which results in a finite lattice.

1 Introduction

Syntactic concept lattices arise from the distributional structure of languages. Their main advantage is that they can be constructed on distributional relations which are weaker than strict equivalence. [3] has shown how these lattices can be enriched with a monoid structure to form residuated lattices. [19] has shown that the resulting class of syntactic concept lattices for arbitrary languages forms a complete class of models for the logics $\mathbf{FL}_\bot$, i.e. the full Lambek calculus, and its reducts $\mathbf{FL}, L1$, for which it is a conservative extension.

In this paper, we will consider syntactic concept lattices over extended monoids: these will no longer consist of (sets of) strings, but rather of (sets of) tuples of strings, first of arbitrary finite, then of infinite size. The monoid operation has to be modified accordingly, of course. We show that the completeness results can be extended to this case for $\mathbf{FL}_\bot$ and its reducts; our proof will be constructed on top of the completeness results in [19] by means of isomorphic embeddings.

Finite tuples have been considered in formal language theory in a huge number of different contexts; the most relevant for us are [5,15]. The use of infinite tuples has not been considered yet (to our knowledge). We show that it comes with some interesting gain in expressive power, while still being well-behaved; we also establish the largest class of language which results in a finite lattice over infinite tuples.

2 Residuated Syntactic Concept Lattices and Extensions

2.1 Equivalences on Strings and Tuples

Syntactic concept lattices originally arose in the structuralist approach to syntax, back when syntacticians tried to capture syntactic structures purely in terms of

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 164–179, 2016.
DOI: 10.1007/978-3-662-53042-9_10

distributions of strings[1] (see, e.g. [10]). An obvious way to do so is by partitioning strings/substrings into *equivalence classes*: we say that two strings w, v are equivalent in a language $L \subseteq \Sigma^*$, in symbols

(1) $w \sim_L^1 v$, iff for all $x, y \in \Sigma^*$, $xwy \in L \Leftrightarrow xvy \in L$.

This can be extended to tuples of strings of arbitrary size:

(2) $(w_1, v_1) \sim_L^2 (w_2, v_2)$, iff for all $x, y, z \in \Sigma^*$, $xw_1yv_1z \in L \Leftrightarrow xw_2yv_2z$, etc.

The problem with equivalence classes is that they are too restrictive for many purposes: assume we want to induce our grammar on the basis of a given dataset; then it is quite improbable that we get the equivalence classes we would usually desire. And even if we have an unlimited supply of examples, it seems unrealistic to describe our grammar on the basis of equivalence classes only: there might be constructions, collocations, idioms which ruin equivalences which we would intuitively consider to be adequate. Another drawback of equivalence classes is that for context-free languages, there is no general way to relate them to the non-terminals of some grammar generating the language, whereas for syntactic concepts, there are some interesting connections (see [6]).

Syntactic concepts provide a somewhat less rigid notion of equivalence, which can be conceived of as equivalence restricted to a given set of contexts. This at least partly overcomes the difficulties we have mentioned here.

2.2 Syntactic Concepts and Polar Maps

For a general introduction to lattices, see [7]; for background on residuated lattices, see [9]. Syntactic concept lattices form a particular case of what is well-known as formal concept lattice (or formal concept analysis) in computer science. In linguistics, they have been introduced in [18]. They were brought back to attention and enriched with residuation in [3,4], as they turn out to be useful representations for language learning.

Given a language $L \subseteq \Sigma^*$, $n \in \mathbb{N}$, we define two maps: a map $\triangleright : \wp((\Sigma^*)^n) \to \wp((\Sigma^*)^{n+1})$, and $\triangleleft : \wp((\Sigma^*)^{n+1}) \to \wp((\Sigma^*)^n)$, which are defined as follows:

(3) $M^{\triangleright} := \{(x_1, ..., x_{n+1}) : \forall (w_1, ..., w_n) \in M, x_1w_1...w_nx_{n+1} \in L\}$;

and dually

(4) $C^{\triangleleft} := \{(w_1, ..., w_n) : \forall (x_1, ..., x_{n+1}) \in C, x_1w_1...w_nx_{n+1} \in L\}$.

That is, a set of tuples of strings is mapped to the set of tuples of contexts in which all of its elements can occur. The dual function maps a set of contexts to the set of strings, which can occur in all of them. Usually, the case where $n = 1$ has been in the focus, as in [3,19]. The more general cases have been considered

[1] Or words, respectively, depending on whether we think of our language as a set of words or a set of strings of words; we will choose the former option.

in one form or other by [5,15]). Obviously, $\triangleleft$ and $\triangleright$ are only defined with respect to a given language L and a given tuple size, otherwise they are meaningless. As long as it is clear of which language (if any particular language) and tuple size (if any particular) we are speaking, we will omit however any reference to it, to keep notation perspicuous. For a set of contexts C, $C^{\triangleleft}$ can be thought of as an equivalence class with respect to the contexts in C; but there might be elements in $C^{\triangleleft}$ which can occur in a context $(v, w) \notin C$ (and conversely). There is one more extension we will consider which is not entirely trivial, namely the one from tuples of *arbitrary* size to tuples of *infinite* size.

(5) for $M \subseteq (\Sigma^*)^\omega$, $M^{\triangleright} := \{(x_1, x_2, ...) : \forall (w_1, w_2, ...) \in M, x_1 w_1 x_2 w_2 ... \in L\}$.

One might consider this meaningless, as L consists of finite words, M of infinite tuples. But this is unjustified: it only entails that for any infinite tuple $\overline{w} \in M$ or $\overline{w} \in M^{\triangleright}$, in order to be "meaningful", all but finitely many components must be ϵ. So for each "meaningful" $(w_1, w_2, ...)$, there is a $k \in \mathbb{N}$ such that for all $j \geq k$, $w_j = \epsilon$. We gladly accept this restriction and remain with tuples where *almost all* components are empty. This is still a proper generalization of any tuple size n, because there is no finite upper bound for non-empty components in sets of tuples.

We define $\cdot$ on (finite or infinite) tuples by componentwise concatenation, that is, $(w_1, w_2, ...) \cdot (v_1, v_2, ...) = (w_1 v_1, w_2 v_2, ...)$. This choice is not unquestionable: some authors seem to prefer concatenation of the type $\oplus$, where $(w, v) \oplus (x, y) = (wx, yv)$. In the context of multiple context-free grammars this is referred to as *well-nestedness* and has attracted great interest (see e.g. [12]). The problem with this type of concatenation is that it is not easily extended beyond tuples of size two. What is interesting in this context is that we can use the ω-tuples to *simulate* $\oplus$-style concatenation with $\cdot$-style concatenation. To briefly sketch what this means, we define the forgetful map fo by $\mathsf{fo}(w_1, w_2, ...) = w_1 w_2 ...$, for arbitrary finite/infinite tuple size. We can now for all sequences of tuples $(x_1, y_1), ..., (x_i, y_i)$ devise ω-tuples $\overline{w}_1, ..., \overline{w}_i$ such that for all $1 \leq j, j' \leq i$, we have

$$\mathsf{fo}((x_j, y_j) \oplus ... \oplus (x_{j'}, y_{j'})) = \mathsf{fo}(\overline{w}_j \cdot ... \cdot \overline{w}_{j'})$$

This is not generally possible with any finite tuple size, and this is what makes infinite tuples interesting for us. Note that we can now also simplify things for ω-tuples, as we have $\overline{v} \in \{\overline{w}\}^{\triangleright}$ iff $\mathsf{fo}(\overline{v} \cdot \overline{w}) \in L$.

Regardless of the underlying objects, the two compositions of the maps, $\triangleleft\triangleright$ and $\triangleright\triangleleft$, are closure operators, that is:

1. $M \subseteq M^{\triangleright\triangleleft}$,
2. $M^{\triangleright\triangleleft} = M^{\triangleright\triangleleft\triangleright\triangleleft}$,
3. $M \subseteq N \Rightarrow M^{\triangleright\triangleleft} \subseteq N^{\triangleright\triangleleft}$,

for $M, N \subseteq \Sigma^*$. The same holds for contexts and $\triangleleft\triangleright$. A set M is **closed**, if $M^{\triangleright\triangleleft} = M$ etc. The closure operator $\triangleright\triangleleft$ gives rise to a lattice $\langle \mathcal{B}_L^n, \leq \rangle$, where the elements of $\mathcal{B}_L^n$ are the sets $M \subseteq (\Sigma^*)^n$ such that $M = M^{\triangleright\triangleleft}$, and $\leq$ is interpreted as $\subseteq$. The same can be done with the set of closed contexts. Given these two lattices, $\triangleright$ and $\triangleleft$ establish a Galois connection between the two:

1. $M \leq N \Leftrightarrow M^{\triangleleft} \geq N^{\triangleleft}$, and
2. $C \leq D \Leftrightarrow C^{\triangleright} \geq D^{\triangleright}$.

A syntactic concept A is usually defined to be an ordered pair, consisting of a closed set of strings, and a closed set of contexts, so it has the form $\langle S, C\rangle$, such that $S^{\triangleright} = C$ and $C^{\triangleleft} = S$. For our purposes, we only need to consider the left component, so we suppress the contexts and only consider the stringsets of the form $M^{\triangleright\triangleleft}$. For all operations we define below, it can be easily seen that the resulting structures are isomorphic. So when we refer to a concept, we only mean a $[-]^{\triangleright\triangleleft}$ closed set of strings, the concept in the classical sense being easily reconstructible.

Definition 1. *The set of concepts of a language L forms a lattice denoted by $SCL_n(L) := \langle \mathcal{B}_L^n, \wedge, \vee, \top, \bot\rangle$, where $\top = (\Sigma^*)^n$, $\bot = \emptyset^{\triangleright\triangleleft}$, and for $M, N \in \mathcal{B}_L^n$, $M \wedge N = M \cap N$, $M \vee N = (M \cup N)^{\triangleright\triangleleft}$.*

It is easy to see that this defines an order in the usual fashion which coincides with $\subseteq$ on closed sets of strings. It is easy to verify that this forms a complete lattice, as infinite joins are defined by (closure of) infinite unions, infinite meets by infinite intersections. Note also that for any set of (tuples of) strings S and contexts C, $S^{\triangleright} = S^{\triangleright\triangleleft\triangleright}$ and $C^{\triangleleft} = C^{\triangleleft\triangleright\triangleleft}$. $SCL_\omega(L)$ denotes the according structure with infinite tuple size. To see that things are properly different in the infinite case, we present the following result:

Lemma 2. *1. For any $n \in \mathbb{N}$, $SCL_n(L)$ is finite iff $L \in Reg$.*
2. There are $L \in Reg$ such that $SCL_\omega(L)$ is infinite.

Let $[\Sigma]^*_{\sim_L^1}$ denote the set of $\sim_L^1$-congruence classes over Σ^*.

Proof. 1. $SCL_1(L)$ is finite iff $[\Sigma]^*_{\sim_L^1}$ is finite iff L is regular (both are well-known). Moreover, $|[\Sigma^*]_{\sim_L^{n+1}}| \leq |[\Sigma^*]_{\sim_L^n}| \cdot |[\Sigma^*]_{\sim_L^1}|$, as $\sim_L^1$-equivalent strings are equivalent in all contexts. From this the claim easily follows.

2. Take the language $L = a^*b^*$, and all tuples of the form $(a, \overbrace{\epsilon, ..., \epsilon}^{n \text{ times}}, a, \epsilon, \epsilon, ...)$ for $n \in \mathbb{N}$. For every $m, m' \in \mathbb{N}$, $m < m'$, we take the tuple $(\overbrace{a, ..., a}^{m+1 \text{ times}}, \overbrace{b, ..., b}^{m+3 \text{ times}}, \epsilon, \epsilon, ...)$; it is easy to see that $\mathsf{fo}((\overbrace{a, ..., a}^{m+1 \text{ times}}, \overbrace{b, ..., b}^{m+3 \text{ times}}, \epsilon, \epsilon, ...) \cdot (a, \overbrace{\epsilon, ..., \epsilon}^{m \text{ times}}, a, \epsilon, \epsilon, ...)) \in L$, whereas if we substitute m with m', the result is not in L.

Consequently, there are infinitely concepts, namely for each $n \in \mathbb{N}$ at least one which contains $(a, \overbrace{\epsilon, ..., \epsilon}^{n \text{ times}}, a, \epsilon, \epsilon, ...)$ but no $(a, \overbrace{\epsilon, ..., \epsilon}^{n' \text{ times}}, a, \epsilon, \epsilon, ...)$ for $n' > n$. ⊣

This raises the question: what is the class C of languages such that $L \in C$ if and only if $SCL_\omega(L)$ is finite? We will give a concise characterization of this class later on.

2.3 Monoid Structure and Residuation

As we have seen, the set of concepts of a language forms a lattice. In addition, we can also give it the structure of a monoid: for concepts M, N, we define $M \circ N := (M \cdot N)^{\triangleright\triangleleft}$, where $M \cdot N = \{\overline{w} \cdot \overline{v} : \overline{w} \in M, \overline{v} \in N\}$. We often write MN for $M \cdot N$. '$\circ$' is associative on concepts: For $M, N, O \in \mathcal{B}_n^L$, $M \circ (N \circ O) = (M \circ N) \circ O$. This follows from the fact that $[-]^{\triangleright\triangleleft}$ is a *nucleus*, that is, it is a closure operator and in addition it satisfies $S^{\triangleright\triangleleft} T^{\triangleright\triangleleft} \subseteq (ST)^{\triangleright\triangleleft}$, and the associativity of ·-concatenation (no matter on which tuple size).

Furthermore, it is easy to see that the neutral element of '$\circ$' is $\{\epsilon\}^{\triangleright\triangleleft}$. The monoid operation respects the partial order of the lattice, that is, for $X, Y, Z, W \in \mathcal{B}_n^L$, if $X \leq Y$, then $W \circ X \circ Z \leq W \circ Y \circ Z$. A stronger property is the following: $\circ$ distributes over infinite joins, that is, we have

$$\bigvee_{Z \in \mathbf{Z}} X \circ Z \circ Y = X \circ \bigvee \mathbf{Z} \circ Y$$

$\leq$ follows algebraically ($\circ$ respects the order), and $\geq$ follows from the fact that $[-]^{\triangleright\triangleleft}$ is a nucleus.

We enrich this with residuals, using the following definition:

Definition 3. *Let X, Y be concepts. We define the right residual $X/Y := \bigvee\{Z : Z \circ Y \leq X\}$, the left residual $Y\backslash X := \bigvee\{Z : Y \circ Z \leq X\}$.*

Note that this is an entirely abstract definition which does not make reference to any underlying structure. That it works is ensured by the following lemma.

Lemma 4. *Let L be a complete lattice with a monoid operation distributing over infinite joins. Then for $X, Y, Z \in L$, residuals defined as above, we have $Y \leq X\backslash Z$ iff $X \circ Y \leq Z$ iff $X \leq Z/Y$.*

Proof. We only prove the first bi-implication.

If: assume $X \circ Y \leq Z$. Then $Y \in \{W : X \circ W \leq Z\}$; so $Y \leq \bigvee\{W : X \circ W \leq Z\} = X\backslash Z$.

Only if: we have $X \circ X\backslash Z = X \circ \bigvee\{W : X \circ W \leq Z\} = \bigvee\{X \circ W : X \circ W \leq Z\} \leq Z$; as $Y \leq X\backslash Z$, we have $X \circ Y \leq X \circ X\backslash Z \leq Z$. ⊣

So every complete lattice with a monoid operation based on a nucleus can be extended to a residuated lattice.

Definition 5. *The **syntactic concept lattice** of a language L is defined as $SCL_n(L) := \langle \mathcal{B}_L^n, \wedge, \vee, \top, \bot, 1, \circ, /, \backslash \rangle$, where $\mathcal{B}_L^n, \wedge, \vee, \top, \bot$ are defined as in Definition 1, $1 = \{\epsilon\}^{\triangleright\triangleleft}$, and $\circ, /, \backslash$ are as defined above.*

Note that we somewhat overloaded the notation of $SCL_n(L)$; in the sequel we will however thereby always refer to Definition 5. Moreover, we will denote by SCL the class of all lattices of the form $SCL_1(L)$ for some language L, without any further requirement regarding L; same for SCL_n for $n \in \mathbb{N} \cup \{\omega\}$.

3 Lambek Calculus and Extensions

3.1 The Logics L, $L1$, FL and $\mathbf{FL}_\bot$

The Lambek calculus L was introduced in [13]. $L1$ is a proper extension of L, and $\mathbf{FL}, \mathbf{FL}_\bot$ are each conservative extensions of $L1$ and the preceding one. Let Pr be a set, the set of **primitive types**, and C be a set of **constructors**, which is, depending on the logics we use, $C_L := \{/, \backslash, \bullet\}$, or $C_{\mathbf{FL}} := \{/, \backslash, \bullet, \vee, \wedge\}$. By $Tp_C(Pr)$ we denote the set of types over Pr, which is defined as the smallest set, such that $Pr \subseteq Tp_C(Pr)$, and if $\alpha, \beta \in Tp_C(Pr)$, $\star \in C$, then $\alpha \star \beta \in Tp_C(Pr)$.

If there is no danger of confusion regarding the primitive types and constructors, we also simply write Tp for $Tp_C(Pr)$. We now present the inference rules corresponding to these constructors. We call an inference of the form $\Gamma \vdash \alpha$ a **sequent**, for $\Gamma \in Tp^*$, $\alpha \in Tp$, where by Tp^* we denote the set of all (possibly empty) *sequences* over Tp, which are concatenated by ',' (keep in mind the difference between *sequents*, which have the form $\Gamma \vdash \alpha$, and *sequences* like Γ, which are in Tp^*).

With few exceptions, rules of inference in our logics are not given in the form of sequents $\Gamma \vdash \alpha$, but rather as rules to derive new sequents from given ones. In general, uppercase Greek letters range as variables over sequences of types. In the inference rules for L, premises of '$\vdash$' (that is, left hand sides of sequents) must be non-empty; in $L1$ they can be empty as well; everything else is equal. In **FL** and $\mathbf{FL}_\bot$ we also allow for empty sequents. Lowercase Greek letters range over single types. Below, we present the standard rules of the Lambek calculus $L/L1$.

$$(ax) \quad \alpha \vdash \alpha \qquad (cut) \quad \frac{\Delta, \beta, \Theta \vdash \alpha \quad \Gamma \vdash \beta}{\Delta, \Gamma, \Theta \vdash \alpha}$$

$$(\mathbf{I}-/) \quad \frac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \beta/\alpha} \qquad (\mathbf{I}-\backslash) \quad \frac{\alpha, \Gamma \vdash \beta}{\Gamma \vdash \alpha\backslash\beta}$$

$$(/-\mathbf{I}) \quad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \beta/\alpha, \Gamma, \Theta \vdash \gamma} \qquad (\backslash-\mathbf{I}) \quad \frac{\Delta, \beta, \Theta \vdash \gamma \quad \Gamma \vdash \alpha}{\Delta, \Gamma, \alpha\backslash\beta, \Theta \vdash \gamma}$$

$$(\bullet-\mathbf{I}) \quad \frac{\Delta, \alpha, \beta, \Gamma \vdash \gamma}{\Delta, \alpha \bullet \beta, \Gamma \vdash \gamma} \qquad (\mathbf{I}-\bullet) \quad \frac{\Delta \vdash \alpha \quad \Gamma \vdash \beta}{\Delta, \Gamma \vdash \alpha \bullet \beta}$$

These are the standard rules of $L/L1$ (roughly as in [13]). We have rules to introduce either slash and '$\bullet$' both on the right hand side of $\vdash$ and on the left hand side of $\vdash$. We will now add two additional connectives, which are well-known from structural logics, namely $\vee$ and $\wedge$. These are not present in $L/L1$, have however been considered as extensions as early as in [14], and have been subsequently studied by [11].

$$(\wedge-\mathbf{I}\,1) \quad \frac{\Gamma, \alpha, \Delta \vdash \gamma}{\Gamma, \alpha \wedge \beta, \Delta \vdash \gamma} \qquad (\wedge-\mathbf{I}\,2) \quad \frac{\Gamma, \beta, \Delta \vdash \gamma}{\Gamma, \alpha \wedge \beta, \Delta \vdash \gamma}$$

$$(\mathbf{I}-\wedge)\qquad \frac{\Gamma\vdash\alpha\quad\Gamma\vdash\beta}{\Gamma\vdash\alpha\wedge\beta}$$

$$(\vee-\mathbf{I})\qquad \frac{\Gamma,\alpha,\Delta\vdash\gamma\quad\Gamma,\beta,\Delta\vdash\gamma}{\Gamma,\alpha\vee\beta,\Delta\vdash\gamma}$$

$$(\mathbf{I}-\vee\,1)\qquad \frac{\Gamma\vdash\alpha}{\Gamma\vdash\alpha\vee\beta}\qquad(\mathbf{I}-\vee\,2)\qquad \frac{\Gamma\vdash\beta}{\Gamma\vdash\alpha\vee\beta}$$

$$(1-\mathbf{I})\qquad \frac{\Gamma,\Delta\vdash\alpha}{\Gamma,1,\Delta\vdash\alpha}\qquad(\mathbf{I}-1)\qquad\vdash 1$$

This gives us the logic **FL**. Note that this slightly deviates from standard terminology, because usually, **FL** has an additional constant 0 (not to be confused with $\bot$!). In our formulation, 0 and 1 coincide. In order to have logical counterparts for the bounded lattice elements $\top$ and $\bot$, we introduce two logical constants, which are denoted by the same symbol.[2]

$$(\bot-\mathbf{I})\qquad \Gamma,\bot\,\Delta\vdash\alpha\qquad(\mathbf{I}-\top)\qquad\Gamma\vdash\top$$

This gives us the calculus $\mathbf{FL}_\bot$. From a logical point of view, all these extensions of L are quite well-behaved: they are conservative, and also allow us to preserve the important result of [13], namely admissibility of the cut-rule.

We say that a sequent $\Gamma\vdash\alpha$ is derivable in a calculus, if it can be derived by the axiom and the rules of inference; we then write $\Vdash_L\Gamma\vdash\alpha$, $\Vdash_{L1}\Gamma\vdash\alpha$, $\Vdash_{\mathbf{FL}}\Gamma\vdash\alpha$ etc., depending on which calculus we use.

3.2 Interpretations of $L1$, FL and $\mathbf{FL}_\bot$

The standard model for $L1$ is the class of residuated monoids. These are structures $(M,\cdot,1,\backslash,/,\leq)$, where $(M,\cdot,1)$ is a monoid, $(M,\leq)$ is a partial order, and $\cdot,/,\backslash$ satisfy the law of residuation: for $m,n,o\in M$,

$$(6)\qquad m\leq o/n\Leftrightarrow m\cdot n\leq o\Leftrightarrow n\leq m\backslash o.$$

Note that this implies that $\cdot$ respects the order $\leq$. The standard model for **FL** is the class of residuated lattices, and for $\mathbf{FL}_\bot$, the class of bounded residuated lattices. A residuated lattice is an algebraic structure $\langle M,\cdot,\vee,\wedge,\backslash,/,1\rangle$, where in addition to the previous requirements, $(M,\vee,\wedge)$ is a lattice; the lattice order $\leq$ need not be stated, as it can be induced by $\vee$ or $\wedge$: for $a,b\in M$,

[2] Whereas L and $L1$ are equally powerful in the sense of languages which are recognizable, [11] shows that **FL** is considerably more powerful than L: whereas L only recognizes context-free languages by the classical result of [17], **FL** can recognize any finite intersection of context-free languages. We only briefly mention this, because we have no space to make precise what it means for a calculus to recognize a class of languages.

$a \leq b$ is a shorthand for $a \vee b = b$. A bounded residuated lattice is a structure $\langle M, \cdot, \vee, \wedge, \backslash, /, 1, \top, \bot \rangle$, where $\langle M, \cdot, \vee, \wedge, \backslash, /, 1 \rangle$ is a residuated lattice, $\top$ is the maximal element of the lattice order $\leq$ and $\bot$ is its minimal element.

For a general introduction see [9]. We will give definitions only once for each operator; we can do so because each definition for a given connector is valid for all classes in which it is present.

We call the class of residuated monoids RM, the class of residuated lattices RL; the class of bounded residuated lattices $RL_\bot$. We now give a semantics for the calculi above. We start with an interpretation $\sigma : Pr \to M$ which interprets elements in Pr in elements of the lattice, and extend σ to $\overline{\sigma}$ by defining it inductively over our type constructors, which is for now the set $C := \{/, \backslash, \bullet, \vee, \wedge\}$. For $\alpha, \beta \in Tp_C(Pr)$,

1. $\overline{\sigma}(\alpha) = \sigma(\alpha) \in M$, if $\alpha \in Pr$
2. $\overline{\sigma}(\top) = \top$
2' $\overline{\sigma}(\top)$ is an arbitrary $m \in M$ such for all $\alpha \in Tp_C(Pr)$, $\overline{\sigma}(\alpha) \leq m$.
3. $\overline{\sigma}(\bot) = \bot$
4. $\overline{\sigma}(1) = 1$
5. $\overline{\sigma}(\alpha \bullet \beta) := \overline{\sigma}(\alpha) \cdot \overline{\sigma}(\beta)$
6. $\overline{\sigma}(\alpha / \beta) := \overline{\sigma}(\alpha) / \overline{\sigma}(\beta)$
7. $\overline{\sigma}(\alpha \backslash \beta) := \overline{\sigma}(\alpha) \backslash \overline{\sigma}(\beta)$
8. $\overline{\sigma}(\alpha \vee \beta) := \overline{\sigma}(\alpha) \vee \overline{\sigma}(\beta)$
9. $\overline{\sigma}(\alpha \wedge \beta) := \overline{\sigma}(\alpha) \wedge \overline{\sigma}(\beta)$

Note that the constructors on the left-hand side and on the right-hand side of the definition look identical (with the exception of $\bullet$ and $\cdot$), but they are not: on the left-hand side, they are type constructors, on the right hand side, they are operators of a residuated lattice. The same holds for the constants $\top, \bot, 1$. Note that there are two alternative interpretations for $\top$: one which interprets it as the upper bound for the lattice, which is the standard interpretation, and one which just interprets it as an arbitrary element. The latter will be called the **non-standard** interpretation and play some role in the sequel. Non-standard interpretations form a generalization of standard interpretations, and as we will see below, this is a proper generalization. From this it trivially follows that every completeness result which holds for standard interpretations also holds for non-standard interpretations, but we have to show that soundness is preserved. This however is also straightforward, as there is only one rule involving $\top$ and it can be easily seen to be sound under non-standard interpretations.

This is how we interpret the types of our logic. What we want to interpret next is the *sequents* of the form $\Gamma \vdash \alpha$. We say that a sequent $R = \gamma_1, ..., \gamma_i \vdash \alpha$ is true in a model $\mathcal{M}$ under assignment σ, in symbols: $(\mathcal{M}, \sigma) \models \gamma_1, ..., \gamma_i \vdash \alpha$, if and only if $\overline{\sigma}(\gamma_1 \bullet ... \bullet \gamma_i) \leq \overline{\sigma}(\alpha)$ holds in $\mathcal{M}$. That is, we interpret the ',', which denotes concatenation in sequents, as $\cdot$ in the model, and $\vdash$ as $\leq$. In the sequel, for Γ a sequence of types, we will often write $\overline{\sigma}(\Gamma)$ as an abbreviation, where we leave the former translation implicit. For the case of theorems, that is, derivable sequents with no antecedent, we have the following convention: $(\mathcal{M}, \sigma) \models \vdash \alpha$,

iff $1 \leq \overline{\sigma}(\alpha)$ in $\mathcal{M}$, where 1 is the unit element of $\mathcal{M}$ (note that this case does not arise in L).

More generally, for a given class of (bounded) residuated lattices (monoids, semigroups) $\mathfrak{C}$, we say that a sequent is *valid* in $\mathfrak{C}$, in symbols, $\mathfrak{C} \models \gamma_1, ..., \gamma_i \vdash \alpha$, if for all $\mathcal{M} \in \mathfrak{C}$ and all interpretations σ, $(\mathcal{M}, \sigma) \models \gamma_1, ...\gamma_i \vdash \alpha$ (here we have to distinguish between standard and non-standard interpretations).

4 Completeness: Previous Results

There are a number of completeness results for the logics we have considered here. We will consider the most general ones, which will be important in the sequel.

Theorem 6. *For the class RM of residuated monoids, the class RL of residuated lattices, the class $RL_\bot$ of bounded residuated lattices,*

1. *$RM \models \Gamma \vdash \alpha$ if and only if $\Vdash_{L1} \Gamma \vdash \alpha$,*
2. *$RL \models \Gamma \vdash \alpha$ if and only if $\Vdash_{\boldsymbol{FL}} \Gamma \vdash \alpha$,*
3. *$RL_\bot \models \Gamma \vdash \alpha$ if and only if $\Vdash_{\boldsymbol{FL}_\bot} \Gamma \vdash \alpha$.*

For reference on Theorem 6, see [1,2,9]. The proofs for the above completeness theorems usually proceed via the Lindenbaum-Tarski construction: we interpret primitive types as atomic terms modulo mutual derivability, and define $\sigma(\alpha) \leq \sigma(\beta)$ iff $\alpha \vdash \beta$. Then we can perform an induction over constructors to get the same for arbitrary formulas/terms. So there are quite simple completeness proofs for the general case. These completeness results can actually be strengthened to the *finite model property*. A logic, equipped with a class of models and interpretations, is said to have finite model property if it is complete in the finite; that is, Theorem 6 remains valid if we restrict ourself to finite models. These results are highly non-trivial; for example, classical first-order logic fails to have finite model property.

Theorem 7. *1. L1 has finite model property;*
*2. **FL** has finite model property;*
3. $\boldsymbol{FL}_\bot$ has finite model property.

For the first claim, consider [8]; the second and third has been established by [16]. We want to establish soundness and completeness of the calculi with respect to the class of syntactic concept lattices and their reducts. The latter results are crucial to show that completeness holds if we restrict ourselves to languages over finite alphabets.

Soundness of interpretations into SCL follows from soundness direction of Theorem 6, because SCL is just a particular class of bounded residuated lattices. As $L, L1, \mathbf{FL}$ are fragments of $\mathbf{FL}_\bot$, we get the same result for $L, L1$ and $\mathbf{FL}$, considering the terms which contain only the operators which have a counterpart in the logic.

Let SCL_{L1} be the class of SCL reducts with $\{\circ, /, \backslash\}$, which specify a unit, and SCL_{FL} be the class of SCL reducts with operators $\{\circ, /, \backslash, \vee, \wedge\}$, that is, without the constants $\top$ and $\bot$.

Theorem 8 *(Completeness).*

1. *If* $SCL_{L1} \models \Gamma \vdash \alpha$, *then* $\Vdash_{L1} \Gamma \vdash \alpha$;
2. *if* $SCL_{FL} \models \Gamma \vdash \alpha$, *then* $\Vdash_{\boldsymbol{FL}} \Gamma \vdash \alpha$;
3. *if* $SCL \models \Gamma \vdash \alpha$, *then* $\Vdash_{\boldsymbol{FL}_\bot} \Gamma \vdash \alpha$;

The completeness proofs can be found in [19]. The proof shows that for any (bounded) residuated lattice (reduct) S, there is a language $L(S)$ such that S can be isomorphically embedded in $SCL(L(S))$. This embedding thus preserves validity in both directions, and thus completeness follows. The language $L(S)$ is constructed with the elements of S as underlying alphabet. By the finite model property, we can conclude that the result remains valid if we restrict ourselves to languages over finite alphabets: if S is finite, $L(S)$ is a language over a finite alphabet (though still infinite!). So Theorem 8 also holds for languages over finite alphabets only.

5 SCL_n – Completeness via Embeddings

We now extend Theorem 8 to the structures $SCL_n : n \in \mathbb{N} \cup \{\omega\}$ (henceforth: $\mathbb{N}_\omega$). Again, we proceed by showing that there is an isomorphic embedding from $SCL(L) \to SCL_n(L)$. To increase readability and avoid misunderstandings, in the following we let $\triangleright, \triangleleft, \circ$ denote operations in SCL, $\blacktriangleleft, \blacktriangleright, \odot$ denote the corresponding operations in SCL_n. This convention however only concerns this section! We will exemplify the embedding for SCL_2, but it is easy to see that this can be extended to any $n \in \mathbb{N}$.

Assume $M \in SCL(L)$. We take a map $\alpha : \wp(\Sigma^*) \to \wp(\Sigma^* \times \Sigma^*)$, which is defined as a lifting of $\alpha' : w \mapsto (w, \epsilon)$ to sets composed with closure, so we define $\alpha(M) = (\alpha'[M])^{\blacktriangleright\blacktriangleleft}$. The following are more or less immediate:

1. $\alpha(M) \in SCL_2(L)$;
2. if $w \in M$, then $(w, \epsilon) \in \alpha(M)$;
3. $\alpha'[M] \star \alpha'[N] = \alpha'[M \star N]$, for $\star \in \{\cdot, \cup, \cap\}$.

The third point ensures that α' is a homomorphism for sets and classical set-theoretic operations of languages. Moreover, it is easy to see that α' is a bijection. So α' is an isomorphic embedding from $(\wp(\Sigma^*), \cdot, \cup, \cap)$ to $(\wp(\Sigma^* \times \Sigma^*), \cdot, \cup, \cap)$ Note that all these points remain valid if we suitably extend α' to α'_n, with $\alpha'_n(w) = (w, \overbrace{\epsilon, ..., \epsilon}^{n \text{ times}})$, with α_n defined accordingly.

This is quite obvious. It becomes much less obvious, if we switch our attention to α, that is, add the closure operation. The reason is as follows: $\alpha(M)$ might contain elements of the form (w, v) with $v \neq \epsilon$; and thus $\alpha(M) \cdot \alpha(N)$ might contain terms of the form $(w_1, w_2) \cdot (v_1, v_2) = (w_1 v_1, w_2 v_2)$. This is obviously problematic, as in $\alpha(M) \cdot \alpha(N)$ substrings occur in an order which differs from the one in $\mathsf{fo}(\alpha(M))\mathsf{fo}(\alpha(N))$. We show that the map α_n is nonetheless an isomorphic embedding $SCL(L) \to SCL_n(L)$. This requires some work, and we prove

the claim step by step via the following lemmas (again, we exemplify this for $n = 2$, but results can be easily extended to the general case). We first make the following observation:

Lemma 9. $\alpha'[M]^{\blacktriangleright} = \{(x, y, z) : (x, yz) \in M^{\triangleright}\}$;

Both inclusions are obvious. This means that $\alpha(M)$ is the set of all (a, b) such that if $xMyz \subseteq L$, then $xaybz \in L$. This allows to show the following:

Lemma 10. *For* $M = M^{\triangleright\triangleleft}$, $(w, \epsilon) \in \alpha(M)$ *if and only if* $w \in M$.

Proof. *If*-direction is immediate. *Only if*: If $(w, \epsilon) \in \alpha(M)$, then whenever $(x, y, z) \in \alpha(M)^{\blacktriangleright}$, we have $xwyz \in L$. Now, $M^{\triangleright}$ is exactly the set of all (x, yz) such that $(x, y, z) \in \alpha(M)^{\blacktriangleright}$. Thus we have $w \in M^{\triangleright\triangleleft} = M$. ⊣

Put $M^a := \{wav : wv \in M\}$.

Lemma 11. $\alpha(M) \odot \alpha(N) \subseteq \alpha(M \circ N)$.

Proof. *Case 1*: Assume $(w, v) \in \alpha(M) \cdot \alpha(N)$. Then $(w, v) = (a_1b_1, a_2b_2)$, where $(a_1, a_2) \in \alpha(M)$, $(b_1, b_2) = \alpha(N)$. So for (a_1, a_2) it follows: if $xMyz \subseteq L$, then $xa_1ya_2z \in L$; the same holds for $(b_1, b_2) \in \alpha(N)$. So we have the following:

1. if $wMv \subseteq L$ $wa_1v^{a_2} \subseteq L$, and
2. if $wNv \subseteq L$, then $wb_1v^{b_2} \subseteq L$.

Case 1a: Assume there are x, y such that $xMNy \subseteq L$. Then it follows (by 2) that for every z_1, z_2 with $z_1z_2 = y$, $xMb_1z_1b_2z_2 \subseteq L$, and consequently (by 1) that $xa_1b_1z_1a_2b_2z_2 \in L$, and so we have $(a_1b_1, a_2b_2) \in (\alpha'[M \circ N])^{\blacktriangleright\blacktriangleleft} = \alpha(M \circ N)$.

Case 1b: There are no x, y, z such that $xMNyz \subseteq L$. Then we have $M \circ N = \top$, and $MN^{\triangleright} = \emptyset$. By Lemma 9, it follows that $\alpha'[MN]^{\blacktriangleright} = \emptyset$, and therefore, $\alpha'[MN]^{\blacktriangleright\blacktriangleleft} = \alpha(M \circ N) = \top$.

Case 2: $(w, v) \notin \alpha(M) \cdot \alpha(N)$. In this case, for all (x, y, z) such that for all $(a, b) \in \alpha(M) \cdot \alpha(N)$, $xaybz \in L$, we have $xwyvz \in L$. So it follows that if $(x, y, z) \in \alpha'[M \circ N]^{\blacktriangleright}$, then $xwyvz \in L$; thus $(w, v) \in \alpha'[M \circ N]^{\blacktriangleright\blacktriangleleft} = \alpha(M \circ N)$. ⊣

This is the first of a number of lemmas which establish the main theorem of this section. The second one establishes the inverse inclusion:

Lemma 12. $\alpha(M \circ N) \subseteq \alpha(M) \odot \alpha(N)$.

Proof. Assume $(w, v) \in \alpha(M \circ N)$.

Case 1: $(w, v) \in \alpha'[M \circ N]$. Then $v = \epsilon$, and $w \in M \circ N$. For all $w \in MN$, $(w, \epsilon) \in \alpha(M) \cdot \alpha(N)$. Furthermore, if $w \in MN^{\triangleright\triangleleft}$, then $(w, \epsilon) \in (\alpha(M) \cdot \alpha(N))^{\blacktriangleright\blacktriangleleft}$ by as simple argument using Lemma 9. Consequently, $(w, v) = (w, \epsilon) \in \alpha(M) \odot \alpha(N)$.

Case 2: Assume $(w, v) \notin \alpha'[M \circ N]$. Consequently, it holds that if $(x, y, z) \in \alpha'[M \circ N]^{\blacktriangleright}$, then $xwyvz \in L$. As $\alpha'[M \circ N] \subseteq \alpha(M) \odot \alpha(N)$ (by case 1), we

have $\alpha'[M \circ N]^{\blacktriangleright} \supseteq (\alpha(M) \odot \alpha(N))^{\blacktriangleright}$, and so if $(x, y, z) \in (\alpha(M) \odot \alpha(N))^{\blacktriangleright}$, then $xwyvz \in L$, and $(w, v) \in (\alpha(M) \odot \alpha(N))^{\blacktriangleright\blacktriangleleft} = \alpha(M) \odot \alpha(N)$. ⊣

So α is a homomorphism of $\circ$ (more generally, every α_n is a $\circ$-homomorphism). To show that it preserves meets and joins does not require a lot of work. In order to save one step, we directly prove the claim for infinite meets.

Lemma 13. $\bigwedge_{i\in I} \alpha(M_i) = \alpha(\bigwedge_{i\in I} M_i)$.

This proof is rather straightforward, as $\wedge$ equals $\cap$ in our case, a fact we will make use of.

Proof. $\subseteq$ Assume $(w, v) \in \alpha(M_i)$ for all $i \in I$. This means for all $i \in I$, if $x(M_i)yz \subseteq L$, then $xwyvz \in L$. We have $(x, yz) \in (\bigwedge_{i\in I} M_i)^{\triangleright}$ iff and only if $(x, y, z) \in \alpha'[\bigwedge_{i\in I} M_i]^{\blacktriangleright}$ (Lemma 9). So if $(x, y, z) \in \alpha'[\bigwedge_{i\in I} M_i]^{\blacktriangleright}$, then $x(\bigcap_{i\in I} M_i)yz \subseteq L$, and so $xwyvz \in L$, and so $(w, v) \in \alpha'[\bigwedge_{i\in I} M_i]^{\blacktriangleright\blacktriangleleft} = \alpha(\bigwedge_{i\in I} M_i)$.

$\supseteq$ Assume $(w, v) \in \alpha(\bigwedge_{i\in I} M_i)$. Because $\alpha(X) = \alpha'[X]^{\blacktriangleright\blacktriangleleft}$, α' being a pointwise map on sets and $[-]^{\blacktriangleright\blacktriangleleft}$ being a closure operator, from $\bigwedge_{i\in I} M_i \subseteq M_j : j \in I$ it follows that $\alpha(\bigwedge_{i\in I} M_i) \subseteq \alpha(M_j)$; and so $\alpha(\bigwedge_{i\in I} M_i) \subseteq \bigcap_{i\in I} \alpha(M_i) = \bigwedge_{i\in I} \alpha(M_i)$. ⊣

Now we can use the fact that in a complete lattice, we can use meets to define joins (and vice versa). This allows us to derive the following:

Lemma 14. $\alpha(M) \vee \alpha(N) = \alpha(M \vee N)$.

Proof. We use the facts that 1. both $SCL(L), SCL_2(L)$ are complete, and 2. α preserves infinite meets. For these reasons, the following equality holds:

$$\alpha(M) \vee \alpha(N) = \bigwedge\{\alpha(X) : X \geq M, N\} = \alpha(\bigwedge\{X : X \geq M, N\}).$$

Moreover, we can easily extend this to the infinite case:

$$\bigvee_{i\in I} \alpha(M_i) = \bigwedge\{\alpha(X) : X \geq M_i : i \in I\} = \alpha(\bigwedge\{X : X \geq M_i : i \in I\}) = \alpha(\bigvee_{i\in I} M_i).$$

⊣

Again, this can be easily extended to any α_n, $n \in \mathbb{N}_\omega$.

Needless to say, every map $\alpha_n : SCL(L) \to SCL_n(L)$ is an injection. To see this, just assume we have $M, N \in SCL(L)$; and assume without loss of generality that $(w, v) \in M^{\triangleright}$, $(w, v) \notin N^{\triangleright}$. Then we have $(w, \epsilon, v) \in \alpha'[M]^{\blacktriangleright}$, but $(w, \epsilon, v) \notin \alpha'[N]^{\blacktriangleright}$, so $\alpha(M) = \alpha'[M]^{\blacktriangleright\blacktriangleleft} \neq \alpha'[N]^{\blacktriangleright\blacktriangleleft} = \alpha(N)$. This together with the with fact that we preserve joins and meets makes the following rather obvious:

Lemma 15. $X \circ N \leq M$ *iff* $\alpha(X) \odot \alpha(N) \leq \alpha(M)$.

Proof. *If*: For contraposition, assume $X \circ N \not\leq M$. Then there is $w \in X \circ N$, $w \notin M$. Consequently, there is $(w, \epsilon) \in \alpha(X) \odot \alpha(N)$, but $w \notin \alpha(M)$ (by Lemma 10).

Only if: Assume $X \circ N \leq M$. Then obviously $\alpha(X \circ N) = \alpha(X) \odot \alpha(N) \leq \alpha(M)$, as α preserves $\subseteq$. ⊣

Lemma 16. $\alpha(M)/\alpha(N) = \alpha(M/N)$

Proof. We have $M/N = \bigvee\{X : X \circ N \leq M\}$; moreover $\alpha(\bigvee\{X : X \circ N \leq M\}) = \bigvee\{\alpha(X) : X \circ N \leq M\}$. Since $X \circ N \leq M$ iff $\alpha(X) \odot \alpha(N) \leq \alpha(M)$, we have $\bigvee\{\alpha(X) : X \circ N \leq M\} = \bigvee\{\alpha(X) : \alpha(X) \odot \alpha(N) \leq \alpha(M)\} = \alpha(M)/\alpha(N)$. ⊣

Again, this proof works perfectly fine for any α_n. To not get confused with $\bot, \top$ in SCL, SCL_2, we denote the latter elements with $\bot_2, \top_2$ etc.

Lemma 17. $\alpha(\bot) = \bot_2$, *but there are languages* L *such that* $\alpha(\top) \neq \top_2$.

Proof. 1. $\bot$ We have defined $\bot = \emptyset^{\triangleright\triangleleft}$. Assume $\bot = \emptyset$; in this case, the result is obvious. Assume there is $w \in \bot$. Then for every $(x, y) \in \Sigma^* \times \Sigma^*$, $xwy \in L$. Consequently, for all $(x, y, z) \in (\Sigma^*)^3$, $xwyz \in L$. So $\alpha'[\bot]^{\blacktriangleright} = \emptyset^{\blacktriangleright}$, and $\alpha(\bot) = \bot_2$.

2. Take the language $L = a(a+b)^*a$. Then $(a, a) \in ((a+b)^*)^{\triangleright}$, where $(a+b)^* = \top$. Consequently, $(a, a, \epsilon) \in \alpha'[\top]^{\blacktriangleright}$. As $abab \notin L$, we have $(b, b) \notin \alpha'[\top]^{\blacktriangleright\blacktriangleleft} = \alpha(\top)$, hence $\alpha(\top) \neq \top_2 = \Sigma^* \times \Sigma^*$. ⊣

Again, this is easily extended to arbitrary $n \in \mathbb{N}_\omega$. So we have a serious problem, because our embedding does not preserve $\top$. We can dodge this however by considering non-standard interpretations (see Sect. 3.2 and proof of Theorem 19 below).

So this proves the first main theorem:

Theorem 18. *For every* $n \in \mathbb{N}_\omega$, *there is an isomorphic embedding* $\alpha_n : SCL(L) \to SCL_n(L)$, *such that* $\alpha_n(\bot) = \bot$, *and which in addition preserves infinite meets and joins.*

From Theorem 18 it is rather easy to extend the completeness result to SCL_n:

Theorem 19. *For arbitrary* $n \in \mathbb{N}_\omega$, $\Vdash_{\mathbf{FL}_\bot} \Gamma \vdash \alpha$ *iff* $SCL_n \models \Gamma \vdash \alpha$.

Proof. Soundness is clear and follows from more general results. Regarding completeness: Assume we have $\nVdash_{\mathbf{FL}_\bot} \Gamma \vdash \gamma$. Then there is an $L, \sigma : Pr \to SCL(L)$ such that $\overline{\sigma}(\Gamma) \not\subseteq \overline{\sigma}(\gamma)$. It follows that $\alpha_n(\overline{\sigma}(\Gamma) \not\subseteq \alpha_n(\overline{\sigma}(\gamma))$, and so $SCL_n(L), \alpha_n \circ \sigma \not\models \Gamma \vdash \gamma$, which proves the claim.

But keep in mind that $\alpha_n \circ \sigma$ is a non-standard interpretation, as $\alpha_n \circ \sigma(\top)$ need not be $\top$ in the lattice! ⊣

The results obviously also hold for the logics **FL**, $L1$ and the corresponding syntactic concept lattice reducts (for these, the notions of standard and non-standard interpretation coincide). Note also that this shows that the notion of a non-standard interpretation properly generalizes standard interpretations.

6 A Characterization for Finite SCL_ω-Structures

Obviously, if $L \notin Reg$, then $SCL_\omega(L)$ is infinite; but the converse is wrong (see Lemma 2). A permutation π is a map on words which preserves all cardinalities of all letters in this word. For any language L, define $\Pi(L) = \{w : \pi(v) = w$ for some permutation π and some $v \in L\}$. Let $PermReg$ be the class of languages, which is defined as follows:

Definition 20. *$L \in PermReg$, iff 1. $L \in Reg$, and 2. $\Pi(L) = L$.*

This concerns, for example, languages like $\{w : |w|_a$ is even for $a \in \Sigma\}$. We will show that $SCL_\omega(L)$ is finite iff $L \in PermReg$. For the *if*-direction, we first show the following lemma, which at the same time gives some understanding of the combinatorics of permutations:

Lemma 21. *Assume $L \neq \Pi(L)$, so there is a permutation π, $w \in L$, such that $\pi(w) \notin L$. Then there are $\overline{w}, \overline{v} \in (\Sigma^*)^\omega$ such that $\mathsf{fo}(\overline{v} \cdot \overline{w}) = w$, $\mathsf{fo}(\overline{w})\mathsf{fo}(v) = \pi(w)$.*

Note firstly that assumptions assure that $w \neq \pi(w)$. From this follows that $\mathsf{fo}(\overline{w}) \neq \epsilon \neq \mathsf{fo}(\overline{v})$, as this would entail $w = \pi(w)$.

Proof. We choose some arbitrary w, π such that $w \in L, \pi(w) \notin L$ (which exist by assumption). We let a_i denote the ith letter of $\pi(w)$. We construct $\overline{w}$ in the following fashion:

Step 1. Take a_1, the first letter of $\pi(w)$, and put $\overline{w} = (a_1, \epsilon, \epsilon, ...)$. Of course, there is $\overline{v} \in (\Sigma^*)^\omega$ such that $\mathsf{fo}(\overline{v} \cdot (a_1, \epsilon, ...)) = w$, because a_1 occurs in some place in w. Now there are two possible cases:

Case 1: $\mathsf{fo}(\overline{w})\mathsf{fo}(\overline{v}) \notin L$; then we change the "target permutation" π from the lemma to ξ, where $\xi(w) = \mathsf{fo}(\overline{w})\mathsf{fo}(\overline{v})$ (this is clearly a permutation). Then we are done, as ξ, w satisfy the claim!

Case 2: $\mathsf{fo}(\overline{w})\mathsf{fo}(\overline{v}) \in L$. In this case, we discard w, π and consider w^1, π^1 instead, where $w^1 = \mathsf{fo}(\overline{w})\mathsf{fo}(\overline{v}) \in L$, and π^1 is defined by $\pi^1(w^1) = \pi(w)$ (this works because w, w^1 are permutations of each other). Then continue with step 2.

Step 2. Having chosen a_i before, we now take a_{i+1} (as $\pi^i(w^i) = \pi(w)$, it does not matter which of the two we consider). Put $\overline{w} = (a_1...a_i, a_{i+1}, \epsilon, ...)$; there is obviously a $\overline{v}$ such that $\mathsf{fo}((a_1...a_i, a_{i+1}, \epsilon, ...) \cdot \overline{v}) = w^i$, because w^i is constructed as $a_1...a_i v$, and v necessarily contains the letter a_{i+1}. Now we can go back to the case distinction and repeat the procedure.

In the end, there are two possibilities: as w is a finite word, either at some point we hit case 1, and the claim follows. Assume we do *not* hit case 1. Then at some point we have $i = |w| = |\pi(w)|$, so we construct $w^{|w|}$ as $a_1...a_{|w|}$. Then by definition and assumption, we have $a_1...a_{|w|} = \pi(w) \notin L$. But we also have, as we do not hit case 1 by assumption, $a_1...a_{|w|} = \mathsf{fo}(\overline{w})\mathsf{fo}(\overline{v}) = w^{|w|} \in L$ – contradiction. ⊣

As we can see, we can even make sure that $\overline{w}$ has the form $(w, a, \epsilon, \epsilon, ...)$, and $\overline{v} = (v_1, v_2, v_3, \epsilon, \epsilon, ...)$.

Lemma 22. *$L \in PermReg$ if and only if $SCL_\omega(L)$ is finite.*

Proof. *Only if*: There are only finitely many non-equivalent concepts of the form $(w, \epsilon, \epsilon, ...)$. Moreover, by permutation closure, we know that if $\mathsf{fo}(\overline{w}) = \mathsf{fo}(\overline{v})$, then $\{\overline{w}\}^\triangleright = \{\overline{v}\}^\triangleright$ and the claim follows easily.

If: For this we need the previous lemma. We prove the contraposition, so assume $L \notin PermReg$. Then either $L \notin Reg$, and the claim follows easily. Or $\Pi(L) \neq L$. In this case, we have w, π such that $w \in L$, $\pi(w) \notin L$, and

there are $\overline{w}, \overline{v} \in (\Sigma^*)^\omega$ such that $\mathsf{fo}(\overline{v} \cdot \overline{w}) = w$, $\mathsf{fo}(\overline{w})\mathsf{fo}(v) = \pi(w)$. Moreover, $\overline{w} = (w, a, \epsilon, \epsilon, ...)$, and $\overline{v} = (v_1, v_2, v_3, \epsilon, \epsilon, ...)$.

Now for every $n \in \mathbb{N}$, we simply take a tuple $(\overbrace{\epsilon, ..., \epsilon}^{2n \text{ times}}, w, \epsilon, \epsilon, ...)$. It is clear that for every n, we get non-equivalent tuples: We have

$$(\#)\ \mathsf{fo}((\epsilon_1, ..., \epsilon_{2n}, v_1, v_2, v_3, \epsilon, \epsilon, ...) \cdot (\epsilon_1, ..., \epsilon_{2(n-1)}, w, a, \epsilon, \epsilon, ...)) = \pi(w) \notin L,$$

whereas

$$\mathsf{fo}((\epsilon_1, ..., \epsilon_{2n}, v_1, v_2, v_3, \epsilon, \epsilon, ...) \cdot (\epsilon_1, ..., \epsilon_{2n}, w, a, \epsilon, \epsilon, ...)) = w \in L.$$

Moreover, $(\#)$ holds if in the term $2n$ is replaced by any number $m \geq 2n$. Put $\overline{w}_m = (\epsilon_1, ..., \epsilon_{2m}, w, a, \epsilon, \epsilon, ...)$. So for any $\overline{w}_m, \overline{w}_n$, if $m \neq n$, then $\{\overline{w}_m\}^\triangleright \neq \{\overline{w}_n\}^\triangleright$, and as these sets are closed and there are infinitely many of them, $SCL_\omega(L)$ is infinite. ⊣

7 Conclusion

We have shown completeness results for extensions of syntactic concepts to finite and infinite tuples; moreover, we have given a precise characterization of the class of languages which result in finite lattices in all cases. Interpreting substructural logics in sets of tuples rather than sets of strings is interesting for a number of reasons: from the perspective of categorial grammar and/or Lambek calculus as language-recognizing devices, the interpretation in tuples allows us to recognize languages which are not context-free (by letting grammars recognize tuples modulo fo). This relates more "classical" categorial approaches to new approaches such as the displacement calculus **D**, which also recognizes languages which are not context-free. In this context, infinite tuples are particularly interesting, as they allow to simulate both the "wrapping"-style extended concatenation in **D** and the "crossing"-style extended concatenation we have looked at in this paper. The usage of formal concept analysis is particularly interesting in connection with learning theory; so the results here might also be of some interest for learning beyond context-free languages.

References

1. Buszkowski, W.: Completeness results for Lambek syntactic calculus. Math. Logic Q. **32**(1–5), 13–28 (1986)
2. Buszkowski, W.: Algebraic structures in categorial grammar. Theor. Comput. Sci. **1998**(1–2), 5–24 (1998)
3. Clark, A.: A learnable representation for syntax using residuated lattices. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) Formal Grammar. LNCS, vol. 5591, pp. 183–198. Springer, Heidelberg (2011)
4. Clark, A.: Learning context free grammars with the syntactic concept lattice. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 38–51. Springer, Heidelberg (2010)

5. Clark, A.: Logical grammars, logical theories. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 7351, pp. 1–20. Springer, Heidelberg (2012)
6. Clark, A.: The syntactic concept lattice: another algebraic theory of the context-free languages? J. Logic Comput. **25**(5), 1203–1229 (2013)
7. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order, 2nd edn. Cambridge University Press, Cambridge (1991)
8. Farulewski, M.: On finite models of the Lambek calculus. Studia Logica **80**(1), 63–74 (2005)
9. Galatos, N., Jipsen, P., Kowalski, T., Ono, H.: Residuated Lattices: An Algebraic Glimpse at Substructural Logics. Elsevier, Amsterdam (2007)
10. Harris, Z.S.: Structural Linguistics. The University of Chicago Press, Chicago (1963)
11. Kanazawa, M.: The Lambek calculus enriched with additional connectives. J. Logic Lang. Inf. **1**, 141–171 (1992)
12. Kanazawa, M., Michaelis, J., Salvati, S., Yoshinaka, R.: Well-nestedness properly subsumes strict derivational minimalism. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 112–128. Springer, Heidelberg (2011)
13. Lambek, J.: The mathematics of sentence structure. Am. Math. Monthly **65**, 154–169 (1958)
14. Lambek, J.: On the calculus of syntactic types. In: Jakobson, R. (ed.) Structure of Language and its Mathematica Vl Aspects, pp. 166–178. American Mathematical Society, Providence (1961)
15. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. J. Logic Lang. Inf. **20**(1), 1–48 (2011)
16. Okada, M., Terui, K.: The finite model property for various fragments of intuitionistic linear logic. J. Symb. Log. **64**(2), 790–802 (1999)
17. Pentus, M.: Lambek grammars are context free. In: Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science, Los Alamitos, California, pp. 429–433. IEEE Computer Society Press (1993)
18. Sestier, A.: Contributions à une théorie ensembliste des classifications linguistiques. In: Actes du Ier Congrès de l'AFCAL, Grenoble, pp. 293–305 (1960) (Contributions to a set-theoretical theory of classifications)
19. Wurm, C.: Completeness of full Lambek calculus for syntactic concept lattices. In: Morrill, G., Nederhof, M.-J. (eds.) Formal Grammar 2012 and 2013. LNCS, vol. 8036, pp. 126–141. Springer, Heidelberg (2013)

Formal Grammar 2016: Contributed Papers

Overtly Anaphoric Control in Type Logical Grammar

María Inés Corbalán[1(✉)] and Glyn Morrill[2]

[1] Universidade Estadual de Campinas, Campinas, Brazil
inescorbalan@yahoo.com.ar
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
morrill@cs.upc.edu

Abstract. In this paper we analyse anaphoric pronouns in control sentences and we investigate the implications of these kinds of sentences in relation to the Propositional Theory versus Property Theory question. For these purposes, we invoke the categorial calculus with limited contraction, a conservative extension of Lambek calculus that builds contraction into the logical rules for a customized slash type-constructor.

Keywords: Jaeger's calculus **LLC** · Obligatory control · Portuguese · Pronominal types · Pronouns · Type Logical Grammar

1 Introduction

In Type Logical Grammar (TLG) the analysis of an expression is a resource-conscious proof. Anaphora represents a particular challenge to this approach in that the antecedent resource is multiplied in the semantics. This duplication, which corresponds logically to the structural rule of Contraction, may be treated lexically or syntactically.

Prototypical cases of control, as exemplified in (1) and (2) below, involve a single overt nominal in the subject or object position of the main clause that appears to carry a subject semantic role in the embedded clause (cf. [9]).[1]

(1) The doctor condescended to examine John.

[1] This paper will deal only with what is called obligatory control (or exhaustive control in the nomenclature of [14]), cases where the controller must be, for each verb, a NP in a particular syntactic position in the sentence, not with types of arbitrary control (cf. [10,24,25]). An anonymous reviewer observes that our approach seems to provide resources to treat also cases of split control, where both the subject and the object matrix clause can jointly form the controlled embedded subject, as in the following Portuguese examples, where the inflection on the infinitive form (INFL) marks the plural predication:

i. Eu convenci a Maria a/de viajar<u>mos</u>.
'I convince Mary to travel.INFL.'

ii. João prometeu ao seu filho ir<u>*em*</u>ao cinema (juntos).
'John promised his son to go.INFL to the cinema (together).'

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 183–199, 2016.
DOI: 10.1007/978-3-662-53042-9_11

(2) Barnett persuaded the doctor to examine Tilman.

Therefore, in the prototypical cases it seems that there is a mismatch between the syntactic and the semantic levels of representation: there is no overt nominal in the surface subject position of the embedded clause that carries the corresponding semantic role.

In most generative theories it is assumed that a deleted copy of the overt nominal or PRO occupies the embedded subject position in some non-phonological level of syntactic representation (cf. [7,8,11,15,23]). Assuming this, most generative theories contend that the embedded clause in control structures denotes a proposition (but see [15,25]). Hence, from these perspectives, the syntactic-semantic mismatch is resolved.

Since categorial grammar is a monostratal framework, the resort to a non-surface level of syntactic representation to avoid the mismatch is not available. Notwithstanding, prototypical obligatory control structures do not present a resource problem in a monostratal grammar such as TLG when there is no syntactic embedded subject controlled by a matrix constituent. From the Type Logical point of view, embedded subjectless clauses in control structures denote a property, not a proposition (cf. [6,10]), and the lexical semantics of a control verb is multiple-bind. Hence, from a categorial perspective, the syntactic-semantic mismatch is resolved in this way. Treating the control complement as a property accounts for the sloppy reading in inferences from ellipsis of VP and quantification (cf. [6]), while treating it as a proposition does not.

Neverthelesss, if we assume the Property Theory the mismatch seems to reappear in a special kind of control structure in some pro-drop languages. The occurrence of overt, semantically controlled, pronouns in some pro-drop languages, as exemplified in (3)–(8) below, raises the question of reusing semantic resources in the context of control structures and raises the issue of the denotation of the controlled complement clause (cf. [4,16], among others):[2]

(3) Pedro quer **ele** chegar (cedo). (BP)
'Peter wants to arrive (early).'

There is not complete agreement in the literature as to whether split control is a type of obligatory control. In footnote 17 we will show how can we deal with split antecedents. As this reviewer notes such cases of control were discussed in the LFG Glue framework. Indeed, [2], following [5], observes that anaphoric control, but not functional control, allows split controlled antecedents. In future research we hope to compare our proposal for control with the one made in the related resource-sensitive formalism of LFG [2,3], which we unfortunately do not have space to discuss here.

[2] BP stands for Brazilian Portuguese, EP for European Portuguese, SPA for Spanish and IT for Italian. The infinitival subjects are highlighted. As is well known, Portuguese has an inflected form of infinitive (INFL), that is, an infinitive form that carries ending marks of agreement with their subject in both person and number. It is generally assumed that inflection is obligatory if there is an overt subject within the infinitive clause. And it is also generally assumed that the inflection must be deleted in cases where the reference of the (null) subject coincides with the reference of a matrix constituent. Thus, both rules confront each other in cases of control sentences with overt subjects.

(4) O João decidiu resolver **ele** o problema. (EP)
'John decided to solve the problem by himself.'
(5) A polícia forçou os manifestantes a **eles** saír(em). (BP)
'The police forced the protesters to leave.'
(6) María quería telefonear **ella**. (SPA)
'Mary wanted to phone.'
(7) Juan prometió a su profesor hacer **él** los deberes. (SPA)
'John promised his teacher to do the homework (personally).'
(8) Gianni me ha promesso di farlo **lui**. (IT)
'John promised me to do it (personally).'

Indeed, since there is an overt pronoun in the (pre- or post-verbal) subject position of the embedded clause, but the embedded clause denotes a property, we seem to have to assume that the denotation of the overt pronoun does not saturate the embedded predicate.

In this paper we analyse anaphoric pronouns in control sentences.[3] For these purposes, we invoke a categorial calculus with limited contraction, a conservative extension of Lambek calculus, that builds contraction into the logical rules for a customized slash type-constructor [13].

The structure of the paper is as follows. Section 2 presents Jaeger's system (in a Gentzen sequent format).[4] Section 3 considers an analysis following Jaeger's approach to pronouns and discusses some difficulties of it for the case of overt controlled pronouns. Section 4 presents an extension of Jaeger's system

[3] An anonymous reviewer indicates that the so-called Richard constructions, as exemplified below, have a number of suggestive parallel features with the variety of control which we deal with. Despite some similarities, it is important to note that Richard constructions are cases of copy raising; raising verbs, unlike control verbs, do not select for a *thematic subject* in the predicative complement, and copy raising verbs, unlike our control examples, *require* a pronominal bound copy in their complement clause:

i. Richard seems like he is ill.
ii. Richard seems like he is in trouble.

The phenomenon of copy raising is also attested in Portuguese. But two differences between typical Richard constructions and cases of copy raising in Portuguese must be pointed out: Firstly, the lexical copy is not obligatory in the Portuguese constructions; and secondly, the embedded copy in Portuguese can be not only a pronoun but also a lexical DP (cf. [4]):

i. Acabou por ir **ele**/**o João** ao mercado.
'It ended up being the case that he/John went to the market.'

For a treatment of copy raising in a resource-conscious framework or in a generative grammar we refer the interested reader to [1,21], respectively.

[4] A anonymous reviewer objects to the use of Gentzen format as "about as unfriendly as possible". Gentzen calculus, labelled and unlabelled natural deductions, proof nets, categorical calculus, etc. are all of repute, all have their respective advantages and disadvantages, and are all notations for the same theory. We think that it is better to try to understand each notation than to censure one. The Gentzen format is really not so hard to read.

and develops our proposal. Section 5 concludes the paper. The Appendix contains a sample of the output generated by a version of the parser/theorem-prover CatLog2 (www.cs.upc.edu/~droman/index.php) for our final proposal.

2 LLC Calculus

The Lambek calculus (**L**) with Limited Contraction (**LLC**) proposed by Jaeger [13] is a conservative extension of the Lambek-style core of TLG. In a nutshell, **LLC** extends **L** with a third kind of implication type-constructor, which compiles a limited version of the structural rule of Contraction into its left and right logical rules. Jaeger's calculus treats resource multiplication syntactically. Like Lambek calculus, Jaeger's calculus **LLC** is free of structural rules.[5]

Definition 1 (syntactic types of LLC). *Where P is a set of basic types, the set F of types of **LLC** is defined as follows:*

$$F ::= \; P \mid F\backslash F \mid F/F \mid F \bullet F \mid F|F$$

Definition 2 (semantic types). *The set T of semantic types is defined on the basis of a set δ of primitive semantic types by:*

$$T ::= \; \delta \mid T \,\&\, T \mid T \rightarrow T$$

As in **L**, the product type-constructor is semantically interpreted as Cartesian product and the implications, as function space formation. So, the category-to-type correspondence for **LLC** is given as follows:

Definition 3 (semantic type map for LLC). *The semantic type map for **LLC** is a mapping τ from syntactic types F to semantic types T such that:*

$$\tau(A \bullet B) = \tau(A) \;\&\; \tau(B)$$
$$\tau(A\backslash B) = \tau(B/A) = \tau(B|A) = \tau(A) \rightarrow \tau(B)$$

The sequent rules for the product and the slash connectives are as in Lambek calculus. The left and right rules for Jaeger's type slash | are as follows (Fig. 1):

$$\frac{\Gamma \Rightarrow M : A \qquad \Delta(x : A; y : B) \Rightarrow N : C}{\Delta(\Gamma; z : B|A) \Rightarrow N\,[M/x]\,[(z\ M)/y)] : C}\,|L$$

$$\frac{\Gamma(x_1 : C_1; \ldots; x_n : C_n) \Rightarrow M : B}{\Gamma(y_1 : C_1|A; \ldots; y_n : C_n|A) \Rightarrow \lambda z.M\,[(y_1\ z)/x_1] \ldots [(y_n\ z)/x_n] : B|A}\,|R$$

Fig. 1. Left and Right rules for |

[5] Both systems also admit the Cut rule, i.e. adding the Cut rule does not give rise to any new theorems.

The **LLC** calculus is designed to treat different linguistic phenomena related to anaphora and thus to semantic resource multiplication. Jaeger uses his calculus to treat cases of personal pronouns bound by *wh*-operators and quantifiers, and reflexives and pronouns in ellipsis of VP, among other linguistic phenomena. In **LLC** anaphoric expressions are assigned a type $B|A$ and, in particular, (personal, possesive, reflexive) pronouns are assigned the syntactic category $n|n$. In semantic terms, a pronoun denotes the identity function $\lambda x.x$ over individuals; the reference of a pronoun is identical to the reference of its antecedent.[6]

As a basic example of application of **LLC**, consider the free and bound reading of the personal pronoun in the sentence in (9):

(9) John said he walks.

In one reading, the pronoun *he* is co-referential with the subject *John*; in the other, the pronoun remains free. In the first case, the category of the clause is s with the semantics $((say'\ (walk'\ j'))\ j')$; in the other case, the category is $s|n$ and the corresponding semantics is the function $\lambda x.((say'\ (walk'\ x))\ j')$. Figure 2 outlines these two derivations.

$$\dfrac{\dfrac{}{n \Rightarrow n}\quad \dfrac{\vdots}{n, (n\backslash s)/s, n, n\backslash s \Rightarrow s}}{n, (n\backslash s)/s, n|n, n\backslash s \Rightarrow s}\,|L \qquad \dfrac{\dfrac{\vdots}{n, (n\backslash s)/s, n, n\backslash s \Rightarrow s}}{n, (n\backslash s)/s, n|n, n\backslash s \Rightarrow s|n}\,|R$$

Fig. 2. Derivations for *John said he walks*

The free and the bound readings for the pronoun can also be obtained when the matrix subject is a quantifier like *everyone*.[7]

Since we defend the Property Theory as the correct semantic analysis for the infinitive clause selected by a control verb, the syntactic category of this clause cannot be the simple type s despite the overt occurrence of a pronoun. Maintaining the Property Theory and the correct sloppy reading in ellipsis of VP, we test **LLC** in relation to anaphoric pronouns in control sentences.

3 Control Structures and Property Theory

3.1 Applying LLC to Portuguese

Besides assuming Jaeger's proposal for pronouns, we use his slash to categorize the matrix control verb. The Spanish control sentence in (6) with a (post-verbal) overt pronoun can be derived assuming the following lexical assignments:

[6] In this respect, Jaeger adopts Jacobson's proposal [12].

[7] Jaeger's proposal in itself does not capture either Binding Principle A (locality of anaphors) or Principle B (antilocality of personal pronouns). A categorial approach to locality of anaphors is given by modalities in [17] and a 'negation as failure' categorial approach including antilocality of personal pronouns is given in [20].

maría : $n : m'$
quería : $(n\backslash s)\,/\,(s|n) : \lambda x.\lambda y.\,((wanted'\ \ (x\ y))\ \ y)$
telefonear : $s/n : \lambda x.\,(phone'\ x)$
ella : $n|n : \lambda x.x$

In words, the control verb *quería* 'wanted' is assigned a functional type that takes an unsaturated sentence with a pronominal gap as its complement. The lambda operator that binds two occurrences of the same variable guarantees the control relation between the matrix subject and the embedded pronoun.[8]

Figure 3 shows the derivation of the Spanish sentence in (6) and Fig. 4 below shows the derivation of the control sentence in (3) containing a pre-verbal overt pronoun. Observe the nominal argument position in the type assigned to the infinitive embedded verb *chegar* 'to arrive'. Lexical assignments for the nominal, the pronoun and the matrix finite verb are as before.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{n \Rightarrow n}\ \dfrac{}{s \Rightarrow s}}{s/n, n \Rightarrow s}\,/L}{s/n, n|n \Rightarrow s|n}\,|R\ \ \dfrac{\dfrac{}{n \Rightarrow n}\ \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\,\backslash L}{n, (n\backslash s)\,/\,(s|n)\,, s/n, n|n \Rightarrow s}\,/L$$

Fig. 3. Derivation of *María quería telefonear* ***ella***

$$\dfrac{\dfrac{\dfrac{\dfrac{}{n \Rightarrow n}\ \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\,\backslash L}{n|n, n\backslash s \Rightarrow s|n}\,|R\ \ \dfrac{\dfrac{}{n \Rightarrow n}\ \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\,\backslash L}{n, (n\backslash s)\,/\,(s|n)\,, n|n, n\backslash s \Rightarrow s}\,/L$$

Fig. 4. Derivation of *Pedro quer* ***ele*** chegar

As we can see from Fig. 5 below, the sequent $n, (n\backslash s)/(s|n), n|n, n\backslash s \Rightarrow s|n$ is not derivable. In words, in embedded sentences selected by control verbs the

[8] In SPA and IT subjects of (adverbial or subject) infinitive constructions necessarily occupy the post-verbal position. By contrast, in BP such subjects normally occupy the pre-verbal position. In EP subjects within infinitive clauses normally occur in the post-verbal position, but the pre-verbal position can also be admitted. Hence, the nominal argument position in the embedded verb in control structures is justified:

i. Al sentir **él** los primeros síntomas de la gripe, Carlos se vacunó. (SPA)
'When he feel.INF the flu symptoms, Carlos gets the vaccine.'
ii. **Os meninos** sairem á noite preocupa suas mães. (BP/EP)
'The boys go out.INFL at night worries his mothers.'
iii. Prima di morire **papá**, mama era felice. (IT)
'Before die.INF dad, mom was happy.'

$$\dfrac{\dfrac{}{n, n\backslash s \Rightarrow s|n}\,{}^{*}\quad \dfrac{\vdots}{n, n\backslash s \Rightarrow s}\,\backslash L}{\dfrac{n, (n\backslash s)\,/\,(s|n)\,, n, n\backslash s \Rightarrow s}{n, (n\backslash s)\,/\,(s|n)\,, n|n, n\backslash s \Rightarrow s|n}\,|R}\,/L$$

Fig. 5. Illicit derivation for the type $s|n$

pronominal gap cannot be free and it has to be syntactically bound by a matrix nominal (if there is one).[9]

To sum up, it seems that applying Jaeger's proposal for pronouns is theoretically and empirically adequate for the analysis of overt pronouns occurring within control sentences in some pro-drop languages: we have used it to derive control sentences with overt pronominal subject warranting the control relation and also the Property Theory.

Notwithstanding, the previous proposal faces two adverse problems: overgeneration and undergeneration. On the one hand, we cannot prove prototypical cases of control, as in (10), that do not contain an overt embedded subject (Fig. 6).

(10) María quería telefonear.
'Mary wanted to make a phone call.'

$$\dfrac{\dfrac{}{n\backslash s \Rightarrow s|n}\,{}^{*}\quad \dfrac{\dfrac{}{n \Rightarrow n}\quad \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\,\backslash L}{n, (n\backslash s)\,/\,(s|n)\,, n\backslash s \Rightarrow s}\,/L$$

Fig. 6. Illicit derivation for *María quería telefonear*

On the other, we can derive several ungrammatical sentences, as exemplified in (11–13) below, containing a non-controlled subject expression within the complement clause selected by the subject control verb *quer* 'wants' (Fig. 7).[10]

[9] If the controller were a pronoun, as in the example below, then the complex category $s|n$ can be derived, but this is in virtue of the matrix subject pronoun.

i. Roberto, eu tentei **eu** enviar meu convite a você.
'Robert, I tried to send my invitation to you.'

[10] As in other Romance languages, object pronouns in Portuguese take the clitic form: $(l)o/(l)a$. In Brazilian spoken language the third person (non-reflexive) clitics are not commonly used; instead, the (nominative) form $ele(s)/a(s)$ is usually used for accusative object:

i. Visitei-o ontem. (EP)
ii. Visitei ele ontem. (BP)
'[I] visited him yesterday.'

(11) *Pedro$_1$ quer **ele**$_2$ ajudá-lo$_1$/ele$_1$.
'Peter wants for him to help him.'
(12) *Pedro$_1$ quer **Maria** ajudá-lo$_1$/ele$_1$.
'Peter wants Mary to date him.'
(13) *João$_1$ disse que Pedro$_2$ quer **ele**$_1$ ajudá-lo$_2$/ele$_2$.
'John said that Peter wants for him to help him.'

$$\dfrac{\dfrac{\dfrac{}{n \Rightarrow n} \quad \dfrac{\dfrac{}{n \Rightarrow n} \quad \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\backslash L}{n, (n\backslash s)/n, n \Rightarrow s}/L \quad \dfrac{\dfrac{}{n \Rightarrow n} \quad \dfrac{}{s \Rightarrow s}}{n, n\backslash s \Rightarrow s}\backslash L}{\dfrac{n, (n\backslash s)/n, n|n \Rightarrow s|n}{n, (n\backslash s)/(s|n), n, (n\backslash s)/n, n|n \Rightarrow s}|R}$$

Fig. 7. Derivation of **Pedro quer **Maria** ajudá-lo/ele*

In order to deal with these difficulties, we propose to extend **LLC**.

4 Proposal: Extending LLC

4.1 Semantically Inactive Disjunction Type

The first problem—undergeneration—can be easily tackled by adding the semantically inactive disjunction $\sqcup$ to **LLC** (cf. [18]) (Fig. 8).

$$\dfrac{\Gamma\,(x : A) \Rightarrow M(x) : C \qquad \Gamma\,(y : B) \Rightarrow M(y) : C}{\Gamma\,(z : A \sqcup B) \Rightarrow M(z) : C}\sqcup L$$

$$\dfrac{\Gamma \Rightarrow M : A}{\Gamma \Rightarrow M : A \sqcup B}\sqcup R_1 \qquad \dfrac{\Gamma \Rightarrow N : B}{\Gamma \Rightarrow N : A \sqcup B}\sqcup R_2$$

Fig. 8. Rules for semantically inactive disjunction type-constructor $\sqcup$

The optionality of the overt controlled pronoun can now be captured using such a disjunction type-constructor. We can deal with prototypical control sentences simply assigning a semantically inactive disjunction type to the complement argument of the control (transitive) verb (Fig. 9):

quería : $(n\backslash s)\,/\,((s|n) \sqcup (n\backslash s))$

$$\dfrac{\dfrac{\dfrac{\vdots}{n\backslash s \Rightarrow n\backslash s}\backslash R \quad \dfrac{}{n \Rightarrow n} \quad \dfrac{}{s \Rightarrow s}}{n\backslash s \Rightarrow (s|n) \sqcup (n\backslash s)} \sqcup R \quad \dfrac{}{n, n\backslash s \Rightarrow s}\backslash L}{n, (n\backslash s) \,/\, ((s|n) \sqcup (n\backslash s)) \,, n\backslash s \Rightarrow s} /L$$

Fig. 9. Derivation for *María quería telefonear*

4.2 Preliminary Proposal: Unlifted Pronominal Types

In order to deal with the second problem—overgeneration of pronoun distribution in control structures—it is important to note, in the first place, that even though the infinitive clauses contain a bound pronoun in (11–13) above, it does not appear as the subject, but the object of the infinitive verb *ajudar* 'to help'. The pronominal type proposed by Jaeger does not distinguish between subject and object pronouns, and so, there is no way to fix the Case of a pronoun.[11] Thus, in **LLC** subject and object (and also reflexive) pronouns are all of type $n|n$; therefore, the following clauses are both of type $s|n$ as they contain a free pronoun in some position.

(14) John saw him.
(15) He saw John.

In the second place, observe that in the three problematic examples, the embedded subject is not controlled by a matrix nominal: in (11) the subject is a free pronoun; in (12) a referential expression occupies the subject position; and in (13) the subject pronoun is bound by a higher nominal. Thus, despite the fact that the control clause contains a bound pronoun, a grammatical control relation is not exercised. While the argument type $(s|n) \sqcup (n\backslash s)$ of the control verb expresses that if there is a pronoun within the complement, it has to be bound, the type $s|n$ is not sufficient to adequately express the control conditions: (i) the infinitive subject has to be a pronoun, and (ii) it has to be bound (by a specific matrix nominal phrase). In other words, in control sentences it is necessary to ensure, first, that the embedded subject is a pronoun, and second, that the antecedent of the verbal argument type $s|n$ and that of *this* pronoun are the same. If the pronoun is the subject of an intransitive verb phrase, as in example (3) above, both conditions are correctly satisfied, but not when the complement contains the pronoun in the object position of a transitive complement.

[11] Jaeger's pronominal type does not distinguish between pre- and post-verbal position, this last difference being incorporated in the infinitive verb type in our previous proposal. As we have said before, Jaeger's proposal does not capture Principles A (locality) and B (antilocality) of the Binding Theory: that reflexive pronouns must be bound in their own clause and that accusative pronouns cannot take a c-commanding antecedent in their own clause. In order to take account of the pronominal *position*, we shall use lifted pronominal types, for example, $(s|n) \,/\, (n\backslash s)$ for pre-verbal subjects.

In order to address both of these control conditions, we propose to extend **LLC** by adding a new type-constructor || for proforms. The right and left rules for this new connective are to be the same as those of |. With this new syntactic type $B||A$ at hand, we can differentiate, in particular, between an expression containing an object pronoun $B|n$ and an expression containing a subject pronoun $B||n$.[12] Consequently, despite the fact that the sentences in (14) and (15) both contain a free pronoun, they will have different types: $s|n$ and $s||n$, respectively. In words, $s|n$ is the type for a sentence that contains a pronominal free object, and type $s||n$ corresponds to sentences with a subject free pronoun.[13]

$$\dfrac{\dfrac{\dfrac{\vdots}{n, n\backslash s \Rightarrow s}\,\backslash L}{n||n, n\backslash s \Rightarrow s||n}\,||R \quad \dfrac{\vdots}{n, n\backslash s \Rightarrow s}}{\dfrac{n||n, n\backslash s \Rightarrow (s||n) \sqcup (n\backslash s)}{n, (n\backslash s) \,/\, ((s||n) \sqcup (n\backslash s)) \,, n||n, n\backslash s \Rightarrow s}\,/L}\,\sqcup R$$

Fig. 10. Derivation for *Pedro quer* ***ele*** *chegar*

Thus the preliminary proposal, with the following lexical entries, derives control sentences without and with an overt controlled pronoun ((16) and (17–18), respectively).[14] And the derivation for a control sentence with an embedded referential subject and a bound object pronoun (as in (19)) is blocked (Figs. 10 and 11):[15]

ajudar : $(n\backslash s)\,/n : \lambda x.\lambda y.\,((help'\ y)\ \ x)$
chegar : $n\backslash s : \lambda x.\,(arrive'\ x)$

[12] An anonymous reviewer observes that this difference could be made by using features instead of introducing a new connective. Although we could have chosen that option, we have preferred to extend Jaeger's proposal because using both proforms we can obtain the corresponding lifted types, and so, we can also distinguish between a pre- and a post-verbal pronoun.

[13] Observe that the double free pronoun reading $\lambda x.\lambda y.((saw'\ y)\ x)$ for *He saw him*, which in Jaeger's system gets the category $(s|n)\,|n$, corresponds on our proposal to the type $(s||n)\,|n$ or $(s|n)||n$.

[14] Note that the bound reading for the object $n|n$ is obtained by using, not |R, but |L.

[15] There are two readings for *Pedro quer* ***ele*** *ajudar ele*: a reflexive bound reading $((wanted'\ \ ((help'\ p)\ \ p))\ \ p)$ and a free reading $\lambda x.\,((wanted'\ \ ((help'\ x)\ \ p))\ \ p)$. In the first case, the clause corresponds to the type s and in the second, to the type $s|n$. There is no derivation for the type $s||n$ with the subject free reading $\lambda x.\,((wanted'\ \ ((help'\ p)\ \ x))\ \ p)$. Considering that in BP the object pronoun can take the nominative form, it seems we have to take the sequence of types $n, (n\backslash s)\,/\,((s||n) \sqcup (n\backslash s))\,, n, (n\backslash s)\,/n, n||n$ into account for the sentence in (12). But, it must to be remembered that infinitive subjects are usually preverbal in BP.

ele (he) : $n||n : \lambda x.x$
joão : $n : j'$
lo/ele (him) : $n|n : \lambda x.x$
pedro : $n : p'$
quer : $(n\backslash s) \,/\, ((s||n) \sqcup (n\backslash s)) : \lambda x.\lambda y.\, ((want'\ \ (x\ y))\ \ y)$

(16) Pedro quer chegar.
(17) Pedro quer **ele** chegar.
(18) Pedro quer **ele** ajudá-lo/ele.
(19) *Pedro quer **João** ajudá-lo/ele.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{n \Rightarrow n} \quad \dfrac{}{n, (n\backslash s)\,/n, n \Rightarrow s||n}\,*}{n, (n\backslash s)\,/n, n|n \Rightarrow s||n}\,{|L} \quad \vdots}{n, (n\backslash s)\,/n, n|n \Rightarrow (s||n) \sqcup (n\backslash s)}\,{\sqcup R} \quad \dfrac{}{n, n\backslash s \Rightarrow s}\,{\backslash L}}{n, (n\backslash s) \,/\, ((s||n) \sqcup (n\backslash s)), n, (n\backslash s)\,/n, n|n \Rightarrow s}\,{/L}$$

Fig. 11. Illicit derivation for *Pedro quer **João** ajudá-lo/ele*

Observe that in embedded sentences selected by propositional verbs both the free and the bound reading for a pronominal subject are possible, as in Jaeger's system. Thus, an embedded pronoun can be bound by a higher quantifier or another nominal expression when it occurs within complement clauses of propositional verbs:[16]

(20) João disse que ele caminha.
'John said he walks.'

[16] In EP, BP, SPA and IT such a free reading for a pronoun (and even a referential expression) is also allowed even if the embedded verb has (inflected or uninflected) *infinitive form*, when the complement clause is selected by a propositional verb (cf. [16,22], among others):
i. Eu penso/afirmo terem **os deputados** trabalhado pouco. (EP)
'I think/affirm the congressmen have.INFL worked poorly.'
ii. As italianas sabem serem **elas** encantadoras. (BP)
'Italian girls know they are.INFL charming.'
iii. Este documento prueba haber **tú** nacido en 1938. (SPA)
'This document proves have.INF you was born in 1938.'
iv. Credevo avere **egli** vinto. (IT)
'[I] believed [that] he has.INF won.'

These two readings result from the following derivations (Fig. 12):

$$\dfrac{\dfrac{\vdots}{n, (n\backslash s)\,/\,s, n, n\backslash s \Rightarrow s}\,/L}{n, (n\backslash s)\,/\,s, n||n, n\backslash s \Rightarrow s||n}\,||R \qquad \dfrac{\dfrac{}{n \Rightarrow n}\quad \dfrac{\vdots}{n, (n\backslash s)\,/\,s, n, n\backslash s \Rightarrow s}\,/L}{n, (n\backslash s)\,/\,s, n||n, n\backslash s \Rightarrow s}\,||L$$

Fig. 12. Derivations for *João disse que ele caminha*

A control sentence with a prepositional control verb can also be derived assuming the anaphoric type $s||n$ as an argument of the selected preposition:[17]

(21) A polícia forçou os manifestantes a **eles** saír(em). (BP)
'The police forced the protesters to leave.'
(22) Acusa os colegas de **eles** ser(em) corruptos. (BP)
'[S/He] accuses the partners of being rascals.'

forçou : $(n\backslash s)\,/\,(n \bullet (n\backslash s)) : \lambda x.\lambda y.\,(((force\ \ (\pi_2 x\ \pi_1 x))\ \ \pi_1 x\ )\ \ y)$
a : $(n\backslash s)\,/\,((s||n) \sqcup (n\backslash s)) : \lambda x.x$
acusa : $(n\backslash s)\,/\,(n \bullet (n\backslash s)) : \lambda x.\lambda y.\,(((charge\ \ (\pi_2 x\ \pi_1 x))\ \ \pi_1 x)\ \ y)$
de : $(n\backslash s)\,/\,((s||n) \sqcup (n\backslash s)) : \lambda x.x$ (Fig. 13)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{n||n, n\backslash s \Rightarrow s||n}\,||R \quad \dfrac{\vdots}{}}{n||n, n\backslash s \Rightarrow (s||n) \sqcup (n\backslash s)}\,{\sqcup R}\ n\backslash s \Rightarrow n\backslash s \quad \dfrac{\vdots}{}}{(n\backslash s)\,/\,((s||n) \sqcup (n\backslash s))\,, n||n, n\backslash s \Rightarrow n\backslash s}\,/L\ n/cn, cn \Rightarrow n}\,/L \quad \dfrac{\vdots}{n/cn, cn, n\backslash s \Rightarrow s}\,/L}{n/cn, cn, (n\backslash s)\,/\,(n \bullet (n\backslash s))\,, n/cn, cn, (n\backslash s)\,/\,((s||n) \sqcup (n\backslash s))\,, n||n, n\backslash s \Rightarrow s}\,/L$$

Fig. 13. Derivation for *A polícia forçou os manifestantes a* ***eles*** *sair*

[17] To analyse split control as exemplified by the Portuguese sentence below, we suggest the following lexical entry for the prepositional control verb *convencer a/de* 'convince', where g groups individuals:

i. Eu convenci a Maria a/de viajarmos.
'I convince Mary to travel.INFL.'

convenci : $(n\backslash s)\,/\,(n \bullet (n\backslash s)) : \lambda x.\lambda y.\,(((convinced\ \ (\pi_2 x\ \pi_1 x))\ \ g(y, \pi_1 x))\ \ y)$
a/de : $(n\backslash s)\,/\,((s||n) \sqcup (n\backslash s)) : \lambda x.x$

Although the previous proposal ensures that the subject embedded pronoun is controlled, the type $n||n$ does not block the derivation of ungrammatical sentences which contain a subject pronoun within a nominal phrase in the embedded subject position,[18] as exemplified below, neither captures the antilocality principle for the object type $n|n$:[19]

(23) *Pedro quer **o fato de que ele chegou** ser comemorado.
'Peter wants the fact that he arrive be.INF celebrated.'

4.3 Final Proposal: Lifted Pronominal Types

Like a type n, a pronominal type $n|n$ (and $n||n$) can also be lifted in **LLC**. In other terms, in addition to (24) and (25), the sequents (26) and (27) can also be derived in Jaeger's system:[20]

(24) $n \Rightarrow (s/n)\backslash s$
(25) $n \Rightarrow s/(n\backslash s)$
(26) $n|n \Rightarrow (s/n)\backslash(s|n)$
(27) $n|n \Rightarrow (s|n)/(n\backslash s)$

Observe, in the first place, that lifted types are differentiated not only with respect to the position—left or right—of the argument, but also with respect to the type of the argument—s/n or $n\backslash s$. It seems clear that the lifted type $(s|n)/(n\backslash s)$ (or $(s||n)/(n\backslash s)$) in our proposal) could be used to categorize a pre-verbal subject pronoun, as it selects a verb phrase to the right.

For accusative pronouns we follow a strategy of lifting ('case as lifting') as well as has been done for nominative pronouns. There are two facts in particular which we capture. First, that accusative pronouns appear in non-subject positions,[21] and second, that they cannot take a subject antecedent in their own

[18] Since the pronoun *ele* is assigned the type $n||n$ and the nominal phrase *os amigos d'ele* contains this pronoun it seems that we have to admit that the pronominal type $n||n$ is also assigned to it, and consequently, the sentence in (ii) could be derived:

ii. João quer **os amigos d'ele** chegar(em).
'John wants his friends to arrive(.INFL)'

Nevertheless, in this case *ele* is the complement of the preposition, and it is used as the third person possessive pronoun in order to avoid the ambiguity between the second and the third reading for the possessive *seu(s)* ('your'/'his'/'her'). Observe that the preposition *de* 'of' cannot take a first pronoun as its complement: **os amigos de mim/eu/nós*.

[19] Although for reasons of space we do not do so here, we believe our eventual, lifted pronoun type, proposal can prohibit this non-locality when it is semantically modalised; cf. the way non-locality for reflexives is blocked in [17].

[20] Note that the sequents are not derivable in the reverse direction. Hence, assigning lifted types preserves some but not all of the distribution of unlifted types.

[21] In the previous proposal for control sentences this fact is captured by assigning different pronominal types for the control argument—$s||n$—and the object pronoun—$n|n$. Notwistanding, as in Jaeger's proposal, there is no way to block the occurrence of an object pronoun in a subject position.

clause (antilocality). Following [20] we assign accusative pronouns types of the form $((s \uparrow n) - (J \bullet (n \backslash s))) \downarrow (s|n)$ where $\uparrow$ and $\downarrow$ and J are the extract and infix and discontinuous unit of the displacement calculus [19], and $-$ is difference [20]. The type says that the pronoun occupies a nominal position within a sentence where the position is not subject position ($-(J \bullet (n \backslash s))$), and then seeks a nominal antecedent outside of the resulting sentence (and hence not the subject of the same sentence: antilocality).

Hence we arrive at an analysis illustrated by the mini-lexicon:

a : $Nt(s(f))/CNs(f) : \iota$
a : $(\exists aNa \backslash Saa)/\exists a((Si||Na) \sqcup (Na \backslash Si)) : \lambda AA$
acusa : $(\exists gNt(s(g)) \backslash Sf)/\exists a(Na \bullet (Na \backslash Sde)) : \lambda A \lambda B(((charge\ (\pi_2 A\ \pi_1 A))\ \pi_1 A)\ B)$
ajudar : $(\exists aNa \backslash Si)/\exists aNa : help$
chegar : $\exists aNa \backslash Si : arrive$
colegas : $\forall g CNp(g) : partners$
corruptos : $CNp(m)/CNp(m) : corrupt$
de : $(\exists aNa \backslash Sde)/\exists a((Si||Na) \sqcup (Na \backslash Si)) : \lambda AA$
decidiu : $\forall a((Na \backslash Sf)/((Si||Na) \sqcup (Na \backslash Si))) : \lambda A \lambda B((decided\ (A\ B))\ B)$
disse : $(\exists gNt(s(g)) \backslash Sf)/CPque : say$
ela : $\forall v(((Sv \uparrow Nt(s(f))) - (J \bullet (Nt(s(f)) \backslash Sv)))^{\downarrow} (Sv|Nt(s(f)))) : \lambda AA$
ela : $\forall v((Sv||Nt(s(f)))/(Nt(s(f)) \backslash Sv)) : \lambda AA$
ele : $\forall v(((Sv \uparrow Nt(s(m))) - (J \bullet (Nt(s(m)) \backslash Sv)))^{\downarrow} (Sv|Nt(s(m)))) : \lambda AA$
ele : $\forall v((Sv||Nt(s(m)))/(Nt(s(m)) \backslash Sv)) : \lambda AA$
eles : $\forall v((Sv||Nt(p(m)))/(Nt(p(m)) \backslash Sv)) : \lambda AA$
forçou : $(\exists gNt(s(g)) \backslash Sf)/\exists a(Na \bullet (Na \backslash Saa)) : \lambda A \lambda B(((force\ (\pi_2 A\ \pi_1 A))\ \pi_1 A)\ B)$
joão : $Nt(s(m)) : j$
manifestantes : $CNp(m) : protesters$
maria : $Nt(s(f)) : m$
namora : $(\exists aNa \backslash Sf)/\exists aNa : love$
namorar : $(\exists aNa \backslash Si)/\exists aNa : love$
o : $Nt(s(m))/CNs(m) : \iota$
os : $Nt(p(m))/CNp(m) : \iota$
pedro : $Nt(s(m)) : p$
policia : $CNs(f) : police$
problema : $CNs(m) : problem$
que : $CPque/Sf : \lambda AA$
quer : $\forall a((Na \backslash Sf)/((Si||Na) \sqcup (Na \backslash Si))) : \lambda A \lambda B((want\ (A\ B))\ B)$
resolver : $(\exists aNa \backslash Si)/\exists aNa : solve$
sair : $\exists aNa \backslash Si : go$
ser : $(\exists aNa \backslash Si)/\exists a(CNa/CNa) : be$

In the Appendix we give some illustrative derivations from this lexicon generated by the parser/theorem prover CatLog2.[22]

5 Conclusions

In this paper we have analyzed overt pronouns in control sentences in Portuguese. Firstly, we have followed Jaeger's proposal for pronouns and we have exposed

[22] www.cs.upc.edu/~droman/index.php.

some problems resulting from adopting the same syntactic type for both subject and object pronouns: $n|n$. We have shown how can we extend Jaeger's system in order to guarantee the control relation with the subject embedded pronoun by distinguishing between subject $n||n$ and object $n|n$ pronominal types. But this strategy was still shown to be limited as we can derive some odd ungrammatical sentences. Finally, we have suggested adoption of lifted pronominal types to avoid these cases.

The analysis can be implemented in the parser/theorem prover CatLog2. In the Appendix we show analyses with lifted types.

Acknowledgements. The first author was supported by a Doctorate scholarship (BEPE) granted by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo, process number 2015/09699-2). The second author was supported by an ICREA Acadèmia 2012, SGR2014-890 (MACDA) of the Generalitat de Catalunya and MINECO project APCOM (TIN2014-57226-P). We thank *Formal Grammar* reviewers for comments and suggestions. All errors are our own.

Appendix

(A) **pedro**+**quer**+**ele**+**chegar** : Sf
$Nt(s(m)) : p, \forall a((Na\backslash Sf)/((Si||Na)\sqcup(Na\backslash Si))) : \lambda A\lambda B((want\ (A\ B))\ B)$,
$\forall v((Sv||Nt(s(m)))/(Nt(s(m))\backslash Sv)) : \lambda CC, \exists aNa\backslash Si : arrive \Rightarrow Sf$
(See Fig. 14)

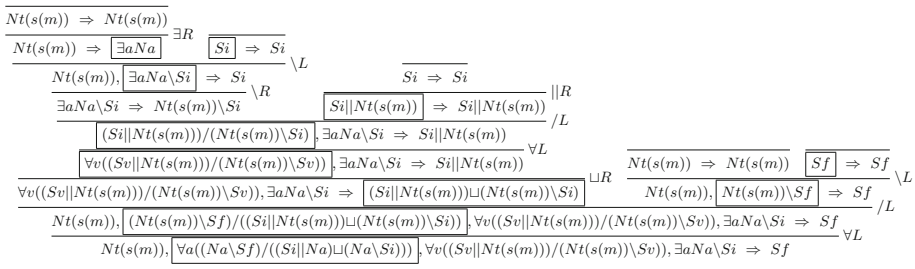


Fig. 14. Derivation of (A)

$((want\ (arrive\ p))\ p)$

(B) **a**+**policia**+**forçou**+**os**+**manifestantes**+**a**+**eles**+**ajudar**+**ele** : $Sf|Nt(s(m))$
$Nt(s(f))/CNs(f) : \iota, CNs(f) : police, (\exists gNt(s(g))\backslash Sf)/\exists a(Na\bullet(Na\backslash Saa)) :$
$\lambda A\lambda B(((force\ (\pi_2 A\ \pi_1 A))\ \pi_1 A)\ B), Nt(p(m))/CNp(m) : \iota, CNp(m) : protesters,$
$(\exists aNa\backslash Saa)/\exists a((Si||Na)\sqcup(Na\backslash Si)) : \lambda CC, \forall v((Sv||Nt(p(m)))/(Nt(p(m))\backslash Sv)) :$
$\lambda DD, (\exists aNa\backslash Si)/\exists aNa :$
$help, \forall v(((Sv\uparrow Nt(s(m)))-(J\bullet(Nt(s(m))\backslash Sv)))\downarrow(Sv|Nt(s(m)))) : \lambda EE \Rightarrow Sf|Nt(s(m))$
(See Fig. 15)
$\lambda A(((force\ ((help\ A)\ (\iota\ protesters)))\ (\iota\ protesters))\ (\iota\ police))$

Fig. 15. Derivation of (B)

References

1. Asudeh, A.: Richard III. In: Andronis, M., Debenport, E., Pycha, A., Yoshimura, K. (eds.) CLS 38: The Main Session, vol. 1, pp. 31–46. Chicago Linguistic Society, Chicago (2002)
2. Asudeh, A.: Control and Semantic Resource Sensitivity. J. Linguist. **41**(3), 465–511 (2005)
3. Asudeh, A., Mortazavinia, M.: Obligatory control in Persian: implications for the syntax-semantics interface. In: Handout from ICIL 4, Uppsala University (2011)
4. Barbosa, P.: Overt subjects in raising and control complements and the null subject parameter. In: LSA Annual Meeting Extended Abstracts (2010)
5. Bresnan, J.: Control and complementation. Linguist. Inquiry **13**, 343–434 (1982)
6. Chierchia, G.: Anaphoric properties of infinitives and gerunds. In: Cobler, M., MacKaye, S., Wescoat, M. (eds.) Proceedings of the Third West Coast Conference on Formal Linguistics, pp. 28–39. Stanford Linguistics Association, Stanford (1984)
7. Chomsky, N.: Lectures on Government and Binding. Kluwer, Dordrecht (1981)
8. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
9. Davies, W.D., Dubinsky, S.: The Grammar of Raising and Control. Blackwell Publishing, Oxford (2004)
10. Dowty, D.R.: On recent analyses of the semantics of control. Linguist. Philos. **8**(3), 291–331 (1985)
11. Hornstein, N.: Movement and control. Linguist. Inquiry **30**(1), 69–96 (1999)
12. Jacobson, P.: Towards a variable-free semantics. Linguist. Philos. **22**(2), 117–184 (1999)
13. Jaeger, G.: Anaphora and Type Logical Grammar. Springer, Dordrecht (2005)
14. Landau, I.: Elements of Control: Structure and Meaning in Infinitival Constructions. Kluwer Academic Publishers, Dordrecht (2000)
15. Landau, I.: A Two-Tiered Theory of Control. MIT Press, Cambridge (2015)
16. Mensching, G.: Infinitive Constructions with Specified Subjects. Oxford University Press, Oxford (2000)
17. Morrill, G.: Intensionality and boundedness. Linguist. Philos. **13**(6), 699–726 (1990)
18. Morrill, G.V.: Type Logical Grammar: Categorial Logic of Signs. Kluwer Academic Publishers, Dordrecht (1994)
19. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. J. Logic Lang. Inform. **20**(1), 1–48 (2011)
20. Morrill, G., Valentín, O.: Displacement logic for anaphora. J. Comput. Syst. Sci. **80**(2), 390–409 (2014)
21. Polinsky, M., Potsdam, E.: Expanding the scope of control and raising. Syntax **9**(2), 171–192 (2006)
22. Raposo, E.: Case theory and Infl-to-Comp: the inflected infinitive in European Portuguese. Linguist. Inq. **18**, 85–109 (1987)
23. Rosenbaum, P.: The Grammar of English Predicate Complement Constructions. MIT Press, Cambridge (1967)
24. Stiebels, B.: Towards a typology of complement control. In: ZAS Papers in Linguistics, vol. 47, pp. 1–80 (2007)
25. Wurmbrand, S.: Syntactic vs. semantical control. In: Zwar, C.J., Abraham, W. (eds.) Proceedings from the 15th Workshop on Comparative Germanic Syntax (2002)

A Single Movement Normal Form for Minimalist Grammars

Thomas Graf(✉), Alëna Aksënova, and Aniello De Santo

Department of Linguistics, Stony Brook University, Stony Brook, USA
mail@thomasgraf.net

Abstract. Movement is the locus of power in Minimalist grammars (MGs) but also their primary source of complexity. In order to simplify future analysis of the formalism, we prove that every MG can be converted into a strongly equivalent MG where every phrase moves at most once. The translation procedure is implemented via a deterministic linear tree transduction on the derivation tree language and induces at most a linear blow-up in the size of the lexicon.

Keywords: Minimalist grammars · Linear tree transductions · Derivation trees · Lexical blow-up · Successive cyclic movement

Introduction

Minimalist grammars (MGs; [17,18]) can be viewed as an extension of context-free grammars where the left-to-right order of leaves in the derivation tree does not necessarily correspond to their linear order in the string yield. These differences in the string yield are a side-effect of the operation *Move*, which removes a subtree from the derivation tree and reinserts it in a different position. Standard MGs are defined in such a way that one and the same subtree may be moved several times. In this case, the subtree is inserted only in the position determined by the final movement step, all previous steps have no tangible effect. This intuitive sketch suggests that these non-final—also called *intermediate*—movement steps can be omitted without altering the generated tree and string languages, a fact we prove in this paper.

The vacuity of intermediate movement is already implicit in the MCFG-equivalence proofs of [9,15]. We improve on this with a fully explicit translation in terms of a deterministic linear tree transduction over Minimalist derivation trees that yields MGs in *single movement normal form* (SMNF), i.e. MGs where every lexical item moves at most once (Sect. 2 and Appendix A). By skipping MCFGs as an intermediate step, the translation should prove easier to generalize to non-standard MGs for which no MCFG translation has been worked out in the literature. We also study the effects of SMNF on grammar size and demonstrate that the induced blow-up is highly dependent on movement configurations, but at most linear (Sect. 3.1). We furthermore discuss possible applications of SMNF (Sect. 3.2)—including certain parallels between syntax and phonology—and we

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 200–215, 2016.
DOI: 10.1007/978-3-662-53042-9_12

explore the ramifications of our result for the Chomskyan tradition of Minimalist syntax, which MGs are modeled after (Sect. 3.3).

1 Defining Minimalist Grammars

Due to space constraints we presume that the reader is already familiar with MGs in general [17,18], and their constraint-based definition in particular [5,7]. Following [14], we decompose MGs into a regular Minimalist derivation tree language (MDTL) and a mapping from derivation trees to phrase structure trees.

Definition 1. *Let* Base *be a non-empty, finite set of* feature names*. Furthermore,* Op $:= \{merge, move\}$ *and* Polarity $:= \{+, -\}$ *are the sets of* operations *and* polarities*, respectively. A* feature system *is a non-empty set* $Feat \subseteq$ Base $\times$ Op $\times$ Polarity*.*

Negative Merge features are called *category features*, positive Merge feature *selector features*, negative Move features *licensee features*, and positive Move features *licensor features*. A $(\Sigma, Feat)$-lexicon *Lex* is a finite subset of $\Sigma \times Feat^*$. Each member of *Lex* is a *lexical item* (LI) of the form $\gamma c \delta$, where γ is a string of licensor and selector features, c is a category feature, and δ is a string of 0 or more licensee features. A *Minimalist grammar* (MG) is $(\Sigma, Feat)$-lexicon coupled with a set $F \subseteq$ Base of *final categories*.

A ranked alphabet Σ is a finite union of finite sets $\Sigma^{(0)}, \ldots, \Sigma^{(n)}$ such that $\sigma \in \Sigma^{(i)}$ has *arity* i—we also write $\sigma^{(i)}$. For every such Σ, T_Σ is the language of finite Σ-trees recursively defined by (I) $\sigma^{(0)} \in T_\Sigma$, and (II) $\sigma(t_1, \ldots, t_n) \in T_\Sigma$ iff $\sigma \in \Sigma^{(n)}$ and $t_i \in T_\Sigma$, $1 \leq i \leq n$. A *free derivation tree over Lex* is a tree over alphabet $\left\{l^{(0)} \mid l \in Lex\right\} \cup \left\{\circ^{(1)}, \bullet^{(2)}\right\}$. An interior node m is *associated* to the i-th positive polarity feature of LI l iff it is the i-th mother of l (the i-th mother of l is the mother of its $(i-1)$-th mother, and l is its own 0-th mother). Such an m is the *slice root* of l iff the mother of m is not associated to l. A free derivation tree is *correctly projected* iff (I) every interior node is associated to the feature of exactly one LI, and (II) for every LI l with $\gamma := f_1 \ldots f_n$ the i-th mother of l exists and is labeled $\circ$ if f_i is a Move feature, and $\bullet$ otherwise.

A *Minimalist derivation tree* t is a correctly projected tree of some *Lex* that obeys four ancillary conditions. We first list the two constraints regulating Merge:

Merge. For every node m of t associated to selector feature f^+ of LI l, one of its daughters is the slice root of an LI l' with category feature f^-.

Final. If the root of t is the slice root of LI l, then the category feature of l is a final category.

Move is also subject to two constraints, which require ancillary terminology. Two features f and g *match* iff they differ only in their polarity. An interior node m matches a feature g iff the feature m is associated to matches g. For every free derivation tree t and LI l of t with string $\mathrm{f}_1^- \cdots \mathrm{f}_n^-$ of licensee features, $n \geq 0$, the *occurrences* of l in t are defined as follows:

– $occ_0(l)$ is the mother of the slice root of l in t (if it exists).
– $occ_i(l)$ is the unique node m of t labeled $\circ$ such that m matches $-f_i$, properly dominates occ_{i-1}, and there is no node n in t that matches $-f_i$, properly dominates occ_{i-1}, and is properly dominated by m.

The occurrence of l with the largest index is its *final occurrence*. For t and l as before, and every node m of t:

Move. There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i^{th} occurrence of l, $1 \leq i \leq n$.
SMC. If m is labeled $\circ$, there is exactly one LI for which m is an occurrence.

Given an MG G with lexicon *Lex*, the MDTL of G is the largest set of correctly projected derivation trees over *Lex* such that every tree in L satisfies the four constraints above. Note that each constraint defines a regular tree language, wherefore all MDTLs are regular [5,14,15].

MDTLs are easily mapped to phrase structure trees (see [7,14]). First we relinearize all siblings m and n such that m precedes n iff m is an LI with a selector feature or otherwise m is the slice root of some LI. Then we project phrases such that a given node is relabeled <only if its label is $\bullet$ and it is the mother of an LI with at least one selector feature. All other interior nodes are labeled>. The purpose of $<$ and $>$ is to indicate which branch (left or right, respectively) contains the head of the phrase. The result is a phrase structure tree where the first argument of a head is always linearized to its right, whereas all other arguments are linearized to the left.

For movement, assume l is an LI with at least one licensee feature and slice root r in the derivation tree. Add a branch from r to the final occurrence of l, and replace the subtree rooted by r with a trace. Every unary branching interior node furthermore receives a trace as a left daughter. At the very end, every LI $\sigma :: \mathrm{f_1 f_2 \cdots f_n}$ is replaced by σ to ensure that no features are present in the phrase structure trees. The combination of these steps can be carried out by a tree-to-tree transduction ϕ that is definable in monadic second-order logic. The tree language generated by MG G is the image of its MDTL under ϕ.

2 Single Movement Normal Form

This section establishes the core result of our paper: every MG G can be converted into a strongly equivalent MG that is in SMNF. Section 2.1 defines a linear tree transduction to rewrite MG derivation trees such that no LI moves more than once (a corresponding transducer is defined in the appendix). Section 2.2 then shows that these derivations still generate the same derived trees, which in combination with some auxiliary lemmata establishes the strong equivalence of standard MGs and MGs in SMNF.

2.1 A Linear Tree Transduction for Single Movement

In principle, single movement could mean that at most one movement step takes place in the whole derivation. But MGs satisfying such a strong constraint only generate context-free languages [cf. 12] and thus aren't even weakly equivalent to standard MGs. Instead, SMNF enforces the weaker condition that each LI undergoes at most one movement step. In other words, there is no intermediate movement; if an LI moves at all, it moves directly to its final landing site.

Definition 2 (SMNF). *An MG G with lexicon Lex is in* single movement normal form *iff every $l \in Lex$ has at most one licensee feature.*

The general idea for bringing an MG G into SMNF is very simple. Starting out with G's MDTL, one deletes from each derivation tree all those Move nodes that aren't a final occurrence for some LI. The removal of movement steps must be matched by (I) the removal of all non-final licensee features on the moving LIs and, (II) the removal of the licensor feature that each intermediate Move node was associated to.

It is this feature modification step that takes some finesse. Without further precautions, the resulting derivation may violate **SMC**, as is shown in Fig. 1 (here and in all following examples, Merge features are written in upper case and Move features in lower case). In the original derivation, LI b is the first to move, which is followed by two movement steps of LI a. Note that both undergo movement triggered by a licensee feature f^-. But the feature g^- triggering intermediate movement of a prevents f^- from becoming active on a until b has checked its feature f^-. Deleting g^- does away with this safeguard—both instances of f^- are active at the same time and receive the same occurrence, which is prohibited by **SMC**. The solution is to carefully rename features to avoid such conflicts while keeping the number of features finite.

All three steps—intermediate Move node deletion, intermediate feature deletion, and final feature renaming—can be carried out by a non-deterministic, linear bottom-up tree transducer τ, wherefore the regularity of the MDTL is preserved [3]. The definition of this transducer is rather unwieldy, so we opt for a high-level exposition at this point and relegate the detailed treatment to the appendix. Let l be an LI such that its final occurrence o is associated to feature

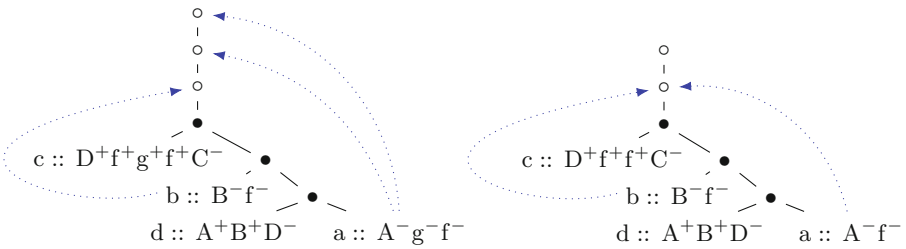


Fig. 1. Removal of intermediate movement steps may result in **SMC** violations

f and the path from o to l contains Move nodes $m_1, \ldots, m_n$ $(0 \leq n)$. Then the following steps are carried out in a bottom-up fashion:

- **Deletion**
  - remove all intermediate occurrences of l, and
  - for every removed intermediate occurrence, delete the licensor feature on the LI that said occurrence was associated to, and
- **Renaming**
  - replace l's string of licensee features by f_j^-, and
  - replace the feature on LI l' that o is associated to by f_j^+, where
  - $j \in \mathbb{N}$ is the smallest natural number distinct from every $k \in \mathbb{N}$ such that some m_i is associated to f_k^+, $1 \leq i \leq n$.

An instance of the mapping computed by τ is given in Fig. 2.

Subscripting is essential for the success of the translation. Intuitively, it may seem more pleasing to contract strings of licensee features into a single licensee feature, as suggested by one reviewer. So the LI $\sigma :: \gamma \mathrm{cf}_1^- \cdots \mathrm{f}_\mathrm{n}^-$ would become $\sigma :: \gamma \mathrm{c}[\mathrm{f}_1 \cdots \mathrm{f}_\mathrm{n}]^-$. But this does not avoid all **SMC** violations. Well-formed MG derivations can contain configurations where two LIs l and l' have the same string δ of licensee features, the final occurrence o of l dominates l', and the final occurrence of l' dominates o. In this case, atomizing δ into a single licensee feature δ^- would incorrectly make o a final occurrence of l', too.

It is also worth mentioning that even though the transducer τ as defined in the appendix is non-deterministic, the transduction itself is deterministic.

Lemma 1. *The transduction computed by τ is a function.*

Proof. Let t be some arbitrary Minimalist derivation tree and suppose that t has two distinct images t_1 and t_2 under τ. Inspection of τ reveals that t_1 and t_2 must be isomorphic and thus can only differ in the choice of indices for licensor and licensee features. We show by induction that these indices are always the same. Consider some arbitrary move node m of t_1 and t_2 that is associated to f_i^+ and f_j^+ in these derivations, respectively. If m does not properly dominate any other move nodes associated to some f_k^+, then $i = j = 0$. Otherwise, m properly dominates a sequence of move nodes $m_0, \ldots, m_n$ with each one associated to some subscripted version of f^+. By our induction hypothesis, each move node is associated to the same licensor feature f_k^+ in t_1 and t_2. But then there is a unique choice for the smallest natural number distinct from each one of these k, wherefore m is associated to the same licensor feature in t_1 and t_2. Consequently, $t_1 = t_2$ after all. □

The lemma shows that the non-determinism of τ is merely due to the restricted locality domain of linear transducers, which forces τ to make educated guesses about the shape of the derivation tree. A transducer with more elaborate lookup mechanism can compute the same transduction in a deterministic fashion. In particular, τ computes a deterministic, regularity preserving, first-order definable tree-to-tree transduction. In the remainder of this paper we take τ to directly refer to the transduction rather than any specific implementation thereof. We also write $\tau(t)$ for the image of t under τ.

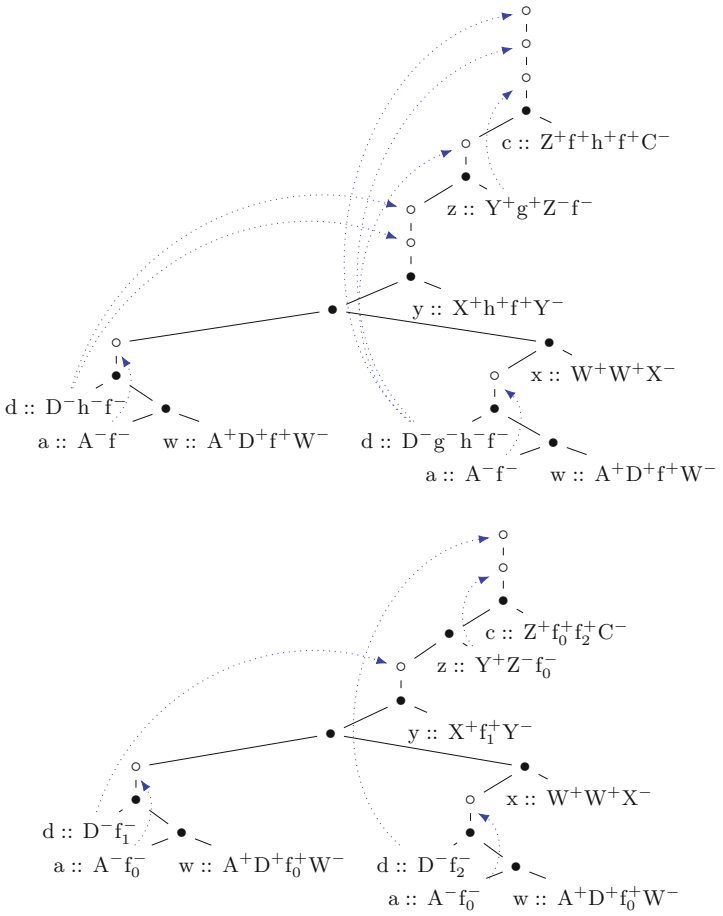


Fig. 2. Derivation tree (top) and its image under τ (bottom)

2.2 Proof of Strong Equivalence

We now show that the conversion carried out by τ does indeed result in an MG that generates the same set of phrase structure trees. To this end, we first establish an important restriction on movement. For every MG G, $\mu := |\{f \mid \langle f, move\rangle \times \textsc{Polarity} \in \mathit{Feat}\}|$. That is to say, μ indicates the number of distinct movement features. Given a derivation tree t of G and node n in t, the *traffic of* n, $\mathrm{traf}(n)$, is a measure of the number of LIs l in t that cross n during the derivation. More precisely, $\mathrm{traf}(n)$ denotes the number of LIs that are properly dominated by n in t and that have at least one occurrence that reflexively dominates n.

Lemma 2. *For every MG G, derivation tree t of G, and node n in t,* $0 \leq \mathrm{traf}(n) \leq \mu$.

Proof. The lower bound is trivial, while the upper bound follows from **SMC** and the definition of occurrence: there can be no two distinct LIs l and l' that are properly dominated by n and whose respective lowest occurrences reflexively dominating n are also associated to instances of the same licensor feature. Since there are only μ distinct licensor features, $\text{traf}(n)$ cannot exceed μ. □

Lemma 3. *It always holds for τ as defined above that $0 \leq j < \mu$.*

Proof. Consider an arbitrary Move node m. Since $\text{traf}(m) \leq \mu$, there cannot be more than μ LIs that are properly dominated by m and whose final (and only) occurrence reflexively dominates m. In order to distinguish the final occurrences of these LIs, then, one needs at most μ distinct indices. □

Lemma 3 guarantees both that τ is indeed a linear tree transduction and that the set of LIs occurring in the image of an MDTL under τ is still finite. As each one of these LIs has a well-formed feature string of the form $\gamma c \delta$, the set of LIs in the derivations produced by τ is an MG. It still remains to be shown, though, that the actual derivation trees produced by τ are well-formed.

Lemma 4. *It holds for every MDTL L and derivation tree $t \in L$ that $\tau(t)$ is a well-formed derivation tree.*

Proof. We have to show that $\tau(t)$ is correctly projected and satisfies **Merge**, **Final**, **Move** and **SMC**. The former holds because Move nodes are deleted iff their licensor features are removed. Furthermore, τ does not remove any Merge nodes or change any category or selector features, so **Merge** and **Final** hold because the domain of τ is an MDTL. Hence we only have to worry about **Move** and **SMC**.

Move. Move node m is an occurrence of some LI l in $\tau(t)$ (and thus its final occurrence) iff m is the lowest node in $\tau(t)$ that properly dominates l and is associated to a licensor feature matching l's single licensee feature. It is easy to see that τ furnishes at least one such m for every LI with a licensee feature.

SMC. We give an indirect proof that every Move node m is an occurrence for at most one LI in derivation tree t. This implies that m is an occurrence for exactly one LI thanks to **Move** and the fact that the images under τ contain as many Move nodes as LIs with licensee features.

Suppose that m in $\tau(t)$ is an occurrence for two distinct LIs l and l'. Then both l and l' have the same licensee feature f_i^-, for some arbitrary choice of f and i. Hence it must hold in t that

1. $\tau^{-1}(m)$ properly dominates $\tau^{-1}(l)$ and $\tau^{-1}(l')$, and
2. $\tau^{-1}(m)$ is a final occurrence o for either $\tau^{-1}(l)$ or $\tau^{-1}(l')$, and
3. $\tau^{-1}(m)$ is properly dominated by the final occurrence o' of the other.

Here $\tau^{-1}(n)$ denotes the node in t that corresponds to n. Now assume w.l.o.g. that o and o' are the final occurrences of $\tau^{-1}(l)$ and $\tau^{-1}(l')$, respectively. Then the path from $\tau^{-1}(l')$ to o' includes o, whence $\tau(o)$ and $\tau(o')$ must be associated to licensor features with distinct indices given the definition of τ. But then l and l' do not have the same licensee feature, either. Contradiction. □

We now know that τ produces a non-empty set of well-formed Minimalist derivation trees. This still does not imply, however, that the image of an MDTL under τ is itself an MDTL. The complicating factor is that MDTLs must be *maximal* sets of well-formed derivation trees—the LIs created by τ may allow for completely new derivations that are not part of the set produced by τ.

As a concrete example, consider the MG with the following lexicon:

$$\mathrm{u} :: \mathrm{B}^{+}\mathrm{C}^{-} \quad \mathrm{v} :: \mathrm{A}^{-}\mathrm{f}^{-}\mathrm{g}^{-} \quad \mathrm{w} :: \mathrm{B}^{+}\mathrm{g}^{+}\mathrm{B}^{-} \quad \mathrm{x} :: \mathrm{B}^{+}\mathrm{f}^{+}\mathrm{B}^{-} \quad \mathrm{y} :: \mathrm{A}^{+}\mathrm{B}^{-}$$

This grammar allows for only one derivation, depicted in Fig. 3 (left), which yields the string *uvwxy*. The image of this derivation tree under τ (see Fig. 3 right) contains almost exactly the same LIs. The only difference is the absence of f^{+} on x and f^{-} on v.

$$\mathrm{u} :: \mathrm{B}^{+}\mathrm{C}^{-} \quad \mathrm{v} :: \mathrm{A}^{-}\mathrm{g}^{-} \quad \mathrm{w} :: \mathrm{B}^{+}\mathrm{g}^{+}\mathrm{B}^{-} \quad \mathrm{x} :: \mathrm{B}^{+}\mathrm{B}^{-} \quad \mathrm{y} :: \mathrm{A}^{+}\mathrm{B}^{-}$$

Minor as this change may be, it has a major effect in that the MDTL is no longer singleton but actually infinite. Without the regulating effect of the licensee features of v, the LI x can be merged an arbitrary number of times. As a result, the generated string language is now $uvwx^{*}y$ instead of *uvwxy*.

Fortunately this issue is easy to avoid. It is always the case that the range R of τ is a subset of the unique MDTL over the set of LIs that occur in the derivation trees of R. Furthermore, R is guaranteed to be regular because MDTLs are regular and τ preserves regularity. Finally, the intersection of an MDTL and a regular tree language is the MDTL of some MG, *modulo* relabeling of selector and category features [4,7,13]. So to ensure that τ yields an MDTL, we only have to intersect its range with the MDTL over the set of all LIs that occur in at least one derivation tree produced by τ. A full algorithm for the intersection step is given in Sect. 3.2.3 of [7]. The algorithm operates directly on the MG lexicon, which allows it to be efficiently combined with the transducer in the appendix. The result is an MG G_{smnf} in SMNF.

It only remains for us to prove that G_{smnf} generates exactly the same derived trees as the original MG G, ignoring intermediate landing sites. To this end we first establish the weaker result that τ preserves the derived trees (again ignoring intermediate landing sites). Let ϕ be the standard mapping from derivation trees

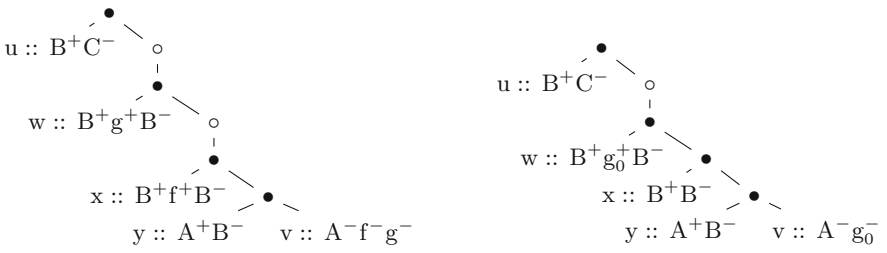


Fig. 3. Only possible derivation of example MG (left) and its image under τ (right)

to phrase structure trees and h a function that removes all intermediate landing sites from the phrase structure trees of an MG.

Theorem 1. *For every MG G with MDTL L, $h(\phi(L)) = \phi(\tau(L))$.*

Proof. Pick some arbitrary $t \in L$. Suppose $h(\phi(t)) \neq \phi(\tau(t))$. There must be some LI l with final occurrence o in t such that $\tau(o)$ is not the final occurrence of $\tau(l)$. But then either $\tau(l)$ has no final occurrence, the same final occurrence as some other LI, or the final occurrences of some LIs, including $\tau(l)$, have been switched. The former two are impossible as τ only generates well-formed derivation trees (Lemma 4). The latter is precluded by τ's choice of unique indices. Hence $h(\phi(t)) = \phi(\tau(t))$ after all and we have $h(\phi(L)) \subseteq \phi(\tau(L))$. The same arguments can be used in the other direction to show that for every $t \in \tau(L)$ and $t' \in \tau^{-1}(t)$ we have $h(\phi(t')) = \phi(t)$. This establishes $h(\phi(L)) \supseteq \phi(\tau(L))$, concluding the proof. □

Corollary 1. *For every MG G there is a strongly equivalent MG G_{smnf} in SMNF.*

Proof. The only addition to the previous proof is that G_{smnf} is obtained from the image of G's MDTL via the category refinement algorithm of [7]. Therefore G and G' may use different category and selector features. But since these features are all removed by ϕ, they are immaterial for the generated set of phrase structure trees. □

3 Evaluation

We now know that every MG has a strongly equivalent counterpart in SMNF. The relevance of such a normal form theorem, however, depends on the properties of the normal form and its overall usefulness. SMNF simplifies movement in MGs while preserving strong generative capacity, but does so at the expense of a larger number of movement features. As we discuss next in Sect. 3.1, the actual blow-up in the size of the lexicon is hard to predict because it depends on how strictly movers are tied to specific positions. The blow-up is still guaranteed to be at most linear, though. Sections 3.2 and 3.3 subsequently discuss applications and linguistic implications of SMNF.

3.1 Effects on Succinctness and Grammar Size

The conversion to SMNF has two sources of lexical blow-up. The first one is τ, which may replace a single LI of the form $\mathrm{a} :: \cdots \mathrm{h}^{+} \cdots \mathrm{A}^{-} \mathrm{g}^{-} \cdots \mathrm{f}^{-}$ by multiple variants $\mathrm{a} :: \cdots \mathrm{h}_{\mathrm{i}}^{+} \cdots \mathrm{A}^{-} \mathrm{f}_{\mathrm{j}}^{-}$ that only differ in the value of i and j ($0 \leq i, j < \mu$) and which licensor features have been removed. Given an MG G, the lexicon size of its SMNF counterpart is thus linearly bounded by $\sum_{l \in Lex} \mu^{\gamma(l)+\delta(l)}$, where $\gamma(l)$ is the number of licensor features of l, and $\delta(l)$ is 1 if l contains a licensee feature and 0 otherwise. The second blow-up is due to the regular intersection step that

turns the range of τ into an MDTL. This step is known to be linear in the size of the original lexicon and polynomial in the size of the smallest (non-deterministic) automaton that recognizes the regular tree language [7]. Crucially, though, the size increase induced by τ depends greatly on the shape of the grammar.

In the optimal case, τ does not cause any lexical blow-up and thus defines a bijection between lexical items.[1] Intuitively, this is the case whenever the position of an f-mover is fixed with respect to other f-movers, such as in the MG for $a^n b^n d^n$ below that uses massive remnant movement:

$$\begin{array}{lll} \varepsilon :: \mathrm{A}^-\mathrm{a}^-\mathrm{f}^- & \mathrm{a} :: \mathrm{D}^+\mathrm{a}^+\mathrm{f}^+\mathrm{A}^-\mathrm{a}^-\mathrm{f}^- & \varepsilon :: \mathrm{D}^+\mathrm{a}^+\mathrm{f}^+\mathrm{A}'^- \\ \varepsilon :: \mathrm{B}^-\mathrm{b}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{A}^+\mathrm{b}^+\mathrm{f}^+\mathrm{B}^-\mathrm{b}^-\mathrm{f}^- & \varepsilon :: \mathrm{A}'^+\mathrm{b}^+\mathrm{f}^+\mathrm{B}'^- \\ \varepsilon :: \mathrm{D}^-\mathrm{d}^-\mathrm{f}^- & \mathrm{d} :: \mathrm{B}^+\mathrm{d}^+\mathrm{f}^+\mathrm{D}^-\mathrm{d}^-\mathrm{f}^- & \varepsilon :: \mathrm{B}'^+\mathrm{d}^+\mathrm{f}^+\mathrm{C}^- \end{array}$$

After applying τ, this yields a notational variant of the standard MG for this language. Crucially, both grammars have exactly the same number of LIs.

$$\begin{array}{lll} \varepsilon :: \mathrm{A}^-\mathrm{f}_0^- & \mathrm{a} :: \mathrm{D}^+\mathrm{f}_0^+\mathrm{A}^-\mathrm{f}_0^- & \varepsilon :: \mathrm{D}^+\mathrm{f}_0^+\mathrm{A}'^- \\ \varepsilon :: \mathrm{B}^-\mathrm{f}_1^- & \mathrm{b} :: \mathrm{A}^+\mathrm{f}_1^+\mathrm{B}^-\mathrm{f}_1^- & \varepsilon :: \mathrm{A}'^+\mathrm{f}_1^+\mathrm{B}'^- \\ \varepsilon :: \mathrm{D}^-\mathrm{f}_2^- & \mathrm{d} :: \mathrm{B}^+\mathrm{f}_2^+\mathrm{D}^-\mathrm{f}_2^- & \varepsilon :: \mathrm{B}'^+\mathrm{f}_2^+\mathrm{C}^- \end{array}$$

Without this restriction to fixed relative positions, blow-up can occur even if the grammar does not allow for remnant movement and only generates a finite language. In the following grammar, the three lexical items a, b, and d can be selected by a in arbitrary order, and their target sites are similarly free in their relative ordering.

$$\begin{array}{lll} \varepsilon :: \mathrm{T}^+\mathrm{C}^- & \mathrm{a} :: \mathrm{M}^-\mathrm{a}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{a}^+\mathrm{f}^+\mathrm{C}^- \\ \varepsilon :: \mathrm{M}^+\mathrm{M}^+\mathrm{M}^+\mathrm{T}^- & \mathrm{b} :: \mathrm{M}^-\mathrm{b}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{b}^+\mathrm{f}^+\mathrm{C}^- \\ & \mathrm{d} :: \mathrm{M}^-\mathrm{d}^-\mathrm{f}^- & \varepsilon :: \mathrm{C}^+\mathrm{d}^+\mathrm{f}^+\mathrm{C}^- \end{array}$$

Conversion into SMNF increases the size from 8 to 20 since each instance of f^- and f^+ must be replaced by f_i^- and f_i^+, $0 \leq i \leq 2$. The size increase of SMNF thus correlates with the number of combinatorial options furnished by the interaction of Merge and Move. Future work will hopefully be able to characterize this correlation in greater detail.

[1] Strictly speaking the optimal case is for τ to reduce the size of the lexicon. But as far as we can tell this only happens with needlessly redundant MGs such as the one below, where the SMNF lexicon contains only 5 instead of 7 entries.

$$\begin{array}{lll} \mathrm{c} :: \mathrm{M}^+\mathrm{C}^- & \mathrm{m} :: \mathrm{M}^-\mathrm{g}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{C}^+\mathrm{g}^+\mathrm{B}^- \\ \mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{M}^+\mathrm{C}^- & \mathrm{m} :: \mathrm{M}^-\mathrm{h}^-\mathrm{f}^- & \mathrm{b} :: \mathrm{C}^+\mathrm{h}^+\mathrm{B}^- \\ \mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{C}^- & & \end{array}$$

A minor change to the grammar immediately undoes the size benefits of SMNF. All it takes is to replace $\mathrm{c} :: \mathrm{B}^+\mathrm{f}^+\mathrm{M}^+\mathrm{C}^-$ by $\mathrm{c} :: \mathrm{B}^+\mathrm{M}^+\mathrm{f}^+\mathrm{C}^-$. The SMNF lexicon then has 8 entries instead of 7 (1 for b, 2 for m and 5 for c).

3.2 Usefulness and Applications

A normal form can serve a multitude of purposes. Our main motivation for SMNF is to simplify proofs and rigorous analysis. The availability of intermediate movement in MGs creates special cases that are hardly ever insightful. For example, the top-down parser in [19] includes two movement rules, one for final and one for intermediate movement, yet the latter does nothing of interest except eliminate a pair of licensor and licensee features. Similarly, the proof in [12] that MGs cannot generate mildly context-sensitive string languages without remnant movement has to cover intermediate movement as a special case that does not add anything of value. Quite generally, intermediate movement is hardly ever relevant but frequently introduces complications that distract from the core ideas of proofs and theorems.

In fact, SMNF has already proven indispensable in ongoing projects. In the wake of our theorem, Graf and Heinz [8] show that SMNF reduces the complexity of MDTLs and renders them very similar to dependencies found in phonology. It has been known for quite a while that MDTLs are subregular [5], and it has been conjectured that segmental phonology is tier-based strictly local [10]. These two insights are combined by [8]: an MDTL is a tier-based strictly local tree language iff its grammar is in SMNF. This suggests that syntax and phonology are very much alike at a sufficient level of formal abstraction.

With other formalisms, normal forms have also been useful for parsing and the construction of automata. Chomsky Normal Form, for instance, is indispensable to guarantee cubic time complexity for CKY parsing of context-free grammars. Greibach Normal Form, on the other hand, simplifies the construction of equivalent, real-time pushdown automata for these grammars. At this point it remains an open question whether SMNF offers comparable advantages in these areas. On the one hand SMNF simplifies movement dependencies, on the other hand any tangible parsing benefits may be so minor that they are vastly outweighed by the lexical blow-up of the SMNF conversion. One conceivable scenario is that SMNF offers a parsing advantage only for those grammars where lexical blow-up is minimal due to movement being more restricted. It would be interesting to see whether this subclass is sufficiently powerful from a linguistic perspective. If so, it might indicate that natural languages restrict how movement dependencies interact in order to aid parsing.

3.3 Linguistic Implications

The unexpected parallels between phonology and MGs in SMNF, as well as our speculations above regarding parsing optimization, show that the existence of SMNF is not of purely technical interest but also raises deep linguistic questions. Yet there are certain linguistic concerns one might raise regarding SMNF.

In the syntactic literature, direct movement to the final landing site without intermediate movement steps is called *one fell swoop* movement. This kind of movement has been argued against on conceptual and empirical grounds (cf. Chap. 1 of [1]). However, the arguments against one fell swoop movement do

not carry over to SMNF. The reason for this is that the linguistic arguments are about derived trees, whereas SMNF is a property of derivation trees. In principle, nothing prevents us from modifying the mapping from derivation trees to derived trees so that landing sites are inserted where syntacticians want them to occur, e.g. at phase edges. Something along these lines was already proposed in [11, Sect. 2.1.1] for successive cyclic wh-movement. The strategy can easily be extended to other intermediate movement steps because most of them are readily predictable from the rest of the structure.

To take but one example, consider the movement steps of a subject wh-phrase as in *Who does John think kissed Mary?*, to which Minimalists would ascribe the structure $[_{\text{CP}}$ who C does John think $[_{\text{CP}}\ t$ C $[_{\text{TP}}\ t$ T t kissed Mary]]]. The subject starts out in Spec,VP of the embedded clause, moves to the embedded subject position in Spec,TP, then to the embedded Spec,CP, and from there finally to matrix Spec,CP. But all these intermediate movement steps are readily predictable from the fact that *who* moves to matrix Spec,CP. A phrase that starts out in Spec,VP and moves to a position above Spec,TP must land there by virtue of being a subject. A phrase that moves to some higher Spec,CP must land in every CP-specifier that occurs between the two. There is no requirement for intermediate movement steps to be linked to feature checking in the derivation because they can be inferred indirectly.

Cases where this indirect inference of intermediate movement steps is impossible are hard to come by. They mostly involve configurations where movement serves the sole purpose of modifying word order, such as scrambling or linearization of siblings to account for the head-initial/head-final contrast between languages. But it is far from evident that intermediate movement matters for scrambling, and the headedness parameter can be captured more directly by encoding the linearization of arguments in the selector features of MGs [cf. 6,18]. Given our current linguistic understanding, then, there is no sound argument or conclusive evidence against SMNF, a point that was already made over 25 years ago in [16, Sect. 3.4.1] (we are indebted to an anonymous reviewer for bringing this to our attention). The only major issue is the lexical blow-up, but this is as much a detriment as an opportunity: a hierarchy that ranks movement dependencies with respect to the SMNF blow-up they induce might furnish novel generalizations with rich typological implications.

Conclusion

Every MG can be efficiently transformed into a strongly equivalent MG in SMNF such that every LI moves at most once. The translation procedure in this paper is specified as a linear transduction over MDTLs, but is easily extended to a mapping between Minimalist lexicons: given an MG lexicon, one can immediately construct a deterministic bottom-up tree automaton that recognizes its MDTL [14], from which one obtains an automaton for the corresponding SMNF tree language via the usual transducer composition algorithm [2]. The nullary symbols of the automaton constitute the new lexicon. A Python implementation of this extended translation is hosted at https://github.com/CompLab-StonyBrook.

In future work, we hope to generalize SMNF from standard MGs to movement-generalized MGs [6]. We also intend to further explore how movement can be restricted to avoid lexical blow-up and whether these restrictions are linguistically feasible. It will also be interesting to see if some of these findings are applicable in the reverse direction to obtain algorithms that minimize (movement-generalized) MGs by adding intermediate movement steps.

A Specification of SMNF Transducer

A *bottom-up tree transducer* is a 5-tuple $\tau := \langle \Sigma, \Omega, Q, F, \Delta \rangle$, where Σ and Ω are ranked alphabets, Q is a finite set of states, $F \subseteq Q$ is the set of final states, and Δ is a finite set of *transduction rules*. Each transduction rule is of the form $f(q_1(x_1), \ldots, q_n(x_n)) \to q(t)$ such that f is an n-ary symbol in Σ ($n \geq 0$), $q, q_1, \ldots, q_n \in Q$, and t is a tree with node labels drawn from Ω and the nullary symbols $x_1, \ldots, x_n$. The transducer is *linear* iff each x_i may occur at most once in t. It is *non-deleting* iff each x_i occurs at least once in t. It is *non-deterministic* iff at least two transduction rules have the same left-hand side.

A $(\Sigma_1, \ldots, \Sigma_n)$-tree is a tree whose nodes are labeled with symbols from $\bigcup_{1 \leq i \leq n} \Sigma_i$. Given (Σ, Q)-tree u and (Ω, Q)-tree v, τ immediately derives v from u ($u \Rightarrow_\tau v$) iff there is a transduction rule such that u has the shape $f(q_1(u_1), \ldots, q_n(u_n))$—where each u_i is the subtree of u immediately dominated by q_i—and v is the result of substituting u_i for x_i in t. We use $\Rightarrow_\tau^+$ to denote the transitive closure of $\Rightarrow_\tau$. The transduction computed by τ is the set $\tau := \{\langle u, v \rangle \mid u \Rightarrow_\tau^+ q_f(v), u \text{ a } \Sigma\text{-tree, and } q_f \in F\}$. We furthermore let $\tau(s) := \{\langle s, t \rangle \in \tau\}$, and $\tau(L) := \bigcup_{s \in L} \tau(s)$ for L a tree language.

We now define a non-deterministic linear bottom-up tree transducer that brings Minimalist derivation trees into SMNF. The transducer is almost non-deleting as it only deletes intermediate Move nodes. Consequently, it can be regarded as the composition of a non-deterministic relabeling and a deterministic transducer that deletes Move nodes marked for removal. Before moving on, we introduce an additional piece of MG notation. In a standard MG, every useful LI must be of the form $\gamma c \delta$, where γ is a string of licensor and selector feature, c is a category feature, and δ is a string of 0 or more licensee features. Given a feature component s, $m(s)$ is obtained from s by removing all Merge features. We overload m such that for every LI $l := \sigma :: \text{s}$, $m(l) := m(s)$.

The SMNF transducer has to handle three tasks in parallel: (I) detect and delete intermediate Move nodes, (II) modify the feature components of LIs, and (III) ensure that each licensee feature is subscripted with the smallest possible natural number. Consequently, each state has a tripartite structure

$$\left\langle \begin{matrix} u_1, \ldots, u_n \\ m_1, \ldots, m_n \\ I_1, \ldots, I_n \end{matrix} \right\rangle$$

such that $n \leq \mu$ (the upper bound on the grammar's traffic), u_i keeps track of the unchecked Move features of some LI l, m_i records how $m(l)$ was modified, and I_i stores which required indices have not been encountered yet. More precisely:

for each u_i there is some LI l with u_i a suffix of $m(l)$; m_i is a string of indexed Move features and the distinguished symbol $\square$ such that removal of indices and $\square$ yields a subsequence of $m(l)$ including the final licensee feature; and I_i is some subset of the closed interval $[0, \mu-1]$ of natural numbers. Among all these states, the only final state is the empty state $\langle\rangle$.

While the transducer has a large number of rules, they can easily be compressed into a few templates using algebraic operations. First, we define a nondeterministic relabeling ℓ operating on MG feature strings that preserves all Merge features and either deletes Move features or relabels them:

$$\ell(f_1 \cdots f_n) := \begin{cases} f_1 \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a Merge feature} \\ f_{1,i} \ell(f_2 \cdots f_n) \text{ or } \square \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a licensor feature, } 0 \leq i < \mu \\ f_{1,i} \ell(f_2 \cdots f_n) & \text{if } f_1 \text{ is a licensee feature, } 0 \leq i < \mu \\ \varepsilon & \text{if } f_1 \cdots f_n = \varepsilon \end{cases}$$

We extend ℓ to LIs: if $l := \sigma :: \gamma \mathrm{cf_1} \cdots \mathrm{f_n}$, then $\ell(l)$ is $\sigma :: \ell(\gamma \mathrm{c})$ if $n = 0$ and $\sigma :: \ell(\gamma \mathrm{cf_n})$ otherwise. In addition, h is a homomorphism that replaces the distinguished symbol $\square$ by ε in every string. The transduction rules for leaf nodes now follow a simple template:

$$\textbf{LIs}. \qquad \mathrm{l} :: \mathrm{s} \rightarrow \begin{cases} \left\langle \begin{matrix} m(l) \\ m(l') \\ \varepsilon \end{matrix} \right\rangle (h(l')) & \text{for } l' = \ell(l) \text{ if } l' \text{ does not end in a licensee feature} \\ \left\langle \begin{matrix} m(l) \\ m(l') \\ [0, k-1] \end{matrix} \right\rangle (h(l')) & \text{for } l' = \ell(l) \text{ if the licensee feature of } l' \text{ is subscripted with } k \end{cases}$$

For Merge we use a binary operator $\otimes$ that combines all the components of the states.

$$\left\langle \begin{matrix} u_1, \ldots, u_j \\ m_1, \ldots, m_j \\ I_1, \ldots, I_j \end{matrix} \right\rangle \otimes \left\langle \begin{matrix} u_{j+1}, \ldots, u_k \\ m_{j+1}, \ldots, m_k \\ I_{j+1}, \ldots, I_k \end{matrix} \right\rangle := \left\langle \begin{matrix} u_1, \ldots, u_j, u_{j+1}, \ldots, u_k \\ m_1, \ldots, m_j, m_{j+1}, \ldots, m_k \\ I_1, \ldots, I_j, I_{j+1}, \ldots, I_k \end{matrix} \right\rangle$$

$$\textbf{Merge}. \qquad \bullet(q(x), q'(y)) \Rightarrow q \otimes q'(\bullet(x, y))$$

The Move rules have to handle most of the work in the transducer. First, they have to delete movement features in the top component and use this information to decide whether the Move node is final or intermediate. Licensor features in the second component must also be removed, and the same goes for licensee features if the Move node is final. In the latter case, the index of the checked licensee feature is removed from all other index sets. Checking of a licensee feature, in turn, is only possible if its index set is empty.

As before, we simplify our presentation by using an algebraic operator $\ominus$, which takes care of updating index sets. Given a state q with index set I_j at position j, $I_j \ominus_f k = I_j - \{k\}$ if m_j ends in some subscripted version of f^-. In all other cases, $I_j \ominus_f k = I_j$. The transition rules for intermediate and final movement are now captured by four distinct cases. We only give two here, the other two are their mirror image with the order of $\mathrm{f}^+\delta_j$ and $\mathrm{f}^-\delta_k$ switched.

$$\textbf{Move}. \quad \circ \left(\left\langle \begin{matrix} u_1, \ldots, \mathrm{f}^+\delta_j, \ldots, \mathrm{f}^-\delta_k, \ldots, u_n \\ m_1, \ldots, m_j, \ldots, m_k, \ldots, m_n \\ I_1, \ldots, I_j, \ldots, I_k, \ldots, I_n \end{matrix} \right\rangle (x) \right)$$

$$:= \begin{cases} \left\langle \begin{matrix} u_1, \ldots, \delta_j, \ldots, \delta_k, \ldots, u_n \\ m_1, \ldots, m'_j, \ldots, m_k, \ldots, m_n \\ I_1, \ldots, I_j, \ldots, I_k, \ldots, I_n \end{matrix} \right\rangle (x) & \text{if } \delta_k \neq \varepsilon \text{ and } m_j = \square m'_j \\ \left\langle \begin{matrix} u_1, \ldots, \delta_j, \ldots, \ldots, u_n \\ m_1, \ldots, m'_j, \ldots, \ldots, m_n \\ I_1 \ominus_f i, \ldots, I_j, \ldots, \ldots, I_n \ominus_f i \end{matrix} \right\rangle (\circ(x)) & \text{if } \delta_k = \varepsilon,\ m_j = \mathrm{f}_\mathrm{i}^+ m'_j, \\ & m_k = \mathrm{f}_\mathrm{i}^-,\ I_k = \emptyset, \text{ and} \\ & \text{only } m_k \text{ starts with } \mathrm{f}_\mathrm{i}^- \end{cases}$$

Note that since the transducer is restricted to well-formed derivation trees, at most one component of a state can contain licensor features. Similarly, the SMC prevents any two u_i from starting with the same licensee feature, so the indices j and k in the template are always uniquely identified.

A few clarifying remarks may be in order. First, note that the transducer always halts if it finds a case of intermediate movement but did not delete the corresponding licensor feature earlier on. This is enforced by m_j starting with $\square$. Second, the index set I_j is not updated for final movement. That is because the corresponding LI has not started to move yet, so its index set is not active yet. If I_j were updated, then an LI that licenses f_2 movement would be allowed to undergo f_3 movement even if f_2 movement is possible, too.

To sum up, given an MG with lexicon Lex, the SMNF transducer τ has input alphabet $\Sigma := Lex^{(0)} \cup \{\circ^{(1)}, \bullet^{(2)}\}$ and output alphabet $\Omega := \bigcup_{l \in Lex} h(\ell(l))^{(0)} \cup \{\circ^{(1)}, \bullet^{(2)}\}$. Its state set Q consists of all possible tripartite tuples as defined at the beginning of this section. While this set is large, it is guaranteed to be finite. The empty state $\langle\rangle$ is the only final state, and the set Δ of transduction rules contains all possible instantiations of the templates above given Q.

References

1. Abels, K.: Successive cyclicity, anti-locality, and adposition stranding. Ph.D. thesis, University of Conneticut (2003)
2. Baker, B.S.: Composition of top-down and bottom-up tree transductions. Inf. Control **41**, 186–213 (1979)
3. Engelfriet, J.: Bottom-up and top-down tree transformations – a comparison. Math. Syst. Theor. **9**, 198–231 (1975)

4. Graf, T.: Closure properties of minimalist derivation tree languages. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 96–111. Springer, Heidelberg (2011)
5. Graf, T.: Locality and the complexity of minimalist derivation tree languages. In: Groote, P., Nederhof, M.-J. (eds.) Formal Grammar 2010/2011. LNCS, vol. 7395, pp. 208–227. Springer, Heidelberg (2012)
6. Graf, T.: Movement-generalized minimalist grammars. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 7351, pp. 58–73. Springer, Heidelberg (2012)
7. Graf, T.: Local and transderivational constraints in syntax and semantics. Ph.D. thesis, UCLA (2013)
8. Graf, T., Heinz, J.: Commonality in disparity: the computational view of syntax and phonology. In: Slides of a talk given at GLOW 2015, 18 April, Paris, France (2015)
9. Harkema, H.: A characterization of minimalist languages. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, pp. 193–211. Springer, Heidelberg (2001)
10. Heinz, J., Rawal, C., Tanner, H.G.: Tier-based strictly local constraints in phonology. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics, pp. 58–64 (2011)
11. Kobele, G.M.: Generating copies: an investigation into structural identity in language and grammar. Ph.D. thesis, UCLA (2006)
12. Kobele, G.M.: Without remnant movement, MGs are context-free. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10/11. LNCS, vol. 6149, pp. 160–173. Springer, Heidelberg (2010)
13. Kobele, G.M.: Minimalist tree languages are closed under intersection with recognizable tree languages. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 129–144. Springer, Heidelberg (2011)
14. Kobele, G.M., Retoré, C., Salvati, S.: An automata-theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds.) Model Theoretic Syntax at 10, pp. 71–80 (2007)
15. Michaelis, J.: Transforming linear context-free rewriting systems into minimalist grammars. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, pp. 228–244. Springer, Heidelberg (2001)
16. Ristad, E.S.: Computational structure of human languages. Ph.D. thesis, MIT (1990)
17. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
18. Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–643. Oxford University Press, Oxford (2011)
19. Stabler, E.P.: Bayesian, minimalist, incremental syntactic analysis. Top. Cogn. Sci. **5**, 611–633 (2013)

Word Ordering as a Graph Rewriting Process

Sylvain Kahane[1(✉)] and François Lareau[2]

[1] Modyco, Université Paris Ouest Nanterre, Nanterre, France
sylvain@kahane.fr
[2] OLST, Université de Montréal, Montreal, Canada
francois.lareau@umontreal.ca

Abstract. This paper shows how the correspondence between a unordered dependency tree and a sentence that expresses it can be achieved by transforming the tree into a string where each linear precedence link corresponds to one specific syntactic relation. We propose a formal grammar with a distributed architecture that can be used for both synthesis and analysis. We argue for the introduction of a topological tree as an intermediate step between dependency syntax and word order.

Keywords: Dependency grammar · Linearization · Syntax · Polarized unification grammar

1 Introduction

Word ordering has been addressed in many formal frameworks, from Categorial Grammar (CG) and Context Free Grammar (CFG), to Tree-Adjoining Grammar (TAG), Lexical Functional Grammar (LFG), Head-driven Phrase Structure Grammar (HPSG), Minimalist Program (MP), etc. The early formalisms did not separate linearization from sub-categorization. The first formalism to clearly separate them was Generalized Phrase Structure Grammar (GPSG) [1, 2]. In grammars that separate linearization rules from the others, the former are generally expressed in a different formalism than the latter, and the two kinds cannot combine freely with one another. Linearization rules must be precompiled with sub-categorization rules, as in metagrammars for TAG [3], or they form a separate module that is applied as a whole before or after other modules. In LFG, the linearization rules are CFG rules endowed with functional features describing the correspondence between c- and f-structures [4], which cannot be used in other modules of the model [5]. In a constraint-based formalism like HPSG, linear order is constrained by features, but its computation is left aside. It is a list of linear precedence statements from which linear order must be deduced by external mechanisms [6–8]. More generally, in phrase structure grammar, word order is expressed on constituents but not words: only sister constituents are ordered, and order between words must be deduced by extra devices.

Our first aim in this paper is to propose a general formalism that allows to write both linearization and sub-categorization rules and to combine them in whatever order, with no implicit procedure. This will allow us to use the rules for analysis as well as synthesis,

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 216–239, 2016.
DOI: 10.1007/978-3-662-53042-9_13

to pre-compile sets of rules from different levels if needed, and to use incremental strategies for parsing or generation. It is important to underline that our approach is mathematical and not computational: our aim is to propose a rule-based formalism to write modular, declarative grammars, but we are not directly interested in procedural strategies.

Our second aim is to propose a formalism for dependency grammar (henceforth DG). In formal DG, the focus has been on valency, sub-categorization, dependency tree generation, and the semantics-syntax interface. One of the well-known advantages of DG is that it separates linear order from syntactic structure proper [9] (Chap. 6) and it elegantly captures non-projective syntactic constructions (equivalent to trees with discontinuous constituents in phrase-based formalisms). Nevertheless, few efforts have been made on the formalization of word ordering and non-projectivity in DG. Formal DGs handling non-projective ordering have been proposed [10–13], but they do not fit our first aim, which is to use the same formalism for linearization and sub-categorization rules. Non-projective dependency parsers have been proposed [14], but the grammar cannot be clearly separated from the parsing procedure. Meaning-Text Theory (MTT) [15] is a model we want to follow because it is very modular and all rules are expressed in similar terms, that is, in terms of correspondence between two levels of representation. But although linearization rules have been proposed within MTT [16, 17], a complete formalization has never been achieved, especially the treatment of non-projective ordering, which remains particularly informal. This paper can be viewed as an attempt to give a clean formalization of MTT's linearization rules.

Our third aim is to show that language modelling, and in particular word ordering, can be viewed as a graph rewriting process. We consider that modelling a natural language consists in associating every uttered string of words to its meaning(s), or, conversely, to associate every linguistic meaning to the string(s) of words that express it. Our approach to language modelling is strongly influenced by MTT, which posits that the core meaning of a sentence can be encoded by a graph representing the predicate-argument relations that exist between the meaning of the words (or morphemes) composing it [18]. In other words, language modelling mainly consists in transforming a semantic graph into a linear graph representing the chain of words (and vice versa). Moreover, except where there is no one-to-one correspondence between minimal semantic units and minimal expressive units, most linear precedence links between two consecutive words in the spoken chain are the image of particular predicate-argument relations. Hence, the correspondence between a semantic graph and a string of words mainly consists in moving the edges of the graph until they form a chain.

2 Natural Language Modelling and Graph Rewriting

Let us consider the following sentence:

(1) *Men from Swaziland often work in mines.*

The text of (1) is a string of words, which can be represented by a linear graph as in Fig. 1. This graph will be called *linear order* in this paper (following [9]). It is similar

to the morphological structure of MTT, an intermediate level of representation between the surface syntactic structure and the phonological representation. Edges of the linear order are called *linear precedence links* (LPLs) and are labelled with the symbol "<".

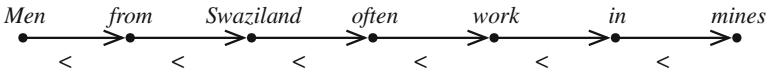


Fig. 1. Linear order of (1)

The core meaning of (1) can be represented as a *semantic graph* as in Fig. 2 [15, 18]. In a semantic graph, edges are called semantic dependencies and represent predicate-argument relations. The source of a semantic dependency is a predicate, and its target is an argument of that predicate. Edges are labelled with a number indicating the rank of the argument (1st argument, 2nd argument, etc.), in decreasing order of salience [19]. The semantic graph of (1) below expresses the fact that the signified of *work* is a unary predicate taking the signified of *men* as its first (and unique) argument. The signified of *often* is also a unary predicate: it takes the signified of *work* as its argument. The signifieds of both locative prepositions *from* and *in* are binary predicates taking the located entity or event as their first argument and the locus as their second argument. It is also possible to encode this graph with an equivalent logical formula, as below [20, 21].

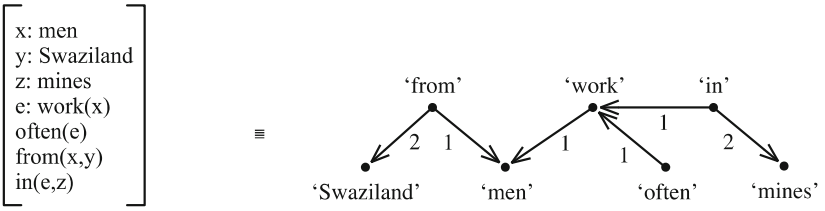


Fig. 2. Semantic representation of (1)

Figure 3 shows the correspondence between the LPLs and the semantic dependencies of (1). The correspondence is generally realized in two main stages: (1) a semantics-syntax interface ensuring the lexicalization and the hierarchization of the semantic graph, giving a syntactic dependency tree and (2) the linearization and the morphologization of the dependency tree, giving a string of words. This paper focuses on linearization. The semantics-syntax interface has been described in several papers [15, 22–26, 43] and will be sketched in Sect. 3.4. The dependency tree of (1) is presented in Fig. 4.

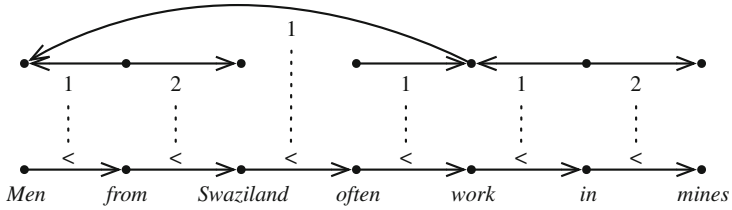


Fig. 3. Correspondence between linear precedence links and semantic dependencies

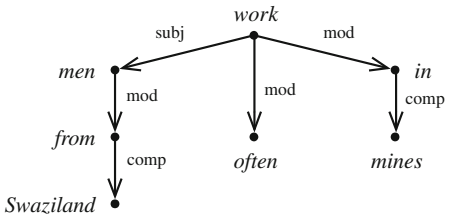


Fig. 4. Syntactic tree of (1)

A syntactic dependency tree for an n word sentence has exactly $n - 1$ edges, as does its linear order. We will show that the edges are in a one-to-one correspondence and that this correspondence can be described by a graph rewriting system that moves the edges of the dependency tree to transform it into a string. The rules will be written in a formalism called Polarized Unification Grammar (PUG), which was introduced by [27, 28] based on previous work by [29–32]. It has been used for the semantics-syntax interface since [23] but has never been really used for word ordering before.[1] There are great advantages to using the same formalism for different modules of the grammar. A common formalism allows us to use the same grammar both for synthesis (producing a string of words from a semantic representation) and analysis (extracting the meaning of a sentence). Many strategies are conceivable, including the possibility to pre-compile some groups of rules, the result of such pre-compilation being expressed in the common formalism.

3 Governor-Dependent Linearization Rules

3.1 Sketch of Governor-Dependent Linearization Rules

We will introduce our approach with a minimalistic example:

(2) *Mary loves Peter.*

[1] [27, 28] showed how to simulate in PUG LFG's phi-projection, which ensures the linearization process. The resulting grammar has not been studied in itself, although a similar grammar was presented in [23].

Figure 5 shows the dependency tree for this sentence, where *Mary* is the subject of the verb and *Peter* its object, and the corresponding linear order.

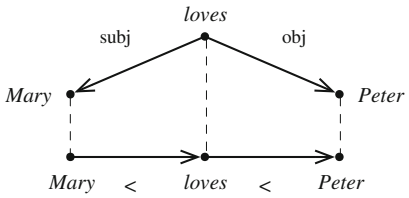


Fig. 5. Syntactic tree and linear order of (2)

To linearize the dependency tree of (2), we only need to know that in English the subject goes before the verb and the object can goes after.[2] This can be formalized by saying that a *subject* dependency corresponds to an LPL in the opposite direction, while the *object* dependency corresponds to an LPL in the same direction.[3] Figure 6 gives a first sketch of these rules, before introducing our formal framework.

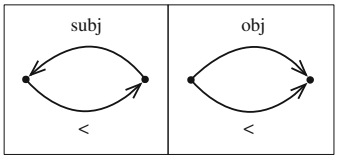


Fig. 6. Sketch of *subj* and *obj* governor-dependent linearization rules

Such linearization rules can be used in synthesis (syntactic dependencies become LPLs) as well as in analysis (LPLs become syntactic dependencies). For instance, the rules of Fig. 6 say that a LPL between two words can correspond to an *object* dependency in the same direction or a *subject* dependency in the opposite direction.[4]

Linearization rules, which realize the correspondence between syntactic dependencies and linear order, constitute the linearization module. The direction in which the

[2] Other rules can apply to an *object* dependency under particular conditions: for instance a wh-word in the object position can be placed before the verb, but even for a wh-word it is possible to place it after the verb (cf. so called *in situ* wh-question: *Mary loves who?*).

[3] Every rule must be read with an epistemic modality: things *can* happen as per the rule. It comes from the fact that alternative rules can *a priori* apply and things can happen differently.

[4] The fact that the subject relation has a finite verb as governor could be indicated in this rule, but this constraint is already verified by the semantics-syntax interface (see Sect. 3.4) and can also be verified by the syntactic well-formedness grammar (see [23]).

linearization rules are used depends on the input structure given to this module.[5] The dependency tree is at the articulation point between the linearization module and the semantics-syntax interface, while the linear order is at the articulation point between the linearization module and the phonological module. In other words, the linearization module will be called by one of these two other modules: the semantics-syntax interface in synthesis, which gives it a dependency structure as input, or the phonological module in analysis, which gives it a linear order as input.

To ensure that a module has processed the whole input structure, and for modules to call one another, we use a unique device: the polarization of objects, which we will now present.

3.2 Polarized Unification Grammar

Polarized Unification Grammar (henceforth, PUG) is a formalism inspired by TAG, where rules are modelled by elementary structures that combine to produce the final structure of an utterance. Unlike TAG though, PUG handles any kind of graph, and not only trees. PUG considers four kinds of entities: objects, functions, atomic values, and polarities. The *objects* handled by the linearization module are nodes representing words, and edges representing syntactic dependencies or LPLs.[6] Syntactic dependencies and LPLs are binary edges, which require two nodes as their source and target. Edges are bound to their source and target nodes through *structural functions*, which take as their argument an object and return as their value another object. Two other kinds of functions are considered in PUG, besides structural functions: *labelling functions*, that associate an object with an *atomic value*, and *polarizing functions*, that link an object to a *polarity*. Figure 7 shows a fragment of the dependency tree of (2) and its formalization when the structural and labelling functions are made explicit. Note that the visual representation of edges as arrows is merely a convenient way to indicate that this object has a source and a target. Despite this visual metaphor, edges really are objects just like nodes, as Fig. 7 makes explicit.

[5] We are presenting our grammar in a *transductive* perspective, where an input structure is given to the grammar and the corresponding structure is produced. As we will see in Sect. 3.6, there are two other ways of considering a correspondence grammar: the *equative* mode, where two structures are given to the grammar and the grammar verifies whether they correspond to each other; and the *generative* mode, where no structure is given to the grammar, which produces couples of structures corresponding to each other.

[6] Previous presentations of PUG distinguished the nodes at different levels of representation. It would be possible here too and even necessary if we wanted to take into account that in some cases, such as amalgams (*ain't* = *am* + *not*), some nodes of a given level can be merged at another level. Here, we consider that the nodes are the same at the different levels of representation and we focus on reordering the graph into a string.

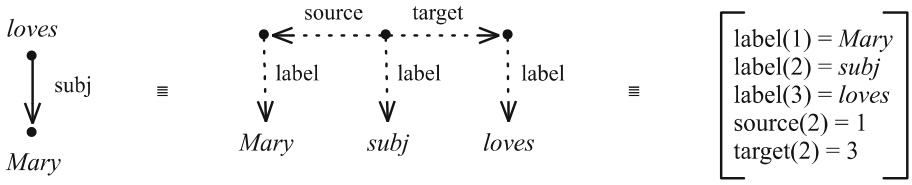


Fig. 7. Making objects and functions explicit

Polarities indicate whether an object must be consumed by a given module. We consider a lot of polarizing functions for our different modules, but we can work with only the same two values for any polarizing function: black and white. A black object, i.e., an object with a black polarity for a given function, represents a resource, while a white object represents a requirement. An object can receive several polarities through as many polarizing functions, generally associated with different modules of the model. When an object has been handled by a given module, it is generally black for the polarizing function pertaining to this module, but it will be white for the polarizing function pertaining to the next module we want to trigger. In other words, the black polarity is *saturated*, which means that a black object does no longer need to be handled, while the white polarity is non saturated, requiring to be saturated by a black polarity. A structure is said to be *saturated* for a given polarizing function if the value of this function is saturated for every object of the structure. At the end of the process, the derived structure must be saturated for every polarizing function considered.

An instance of PUG is a finite set of *elementary structures*, with a subset of initial structures. An elementary structure comprises a finite number of objects linked by structural functions (which define the structure proper) and associated to a finite number of labels and polarities (by labelling and polarizing functions). Structures combine by the unification of at least one object. When two objects are unified, the value of every function applying to both objects must be unified too. If the values of a function are atomic values, they must be identical, otherwise unification fails. If they are polarities, they combine by a special operation called the *product on polarities*. The white polarity is the identity for the product (white · white = white, white · black = black), while the product of two black polarities fails (black · black = $\perp$), which means that two black objects cannot be unified. A derivation consists in saturating the derived structure. The process starts with any elementary structure, and at each step a new elementary structure is combined with the structure resulting from the previous step. This process can only stop when all the objects are black. Special structures, marked as *initial structures* must be used exactly once.[7]

We will introduce several grammars in the paper. Each grammar has its own polarizing function but it will also be articulated with the other grammars using white polarities pertaining to these grammars. In order to distinguish the main polarizing function

[7] Such a structure can be compared to the initial non-terminal symbol of a CFG. But contrarily to a CFG, we do not impose the derivation process to start with an initial structure. Moreover, we accept to have several initial structures, which do not increase the generative capacity of the formalism but allows us to write more elegant grammars.

of each grammar, we will use different symbols (Fig. 8). For instance, an upward triangle Δ is used for syntax and the syntactic grammar is called G^{Δ}, a pentagon $\pentagon$ is used for the semantics-syntax interface, which is called $G^{\pentagon}$, and so on. The topology and the syntax-topology interface will only be introduced in Sect. 5 and in previous sections the linearization grammar $G^{\diamond}$ will make a direct interface between syntax and linear order.

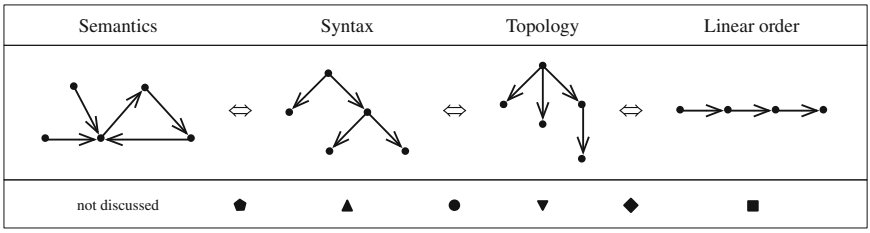


Fig. 8. The modules and their polarities

3.3 Trees and Strings in PUG

It is very easy to write a grammar that builds only trees in PUG, i.e., to force the derived graphs to be trees. Such a grammar, that we will call G^{Δ}, has only two rules: an initial rule introducing a black node, and another rule introducing a black dependency and a black node as its target.[8] This PUG is represented in Fig. 9. The polarity of objects is represented by a triangle next to it, white or black according to its value. Rules are placed in square boxes, with initial rules in double boxes.

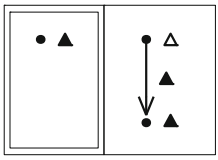


Fig. 9. G^{Δ} (first version: the tree grammar)

The initial rule, which must be used exactly once by definition, introduces the only node that will not be governed, i.e., the root of the tree. The connectedness of the structure is ensured by the process itself, which requires that each time a rule is used, at least one object is unified with the structure obtained at the previous step. The second rule ensures that each node except the root has exactly one governor.

G^{Δ} can be used as a generative grammar, generating all possible trees. But in our case, G^{Δ} will be used as a well-formedness grammar, verifying that a given structure is

8 As our objects are not typed, this grammar does not prevent an edge from becoming the vertex of another edge (see [42] for such structures, called polygraphs). This is avoided here by labelling functions on objects.

a tree. It can be applied for instance on the syntactic structure of (2) (Fig. 5) to verify that it is a tree. To do so, the whole structure will be polarized in white with the polarizing function of the grammar we want to call (cf. the input structure of Fig. 10, which results from the semantic-syntax interface, see Sect. 3.4). After the application of $G^{\triangle}$, we obtain a structure entirely black (output structure of Fig. 10), which means that the structure has been validated as a well-formed tree.

Fig. 10. Input and output structures for $G^{\triangle}$

Applying $G^{\triangle}$ to our input structure does not only allow us to verify that it is a dependency tree, but also enables us to read the whole input structure and to call other grammars that will apply to it. In our case, we want to articulate $G^{\triangle}$ with several grammars, in particular the semantics-syntax interface $G^{\pentagon}$, the linearization grammar $G^{\diamond}$ and the linear order well-formedness grammar $G^{\square}$. For this, we enrich $G^{\triangle}$ to introduce white polarities from these grammars. The enriched version of $G^{\triangle}$ forces $G^{\diamond}$ to apply on every syntactic dependency and $G^{\square}$ to apply on every nodes.[9] Figure 11a shows $G^{\triangle}$ and Fig. 11b shows the output structure when $G^{\triangle}$ is applied to the syntactic structure of (2). Grammars normally only saturate their own polarity; the extra white polarities pertain to other grammars, and are used to articulate them.

Fig. 11. **a.** $G^{\triangle}$ (second version, with articulation polarities) **b.** The output for the syntactic structure of (2)

It is a little more difficult to write a grammar that builds only strings. The simplest way is to use two tree grammars, such as the one in Fig. 9, and to verify that the structure is a tree if we read it in both directions. To couple two grammars and force them to apply on the same structure, we just need to add, on each object of each elementary structures of each grammar, a white polarity of the other grammar. These white polarities call the

9 We could have introduced less (or more) white polarities. Adding diamond polarities on edges to call $G^{\diamond}$ is sufficient. Adding white square polarities on nodes allows us to relax the constraints on possible procedures to trigger our different grammars. With its white square polarities on nodes, $G^{\triangle}$ calls $G^{\square}$, which allows us to trigger $G^{\square}$ before $G^{\diamond}$.

other grammar and ensure that every object has been handled by both grammars. Figure 12 shows two tree grammars coupled together. In order to distinguish the polarity pertaining to each grammar, we use an upward triangle for the polarity pertaining to the first tree grammar and a downward triangle for the polarity pertaining to the second grammar.

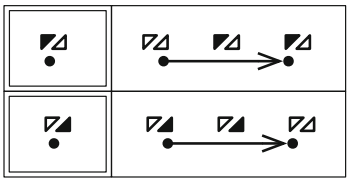


Fig. 12. $G^{\square}$ (first version: two tree grammars coupled together)

After being coupled, the two tree grammars of Fig. 12 form a single grammar. The rules of the coupled grammar can be used in whatever order, despite the fact that they originally come from two different grammars. This property illustrates one of the major advantages of polarities, which allows to couple different modules in a same model without introducing procedural constraints on the order in which the different modules must be triggered.

It is possible to simplify the previous coupled grammar, because both rules involving an edge must apply on each edge and can be pre-combined. The resulting grammar is given in Fig. 13.

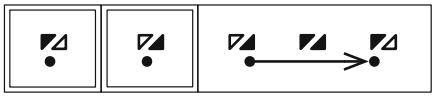


Fig. 13. $G^{\square}$ (second version: pre-combined edge rules)

It is also possible to merge the two polarizing functions. The resulting polarizing function has four possible values: (black, black), (black, white), (white, black), and (white, white). The polarities (black, white) and (white, black) are two opposite polarities, which, combined together, give (black, black), indicating that an object is saturated. In previous publications, these polarities have been called *positive* and *negative* [27, 28, 31]. Figure 14 shows $G^{\square}$ after merging the polarizing function. Positive and negative polarities are represented by + and – signs in squares, and (black, black) is represented by a black square.

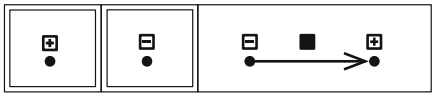


Fig. 14. $G^{\square}$ (third version: polarizing functions merged)

Finally, $G^{\square}$ will be articulated with $G^{\diamond}$ and G^{Δ} to be applied to the output of these grammars, so that it triggers them and it is triggered by them, so that the output of one module becomes the input of another. Figure 15 presents the resulting grammar, where triangle and diamond polarities pertaining to, respectively, $G^{\diamond}$ and G^{Δ} have been added.

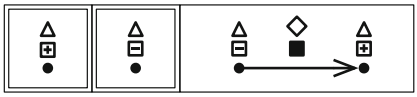


Fig. 15. $G^{\square}$ (fourth version: articulated with $G^{\diamond}$ and G^{Δ})

Note that as soon as the polarities have been added, it is no longer necessary to add the label "<" to distinguish LPLs from syntactic dependencies. The polarities suffice: an edge with a triangle polarity is an edge that must be handled by G^{Δ}, that is, a syntactic dependency, while an edge with a square polarity is an edge that must be handled by $G^{\square}$, that is, an LPL.[10]

3.4 Sub-categorization Rules and Semantics-Syntax Interface

PUG was initially introduced for writing DGs producing unordered dependency trees [26, 29, 32]. Figure 16 shows the semantics-syntax interface $G^{\pentagon}$, that is, a grammar that generates the dependency tree of (1), on semantic grounds. The grammar here is simplified to show only syntactic edges, and morphosyntax (i.e., inflection) is not considered. See [23] or [26] for a more detailed grammar.

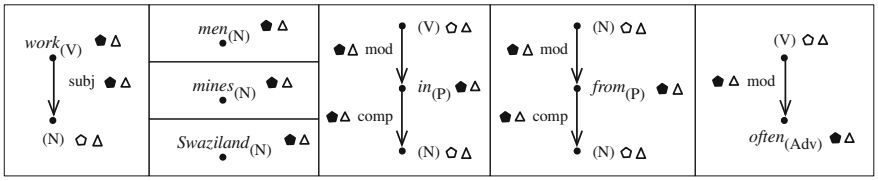


Fig. 16. $G^{\pentagon}$, a dependency grammar for (1)

The rule introducing *work* specifies that it is an intransitive verb with only one semantic argument, its subject. The main polarities for $G^{\pentagon}$ are represented by pentagons.

10 To some extent, it is not even necessary to separate syntactic edges and LPLs. A same edge can be both and bear simultaneously a triangle and a square polarity, exactly as a node can be both a syntactic node and a node of the linear order. It is viewed as a syntactic object when the triangle polarity is handled and as a morphological object when the square polarity is handled. In some sense, every edge we consider has two faces: a semantic one (represented here by the syntactic dependency) and a morphological one (represented by an LPL). They describe constructions, that is, linguistic signs with a signified (the semantic face) and a signifier (the morphological face).

White syntactic polarities are added to black pentagons in order to call G^{Δ}. Two labelling functions for nodes are considered: one has a word as value, the other its category, written between parentheses here. The rule for *work* introduces three objects. It says that *work* is of category V and has a subject dependent. The node *work* and the *subj* dependency are resources produced by the rule (they have black pentagons), while the dependent node of category N is a requirement. This need will be filled by the unification with the elementary structure for *men*, which is a black node of category N. The rule for the preposition *in* says that *in* is a preposition that needs a V as governor and an N as dependent. In other words, the elementary structure for *in* will adjoin on *works*. Prepositions *in* and *from* are binary predicates. The adverb *often* is a unary predicate modifying its only argument. Nouns in this sentence have no arguments.

It is important to note that $G^{\pentagon}$ does not force the derived structure to be a tree. This condition is often verified by a hidden procedural device (as in [29] or [32]). In PUG, this is made explicit by forcing the tree grammar G^{Δ} to apply on the syntactic part of the structure.

3.5 Governor-Dependent Linearization Rules

Sketches of the two rules needed to linearize sentence (2) were introduced in Fig. 6. To include these rules in PUG we must decide which objects in these rules are resources and which are requirements. Let us call $G^{\diamond}$ the linearization grammar that includes these rules. It is a correspondence grammar, like $G^{\pentagon}$ (Sect. 3.4), as it puts in correspondence two structures, the syntactic tree and the linear order. Hence, we propose that $G^{\diamond}$ produces both structures. Moreover, it calls G^{Δ} to verify that the syntactic part of the structure is a tree, as well as $G^{\square}$ to verify that the corresponding order is a string. We obtain the rules in Fig. 17, where the values of the polarizing function pertaining to $G^{\diamond}$ are represented by diamonds. Syntactic dependencies receive a black diamond and a white triangle, to trigger G^{Δ}, while LPLs receive a black diamond and a white square, to trigger $G^{\square}$. We consider that nodes are not handled by $G^{\diamond}$ and are just not polarized.[11]

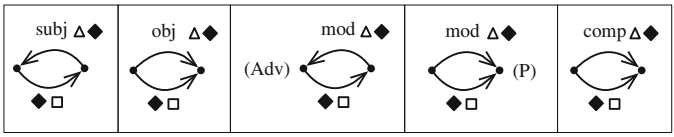


Fig. 17. Governor-dependent linearization rules of $G^{\diamond}$

11 In [28], such objects received a neutral grey polarity that was the identity for the product on polarities, which is equivalent to having no polarity.

3.6 Architecture of the Model and Procedure

The three grammars we have introduced, G^{Δ}, $G^{\square}$ and $G^{\diamond}$, call one another. This illustrates the fact that PUG can formalize complex architectures and is not restricted to pipeline architectures where modules form a chain. PUG has the advantage of enabling various procedures.[12]

In fact, our three grammars are now gathered in only one grammar, which we call $G^{\Delta} \times G^{\diamond} \times G^{\square}$. The three grammars are still visible as three modules of our combined grammar, but we do not need to separate them. Elementary trees of the three grammars can be triggered in whatever order we want. In particular, the combined grammar is reversible and can be used for both synthesis and analysis. In synthesis, G^{Δ} is triggered first to verify that the input structure is a dependency tree, then $G^{\diamond}$ associates a linear order to it, and finally $G^{\square}$ verifies that the output is a string. In analysis, the reverse occurs: $G^{\square}$ is triggered first, calling $G^{\diamond}$ for the correspondence, and G^{Δ} verifies the well-formedness of the output. It is also possible to trigger G^{Δ} and $G^{\square}$ first, and only then verify that the generated tree and string are compatible via $G^{\diamond}$. It is as well possible to mix the grammars and to alternate rules of the different modules. For instance, we can use the grammar for incremental parsing by building the syntactic tree edge by edge as we are consuming the string. Whatever order we choose, the polarization ensures that, if we obtain a saturated structure at the end, we will have a tree and a string corresponding to each other by our linearization rules and that the three modules of the grammar have been applied.

To sum up, there are three main ways to use a correspondence grammar such as $G^{\diamond}$ or $G^{\Delta} \times G^{\diamond} \times G^{\square}$ [33]. First, the grammar can be used as a purely *generative* process, building couples of trees and strings corresponding to one another, from scratch. In our case, the grammar will be used in a *transductive* way, transforming a tree into a string, or a string into a tree. In these cases, the process starts with one of the two structures. To trigger the grammar, this input structure is polarized in white: with the white polarity of $G^{\square}$ if we suppose that it is a string, and with the white polarity of G^{Δ} if we suppose that it is a dependency tree. The "transduction" of the input structure into another one will be automatically achieved by the need to saturate it. There is a third way to use the grammar: the *equative* way. In this case, a couple of structures is considered and the grammar will verify that they are a tree and a string and that they correspond to each other by trying to saturate both of them. [15] said that a Meaning-Text model "is by no means a generative or, for that matter, transformational system: it is a purely EQUATIVE (or translative) device" (p. 45). But in fact, correspondence grammars, like modules of MTT, can be used in all three modes—equative, transductive, and generative.

---

12 It must nevertheless be noted that the space of possible procedures is controlled by the articulation polarities we add in the rules of each grammars. In this presentation we chose to leave open a maximum number of possibilities: each time an object in a rule can be handled by another grammar, we add an articulation polarity to call this grammar. But it is possible to restrict the use of articulation polarities in order to have a pipeline architecture.

In particular, our correspondence grammars are reversible and can be used for synthesis as well as analysis.

4 Linearization Module

The linearization grammar presented so far is not sufficient to order all dependency trees. Our first linearization module (Sect. 3.5) contained only governor-dependent linearization rules, i.e., rules specifying the position of a node in relation to its governor. A second set of rules is needed to position co-dependents in relation to each other (Sect. 4.1). A third set of rules propagates LPLs to nodes that are not in a so close relation in the dependency tree (Sect. 4.2). Non-projective orders will be studied in Sect. 5.

4.1 Co-dependent Linearization Rules

Let us take an example:

(3) *Mary often works.*

Figure 18 shows the dependency tree and linear order of (3). The subject (*Mary*) and the modifier (*often*) are both on the left of their governor (*works*), but the subject must be before the modifier. The subject dependency between *Mary* and *works* now corresponds to an LPL between *Mary* and *often*. This means that we need a linearization rule involving these three nodes—*Mary*, *works*, and *often*—and both syntactic dependencies —*subj* and *mod*. This is the first rule in Fig. 19. It associates the *subj* dependency with an LPL between two co-dependents. Both edges are consumed by this rule, and thus have a black diamond polarity, while the *mod* dependency has a white diamond polarity because it will be consumed by another rule. This latter rule is the second in Fig. 19. It has category constraints and only applies to an adverb (Adv) modifying a verb (V).

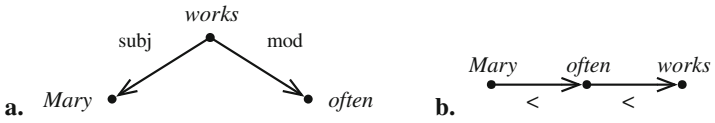


Fig. 18. **a.** Dependency tree **b.** linear order of (3)

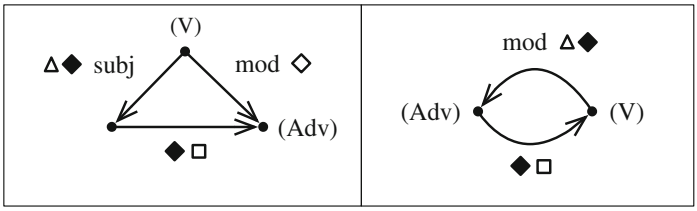


Fig. 19. The *subj-mod* co-dependent and the *mod* governor-dependent linearization rules of $G^{\diamond}$

Figure 20 shows the application of these two rules for (3). The resulting structure contains a dependency tree and a linear order. We only kept the articulation polarities: dependencies have a white triangle polarity calling $G^{\triangle}$ and LPLs have a white square polarity calling $G^{\square}$. The structure is presented twice: once with a dependency tree-based layout (Fig. 20a), the other with a linear order-based layout (Fig. 20b). Let us emphasize that the two schemata are just different views of the exact same graph.

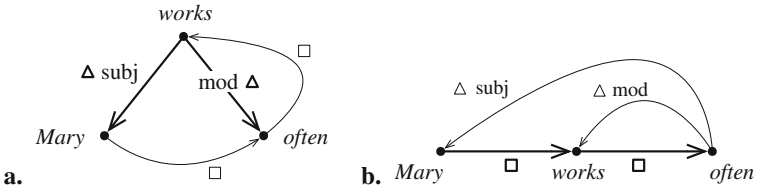


Fig. 20. Syntactic dependencies and LPLs of (3) after application of $G^{\diamond}$

The *subj* governor-dependent linearization rule of Fig. 17, placing the subject before the verb, is still needed in $G^{\triangle}$ because the adverb can be absent. If this rule is applied to the syntactic tree of (3) instead of the *subj-mod* co-dependent linearization rule, we will obtain a structure that is not a string and will be rejected by $G^{\square}$.

The grammar presented here only handles the linearization of what we will call direct projections. The *direct projection* of a node x in a dependency tree is the node itself and its direct dependents. For instance, the direct projection of *works* in (1) is *men often work in*. We will see in Sect. 4 how to propagate the LPLs between the words of the direct projections to the whole set of nodes. Before that, we will see how to homogenize the rules used to order direct projections.

4.2 Propagation and Projectivity

After applying governor-dependent and co-dependent linearization rules on the dependency tree of (1), we obtain the structure in Fig. 21. As before, we kept the articulation polarities and the structure is presented twice, in a dependency tree-based layout (Fig. 21a) and in a linear order-based layout (Fig. 21b).

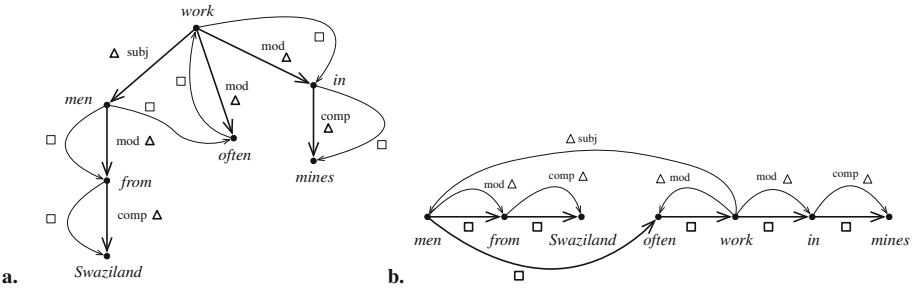


Fig. 21. Syntactic dependencies and LPLs of (1) after application of $G^{\diamond}$

As we can see, we cannot obtain a string with the application of only governor-dependent and co-dependent linearization rules. We know that *men* is before its dependent *from* and its co-dependent *often*, but we do not know the relative order of *from* and *often*, and this order cannot be computed by our current $G^{\diamond}$, because *from* and *often* are in an "aunt-niece" relation in the dependency tree, and $G^{\diamond}$ covers only governor-dependent (i.e., mother-daughter) and co-dependent (i.e., sister) relations.

To obtain the complete linearization of the dependency tree, we must use a property of the correspondence between a dependency tree and a linear order: projectivity. There are several equivalent definitions of projectivity. The definition we use, stated by [34], who coined the term *projectivity*, is based on the notion of projection. The *maximal projection* of a node is the set formed by the node itself and all the nodes it dominates (directly or indirectly) in the dependency tree. A linearly ordered tree is *projective* if and only if the maximal projection of every node of the dependency tree is continuous in the linear order.

A consequence of projectivity is that, if a node x is before a node y that is not in the projection of x, then the whole projection of x is before y and, in particular, every dependent z of x is before y. This property (and the symmetric property where x is after y) can be translated into a rule. This rule (on the right side of Fig. 22) says that any LPL from a node x to a node y can be replaced by an LPL from z to y. The LPL between x and y receives a black square polarity, which means that it can no longer be covered by $G^{\square}$, while the new LPL that replaces it receives a white square polarity and is active for $G^{\square}$. In analysis, the rule is read in the reverse order: any LPL from a node z to a node y can be replaced by an LPL from the governor of z to y. The former LPL receives a black diamond polarity and is no longer active for $G^{\diamond}$, while the new LPL that replaces it receives a white diamond polarity and is active for $G^{\diamond}$. The white triangle polarity between the sources of the two LPLs of rules of Fig. 22 forces these two nodes to be linked by a syntactic dependency (which G^{Δ} will saturate).

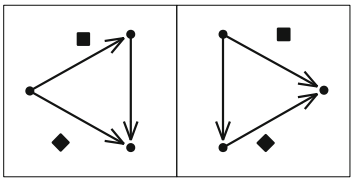


Fig. 22. Propagation rules of $G^{\diamond}$

Propagation rules propagate LPLs downwards in synthesis and upwards in analysis. Figure 23 shows their application in synthesis for a fragment of the structure in Fig. 21. The first rule of Fig. 22 is applied twice to propagate the LPL between *men* and *often* to *from* and then to *Swaziland*. The result is a string and will be saturated (i.e., accepted) by $G^{\square}$.

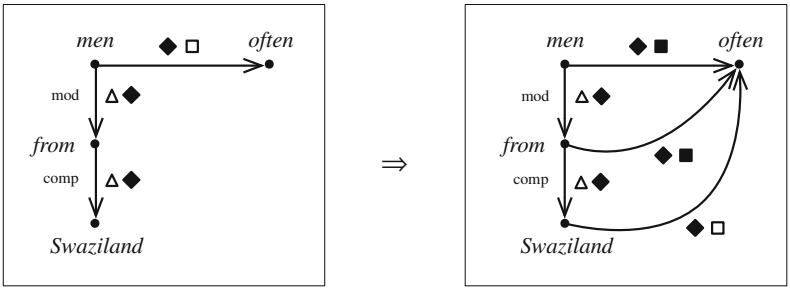


Fig. 23. Application of propagation rules on the structure of (1)

Let us make a remark about the set of LPLs built by our grammar. Order is a transitive relation. From a computational point of view (and certainly also from a cognitive point of view), it is not necessary to grasp all the LPLs that a linear order implies. As we see here, we need mainly to consider immediate LPLs, i.e., LPLs between two successive words, and some other LPLs that result from the propagation of immediate LPLs. Thus, propagation is a kind of transitive closure of linear order, but highly constrained by syntactic structure.

The previous remark is illustrated by Fig. 24, which shows the application of the linearization module for the analysis of (1). The input structure is the string of words, which contains only immediate LPLs (cf. Figure 1). The application of $G^{\square}$ introduces white diamond polarities on immediate LPLs, which call $G^{\diamond}$ (Fig. 24a). After applying all the rules of $G^{\diamond}$ that can saturate a white diamond polarity now, we obtain the structure in Fig. 24b. Only one LPL has not been saturated, the LPL from *Swaziland* to *often*. Neither governor-dependent nor co-dependent linearization rules can be triggered on this LPL. But this LPL can be propagated, producing the configuration in Fig. 24c, on which the *subj-mod* co-dependent linearization rule can now apply, producing the output structure in Fig. 24d. This structure contains the whole dependency tree.

Propagation rules can only produce a projective structure and they suffice to linearize any output of governor-dependent and co-dependent rules. They would apply to co-dependents if we did not take necessary precautions. In fact, any co-dependent linearization rule is the combination of a governor-dependent linearization rule and a propagation rule, as shown in Fig. 25. When the relative order of two co-dependents is free, it can be realized by propagation rules, but when it is not free, then propagation rules must be blocked. This could be done by adding a special feature to the LPL introduced by a governor-dependent rule, as well as the LPL consumed by a propagation rule. In other words, the two LPLs that unify in Fig. 25 would need to have a different value for this special feature (not represented here) if we wanted to block this unification.

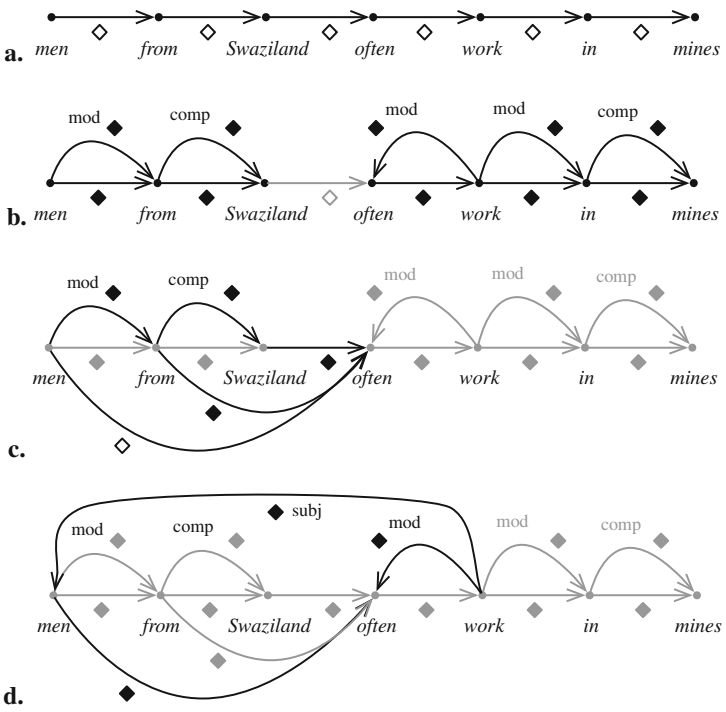


Fig. 24. Application of $G^{\diamond}$ for the analysis of (1)

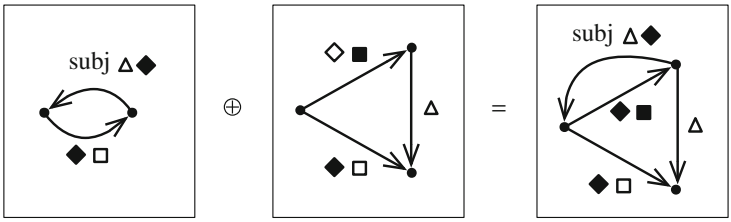


Fig. 25. Co-dependent linearization rules as combinations of governor-dependent linearization and propagation rules

5 Emancipation

Non-projective correspondence between a dependency tree and linear order is common in natural language. In English, it is illustrated by so-called extraction phenomena, like the anteposition of a complement governed by a subordinate verb outside the subordinate clause (cf. sentence (4) and its linearized dependency tree in Fig. 26).

(4) *To Mary Peter thinks we should not speak again.*

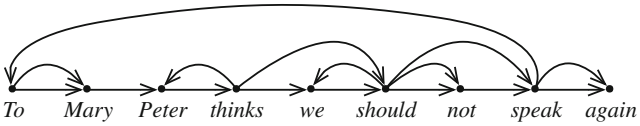


Fig. 26. Non-projective linearized dependency tree of (4)

Such sentences are quite frequent in German, because the first element of a sentence can easily be the dependent of a subordinate verb, as illustrated in (5) (see Fig. 27a, b).

(5) *Das Buch hat diesem Mann niemand zu lesen versprochen* the$_{ACC}$ book has this$_{DAT}$ man nobody$_{NOM}$ to read promised 'Nobody promised this man to read the book'.

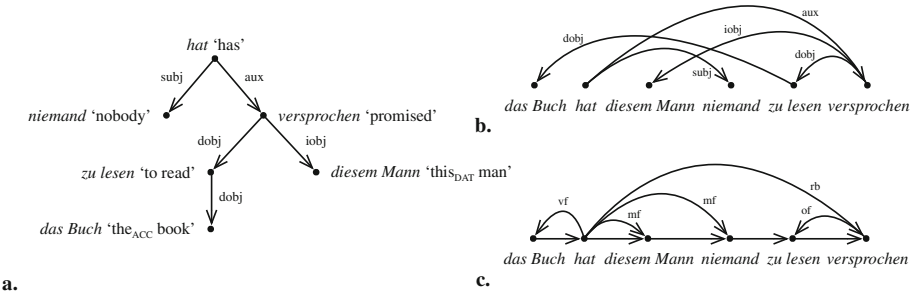


Fig. 27. **a.** The dependency tree of (5) **b.** Its non-projective linearization **c.** The corresponding projective topological tree

In almost any formalism, non-projective structures are solved by the raising of problematic elements in the syntactic structure. This is achieved by similar devices in all theories: movements in Generative Grammar (Move α [35]), non-local features in HPSG (the slash feature [36]), functional uncertainty in LFG [37], etc. In DG, the problem can be solved by raising problematic elements in the syntactic dependency tree [10, 38, 39]. The idea underlying raising in all of these frameworks is that non-projectivity occurs when an element is not positioned in relation with its direct governor, but one of its indirect governors. For instance, in (4), *to Mary* is not positioned in relation to its governor *speak*, but to *thinks*, the main verb. In (5), *das Buch* 'the book' is not positioned in relation to its governor *zu lesen* 'to read', but to *hat* 'has', the main verb (Fig. 27).

There are different ways to control raising. In Generative Grammar, the movement is constrained by syntactic categories: some syntactic constituents are "islands" and it is not possible to cross their boundaries (see [40] for a first description of island constraints and [41] for a recent formalization). In LFG or traditional DG, the constraints are expressed in terms of syntactic functions: the chain of syntactic dependencies between the raised element and the ancestor that "hosts" it in the linear order is constrained by the nature of their syntactic functions [15, 16].

In this paper we adopt a third solution: the topological model. This model has been developed from the 19th century for the description of word order in German and has been formalized in HPSG [6] and DG [11, 12]. It elegantly models the fact that German

is a V2 language (the main verb of a declarative sentence always occupies the second position), with a verb cluster at the end of the sentence, possibly followed by some extraposed heavy constituents. This is modeled by decomposing the main domain of a German sentence into five fields (Vorfeld, left bracket, Mittelfeld, right bracket, and Nachfeld = vf, lb, mf, rb, and nf), with the following conditions: the main verb goes in the left bracket, the other verbs in the right bracket, where they form a verb cluster, one constituent goes in the Vorfeld, the others in the Mittelfeld, and some heavy constituents in the Nachfeld. In the verb cluster [vc], each verb is placed to the left of its governor, in a field called the Oberfeld (of). Noun phrases cannot be placed in the verb cluster and if they depend on a verb in the verb cluster, they must emancipate and go in one of the major fields (Vorfeld, Mittelfeld, or Nachfeld).

In (5), the main verb is the finite auxiliary *hat* 'has', which must be in second position, in the left bracket. Its verbal dependent, the participle *versprochen* 'promised', is placed in the right bracket, where it forms a verb cluster accommodating its verbal dependent *zu lesen* 'to read'. The noun phrases of these two verbs cannot be placed in the verb cluster and will be emancipated to be placed in fields of the main domain: *das Buch* 'the book' goes in the Vorfeld, while *diesem Mann* 'to this man' is placed in the Mittelfeld, where it joins the subject *niemand* 'nobody' and can be ordered freely in relation to it.

The topological structure can be represented by a constituent structure as in [12] or by a dependency tree as in [11]. This second representation is adopted here (Fig. 27c). The topological tree is added as an intermediate structure between the syntactic tree and linear order. This new structure receives its own polarity, represented by a downward triangle (∇), to be contrasted with the upward triangle (Δ) of syntactic trees. The diamond ($\diamond$) is still used for the linearization module, which now ensures the correspondence between the topological tree and linear order, but uses the same rules as the ones described in previous sections, because the topological tree corresponds to the linear order projectively. A new polarity, represented by a circle ($\circ$), is introduced for the syntax-topology interface, which ensures the correspondence between the syntactic tree and the topological tree.

The rules of the syntax-topology interface are very similar to the rules of the linearization module: the majority are correspondence rules associating a syntactic edge to a topological edge. The other rules resolve mismatches between the two structures. For the linearization module, they are propagation rules. For the syntax-topology interface, they are emancipation rules, which are very similar to propagation rules.[13] Figure 28 shows examples of different rules used in the syntax-linear order correspondence: the first two transform a syntactic *aux* edge into a topological *rb* edge (allowing the dependent of the auxiliary to go in the right bracket where it heads a verb cluster [vc]) and a syntactic *obj* into a topological *vf* (allowing a direct object to go in the Vorfeld); the next rule is an emancipation rule lifting a *vf* and allowing a node placed in the Vorfeld to emancipate from the right bracket (note that the lower *vf* is only visible for the syntax-topology interface, while the upper *vf* is only visible for the topological well-formedness

[13] The semantics-syntax interface also contains mismatch rules for phenomena such as raising (*Peter seems to sleep*), auxiliaries (*I will read*), *tough*-movement (*a paper hard to read*), or extraction (*a paper I would like to read*). See [25, 26, 32].

module, as shown by their polarities); the following rules are topological well-formedness rules verifying that the topological structure is a tree and the Vorfeld is in the main domain [md]; the last rules are linearization rules placing the Vorfeld to the left and the right bracket after the Mittelfeld, as well as a propagation rule.

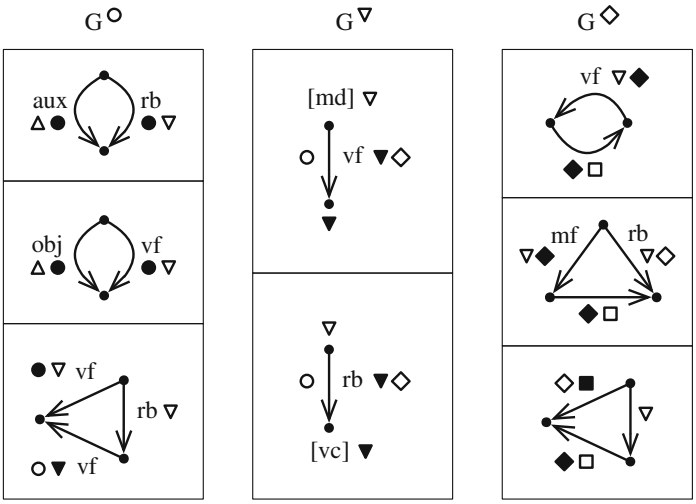


Fig. 28. Some rules of syntax-topology-linear order correspondence

The application of syntax-topology correspondence rules produces a copy of the syntactic tree where the node and edge labels have changed. The application of the emancipation rules lifts some edges (Fig. 29). The emancipation could have been done directly in the syntactic tree, but the advantage of relabelling the tree in the syntax-topology is to have two different grammars for the well-formedness of the syntactic tree and the topological tree.

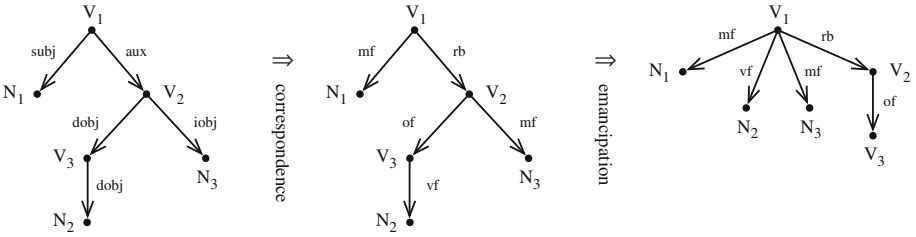


Fig. 29. The syntax-topology interface applied on the trees of (5)

A more complete formal topological model for German can be found in [12].

6 Conclusion

This paper pursued several goals: showing that linearization can be seen as a graph rewriting process, showing that each LPL corresponds to a syntactic dependency and that the path between the two can be traced, and formalizing the linearization module in a modular, declarative formalism, allowing to combine it with other modules of a linguistic model.

The paper gives a large overview of Polarized Unification Grammar. The formalism is well adapted to writing grammars that generate various structures, such as strings, trees, and graphs, and correspondence grammars between such structures. Polarization allows us to split the model into small modules articulated with one another, and to maintain a distributed architecture where every module calls the other modules to handle the same structures. PUG also allows us to deal with node expansions, which has not been developed here.

From a theoretical point of view, this paper proposes a linguistic model with several levels of representation, including a syntactic level where the structure is an unordered dependency tree. The choice of dependency trees for the representation of syntax is supported by two arguments. First, it elegantly interfaces with both semantics and linear order, including non-projective orders. Second, there is a one-to-one correspondence between the syntactic dependencies and the linear precedence links between couples of successive words, and this correspondence can be exploited in various ways, in particular to predict prosodic breaks.

From a formal point of view, we have a quite complex architecture, but the articulation between the different modules is ensured by a single mechanism (polarization), which allows us to control rule combination. This is done without imposing any order on the combination of rules, thus preserving a distributed architecture. The typology of the rules we introduce is also quite simple. We have well-formedness rules verifying that the structure is well-formed (for instance that it is a tree or a string), correspondence rules transforming a structure in another one, and propagation/emancipation rules in case of mismatches.

We did not address the implementation of PUG. In fact, we tried to underspecify procedure as much as possible. Our grammar allows various processing chains, and it was not our purpose to decide which procedure is better for synthesis or parsing. On the order hand, our grammar is very precise on which word order specification must be computed to linearize a dependency tree. We showed that we only need to consider LPLs between successive words, between nodes linked by a syntactic dependency, and between all couples of words that propagation rules may go through. Although no procedure is given, the set of objects that any procedure would have to handle and the set of elementary operations that must be triggered is clearly delimited. Moreover, the word order we produce is explicit (it is a linear graph on words).

References

1. Gazdar, G.: Unbounded dependencies and coordinate structure. Linguist. Inquiry **12**(1), 155–184 (1981)

2. Gazdar, G., Klein, E., Pullum, G., Sag, I.: Generalized Phrase Structure Grammar. Harvard University Press, Cambridge (1985)
3. Candito, M.-H.: A principle-based hierarchical representation of LTAG. In: Proceedings of COLING, Copenhagen (1996)
4. Kaplan, R., Bresnan, J.: Lexical-functional grammar: a formal system for grammatical representation. In: Bresnan, J. (ed.) The Mental Representation of Grammatical Relations, pp. 173–281. The MIT Press, Cambridge (1982)
5. Bresnan, J.: Lexical-Functional Syntax. Blackwell, Malden (2001)
6. Kathol, A.: Linerarization-based German syntax. Ph.D. thesis, Ohio State University (1995)
7. Richter, F., Sailer, M.: Remarks on linearization. reflections on the treatment of LP-rules in HPSG in a typed feature logic. Master's dissertation, Eberhard-Karls-Universität, Tübingen (1995)
8. Müller, S., Kasper, W.: HPSG analysis of German. In: Wahlster, W. (ed.) Verbmobil: Foundations of Speech-to-Speech Translation, pp. 238–253. Springer, Heidelberg (2000)
9. Tesnière, L.: Éléments de syntaxe structurale. Klincksieck, Paris (1959). English translation: Elements of structural syntax. John Benjamins (2015)
10. Kahane, S., Nasr, A., Rambow, O.: Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In: Proceedings of COLING-ACL, Montreal, pp. 646–652 (1998)
11. Duchier, D., Debusmann, R.: Topological dependency trees: a constraint-based account of linear precedence. In: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, Toulouse, pp. 180–187 (2001)
12. Gerdes, K., Kahane, S.: Word order in German: a formal dependency grammar using a topological hierarchy. In: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001), Toulouse, pp. 220–227 (2001)
13. Kuhlman, M.: Mildly non-projective dependency grammar. Comput. Linguist. **30**(2), 355–387 (2013)
14. Kuhlmann, M., Nivre, J.: Transition-based techniques for non-projective dependency parsing. Northern Eur. J. Lang. Technol. **2**(1), 1–19 (2010)
15. Mel'čuk, I.: Dependency Syntax: Theory and Practice. State University of New York Press, Albany (1988)
16. Mel'čuk, I., Pertsov, N.: Surface Syntax of English: A Formal Model Within the Meaning-Text Framework. John Benjamins, Amsterdam (1987)
17. Iordanskaja, L., Mel'čuk, I.: Ordering of Simple Clauses in an English Complex Sentence. Rhema **4**, 17–59 (2015)
18. Mel'čuk, I.: Semantics: From Meaning to Text, vol. 1. John Benjamins, Amsterdam (2012)
19. Mel'čuk, I.: Actants in semantics and syntax I: actants in semantics. Linguistics **42**(1), 1–66 (2004)
20. Kahane, S.: Dependency and Valency: An International Handbook of Contemporary Research. Walter de Gruyter, Berlin (2003)
21. Copestake, A.: Slacker semantics: why superficiality, dependency and avoidance of commitment can be the right way to go. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, Athens, pp. 1–9 (2009)
22. Kahane, S., Mel'čuk, I.: Synthèse des phrases à extraction en français contemporain (du réseau sémantique à l'arbre syntaxique). Traitement Automatique des Langues **40**(2), 25–85 (1999)
23. Kahane, S., Lareau, F.: Meaning-Text Unification Grammar: modularity and polarization. In: Proceedings of the Second International Conference on Meaning-Text Theory, Moscow, pp. 163–173 (2005)

24. Lareau, F.: Vers une grammaire d'unification Sens-Texte du français: le temps verbal dans l'interface sémantique-syntaxe. Ph.D. thesis, Université de Montréal/Université Paris 7 (2008)
25. Lareau, F.: Le temps verbal dans l'interface sémantique-syntaxe du français. In: Proceedings of the Fourth International Conference on Meaning-Text Theory, Barcelona (2009)
26. Kahane, S.: Predicative Adjunction in a Modular Dependency Grammar. In: Proceedings of the 2nd International Conference on Dependency Linguistics (DepLing), Prague, pp. 137–146 (2013)
27. Kahane, S.: Grammaires d'Unification Polarisées. In: Actes de la 11ème conférence sur le Traitement Automatique des Langues Naturelles, Fès, pp. 233–242 (2004)
28. Kahane, S.: Polarized unification grammars. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, pp. 137–144 (2006)
29. Nasr, A.: A formalism and a parser for lexicalized dependency grammars. In: 4th International Workshop on Parsing Technologies, Prague, pp. 186–195 (1995)
30. Perrier, G.: Interaction grammars. In: Proceedings of the 18th International Conference on Computational Linguistics, Saarbrücken, pp. 600–606 (2000)
31. Duchier, D., Thater, S.: Parsing with tree descriptions: a constraint-based approach. In: Proceedings of the Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP), Las Cruces, NM, pp. 17–32 (1999)
32. Kahane, S.: A fully lexicalized grammar for french based on meaning-text theory. In: Gelbukh, A. (ed.) CICLing 2001. LNCS, vol. 2004, pp. 18–31. Springer, Heidelberg (2001)
33. Kahane, S.: Des grammaires formelles pour définir une correspondance. In: Actes de la 7e conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN), Lausanne (2000)
34. Lecerf, Y.: Une représentation algébrique de la structure des phrases dans diverses langues natuelles. Comptes Rendus de l'Académie des Sciences de Paris **252**, 232–234 (1961)
35. Chomsky, N.: The Minimalist Program. The MIT Press, Cambridge (1995)
36. Pollard, C., Sag, I.: Head-Driven Phrase Structure Grammar. CSLI, Stanford (1994)
37. Kaplan, R., Zaenen, A.: Long-distance dependencies, constituent structure, and functional uncertainty. In: Baltin, M., Kroch, A. (eds.) Alternative Conceptions of Phrase Structure, pp. 17–42. Chicago University Press, Chicago (1989)
38. Bröker, N.: Unordered and non-projective dependency grammars. Traitement Automatique des Langues **41**(1), 245–272 (2000)
39. Hudson, R.: Discontinuity. Traitement Automatique des Langues **41**(1), 15–56 (2000)
40. Ross, J.: Constraints on variables in syntax. Ph.D. thesis, MIT, Cambridge, MA (1967)
41. Graf, T.: Movement-generalized minimalist grammars. In: Béchet, D., Dikovsky, A. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 7351, pp. 58–73. Springer, Heidelberg (2012)
42. Kahane, S., Mazziotta, N.: Syntactic polygraphs: a formalism extending both constituency and dependency. In: Proceedings of the 14th Meeting on the Mathematics of Language, Chicago, pp. 152–164 (2015)
43. Lareau, F.: Vers une formalisation des décompositions sémantiques dans la Grammaire d'Unification Sens-Texte. In: Actes de la 14ème conférence sur le Traitement Automatique des Langues Naturelles, Toulouse, pp. 163–172 (2007)

Undecidability of the Lambek Calculus with a Relevant Modality

Max Kanovich[1,4], Stepan Kuznetsov[2(✉)], and Andre Scedrov[3,4]

[1] University College London, London, UK
m.kanovich@ucl.ac.uk
[2] Steklov Mathematical Institute, Moscow, Russian Federation
sk@mi.ras.ru
[3] University of Pennsylvania, Philadelphia, USA
scedrov@math.upenn.edu
[4] National Research University Higher School of Economics, Moscow, Russian Federation

Abstract. Morrill and Valentín in the paper "Computational coverage of TLG: Nonlinearity" considered an extension of the Lambek calculus enriched by a so-called "exponential" modality. This modality behaves in the "relevant" style, that is, it allows contraction and permutation, but not weakening. Morrill and Valentín stated an open problem whether this system is decidable. Here we show its undecidability. Our result remains valid if we consider the fragment where all division operations have one direction. We also show that the derivability problem in a restricted case, where the modality can be applied only to variables (primitive types), is decidable and belongs to the NP class.

1 The Lambek Calculus Extended by a Relevant Modality

We start with the version of the Lambek calculus, $\mathbf{L}^*$, that allows empty left-hand sides of sequents (introduced in [10]). We will introduce $\mathbf{!L}^*$—an extension of $\mathbf{L}^*$ with one modality, denoted by !.

Formulae of $\mathbf{!L}^*$ are built from a set of variables ($\mathrm{Var} = \{p, q, r, \ldots\}$) using two binary connectives, $/$ *(right division)* and $\backslash$ *(left division)*, and additionally one unary connective, !. Capital Latin letters denote formulae; capital Greek letters denote finite (possibly empty) *linearly ordered sequences* of formulae.

Following the linguistic tradition, formulae of the Lambek calculus (and its extensions) are also called *types.* In this terminology, variables are called *primitive types.*

We present $\mathbf{!L}^*$ in the form of sequent calculus. Sequents of $\mathbf{!L}^*$ are of the form $\Pi \to A$, where A is a formula and Π is a finite (possibly empty) linearly ordered sequence of formulae. Π and A are called the antecedent and the succedent respectively.

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 240–256, 2016.
DOI: 10.1007/978-3-662-53042-9_14

The axioms and rules of $\mathbf{!L}^*$ are as follows:

$$\frac{}{A \to A}$$

$$\frac{\Gamma \to A \quad \Delta_1, B, \Delta_2 \to C}{\Delta_1, B \,/\, A, \Gamma, \Delta_2 \to C}\ (/ \to) \qquad \frac{\Gamma, A \to B}{\Gamma \to B \,/\, A}\ (\to /)$$

$$\frac{\Gamma \to A \quad \Delta_1, B, \Delta_2 \to C}{\Delta_1, \Gamma, A \setminus B, \Delta_2 \to C}\ (\setminus \to) \qquad \frac{A, \Gamma \to B}{\Gamma \to A \setminus B}\ (\to \setminus)$$

$$\frac{\Gamma_1, A, \Gamma_2 \to C}{\Gamma_1, !A, \Gamma_2 \to C}\ (! \to) \qquad \frac{!A_1, \ldots, !A_n \to B}{!A_1, \ldots, !A_n \to !B}\ (\to !)$$

$$\frac{\Delta_1, !A, \Gamma, \Delta_2 \to C}{\Delta_1, \Gamma, !A, \Delta_2 \to C}\ (\mathrm{perm}_1) \qquad \frac{\Delta_1, \Gamma, !A, \Delta_2 \to C}{\Delta_1, !A, \Gamma, \Delta_2 \to C}\ (\mathrm{perm}_2)$$

$$\frac{\Delta_1, !A, !A, \Delta_2 \to C}{\Delta_1, !A, \Delta_2 \to C}\ (\mathrm{contr})$$

We call ! the *relevant modality*, since it behaves in a relevant logic style, allowing contraction and permutation, but not weakening. Recall that in the original Lambek calculus there is neither contraction, nor permutation, nor weakening. The modality is introduced to restore contraction and permutation in a controlled way.

The cut rule is not officially included in $\mathbf{!L}^*$. Morrill and Valentín [12] claim that it is admissible and that this fact can be proved using the standard procedure (cf. [11]). In this paper we consider the system without cut and don't need its admissibility.

We also consider fragments of $\mathbf{!L}^*$. Since there is no cut rule in this system, it enjoys the subformula property, and therefore if we restrict the set of connectives, we obtain conservative fragments of $\mathbf{!L}^*$: $\mathbf{!L}^*_{/}$ (where we have only / and !), $\mathbf{L}^*$ (this is the "pure" Lambek calculus without !), $\mathbf{L}^*_{/}$.

As we discuss in more detail in [7], $\mathbf{L}^*$ can be considered [1,18] as a fragment of non-commutative variant of Girard's linear logic [5]. Our modality, !, follows the spirit of the exponential connective in linear logic, allowing contraction and permutation. However, in contrast with the linear logic case, we don't allow weakening. On the other hand, as we discuss in Sect. 2, our ! is motivated from the linguistical point of view.

Theorem 1. *The derivability problem for* $\mathbf{!L}^*$ *is undecidable. Moreover, the derivability problem is undecidable even for* $\mathbf{!L}^*_{/}$.

Remark 1. $\mathbf{!L}^*$ has been constructed as a conservative fragment of a larger system **Db!?**, introduced in [12]. Thus Theorem 1 provides undecidability of **Db!?** (solving an open question raised in [12]).

2 Linguistic Examples and Motivations

In this section we start from the standard examples of Lambek-style syntactic analysis [4,9] and then follow [12].

In syntactic formalisms based on the Lambek calculus and its variants, Lambek types (formulae) denote syntactic categories. We use the following standard primitive types: n stands for *common noun* (like *"book"* or *"person"*); np stands for *noun phrase* (like *"John"* or *"the book"*); s stands for the whole *sentence.* Actually, n and np represent not only isolated nouns and noun phrases, but also syntactic groups with similar properties: *e.g., "the red book"* or *"the person whom John met yesterday,"* from the lingustic point of view, should be treated as a noun phrase (np) as well. The latter cannot be proved to be of type np by means of $\mathbf{L}^*$, but $\mathbf{!L}^*$ can handle this.

For simplicity, in our examples we don't distinguish singular and plural forms.

Other parts of speech receive compound types: $np \setminus s$ stands for *intransitive verb* (like *"runs"* or *"sleeps"*); $(np \setminus s) / np$ stands for *transitive verb* (*"likes," "reads," "met," "admire"*); $(np \setminus s) \setminus (np \setminus s)$ is the type for *adverbs* like *"yesterday"* (it takes an intransitive verb group from the left-hand side and yields a compound intransitive verb group); np / n is the type for *"the,"* etc.

If the sequent $A_1, \ldots, A_n \to B$ is derivable in the Lambek calculus or its extension, syntactic objects of type $A_1, \ldots, A_n$, taken together in the specified linear order, are considered to form an object of type B. For example, since $(np \setminus s) / np, np / n, n \to np \setminus s$ is derivable, *"reads the book"* is an expression of type $np \setminus s$, or, in other words, acts as an intransitive verb.

Example 1.

"John met Pete." *"John met Pete yesterday."*

These two sentences receive type s, since the sequents $np, (np \setminus s) / np, np \to s$ and $np, (np \setminus s) / np, np, (np \setminus s) \setminus (np \setminus s) \to s$ are both derivable in $\mathbf{L}^*$.

Example 2.

"the person whom John met"

As mentioned above, we want this phrase to receive type np. This is obtained by assigning type $(n \setminus n) / (s / np)$ to *"whom." "John met"* has type s / np, which means "a sentence that lacks a noun phrase on the right-hand side." In other terms, we have a *gap* after *"met."* In Example 1 this gap is filled by *"Pete,"* and here it is intentionally left blank.

Example 3.

"the person whom John met yesterday"

Here the gap appears in the middle of the clause (between *"met"* and *"yesterday"*), therefore *"John met yesterday"* is neither of type s / np, nor of type $np \setminus s$. This situation is called *medial extraction* and is not handled by $\mathbf{L}^*$.

To put np into the gap, we use ! and the (perm_1) rule:

$$\dfrac{np / n, n, n \setminus n \to np \qquad \dfrac{\dfrac{\dfrac{np, (np \setminus s) / np, np, (np \setminus s) / (np \setminus s) \to s}{np, (np \setminus s) / np, !np, (np \setminus s) / (np \setminus s) \to s}\,(! \to)}{np, (np \setminus s) / np, (np \setminus s) / (np \setminus s), !np \to s}\,(\text{perm}_1)}{np, (np \setminus s) / np, (np \setminus s) / (np \setminus s) \to s / !np}}{\underset{\textit{the}}{np / n}, \underset{\textit{person}}{n}, \underset{\textit{whom}}{(n \setminus n) / (s / !np)}, \underset{\textit{John}}{np}, \underset{\textit{met}}{(np \setminus s) / np}, \underset{\textit{yesterday}}{(np \setminus s) / (np \setminus s)} \to np}$$

The sequent on top is the same schema as for *"John met Pete yesterday"* (see Example 1).

Note that $s / !np$ and $!np \setminus s$ are equivalent (due to the permutation rules).

Example 4.

"the paper that John signed without reading"

Finally, this is the case called *parasitic extraction,* with two np gaps (after *"signed"* and after *"reading"*). If the *that*-clause were an independent sentence, the gaps would have been filled like this: *"John signed the paper without reading the paper."* To fill both gaps with the same np, we use the (contr) rule:

$$\dfrac{np / n, n, n \setminus n \to np \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{np, (np \setminus s) / np, np, ((np \setminus s) / (np \setminus s)) / np, np / np, np \to s}{np, (np \setminus s) / np, !np, ((np \setminus s) / (np \setminus s)) / np, np / np, !np \to s}}{np, (np \setminus s) / np, ((np \setminus s) / (np \setminus s)) / np, np / np, !np, !np \to s}}{np, (np \setminus s) / np, ((np \setminus s) / (np \setminus s)) / np, np / np, !np \to s}}{np, (np \setminus s) / np, ((np \setminus s) / (np \setminus s)) / np, np / np \to s / !np}}{\underset{\textit{the paper}}{np / n, n}, \underset{\textit{that}}{(n \setminus n) / (s / !np)}, \underset{\textit{John}}{np}, \underset{\textit{signed}}{(np \setminus s) / np}, \underset{\textit{without}}{((np \setminus s) / (np \setminus s)) / np}, \underset{\textit{reading}}{np / np} \to np}$$

Here *"that"* acts exactly as *"whom,"* and *"without"* modifies the verb group like *"yesterday"* does, but also requires a noun phrase *"reading the paper"* on the right side. The sequents on the top are easily derivable in $\mathbf{L}^*$.

Remark 2. Our calculus $\mathbf{!L}^*$, as well as **Db!?**, works well for pure complex sentences and pure compound sentences. However, we meet with difficulties in the mixed case, caused by sophisticated nature of *"and"* and the like. For example, the fact that *"John met Pete yesterday and Mary met Ann today"* has type s, leads to and unwanted classification of **"the person whom John met yesterday and Mary met Ann today"* as a noun phrase (type np), cf. Example 3. In order to address this issue, Morrill and Valentín [12] suggest another variant of the system, denoted by $\mathbf{Db!?_b}$. This variant includes *brackets* that disallow gapping in certain situations. Morrill and Valentín pose the decidability question both for **Db!?** and $\mathbf{Db!?_b}$. In this paper we solve the first question.

Remark 3. The whole system $\mathbf{!L}^*$ turns out to be undecidable (Theorem 1). On the other hand, notice that in these examples and the like can be treated using

types of a very restricted form. Namely, ! is applied only to a primitive type (for instance, $!np$). In Sect. 5 we show that this restricted fragment of $\mathbf{!L}^*$ is decidable. Moreover, it belongs to NP, *i.e.,* can be resolved by a nondeterministic polynomial algorithm.

3 L* with Buszkowski's Rules

In this section we build an undecidable extension of $\mathbf{L}^*$ with a finite set of rules, generally following the construction by W. Buszkowski from [2]. Buszkowski, however, considers another version of the Lambek calculus, $\mathbf{L}$, introduced in [9]. The difference between $\mathbf{L}$ and $\mathbf{L}^*$ is the so-called Lambek's restriction: in $\mathbf{L}$, the antecedents of all sequents are forced to be non-empty. In this paper, following Morrill and Valentín [12], we allow empty antecedents, and Lambek's restriction is not valid in $\mathbf{L}^*$ (*e.g.,* $\to p \,/\, p$ is derivable in $\mathbf{L}^*$). The relationship between $\mathbf{L}$ and $\mathbf{L}^*$ is very subtle. For instance, the sequent $q \,/\, (p \,/\, p) \to q$ is derivable in $\mathbf{L}^*$, but becomes underivable when Lambek's restiction is imposed (despite the fact that this sequent itself has a non-empty antecedent). Therefore one has to be very cautious with this issue, and for this reason here we provide a modification of Buszkowski's construction for $\mathbf{L}^*$ rather than directly use results from [2].

Let $\mathbf{L}^* + \mathcal{R}$ be $\mathbf{L}^*$ extended with a finite set $\mathcal{R}$ of rules of two special forms:

$$\frac{\Pi_1 \to p \quad \Pi_2 \to q}{\Pi_1, \Pi_2 \to r}\ (\mathrm{B}_1) \qquad \text{or} \qquad \frac{\Pi, q \to p}{\Pi \to r}\ (\mathrm{B}_2),$$

where p, q, r are fixed *primitive* types. We call these rules *Buszkowski's rules.*

Theorem 2. *The cut rule*

$$\frac{\Pi \to A \quad \Delta_1, A, \Delta_2 \to C}{\Delta_1, \Pi, \Delta_2 \to C}\ (\mathrm{cut})$$

is admissible in $\mathbf{L}^* + \mathcal{R}$ *for an arbitrary set* $\mathcal{R}$ *of Buszkowski's rules.*

Proof. We proceed by double induction. We consider a number of cases, and in each of them the cut either disappears, or is replaced by cuts with simpler cut formulae (A), or is replaced by a cut for which the depth of at least one derivation tree of a premise ($\Pi \to A$ or $\Delta_1, B, \Delta_2 \to C$) is less than for the original cut, and the cut formula remains the same. Thus by double induction (on the outer level—on the complexity of A, on the inner level—on the sum of premise derivation tree depths) we get rid of the cut.

Case 1: A is not the type that is introduced by the lowermost rule in the derivation of $\Delta_1, A, \Delta_2 \to C$. In this case (cut) can be interchanged with that lowermost rule. Consider the situation when it was (B_1) (other cases are similar):

$$\dfrac{\Pi \to A \qquad \dfrac{\Delta_1', A, \Delta_1'' \to p \quad \Delta_2 \to q}{\Delta_1', A, \Delta_1'', \Delta_2 \to r}\ (\mathrm{B}_1)}{\Delta_1', \Pi, \Delta_1'', \Delta_2 \to r}\ (\mathrm{cut})$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\Pi \to A \quad \Delta_1', A, \Delta_1'' \to p}{\Delta_1', \Pi, \Delta_1'' \to p}\ (\mathrm{cut}) \qquad \Delta_2 \to q}{\Delta_1', \Pi, \Delta_1'', \Delta_2 \to r}\ (\mathrm{B}_1)$$

Case 2: $A = E \,/\, F$, and it is introduced by the lowermost rules both into $\Pi \to A$ and into $\Delta_1, A, \Delta_2 \to C$.

$$\dfrac{\dfrac{\Gamma, F \to E}{\Gamma \to E \,/\, F}\ (\to /) \qquad \dfrac{\Pi \to F \quad \Delta_1, E, \Delta_2 \to C}{\Delta_1, E \,/\, F, \Pi, \Delta_2 \to C}\ (/ \to)}{\Delta_1, \Gamma, \Pi, \Delta_2 \to C}\ (\mathrm{cut})$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\Pi \to F \quad \Gamma, F \to E}{\Gamma, \Pi \to E}\ (\mathrm{cut}) \qquad \Delta_1, E, \Delta_2 \to C}{\Delta_1, \Gamma, \Pi, \Delta_2 \to C}\ (\mathrm{cut})$$

Case 2 for $\setminus$ is handled symmetrically.

Case 3: one of the premises of (cut) is the axiom ($A \to A$). Then the goal coincides with the other premise.

Note that since (B_1) and (B_2) introduce new primitive types only into the succedent, the "bad" case, where both premises of the cut rule are derived using Buszkowski's rules and the cut formula is the formula introduced by both of them, does not occur. This is the key trick that allows to formulate the extended calculus in a cut-free way. □

In the presence of (cut) Buszkowski's rules (B_1) and (B_2) are equivalent to axioms $p, q \to r$ and $p \,/\, q \to r$ respectively, as shown by the following derivations:

$$\dfrac{\Pi_1 \to p \qquad \dfrac{\Pi_2 \to q \quad p, q \to r}{p, \Pi_2 \to r}\ (\mathrm{cut})}{\Pi_1, \Pi_2 \to r}\ (\mathrm{cut}) \qquad \dfrac{\dfrac{\Pi, q \to p}{\Pi \to p \,/\, q}\ (\to /) \qquad p \,/\, q \to r}{\Pi \to r}\ (\mathrm{cut})$$

and in the opposite direction:

$$\dfrac{p \to p \quad q \to q}{p, q \to r}\ (\mathrm{B}_1) \qquad \dfrac{\dfrac{q \to q \quad p \to p}{p \,/\, q, q \to p}\ (/ \to)}{p \,/\, q \to r}\ (\mathrm{B}_2)$$

From this perspective, $\mathbf{L}^* + \mathcal{R}$ can be viewed as a finite *axiomatic extension* of $\mathbf{L}^*$ (with non-logical axioms of a special kind). However, for our purposes it is more convenient to consider rules instead of axioms.

Theorem 3. *Let M be a recursively enumerable set of words over an alphabet Σ without the empty word. If $\Sigma \subset \mathrm{Var}$, and* Var *also contains an infinite number of variables not belonging to Σ, then there exists a finite set $\mathcal{R}_M$ of Buszkowski's rules and $s \in \mathrm{Var}$ such that for any word $a_1 \dots a_n$ over Σ*

$$a_1 \dots a_n \in M \quad \textit{iff} \quad a_1, \dots, a_n \to s \textit{ is derivable in } \mathbf{L}^* + \mathcal{R}_M.$$

We shall use the fact that any recursively enumerable language without the empty word can be generated by a *binary grammar* [3]. A binary grammar is a quadruple $G = \langle N, \Sigma, P, s\rangle$, where N and Σ are disjoint alphabets (Σ is the original alphabet of the language), $s \in N$, and P is a finite set of *productions* of the form[1]

$$w \Rightarrow v_1 v_2 \quad \text{or} \quad v_1 v_2 \Rightarrow w,$$

where $v_1, v_2, w \in N \cup \Sigma$. If $(\alpha \Rightarrow \beta) \in P$ and η and θ are arbitrary (possibly empty) words over $N \cup \Sigma$, then $\eta\alpha\theta \Rightarrow_G \eta\beta\theta$. The relation $\Rightarrow_G^*$ is the reflexive-transitive closure of $\Rightarrow_G$. Finally, the language generated by G is the set of all words $a_1 \dots a_n$ over Σ such that $s \Rightarrow_G^* a_1 \dots a_n$.

Proof. Let M be an arbitrary recursively enumerable language without the empty word and let G be a binary grammar that generates M. We construct the corresponding extension of $\mathbf{L}^*$. Let $N \cup \Sigma \subset \mathrm{Var}$, and let Var contain an infinite number of extra fresh variables that we'll need later. For every production $(w \Rightarrow v_1 v_2) \in P$ we add one rule

$$\frac{\Delta_1 \to v_1 \quad \Delta_2 \to v_2}{\Delta_1, \Delta_2 \to w} \ (\mathrm{E})$$

For productions of the form $v_1 v_2 \Rightarrow w$ the construction is more complex. First for every pair $\mathfrak{p} = \langle (v_1 v_2 \Rightarrow w), x\rangle$, where $(v_1 v_2 \Rightarrow w) \in P$ and $x \in N \cup \Sigma$, we introduce new variables $\widetilde{y}^{\mathfrak{p}}$ for every $y \in N \cup \Sigma$ and five extra variables $\mathbf{a}^{\mathfrak{p}}$, $\mathbf{b}^{\mathfrak{p}}$, $\mathbf{c}^{\mathfrak{p}}$, $\mathbf{e}^{\mathfrak{p}}$, $\mathbf{f}^{\mathfrak{p}}$. Then for every $\mathfrak{p}$ we add the following rules. Some of these rules are not in Buszkowski's form. We'll transform them into the correct format below.

$$\frac{\Delta_1 \to \mathbf{e}^{\mathfrak{p}} \quad \Delta_2 \to x}{\Delta_1, \Delta_2 \to \mathbf{a}^{\mathfrak{p}}} \ (1_{\mathfrak{p}}) \qquad \frac{\Delta_1 \to \widetilde{y}^{\mathfrak{p}} \quad \Delta_2, y \to \mathbf{a}^{\mathfrak{p}}}{\Delta_1, \Delta_2 \to \mathbf{a}^{\mathfrak{p}}} \ (2_{\mathfrak{p}})$$

$$\frac{\Delta_1 \to \widetilde{w}^{\mathfrak{p}} \quad \Delta_2, v_1, v_2 \to \mathbf{a}^{\mathfrak{p}}}{\Delta_1, \Delta_2 \to \mathbf{b}^{\mathfrak{p}}} \ (3_{\mathfrak{p}}) \qquad \frac{\Delta_1 \to \widetilde{y}^{\mathfrak{p}} \quad \Delta_2, y \to \mathbf{b}^{\mathfrak{p}}}{\Delta_1, \Delta_2 \to \mathbf{b}^{\mathfrak{p}}} \ (4_{\mathfrak{p}})$$

$$\frac{\Delta_1 \to \mathbf{f}^{\mathfrak{p}} \quad \Delta_2, \mathbf{e}^{\mathfrak{p}} \to \mathbf{b}^{\mathfrak{p}}}{\Delta_1, \Delta_2 \to \mathbf{c}^{\mathfrak{p}}} \ (5_{\mathfrak{p}}) \qquad \frac{\Delta_1 \to y \quad \Delta_2, \widetilde{y}^{\mathfrak{p}} \to \mathbf{c}^{\mathfrak{p}}}{\Delta_1, \Delta_2 \to \mathbf{c}^{\mathfrak{p}}} \ (6_{\mathfrak{p}})$$

$$\frac{\Delta, \mathbf{f}^{\mathfrak{p}} \to \mathbf{c}^{\mathfrak{p}}}{\Delta \to x} \ (7_{\mathfrak{p}})$$

[1] In the definition from [3], P could also include productions of the form $u \Rightarrow v$ for $u, v \in N \cup \Sigma$. Such a rule can be equivalently replaced by two productions $u \Rightarrow w_1 w_2$, $w_1 w_2 \Rightarrow v$, where w_1 and w_2 are new elements added to N (different for different rules). We encode these simple productions using more complex ones in order to reduce the number of cases to be considered in the proofs.

As already said, some of these rules are not actually Buszkowski's rules. However, any rule of the form

$$\frac{\Delta_1 \to p \quad \Delta_2, q \to r}{\Delta_1, \Delta_2 \to t}$$

(these are rules $(2_\mathfrak{p})$, $(4_\mathfrak{p})$, $(5_\mathfrak{p})$, and $(6_\mathfrak{p})$) can be equivalently replaced by two rules

$$\frac{\Delta, q \to r}{\Delta \to u}\ (\mathrm{B}_2) \qquad \text{and} \qquad \frac{\Delta_1 \to p \quad \Delta_2 \to u}{\Delta_1, \Delta_2 \to t}\ (\mathrm{B}_1)$$

where u is a fresh variable.

Similarly, $(3_\mathfrak{p})$ is a shortcut for three rules:

$$\frac{\Delta, v_2 \to \mathbf{a}^\mathfrak{p}}{\Delta \to u_1}\ (\mathrm{B}_2) \qquad \frac{\Delta, v_1 \to u_1}{\Delta \to u_2}\ (\mathrm{B}_2) \qquad \frac{\Delta_1 \to \widetilde{w}^\mathfrak{p} \quad \Delta_2 \to u_2}{\Delta_1, \Delta_2 \to \mathbf{b}^\mathfrak{p}}\ (\mathrm{B}_1)$$

Rules $(1_\mathfrak{p})$, $(7_\mathfrak{p})$, and (E) are already in the correct format. Thus we've actually constructed a calculus of the form $\mathbf{L}^* + \mathcal{R}$. Denote it by $\mathbf{L}^* + \mathcal{R}_M$.

Now to achieve our goal it is sufficient to prove that for any $x, z_1, \ldots, z_m \in N \cup \Sigma$

$$x \Rightarrow_G^* z_1 \ldots z_m \qquad \text{iff} \qquad z_1, \ldots, z_m \to x \text{ is derivable in } \mathbf{L}^* + \mathcal{R}_M.$$

The proof consists of two directions.

$\boxed{\Leftarrow}$ All types in the sequent $z_1, \ldots, z_m \to x$ are primitive, therefore its derivation includes only axioms and Buszkowski's rules, but not original rules of $\mathbf{L}^*$ $((\to /)$, $(\to \backslash)$, $(/ \to)$, $(\backslash \to))$.

Since $\mathbf{e}^\mathfrak{p}$, $\mathbf{f}^\mathfrak{p}$, and $\widetilde{y}^\mathfrak{p}$ (for all $y \in N \cup \Sigma$, including w) do not appear in the succedents of goal sequents in Buszkowski's rules from $\mathcal{R}_M$, the only possible situation when $\mathbf{e}^\mathfrak{p}$, $\mathbf{f}^\mathfrak{p}$, or $\widetilde{y}^\mathfrak{p}$ actually appears in the succedent is the axiom. Hence rules $(1_\mathfrak{p})$–$(5_\mathfrak{p})$ can be rewritten in a simpler way (rules $(6_\mathfrak{p})$ and $(7_\mathfrak{p})$ are not affected by this simplification):

$$\frac{\Phi \to x}{\mathbf{e}^\mathfrak{p}, \Phi \to \mathbf{a}^\mathfrak{p}}\ (1'_\mathfrak{p}) \qquad \frac{\Phi, y \to \mathbf{a}^\mathfrak{p}}{\widetilde{y}^\mathfrak{p}, \Phi \to \mathbf{a}^\mathfrak{p}}\ (2'_\mathfrak{p}) \qquad \frac{\Phi, v_1, v_2 \to \mathbf{a}^\mathfrak{p}}{\widetilde{w}^\mathfrak{p}, \Phi \to \mathbf{b}^\mathfrak{p}}\ (3'_\mathfrak{p})$$

$$\frac{\Phi, y \to \mathbf{b}^\mathfrak{p}}{\widetilde{y}^\mathfrak{p}, \Phi \to \mathbf{b}^\mathfrak{p}}\ (4'_\mathfrak{p}) \qquad \frac{\Phi, \mathbf{e}^\mathfrak{p} \to \mathbf{b}^\mathfrak{p}}{\mathbf{f}^\mathfrak{p}, \Phi \to \mathbf{c}^\mathfrak{p}}\ (5'_\mathfrak{p}) \qquad \frac{\Delta_1 \to y \quad \Delta_2, \widetilde{y}^\mathfrak{p} \to \mathbf{c}^\mathfrak{p}}{\Delta_1, \Delta_2 \to \mathbf{c}^\mathfrak{p}}\ (6_\mathfrak{p})$$

$$\frac{\Phi, \mathbf{f}^\mathfrak{p} \to \mathbf{c}^\mathfrak{p}}{\Phi \to x}\ (7_\mathfrak{p})$$

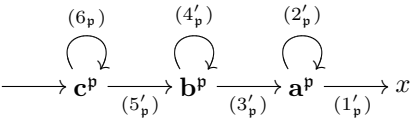
Proceed by induction on the cut-free derivation. The sequent $z_1, \ldots, z_m \to x$ could either be an axiom (and then $n = 1$, $z_1 = x$, and trivially $x \Rightarrow_G^* x$) or be derived by one of the Buszkowski's rules. Since $x \in N \cup \Sigma$, the only possible rules are (E) and $(7_\mathfrak{p})$.

If $z_1, \ldots, z_m \to x$ is derived using (E):

$$\frac{z_1, \ldots, z_k \to v_1 \quad z_{k+1}, \ldots, z_m \to v_2}{z_1, \ldots, z_k, z_{k+1}, \ldots z_m \to x}\ (\mathrm{E}),$$

then we have $z_1, \ldots, z_k \to v_1$, $z_{k+1}, \ldots, z_m \to v_2$, and $(x \Rightarrow v_1 v_2) \in P$. By induction hypothesis, $v_1 \Rightarrow_G^* z_1 \ldots z_k$ and $v_2 \Rightarrow_G^* z_k \ldots z_n$, therefore we get $x \Rightarrow_G v_1 v_2 \Rightarrow_G^* z_1 \ldots z_k z_{k+1} \ldots z_m$.

If the last rule in the derivation is $(7_\mathfrak{p})$, then we get $z_1, \ldots, z_m, \mathbf{f}^\mathfrak{p} \to \mathbf{c}^\mathfrak{p}$. Trace the type in the succedent. Since the antecedent doesn't contain $\mathbf{c}^\mathfrak{p}$, $\mathbf{b}^\mathfrak{p}$, or $\mathbf{a}^\mathfrak{p}$, sequents with these types in the succedent could not appear as axioms, and the only ways to derive such sequents are represented by the following schema (the arrows go from goal to premises):



Therefore, the sequent $z_1, \ldots, z_m, \mathbf{f}^\mathfrak{p} \to \mathbf{c}^\mathfrak{p}$ is derived in the following way: several (possibly zero) applications of $(6_\mathfrak{p})$, then $(5'_\mathfrak{p})$, then several $(4'_\mathfrak{p})$, then several $(2'_\mathfrak{p})$, then $(1'_\mathfrak{p})$. Finally, on top of this last $(1'_\mathfrak{p})$ rule we again get a sequent with x in the succedent. The whole derivation has the following form. Here * means several consecutive applications of the same rule, and $\Delta_1, \ldots, \Delta_n = z_1, \ldots, z_m$.

$$\dfrac{\Delta_1 \to y_1 \quad \ldots \quad \Delta_k \to w \quad \ldots \quad \Delta_n \to y_n \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{y_1, \ldots, y_{k-1}, v_1, v_2, y_{k+1}, \ldots, y_n \to x}{\mathbf{e}^\mathfrak{p}, y_1, \ldots, y_{k-1}, v_1, v_2, y_{k+1}, \ldots, y_n \to \mathbf{a}^\mathfrak{p}}\,(1'_\mathfrak{p})}{\widetilde{y}^\mathfrak{p}_{k+1}, \ldots, \widetilde{y}^\mathfrak{p}_n, \mathbf{e}^\mathfrak{p}, y_1, \ldots, y_{k-1}, v_1, v_2 \to \mathbf{a}^\mathfrak{p}}\,(2'_\mathfrak{p})^*}{\widetilde{w}^\mathfrak{p}, \widetilde{y}^\mathfrak{p}_{k+1}, \ldots, \widetilde{y}^\mathfrak{p}_n, \mathbf{e}^\mathfrak{p}, y_1, \ldots, y_{k-1} \to \mathbf{b}^\mathfrak{p}}\,(3'_\mathfrak{p})}{\widetilde{y}^\mathfrak{p}_1, \ldots, \widetilde{y}^\mathfrak{p}_{k-1}, \widetilde{w}^\mathfrak{p}, \widetilde{y}^\mathfrak{p}_{k+1}, \ldots, \widetilde{y}^\mathfrak{p}_n, \mathbf{e}^\mathfrak{p} \to \mathbf{b}^\mathfrak{p}}\,(4'_\mathfrak{p})^*}{\mathbf{f}^\mathfrak{p}, \widetilde{y}^\mathfrak{p}_1, \ldots, \widetilde{y}^\mathfrak{p}_{k-1}, \widetilde{w}^\mathfrak{p}, \widetilde{y}^\mathfrak{p}_{k+1}, \ldots, \widetilde{y}^\mathfrak{p}_n \to \mathbf{c}^\mathfrak{p}}\,(5'_\mathfrak{p})}\,(6_\mathfrak{p})^*}{\dfrac{\Delta_1, \ldots, \Delta_n, \mathbf{f}^\mathfrak{p} \to \mathbf{c}^\mathfrak{p}}{\Delta_1, \ldots, \Delta_n \to x}\,(7_\mathfrak{p})}$$

Here rule $(1'_\mathfrak{p})$ introduces $\mathbf{e}^\mathfrak{p}$, $(2'_\mathfrak{p})$ moves $\mathbf{e}^\mathfrak{p}$ to the left and marks y_i as $\widetilde{y}^\mathfrak{p}_i$, $(3'_\mathfrak{p})$ actually applies the production $(v_1 v_2 \Rightarrow w)$, which is possible, since v_1, v_2 is now on the edge of the antecedent, $(4'_\mathfrak{p})$ continues the movement, and finally $(5'_\mathfrak{p})$, $(6_\mathfrak{p})$, and $(7_\mathfrak{p})$ move the letters backwards, unmark them and return the antecedent to x.

By induction hypothesis, $y_1 \Rightarrow_G^* \Delta_1$, …, $y_{k-1} \Rightarrow_G^* \Delta_{k-1}$, $w \Rightarrow_G^* \Delta_k$, $y_{k+1} \Rightarrow_G^* \Delta_{k+1}$, …, $y_n \Rightarrow_G^* \Delta_n$, and $x \Rightarrow_G^* y_1 \ldots y_{k-1} v_1 v_2 y_{k+1} \ldots y_n$. By application of $v_1 v_2 \Rightarrow w$ we get $x \Rightarrow_G^* \Delta_1 \ldots \Delta_n$.

We notice that the first type of productions of the binary grammar is handled much easier than the second one. This is due to the fact that in the first case we simulate standard context-free derivation, while in the second case the production is not context-free and even not context-sensitive.

$\boxed{\Rightarrow}$ Proceed by induction on $\Rightarrow_G^*$. For the base case $(x \Rightarrow_G^* x)$ the corresponding sequent $(x \to x)$ is an axiom.

If the last production is $w \Rightarrow v_1 v_2$:

$$x \Rightarrow_G^* z_1 \ldots z_{k-1} w z_{k+1} \ldots z_m \Rightarrow_G z_1 \ldots z_{k-1} v_1 v_2 z_{k+1} \ldots z_m,$$

then by induction hypothesis $z_1, \ldots, z_{k-1}, w, z_{k+1}, \ldots, z_m \to x$ is derivable in $\mathbf{L}^* + \mathcal{R}_M$. Also $v_1, v_2 \to w$ is derivable by (B_1), and by (cut) we obtain $z_1, \ldots, z_{k-1}, v_1, v_2, z_{k+1}, \ldots, z_m \to x$. The cut rule is admissible in $\mathbf{L}^* + \mathcal{R}$ by Theorem 2.

For the $v_1 v_2 \Rightarrow w$ case, *i.e.,* the last production is applied like this:

$$x \Rightarrow_G^* z_1 \ldots z_{k-1} v_1 v_2 z_{k+1} \ldots z_m \Rightarrow_G z_1 \ldots z_{k-1} w z_{k+1} \ldots z_m,$$

the derivation is as follows (here $\mathfrak{p} = \langle (v_1 v_2 \Rightarrow w), x \rangle$):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{z_1, \ldots, z_{k-1}, v_1, v_2, z_{k+1}, \ldots, z_m \to x}{\mathbf{e}^{\mathfrak{p}}, z_1, \ldots, z_{k-1}, v_1, v_2, z_{k+1}, \ldots, z_m \to \mathbf{a}^{\mathfrak{p}}}\,(1'_{\mathfrak{p}})}{\widetilde{z}^{\mathfrak{p}}_{k+1}, \ldots, \widetilde{z}^{\mathfrak{p}}_m, \mathbf{e}^{\mathfrak{p}}, z_1, \ldots, z_{k-1}, v_1, v_2 \to \mathbf{a}^{\mathfrak{p}}}\,(2'_{\mathfrak{p}})^*}{\widetilde{w}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_{k+1}, \ldots, \widetilde{z}^{\mathfrak{p}}_m, \mathbf{e}^{\mathfrak{p}}, z_1, \ldots, z_{k-1} \to \mathbf{b}^{\mathfrak{p}}}\,(3'_{\mathfrak{p}})}{\widetilde{z}^{\mathfrak{p}}_1, \ldots, \widetilde{z}^{\mathfrak{p}}_{k-1}, \widetilde{w}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_{k+1}, \ldots, \widetilde{z}^{\mathfrak{p}}_m, \mathbf{e}^{\mathfrak{p}} \to \mathbf{b}^{\mathfrak{p}}}\,(4'_{\mathfrak{p}})^*}{\mathbf{f}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_1, \ldots, \widetilde{z}^{\mathfrak{p}}_{k-1}, \widetilde{w}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_{k+1}, \ldots, \widetilde{z}^{\mathfrak{p}}_m \to \mathbf{c}^{\mathfrak{p}}}\,(5'_{\mathfrak{p}})}{\dfrac{z_1 \to z_1 \quad \ldots \quad w \to w \quad \ldots \quad z_m \to z_m \qquad \mathbf{f}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_1, \ldots, \widetilde{z}^{\mathfrak{p}}_{k-1}, \widetilde{w}^{\mathfrak{p}}, \widetilde{z}^{\mathfrak{p}}_{k+1}, \ldots, \widetilde{z}^{\mathfrak{p}}_m \to \mathbf{c}^{\mathfrak{p}}}{z_1, \ldots, z_{k-1}, w, z_{k+1}, \ldots, z_m, \mathbf{f}^{\mathfrak{p}} \to \mathbf{c}^{\mathfrak{p}}}\,(6_{\mathfrak{p}})^*}{z_1, \ldots, z_{k-1}, w, z_{k+1}, \ldots, z_m \to x}\,(7_{\mathfrak{p}})$$

The sequent on the top is derivable by inductive hypothesis. □

Since there exist undecidable recursively enumerable languages, Theorem 3 now yields the following result:

Theorem 4. *There exists a finite set of Buszkowski's rules $\mathcal{R}_0$ such that the derivability problem for $\mathbf{L}^* + \mathcal{R}_0$ is undecidable.*

Note that the $\backslash$ connective is not used in the construction, so we've actually obtained undecidability for $\mathbf{L}^*_{/} + \mathcal{R}_0$.

4 Undecidability of !L*

We prove undecidability of $\mathbf{!L}^*$ by encoding $\mathbf{L}^* + \mathcal{R}$ derivations in this calculus. In order to do that, we first prove a technical proposition.

If $\mathcal{R}$ is a set of Buszkowski's rules, let

$$\mathcal{G}_{\mathcal{R}} = \left\{ (r \,/\, q) \,/\, p \mid \frac{\Pi_1 \to p \quad \Pi_2 \to q}{\Pi_1, \Pi_2 \to r} \in \mathcal{R} \right\} \cup \left\{ r \,/\, (p \,/\, q) \mid \frac{\Pi, q \to p}{\Pi \to r} \in \mathcal{R} \right\}.$$

If $\mathcal{B} = \{B_1, \ldots, B_n\}$ is a finite set of formulae, let $!\Gamma_{\mathcal{B}} = !B_1, \ldots, !B_n$. (The order of the elements in $\mathcal{B}$ doesn't matter, since $!\Gamma_{\mathcal{B}}$ will appear in left-hand sides of $\mathbf{!L}^*$ sequents, and in $\mathbf{!L}^*$ we have the $(\mathrm{perm}_{1,2})$ rules.)

Theorem 5. $\mathbf{L}^* + \mathcal{R} \vdash \Pi \to A$ *if and only if there exists* $\mathcal{B} \subseteq \mathcal{G}_{\mathcal{R}}$ *such that* $\mathbf{!L}^* \vdash !\Gamma_{\mathcal{B}}, \Pi \to A$.

In this theorem a finite *theory* ($\mathcal{R}$) that extends the basic calculus ($\mathbf{L}^*$) gets embedded into the formula (more precisely, the sequent $\Pi \to A$) being derived. In linear logic this is possible with the help of the exponential modality (!). However, our version of ! doesn't enjoy the weakening rule, therefore we cannot always take $\mathcal{B} = \mathcal{G}_{\mathcal{R}}$, as one usually could expect. Generally, with $\mathcal{B} = \mathcal{G}_{\mathcal{R}}$ the "only if" statement is false. For example, $!(r \,/\, (p \,/\, q)), s \to s$ is not derivable in $\mathbf{!L}^*$, but $s \to s$ is indeed derivable in $\mathbf{L}^* + \mathcal{R}$ for any $\mathcal{R}$. This happens because this particular Buszkowski's rule, encoded by $r \,/\, (p \,/\, q)$, is not *relevant* to $s \to s$.

Proof. $\boxed{\Rightarrow}$ Proceed by induction.

If $\Pi \to A$ is an axiom ($A \to A$), just take $\mathcal{B} = \varnothing$.

If $A = B \,/\, C$, and $\Pi \to A$ is obtained using the $(\to /)$ rule from $\Pi, C \to B$, then take the same $\mathcal{B}$ and apply the same rule:

$$\frac{!\Gamma_{\mathcal{B}}, \Pi, C \to B}{!\Gamma_{\mathcal{B}}, \Pi \to B \,/\, C}$$

If $\Pi = \Phi_1, B \,/\, C, \Psi, \Phi_2$, and $\Pi \to A$ is obtained by $(/ \to)$ from $\Psi \to C$ and $\Phi_1, B, \Phi_2 \to A$, then by induction hypothesis $\mathbf{!L}^* \vdash !\Gamma_{\mathcal{B}_1}, \Psi \to C$ and $\mathbf{!L}^* \vdash !\Gamma_{\mathcal{B}_2}, \Phi_1, B, \Phi_2 \to A$ for some $\mathcal{B}_1, \mathcal{B}_2 \subseteq \mathcal{G}_{\mathcal{A}}$. Let $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$. Then for $!\Gamma_{\mathcal{B}}, \Pi \to A$ we have the following derivation in $\mathbf{!L}^*$, where * means several applications of the rules in any order.

$$\dfrac{\dfrac{!\Gamma_{\mathcal{B}_1}, \Psi \to C \quad !\Gamma_{\mathcal{B}_2}, \Phi_1, B, \Phi_2 \to A}{!\Gamma_{\mathcal{B}_2}, \Phi_1, B \,/\, C, !\Gamma_{\mathcal{B}_1}, \Psi, \Phi_2 \to A}\ (/ \to)}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2}, \Phi_1, B \,/\, C, \Psi, \Phi_2 \to A}\ (\mathrm{contr}, \mathrm{perm})^*$$

Finally, $\Pi \to A$ can be obtained by application of Buszkowski's rules (B_1) or (B_2). In the first case, $A = r$, $\Pi = \Pi_1, \Pi_2$, and both $\Pi_1 \to p$ and $\Pi_2 \to q$ are derivable in $\mathbf{L}^* + \mathcal{R}$. Thus by induction hypothesis we get $\mathbf{!L}^* \vdash !\Gamma_{\mathcal{B}_1}, \Pi_1 \to p$ and $\mathbf{!L}^* \vdash !\Gamma_{\mathcal{B}_2}, \Pi_2 \to q$ for some $\mathcal{B}_1, \mathcal{B}_2 \subseteq \mathcal{G}_{\mathcal{R}}$. Moreover, $(r \,/\, q) \,/\, p \in \mathcal{G}_{\mathcal{R}}$. Now take $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \{(r \,/\, q) \,/\, p\}$ and enjoy the following derivation for $!\Gamma_{\mathcal{B}}, \Pi_1, \Pi_2 \to r$ in $\mathbf{!L}^*$:

$$\dfrac{\dfrac{\dfrac{!\Gamma_{\mathcal{B}_1}, \Pi_1 \to p \qquad \dfrac{!\Gamma_{\mathcal{B}_2}, \Pi_2 \to q \quad r \to r}{r \,/\, q, !\Gamma_{\mathcal{B}_2}, \Pi_2 \to r}\ (/ \to)}{(r \,/\, q) \,/\, p, !\Gamma_{\mathcal{B}_1}, \Pi_1, !\Gamma_{\mathcal{B}_2}, \Pi_2 \to r}\ (/ \to)}{!((r \,/\, q) \,/\, p), !\Gamma_{\mathcal{B}_1}, \Pi_1, !\Gamma_{\mathcal{B}_2}, \Pi_2 \to r}\ (! \to)}{!\Gamma_{\mathcal{B}}, \Pi_1, \Pi_2 \to r}\ (\mathrm{contr}, \mathrm{perm})^*$$

In the (B_2) case, $A = r$, and we have $!\Gamma_{\mathcal{B}'}, \Pi, q \to p$ in the induction hypothesis for some $\mathcal{B}' \subseteq \mathcal{G}_{\mathcal{R}}$. Let $\mathcal{B} = \mathcal{B}' \cup \{r \,/\, (p \,/\, q)\}$ (recall that $r \,/\, (p \,/\, q) \in \mathcal{G}_{\mathcal{R}}$), and proceed like this:

$$\dfrac{\dfrac{\dfrac{\dfrac{!\Gamma_{\mathcal{B}'}, \Pi, q \to p}{!\Gamma_{\mathcal{B}'}, \Pi \to p \,/\, q}\ (\to /) \qquad r \to r}{r \,/\, (p \,/\, q), !\Gamma_{\mathcal{B}'}, \Pi \to r}\ (/ \to)}{!(r \,/\, (p \,/\, q)), !\Gamma_{\mathcal{B}'}, \Pi \to r}\ (! \to)}{!\Gamma_{\mathcal{B}}, \Pi \to r}\ (\mathrm{contr}, \mathrm{perm})^*$$

$\boxed{\Leftarrow}$ Recall that if $\mathcal{B} = \{B_1, \dots, B_n\}$ is a finite set of formulae, then $!\Gamma_{\mathcal{B}} = !B_1, \dots, !B_n$ (as stated above, the order of the elements in $\mathcal{B}$ doesn't matter due to the ($\mathrm{perm}_{1,2}$) rules). For deriving sequents of the form $!\Gamma_{\mathcal{B}}, \Pi \to C$, where Π, C, and $\mathcal{B}$ do not contain ! and \, one can use a simpler calculus than $\mathbf{!L}^*$:

$$\frac{}{p \to p} \qquad \frac{!\Gamma_{\mathcal{B}}, \Pi, B \to A}{!\Gamma_{\mathcal{B}}, \Pi \to A \,/\, B}\ (\to /)$$

$$\frac{!\Gamma_{\mathcal{B}_1}, \Pi \to B \quad !\Gamma_{\mathcal{B}_2}, \Delta_1, A, \Delta_2 \to C}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2}, \Delta_1, A \,/\, B, \Pi, \Delta_2 \to C}\ (/ \to) \qquad \frac{!\Gamma_{\mathcal{B}}, \Delta_1, A, \Delta_2 \to C}{!\Gamma_{\mathcal{B} \cup \{A\}}, \Delta_1, \Delta_2 \to C}\ (! \to)$$

Moreover, the $(! \to)$ rule is interchangeable with the others in the following ways:

$$\frac{\dfrac{!\Gamma_{\mathcal{B}}, \Delta_1, C, \Delta_2, B \to A}{!\Gamma_{\mathcal{B}}, \Delta_1, C, \Delta_2 \to A \,/\, B}\ (\to /)}{!\Gamma_{\mathcal{B} \cup \{C\}}, \Delta_1, \Delta_2 \to A \,/\, B}\ (! \to) \quad \rightsquigarrow \quad \frac{\dfrac{!\Gamma_{\mathcal{B}}, \Delta_1, C, \Delta_2, B \to A}{!\Gamma_{\mathcal{B} \cup \{C\}}, \Delta_1, \Delta_2, B \to A}\ (! \to)}{!\Gamma_{\mathcal{B} \cup \{C\}}, \Delta_1, \Delta_2 \to A \,/\, B}\ (\to /)$$

$$\frac{\dfrac{!\Gamma_{\mathcal{B}_1}, \Pi \to B \quad !\Gamma_{\mathcal{B}_2}, \Delta_1, A, \Delta_2', D, \Delta_2'' \to C}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2}, \Delta_1, A \,/\, B, \Pi, \Delta_2', D, \Delta_2'' \to C}\ (/ \to)}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2 \cup \{D\}}, \Delta_1, A \,/\, B, \Pi, \Delta_2', \Delta_2'' \to C}\ (! \to)$$

$$\wr$$

$$\frac{!\Gamma_{\mathcal{B}_1}, \Pi \to B \quad \dfrac{!\Gamma_{\mathcal{B}_2}, \Delta_1, A, \Delta_2', D, \Delta_2'' \to C}{!\Gamma_{\mathcal{B}_2 \cup \{D\}}, \Delta_1, A, \Delta_2', \Delta_2'' \to C}\ (! \to)}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2 \cup \{D\}}, \Delta_1, A \,/\, B, \Pi, \Delta_2', \Delta_2'' \to C}\ (/ \to)$$

And the same, if D appears inside Δ_1 or Π. Finally, consecutive applications of $(! \to)$ are always interchangeable.

After applying these transformations, we achieve a derivation where $(! \to)$ is applied immediately after applying $(/ \to)$ with the same active type (the other case, when it is applied after the axiom to p, is impossible, since $\mathcal{B}$ is always a subset of $\mathcal{G}_{\mathcal{R}}$, and the latter doesn't contain sole variables). In other words, applications of $(! \to)$ appear only in the following two situations:

$$\frac{\dfrac{!\Gamma_{\mathcal{B}_1}, \Pi \to p \quad !\Gamma_{\mathcal{B}_2}, \Delta_1, r \,/\, q, \Delta_2 \to A}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2}, \Delta_1, (r \,/\, q) \,/\, p, \Pi, \Delta_2 \to A}\ (/ \to)}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2 \cup \{(r \,/\, q) \,/\, p)\}}, \Delta_1, \Pi, \Delta_2 \to A}\ (! \to)$$

and

$$\frac{\dfrac{!\Gamma_{\mathcal{B}_1}, \Pi \to p \,/\, q \quad !\Gamma_{\mathcal{B}_2}, \Delta_1, r, \Delta_2 \to A}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2}, \Delta_1, r \,/ (p \,/\, q), \Pi, \Delta_2 \to A}\ (/ \to)}{!\Gamma_{\mathcal{B}_1 \cup \mathcal{B}_2 \cup \{r \,/ (p \,/\, q)\}}, \Delta_1, \Pi, \Delta_2 \to A}\ (! \to)$$

Now we prove the statement $\mathbf{!L}^* \vdash {!}\Gamma_{\mathcal{B}}, \Pi \to A$ (where $\mathcal{B} \subseteq \mathcal{G}_{\mathcal{R}}$) $\Rightarrow \mathbf{L}^* + \mathcal{R} \vdash \Pi \to A$ by induction on the above canonical derivation. For the case of axiom or applications of rules $(\to /)$ and $(/ \to)$ we just apply the same rules in $\mathbf{L}^* + \mathcal{R}$, so the only interesting case is $(! \to)$. Consider the two possible situations.

In the $(r / q) / p$ case, by induction hypothesis we get $\mathbf{L}^* + \mathcal{R} \vdash \Pi \to p$ and $\mathbf{L}^* + \mathcal{R} \vdash \Delta_1, r / q, \Delta_2 \to A$, and then we develop the following derivation in $\mathbf{L}^* + \mathcal{R}$ (recall that (cut) is admissible there):

$$\dfrac{\Pi \to p \qquad \dfrac{\dfrac{p, q \to r}{p \to r / q}\,(\to /) \qquad \Delta_1, r / q, \Delta_2 \to A}{\Delta_1, p, \Delta_2 \to A}\,(\text{cut})}{\Delta_1, \Pi, \Delta_2 \to A}\,(\text{cut})$$

In the case of $r /(p / q)$, the derivation looks like this:

$$\dfrac{\Pi \to p / q \qquad \dfrac{p / q \to r \quad \Delta_1, r, \Delta_2 \to A}{\Delta_1, p / q, \Delta_2 \to A}\,(\text{cut})}{\Delta_1, \Pi, \Delta_2 \to A}\,(\text{cut})$$

This completes the proof of Theorem 5. □

Now we can return to our main claim.

Proof (of Theorem 1). Take $\mathcal{R}_0$ from Theorem 4 and suppose that $\mathbf{!L}^*$ is decidable. Then we can present an algorithm that solves the derivability problem for $\mathbf{L}^* + \mathcal{R}_0$. Namely, for a sequent $\Pi \to A$ we search through all subsets $\mathcal{B} \subseteq \mathcal{G}_{\mathcal{R}}$ (and there is a finite number of them) and test derivability of ${!}\Gamma_{\mathcal{B}}, \Pi \to A$ in $\mathbf{!L}^*$. By Theorem 5, $\Pi \to A$ is derivable in $\mathbf{L}^* + \mathcal{R}_0$ if and only if at least one of these tests succeeds. This contradicts Theorem 4. Therefore $\mathbf{!L}^*$ is undecidable.

Since we never used $\backslash$ in the construction, we get undecidability for $\mathbf{!L}^*_{/}$. □

This proof of Theorem 1 is in the spirit of our previous work [7]. The significant difference between this paper and [7] is that here the modality does not satisfy the weakening rule and the system $\mathbf{!L}^*$ doesn't obey any version of Lambek's restriction (*i.e.,* the antecedents are allowed to be empty). Due to the lack of the weakening rule, in Theorem 5 it is not sufficient to check derivability only for $\mathcal{B} = \mathcal{G}_{\mathcal{R}}$, and therefore Theorem 5 is formulated in the relevant logic style. We also had to open up and reassemble Buszkowski's proof from [2,3] to make it work without Lambek's restriction (in $\mathbf{L}^*$).

5 A Decidable Fragment of !L*

Undecidability of $\mathbf{!L}^*$ is somewhat unfortunate, because this calculus is liguistically motivated (see Sect. 2). However, in our examples ! was applied only to primitive types (np). If we consider only sequents with this restriction, the

situation is different: the derivability problem becomes decidable. Moreover, it belongs to NP.

Let's call the *size* of a formula A (denoted by $|A|$) the total number of variable and connective occurrences in A. More formally, $|A|$ is defined recursively: $|p| = 1$ for $p \in \mathrm{Var}$, $|A \setminus B| = |B \,/\, A| = |A|+|B|+1$, $|!A| = |A|+1$. The size of a sequent $A_1, \ldots, A_n \to B$ is $|A_1| + \ldots + |A_n| + |B|$.

In the pure Lambek calculus, the size of any derivation is necessarily bounded by the size of the goal sequent. In our case, a sequent could have derivations of arbitrary size due to uncontrolled application of permutation rules: two consecutive applications of (perm_1) and (perm_2) (with the same formula at the same places) do nothing with the sequent, but increase the derivation size. Nevertheless, the following lemma shows that every sequent has a derivation of quadratic size.

Lemma 1. *If the sequent $\Pi \to C$ is derivable in $\mathbf{!L}^*$ and ! in this sequent is applied only to variables, then this sequent has a derivation of size less than $12n^2 + 3n$, where n is the size of $\Pi \to C$.*

Recall that (cut) is not included in the system, all derivations are cut-free.

Proof. We represent the derivation of $\Pi \to C$ as a tree. The leaves of the tree are instances of axioms, and the inner nodes correspond to applications of rules. Rules $(/ \to)$ and $(\setminus \to)$ form *branching points* of the tree. The number of leaves is equal to the number of branching points plus one.

Let's call $(\mathrm{perm}_{1,2})$ and (contr) *structural* rules; other rules are *logical* ones.

Each logical rule introduces exactly one connective into the goal sequent $\Pi \to C$. The key note here is the fact that, since only variables can appear under !, the contraction rule (contr) cannot merge two connectives. Therefore, since the total number of connectives is less than n, the number of logical rule applications is also less than n.

Each branching point corresponds to an application of a logical rule, whence the number of branching points is also less than n. Therefore, in the tree there are no more than n axiom leaves, and each axiom introduces two variable occurrences. Let's trace these occurrences down the tree. Each occurrence either traces to an occurrence in the goal sequent, or disappears (gets merged with another occurrence) in an application of (contr). Thus, the number of (contr) applications is less than the total number of variable occurrences in axiom leaves, and, therefore, less then $2n$.

Finally, we limit the number of $(\mathrm{perm}_{1,2})$ applications. As said above, a block of consecutive applications of $(\mathrm{perm}_{1,2})$ can include an arbitrarily large number of $(\mathrm{perm}_{1,2})$ applications. However, we can always reduce it. Each block of consecutive permutations has the following form:

$$\frac{\Delta_1, !A_1, \Delta_2, !A_2, \Delta_3, \ldots, \Delta_k, !A_k, \Delta_{k+1} \to B}{\Delta'_1, !A_{i_1}, \Delta'_2, !A_{i_2}, \Delta'_3, \ldots, \Delta'_k, !A_{i_k}, \Delta'_{k+1} \to B}\ (\mathrm{perm}_{1,2})^*,$$

where the sequences $\Delta_1, \ldots, \Delta_{k+1}$ and $\Delta'_1, \ldots, \Delta'_{k+1}$ coincide and $\{i_1, \ldots, i_k\} = \{1, \ldots, k\}$.

The number of formulae in the left-hand side of the sequent here is bounded by $3n$ (it was less than n in the goal sequent $\Pi \to C$, and, in the worst case, it was increased by less than $2n$ applications of (contr)). Therefore, $k < 3n$. Now we replace this block with a block of k permutations: each $!A_i$ is moved to its place by one permutation. Thus, in each block we have less than $3n$ permutations.

Each $(\mathrm{perm}_{1,2})$ block is preceded by an application of a rule different from $(\mathrm{perm}_{1,2})$ or an axiom leaf. Thus the number of such blocks is bounded by $4n$ (n for logical rules, $2n$ for contractions, n for axioms).

Therefore, the number of $(\mathrm{perm}_{1,2})$ applications is less than $12n^2$, and the total size of the derivation is less than $12n^2 + 3n$. □

This lemma yields the following decidability result:

Theorem 6. *The derivability problem in* $\mathbf{!L}^*$ *for sequents in which* ! *is applied only to variables is decidable and belongs to the* NP *class (i.e., can be resolved by a nondeterministic polynomial algorithm).*

6 Future Work

Since the Lambek calculus itself is NP-complete [14,17], we get NP-completeness of $\mathbf{!L}^*$ in the restricted case, where ! can be applied only to variables. On the other hand, it is known that the derivability problem for the fragment of the pure Lambek calculus with only one division operation is decidable in polynomial time [15,16]. Therefore, the complexity for the restricted case of $\mathbf{!L}^*_{/}$ (where we have only one division, and ! can be applied only to variables) yet should be studied. It belongs to NP (by our Theorem 6), and the question is whether this fragment is poly-time decidable or NP-hard. Recall that in the unrestricted case we've proved undecidability not only for the whole $\mathbf{!L}^*$, but also for its one-division fragment, $\mathbf{!L}^*_{/}$.

Another interesting question is whether our decidability result (Theorem 6) can be extended to the situation where ! can be applied to formulae of Horn depth 1, *i.e.,* formulae, in which all denominators of / and \ are primitive types, for instance, $(p \setminus (q / r)) / s$. Notice that if we allow formulae of Horn depth 2 (of the form $r / (p / q)$) under !, then we immediately get undecidability (see Sect. 4).

Our encoding in Theorem 5 actually shows that *grammars* based on $\mathbf{!L}^*$ can generate arbitrary recursively enumerable languages. On the other hand, pure Lambek grammars generate precisely context-free languages [13]. Moreover, this holds also in the so-called strong sense, *i.e.,* context-free grammars and Lambek grammars can assign the same Montague-style semantic values to the words derived [6,8]. The question is what class of grammars in the Chomsky hierarchy corresponds to grammars based on the fragment of $\mathbf{!L}^*$, restricted as in Theorem 6, and could one add Montague-style semantics to such grammars.

Acknowledgements. Stepan Kuznetsov's research was supported by the Russian Foundation for Basic Research (grants 15-01-09218-a and 14-01-00127-a) and by the

Presidential Council for Support of Leading Scientific Schools (grant NŠ-9091.2016.1). Max Kanovich's research was partially supported by EPSRC. Andre Scedrov's research was partially supported by ONR.

This research was performed in part during visits of Stepan Kuznetsov and Max Kanovich to the University of Pennsylvania. We greatly appreciate support of the Mathematics Department of the University. A part of the work was also done during the stay of Andre Scedrov at the National Research University Higher School of Economics. We would like to thank S.O. Kuznetsov and I.A. Makarov for hosting there.

The paper was prepared in part within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) and was partially supported within the framework of a subsidy by the Russian Academic Excellence Project '5–100'.

We are indepted to the participants of the research seminars "Logical Problems in Computer Science" and "Algorithmic Problems in Algebra and Logic" at Moscow (Lomonosov) University, in particular, S.I. Adian, L.D. Beklemishev, V.N. Krupski, I.I. Osipov, F.N. Pakhomov, M.R. Pentus, D.S. Shamkanov, I.B. Shapirovsky, V.B. Shehtman, A.A. Sorokin, T.L. Yavorskaya, and others for fruitful discussions and suggestions that allowed us to improve our presentation significantly.

References

1. Abrusci, V.M.: A comparison between Lambek syntactic calculus and intuitionistic linear propositional logic. Zeitschr. für math. Logik und Grundl. der Math. (Math. Logic Q.) **36**, 11–15 (1990)
2. Buszkowski, W.: Some decision problems in the theory of syntactic categories. Zeitschr. für math. Logik und Grundl. der Math. (Math. Logic Q.) **28**, 539–548 (1982)
3. Buszkowski, W.: Lambek calculus with nonlogical axioms. In: Language and Grammar. CSLI Lecture Notes, vol. 168, pp. 77–93 (2005)
4. Carpenter, B.: Type-Logical Semantics. MIT Press, Cambridge (1998)
5. Girard, J.-Y.: Linear logic. Theoret. Comput. Sci. **50**(1), 1–102 (1987)
6. Kanazawa, M., Salvati, S.: The string-meaning relations definable by Lambek grammars and context-free grammars. In: Morrill, G., Nederhof, M.-J. (eds.) Formal Grammar 2012 and 2013. LNCS, vol. 8036, pp. 191–208. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39998-5_12
7. Kanovich, M., Kuznetsov, S., Scedrov, A.: On Lambek's restriction in the presence of exponential modalities. In: Artemov, S., Nerode, A. (eds.) LFCS 2016. LNCS, vol. 9537, pp. 146–158. Springer, Heidelberg (2016). doi:10.1007/978-3-319-27683-0_11
8. Kuznetsov, S.L.: On translating context-free grammars into Lambek grammars. Proc. Steklov Inst. Math. **290**, 63–69 (2015)
9. Lambek, J.: The mathematics of sentence structure. Am. Math. Mon. **65**(3), 154–170 (1958)
10. Lambek, J.: On the calculus of syntactic types. In: Proceedings of Symposia in Applied Mathematics: Structure of Language and its Mathematical Aspects, vol. 12, pp. 166–178. AMS (1961)
11. Lincoln, P., Mitchell, J., Scedrov, A., Shankar, N.: Decision problems for propositional linear logic. Ann. Pure Appl. Logic **56**(1–3), 239–311 (1992)

12. Morrill, G., Valentín, O.: Computational coverage of TLG: nonlinearity. In: Proceedings of NLCS 2015. EPiC Series, vol. 32, pp. 51–63 (2015)
13. Pentus, M.: Product-free Lambek calculus and context-free grammars. J. Symbolic Logic **62**(2), 648–660 (1997)
14. Pentus, M.: Lambek calculus is NP-complete. Theor. Comput. Sci. **357**, 186–201 (2006)
15. Pentus, M.: Complexity of the Lambek calculus and its fragments. Adv. Modal Logic **8**, 310–329 (2010). College Publications
16. Savateev, Y.: Lambek grammars with one division are decidable in polynomial time. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 273–282. Springer, Heidelberg (2008). doi:10.1007/978-3-540-79709-8_28
17. Savateev, Y.: Product-free Lambek calculus is NP-complete. In: Artemov, S., Nerode, A. (eds.) LFCS 2009. LNCS, vol. 5407, pp. 380–394. Springer, Heidelberg (2008). doi:10.1007/978-3-540-92687-0_26
18. Yetter, D.N.: Quantales and (noncommutative) linear logic. J. Symbolic Logic **55**(1), 41–64 (1990)

Introducing a Calculus of Effects and Handlers for Natural Language Semantics

Jirka Maršík(✉) and Maxime Amblard

LORIA, UMR 7503, Université de Lorraine, CNRS, Inria,
Campus Scientifique, 54506 Vandœuvre-lés-Nancy, France
{jirka.marsik,maxime.amblard}@loria.fr

Abstract. In compositional model-theoretic semantics, researchers assemble truth-conditions or other kinds of denotations using the lambda calculus. It was previously observed [26] that the lambda terms and/or the denotations studied tend to follow the same pattern: they are instances of a monad. In this paper, we present an extension of the simply-typed lambda calculus that exploits this uniformity using the recently discovered technique of effect handlers [22]. We prove that our calculus exhibits some of the key formal properties of the lambda calculus and we use it to construct a modular semantics for a small fragment that involves multiple distinct semantic phenomena.

Keywords: Compositionality · Side effects · Monads · Handlers · Deixis · Conventional implicature

1 Introduction

The prevailing methodology of formal semantics is compositionality in the sense of Frege: denotations of complex phrases are functions of the denotations of their immediate constituents. However, several phenomena have been identified that challenge this notion of compositionality. Examples include anaphora, presupposition, quantification, deixis and conventional implicature. In all of these examples, simple models of denotation (i.e. noun phrases are individuals, sentences are truth-values) run into complications as the denotations can depend on external values (anaphora, deixis) or on something which is not an immediate constituent (presupposition, quantification, conventional implicature).

Among the solutions to these challenges, we find (at least) two types of solutions. First, we have those that relax the condition of compositionality. Notably, the denotation of a complex phrase is no longer a *function per se* of the denotations of its immediate subconstituents. Rather, it is some other formally defined process.[1] Examples of this approach include:

[1] This kind of distinction is the same distinction as the one between a mathematical function and a function in a programming language, which might have all kinds of side effects and therefore not be an actual function.

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 257–272, 2016.
DOI: 10.1007/978-3-662-53042-9_15

- the incremental algorithm used to build discourse representation structures in DRT, as presented in [12]
- the $\lambda\mu$ calculus, used in [6] to analyze quantification, since, due to the lack of confluence, function terms do not denote functions over simple denotations
- the use of exceptions and exception handlers in [18] to model presuppositions in an otherwise compositional framework
- the parsetree interpretation step in the logic of conventional implicatures of [23] that builds the denotation of a sentence by extracting implicatures from the denotations of its subparts (including the non-immediate ones)

The other approach is to enrich the denotations so that they are parameterized by the external information they need to obtain and contain whatever internal information they need to provide to their superconstituents. Here are some examples of this style:

- any kind of semantic indices (e.g. the speaker and addressee for deixis, the current world for modality), since they amount to saying that a phrase denotes an indexed set of simpler meanings
- the continuized semantics for quantification [1] in which denotations are functions of their own continuations
  - and more generally, any semantics using type raising or generalized quantifiers for noun phrase denotations
- the dynamic denotations of [7] that are functions of the common ground and their continuation
- compositional event semantics, such as the one in [25], that shift the denotations of sentences from truth-values to predicates on events

We want to find a common language in which we could express the above techniques. Our inspiration comes from computer science. There, a concept known as *monad* has been used:

- in denotational semantics to give the domain of interpretation for programming languages that involve side effects [21].
- in functional programming to emulate programming with side effects via term-level encodings of effectful programs [29].

These two principal applications of monads align with the two approaches we have seen above. The one where we change our calculus so it no longer defines pure functions (e.g. is non-deterministic, stateful or throws exceptions) and the one where we use a pure calculus to manipulate terms (denotations) that encode some interaction (e.g. dynamicity, continuations or event predication).

Monad is a term from category-theory. Its meaning is relative to a category. For us, this will always be the category whose objects are types and whose arrows are functions between different types. A monad is formed by a functor and a pair of natural transformations that satisfy certain laws. In our case, this means that a monad is some type constructor (the functor part) and some combinators (the natural transformations) that follow some basic laws. To give an example of this,

we can think of the functor $T(\alpha) = (\alpha \to o) \to o$ together with combinators such as the type raising $\eta(x) = \lambda P.P\,x$ as a monad of quantification.

The relationship between side effects in functional programming and computational semantics has been developed in several works [27,28],[2] stretching as far back as 1977 [10]. The usefulness of monads in particular has been discovered by Shan in 2002 [26]. Since then, the problem that remained was how to compose several different monads in a single solution. Charlow used the popular method of monad morphisms[3] to combine several monads in his dissertation [4]. Giorgolo and Asudeh have used distributive laws to combine monads [8], while Kiselyov has eschewed monads altogether in favor of applicative functors which enjoy easy composability [13].

Our approach follows the recent trend in adopting effects and handlers to combine side effects [2,11] and to encode effectful programs in pure functional programming languages [3,14].

The idea is that we can represent each of the relevant monads using an algebra. We can then combine the signatures of the algebras by taking a disjoint union. The free algebra of the resulting signature will serve as a universal representation format for the set of all terms built from any of the source algebras and closed under substitution. Then, we will build modular interpreters that will give meanings to the operators of the algebras in terms of individuals, truth-values and functions.

In Sect. 2, we will introduce a formal calculus for working with the algebraic terms that we will use in our linguistic denotations. In Sect. 3, we will incrementally build up a fragment involving several of the linguistic phenomena and see the calculus in action. Before we conclude in Sect. 5, we will also discuss some of the formal properties of the calculus in Sect. 4.

2 Definition of the Calculus

Our calculus is an extension of the simply-typed lambda calculus (STLC). We add terms of a free algebra into our language and a notation for writing handlers, composable interpreters of these terms. An operator of the free algebra corresponds to a particular interaction that a piece of natural language can have with its context (e.g. a deictic expression might request the speaker's identity using some operator `speaker` in order to find its denotation). A handler gives an interpretation to every occurrence of an operator within a term (e.g. direct speech introduces a handler for the operator `speaker` that essentially rebinds the current speaker to some other entity).

Having sketched the general idea behind our calculus, we will now turn our attention to the specifics. We start by defining the syntactic constructions used to build the terms of our language.

[2] Side effects are to programming languages what pragmatics are to natural languages: they both study how expressions interact with the worlds of their users. It might then come as no surprise that phenomena such as anaphora, presupposition, deixis and conventional implicature yield a monadic description.

[3] Also known as monad transformers in functional programming.

2.1 Terms

First off, let $\mathcal{X}$ be a set of variables, Σ a typed signature and $\mathcal{E}$ a set of operation symbols. In the definition below, we will let M, $N \ldots$ range over terms, $x, y, z \ldots$ range over variables from $\mathcal{X}$, $c, d \ldots$ range over the names of constants from Σ and $\mathtt{op}$, $\mathtt{op}_i \ldots$ range over the operation symbols in $\mathcal{E}$.

The terms of our language are composed of the following:

$$
\begin{aligned}
M, N ::= {} & \lambda x.M && \text{[abstraction]} \\
\mid {} & M\,N && \text{[application]} \\
\mid {} & x && \text{[variable]} \\
\mid {} & c && \text{[constant]} \\
\mid {} & \mathtt{op}\, M_{\mathrm{p}}\, (\lambda x.M_{\mathrm{c}}) && \text{[operation]} \\
\mid {} & \eta\, M && \text{[injection]} \\
\mid {} & (\!|\, \mathtt{op}_1 \colon M_1,\ \ldots,\ \mathtt{op}_n \colon M_n,\ \eta \colon M_\eta \,|\!)\, N && \text{[handler]} \\
\mid {} & \downarrow_{\circ} M && \text{[extraction]} \\
\mid {} & \mathcal{C}\, M && \text{[exchange]}
\end{aligned}
$$

The first four constructions — abstraction, application, variables and constants — come directly from STLC with constants.

The next four deal with the algebraic expressions used to encode computations. Let us sketch the behaviors of these four kinds of expressions.

The operation ($\mathtt{op}$) and injection (η) expressions will serve as the constructors for our algebraic expressions. Algebraic expressions are usually formed by operation symbols and then variables as atoms. Instead of variables, our algebraic expressions use terms from our calculus for atoms. The η constructor can thus take an ordinary term from our calculus and make it an atomic algebraic expression. The operation symbols $\mathtt{op}$ are then the operations of the algebra.

The other three expression types correspond to functions over algebraic expressions.

- The most useful is the handler $(\!|\ |\!)$.[4] It is an iterator for the type of algebraic expressions. The terms $M_1, \ldots, M_n$ and M_η in $(\!|\, \mathtt{op}_1 \colon M_1,\ \ldots,\ \mathtt{op}_n \colon M_n,\ \eta \colon M_\eta \,|\!)$ are the clauses for the constructors $\mathtt{op}_1, \ldots, \mathtt{op}_n$ and η, respectively. We will use handlers to define interpretations of operation symbols in algebraic expressions.
- The cherry $\downarrow_{\circ}$ operator allows us to extract terms out of algebraic expressions. If an algebraic expression is of the form $\eta\, M$, applying $\downarrow_{\circ}$ to it will yield M.
- The exchange operator $\mathcal{C}$ permits a kind of commutation between the λ-binder and the operation symbols. We will see its use later.

[4] Pronounced "banana". See [20] for the introduction of banana brackets.

2.2 Types

We now give a syntax for the types of our calculus along with a typing relation. In the grammar below, α, β, γ... range over types, ν ranges over atomic types from some set $\mathcal{T}$ and E, E'... range over effect signatures (introduced below).

The types of our language consist of:

$$\begin{aligned} \alpha, \beta, \gamma :: &= \alpha \to \beta && \text{[function]} \\ &\mid \nu && \text{[atom]} \\ &\mid \mathcal{F}_E(\alpha) && \text{[computation]} \end{aligned}$$

The only novelty here is the $\mathcal{F}_E(\alpha)$ computation type. This is the type of algebraic expressions whose atoms are terms of type α and whose operation symbols come from the effect signature E. We call them *computation types* and we call terms of these types *computations* because our algebraic expressions will always represent some kind of program with effects.

Effect signatures are similar to typing contexts. They are partial mappings from the set of operation symbols $\mathcal{E}$ to pairs of types. We will write the elements of effect signatures the following way — $\mathtt{op} : \alpha \rightarrowtail \beta \in E$ means that E maps $\mathtt{op}$ to the pair of types α and β.[5] When dealing with effect signatures, we will often make use of the disjoint union operator $\uplus$. The term $E_1 \uplus E_2$ serves as a constraint demanding that the domains of E_1 and E_2 be disjoint and at the same time it denotes the effect signature that is the union of E_1 and E_2.

The typing rules are presented in Fig. 1.

The typing rules mirror the syntax of terms. Again, the first four rules come from STLC. The $[\eta]$ and $[\downarrow]$ rules are self-explanatory and so we will focus on the $[\mathtt{op}]$, $[(\!|\,|\!)]$ and $[\mathcal{C}]$ rules.

$[\mathtt{op}]$ To use an operation $\mathtt{op} : \alpha \rightarrowtail \beta$, we provide the input parameter $M_\mathrm{p} : \alpha$ and a continuation $\lambda x.M_\mathrm{c} : \beta \to \mathcal{F}_E(\gamma)$, which expects the output of type β. The resulting term has the same type as the body of the continuation, $\mathcal{F}_E(\gamma)$.

Before, we have spoken of terms of type $\mathcal{F}_E(\gamma)$ as of algebraic expressions generated by the terms of type γ and the operators in the effect signature E. However, having seen the typing rule for operation terms, it might not be obvious how such a term represents an algebraic expression. Traditionally, algebraic signatures map operation symbols to arities, which are natural numbers. Our effect signatures map each operation symbol to a pair of types $\alpha \rightarrowtail \beta$.

– We can explain α by analogy to the single-sorted algebra of vector spaces. In a single-sorted vector space algebra, scalar multiplication is viewed as a unary operation parameterized by some scalar. So technically, there is a different unary operation for each scalar. All of our operations are similarly parameterized and α is the type of that parameter.

[5] The two types α and β are to be seen as the operation's *input* and *output* types, respectively.

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.\, M : \alpha \to \beta}\ [\text{abs}] \qquad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash M N : \beta}\ [\text{app}]$$

$$\frac{x : \alpha \in \Gamma}{\Gamma \vdash x : \alpha}\ [\text{var}] \qquad \frac{c : \alpha \in \Sigma}{\Gamma \vdash c : \alpha}\ [\text{const}]$$

$$\frac{\Gamma \vdash M : \alpha}{\Gamma \vdash \eta\, M : \mathcal{F}_E(\alpha)}\ [\eta] \qquad \frac{\Gamma \vdash M_\text{p} : \alpha \qquad \Gamma, x : \beta \vdash M_\text{c} : \mathcal{F}_E(\gamma) \qquad \mathtt{op} : \alpha \rightarrowtail \beta \in E}{\Gamma \vdash \mathtt{op}\, M_\text{p}\, (\lambda x.\, M_\text{c}) : \mathcal{F}_E(\gamma)}\ [\text{op}]$$

$$\frac{\Gamma \vdash M : \mathcal{F}_\emptyset(\alpha)}{\Gamma \vdash \mathbin{\downarrow}_\circ M : \alpha}\ [\downarrow_\circ]$$

$$\frac{\begin{array}{c} E = \{\mathtt{op}_i : \alpha_i \rightarrowtail \beta_i\}_{i \in I} \uplus E_\text{f} \\ E' = E'' \uplus E_\text{f} \\ [\Gamma \vdash M_i : \alpha_i \to (\beta_i \to \mathcal{F}_{E'}(\delta)) \to \mathcal{F}_{E'}(\delta)]_{i \in I} \\ \Gamma \vdash M_\eta : \gamma \to \mathcal{F}_{E'}(\delta) \\ \Gamma \vdash N : \mathcal{F}_E(\gamma) \end{array}}{\Gamma \vdash (\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, N : \mathcal{F}_{E'}(\delta)}\ [(\!|\,|\!)]$$

$$\frac{\Gamma \vdash M : \alpha \to \mathcal{F}_E(\beta)}{\Gamma \vdash \mathcal{C}\, M : \mathcal{F}_E(\alpha \to \beta)}\ [\mathcal{C}]$$

Fig. 1. The typing rules for our calculus.

– The type β expresses the arity of the operator. When we say that an operator has arity β, where β is a type, we mean that it takes one operand for every value of β [24]. We can also think of the operator as taking one operand containing $x : \beta$ as a free variable.

We can look at the algebraic expression $\mathtt{op}\, M_\text{p}\, (\lambda x.M_\text{c})$ as a description of a program that:

– interacts with its context by some operator called $\mathtt{op}$
– to which it provides the input M_p
– and from which it expects to receive an output of type β
– which it will then bind as the variable x and continue as the program described by M_c.

$[(\!|\,|\!)]$. The banana brackets describe iterators/catamorphisms.[6] In the typing rule, E is the input's signature, E' is the output's signature, γ is the input's atom type and δ is the output's atom type. E is decomposed into the operations that our iterator will actually interpret, the other operations form a residual signature E_f. The output signature will then still contain the uninterpreted operations E_f combined with any operations E'' that our interpretation might introduce.

[6] These are similar to recursors/paramorphisms. See [20] for the difference. Catamorphisms are also known as folds and the common higher-order function *fold* found in functional programming languages is actually the iterator/catamorphism for lists.

$[\mathcal{C}]$. We said before that the $\mathcal{C}$ function will let us commute λ and operations. Here we see that, on the type level, this corresponds to commuting the $\mathcal{F}_E(\_)$ and the $\alpha \to \_$ type constructors.

2.3 Reduction Rules

We will now finally give a semantics to our calculus. The semantics will be given in the form of a reduction relation on terms. Even though the point of the calculus is to talk about effects, the reduction semantics will not be based on any fixed evaluation order; any subterm that is a redex can be reduced in any context. The reduction rules are given in Fig. 2.

$$
\begin{array}{lr}
(\lambda x.\, M)\, N \to & \text{rule } \beta \\
M[x := N] & \\
 & \\
\lambda x.\, M\, x \to & \text{rule } \eta \\
M & \text{where } x \notin \mathrm{FV}(M) \\
 & \\
(\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, (\eta\, N) \to & \text{rule } (\!|\, \eta \,|\!) \\
M_\eta\, N & \\
 & \\
(\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, (\mathtt{op}_j\, N_\mathrm{p}\, (\lambda x.\, N_\mathrm{c})) \to & \text{rule } (\!|\, \mathtt{op} \,|\!) \\
M_j\, N_\mathrm{p}\, (\lambda x.\, (\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, N_\mathrm{c}) & \text{where } j \in I \\
 & \text{and } x \notin \mathrm{FV}((M_i)_{i \in I}, M_\eta) \\
 & \\
(\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, (\mathtt{op}_j\, N_\mathrm{p}\, (\lambda x.\, N_\mathrm{c})) \to & \text{rule } (\!|\, \mathtt{op}' \,|\!) \\
\mathtt{op}_j\, N_\mathrm{p}\, (\lambda x.\, (\!|\, (\mathtt{op}_i \colon M_i)_{i \in I},\ \eta \colon M_\eta \,|\!)\, N_\mathrm{c}) & \text{where } j \notin I \\
 & \text{and } x \notin \mathrm{FV}((M_i)_{i \in I}, M_\eta) \\
 & \\
\downarrow\!\!\circ\, (\eta\, M) \to & \text{rule } \downarrow\!\!\circ \\
M & \\
 & \\
\mathcal{C}\, (\lambda x.\, \eta\, M) \to & \text{rule } \mathcal{C}_\eta \\
\eta\, (\lambda x.\, M) & \\
 & \\
\mathcal{C}\, (\lambda x.\, \mathtt{op}\, M_\mathrm{p}\, (\lambda y.\, M_\mathrm{c})) \to & \text{rule } \mathcal{C}_{\mathtt{op}} \\
\mathtt{op}\, M_\mathrm{p}\, (\lambda y.\, \mathcal{C}\, (\lambda x.\, M_\mathrm{c})) & \text{where } x \notin \mathrm{FV}(M_\mathrm{p})
\end{array}
$$

Fig. 2. The reduction rules of our calculus.

We have the β and η rules, which, by no coincidence, are the same rules as the ones found in STLC. The rest are function definitions for $(\!|\, |\!)$, $\downarrow\!\!\circ$ and $\mathcal{C}$.

By looking at the definition of $(\!|\, |\!)$, we see that it is an iterator. It replaces every occurrence of the constructors $\mathtt{op}_j$ and η with M_j and M_η, respectively.

The $\mathcal{C}$ function recursively swaps $\mathcal{C}\, (\lambda x.\_)$ with $\mathtt{op}\, M_\mathrm{p}\, (\lambda y.\_)$ using the $\mathcal{C}_{\mathtt{op}}$ rule. When $\mathcal{C}$ finally meets the η constructor, it swaps $(\lambda x.\_)$ with $\eta\, \_$ and terminates.

Note that the constraint $x \notin \mathrm{FV}(M_\mathrm{p})$ in rule $\mathcal{C}_{\mathsf{op}}$ cannot be dismissed by renaming of bound variables. If the parameter M_p contains a free occurrence of x, the evaluation of $\mathcal{C}$ will get stuck. $\mathcal{C}$ is thus a partial function: it is only applicable when none of the operations being commuted with the λ-binder actually depend on the bound variable.

2.4 Common Combinators

When demonstrating the calculus in the next section, the following combinators will be helpful. First, we define a sequencing operator. The operator $\gg\!=$, called bind, replaces all the α-typed atoms of a $\mathcal{F}_E(\alpha)$-typed expression with $\mathcal{F}_E(\beta)$-typed expressions. More intuitively, $M \gg\!= N$ is the program that first runs M to get its result x and then continues as the program $N\,x$.

$$\begin{aligned} &\_ \gg\!= \_ : \mathcal{F}_E(\alpha) \to (\alpha \to \mathcal{F}_E(\beta)) \to \mathcal{F}_E(\beta) \\ &M \gg\!= N = (\!|\, \eta\colon N \,|\!)\, M \end{aligned}$$

The type constructor $\mathcal{F}_E$ along with the operators η and $\gg\!=$ form a free monad. Using this monadic structure, we can define the following combinators (variations on application) which we will make heavy use of in Sect. 3.

$$\begin{aligned} &\_ \ll\cdot \_ : \mathcal{F}_E(\alpha \to \beta) \to \alpha \to \mathcal{F}_E(\beta) \\ &F \ll\cdot x = F \gg\!= (\lambda f.\eta\,(f\,x)) \\ &\_ \cdot\!\gg \_ : (\alpha \to \beta) \to \mathcal{F}_E(\alpha) \to \mathcal{F}_E(\beta) \\ &f \cdot\!\gg X = X \gg\!= (\lambda x.\eta\,(f\,x)) \\ &\_ \ll\!\cdot\!\gg \_ : \mathcal{F}_E(\alpha \to \beta) \to \mathcal{F}_E(\alpha) \to \mathcal{F}_E(\beta) \\ &F \ll\!\cdot\!\gg X = F \gg\!= (\lambda f.X \gg\!= (\lambda x.\eta\,(f\,x))) \end{aligned}$$

All of these operators associate to the left, so $f \cdot\!\gg X \ll\!\cdot\!\gg Y$ should be read as $(f \cdot\!\gg X) \ll\!\cdot\!\gg Y$.

Let $\circ : o \to o \to o$ be a binary operator on propositions. We define the following syntax for the same operator lifted to computations of propositions.

$$\begin{aligned} &\_ \mathbin{\overline{\circ}} \_ : \mathcal{F}_E(o) \to \mathcal{F}_E(o) \to \mathcal{F}_E(o) \\ &M \mathbin{\overline{\circ}} N = (\lambda mn.m \circ n) \cdot\!\gg M \ll\!\cdot\!\gg N \end{aligned}$$

3 Linguistic Phenomena as Effects

3.1 Deixis

We will now try to use this calculus to do some semantics. Here is our tectogrammar in an abstract categorial grammar presentation [5].

$$\begin{aligned} \textsc{John}, \textsc{Mary}, \textsc{me} &: NP \\ \textsc{loves} &: NP \multimap NP \multimap S \end{aligned}$$

And here is our semantics.

$$\begin{aligned}
[\![\textsc{John}]\!] &:= \eta\,\mathbf{j} \\
[\![\textsc{Mary}]\!] &:= \eta\,\mathbf{m} \\
[\![\textsc{me}]\!] &:= \texttt{speaker} \star (\lambda x.\eta\,x) \\
[\![\textsc{loves}]\!] &:= \lambda OS.\mathbf{love} \cdot{\gg} S \ll{\cdot}{\gg} O
\end{aligned}$$

In the semantics for $[\![\textsc{me}]\!]$, we use the `speaker` operation to retrieve the current speaker and make it available as the value of the variable x. The star ($\star$) passed to `speaker` is a dummy value of the unit type 1.

This, and all the semantics we will see in this paper, satisfies a homomorphism condition that whenever $M : \tau$, then $[\![M]\!] : [\![\tau]\!]$. In our case, $[\![NP]\!] = \mathcal{F}_E(\iota)$ and $[\![S]\!] = \mathcal{F}_E(o)$, where ι and o are the types of individuals and propositions, respectively. Of E, we assume that $\texttt{speaker} : 1 \rightarrowtail \iota \in E$, since that is the type of `speaker` used in our semantics.[7]

With this fragment, we can give meanings to trivial sentences like:

(1) John loves Mary.
(2) Mary loves me.

whose meanings we can calculate as:

$$[\![\textsc{loves Mary John}]\!] \twoheadrightarrow \eta\,(\mathbf{love}\,\mathbf{j}\,\mathbf{m}) \tag{1}$$

$$[\![\textsc{loves me Mary}]\!] \twoheadrightarrow \texttt{speaker} \star (\lambda x.\eta\,(\mathbf{love}\,\mathbf{m}\,x)) \tag{2}$$

The meaning of (1) is a proposition of type o wrapped in η, i.e. something that we can interpret in a model. As for the meaning of (2), the `speaker` operator has propagated from the ME lexical entry up to the meaning of the whole sentence. We now have an algebraic expression having as operands the propositions $\mathbf{love}\,\mathbf{m}\,x$ for all possible $x : \iota$. In order to get a single proposition which is to be seen as the truth-conditional meaning of the sentence and which can be evaluated in a model, we will need to fix the speaker. We will do so by defining an interpreting handler.

$$\begin{aligned}
&\text{withSpeaker} : \iota \to \mathcal{F}_{\{\texttt{speaker}:1\rightarrowtail\iota\}\uplus E}(\alpha) \to \mathcal{F}_E(\alpha) \\
&\text{withSpeaker} = \lambda sM.(\!|\, \texttt{speaker}\colon (\lambda xk.k\,s)\,|\!)\,M
\end{aligned}$$

Note that we omitted the η clause in the banana brackets above. In such cases, we say there is a default clause $\eta\colon(\lambda x.\eta\,x)$.

$$\text{withSpeaker}\ s[\![\textsc{loves me Mary}]\!] \twoheadrightarrow \eta\,(\mathbf{love}\,\mathbf{m}\,s)$$

So far, we could have done the same by introducing a constant named **me** to stand in for the speaker. However, since handlers are part of our object language,

[7] 1 is the unit type whose only element is written as $\star$.

we can include them in lexical entries. With this, we can handle phenomena such as direct (quoted) speech, that rebinds the current speaker in a certain scope.

$$\textsc{said}_{\textsc{is}} : S \multimap NP \multimap S$$
$$\textsc{said}_{\textsc{ds}} : S \multimap NP \multimap S$$

Those are our new syntactic constructors: one for the indirect speech use of *said* and the other for the direct speech use (their surface realizations would differ typographically or phonologically). Let us give them some semantics.

$$\begin{aligned}\llbracket\textsc{said}_{\textsc{is}}\rrbracket &= \lambda CS.\mathbf{say} \cdot\gg S \ll\cdot\gg C \\ &= \lambda CS.S \gg\!= (\lambda s.\mathbf{say}\, s \cdot\gg C) \\ \llbracket\textsc{said}_{\textsc{ds}}\rrbracket &= \lambda CS.S \gg\!= (\lambda s.\mathbf{say}\, s \cdot\gg (\text{withSpeaker}\ s\, C))\end{aligned}$$

Here we elaborated the entry for indirect speech so it is easier to compare with the one for direct speech, which just adds a use of the withSpeaker operator.

(3) John said Mary loves me.
(4) John said, "Mary loves me".

$$\llbracket\textsc{said}_{\textsc{is}}\ (\textsc{loves me Mary})\ \textsc{John}\rrbracket \twoheadrightarrow \texttt{speaker} \star (\lambda x.\eta\,(\mathbf{say\,j}\,(\mathbf{love\,m}\,x))) \tag{3}$$
$$\llbracket\textsc{said}_{\textsc{ds}}\ (\textsc{loves me Mary})\ \textsc{John}\rrbracket \twoheadrightarrow \eta\,(\mathbf{say\,j}\,(\mathbf{love\,m\,j})) \tag{4}$$

The meaning of sentence (3) depends on the speaker (as testified by the use of the `speaker` operator) whereas in (4), this dependence has been eliminated due to the use of direct speech.

3.2 Quantification

Now we turn our attention to quantificational noun phrases.

$$\textsc{every}, \textsc{a} : N \multimap NP$$
$$\textsc{man}, \textsc{woman} : N$$

$$\begin{aligned}\llbracket\textsc{every}\rrbracket &:= \lambda N.\texttt{scope}\,(\lambda c.\forall \cdot\gg (\mathcal{C}\,(\lambda x.(N \ll\cdot x) \Rightarrow (c\,x))))\,(\lambda x.\eta\,x) \\ \llbracket\textsc{a}\rrbracket &:= \lambda N.\texttt{scope}\,(\lambda c.\exists \cdot\gg (\mathcal{C}\,(\lambda x.(N \ll\cdot x) \barwedge (c\,x))))\,(\lambda x.\eta\,x) \\ \llbracket\textsc{man}\rrbracket &:= \eta\,\mathbf{man} \\ \llbracket\textsc{woman}\rrbracket &:= \eta\,\mathbf{woman}\end{aligned}$$

The entries for EVERY and A might seem intimidating. However, if we ignore the $\cdot\gg$, the $\mathcal{C}$, the $\ll\cdot$ and the overline on the logical operator, we get the familiar generalized quantifiers. These decorations are the plumbing that takes care of the proper sequencing of effects.

Note that we make use of the $\mathcal{C}$ operator here. In the denotation of $\llbracket\textsc{a}\rrbracket$, the term $(\lambda x.(N \ll\cdot x) \barwedge (c\,x))$ describes the property to which we want to apply the quantifier $\exists$. However, this term is of type $\iota \to \mathcal{F}_E(o)$. In order to apply

$\exists$, we need something of type $\iota \to o$. Intuitively, the effects of E correspond to the process of interpretation, the process of arriving at some logical form of the sentence. They should thus be independent of the particular individual that we use as a witness for x when we try to model-check the resulting logical form. This independence allows us use the $\mathcal{C}$ operator without fear of getting stuck. Once we arrive at the type $\mathcal{F}_E(\iota \to o)$, it is a simple case of using $\exists \cdot \gg\!=\! \_$ to apply the quantifier within the computation type.[8,9]

While the terms that use the `scope` operator might be complex, the handler that interprets them is as simple as can be.

$$\mathrm{SI} = \lambda M.(\!|\, \mathtt{scope}\colon (\lambda ck.c\,k)\,|\!)\, M$$

Same as with withSpeaker, SI will also be used in lexical items. By interpreting the `scope` operation in a particular place, we effectively determine the scope of the quantifier. Hence the name of SI, short for Scope Island. If we want to model clause boundaries as scope islands, we can do so by inserting SI in the lexical entries of clause constructors (in our case, the verbs).

$$\begin{aligned}
[\![\textsc{loves}]\!] &:= \lambda OS.\,\mathrm{SI}\,([\![\textsc{loves}]\!]\,O\,S)\\
[\![\textsc{said}_{\textsc{is}}]\!] &:= \lambda CS.\,\mathrm{SI}\,([\![\textsc{said}_{\textsc{is}}]\!]\,C\,S)\\
[\![\textsc{said}_{\textsc{ds}}]\!] &:= \lambda CS.\,\mathrm{SI}\,([\![\textsc{said}_{\textsc{ds}}]\!]\,C\,S)
\end{aligned}$$

Whenever we use the semantic brackets on the right-hand side of these revised definitions, they stand for the denotations we have assigned previously.

(5) Every man loves a woman.
(6) John said every woman loves me.
(7) John said, "Every woman loves me".

$$\begin{aligned}
&[\![\textsc{loves}\ (\textsc{a woman})\ (\textsc{every man})]\!]\\
&\twoheadrightarrow \eta\,(\forall x.\mathbf{man}\,x \to (\exists y.\mathbf{woman}\,y \wedge \mathbf{love}\,x\,y)) \qquad (5)\\
&\mathrm{withSpeaker}\ s\ [\![\textsc{said}_{\textsc{is}}\ (\textsc{loves me}\ (\textsc{every woman}))\ \textsc{John}]\!]\\
&\twoheadrightarrow \eta\,(\mathbf{say}\,\mathbf{j}\,(\forall x.\mathbf{woman}\,x \to \mathbf{love}\,x\,s)) \qquad (6)\\
&[\![\textsc{said}_{\textsc{ds}}\ (\textsc{loves me}\ (\textsc{every woman}))\ \textsc{John}]\!]\\
&\twoheadrightarrow \eta\,(\mathbf{say}\,\mathbf{j}\,(\forall x.\mathbf{woman}\,x \to \mathbf{love}\,x\,\mathbf{j})) \qquad (7)
\end{aligned}$$

The calculus offers us flexibility when modelling the semantics. We might choose to relax the constraint that clauses are scope islands by keeping the old

[8] Other solutions to this problem include separating the language of logical forms and the metalanguage used in the semantic lexical entries to manipulate logical forms as objects [13].

[9] Our $\mathcal{C}$ has been inspired by an operator of the same name proposed in [9]: de Groote introduces a structure that specializes applicative functors in a similar direction as monads by introducing the $\mathcal{C}$ operator and equipping it with certain laws; our $\mathcal{C}$ operator makes the $\mathcal{F}_E$ type constructor an instance of this structure.

entries for verbs that do not use the SI handler. We might then want to add the SI handler to the lexical entry of SAID_{DS}, next to the withSpeaker handler, so that quantifiers cannot escape quoted expressions. We might also allow for inverse scope readings by, e.g., providing entries for transitive verbs that evaluate their arguments right-to-left (though then we would have to watch out for crossover effects if we were to add anaphora).

3.3 Conventional Implicature

Our goal is to show the modularity of this approach and so we will continue and plug in one more phenomenon into our growing fragment: conventional implicatures, as analyzed by Potts [23]. Specifically, we will focus on nominal appositives.

$$\begin{aligned} \text{APPOS} &: NP \multimap NP \multimap NP \\ \text{BEST-FRIEND} &: NP \multimap NP \end{aligned}$$

$$\begin{aligned} [\![\text{APPOS}]\!] &:= \lambda XY.X \gg\!= (\lambda x.\,\text{SI}\,(\eta\, x \equiv Y) \gg\!= (\lambda i.\texttt{implicate}\, i\, (\lambda z.\eta\, x))) \\ [\![\text{BEST-FRIEND}]\!] &:= \lambda X.\textbf{best-friend} \cdot\!\gg X \end{aligned}$$

In the denotation of the nominal appositive construction, APPOS, we first evaluate the head noun phrase $X : [\![NP]\!]$ to find its referent $x : \iota$. We then want to implicate that x is equal to the referent of Y. The term $\eta\, x \equiv Y$ (note the line over $=$) is the term that computes that referent and gives us the proposition we want. We also want to state that no quantifier from within the appositive Y should escape into the matrix clause and so we wrap this computation in the SI handler to establish a scope island. Finally, we pass this proposition as an argument to `implicate` and we return x as the referent of the noun phrase.

The point of the `implicate` operation is to smuggle non-at-issue content outside the scope of logical operators. The contribution of an appositive should survive, e.g., logical negation.[10] The place where we will accommodate the implicated truth-conditions will be determined by the use of the following handler:

$$\begin{aligned} &\text{accommodate} : \mathcal{F}_{\{\texttt{implicate}:o \rightarrowtail 1\} \uplus E}(o) \to \mathcal{F}_E(o) \\ &\text{accommodate} = \lambda M.(\!|\, \texttt{implicate}\colon (\lambda ik.\eta\, i \mathbin{\overline{\wedge}} k\star)\,|\!)\, M \end{aligned}$$

We want conventional implicatures to project out of the common logical operators. However, when we consider direct quotes, we would not like to attribute the implicature made by the quotee to the quoter. We can implement this by inserting the accommodate handler into the lexical entry for direct speech.

$$[\![\text{SAID}_{\text{DS}}]\!] := \lambda CS.\,\text{SI}\,(S \gg\!= (\lambda s.\textbf{say}\, s \cdot\!\gg (\text{withSpeaker}\ s\,(\text{accommodate}\ C))))$$

Consider the following three examples.

(8) John, my best friend, loves every woman.

[10] In our limited fragment, we will only see it sneak out of a quantifier.

(9) Mary, everyone's best friend, loves John.
(10) A man said, "My best friend, Mary, loves me".

In (8), the conventional implicature that John is the speaker's best friend projects from the scope of the quantifier. On the other hand, in (10), the implicature does not project from the quoted clause and so it is not misattributed.

$$\begin{aligned}
&\text{withSpeaker } s\,(\text{accommodate } [\![\textsc{loves}\,(\textsc{every woman})\,(\textsc{appos John}\,(\textsc{best-friend me}))]\!]) \\
&\twoheadrightarrow \eta\,((\mathbf{j} = \mathbf{best\text{-}friend}\,s) \wedge (\forall x.\mathbf{woman}\,x \rightarrow \mathbf{love}\,\mathbf{j}\,x)) \qquad (8) \\
&\text{accommodate } [\![\textsc{loves John}\,(\textsc{appos Mary}\,(\textsc{best-friend everyone}))]\!] \\
&\twoheadrightarrow \eta\,((\forall x.\mathbf{m} = \mathbf{best\text{-}friend}\,x) \wedge (\mathbf{love}\,\mathbf{m}\,\mathbf{j})) \qquad (9) \\
&[\![\textsc{said}_{\textsc{ds}}\,(\textsc{loves me}\,(\textsc{appos}\,(\textsc{best-friend me})\,\textsc{Mary}))\,(\textsc{a man})]\!] \\
&\twoheadrightarrow \eta\,(\exists x.\mathbf{man}\,x \wedge \mathbf{say}\,x\,((\mathbf{best\text{-}friend}\,x = \mathbf{m}) \wedge (\mathbf{love}\,(\mathbf{best\text{-}friend}\,x)\,x))) \qquad (10)
\end{aligned}$$

3.4 Summary

Let us look back at the modularity of our approach and count how often during the incremental development of our fragment we either had to modify existing denotations or explicitly mention previous effects in new denotations.

When adding quantification:

– in the old denotations of verbs, we added the new SI handler so that clauses form scope islands

When adding appositives and their conventional implicatures:

– in the old denotations $[\![\textsc{said}_{\textsc{ds}}]\!]$, we added the new accommodate handler to state that conventional implicatures should not project out of quoted speech
– in the new denotation $[\![\textsc{appos}]\!]$, we used the old SI handler to state that appositives should form scope islands

Otherwise, none of the denotations prescribed in our semantic lexicon had to be changed. We did not have to type-raise non-quantificational NP constructors like $[\![\textsc{John}]\!]$, $[\![\textsc{me}]\!]$ or $[\![\textsc{best-friend}]\!]$. With the exception of direct speech, we did not have to modify any existing denotations to enable us to collect conventional implicatures from different subconstituents.

Furthermore, all of the modifications we have performed to existing denotations are additions of handlers for new effects. This gives us a strong guarantee that all of the old results are conserved, since applying a handler to a computation which does not use the operations being handled changes nothing.

The goal of our calculus is to enable the creation of semantic lexicons with a high degree of separation of concerns. In this section, we have seen how it can be done for one particular fragment.

4 Properties of the Calculus

The calculus defined in Sect. 2 and to which we will refer as $(\!|\lambda|\!)$, has some satisfying properties.

First of all, the reduction rules preserve types of terms (subject reduction). The reduction relation itself is confluent and, for well-typed terms, it is also terminating. This means that typed $(\!|\lambda|\!)$ is strongly normalizing.

The proof of subject reduction is a mechanical proof by induction. For confluence and termination, we employ very similar strategies: we make use of general results and show how they apply to our calculus. Due to space limitations, we will pursue in detail only the proof of confluence.

Our reduction relation is given as a set of rules which map redexes matching some pattern into contracta built up out of the redexes' free variables. However, our language also features binding, and so some of the rules are conditioned on whether or not certain variables occur freely in parts of the redex. Fortunately, such rewriting systems have been thoroughly studied. Klop's Combinatory Reduction Systems (CRSs) [16] is one class of such rewriting systems.

We will make use of the result that orthogonal CRSs are confluent [16]. A CRS is *orthogonal* if it is left-linear and non-ambiguous. We will need to adapt our formulation of the reduction rules so that they form a CRS and we will need to check whether we satisfy left-linearity and non-ambiguity (we will see what these two properties mean when we get to them).

We refer the reader to [16] for the definition of CRSs. The key point is that in a CRS, the free variables which appear in the left-hand side of a rewrite rule, called metavariables, are explicitly annotated with the set of all free variables that are allowed to occur within a term which would instantiate them. This allows us to encode all of our $x \notin FV(M)$ constraints.

One detail which must be taken care of is the set notation $(\mathtt{op}_i\colon M_i)_{i\in I}$ and the indices I used in the $(\!|\ |\!)$ rules. We do away with this notation by adding a separate rule for every possible instantiation of the schema. This means that for each sequence of distinct operation symbols $\mathtt{op}_1,\ldots,\mathtt{op}_n$, we end up with:

- a special rewriting rule $(\!|\,\mathtt{op}_1\colon M_1,\ldots,\mathtt{op}_n\colon M_n,\ \eta\colon M_\eta\,|\!)\,(\eta\,N) \to M_\eta\,N$
- for every $1 \le i \le n$, a special rewriting rule
  $(\!|\,\mathtt{op}_1\colon M_1,\ldots,\mathtt{op}_n\colon M_n,\ \eta\colon M_\eta\,|\!)\,(\mathtt{op}_i\,N_\mathrm{p}\,(\lambda x.N_\mathrm{c}(x)))$
  $\to M_i\,N_\mathrm{p}\,(\lambda x.(\!|\,\mathtt{op}_1\colon M_1,\ldots,\mathtt{op}_n\colon M_n,\ \eta\colon M_\eta\,|\!)\,N_\mathrm{c}(x))$
- for every $\mathtt{op}' \in \mathcal{E} \setminus \{\mathtt{op}_i \| 1 \le i \le n\}$, a special rewriting rule
  $(\!|\,\mathtt{op}_1\colon M_1,\ldots,\mathtt{op}_n\colon M_n,\ \eta\colon M_\eta\,|\!)\,(\mathtt{op}'\,N_\mathrm{p}\,(\lambda x.N_\mathrm{c}(x)))$
  $\to \mathtt{op}'\,N_\mathrm{p}\,(\lambda x.(\!|\,\mathtt{op}_1\colon M_1,\ldots,\mathtt{op}_n\colon M_n,\ \eta\colon M_\eta\,|\!)\,N_\mathrm{c}(x))$

So now we have a CRS which defines the same reduction relation as the rules we have shown in Sect. 2.3. Next, we verify the two conditions. Left-linearity states that no left-hand side of any rule contains multiple occurrences of the same metavariable. By examining our rules, we find that this is indeed the case.[11]

Non-ambiguity demands that there is no non-trivial overlap between any of the left-hand sides.[12] In our CRS, we have overlaps between the β and the η rules. We split our CRS into one with just the η rule ($\to_\eta$) and one with all the

[11] Multiple occurrences of the same $\mathtt{op}_i$ are alright, since those are not metavariables.

[12] The definition of (non-trivial) overlap is the same one as the one used when defining critical pairs. See [16] for the precise definition.

other rules ($\rightarrow_{(\!|\lambda|\!)}$). Now, there is no overlap in either of these CRSs, so they are both orthogonal and therefore confluent.

We then use the Lemma of Hindley-Rosen [17, p.7] to show that the union of $\rightarrow_{(\!|\lambda|\!)}$ and $\rightarrow_\eta$ is confluent when $\rightarrow_{(\!|\lambda|\!)}$ and $\rightarrow_\eta$ are both confluent and commute together. For that, all that is left to prove is that $\rightarrow_{(\!|\lambda|\!)}$ and $\rightarrow_\eta$ commute. Thanks to another result due to Hindley [17, p.8], it is enough to prove that for all a, b and c such that $a \rightarrow_{(\!|\lambda|\!)} b$ and $a \rightarrow_\eta c$, we have a d such that $b \twoheadrightarrow_\eta d$ and $c \rightarrow^{=}_{(\!|\lambda|\!)} d$. The proof of this is a straightforward induction on the structure of a.

5 Conclusion

In our contribution, we have introduced a new calculus motivated by modelling detailed semantics and inspired by current work in programming language theory. Our calculus is an extension of the simply-typed lambda calculus which is the de facto lingua franca of semanticists. Its purpose is to facilitate the communication of semantic ideas without depending on complex programming languages [15,19] and to do so with a well-defined formal semantics.

We have demonstrated the features of our calculus on several examples exhibiting phenomena such as deixis, quantification and conventional implicature. While our calculus still requires us to do some uninteresting plumbing to be able to correctly connect all the denotations together, we have seen that the resulting denotations are very generic. We were able to add new phenomena without having to change much of what we have done before and the changes we have made arguably corresponded to places where the different phenomena interact.

Finally, we have also shown that the calculus shares some of the useful properties of the simply-typed lambda calculus, namely strong normalization.

In future work, it would be useful to automate some of the routine plumbing that we have to do in our terms. It will also be important to test the methodology on larger and more diverse fragments (besides this fragment, we have also created one combining anaphora, quantification and presupposition [19]). Last but not least, it would be interesting to delve deeper into the foundational differences between the approach used here, the monad transformers used by Charlow [4] and the applicative functors used by Kiselyov [13].

References

1. Barker, C.: Continuations and the nature of quantification. Nat. Lang. Semant. **10**(3), 211–242 (2002)
2. Bauer, A., Pretnar, M.: Programming with algebraic effects and handlers (2012). arXiv preprint: arXiv:1203.1539
3. Brady, E.: Programming and reasoning with algebraic effects and dependent types. In: ACM SIGPLAN Notices (2013)
4. Charlow, S.: On the semantics of exceptional scope. Ph.D. thesis, New York University (2014)

5. de Groote, P.: Towards abstract categorial grammars. In: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (2001)
6. de Groote, P.: Type raising, continuations, and classical logic. In: Proceedings of the Thirteenth Amsterdam Colloquium (2001)
7. de Groote, P.: Towards a Montagovian account of dynamics. In: Proceedings of SALT, vol. 16 (2006)
8. Giorgolo, G., Asudeh, A.: Natural language semantics with enriched meanings (2015)
9. de Groote, P.: On logical relations and conservativity. In: Third Workshop on Natural Language and Computer Science, NLCS 2015 (2015)
10. Hobbs, J., Rosenschein, S.: Making computational sense of montague's intensional logic. Artif. Intell. **9**(3), 287–306 (1977)
11. Kammar, O., Lindley, S., Oury, N.: Handlers in action. In: ACM SIGPLAN Notices (2013)
12. Kamp, H., Reyle, U.: From Discourse to Logic. Kluwer Academic Pub, Dordrecht (1993)
13. Kiselyov, O.: Applicative abstract categorial grammars. In: Proceedings of the Third Workshop on Natural Language and Computer Science (2015)
14. Kiselyov, O., Sabry, A., Swords, C.: Extensible effects: an alternative to monad transformers. In: ACM SIGPLAN Notices (2013)
15. Kiselyov, O., Shan, C.: Lambda: the ultimate syntax-semantics interface (2010)
16. Klop, J.W., Van Oostrom, V., Van Raamsdonk, F.: Combinatory reduction systems: introduction and survey. Theor. Comput. Sci. **121**(1–2), 279–308 (1993)
17. Klop, J.W., et al.: Term rewriting systems. Handb. Logic Comput. Sci. **2**, 1–116 (1992)
18. Lebedeva, E.: Expression de la dynamique du discours à l'aide de continuations. Ph.D. thesis, Université de Lorraine (2012)
19. Maršík, J., Amblard, M.: Algebraic effects and handlers in natural language interpretation. In: Natural Language and Computer Science, July 2014
20. Meijer, E., Fokkinga, M., Paterson, R.: Functional programming with bananas, lenses, envelopes and barbed wire. In: Functional Programming Languages and Computer Architecture (1991)
21. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (1991)
22. Plotkin, G., Pretnar, M.: Handlers of algebraic effects. In: Castagna, G. (ed.) ESOP 2009. LNCS, vol. 5502, pp. 80–94. Springer, Heidelberg (2009)
23. Potts, C.: The Logic of Conventional Implicatures. Oxford University Press, Oxford (2005)
24. Pretnar, M.: Logic and handling of algebraic effects. Ph.D. thesis, The University of Edinburgh (2010)
25. Qian, S., Amblard, M.: Event in compositional dynamic semantics. In: Pogodalla, S., Prost, J.-P. (eds.) Logical Aspects of Computational Linguistics. LNCS, vol. 6736, pp. 219–234. Springer, Heidelberg (2011)
26. Shan, C.: Monads for natural language semantics. arXiv:cs/0205026v1 (2002)
27. Shan, C.: Linguistic side effects. Ph.D. thesis, Harvard University (2005)
28. Van Eijck, J., Unger, C.: Computational Semantics with Functional Programming. Cambridge University Press, Cambridge (2010)
29. Wadler, P.: The essence of functional programming. In: Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (1992)

Proof Nets for the Displacement Calculus

Richard Moot(✉)

CNRS (LaBRI), Talence, France
Richard.Moor@labri.fr

Abstract. We present a proof net calculus for the Displacement calculus and show its correctness. This is the first proof net calculus which models the Displacement calculus directly and not by some sort of translation into another formalism. The proof net calculus opens up new possibilities for parsing and proof search with the Displacement calculus.

1 Introduction

The Displacement calculus was introduced by Morrill et al. [7] as an extension of the Lambek calculus with discontinuous operators. These discontinuous connectives allow the Displacement calculus to solve a large number of problems with the Lambek calculus. Examples of the phenomena treated by Morrill et al. [7] include discontinuous idioms (such as "ring up" and "give the cold shoulder"), quantifier scope, extraction (including pied-piping) and gapping.

This paper extends earlier work by Morrill and Fadda [6], Moot [3] and Valentín [9], combining the strengths of these different approaches while at the same time diminishing the inconveniences. Notably, it is the first proof net calculus which does not operate by translation into some other logic, but provides proof nets for the Displacement calculus directly.

2 The Displacement Calculus

The presentation of the Displacement calculus closely follows the natural deduction calculus used by Morrill et al. [7]. String terms are built over a countably infinite alphabet of variables (for readability, we will often used natural language words as if they were variables), a special separator symbol "$\mathbf{1}$", where string concatenation is denoted by "+" (a binary, associative infix operator on string terms). As usual, ϵ denotes the empty string. The *sort* of a string term is the number of occurrences of the separator "$\mathbf{1}$".

I use lower-case roman letters p, q ... for atomic string terms (for enhanced readability, I will often use the standard convention of using words from the lexicon in the place of such atomic string terms), lower-case roman letters a, b, ... for string terms without separator symbols and lower-case greek letters α, β, ... for strings containing any number of separator symbols. So the string term $p + \mathbf{1} + q + \mathbf{1} + r$ is a string of sort 2 with three atomic subterms.

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 273–289, 2016.
DOI: 10.1007/978-3-662-53042-9_16

The key to the Displacement calculus is the wrap operator $\alpha \times_k \beta$. There is some minor variation in the definition of this operator: sometimes [7] k is either the constant ">" or the constant "<" (in which case α is of sort greater than zero and the denotation of the term replaces respectively the first and the last occurrences of $\mathbf{1}$ in α by β. Sometimes [5] k is an integer (between 1 and the sort of α) and $\alpha \times_k \beta$ replaces the kth separator in α by β. The equations below give the definition of "$\times_k$".

$$(a + \mathbf{1} + \alpha) \times_> \beta =_{def} a + \beta + \alpha \tag{1}$$
$$(\alpha + \mathbf{1} + a) \times_< \beta =_{def} \alpha + \beta + a \tag{2}$$
$$(a_1 + \mathbf{1} + \ldots + a_n + \mathbf{1} + \alpha) \times_n \beta =_{def} a_1 + \mathbf{1} + \ldots + a_n + \beta + \alpha \tag{3}$$

Where the Lambek calculus connectives get their meaning with respect to concatenation "+", the discontinuous connectives of the Displacement calculus get their meaning with respect to "$\times_k$" (this entails different connectives for different values of k). The standard interpretation of the Lambek calculus connectives for string models, with "+" denoting concatenation, is the following.

$$|A\backslash C| =_{def} \{\beta \,|\, \forall \alpha \in |A|, \alpha + \beta \in |C|\} \tag{4}$$
$$|C/B| =_{def} \{\alpha \,|\, \forall \beta \in |B|, \alpha + \beta \in |C|\} \tag{5}$$
$$|A \bullet B| =_{def} \{\alpha + \beta \,|\, \alpha \in |A| \wedge \beta \in |B|\} \tag{6}$$

The discontinuous connectives of the Displacement calculus use "$\times_k$" instead of "+" (we present only the connectives for $>$ here).

$$|A \downarrow_> C| =_{def} \{\beta \,|\, \forall \alpha \in |A|, \alpha \times_> \beta \in |C|\} \tag{7}$$
$$|C \uparrow_> B| =_{def} \{\alpha \,|\, \forall \beta \in |B|, \alpha \times_> \beta \in |C|\} \tag{8}$$
$$|A \odot_> B| =_{def} \{\alpha \times_> \beta \,|\, \alpha \in |A| \wedge \beta \in |B|\} \tag{9}$$

We can further unfold these definitions, using Definition 1 for "$\times_>$" to obtain.

$$|A \downarrow_> C| =_{def} \{\beta \,|\, \forall (a + \mathbf{1} + \alpha) \in |A|, a + \beta + \alpha \in |C|\} \tag{10}$$
$$|C \uparrow_> B| =_{def} \{(a + \mathbf{1} + \alpha) \,|\, \forall \beta \in |B|, a + \beta + \alpha \in |C|\} \tag{11}$$
$$|A \odot_> B| =_{def} \{a + \beta + \alpha \,|\, (a + \mathbf{1} + \alpha) \in |A| \wedge \beta \in |B|\} \tag{12}$$

Given these definitions, the meaning of $A \downarrow_> C$ is defined as the set of expressions which select a circumfix A, which wraps around the string denoted by $A \downarrow_> C$ to form an expression C. Similarly, $C \uparrow_> B$ extracts a B formula not occurring after a separator.

2.1 Formulas and Sorts

We have already defined the sort of a string term as the number of occurrences of the separator constant "$\mathbf{1}$". The *sort* of a formula corresponds to the number of separators "$\mathbf{1}$" occurring in its denotation. That is, a formula of sort n is assigned

a string term of the form $a_0 + \mathbf{1} + \ldots + \mathbf{1} + a_n$ (with all a_i of sort 0 according to our notational convention). For a given grammar, its signature defines the sort of all atomic formulas occurring in the grammar. We assume throughout that the atomic formulas s, n, np, pp have sort 0 (some other atomic formulas, such as *inf* when used for Dutch verb clusters, have sort 1).

Table 1 shows how to compute the sort of complex formulas. All subformulas of a formula are assigned a sort, so when we compute $s(C/B)$ using its entry in Table 1 we know that $s(C) \geq s(B)$, because if not, then $s(C/B)$ would be less than zero and therefore not a valid (sub)formula (similar constraints can be derived from the other implications, eg. we can show that $s(C \uparrow B) \geq 1$).

Table 1. Computing the sort of a complex formula given the sort of its immediate subformulas

$s(A \bullet B) = s(A) + s(B)$	$s(A \odot B) = s(A) + s(B) - 1$	$s(A) \geq 1$
$s(A \backslash C) = s(C) - s(A)$	$s(A \downarrow C) = s(C) + 1 - s(A)$	$s(A) \geq 1$
$s(C/B) = s(C) - s(B)$	$s(C \uparrow B) = s(C) + 1 - s(B)$	$s(C) \geq s(B)$

As an example, following Morrill et al. [7], we can assign a discontinuous lexical entry like "give the cold shoulder" the lexical formula $(np \backslash s) \uparrow_{>} np$ and string term $gave + \mathbf{1} + the + cold + shoulder$ (of the required sort 1).

2.2 Natural Deduction Rules

Figures 1 and 2 give the natural deduction rules for the Lambek calculus and for the left wrap rules respectively (the other wrap rules follow the same pattern). The left wrap rules of Fig. 2 correspond rather closely to the interpretation of the formulas given in Definitions 10 to 12.

3 Proof Nets

One of the goals of proof search in type-logical grammars is to enumerate all possible readings for a given sentence. The bureaucratic aspects of the sequent calculus proof search make it hard to use sequent calculus directly for this goal, since sequent calculus allows a great number of inessential rule permutations. The situation for natural deduction is somewhat better, since the proof rules correspond directly to steps in meaning composition, even though there is still a large number of possible rule permutations for the $\bullet E$ and $\odot_k E$ rules.

Proof nets are a way of representing proofs which removes the "bureaucratic" aspects of sequent proofs and simplifies the product rules of Lambek calculus natural deduction. One of the open questions of Morrill [5] is whether the Displacement calculus has a proof net calculus.

Valentín [9] provides a translation of the Displacement calculus to a multimodal system. However, this system uses a rather large set of structural rules

$$\frac{\alpha : A \quad \gamma : A \setminus C}{\alpha{+}\gamma : C}\ \setminus E \qquad \frac{\begin{array}{c}[\alpha : A]^i \\ \vdots \\ \alpha{+}\gamma : C\end{array}}{\gamma : A \setminus C}\ \setminus I_i$$

$$\frac{\gamma : C \,/\, B \quad \beta : B}{\gamma{+}\beta : C}\ / E \qquad \frac{\begin{array}{c}[\beta : B]^i \\ \vdots \\ \gamma{+}\beta : C\end{array}}{\gamma : C \,/\, B}\ / I_i$$

$$\frac{\delta : A \bullet B \qquad \begin{array}{c}[\alpha : A]^i \quad [\beta : B]^i \\ \vdots \\ \gamma[\alpha{+}\beta] : C\end{array}}{\gamma[\delta] : C}\ \bullet E_i \qquad \frac{\alpha : A \quad \beta : B}{\alpha{+}\beta : A \bullet B}\ \bullet I$$

Fig. 1. Proof rules – Lambek calculus

$$\frac{a{+}\mathbf{1}{+}\alpha : A \quad \gamma : A \downarrow_> C}{a{+}\gamma{+}\alpha : C}\ \downarrow_> E \qquad \frac{\begin{array}{c}[a{+}\mathbf{1}{+}\alpha : A]^i \\ \vdots \\ a{+}\gamma{+}\alpha : C\end{array}}{\gamma : A \downarrow_> C}\ \downarrow_> I_i$$

$$\frac{c{+}\mathbf{1}{+}\gamma : C \uparrow_> B \quad \beta : B}{c{+}\beta{+}\gamma : C}\ \uparrow_> E \qquad \frac{\begin{array}{c}[\beta : B]^i \\ \vdots \\ c{+}\beta{+}\gamma : C\end{array}}{c{+}\mathbf{1}{+}\gamma : C \uparrow_> B}\ \uparrow_> I_i$$

$$\frac{\delta : A \odot_> B \qquad \begin{array}{c}[a{+}\mathbf{1}{+}\alpha : A]^i \quad [\beta : B]^i \\ \vdots \\ \gamma[a{+}\beta{+}\alpha] : C\end{array}}{\gamma[\delta] : C}\ \odot_> E_i \qquad \frac{a{+}\mathbf{1}{+}\alpha : A \quad \beta : B}{a{+}\beta{+}\alpha : A \odot_> B}\ \odot_> I$$

Fig. 2. Proof rules — leftmost infixation, extraction

and these rules are defined modulo equivalence classes, which makes their use in existing multimodal theorem provers [2] difficult. In this section, I will extend the proof net calculus for the Lambek calculus of Moot and Puite [4] to the Displacement calculus. I will, in particular, provide an efficiently checkable correctness condition in the form of graph contractions.

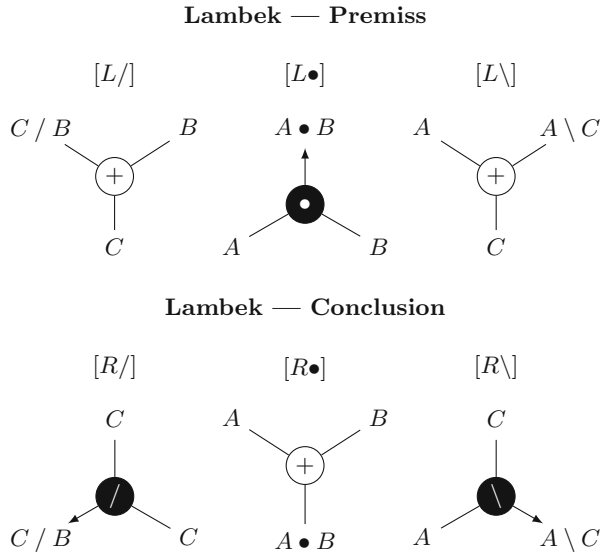


Fig. 3. Links for the Lambek calculus connectives of the Displacement calculus

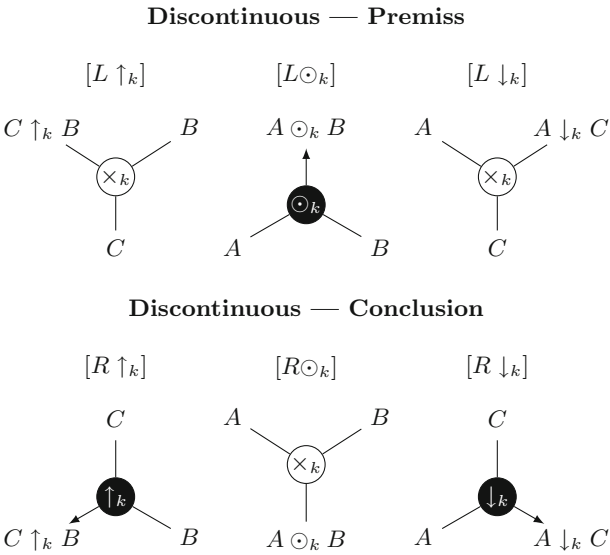


Fig. 4. Links for the discontinuous connectives of the Displacement calculus

3.1 Links

Figures 3 and 4 show the links for Displacement calculus proof structures. Each link connects three formulas to a central node. The formulas written above the

central node of a link are the *premisses* of the link, the formulas written below it are its *conclusions*. The linear order of both the premisses and the conclusions of a link is important.

We distinguish between par links, where the central node is filled black, and tensor links, where the central node is not filled (this is the familiar tensor/par distinction of multiplicative linear logic). Par nodes are further distinguished by an arrow pointing to the main formula of the link.

3.2 Proof Structures

A *proof structure* is a set of formulas and a set of links such that.

1. each link instantiates one of the links shown in Figs. 3 and 4 (for specific values of A, B, C and k),
2. each formula is the premiss of at most one link,
3. each formula is the conclusion of at most one link.

Formulas which are not the premiss of any link are the conclusions of the proof structure. Formulas which are not the conclusion of any link are the hypotheses of the proof structure (the word "conclusion" is overloaded: we talk about conclusions of proofs, conclusions of rules, conclusions of links and conclusions of proof structures; when the intended use is clear from the context, I will often simply use the word "conclusion" without further qualification). The *inputs* of a proof structure are its hypotheses and the active conclusions of its par links (that is, the conclusions of all par links in the proof structure except, for the implications, the one with the arrow); we will call the inputs which are not hypotheses the *auxiliary inputs* of a proof structure.

To construct a proof structure for a given sequent $A_1, \ldots, A_n \vdash C$, we unfold the A_i as premisses and C as a conclusion. This will provide a proof structure with (atomic) conclusions other than C and (atomic) hypotheses other than the A_i. We identify these atomic hypotheses with atomic conclusions (of the same atomic formula) until we obtain a proof structure of $A_1, \ldots, A_n \vdash C$. This can fail if an atomic formula has more occurrences as a hypothesis than as a conclusion (as it should, since such sequents are underivable).

Figure 5 gives an unfolding for the sentence "Mary rang everyone up", a sentence with the discontinuous idiom "rang up" and a non-peripheral quantifier "everyone", following lexical entries of Morrill et al. [7]. Figure 6 shows (on the left of the figure) one of the possibilities for connecting the atomic formulas.

Not all proof structures correspond to natural deduction proofs. Proof structures which correspond to natural deduction proofs are *proof nets*. Of course, defining proof nets this way is not very satisfactory: we want to have a condition which, given a proof structure tells us whether or not this proof structure is a proof net using only properties of the proof structure itself.

3.3 Abstract Proof Structures

The general strategy we follow to define a correctness criterion for proof structures is as follows: we first simplify by removing some of the information which is

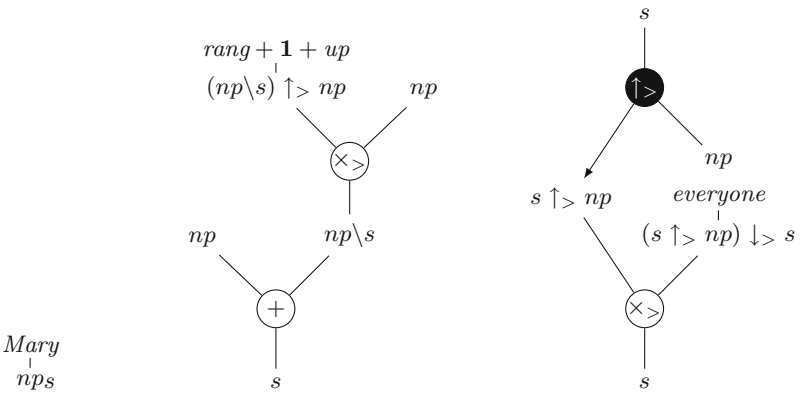


Fig. 5. Unfolding for the sentence "Mary rang everyone up".

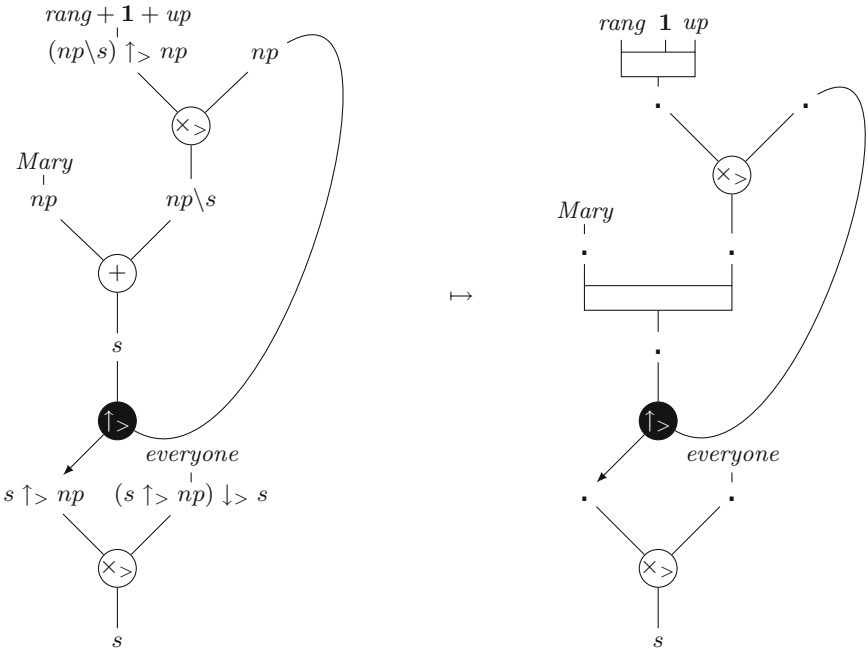


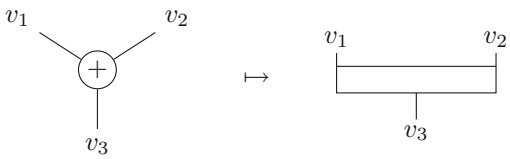
Fig. 6. Proof structure (left) and abstract proof structure (right) for the unfolding of "Mary rang everyone up" shown in Fig. 5.

irrelevant for deciding correctness to obtain abstract proof structures, then specify the correctness condition on these abstract proof structures, using a graph contraction criterion, generalizing the proof nets of the Lambek calculus from Moot and Puite [4].

Tensor Trees and Combs. A *tensor tree* is a connected, acyclic set of tensor links (to be more precise, the underlying undirected graph must by acyclic and connected). A single vertex is a tensor tree. Given an (abstract) proof structure, its tensor trees are the maximal substructures which are tensor trees; this is simply the forest we obtain when we remove all par links from a proof structure. The proof structure of Fig. 6 has two tensor trees.

A *comb* is a link with any number of premisses and a single conclusion. None of the premisses of the comb can be identical to its conclusion. The general conditions on links prevent premisses from being connected more than once as a premiss of a comb. The premisses of combs, as links in general, are linearly ordered. Premisses of a comb can be hypotheses of the proof structure, the conclusions of a link or the special constant $\mathbf{1}$. The *sort* of a comb, that is the sort assigned to its conclusion, is the sum of the sorts of its premisses (the constant $\mathbf{1}$ is of sort 1). Combs play the same role as tensor trees do for Moot and Puite [4]: they allow us to go back a forth between sequents $\Gamma \vdash C$ and combs with premisses Γ and conclusion C. Given a comb, we will refer to subsequences of its premisses as prefixes, postfixes, etc., and assign them sorts as well.

Translating a Proof Structure to an Abstract Proof Structure. To translate a proof structure $\mathcal{P}$ to an abstract proof structure $\mathcal{A}$, we define a function, $\mathcal{P} \mapsto \mathcal{A}$, which replaces "+" links by 2-premiss combs as follows



which leaves all other links the same and which replaces the vertices/formulas of $\mathcal{P}$ as shown in Fig. 7. The only slight complication is for the input formulas (lexical our auxiliary). Proof structures are defined as ways of connecting formulas, but for formulating correctness we need to know about the strings denoted by these formulas, for example, about their position relative to other formulas, separator symbols or the left/rightmost position. Another way of seeing this is that we need to replace sorted variables α (such as those assigned to hypotheses) by variables $p_0 + \mathbf{1} + \ldots + \mathbf{1} + p_n$, with each p_i of sort 0 (such a strategy is already implicitly used for the natural deduction rules for $/I$, $\backslash I$, $\bullet E$, $\uparrow_k I$, $\downarrow_k I$ and $\odot_k E$, that is the natural deduction rules corresponding to the par links). As shown in Fig. 7, auxiliary inputs separate the path leaving the par link by adding n new subpaths (this appears somewhat odd, but is required for the correct behaviour of the contractions when sorts are greater than 0, as we will see below). Because of the sorts of the formulas, the par links for $\downarrow_k$ and $\odot_k$ necessarily involve at least one such split, though the other par links need not.

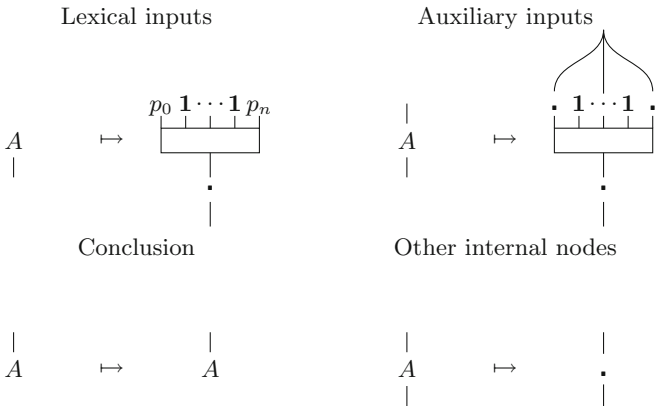


Fig. 7. Conversion to abstract proof structures for vertices/formulas.

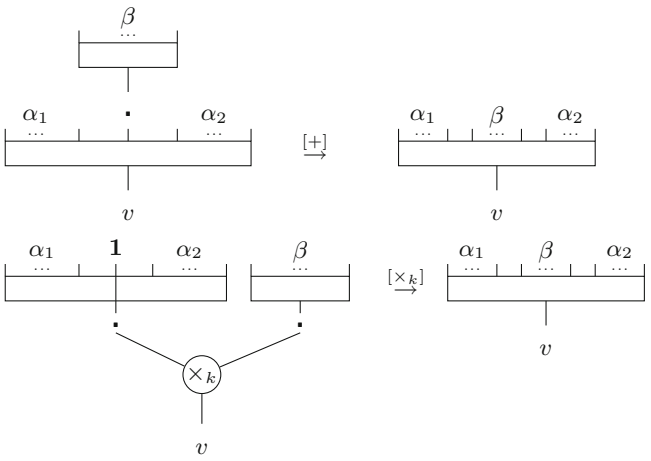


Fig. 8. Structural contractions

3.4 Contractions

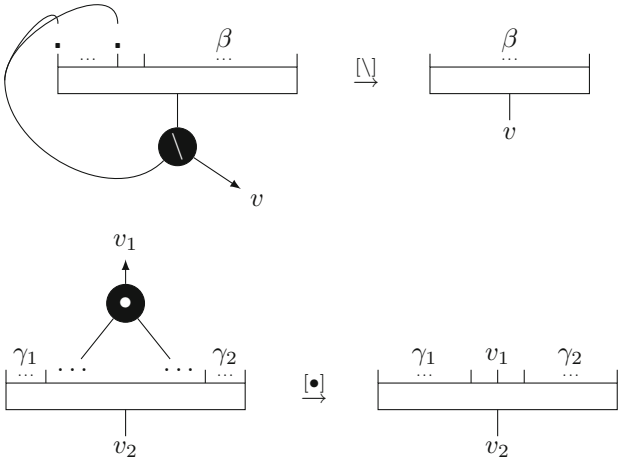
Structural Contractions. Figure 8 shows the structural contractions. The "+" contraction composes two combs, combining the premisses by a simple left-to-right traversal. It is worth mentioning some immediate corollaries of this contraction here: first, we simply eliminate trivial combs (containing a single premiss and a single conclusion, that is, when β contains only a single premiss) when their conclusion is the premiss of another comb, and, second, the structural contractions contract tensor trees to unique combs (this is no longer guaranteed once we add the synthetic connectives, as discussed in Sect. 4).

The wrap operation "$\times_k$" reflects the wrap operation on strings at the level of abstract proof structures, it inserts β at the separator indicated by k: if k is

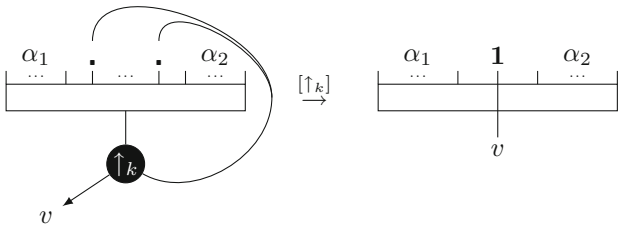
">", the α_1 must be of sort 0 (we replace the first separator by β) and if k is "<" α_2 must be of sort 0 (we replace the last separator by β).

Note that α_1, α_2 and β are allowed to have zero premisses.

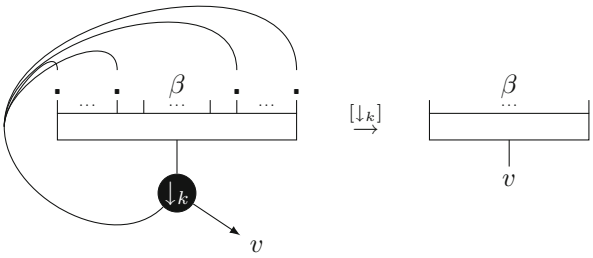
Logical Contractions. The logical contractions ensure the logical symmetry of the connectives in the calculus. Each par rule has its own contraction. The contraction for $\backslash$, shown below, essentially checks whether the string term of the premiss is equivalent to $p_0 + \mathbf{1} + \ldots + \mathbf{1} + p_n + \beta$ and withdraws the hypothesis $p_0 + \mathbf{1} + \ldots + \mathbf{1} + p_n$ (where n is the sort of the withdrawn formula in the corresponding $\backslash I$ rule) to reduce to β (the / contraction is left-right symmetric).



The contraction for $\uparrow_k$ essentially checks that its auxiliary input is an infix of the appropriate sort.

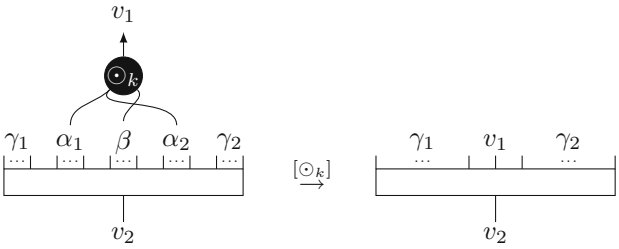


Depending on k, there are restrictions on the sorts: for ">", α_1 must be of sort 0 (that is, all premisses of the comb to the left β are of sort 0), for "<", α_2 must be of sort 0 (that is, all premisses of the comb to the right β are of sort 0), for $k = n$, α_1 is a prefix of sort $n - 1$ (that is, the sorts of the premisses of the comb to the left β sum to $n - 1$).



If k is "$>$", the premisses to the left of β are of sort 0. If k is "$<$", the premisses to the right of β are of sort 0. If $k = n$, the premisses to the left of β are of sort $n - 1$. This contraction looks odd until we realize that we are dealing with a circumfix operation and that, as a consequence the subformula A of a formula $A \downarrow_k B$ denotes a discontinuous circumfix with corresponding string $\alpha_1 + \mathbf{1} + \alpha_2$ (look back to the introduction rule for $\downarrow_k$ on the top right of Fig. 2 for comparison).

The contraction for $\odot_k$ generalizes the contraction for $\bullet$. Whereas the contraction for $A \bullet B$ verifies the strings of the subformulas A and B are adjacent, the contraction for $A \odot_k B$ verifies whether the string $\alpha_1 + \mathbf{1} + \alpha_2$ of A is circumfixed around the string β of B. The sorts of α_1, α_2 and β depend on k and on the sorts of A and B, exactly as for the other rules (in the rule below, the labels α_1, α_2 and β represent sequences of vertices which are premisses of the comb and conclusions of the $\odot_k$ rule).



As an example of how we can use the contraction criterion to verify whether a proof structure is a proof net, the abstract proof structure of Fig. 6 can be contracted first by using a "$+$" and a "$\times_>$" contraction to produce the abstract proof structure shown on the left of Fig. 9, then by performing the "$\uparrow_>$" and "$\times_>$" contractions as indicated, giving a proof of "Mary rang everyone up".

Brief Remarks on Complexity. It is easy to see the given contraction calculus is confluent and that each of the contraction steps reduces the total number of links in the structure. Therefore, even a naive implementation of this contraction calculus checks whether or not a given abstract proof structure with n links contracts to a comb in $O(n^3)$ steps, simply by repeatedly traversing the links in the graph to find contractible configurations and contracting them once they are found (we can likely improve upon this worst case, but I will leave this to further research).

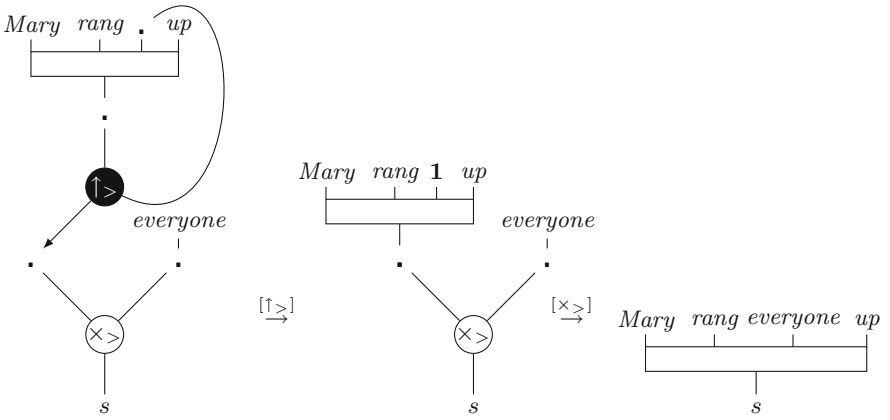


Fig. 9. Contractions for the abstract proof structure for the sentence "Mary rang everyone up" shown on the right of Fig. 6.

In particular, this shows NP-completeness of the Displacement calculus. NP-hardness follows from NP-completeness of the Lambek calculus [8] and we can show it is in NP since we can verify in polynomial time whether or not a candidate proof (that is, a proof structure) in the Displacement calculus is a proof (that is, a proof net).

3.5 Correctness of the Calculus

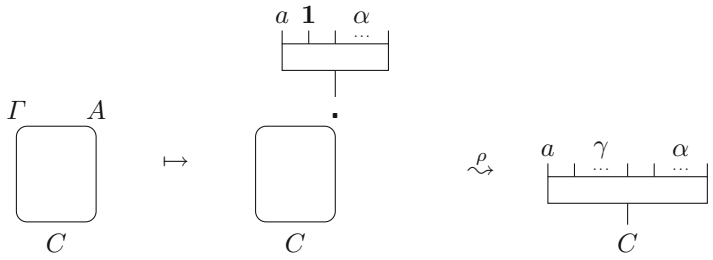
We show that the two definitions of proof net, contractibility and corresponding to a natural deduction proof coincide, thereby establishing that the contraction criterion is correct.

Lemma 1. *Let δ be a Displacement calculus natural deduction proof of $\alpha_1 : A_1, \ldots, a_n : A_n \vdash \gamma : C$. There is a proof net with the same hypotheses whose abstract proof structure contracts to a $\gamma : C$.*

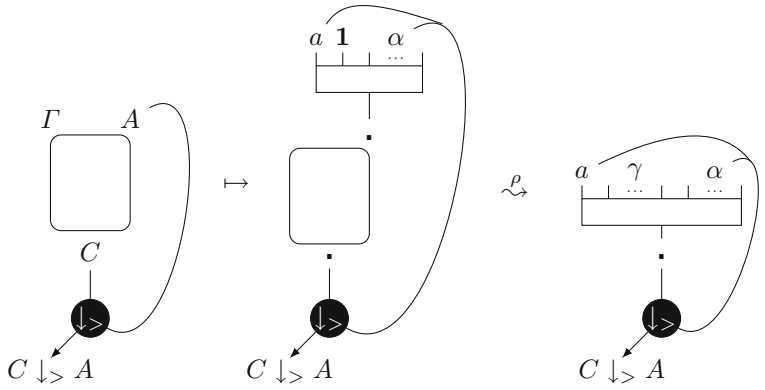
Proof. This is a simple induction on the length of the proof. Axioms $\alpha : A \vdash \alpha : A$ correspond directly to proof nets with the required combs. Otherwise, we proceed by case analysis on the last rule of the proof. Each logical rule correspond to adding a link to the proof net(s) given by induction hypothesis and a contraction to the sequence of contractions for the abstract proof structure. We show only the case for $\downarrow_>$. In this case, the last rule looks as follows.

$$\dfrac{\begin{array}{c}[a{+}\mathbf{1}{+}\alpha : A]^i \\ \vdots\ \delta \\ a{+}\gamma{+}\alpha : C\end{array}}{\gamma : A \downarrow_> C}\ \downarrow_> I_i$$

Induction hypothesis gives use a proof net of $\Gamma, a{+}\mathbf{1}{+}\alpha : A \vdash a{+}\gamma{+}\alpha : C$. That is we are in the situation shown below, with the proof structure shown below on the right, the corresponding abstract proof structure in the middle, for which we are given a sequence of reductions ρ to a comb $a{+}\gamma{+}\alpha : C$. We have simply spelled out the definition of proof net of $\Gamma, a{+}\mathbf{1}{+}\alpha : A \vdash a{+}\gamma{+}\alpha : C$.



Adding the par link for $\downarrow_>$ to the above proof net produces to following proof structure, which contracts using the same sequence of contractions ρ as follows.

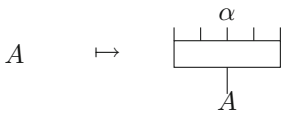


Simply performing the contraction for $\downarrow_>$ to the final abstract proof structure produces a comb of $\gamma : C \downarrow_> A$ and hence a proof net of $\Gamma \vdash \gamma : C \downarrow_> A$ as required. □

Lemma 2. *Let Π be a proof net of $\alpha_1 : A_1, \ldots, a_n : A_n \vdash \gamma : C$, that is a proof net with hypotheses $\alpha_1 : A_1, \ldots, a_n : A_n$ and conclusion C and an abstract proof structure contracting to γ using contractions ρ. There is a natural deduction proof of $\alpha_1 : A_1, \ldots, a_n : A_n \vdash \gamma : C$.*

Proof. We proceed by induction on the number of logical contractions l in the sequence ρ (this number is equal to the number of par links in the structure).

If there are no logical contractions ($l = 0$), then there are only structural contractions and our proof net contains only tensor links. We proceed by induction on the number t of tensor links. If there are no tensor links ($t = 0$), we have an axiom and its abstract proof structure is a comb by definition.

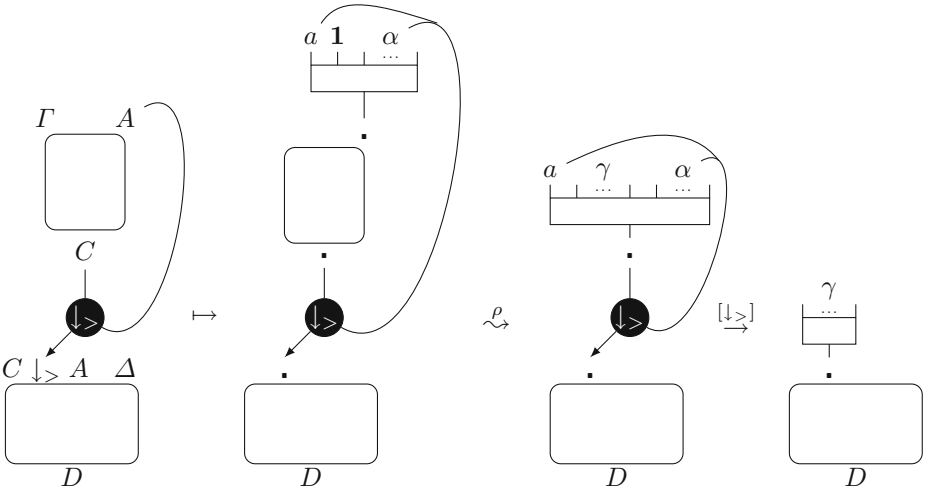


This directly gives us the natural deduction proof $\alpha : A \vdash \alpha : A$.

If there are tensor links ($t > 0$), then either one of the hypotheses or the conclusion of the proof structure must be the main formula of its link (this is easy to see since if none of the leaves is the main formula of its link, then the proof structure contains only introduction rules for $\bullet$ and $\odot_k$ and therefore the conclusion is the main formula of its link). Suppose a proof net has a leaf which is the main formula of its link and suppose this formula is $A \downarrow_> C$ (the cases of other formulas being main formulas, and of a conclusion of the proof net being the main formula are similar). Then, since all tensor trees contract to combs, we can apply the induction hypothesis to the two structures obtained by removing the tensor link and obtain proofs π_1 of $\Gamma \vdash a + \mathbf{1} + \alpha : A$ and π_2 of $\Delta, a + \gamma + \alpha : C \vdash \delta : D$ (technically, we have a proof with hypothesis $\gamma' : C$ and use substitution of the proof with conclusion $a + \gamma + \alpha : C$ shown below). We can combine these proofs as follows.

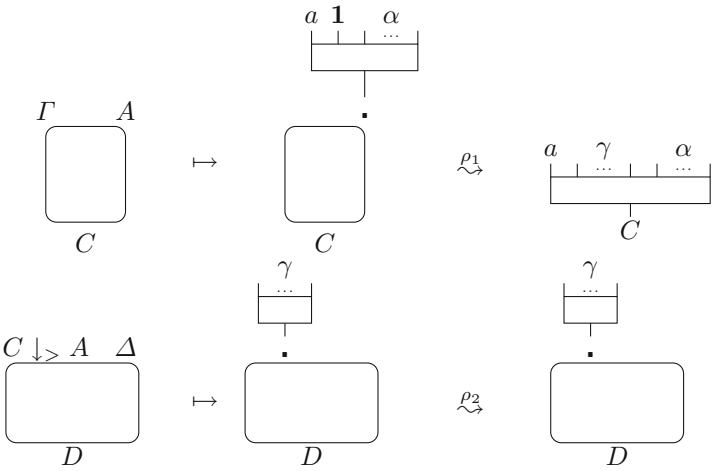
$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\ \pi 1\\ a + \mathbf{1} + \alpha : A \quad \gamma : A \downarrow_> C\end{array}}{\begin{array}{c}a + \gamma + \alpha : C\\ \vdots\ \pi 2\\ \delta : D\end{array}} \downarrow_> E \quad \Delta$$

If the sequence ρ has logical contractions ($l > 0$), we look at the last such contraction and proceed by case analysis. If the last contraction is the $\downarrow_>$ contraction, our proof net and contraction sequence look as follows.



The initial proof structure is shown above of the left and its corresponding abstract proof structure to its immediate right (note that vertex A has been replaced by $a{+}\mathbf{1}{+}\alpha$, since it is an auxiliary input, corresponding to a withdrawn hypothesis in the natural deduction proof). The reduction sequence is of the form ρ, followed by the $\downarrow_>$ contraction, possibly followed by a number of structural contractions (not displayed in the figure above).

When we remove the par link from the figure above, we are in the following situation. All contractions from ρ are either fully in the abstract proof structure shown below at the top of the picture or fully in the abstract proof structure shown below at the bottom of the picture, so ρ splits naturally in ρ_1 and ρ_2.



We need to show that $\Gamma, \Delta \vdash \delta : D$ (where $\delta : D$ is the comb). Since we have two proof nets with strictly shorter sequences of contractions, we can apply the induction hypothesis for proofs π_1 of $a{+}\mathbf{1}{+}\alpha : A, \Gamma \vdash a{+}\gamma{+}\alpha : C$ and π_2 of $\gamma : C \downarrow_> A, \Delta \vdash \delta : D$. We can combine these two proofs into a proof of $\Gamma, \Delta \vdash \delta : D$ as follows.

$$\dfrac{\dfrac{[a{+}\mathbf{1}{+}\alpha : A]_i \quad \Gamma}{\vdots\ \pi_1}}{\dfrac{a{+}\gamma{+}\alpha : C}{\gamma : C \downarrow_> A}\downarrow_> I_i \quad \Delta}$$
$$\vdots\ \pi_2$$
$$\delta : D$$

The other cases are similar and easily verified. □

Theorem 1. *A proof structure is a* proof net *iff its abstract proof structure contracts to a comb.*

Proof. Immediate from Lemmas 1 and 2. □

4 Extension to Other Connectives

One of the benefits of the current calculus is that it extends easily to other connectives, such as the unary/bracket connectives of Morrill [5, Chapter 5] (although incorporating the treatment of parasitic gapping of Sect. 5.5 would require a considerable complication of the proof theory).

The synthetic connectives of Morrill et al. [7] require us to extend our methodology somewhat: as currently formulated the proof net calculus produces a single comb for each proof net. When adding the synthetic connectives, we can introduce a separation marker in a way which is only partially specified by the premiss of the rule. For example, the denotation of (leftmost) split $\check{}A$, shown below, is the set of strings obtained by inserting a separator symbol at any place before other separator symbols (if any), and therefore the introduction rule for this connective doesn't produce a unique string term.

$$|\check{}A| =_{def} \{a + \mathbf{1} + \alpha \mid a + \alpha \in |A|\} \tag{13}$$

$$|\hat{}A| =_{def} \{a + \alpha \mid a + \mathbf{1} + \alpha \in |A|\} \tag{14}$$

This moves us to a system where a tensor tree contracts to a set of combs (or, alternatively, a partially specified comb). Apart from this, it is not hard to add links and contractions for the synthetic connectives. For example, the contraction for $\check{}$ can be obtained from the contraction for $\uparrow_k$ by removing the links to the auxiliary hypothesis: instead of replacing the auxiliary hypothesis by $\mathbf{1}$ (which defines the position of the insertion point uniquely), there will be multiple, non-confluent ways to matching the contraction and to insert the separator symbol. For lack of space, we will not develop these ideas further here.

5 Conclusion

We have presented a proof net calculus for the Displacement calculus and shown its correctness. This is the first proof net calculus which models the Displacement calculus directly and not by some sort of translation into another formalism. The proof net calculus opens up new possibilities for parsing and proof search with the Displacement calculus.

References

1. Casadio, C., Coecke, B., Moortgat, M., Scott, P. (eds.): Categories and Types in Logic, Language, and Physics: Essays dedicated to Jim Lambek on the Occasion of this 90th Birthday. LNAI, vol. 8222. Springer, Heidelberg (2014)
2. Moot, R.: Filtering axiom links for proof nets. In: Kallmeyer, L., Monachesi, P., Penn, G., Satta, G. (eds.) Proccedings of Formal Grammar (2007)
3. Moot, R.: Extended Lambek calculi and first-order linear logic. In: [1], pp. 297–330
4. Moot, R., Puite, Q.: Proof nets for the multimodal Lambek calculus. Stud. Logica **71**(3), 415–442 (2002)

5. Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press, New York (2011)
6. Morrill, G., Fadda, M.: Proof nets for basic discontinuous Lambek calculus. J. Logic Comput. **18**(2), 239–256 (2008)
7. Morrill, G., Valentín, O., Fadda, M.: The Displacement calculus. J. Logic Lang. Inform. **20**(1), 1–48 (2011)
8. Pentus, M.: Lambek calculus is NP-complete. Theor. Comput. Sci. **357**(1), 186–201 (2006)
9. Valentín, O.: The hidden structural rules of the discontinuous Lambek calculus. In: [1], pp. 402–420

Countability: Individuation and Context

Peter R. Sutton(✉) and Hana Filip(✉)

Heinrich Heine University, Düsseldorf, Germany
peter.r.sutton@icloud.com, hana.filip@gmail.com

Abstract. Much of the recent countability research agrees on the idea that a satisfactory account of *individuation* in terms of what counts as "one" unit for counting is highly relevant for characterizing a semantics of the mass/count distinction. (What counts as "one" is not necessarily a formal atom in a Boolean algebra or a natural unit associated with natural kinds like *cat*.) Taking the most parsimonious stance, our main question is: If we have a full-fledged formal theory of individuation (what counts as "one"), what data would still remain to be explained for a formal theory of the mass/count distinction? Would classical mereology be sufficient to cover such data? And, if not, what minimal extensions would be required to classical mereology to do so? We conclude that, minimally, two dimensions of context sensitivity are needed to enrich a mereological semantics of count nouns denotations. The first kind of context sensitivity enables counting operations to apply by removing overlap from an overlapping set of individuated entities. The second kind of context sensitivity allows us to motivate how a set of individuated entities can sometimes be taken as a counts as "one" despite not being a formal atom or a natural unit associated with a natural kind.

Keywords: Mass/count distinction · Mereology · Individuation · Overlap · Context sensitivity

1 Introduction: From Atomicity to Counting as 'One'

Early proposals regarding the nature of the mass/count distinction, many of which are inspired by Quine (1960), attempt to distinguish mass and count noun denotations rely on properties like cumulativity, divisivity, atomicity, and homogeneity.[1] In mereological theories, starting with Link (1983) and Krifka (1989), they were recast as second-order predicates and defined as follows:

$$AT(P) \leftrightarrow \forall x[P(x) \rightarrow \exists y \forall z[P(y) \wedge (P(z) \rightarrow \neg(z \sqsubset y))]] \quad (1)$$

$$CUM(P) \leftrightarrow \forall x \forall y[(P(x) \wedge P(y)) \rightarrow P(x \sqcup y)] \quad (2)$$

$$DIV(P) \leftrightarrow \forall x \forall y[(P(x) \wedge y \sqsubset x) \rightarrow P(y)] \quad (3)$$

Link (1983) proposes a sortal distinction between count and mass nouns, based on the atomic/non-atomic ontological distinction which is modeled by

[1] Also see Pelletier (1979) and references therein.

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 290–305, 2016.
DOI: 10.1007/978-3-662-53042-9_17

means of atomic and non-atomic semilattice structures. Count nouns are interpreted in the atomic lattice, while mass in the non-atomic one. Mass nouns pattern with plurals in being cumulative. Mass nouns and plurals also pattern alike in being divisive (down to a certain lower limit). However, none of these properties turned out to be either necessary or sufficient conditions for underpinning the mass/count distinction. For example, mass nouns such as *furniture* are atomic, count nouns such as *fence* pattern with mass terms in being divisive and cumulative (modulo contextual restrictions).

Krifka (1989) rejects Link's sortal 'dual-domain' approach on which count and mass nouns are interpreted in two different domains. Instead, his strategy is to combine the notion of a single complete join semi-lattice (a general partial order), which structures a single domain of objects, with the notion of an extensive measure function, which serves to derive quantized semantic predicates; such predicates fail to be divisive, and hence are count. The lexical semantic structure of basic lexical count nouns like *cat* contains the extensive measure function NU (for 'Natural Unit'), which does the "individuating job" of determining singular clearly discrete objects in their denotation. But this means that basic lexical count nouns are quantized, and fail to be divisive. This strategy amounts to a typal distinction between count and mass nouns.

As Zucchi and White (1996, 2001) show, there are count nouns like *twig, sequence, fence* which fail be quantized. Problematic for Krifka's account are also superordinate (aka object) mass nouns like *furniture*, because they have 'natural units' in their denotation, like stools, coffee tables, chairs. Indeed, object mass nouns like *furniture* pose problems even for more recent accounts. For example, Chierchia (2010) argues that mass nouns differ from count nouns in having vague ("unstable") minimal individuals in their denotation, but this means that he is forced to claim that the mass behavior of object mass nouns is not due to the vagueness of minimal elements in their denotation, but require an alternative explanation.

In some recent work (Rothstein 2010; Landman 2011; Grimm 2012; Sutton and Filip 2016), it has been argued that a formal representation of what counts as "one" is needed, which is not based on the notion of a 'natural unit' in any sense. For example, Rothstein (2010) argues that there are count nouns like *fence, wall* which fail to be naturally atomic, and are divisive. Their grammatical count behavior is motivated by the assumption that their denotation is indexed to counting contexts with respect to which what is "one" entity in their denotation is determined. Landman (2011) proposes that mass nouns have overlapping sets of entities that count as "one", his "generator sets", in their denotation, while count nouns have non-overlapping generator sets. Grimm, just like Rothstein and Landman, presupposes the assumptions of a classical extensional mereology, but enriches it with topological notions. Countable entities have the mereotopological property of being *maximally strongly self connected*. Sutton and Filip (2016) argue that both vagueness with respect to what counts as "one" and non-overlap in the set of entities that count as "one" interact in such a way as to either block or facilitate variation in mass/count encoding.

However, as we shall argue, even with a formal representation of what counts as "one", that is, a formal account of *individuation*, we do not have a satisfactory account of the grounding of the mass/count distinction. In this paper, we will point out some weaknesses of such an account, and seek to clarify what, in formal terms is minimally required to model the mass/count distinction in a more adequate way. Our strategy is to proceed in the most parsimonious way and answer the following fundamental questions: If we have a full-fledged formal theory of individuation (what counts as "one"), what data would still remain to be explained for a formal theory of the mass/count distinction? Would classical mereology be sufficient to cover such data? And, if not, what minimal extensions would be required to classical mereology to do so?

In Sect. 2, we identify a number of classes of nouns that have been given a lot of attention in the countability literature and outline the sense in which these nouns individuate entities into what counts as "one". We then schematize these results in mereological terms and suggest that concrete nouns form at least four individuation patterns. The result, we argue, is that the mass/count distinction cuts across the individuated/non-individuated divide. In Sect. 3, we relate key recent research into theories of individuation and the mass/count distinction to discuss which formal enrichments have been suggested as necessary for a theory of individuation. In Sect. 4, we propose that classical mereology must be minimally extended with two separate conceptions of context in order to cover the data that are intractable within formal theories relying on the notion of individuation.

2 Individuation Patterns Across Classes Of Nouns

Considering just concrete nouns, we may distinguish five classes of nouns depending on their main lexicalization patterns. They are summarized in Table 1, where the 'Noun Class' is a cover term for the descriptive labels below it.[2]

All accounts of the mass/count distinction at least aim to cover nouns that we describe as "prototypical objects" (*chair*, *boy*) and as "substances, liquids and gasses" (*mud*, *blood*, *air*). Indeed, an absolutely minimal requirement on any account of the mass/count distinction is that it can at least semantically distinguish between these two classes.

2.1 Prototypical Objects

Nouns in the *prototypical objects* category are what have been called "naturally atomic" (see e.g., Rothstein 2010, and references therein). With these nouns, the

[2] A notable omission from Table 1 are so-called "dual life" nouns such as $stone_{+C}/stone_{-C}$ ($+C$ abbreviates COUNT and $-C$ abbreviates MASS). We leave these aside in this paper, given that it is unclear whether one should take either the mass or count sense to be primary, or, indeed, whether such nouns should be classified as being of some intermediary morphosyntactic category between count and mass.

Table 1. Classes of nouns and mass/count variation

Noun class	Examples
Proto-typical objects	$chair_{+C}$; $tuoli_{+C}$ ('chair' Finnish); $Stuhl_{+C}$ ('chair' German)
	dog_{+C}; $koira_{+C}$ ('dog' Finnish); $Hund_{+C}$ ('dog' German)
	boy_{+C}; $poika_{+C}$ ('boy' Finnish); $Junge_{+C}$ ('boy' German)
Super-ordinate artifacts	$furniture_{-C}$; $huonekalu\text{-}t_{+C,PL}$ ('furniture' Finnish)
	$meubel\text{-}s_{+C,PL}$, $meubilair_{-C}$ ('furniture' Dutch)
	$kitchenware_{-C}$; $Küchengerät\text{-}e_{+C,PL}$ (German, lit. kitchen device-s)
	$footwear_{-C}$; $jalkinee\text{-}t_{+C,PL}$ ('footwear' Finnish)
Homogenous objects	$fence_{+C}$, $fencing_{-C}$; $hedge_{+C}$, $hedging_{-C}$
	$wall_{+C}$, $walling_{-C}$; $shrub_{+C}$, $shrubbery_{-C}$
Granulars	$lentil\text{-}s_{+C,PL}$; $linse\text{-}n_{+C,PL}$ ('lentils' German)
	$lešta_{-C}$ ('lentil' Bulgarian); $čočka_{-C}$ ('lentil' Czech)
	$oat\text{-}s_{+C,PL}$; $oatmeal_{-C}$;
	$kaura_{-C}$ ('oat' Finnish); $kaurahiutale\text{-}et_{+C,PL}$ (Finnish, lit. oat.flake-s)
Substances, liquids, gases	mud_{-C}; $muta_{-C}$ ('mud' Finnish); $Schlamm_{-C}$ ('mud' German)
	$blood_{-C}$; $veri_{-C}$ ('blood' Finnish); $Blut_{-C}$ ('blood' German)
	air_{-C}; $lenta_{-C}$ ('air' Finnish); $Luft_{-C}$ ('air' German)

"natural units", or the entities we would count as "one" are just the minimal entities in their denotations for which nothing else in the denotation is a proper part, i.e., they are quantized in the sense of Krifka (1989). As Table 1 shows, nouns in these classes display a strong tendency to be lexicalized as count as in confirmed by the felicity and grammaticality of direct modification with numerical expressions (such as the Finnish example in (4) and its English translation).

(4) Kolme tuoli-a/koira-a/poika-a.
Three chair-PART dog-PART boy-PART.
"Three chairs/dogs/boys".

Assuming that count nouns take their interpretation from an atomic join semilattice, the denotation of prototypical objects is represented in Fig. 1: the basic (extensional) meaning of a count noun is a number neutral property and hence its denotation comprises the whole lattice; the shaded area highlights the set of singular (atomic) entities that count as "one".

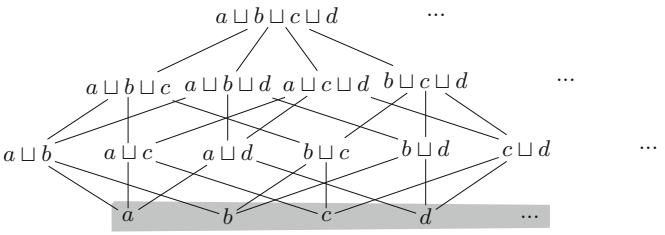


Fig. 1. Prototypical objects: Individuals are disjoint bottom elements

2.2 Substances, Liquids and Gasses

Prototypical mass nouns tend to denote *substances, liquids and gasses*. In stark contrast to prototypical objects, nouns in this class lack any clear individuable entities/units at all. That is to say, whether or not one assumes the denotations of these nouns are defined over atomic or non-atomic semilattices, there are no entities that "stand out" as individuals (things that count as "one").

Nouns in this class are very stably lexicalized as mass cross- and intralinguistically. For example, as the German example in (5) and its English translation show, direct numerical modification is infelicitous in the absence of a classifier-like expression.

(5) # Vier Lüft-e/Schlämm-e.
four air-PL/mud-PL
"Three bloods/airs".

We schematize the denotation of nouns describing substances, liquids and gasses as in Fig. 2. Unlike Fig. 1, there is no shaded area for entities that count as "one" clearly individuated atomic unit.

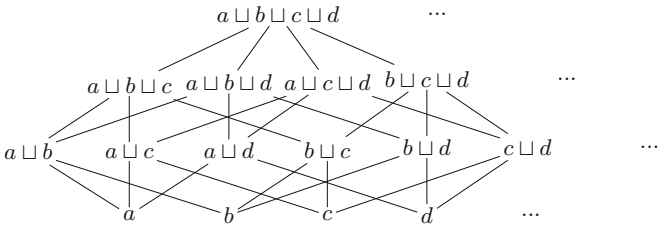


Fig. 2. Substances: No individuals

2.3 Superordinate Artifacts

Superordinate artifacts are among the three categories in between *Prototypical Objects* and *Substances, Liquids, and Gasses* nouns, the two extreme poles of a countability scale (see also Allan 1980), along with *Homogenous Objects* and *Granulars*. These three classes are far more complex than *Prototypical Objects* and *Substances, Liquids, and Gasses* and all display cross and intralinguistic count/mass variation.

Nouns in this class have been variously labelled "fake mass nouns", "superordinate aggregates", "object mass nouns", "neat mass nouns", however arguably superordinate count nouns such as *vehicle* should also fall in the same grouping. A well known puzzle with such nouns is why so many languages lexicalize them as mass nouns at all. The denotations of these nouns have clearly individuable entities at the 'bottom' of them which should *prima facie* be good candidates

for counting, and superordinacy in itself is no bar to countability (as evidenced by count nouns such as *vehicles*, *fruits* (US English), *vegetables*).

Nonetheless, one may find both mass and count nouns in this category with both cross- and intralinguistic variation. For example, the grammaticality of the Finnish example in (6) contrasts with the complete infelicity of the English (7).

(6) Kaksi huonekalu-a.
two furniture-PART
"Two items of furniture".

(7) ## Two furnitures

What counts as "one" for nouns in this category is not, however, restricted to entities at the bottom of the lattice (as is the case with prototypical objects). As observed by Landman (2011), sums of minimal entities may also count as "one". For example, for *kitchenware*, a teacup and a saucer count as "one" item of kitchenware, but in many contexts, so might a teacup and saucer together, and, for *furniture*, tables/desks, stools/chairs and mirrors all count as single items of furniture, but so does a vanity formed of one of all three. This pattern is schematized in Fig. 3. The minimal entities that count as "one" are shaded in darker gray. The rest of the lattice is also shaded, but the lighter color indicates that some, but not necessarily all sums of minimal entities count as "one".

2.4 Homogenous Objects

This class includes nouns such as *fence* and *fencing*. We use 'homogenous' here in the sense of Rothstein (2010). For example, for some entities that count as fences, proper parts of these entities themselves count as fences. Note that such nouns denote divisive predicates in the sense of (3) above (down to a certain lower limit). These nouns are cumulative, even with respect to what counts as "one". For example, if two fences are attached together, then there will be contexts in which their sum counts as a single fence.

One difference between homogenous objects and superordinate artifacts is that, at least for some nouns in this class, there seem to be fewer restrictions on what can be put together to count as "one". This derives from the homogeneity of these objects, as opposed to the more functionally defined homogenous artifacts.

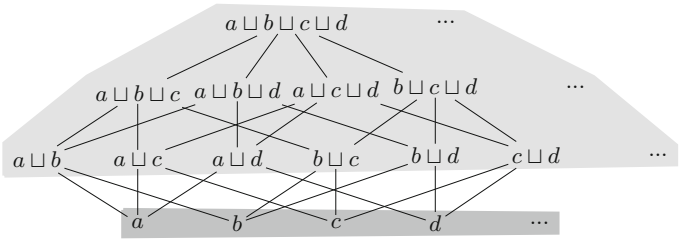


Fig. 3. Superordinate artifacts and homogenous objects: Individuals are disjoint bottom elements plus at least some sums thereof

A whisk and a teaspoon cannot, at least in ordinary contexts, count as a single item of kitchenware since the two together do not *function* as a single item. A pestle and mortar can count as a single item of kitchenware, since these two do have a joint function with respect to being a kitchenware tool. For fences and bushes, however, provided that two or more portions of fencing/hedge are relevantly similar, one can be appended to the other to make what could count as a single fence/hedge. Nonetheless, it is not the case that, for example, any two entities in the denotation of *fence* could be put together to form one fence. It is hard to imagine how a 50 cm high picket fence and a 4 m high chainlink fence could count as "one". For this reason, we schematically represent homogenous objects in the same way as superordinate artifacts in Fig. 3.

For at least some languages with count/mass counterparts in this category such as English and German, the morphologically simple item tends to be count (*hedge*, *Busch* ('bush'/'shrub', German)) and a morphologically more complex item formed from this root tends to be mass (*hedging*, *Gebüsch* ('shrubbery' German)).

2.5 Granulars

Granulars align roughly with what Grimm (2012) labels "collective aggregates". These tend to be formed of collections of similar items with food items as common examples (*rice, lentils, beans*), however other examples include *shingle*, *gravel*, *pebbles*. Their denotation most obviously, from a perceptual perspective, consists of non-overlapping grains, flakes, granules and the like. Furthermore, whenever one finds a count noun in this class, it is precisely the single flakes, grains or granules that are denoted by the singular form.

However, parts of these grains, flakes or granules are also in the number neutral denotations of these nouns. For example, rice flour counts as rice, and red lentils, after cooking down into a kind of pulp, still count as *lentils*. This pattern of the relationship between the regular denotation and what counts as "one" is schematized in Fig. 4. To reflect that these nouns are granular, the individuated entities are disjoint (non-overlapping), but are not at the bottom of the lattice. A pile of broken up rice grains or lentils still count as *rice* or *lentils*, respectively. This category also displays a large amount of count/mass variation as Table 1 helps to show.

2.6 Individuation Does Not Determine Mass/Count Encoding

With these four individuation patterns outlined,[3] it should be clear that being individuated in the sense of having entities that count as "one" or not is insufficient for determining whether nouns are count or mass. We grant that the only completely non-individuated class tends strongly towards containing only mass nouns, but superordinate artifacts, homogenous objects, and granulars all contain many examples of both mass and count nouns.

[3] We do not rule out there being more than these four.

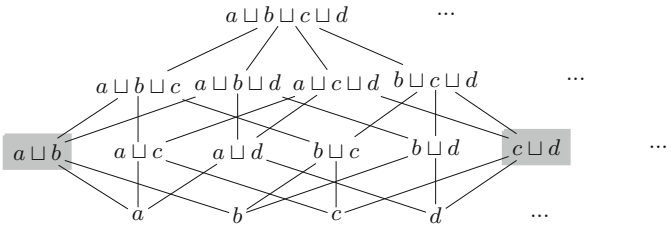


Fig. 4. Granulars: Individuals are non-bottom, disjoint individuals

Notice too, that being individuated is not the same as coming in natural units. Sums of fence units can still count as *one fence*, and collections of items of kitchenware such as pestles and mortars can still count as *one item of kitchenware* or as *ein Küchengerät* ('one (item of) kitchenware', German).

3 Enrichments Needed for a Theory of Individuation

The four patterns of individuation we have just outlined could be understood as the outcome of some fully-fledge theory of individuation. In Sect. 4, where we make the main point in this article, we will show that even after assuming such a fully fledged theory of individuation has already been given, the basic tools of mereology are insufficient for accounting for count/mass variation data. Specifically, we argue that at least two contextual parameters must be added to classical mereology and a theory of individuation to accommodate these data.

However, within the count/mass literature, these issues of individuation and the count/mass distinction are not always articulated. Also, suggestions for enrichments to mereology have been made, such as mereotopology (Grimm 2012), context sensitivity (Rothstein 2010) and vagueness (Chierchia 2010). Therefore, in this section we review some of these suggested enrichments, and point out which pertain to a theory of individuation (defined as producing at least the four patterns above), and which develop a theory of the count/mass distinction on top of and further to the issue of individuation.

The reason for pursuing this line is that the mass/count distinction cuts across the individuated/non-individuated divide, as the schemas in Figs. 1, 2, 3 and 4 show. For example, even though homogenous objects, superordinate artifacts and granulars are all individuated, some of these nouns are count, and others are mass.

3.1 Mereotopology

The schema in Fig. 1 for prototypical objects is a standard representation of the denotation for a count noun, however, given a single domain assumption on which all nouns are interpreted relative to the same domain, it actually hides a wealth of assumptions. If we assume a single general domain, with respect to

which the denotations of predicates are defined, then most common concrete count nouns will not have, as atoms at the bottom of their denotations, entities which are atoms with respect to the whole domain. This is a point one can find in the background of Rothstein (2010) where formal atoms of the domain need not be specified to give an account of the semantic atoms (things that count as 'one') for a predicate, and it is explicit in recent work from Landman (Landman 2015). However, this means that, for example, the $a, b, c, d, \ldots$ in Fig. 1 are atoms relative to a predicate, but not necessarily formal atoms in the whole domain. For example, if the domain contains chair backs, chair seats, and chair legs, the denotation of *chair* will contain, as individuated entities, entities that are, formally, sums of chair parts. Now, if one accepts this characterization, then certain arguments, such as those employed in Grimm (2012), gain some traction.

Grimm (2012) argues that classical mereology is not sufficient for characterizing individuals. This is because, in classical mereology, all sums are equal. That is to say that the sum of any two entities is considered to be a mereological entity in its own right. However, many sums are not good candidates to be individuals. For example, the sum of Donald Trump and a Blancmange sitting a thousand miles away does not make for a good candidate to be an individual. Less bizarrely, even if we were to pile an assortment of chair legs, a chair back and a chair seat together, we do not have a chair individual, or if we took a sum of twenty starlings, they would not make a flock unless reasonably proximate. Taking these considerations seriously suggests that a formal theory of individuation must involve not only mereological relations, but also topological ones (mereotopology). The need for mereotopology can also be seen in Fig. 4. For granular nouns, we have assumed in that what counts as "one" are the (whole) grains or granules. For example, single lentils count as "one" with respect to the predicate *lentil*. However, if the number neutral property for *lentil* also denotes parts of lentils and lentil flour, then only a privileged selection of sums will be whole lentils (hence the highlighting of $a \sqcup b$ and $c \sqcup d$, but not, at the same time, e.g. $a \sqcup d$ in Fig. 4). Similarly, the things we would count as "one" for homogenous objects can be affected by mereotopological factors too. If two box bushes are placed in close enough proximity, they may count as a single hedge/bush, but not if they are too far apart.

Mereotopology, as an enrichment of mereological semantics, therefore best addresses the issue of individuation (what counts as "one"). Indeed, there is good reason to assume that the four individuation patterns above could not be provided without some appeal to topological relations. For the remains of this paper, we therefore view the topology in 'mereotopology' as part of a theory of individuation. This will allow us to ask what, beyond individuation and mereology, we need to account for patterns in count/mass variation.

3.2 Vagueness and Underspecification or Overspecification

For substances, liquids and gasses, there are no entities that intuitively count as "one". This loosely reflects a general consensus in the semantics of countability literature, but the details differ. For example, Chierchia (1998, 2010), and

Rothstein (2010) appeal to vagueness or underspecification with respect to what the atoms of prototypical mass nouns are. In other words, for example, there are no entities that count as "one" *mud*, because the meaning of mud does not determine or specify such a set.

An alternative position, defended by Landman (2011) is overdetermination. On this view, it is not quite right to say that there are no entities that count as "one" for substances, liquids and gasses, but rather that there is no single set of entities that count as "one". There are, on this view, many ways one could cut the cake and so no single way to count. With more than one way to count where the answer to the question 'how many' depends on the way one does count, then counting goes wrong.

We do not wish to argue in favor of over- or underdetermination here. What matters is that these enrichments to one's model are needed already to form an account of individuation. It remains to be seen, however, whether under/overdetermination will still be needed for a theory of countability if a theory of individuation is taken as basic.

3.3 Context Sensitivity

The need for context sensitivity in a theory of individuation comes from two sources. First, there are data from Yudja, a language in which, it is argued, all nouns can be directly attached to numerals and then counted (Lima 2014). In the examples cited by Lima (2014), counting with substances, liquids and gasses, for example *apeta* ('blood', Yudja), was assessed in relation to a context. Informants were shown pictures as contexts. The portions of blood, sand etc., are clearly separated, albeit not always the same size or quantity and were reliably described these situations using direct numeral attachment to nouns in this class. Lima argues that such constructions are not coercions from mass interpretations into conventional packaged units. It is not so contentious, therefore, to assume some form of context dependency for what counts as "one" for nouns in the substances, liquids and gasses category in Yudja. That is to say that Yudja speakers do seem to individuate substances, liquids and gasses albeit in a context dependent way. Such a role for context sensitivity, namely determining when portions of substances are sufficiently spatially demarcated to count as "one", would, on our terms, be part of a theory of individuation.[4]

Second, vague, but nonetheless countable nouns such as *heap*, *fence*, and *mountain* provide some evidence for context-sensitivity in what counts as "one". Chierchia (2010), in the context of arguing that it is vague what the countable entities in the denotations of mass nouns are, claims that nouns such as *fence* are not vague in the same way. Although it might be vague what the smallest fence unit is, just as it is famously vague, via sorites arguments, at how many grains a heap becomes a non-heap, this vagueness is different from the underspecification of what the atoms for a noun such as *mud* are. The point, simply put, is that there

[4] We do not view this role for context as the same as that employed by Rothstein (2010) which specifies what is lexically accessible for counting in context.

are clear cases of mountains, fences and heaps, and so, provided that one sets what Chierchia calls a "ground context", one can be assured of, for example, a set of non-overlapping minimal fence units at every ground context. This claim strongly resembles one made by Rothstein (2010) who argues that "counting context" dependency should be encoded into the semantics of count nouns so as to capture the countability of nouns such as *fence* and *twig*. In other words, there is a element of context that determines what counts, minimally, as fences or twigs. It is important to note that the same applies for mass nouns such as *fencing*. What counts minimally, as an item of fencing may vary with context. This role for context more clearly applies to the count/mass distinction that to only a theory of individuation. We will return to this role for context in Sect. 4.

3.4 Summary

Our list of mereotopology, underspecification/vagueness, overspecification, and context sensitivity is surely not exhaustive of the formal devices needed for a full account of individuation. For example, for artifacts such as *chair* as well as superordinate artifacts such as *furniture*, some form of lexical semantic representation of function will probably be needed. However, what we have aimed to do thus far is clear the decks of some of the complications involved in the semantics of the mass/count distinction. In the next section, we assume that some account of individuation can be given and so the schemas in Figs. 1, 2, 3 and 4 can be taken for granted. As we have just seen, providing a fully-fledged theory of individuation would be a highly complex and detailed task. Therefore, our assumption that one can be adequately given is a substantial simplifying assumption.

Let us henceforth suppose that there is a fully-fledged theory of individuation that predicts the four individuation patterns from Sect. 2. This means that having entities that count as "one" is not a sufficient condition for being a count noun, but it also enables us to ascertain how far we can proceed in specifying the mass/count distinction in purely mereological terms once a theory of individuation is taken as basic.

4 Applying Mereology to the Noun Classes

4.1 Formally Characterizing the Noun Classes in Mereological Terms

Since the domain is, as standardly assumed in mereological semantics, a Boolean algebra minus the empty set, we can say the following about the relationship between the denotations of all classes of nouns and the set of entities that count as "one". We assume that $Ind(P)$, denotes the set entities that count as "one" P. This is licensed because at this point we are assuming that a fully-fledged account of individuation along with classical mereology.

On these assumptions, being a P-individual is a sufficient condition for being P:

$$\forall P \forall x[Ind(P(x)) \rightarrow P(x)] \tag{8}$$

(8) is trivially satisfied by substances, liquids and gasses, which have no entities that count as "one", and is non-trivially satisfied for all other noun classes.

Another property that can be seen is with respect to the distributional properties of mass and count nouns amongst these classes. As noted in Sect. 2, substances, liquids and gasses are in all but rare cases lexicalized as mass nouns. A notable exception is Yudja (Lima 2014), which appears to allow direct counting with all nouns. All other noun classes, which have non-empty *Ind* sets contain some nouns lexicalized as count. This suggests that individuation (being able to identify what counts as "one") is a necessary, but not sufficient condition for countability, and lacking individuated entities is a sufficient condition for being uncountable.[5] In other words, if C is a second order property of predicates (being countable), such that $\Diamond C$ means possibly countable and $\Box\neg C$ means necessarily not countable:

$$\forall P[\Diamond C(P) \leftrightarrow \exists x[Ind(P(x))]] \tag{9}$$

$$\forall P[\Box\neg C(P) \leftrightarrow \neg\exists x[Ind(P(x))]] \tag{10}$$

The intuitive explanation for why (9) should make the lexicalization of a count noun possible is that individuation, finding at least some individuals in a noun's denotation, is the first minimal step towards being able to count. Otherwise, there would be nothing to count.

However, we may also describe a property held by the three classes of nouns which display variation. In other words, for a predicate P we can describe when mass/count variation is to be expected ($\Diamond C(P) \wedge \Diamond\neg C(P)$), and, by negation of both sides of the biconditional, when it is not permitted ($\Box C(P) \vee \Box\neg C(P)$).

$$\forall P[\Diamond C(P) \wedge \Diamond\neg C(P) \leftrightarrow \exists x \exists y[Ind(P(x)) \wedge P(y) \wedge y \sqsubset x]] \tag{11}$$

$$\forall P[\Box C(P) \vee \Box\neg C(P) \leftrightarrow \forall x \forall y[(Ind(P(x)) \wedge P(y)) \rightarrow \neg y \sqsubset x]] \tag{12}$$

In words, it turns out that mass/count variation is licensed when a particular kind of relationship exists between members of the set of Ps and members of the set of P individuals. For superordinate artifacts, homogenous objects, and granulars, there are entities in the set of Ps that are proper parts of entities in the set of P individuals. Examples of this are summarized in Table 2. For prototypical objects this is not so, since, vagueness aside, parts of boys, cats and

[5] This would require taking the view that in Yudja, a more liberal view is taken on what counts as "one" such that nouns denoting, for example, *mud* would have non-empty *Ind* sets. This may be relative to some specific context only, however. The examples for counting with nouns denoting substances and liquids in Lima (2014) tend to be in contexts where there are clearly perceptual portions involved such as drops of blood.

chairs are not boys, cats and chairs, respectively.[6] It is also not the case that substances, liquids and gasses have entities in their denotations that are parts of individuals, since they have empty *Ind* sets.

The question is why should the property in (11) facilitate the possibility of either mass or count encoding and its negation in (12) prevent it? Furthermore, we may ask whether such an explanation needs to appeal to more than just mereological relations and properties. In the remains of this section, we show how far we consider one can get with mereology alone. In Sect. 4.2, we then argue that simple mereology is insufficient for providing a satisfactory analysis of countability.

Since, of recent accounts, Landman (2011) remains truest to a purely mereological account, we will adopt two further notions from Landman, namely *disjointness* (not overlapping) and *generator set*. One area of loose consensus in the semantics of countability, that was emphasized most notably in Landman (2011) is that one should not, usually, allow for overlap in the set of entities which one wishes to count. Landman's idea was that overlap leads to overdetermination of how many entities there are and that if this overlap cannot somehow be ignored, then counting goes wrong. The importance of non-overlap is also mentioned by Chierchia (2010), and via the notion of a "default" counting context in Rothstein (2010). The standard mereological notions of disjointness are given in (13) and (14):

$$DISJ(x, y) \leftrightarrow x \neq y \rightarrow x \sqcap y = \varnothing \tag{13}$$

$$DISJ(P) \leftrightarrow \forall x \forall y[(P(x) \wedge P(y) \wedge x \neq y) \rightarrow x \sqcap y = \varnothing] \tag{14}$$

Generator sets in Landman (2011), informally, play the role of what we have referred to here as *Ind* sets, namely the sets of entities that count as "one". However, Landman (2011) formally characterizes them in the following way:

> A generating set for X is a set $\mathbf{gen}(X) \subseteq X - \{\varnothing\}$ such that:
> $\forall x \in X : \exists Y \subseteq \mathbf{gen}(X) : x = \sqcup Y$

In other words, the a generator set should, when closed under sum, yield the set it generates. These two notions can be put together to form two reasonable *countability norms* which are inspired by Landman's (2011) notions of *disjointness* and *generator sets*:

Disjointness Condition: *Ind* sets should be (non-trivially) disjoint.

Generator Condition: *Ind* sets should be generator sets.

Now, provably, the property in (11) entails that one or the other of these norms must be violated. This should be clear from Figs. 3 and 4, but can also be demonstrated. The right hand side of (11) can only be satisfied if $x \neq y$, since $y \sqsubset x$. Either (i) $y \in Ind(P)$ or (ii) $y \in P$ and $y \notin Ind(P)$. If (i), then there are $x, y \in Ind(P)$ such that $y \sqsubset x$. Since $y \sqsubset x \leftrightarrow x \sqcap y \neq \varnothing$, it is not the case that $DISJ(Ind(P))$. If (ii), then there is a $y \in P$ and an $x \in Ind(P)$ such that

[6] This is, again, not absolute. For example, *table* seems to behave closer to fence insofar as two tables pushed together can count as "one" table.

Table 2. Examples of individuals and parts of individuals in the denotation of predicates

Noun class	Examples	Individuals	Parts of individuals
Superordinate artifacts	*kitchenware*, *Küchengerät-e*$_{+C,PL}$ ('kitchenware', German)	pestle, mortar, pestle $\sqcup$ mortar	pestle, mortar
Homogenous objects	*fence*, *fencing*	fence$_1$, fence$_2$ fence$_1 \sqcup$ fence$_2$	fence$_1$, fence$_2$
Granulars	*rice*, *lentil-s*	grains of rice lentils	parts of grains, rice flour parts of lentils, lentil flour

$y \sqsubset x$ and $y \notin Ind(P)$. If $Ind(P)$ is disjoint, then there is also no $z \sqsubset y$ such that $z \in Ind(P)$. Hence, there can be no subset of $Ind(P)$, Y such that $\sqcup Y = y$, hence $Ind(P)$ is not a generating set for P.

Homogenous objects (*fence*) and superordinate artifacts (*furniture*) have the potential of being countable, since they do have non-empty *Ind* sets. However, countability can be undermined unless the overlap in their *Ind* sets is somehow ignored.

Granulars (*rice, lentils*) have the potential of being countable, since they have non-empty *Ind* sets. However, countability can be undermined because their *Ind* sets do not generate the entirety of their denotation (i.e. do not generate anything 'below' the granular level).

4.2 Beyond Mereology

It is at this point that it seems that a purely mereological analysis gives out. Superordinate artifacts and homogenous objects breach the *disjointness condition*, but some nouns in these classes are count nouns. Granulars breach the *generator condition*, but some nouns in these classes are count nouns.

In cases where the *disjointness condition* is violated, a noun may still be lexicalized as count (as with *fence* and *huonekalu* ('furniture', Finnish)) if some mechanism exists by which one can ignore overlap in an *Ind* set in such a way as to ensure only one counting result in any given context. At the very least, this seems to require that count nouns in the homogenous objects (*fence*) and superordinate artifacts (*huonekalu*) groups must be indexed in some way to either something akin to Rothstein's counting contexts (Rothstein 2010), or to a formal device such as Landman's "variants", where a variant of a generator set is a maximally disjoint subset (Landman 2011). Although a maximally disjoint subset (variant) can be specified in mereological terms, we nonetheless require a formal device which applies a single variant to an *Ind* set at a given occasion. For example, if a teacup and saucer count as two items of kitchenware on one

occasion, then this formal device should make it possible that the cup and saucer sum count as a single item on another occasion.

In cases where the *generator condition* is violated, a noun may still be lexicalized as count (as with *lentil*) if one has a formal device that can explain why an *Ind* set can sometimes be taken as a counting base despite not generating the whole nouns denotation. We argued in Sutton and Filip (2016), that an explanation of this phenomena will need to be derived from a distinct form of context-sensitivity such as the one that underpins Chierchia's vagueness-based account of the mass/count distinction (Chierchia 2010). Chierchia points towards examples where what we have identified as the individuated units of granulars (the single grains, flakes etc.) fall in and out of the extensions of granular predicates depending on context. For example, a single grain of rice is sufficient in quantity to count as *rice* in contexts where someone has an allergy, but insufficient to count as *rice* when one is making paella. While this evidence is not sufficient to fully justify supervaluationism as applied by Chierchia, it does suggest that the *Ind* set for granulars is on looser grounds that the *Ind* sets in other categories. For example, for prototypical objects, homogenous objects, and superordinate artifacts, if $x \in Ind(P)$ at a context c, then $x \in P$ at all contexts c', however, for nouns such as rice, there are at least some contexts in which, if $x \in Ind(P)$ at c, it is possible that $x \notin P$ at some context c'. However, this does not imply that the individuated units for granulars are underspecified or vague. Minimally, therefore, we require another index to another form of context, or else a richer story about how extensions of granular nouns can vary from use to use.

5 Conclusions

In this paper we have identified a number of complications that arise in trying to establish the right formal tools to model the mass/count distinction in concrete nouns. We have argued that, as important as a theory of individuation is, the development of a theory of individuation does not entail that one has a theory of the mass/count distinction. Furthermore, we have argued that, even if one assumes an independently motivated theory of individuation, classical mereology is still insufficient for capturing the puzzling mass/count variation data for superordinate artifacts, homogenous objects and granulars.

We have concluded that, minimally, two dimensions of context sensitivity need to be incorporated into a mereological semantics. The first to formally remove overlap from an overlapping *Ind* set, the second to track the way in which the truth conditions of, especially, granular nouns shift in such a way as to change the position of the *Ind* set in the lattice or even remove it from the noun's denotation entirely in some situations.

Acknowledgements. This research was funded by the German Research Foundation (DFG) CRC991, project C09.

References

Allan, K.: Nouns and countability. Language **56**(3), 541–567 (1980)
Chierchia, G.: Plurality of nouns and the notion of semantic parameter. In: Rothstein, S. (ed.) Events and Grammar, pp. 53–103. Springer, Heidelberg (1998)
Chierchia, G.: Mass nouns, vagueness and semantic variation. Synthese **174**, 99–149 (2010)
Grimm, S.: Number and individuation. Ph.D. dissertation, Stanford (2012)
Krifka, M.: Nominal reference, temporal constitution and quantification in event semantics. In: Bartsch, R., van Benthem, J.F.A.K., van Emde Boas, P. (eds.) Semantics and Contextual Expression, pp. 75–115. Foris Publications, Dordrecht (1989)
Landman, F.: Count nouns – mass nouns – neat nouns – mess nouns. Baltic Int. Yearb. Cogn. **6**, 1–67 (2011)
Landman, F.: Iceberg semantics for count nouns and mass nouns: the evidence from portions. Handout, Presentation at Henirich Heine University Düsseldorf (2015)
Lima, S.: All notional mass nouns are count nouns in Yudja. Proc. SALT **24**, 534–554 (2014)
Link, G.: The logical analysis of plurals and mass terms: a lattice-theoretic approach. In: Portner, P., Partee, B.H. (eds.) Formal Semantics - The Essential Readings, pp. 127–147. Blackwell, Oxford (1983)
Pelletier, F.J. (ed.): Mass Terms Some Philosophical Problems. D. Reidel Publishing Company, Holland, Boston (1979)
Quine, W.V.: Word and Object. The MIT Press, Cambridge (1960)
Rothstein, S.: Counting and the mass/count distinction. J. Semant. **27**, 343–397 (2010)
Sutton, P.R., Filip, H.: Vagueness, overlap, and countability. In: Proceedings of Sinn und Bedeutung 20 (2016, forthcoming)
Zucchi, S., White, M.: Twigs, sequences and the temporal constitution of predicates. In: Galloway, T., Spence, J. (eds.) Proceedings of SALT, vol. 6 (1996)
Zucchi, S., White, M.: Twigs, sequences and the temporal constitution of predicates. Linguist. Philos. **24**(2), 223–270 (2001)

The Proper Treatment of Linguistic Ambiguity in Ordinary Algebra

Christian Wurm and Timm Lichte(✉)

University of Düsseldorf, Düsseldorf, Germany
{cwurm,lichte}@phil.hhu.de

Abstract. We present a first algebraic approximation to the semantic content of linguistic ambiguity. Starting from the class of ordinary Boolean algebras, we add to it an ambiguity operator $\|$ and a small set of axioms which we think are correct for linguistic ambiguity beyond doubt. We then show some important, non-trivial results that follow from this axiomatization, which turn out to be surprising and not fully satisfying from a linguistic point of view. Therefore, we also sketch promising algebraic alternatives.

1 Introduction

The term LINGUISTIC AMBIGUITY designates cases where expressions, or exponents, of natural language give raise to two or more sharply distinguished meanings.[1] Ambiguity is a truly pervasive phenomenon in natural language, and therefore perhaps also a very heterogeneous one. It affects single-word expressions such as `bank` ('financial institute', 'strip of land along a river'), multi-word expressions such as `kick the bucket` ('kick the bucket', 'die'), VPs with varying PP attachment (`see the man with the telescope`), as well as larger syntactic units up to full clauses, e.g. `every boy loves a movie` with its two readings of quantifier scopes.

To deal with ambiguity of this kind, say an exponent e with two meanings m_1, m_2, two general approaches are at choice. The first approach (aka. SYNTACTIC APPROACH) is the following: we have two separate entities $(e,\ m_1)$ and $(e,\ m_2)$ (these are form-meaning pairs, which we henceforth call SYMBOLS). These are unrelated, except for the fact that (for whatever reason) they turn out to have the same exponent. If we encounter e, we do not know which of the two entities we have; so we just pick one and proceed. Importantly, in the meaning component itself, there is no ambiguity; it only occurs at the point where we choose one of the two symbols. This process is however non-deterministic, and strictly speaking, there is no function from form to meaning. The second approach (aka. SEMANTIC APPROACH) is to assume the existence of one ambiguous symbol $(e,\ m_1 \| m_2)$, which has a genuinely ambiguous meaning. This comes

In the originally published version there is an error in the proof of Lemma 1. The erratum to this chapter is available at https://doi.org/10.1007/978-3-662-53042-9_19

[1] This roughly distinguishes ambiguity from cases of vagueness [8].

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. 306–322, 2016.
DOI: 10.1007/978-3-662-53042-9_18

with a number of technical advantages: there is a function from form to meaning, lexical entries do not multiply, there are no questionable choices, and we always keep track of possible ambiguities during the process of meaning composition.

There is, however, one big question: as in the second approach, ambiguity enters into semantics proper, we have to ask ourselves: what does it actually *mean*? In this paper, we will approach this question in an algebraic fashion. In doing so, we will see that the answer is far from obvious, and comes with a number of surprising insights.

2 Linguistic Ambiguity

The ambiguity of linguistic exponents is commonly treated by mixing syntactic and semantic approaches – and it is sometimes difficult to sort out which of the two should be preferred given a specific exponent. Still there seem to be cases where there is inherent reason (besides the technical advantages mentioned above) to choose only the semantic approach. One sort of evidence is the concurrent and persistent availability of different meanings of one exponent, such as in (1) taken from [3]:

(1) a. `The federal agency decided to take the project under its well-muscled, federally-funded wing.`
b. `We pulled his cross-gartered leg.`

In (1-a), the modifiers `well-muscled` and `federally-funded` refer to different meanings of `wing`, namely either its literal or idiomatic meaning. This is clearly out of reach for any syntactic approach. The same holds for cases of "conjunctive modification" (Ernst) such as in (1-b). Here, `cross-gartered` modifies the literal meaning of `leg`, while `pulled leg` is interpreted idiomatically. Another sort of evidence in favor of choosing a semantic approach comes from psycholinguistic experiments (e.g. [7,12]) that suggest that, in the face of ambiguity, processing costs rather emerge in the semantics than in syntax. So even if we don't claim that all cases of linguistic ambiguity must be treated within a semantic approach, there's certainly no way around a notion of ambiguity that is genuinely semantic. Therefore, and for the sake of exposition, we want to apply the semantic approach as generously as possible in what follows.

It seems furthermore clear to us that we want to draw inferences from ambiguous statements, even if we cannot disambiguate them. Hence, we do not assume that disambiguation strictly precedes the inferencing step, as does [10]. So given an ambiguous sentence such as in (2), taken from [10], it is clearly possible to infer 'there is a big car in New York' without disambiguating the meaning of `strikes`:

(2) `The first thing that strikes a stranger in New York is a big car.`

One might think that this is no argument in our favor, because the fact that there is a big car in New York obviously follows from both readings of (2). But the example actually makes a strong point: since we can easily and obviously

infer things from ambiguous statements, we have to clarify what the meaning of ambiguity actually *is*, at least from an inferential point of view.

Usually, there is a quick answer to the question what ambiguity means, namely: ambiguity means the same as disjunction. This is correct in the sense that, if we get an ambiguous meaning $m_1 \| m_2$, then the only thing we can infer for sure is $m_1 \vee m_2$ ($\vee$ being disjunction). However, a quick argument shows that equating $\|$ and $\vee$ is inadequate even in simple logical contexts. For example, assume an ambiguous meaning $m_1 \| m_2$ that is in the scope of a negation. If we treat $\|$ as $\vee$, then the meaning of $\neg(m_1 \| m_2)$ would be $\neg m_1 \wedge \neg m_2$ – which is obviously wrong, because intuition clearly tells us that $\neg(m_1 \| m_2) = \neg m_1 \| \neg m_2$, for example:

(3) `There is no bank.`

This sentence is ambiguous between the meanings of `there is no financial institute` and `there is no strip of land along the river`, but it surely does not mean that there is neither of the two. The same holds if we have $m_1 \| m_2$ in the left-hand side of an implication etc.[2] The semantic treatment of ambiguity also requires a proper behavior in the course of meaning composition, which is definitely not granted by treating it as disjunction. Hence treating ambiguity as disjunction not only runs counter to most basic intuition, but also destroys the parallelism between the two approaches to ambiguity, whereas they should be equivalent (or at least parallel) at some level. Another problem we only briefly mention is the following: we often find ambiguities between meanings m_1, m_2 where m_1 entails m_2 (as in the two readings of `every boy loves a movie`). In this case however, $m_1 \vee m_2$ is obviously equivalent to m_2 – so there would be no ambiguity in the first place! This, however, runs counter to any intuition and again destroys the parallelism between the two approaches to ambiguity. To sum up, we think it is completely inadequate to think of ambiguity in terms of disjunction.

Considering all this, it is surprising that until now, there has not been any serious work on the *proper* meaning of ambiguity. Of course, there are ample works on aspects of linguistic ambiguity in terms of parasemantic underspecification (see, e.g., [1], [2, Chapter 10], [9,10]), but none of them, to the best of our knowledge, addresses the meaning of ambiguity on the level of object terms. This is what we want to start in this paper. Obviously, this work can be only preliminary; we will start by considering a very particular case, namely what happens when we add an operator $\|$, which satisfies laws we consider adequate for ambiguity, to a Boolean algebra. So we consider Boolean algebras with an additional operator and with some additional axioms fixing its properties. As we will see, it is sufficient to provide a very small set of axioms which we think are

[2] [10] delivers another example for the "disjunction fallacy" based on intensional predicates: given a sentence S that is ambiguous between P and Q, the following should hold if S meant $P \vee Q$: $[A \text{ means that } S] = [A \text{ means that } [P \vee Q]] = [[A \text{ means that } P] \vee [A \text{ means that } Q]]$. However, this is obviously not the case. See also [11] and [9] for similar ideas and examples.

correct for ambiguity beyond doubt. From these axioms already follow a lot of other properties we think are correct for ambiguity (in fact, all we can think of). On the other side, we can also derive many more strong properties no one would think are generally true for ambiguity. In this sense, the results of this paper are not fully satisfying. However, they leave us with an interesting puzzle and are highly relevant to anyone who thinks that a semantic approach to ambiguity is worth pursuing. Note that all results are algebraic in nature and thus very general, and by no means bound to any particular notion of meaning, except for the assumption that logical operations are Boolean.

The structure of the rest of this paper is as follows: first, we discuss some fundamental properties of $\|$ (that is, semantic ambiguity). This provides the theoretical/semantic motivation for our axiomatization of ambiguous algebras, which is presented in the next section. Subsequently, we present the most important results on ambiguous algebras, which are surprisingly strong, and finally, we discuss what it all means and what purpose it might serve.

3 The Semantics of Linguistic Ambiguity

First let us discuss the relation between $\|$ and $\vee$. It is not by chance that, in approaches so far, ambiguity is mostly treated as a disjunction. On thing is obvious: $a\|b$ entails $a \vee b$, and moreover, $a \vee b$ is the *smallest* Boolean object generally entailed by $a\|b$. That the two are *not* identical can be seen from the following considerations: by definition, a entails $a \vee b$ and b entails $a \vee b$. However, it is surely not generally true that a entails $a\|b$: 'financial institute' does not generally entail 'bank'; in some uses it does, in other it does not. This clearly depends on the question in which sense the speaker *intends* an expression with meaning $a\|b$ – he could for example intend b by it. So, $a\|b$ has the following strictly weaker property: either a entails $a\|b$, or b entails $a\|b$. Which one of the two holds depends on something we might call the speaker's "INTENTION", but the latter is nothing we can directly observe. The notion of intention of course is very difficult to grasp, and serious philosophers have done major work just in order to clarify this concept (we just mention [4,13]). This is not the place to dwell on this notion, so we just clarify our point with an example: suppose speaker A utters:

(4) `I need some pastry or some money!`

Then no matter which one of the two you bring him, you surely have fulfilled his desire. Conversely, suppose speaker B utters:

(5) `I need some dough!`

If you bring him some pastry, he might say: `But I needed some money!`; conversely, if you give him money, he might say: `But I just needed pastry!`. He might of course also be satisfied with either of the two, he might even be satisfied with just a cup of coffee instead, but that is beside the point. The point is: by uttering `dough`, he is committed to either 'money' or 'pastry', but just

one of the two in particular, and not an arbitrary one of the two.[3] The notion of ambiguity has an epistemic flavor which the truth-functional connectives do not have at all. This will correspond with the fact that $\|$, contrarily to Boolean operators, cannot be defined in terms of truth-functions (if it could, it would in fact be redundant, as all binary truth functional operators can be defined by $\wedge$, $\neg$). Therefore, it is most natural to consider $\|$ in an algebraic (rather than a logical) setting, and it seems most natural to us to start with Boolean algebras, because they correspond to classical propositional logic. Though the correspondence of Boolean algebras and classical logic is close, it is important to keep in mind that all formal treatment in this paper will be algebraic, not logical (we therefore use $\sim$ for algebraic complementation, not $\neg$ as is generally used for logical negation). Nonetheless, the algebraic relation $\leq$[4] can be intuitively read as logical entailment, and the algebraic equality $=$ as logical equivalence. Both will help the intuitive understanding and hardly do any harm, and consequently, we will use $a \leq b$ and "a entails b" with a parallel meaning, the former being the algebraic counterpart of the latter. To connect the algebras that we are about to introduce properly to intuition, it is important to keep the following in mind: the objects of the algebra are supposed to be *meanings*; we are completely agnostic to what they actually are and whether they have any internal structure. These meanings are related by the relations $\leq$ (corresponding entailment, definable in terms of $\wedge, \vee$), and can be combined by means of Boolean connectives and $\|$.

Having said this, we can state some intuitively uncontroversial properties: it is obvious that $a \wedge b$ entails $a\|b$, and algebraically, $a \wedge b$ is the largest element generally entailing $a\|b$. For example, in the particular case where a entails b, this means: a entails $a\|b$, and $a\|b$ entails b, but $a\|b$ does *not* entail a. This is one important property; the other important property we need is what we call the property of UNIVERSAL DISTRIBUTION. This means that[5]

$$\sim (a\|b) = \sim a\| \sim b \tag{6}$$

$$(a\|b) \vee c) = (a \vee c)\|(b \vee c) \tag{7}$$

$$(a\|b) \wedge c) = (a \wedge c)\|(b \wedge c) \tag{8}$$

$$(a\|b) \rightarrow c = (a \rightarrow c)\|(b \rightarrow c) \tag{9}$$

$$a \rightarrow (b\|c) = (a \rightarrow b)\|(a \rightarrow c) \tag{10}$$

This property of universal distribution is what makes the semantic approach to ambiguity parallel to the syntactic approach. Logically speaking, this means that $\|$ is SELF-DUAL. Intuitively, this is clear, because for every ambiguous object, usually one meaning is intended, and this property is preserved over construction of arbitrary Boolean terms. What is very peculiar about $\|$ is that it preserves over negative contexts such as negation or the left-hand side of implication.

[3] This also makes clear that ambiguity cannot be interpreted, algorithmically speaking, as non-deterministic choice (which corresponds to disjunction), as we cannot pick an arbitrary meaning.

[4] $\leq$ in Boolean algebras is defined in terms of $\wedge$ (or $\vee$): $a \wedge b = a$ iff $a \leq b$ iff $a \vee b = b$.

[5] Here we use connectives $\wedge$, $\vee$, $\sim$, $\rightarrow$ in their Boolean meaning, but in principle this would not make a difference.

There are some more properties of $\|$ we should mention. One which should be clear is associativity, that is:

(ass) $$(a\|b)\|c = a\|(b\|c)$$

This should go without comment, as idempotence:

(id) $$a\|a = a$$

What is more problematic is commutativity, that is:

(com) $$a\|b = b\|a$$

One might object that meanings are ordered due to the existence of PRIMARY and SECONDARY MEANINGS (cf. the distinction between literal and idiomatic meaning), so commutativity should be rejected. One might alternatively think that in ambiguity, all meanings have the same status, so $\|$ should be commutative. We will see, however, that the latter assumption is unwarranted in our algebraic approach, as ambiguous algebras with commutative $\|$ are necessarily trivial, that is, they have only one element (so there would be only one meaning).

4 Ambiguous Algebras

4.1 Uniform Usage and Axiomatization

If we want to define $\|$ – that is, ambiguity – as an algebraic operation, we have to be aware that in any case, it is a function. This is already a strong commitment, because this implies that for a given linguistic context, in which we use a particular ambiguous algebra,[6] we have a UNIFORM USAGE. This can be justified by the hypothesis of uniform usage (UU):

(UU) In a given context, an ambiguous statement is used consistently in *only one* sense.

That might seem too strong, but is a prerequisite for any algebraic treatment of ambiguity. Note that this is before all axiomatization, and only concerns the fact that we conceive of $\|$ as an operation in an algebra.

A central notion of this paper is the one of a BOOLEAN ALGEBRA, which is a structure $\mathbf{B} = (B, \wedge, \vee, \sim, 0, 1)$. We use the convention that algebras are denoted by boldface letters, whereas their carrier sets are denoted by the same letter without boldface. As Boolean algebras are most well-known, we do not introduce them (the reader interested in background might consider [5,6], or many other sources). In this paper, we will only use elementary properties of

[6] Linguistically, it is of course unclear how to determine such a context. We would (vaguely) say it is a discourse, but of course this is arguable. Instead of being vague we could also be circular and say: such a context is a discourse where ambiguous terms are used consistently in one sense.

Boolean algebras, these however frequently and without proof or explicit reference. We now provide an axiomatization for ambiguous algebras. An AMBIGUOUS ALGEBRA is a structure $\mathbf{A} = (A, \wedge, \vee, \sim, \|, 0, 1)$, where $(A, \wedge, \vee, \sim, 0, 1)$ is a Boolean algebra, and $\|$ is a binary operation for which the following holds:

$$(\|1) \qquad \sim (a\|b) = \sim a\| \sim b$$

$$(\|2) \qquad a \wedge (b\|c) = (a \wedge b)\|(a \wedge c)$$

$$(\|3) \qquad \text{At least one of} \quad a \leq a\|b \quad \text{or} \quad b \leq a\|b \quad \text{holds}$$

This is sufficient for us: note that this already entails all Eqs. (6)–(10). As is well-known, $\vee$ and $\rightarrow$ are redundant (and the latter will furthermore play no role), and to see that $(a\|b) \vee c = (a \vee c)\|(b \vee c)$, just consider that

$$\begin{aligned}(a\|b) \vee c &\equiv \sim (\sim (a\|b) \wedge \sim c) \\ &= \sim ((\sim a\| \sim b) \wedge \sim c) \\ &= \sim ((\sim a \wedge \sim c)\|(\sim b \wedge \sim c)) \\ &= (\sim (\sim a \wedge \sim c))\|(\sim (\sim b \wedge \sim c)) \\ &\equiv (a \vee c)\|(b \vee c)\end{aligned}$$

We will see in the structure theory that (ass) and (id) are also derivable from these axioms. As is usual, such an axiomatization comes with a number of questions, to which we shortly provide the following answers (details are to be found in the subsequent sections):

1. Do these axioms entail all properties we find intuitively true for ambiguity?
   – As far as we can see, clearly yes.
2. Do they imply some properties we find intuitively incorrect for ambiguity in general?
   – Unfortunately, also clearly yes.
3. Do non-trivial algebras exist which satisfy these axioms? (That is, for example, algebras with more than one element?)
   – Clearly yes, but if we add commutativity for $\|$, then no.
4. Are there ambiguous algebras, where $a\|b \neq a$ and $a\|b \neq b$?
   – No, there are not.

Note, by the way, that ambiguous algebras have a very peculiar axiom, namely ($\|3$), which is a disjunction. This fact entails that there is no such thing as the *free ambiguous algebra* over a given set of generators,[7] and so one of the most important concepts of general algebra is not applicable. Put differently, an algebra (over some set M) in which only the inequations hold which hold in all

[7] We do not explain these concepts here, as, to the algebraist, they are clear anyway, and for the non-algebraist they are of no relevance in this paper.

ambiguous algebras, is not ambiguous algebra! However, we will see later that for every generator set, there are exactly *two* "free" algebras, that is, ambiguous algebras with a minimal set of equations that hold.

Though this is strange to the algebraist, it nicely models the epistemic component of ambiguity: if we are confronted with a certain algebra, there *always* hold certain equalities we cannot deduce from general considerations, as in an ambiguous expression, there is always one intended meaning we cannot uniquely construct.

We now show a simple, non-trivial ambiguous algebra. Take the obvious Boolean algebra over the set $\{0, a, b, 1\}$. Put

$$\begin{array}{llll} a\|b = a; & b\|a = b; & 0\|a = 0; & 1\|a = 1; \\ a\|1 = a; & b\|1 = b; & 0\|b = 0; & 1\|b = 1; \\ a\|0 = a; & b\|0 = b; & 0\|1 = 0; & 1\|0 = 1; \end{array}$$

We can see that $\wedge$-distribution holds: $a = a\|b = (a\|b) \wedge a = a\|0 = a$. $0 = 0\|1 = (0\|1) \wedge a = 0\|a = 0$, and so on, same for $\vee$, $\sim$. We thus have a proper non-trivial 4-element algebra. The results of the next section will show that this is (up to isomorphism) one of exactly two 4-element ambiguous algebras; in fact, by fixing $a\|b = a$, we have fixed the value of $\|$ for *all* arguments.

5 Structure Theory I: Uniformity

We now show the most important results which follow from our axiomatization; in this section we presuppose some familiarity with the elementary theory of Boolean algebras:

Lemma 1. *In every ambiguous algebra* $\mathbf{A}$ *and for all* $a, b \in A$*, either* $a = a\|b$ *or* $b = a\|b$*.*

Proof. Case 1: Assume that $b \not\leq a\|b$. Then $a \leq a\|b$, hence $\sim a \geq \sim(a\|b) = \sim a\|\sim b$. Now by ($\|3$), there are two subcases:

Case 1a: $\sim a \leq \sim a\|\sim b$. Then by another complementation, $a\|b \leq a$, so $a\|b = a$.

Case 1b: $\sim b \leq \sim a\|\sim b$. Then by iterating complementation, we have $a\|b \leq b$, hence $a \leq a\|b \leq b$. Hence $(a\|b) \vee a = (a \vee a)\|(a \vee b) = a\|b$ – which by definition of $\leq$ means that $a\|b \leq a$. Hence $a\|b = a$.

Case 2: $a \not\leq a\|b$ – this case is parallel to case 1.

Case 3: Assume that $a, b \leq a\|b$. Then $a \vee b \leq a\|b$, and so $\sim a\|\sim b \leq \sim a, \sim b$. By our axioms, we then must have $\sim a\|\sim b = \sim a$ or $\sim a\|\sim b = \sim b$; assume the former. Then $a = a\|b$ by an iterated complementation. Parallel if we assume the latter. $\dashv$

It is now easy to see why we cannot reasonably have an axiom for commutativity for $\|$: assume we have $a\|{\sim}a = a$.[8] Then ${\sim}a\|a = a$ as well; but also ${\sim}({\sim}a\|a) = {\sim}\sim a\|{\sim}a = a\|{\sim}a = a$, hence ${\sim}a = a$, which only holds in 1-element algebras. A parallel argument can obviously be applied if $a\|{\sim}a = {\sim}a$.

Corollary 2. *If* $\mathbf{A}$ *is an ambiguous algebra such that for all* $a, b \in A$, $a\|b = b\|a$, *then* A *has at most one element.*

The next result also follows in a straightforward fashion:

Corollary 3. *For all ambiguous algebras* $\mathbf{A}$, $a, b \in A$, *we have*

1. $a\|a = a$
2. $a \wedge b \leq a\|b \leq a \vee b$

So we have two more desired properties of $\|$ which follow from our axiomatization. The next result is more difficult to obtain and is the first one in a series of results which are stronger than our intuition on ambiguity.

Lemma 4 *(Monotonicity of* $\|$*).* *Assume* $a' \leq a$ *and* $b' \leq b$. *Then* $a'\|b' \leq a\|b$.

Proof. There are four possibilities for the values of $a\|b$ and $a'\|b'$. Two of them trivially entail the claim, and the other two are parallel. So assume without loss of generality that $a\|b = a$ whereas $a'\|b' = b'$. We then have

$$(a'\|b') \vee a = a\|(a \vee b') = a \vee b' \tag{11}$$

Conversely, we have

$$a = a\|b = (a\|b) \wedge (a \vee b') = a\|(b \wedge (a \vee b')) = a\|((a \wedge b) \vee b') \tag{12}$$

and consequently

$$(a\|((a \wedge b) \vee b')) \vee a = a\|((a \wedge b) \vee b' \vee a) = a\|(a \vee b') = a \tag{13}$$

So we have (by (11) and (13)) $a = a\|(a \vee b') = a \vee b'$, which by definition means that $b' \leq a$, hence $a'\|b' = b' \leq a = a\|b$. ⊣

This result is seemingly strong, but it is only an intermediate step in the proof of the still stronger uniformity lemma. However, it already has the following immediate consequence: it means that ambiguity roughly behaves in a *logical way*, respecting logical entailment. In fact, if it where wrong, then there would be hardly any way to think of $\|$ as a (non-classical) logical connective, because it would not necessarily respect any consequence relation. Secondly, the result has an intuitive semantic meaning: it means that if we assume the weak hypothesis

[8] This term is very important for the following proofs. One might argue that ambiguity of this kind does not arise in natural language, an argument which leads to PARTIALLY AMBIGUOUS ALGEBRAS, which we discuss shortly later on. On the other side, there are words such as `sacré` in French which – though in different contexts – can both mean 'cursed' and 'holy'.

of uniform usage (UU), then this already entails a very strong uniform usage, namely if we use an ambiguous term s with meaning $a\|b$ in the sense of a, then we must use all terms with a meaning which is related by implication in a way which is consistent with this usage. This is a strong surprising feature which comes for free, given UU. In the following, we will strengthen this. First, take the following list of results:

Lemma 5. *Let* **A** *be an ambiguous algebra, $a \in A$. If $a\|{\sim}a = a$, then*

1.	${\sim}a\|a = {\sim}a$	*5.*	$0\|{\sim}a = 0$	*9.*	${\sim}a\|0 = {\sim}a$
2.	$1\|a = 1$	*6.*	$a\|1 = a$	*10.*	$0\|1 = 0$
3.	$0\|a = 0$	*7.*	$a\|0 = a$	*11.*	$1\|0 = 1$
4.	$1\|{\sim}a = 1$	*8.*	${\sim}a\|1 = {\sim}a$		

Proof. 1. follows by negation distribution; 2. is because $({\sim}a\|a) \vee a = 1\|a = {\sim}a \vee a = 1$. Results 3.–9. follow in a similar fashion from the distributive laws. To see why 10. holds, assume conversely that $0\|1 = 1$. Then we have

$$1 \wedge a = (0\|1) \wedge a = (0 \wedge a)\|(1 \wedge a) = 0\|a = a \tag{14}$$

– a contradiction to 3. 11. follows by distribution of ${\sim}$. ⊣

Obviously, this lemma has a dual where $a\|{\sim}a = {\sim}a$, and where all results are parallel.

Lemma 6. *Let* **A** *be an ambiguous algebra. If for an arbitrary $a \in A$, we have $a\|{\sim}a = a$, then for all $b, c \in A$ we have $b\|c = b$; conversely, if we have $a\|{\sim}a = {\sim}a$, then for all $b, c \in A$ we have $b\|c = c$.*

Proof. We only prove the first part, the second one is dual.

Assume $a\|{\sim}a = a$, and assume $b\|c = c$. By the previous lemma, we know that $0\|1 = 0$. We then have $b\|1 = (0\|1) \vee b = 0 \vee b = b$. Now assume $c \not\leq b$. Then $b\|c \not\leq b\|1$ – contradiction to monotonicity (Lemma 4). Hence $c \leq b$.

By the previous lemma, also $1\|0 = 1$. So we have $1\|c = (1\|0) \vee c = 1$. As $c \leq b$, $c \wedge b = c$, so we have

$$(1\|c) \wedge b = (1 \wedge b)\|(b \wedge c) = b\|c \tag{*}$$

Conversely,

$$(1\|c) \wedge b = 1 \wedge b = b \tag{+}$$

Hence by (*), (+) $b\|c = b$, so by assumption $b = c$ and the claim follows. ⊣

Now we can prove the strongest result on $\|$, the uniformity lemma.

Lemma 7 *(Uniformity lemma). Assume we have an ambiguous algebra* **A** *$a, b \in A$ such that $a \neq b$.*

1. If $a\|b = a$, then for all $c, c' \in A$, we have $c\|c' = c$;

2. if $a\|b = b$, then for all $c, c' \in A$, we have $c\|c' = c'$.

Proof. We only prove 1., as 2. is completely parallel. Assume there are $a, b \in A$, $a \neq b$ and $a\|b = a$. Assume furthermore there are $c, c' \in A$ such that $c\|c' \neq c$. There are two cases:

(i) there is such a pair c, c' such that $c' = {\sim}c$. Then the dual result of the previous lemma leads us to a contradiction, because we then have $c\|{\sim}c = {\sim}c$, and consequently $a\|b = b$, which is wrong by assumption – contradiction.

(ii) there are no such pair c, c'. Then however we necessarily have (among other) $a\|{\sim}a = a$, and by the previous lemma, this entails $c\|c' = c$ – contradiction. ⊣

Put differently:

Corollary 8. *If* $\mathbf{A}$ *is an ambiguous algebra,* $a, b \in \mathbf{A}$*, then either for all* $a, b \in A$*, we have* $a\|b = a$*, or for all* $a, b \in A$*, we have* $a\|b = b$*.*

We therefore can say an ambiguous algebra $\mathbf{A}$ is LEFT-SIDED, if for all $a, b \in A$, $a\|b = a$; it is RIGHT-SIDED if for all $a, b \in A$, $a\|b = b$. The uniformity lemma says that every ambiguous algebra is either right-sided or left-sided, and only the trivial one-element algebra is both. Needless to say, it is hard to intuitively make sense of this result, as it is much stronger property than any of our intuitions on the meaning of ambiguity would suggest. Still it is very important as a *negative* result for the semantic treatment of ambiguity in the context of Boolean algebras.

One might consider this a triviality result, and in some sense it is. However, it does not imply triviality for the equations which hold in *all* ambiguous algebras, which is the really crucial notion for inference from ambiguous meanings. Let us consider the following examples:

$$(a \to a\|b) \vee (b \to a\|b) = 1 \tag{15}$$

holds in *all* ambiguous algebras. This can be shown as follows: we have

$$(a \to a\|b) \vee (b \to a\|b) = ((a \to a)\|(a \to b)) \vee ((b \to a)\|(b \to b)) \tag{16}$$

Now in every left-sided ambiguous algebra, we have $((a \to a)\|(a \to b)) = 1$; in every right-sided ambiguous algebra we have $((b \to a)\|(b \to b)) = 1$, and as every ambiguous algebra is either right-sided or left-sided, the claim follows. Conversely,

$$(a \to a\|b) \vee (b \to b\|a) = 1 \tag{17}$$

holds in some, but not all ambiguous algebras, as can be easily seen in case where $a\|b = b, b\|a = a$. Another equation which holds in all ambiguous algebras is the following:

$$a\|(b\|c) = (a\|b)\|(a\|c) \tag{18}$$

This is to say $\|$ distributes "over itself". The next is also easy to show:

Lemma 9. *In all ambiguous algebras, we have* $(a\|b)\|c = a\|c = a\|(b\|c)$.

Proof. Every algebra is either left sided, and then $(a\|b)\|c = a = a\|c$, or right-sided, then $(a\|b)\|c = c = a\|c$. ⊣

So the associativity of $\|$ follows from ($\|1$)–($\|3$). Moreover, every multiple ambiguity can be reduced to an ambiguity of only two terms. This is a very interesting result for decidability,[9] though it runs counter to our intuitions on ambiguity: linguistically it would mean that any ambiguity between an arbitrary number of meanings is equivalent to an ambiguity between two meanings.

6 The Meaning of Uniformity

The uniformity lemma is obviously a very strong result. Whereas its algebraic meaning should be clear, it is not so clear how it relates to our intuitions on ambiguity, from which we have started after all. To clarify this, it is preferable to first consider the monotonicity Lemma 4, which is a weaker result. This lemma states that if $a \leq a'$, $b \leq b'$, then $a\|b \leq a'\|b'$. This roughly means: ambiguity respects logical entailment. Let us illustrate this with an example; take the lexically ambiguous word `bank`; and assume that it has the meaning $a\|b$, a being 'strip of land along a river', b being 'financial institute'. Next consider the (complex) expression `bank or restaurant`. Say `restaurant` has meaning c, hence the expression has the meaning $(a\|b) \vee c = (a \vee c)\|(b \vee c) = a'\|b'$. Now as $\leq$ corresponds to entailment, this means that if monotonicity were wrong, then `bank` would *not* (generally) entail `bank or restaurant`, simply because in the one expression it could mean one thing, in the other expression the other. This however obviously contradicts the hypothesis of uniform usage.[10] Now monotonicity can be read as a strengthening of this hypothesis: even if there was a single word `kank` with the same meaning as `bank or restaurant`, using `bank` in one sense would constrain us to using `kank` in the related sense. Hence monotonicity is like uniform usage for expressions which are connected by the relation of entailment. In this sense we say that ambiguity *respects* entailment.

Now the underlying reason for uniformity is that in ambiguous algebras, all elements are strongly connected, in particular because there are elements such as $0\|1$, $1\|0$ (which surely do not have any linguistic counterpart). Thereby, we can establish that if we use one ambiguous expression in the left/right sense (whatever that means), we have to use all expressions in the same sense. This is surely completely unintuitive, and a consequence of two things:

1. the strong axioms of Boolean algebras, in particular the equality $\sim\sim a = a$ we repeatedly use, and
2. the fact that $\|$ is a total operator, that is, for all a, b, we have an object $a\|b$.

[9] From the results in this paper, decidability results easily follow, but for reasons of space we do not include them here.

[10] However, one can say such a thing as: `I do not need such a bank, I need the other bank!` For us, this would count as a disambiguation, hence strictly speaking it takes the ambiguity from the semantics.

So there are two ways to remedy the situation: 1. Consider algebras with weaker axioms than Boolean algebras. In particular, if we add the ambiguity operator and axioms ($\|1$)–($\|3$) to Heyting algebras (corresponding to intuitionistic logic), many of the results presented so far do no longer seem to hold. 2. to assume that $\|$ is a partial operator, or put differently, that a Boolean algebra only contains certain ambiguous elements. To us, both roads seem to be promising, in particular the one of PARTIALLY AMBIGUOUS ALGEBRAS. In these, we can investigate which ambiguous elements are independent and which not (in the sense that $\|$ has to be uniform for them). However, both topics deserve and need a treatment on their own, so for the rest of the paper, we rather complete the theory of ambiguous Boolean algebras by showing some results on their existence and construction, from which it is easy to derive results on decidability (however we do not present the latter for reasons of space).

7 Structure Theory II: Completions

We have given an example of one non-trivial ambiguous algebra. The previous section has provided us with restrictions on possible algebras. In this part, we will show that nonetheless every Boolean algebra can be completed to an ambiguous algebra. From the uniformity lemma, it easily follows that there are at most two such completions; we now prove there are exactly two. It also easily follows that every ambiguous algebra is the completion of a Boolean algebra.

Definition 10. *Let* $\mathbf{B} = (B, \wedge, \vee, \sim, 0, 1)$ *be a Boolean algebra. We define the* LEFT AMBIGUOUS COMPLETION *of* $\mathbf{B}$ *to be the ambiguous algebra* $C_l(\mathbf{B}) := (B, \wedge, \vee, \sim, \|, 0, 1)$*, where for all* $a, b \in B$*, we have* $a\|b = a$*; the* RIGHT AMBIGUOUS COMPLETION $C_r(\mathbf{B})$ *is defined in the parallel fashion, where* $a\|b = b$*.*

We mostly say only right/left completion, as there is no source of confusion.

Lemma 11. *If* $\mathbf{B}$ *is a Boolean algebra, then both* $C_l(\mathbf{B})$ *and* $C_r(\mathbf{B})$ *are ambiguous algebras.*

Proof. We simply check whether it satisfies the axioms. ($\|3$) is clear by definition. We check the distributivity axioms (for simplicity, we only consider left-completions; the right case is completely parallel):

$\sim$-distributivity: we have $a\|b = a$, so $\sim(a\|b) = \sim a = \sim a\|\sim b$.

$\wedge$-distributivity: we have $(a\|b) \wedge c = a \wedge c = (a \wedge c)\|(b \wedge c)$. ⊣

The following is also quite simple:

Lemma 12. *Every ambiguous algebra* $\mathbf{A}$ *is isomorphic either to* $C_l(\mathbf{B})$ *or* $C_r(\mathbf{B})$ *for some Boolean algebra* $\mathbf{B}$*.*

Proof. Straightforward consequence of the uniformity lemma. ⊣

We now formally prove the existence of non-trivial algebras, more concretely: for every Boolean algebra $\mathbf{B}$, the Boolean algebra reduct of both $C_l(\mathbf{B})$ and $C_r(\mathbf{B})$ is identical to $\mathbf{B}$; this means that the ambiguous algebra axioms do not collapse any two elements. This has a number of consequences, among other for decidability.

Let t be a term of an ambiguous algebra. We define the map π_l as follows:

1. $\pi_l(x) = x$, for x atomic
2. $\pi_l(\sim t) = \sim \pi_l(t)$
3. $\pi_l(t \wedge t') = \pi_l(t) \wedge \pi_l(t')$
4. $\pi_l(t \vee t') = \pi_l(t) \vee \pi_l(t')$
5. $\pi_l(t \| t') = \pi_l(t)$

The right projection π_r is defined in the same way, with 5. changed to $\pi_r(t \| t') = \pi_r(t')$. An easy induction yields the following:

Lemma 13. *Let* $\mathbf{A}$ *be a left- (right-)sided ambiguous algebra,* s, t *terms over* $\mathbf{A}$*. Then* $s = t$ *holds in* $\mathbf{A}$ *iff* $\pi_l(s) = \pi_l(t)$ *(*$\pi_r(s) = \pi_r(t)$*) holds in* $\mathbf{A}$*.*

Proof. Easy induction over complexity of s, t. ⊣

In order to prove the crucial result of this section, we need to introduce a class of structures which we call AA'-algebras. These have the same signature as ambiguous algebras, but a different set of axioms:

Definition 14. *A structure* $\mathbf{A} = (A, \wedge, \vee, \sim, \|, 0, 1)$ *is a* LEFT AA'-ALGEBRA*, if* $(A, \wedge, \vee, \sim, 0, 1)$ *is a Boolean algebra, and for all* $a, b \in A$*, we have* $a \| b = a$*.* $\mathbf{A}$ *is a* RIGHT AA'-ALGEBRA*, if* $(A, \wedge, \vee, \sim, 0, 1)$ *is a Boolean algebra, and for all* $a, b \in A$*, we have* $a \| b = b$*.*

So AA'-algebras are less restrictive in the sense that they do not satisfy the additional axioms ($\|1$)–($\|3$). For us, they are useful because they allow to establish the following lemma. By id_M, we generally denote functions which compute the identity on some domain M.

Lemma 15. *Every Boolean algebra* $\mathbf{B} = (B, \wedge, \vee, \sim, 0, 1)$ *can be completed to a left/right* AA'*-algebra* $\boldsymbol{A}\boldsymbol{'} = (B, \wedge, \vee, \sim, \|, 0, 1)$ *such that the map* $\mathsf{id} : \boldsymbol{A}\boldsymbol{'} \to \mathbf{B}$ *is a Boolean algebra isomorphism.*

Proof. The completion is obvious, and a straightforward induction on its Boolean terms shows that $\mathsf{id} : \mathbf{A'} \to \mathbf{B}$ is a Boolean algebra isomorphism. ⊣

The completions are unique, so we call this the left/right AA'-completion, and denote it by $C'_l(\mathbf{B})$, $C'_r(\mathbf{B})$. Let $\cong$ denote the relation of isomorphy, which says that there is a bijection between two structures which preserves the results of all operations.

Lemma 16. *For all Boolean algebras* $\mathbf{B}$*, we have* $C'_l(\mathbf{B}) \cong C_l(\mathbf{B})$ *and* $C'_r(\mathbf{B}) \cong C_r(\mathbf{B})$*.*

Proof. We only consider the left case. $C_l(\mathbf{B})$ is the algebra which satisfies (1) all equations of $\mathbf{B}$, (2) $a\|b = a$, and (3) all of ($\|1$)–($\|3$). To prove the lemma, we only need to show that the same holds for $C'_l(\mathbf{B})$. For (1) and (2), this is straightforward. So we only prove that $C'_l(\mathbf{B})$ satisfies ($\|1$)–($\|3$).

($\|1$) $\sim(a\|b) = \sim a = \sim a\|\sim b$

($\|2$) $a \wedge (b\|c) = a \wedge b = (a \wedge b)\|(a \wedge c)$.

($\|3$) By definition, $a \leq a\|b$. ⊣

The next result is really a central lemma, in particular our subsequent decidability results rely on it. In particular it shows the existence of infinitely many non-isomorphic ambiguous algebras.

Lemma 17 *(Completion lemma). Every Boolean algebra* $\mathbf{B} = (B, \wedge, \vee, \sim, 0, 1)$ *can be completed to an ambiguous algebra* $\mathbf{A} = (B, \wedge, \vee, \sim, \|, 0, 1)$ *such that the map* $\mathsf{id} : \mathbf{A} \to \mathbf{B}$ *is a Boolean algebra isomorphism.*

Proof. By Lemma 15, we know the claim holds for $C'_l(\mathbf{B})$, $C'_r(\mathbf{B})$. By Lemma 16, $C'_l(\mathbf{B}) \cong C_l(\mathbf{B})$ etc., and as the composition of two isomorphisms is still an isomorphism, the claim follows. ⊣

Corollary 18. *Let* $\mathbf{B} = (B, \wedge, \vee, \sim, 0, 1)$ *be a Boolean algebra, and assume* $a, b \in B$*. Then* $a =_{\mathbf{B}} b$ *iff* $a =_{C_l(\mathbf{B})} b$ *iff* $a =_{C_r(\mathbf{B})} b$*.*

This states that we cannot derive new equalities between objects which are distinct in $\mathbf{B}$, which follows from Lemma 17 (because otherwise id would not be a bijection). We can formulate this result in another, more general fashion:

Corollary 19. *For every Boolean algebra* $\mathbf{B}$ *there are (up to isomorphism) exactly two ambiguous algebras* $\mathbf{A}$ *such that there is a bijection* $i : \mathbf{B} \to \mathbf{A}$ *which is a Boolean algebra isomorphism.*

The existence of two algebras is witnessed by $C_l(\mathbf{B})$, $C_r(\mathbf{B})$; the fact that up to isomorphism there are at most two such algebras follows from the uniformity lemma.

8 Conclusion and Further Work

Our treatment of ambiguous Boolean algebras seems to be complete in the sense that every interesting result on them seems to be either stated explicitly or easily derivable from the results stated here. This is not as much due to excessive treatment as to the fact that ambiguous Boolean algebras are very similar to Boolean algebras (which are excessively treated in many places), maybe too similar to be really interesting. In particular the uniformity lemma and the completion lemma show that there is little of interest to say about ambiguous algebras which does not already hold for Boolean algebras. Still we consider it important to have established these results, which are really obvious or trivial.

Furthermore, our algebraic results are surely not fully compatible with linguistic intuition. The uniformity lemma suggests that all ambiguous terms are

either right- or left-ambiguous. Even if there is some flexibility in choosing an algebra depending on the context, this would have to be fixed in advance and ironically contravenes any possibility to establish an order of operands in terms of literal and idiomatic meaning. The weaker monotonicity result, by contrast, which states that words in entailment relations are used consistently meaning-wise, seems much more intuitive and preferable. Nevertheless, the presented work is a first necessary approximation relying on an obvious choice, namely Boolean algebras, corresponding to classical propositional logic. Note that our negative results also have a positive side: it entails that for establishing a semantic notion of ambiguity, we either need to reject the Boolean axioms, or the idea that ambiguity is possible between arbitrary meanings. So if we start to take semantic ambiguity seriously, it can help us gain genuine insights into semantic structure.

The further investigation of the algebraic treatment of ambiguity seems to be very interesting and promising. As we have already mentioned, there are two main directions to pursue: One approach would be the following: instead considering $\|$ and ($\|1$)–($\|3$) in the context of Boolean algebras, we can consider it in other, more general classes of algebras which are important for reasoning purposes, such as (distributive, modular) lattices, residuated lattices, Heyting algebras and many more. The proof of the uniformity lemma (and many other results) rely on the law of excluded middle ($a \vee \sim a = 1$), which does not hold in all these algebras, so enriching them with an operator $\|$ and axioms ($\|1$)–($\|3$) might result in a much richer structure theory. What can still be derived in more general cases is the monotonicity for $\|$, which is not nearly as strong as uniformity. The second approach would be to investigate Boolean algebras in which $\|$ is partial, that is, there are some ambiguous elements, but ambiguity is not a total operator for arbitrary objects. We plan to further pursue both lines in further research.

References

1. Asher, N., Denis, P.: Lexical ambiguity as type disjunction. In: Bouillon, P., Kanzaki, K. (eds.) Proceedings of the International Workshop on Generative Approaches to the Lexicon (GL2005), Genève, Switzerland, pp. 10–17 (2005)
2. Cooper, R., Crouch, R., van Eijck, J., Fox, C., van Genabith, J., Jaspars, J., Kamp, H., Milward, D., Pinkal, M., Poesio, M., Pulman, S. (eds.) Using the framework. The FraCaS Consortium, Technical report, FraCaS deliverable D-16 (1996)
3. Ernst, T.: Grist for the linguistic mill: idioms and 'extra' adjectives. J. Linguist. Res. **1**, 51–68 (1981)
4. Husserl, E.: V. logische Untersuchung: Über intentionale Erlebnisse und ihre "Inhalte". No. 290 in Philosophische Bibliothek, Meiner, Hamburg (1975)
5. Kracht, M.: Mathematics of Language. Mouton de Gruyter, Berlin (2003)
6. Maddux, R.: Relation Algebras. Elsevier, Amsterdam (2006)
7. Peterson, R.R., Burgess, C.: Syntactic and semantic processing during idiom comprehension: neurolinguistic and psycholinguistic dissociations. In: Cacciari, C., Tabossi, P. (eds.) Idioms: Processing, Structure, and Interpretation, pp. 201–225. Lawrence Erlbaum, Hillsdale (1993)

8. Pinkal, M.: Logic and Lexicon: The Semantics of the Indefinite. Kluwer, Dordrecht (1995)
9. Pinkal, M.: On semantic underspecification. In: Bunt, H., Muskens, R. (eds.) Computing Meaning, vol. 1, pp. 33–55. Springer, Dordrecht (1999)
10. Poesio, M.: Semantic ambiguity and perceived ambiguity. In: van Deemter, K., Peters, S. (eds.) Semantic Ambiguity and Underspecification, pp. 159–201. CSLI Publications, Stanford (1994)
11. Stallard, D.: The logical analysis of lexical ambiguity. In: Proceedings of the 25th Annual Meeting on Association for Computational Linguistics (ACL 1987), pp. 179–185 (1987)
12. Wittenberg, E., Jackendoff, R.S., Kuperberg, G., Paczynski, M., Snedeker, J., Wiese, H.: The processing and representation of light verb constructions. In: Bachrach, A., Roy, I., Stockall, L. (eds.) Structuring the Argument. John Benjamins, Amsterdam (2014)
13. Wittgenstein, L.: Philosophische Untersuchungen. No. 1372 in Bibliothek Suhrkamp, Suhrkamp, Frankfurt am Main, 1. edn. (2010)

Erratum to: The Proper Treatment of Linguistic Ambiguity in Ordinary Algebra

Christian Wurm and Timm Lichte

Erratum to:
Chapter "The Proper Treatment of Linguistic Ambiguity in Ordinary Algebra" in: A. Foret et al. (Eds.):
***Formal Grammar, LNCS 9804*,**
https://doi.org/10.1007/978-3-662-53042-9_18

This is a correction note to the paper starting on p. 306. There is an error in the proof of Lemma 1, which states that the axiom

($\|3$) At least one of $a \leq a\|b$ or $b \leq a\|b$ holds

together with the other axioms entails the stronger statement that either $a = a\|b$ or $b = a\|b$. The proof of this lemma is incorrect, and the claim is wrong: we can construct an algebra with 4 elements $\{0, 1, 0\|1, 1\|0\}$ with the obvious Boolean algebra order, and $\|$ defined by the *margin property*: $a\|b\|c = a\|c$, with $a, b \in \{0, 1\}$, c an arbitrary term. It is not difficult to check that this is an ambiguous algebra in the sense of Section 4 of the paper, yet $0 \neq 0\|1 \neq 1$. As almost all later results are based upon Lemma 1, they are technically unproved. However, all problems can be remedied very easily by changing ($\|3$) from the paper to:

($\|3'$) At least one of $a = a\|b$ or $b = a\|b$ holds

Hence to make the paper correct, all we need is a slightly different axiom ($\|3'$), and Lemma 1 becomes basically part of the definition, so all problems are solved. So far our correction; there are two notes which might be interesting to the reader:

The original online version of this chapter can be found at https://doi.org/10.1007/978-3-662-53042-9_18

A. Foret et al. (Eds.): FG 2015/2016, LNCS 9804, pp. E1–E2, 2016.
https://doi.org/10.1007/978-3-662-53042-9_19

Note 1 From the point of view of the linguistic motivation of the axioms, $(\|3')$ is actually more natural than the original $(\|3)$, because it basically states that an ambiguous meaning is supposed to *intend* one of the meanings between which it is ambiguous. The weaker $(\|3)$ just states that it is supposed to *entail* one of these meanings, which is not what we would intuitively think. Actually, the authors of the paper preferred $(\|3)$ over $(\|3')$ not on a conceptual base, but rather because it is simply weaker and they believed the two to be equivalent anyway (this is what Lemma 1 of the mentioned paper states).

Note 2 In (Wurm, 2017), the authors have introduced the class of **universal distribution algebras (UDA)**, which is still weaker. If we take the class of ambiguous algebras as introduced in the 2016 paper and add an axiom for $\|$-associativity $(a\|(b\|c) = (a\|b)\|c$, which does not seem derivable so far), then it is not difficult to show that **UDA** subsumes this class. What is interesting is that **UDA** seems to have the same equational theory as ambiguous algebras in the strong sense (with $(\|3')$). Now since the class as defined in 2016, with associativity added, lies in between the two, it is neatly characterized by this (yet unpublished) result.

Reference

Wurm, C.: The logic of ambiguity: the propositional case. In: Foret, A., Muskens, R., Pogodalla, S. (eds.) Formal Grammar. 22th Conference, FG 2017, Toulouse, France, July 2017, Proceedings. Lecture Notes in Computer Science, vol. 10686. Springer (2017)

Author Index