Sylvain Pogodalla
Jean-Philippe Prost (Eds.)

Logical Aspects of Computational Linguistics

LNAI 6736

6th International Conference, LACL 2011
Montpellier, France, June/July 2011
Proceedings

Springer

Lecture Notes in Artificial Intelligence 6736

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

FoLLI Publications on Logic, Language and Information

Sylvain Pogodalla Jean-Philippe Prost (Eds.)

Logical Aspects of Computational Linguistics

6th International Conference, LACL 2011
Montpellier, France, June 29 – July 1, 2011
Proceedings

Springer

Volume Editors

Sylvain Pogodalla
INRIA Nancy – Grand Est
615, rue du Jardin Botanique
54602 Villers-lès-Nancy Cedex, France
E-mail: sylvain.pogodalla@inria.fr

Jean-Philippe Prost
Université Montpellier 2, LIRMM
UMR 5506 - CC 477
161, rue Ada, 34095 Montpellier Cedex 5, France
E-mail: jean-philippe.prost@lirmm.fr

ISSN 0302-9743 e-ISSN 1611-3349

ISBN 978-3-642-22220-7 e-ISBN 978-3-642-22221-4

DOI 10.1007/978-3-642-22221-4
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: Applied for

CR Subject Classification (1998): I.2, F.4.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the proceedings of the 6th International Conference on Logical Aspects of Computational Linguistics 2011 (LACL 2011), held June 29-July 1, 2011 in Montpellier, France. The LACL conferences aim at providing a forum for the presentation and discussion of current research in all the formal and logical aspects of computational linguistics.

The Program Committee selected 18 papers out of 31 submissions for presentation at LACL 2011. The topics they cover include type-theoretical grammars, dependency grammars, formal language theory, grammatical inference, minimalism, generation, lexical and formal semantics, by authors from Argentina, France, Germany, Japan, The Netherlands, Spain, Sweden, UK, and USA.

In addition to the contributed talks, two invited talks were delivered by T. Fernando (Trinity College) and C. Gardent (LORIA/CNRS). The latter was a joint event with Traitement Automatique des Langues Naturelles 2011 (TALN 2011), the international French-speaking conference on natural language processing.

We are grateful to all the people who made this meeting possible and are responsible for its success: the members of the Program Committee and the external reviewers, the invited speakers, the contributors, and the people who were involved in organizing the conference.

We would also like to express our gratitude to the following organizations for supporting LACL 2011: the Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), the Université de Montpellier 2 (UM2), the Centre National de la Recherche Scientifique (CNRS), the Association pour le Traitement Automatique des Langues (ATALA), the Institut National de Recherche en Informatique et en Automatique (INRIA), the Université de Provence, Orange, Syllabs, and the Laboratoire Parole et Langage (LPL).

The reviewing for the conference and the preparation of the proceedings were greatly aided by the free EasyChair conference management system, and we wish to thank its developers.

April 2011

Sylvain Pogodalla
Jean-Philippe Prost

Organization

Program Committee

Pascal Amsili	Université Paris Diderot (Paris 7)/LLF, UMR CNRS 7110, France
Nicholas Asher	CNRS Laboratoire IRIT, France
Raffaella Bernardi	University of Trento, Italy
Philippe Blache	CNRS LPL Aix-en-Provence, France
Johan Bos	University of Groningen, The Netherlands
Joan Busquets	Université Bordeaux 3, ERSS-CNRS and INRIA-Bordeaux, France
Benoît Crabbé	Paris 7 and INRIA, France
Philippe de Groote	LORIA/INRIA Nancy, France
Denys Duchier	Université d'Orléans, France
Markus Egg	Humboldt-Universität Berlin, Germany
Nissim Francez	Technion - IIT, Israel
Makoto Kanazawa	National Institute of Informatics, Japan
Greg Kobele	University of Chicago, USA
Marcus Kracht	Universität Bielefeld, Germany
Alain Lecomte	Université Paris 8 - CNRS, France
Michael Moortgat	Utrecht Institute of Linguistics - OTS, The Netherlands
Richard Moot	LaBRI(CNRS), INRIA Bordeaux SW and Bordeaux University, France
Glyn Morrill	Universitat Politècnica de Catalunya, Spain
Reinhard Muskens	Tilburg Center for Logic and Philosophy of Science, The Netherlands
Uwe Mönnich	University of Tübingen, Germany
Gerald Penn	University of Toronto, Canada
Sylvain Pogodalla	LORIA/INRIA Lorraine, France
Carl Pollard	Ohio State University, USA
Anne Preller	LIRMM/CNRS, France
Jean-Philippe Prost	LIRMM/Université de Montpellier 2, France
Laurent Prévot	LPL, Université de Provence, France
Aarne Ranta	Chalmers University of Technology and Göteborg University, Sweedan
Christian Retoré	LaBRI, INRIA and Université de Bordeaux 1, France
James Rogers	Earlham College, USA
Chung-chieh Shan	Rutgers University, USA
Mark Steedman	University of Edinburgh, UK
Isabelle Tellier	Lifo, Université d'Orléans, France

Additional Reviewers

Timothy Fowler
Michael Kaminski

Organizing Committee

Philippe Blache	CNRS LPL Aix-en-Provence, France
Alexandre Labadié	LIRMM Université de Montpellier 2, France
Mathieu Lafourcade	LIRMM Université de Montpellier 2, France
Cédric Lopez	LIRMM Université de Montpellier 2, France
Anne Preller	LIRMM/CNRS, France
Violaine Prince	LIRMM Université de Montpellier 2, France
Sylvain Pogodalla	LORIA/INRIA Lorraine, France
Jean-Philippe Prost	LIRMM Université de Montpellier 2, France
Christian Retoré	LaBRI, INRIA and Université de Bordeaux 1, France
Johan Segura	LIRMM Université de Montpellier 2, France

Table of Contents

Encoding Phases Using Commutativity and Non-commutativity in a Logical Framework

Maxime Amblard

LORIA - INRIA Nancy Grand Est - BP 239 - 54506 Vandoeuvre-lès-Nancy Cedex
Université Nancy 2,13 rue Maréchal Ney - 54037 Nancy cedex
INPL, 2 av. de la Forêt de Haye - BP 3 - F-54501 Vandoeuvre
amblard@loria.fr

Abstract. This article presents an extension of Minimalist Categorial Grammars (MCG) to encode Chomsky's *phases*. These grammars are based on Partially Commutative Logic (PCL) and encode properties of Minimalist Grammars (MG) of Stabler [22]. The first implementation of MCG were using both non-commutative properties (to respect the linear word order in an utterance) and commutative ones (to model features of different constituents). Here, we propose to augment Chomsky's *phases* with the non-commutative tensor product of the logic. Then we can give account of the PIC [7] just with logical properties of the framework instead of defining a specific rule.

Keywords: Type theory, syntax, linguistic modeling, generative theory, phase, Partially Commutative Logic.

Generative theory has undergone many changes since Chomsky's Syntactic Structures [5] leading up to what is the Minimalist Program (MP) [6]. The most frequent criticism made is certainly the non-computational and non-formal nature of such an approach. It is nevertheless rich of a vast literature for the linguistic approach. Fundamentals to the MP are the description of a main calculus which takes into account the syntax, and the production of two forms: one supposed to reflect the sequence of words, and another one for the semantic structure of the utterance. Following the MP, Chomsky claims the identification of *phases* in the syntactic derivation [7]. The verb, the main driving force of the analysis, is being transformed, opening the possibility of specific modifications, especially the definition of the Phase Impenetrability Condition (PIC).

The first proposal of formalization for MP was made by Stabler in [22]. However, this formulation is far from the usual Montagovian approach of semantics. Therefore, translations of this formulation into logic systems have been proposed, in particular [15], [13]. Much has been done, exploiting Curry's distinction between the tectogrammatical and the phenogrammatical levels, and this has led to interesting proposals [17], [9], [18], [19]. The latest proposals of extension defined the MCG based on a fragment of Partially Commutative Logic (PCL) [1], [2]. The authors highlighted the simultaneous need of commutative and non-commutative properties to produce a useful framework .

In this paper, we propose a reconsideration of the properties of commutativity and non-commutativity in MCG to account the concept of *phase* introduced by Chomsky.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 1–16, 2011.

Due to space consideration, we will limit ourselves to the problems of parsing, leaving aside the semantic aspects. However, it should be noted that the syntax-semantics interface of MCG contains all the necessary material for its integration.

We first discuss the need for the two different relations in MCG. Then, we define Minimalist Categorial Grammars (MCG). Based on these definitions, the third section presents and encodes *phases* in MCG, and shows the implementation of the PIC using only logical properties.

1 Commutativity vs. Non-commutativity in Standard MCG and Phases

To link logic and Generative Theory, MCG's derivations are proofs of a restriction of Partially Commutative Logic (PCL), [20], seen as syntactic representations in generative theory. This logic is an extension of Lambek calculus containing simultaneously commutative and non-commutative connectives (*ie* introduction and elimination of implication and tensor). To handle the different relations between hypotheses in the same framework, an entropy rule (restriction of order) is added. Moreover,[3] shows a weak normalization of this calculus, to produce regular analysis in MCG.

All definitions of these grammars are given in [1],[1], with a composition of rules (note that according to these definitions, normalisation is strong). Moreover, this restriction does not use introduction of hypothesis. They appear in the derivation only from specific lexical entries: in the hypotheses of a given category and that category as a formula. The lexicon will contain the entry $\phi \vdash \phi$ with ϕ as a given category.

The concrete part of the proof is achieved by a *merge* whose heart is the elimination of $/$ or $\backslash$ (rules that are found in different versions from categorial grammars, [12], [24] [16]) plus the entropy rule. This is one point where commutativity and non commutativity play a crucial role. In particular, for the word order, it is clear that the relation is non-commutativite: being on the right or the left of a given word could not just be the same. Non-commutativity is also needed here, because of the second rule in MCG.

The hypotheses are seen as special position markers in the sequence of hypotheses of the proof. They are considered as resources of prominent features related to a phrase. They are unloaded by using the *move* operation of the generative theory. A direct implication is that the sequence of hypotheses in the proof contains exactly the sequence of available resources for further derivation. In this case, this sequence is a collection of resources and could not be a strict list. Unless a canonical order on the sequence of applied rules is presupposed, the only way to express this is with non-commutativity. The *Merge* rule must contain a release of the order. Definitions of basic rules of MCG involve commutativity and non-commutativity. We will show how we could use these properties in another perspective to encode a linguistic concept, more precisely by noting that the changing category of the verb controls the process flow.

Chomsky's theory introduces the notion of *phase*, which corresponds to the evolution of the verb. Thus, we assume that the lexical item associated with the verb carries only

[1] These definitions contain also a syntax / semantic interface. However, we leave this part out of this article due to space.

part of its achievement in the sentence. This is an important use in the syntax-semantic interface for reporting UTAH[2]. The needed hypothesis to introduce a DP is provided by verb lexical items. And this is the correspondence of verb's resources with the DP category which allows the DP to be in the proof.

In the *phase*'s definition, Chomsky assumes that some instance of *move* (unification of features in our formalism) must be completed before the end of the *phase*. Once it is reached, the specific resources of the process are no longer accessible to the rest of the analysis. It defines an *island* in the analysis, called Phase Impenetrability Condition (PIC). Translating this definition into our formalism implies that a *phase* (or its representation) block access to part of the sequence of hypotheses of the proof. We assume that the interpretation in MCG is a non-commutative point in the sequence of hypotheses.

The direct implication is that the analysis of a sentence simultaneously uses relations to link noun and verb phrases, and non-commutative relations to construct the analysis (changing the category of the verb). The use of non-commutativity involves a strict order to control the verb's role in the analysis. Formal properties used by items of different categories are disjointed and it allows to fully exploit their relations.

2 Minimalist Categorial Grammars

Minimalist Categorial grammars are based on a fragment of PCL to encode the MP of Chomsky. Then it uses an abstract calculus to produce both a string (sequence of words) and a semantic representation (formula). Word order and semantics are synchronized over the main calculus which uses a structured lexicon and specific rules. First, we briefly present rules of PCL, then we introduce labels which encode word order, then we define lexical items and finally introduce rules of MCG.

2.1 Partially Commutative Logic (PCL)

The logic introduced in [8] and extended in [21] is a superimposition of the Lambek calculus (Intuitionistic Non-Commutative Multiplicative Linear Logic) and Intuitionistic Commutative Multiplicative Linear Logic. Connectives are:

- the Lambek calculus connectives: $\odot$, $\backslash$ and $/$ (non-commutatives)
- commutative multiplicative linear connectives: $\otimes$ and $\multimap$ (commutatives)

The use of these connectives is presented in figure 1.

In this logic, commutative and non-commutative relations could be used simultaneously. Then contexts are partially ordered multisets of formulae and in order to relax this order, we need an *entropy* rule noted $\sqsubset$ which is defined as the replacement of ; by ,. The restriction of elimination rules and the entropy rule is called Minimalist Logic (the logic used to define MCG). In the following, we note F the set of categories (F stands for features).

[2] Uniform Theta Assignment Hypothesis or assignment of thematic roles.

$$\frac{\Gamma \vdash A \quad \Delta \vdash A\backslash C}{< \Gamma; \Delta > \vdash C}\,[\backslash_e] \qquad \frac{\Delta \vdash A/C \quad \Gamma \vdash A}{< \Delta; \Gamma > \vdash C}\,[/_e] \qquad \frac{\Gamma \vdash A \quad \Delta \vdash A \multimap C}{(\Gamma, \Delta) \vdash C}\,[\multimap_e]$$

$$\frac{< A; \Gamma > \vdash C}{\Gamma \vdash A\backslash C}\,[\backslash_i] \qquad \frac{< \Gamma; A > \vdash C}{\Gamma \vdash C/A}\,[/_i] \qquad \frac{(A, \Gamma) \vdash C}{\Gamma \vdash A \multimap C}\,[\multimap_i]$$

$$\frac{\Delta \vdash A \odot B \quad \Gamma, < A; B >, \Gamma' \vdash C}{\Gamma, \Delta, \Gamma' \vdash C}\,[\odot_e] \qquad \frac{\Delta \vdash A \otimes B \quad \Gamma, (A, B), \Gamma' \vdash C}{\Gamma, \Delta, \Gamma' \vdash C}\,[\otimes_e]$$

$$\frac{\Delta \vdash A \quad \Gamma \vdash B}{< \Delta; \Gamma > \vdash A \odot B}\,[\odot_i] \qquad \frac{\Delta \vdash A \quad \Gamma \vdash B}{(\Delta, \Gamma) \vdash A \otimes B}\,[\otimes_i]$$

$$\frac{}{A \vdash A}\,[axiom] \qquad \frac{\Gamma \vdash C}{\Gamma' \vdash C}\,[\text{entropy — whenever } \Gamma' \sqsubset \Gamma]$$

Fig. 1. Partially Commutative Logic rules

2.2 Labels Encoding Word Order

Derivations of MCG are labelled proofs of the PCL. Before defining labelling, we define labels and operations on them. To do this, we use the set of phonological form Ph and a set V of variables such that: $Ph \cap V = \emptyset$. We note T the union of Ph and V. We define the set Σ, called *labels set* as the set of triplets of elements of T^*. Every position in a triplet has a linguistic interpretation: they correspond to specifier/head/complement relations of minimalist trees. A label r will be considered as $r = (r_s, r_h, r_c)$.

We introduce variables in the string triplets and a substitution operation. They are used to modify a position inside a triplet by a specific material. Intuitively, this is the counterpart in the phonological calculus of the product elimination.

A *substitution* is a partial function from V to T^*. For σ a substitution, s a string of T^* and r a label, we note respectively $s.\sigma$ and $r.\sigma$ the string and the label obtained by the simultaneous substitution in s and r of the variables with the values associated by σ (variables for which σ is not defined remain the same).

If the domain of definition of a substitution σ is finite and equal to $x_1, \ldots, x_n$ and $\sigma(x_i) = t_i$, then σ is denoted by $[t_1/x_1, \ldots, t_n/x_n]$. Moreover, for a sequence s and a label r, $s.\sigma$ and $r.\sigma$ are respectively denoted $s[t_1/x_1, \ldots, t_n/x_n]$ and $r[t_1/x_1, \ldots, t_n/x_n]$. Every injective substitution which takes values in V is called *renaming*. Two labels r_1 and r_2 (respectively two strings s_1 and s_2) are equal modulo a renaming of variables if there exists a renaming σ such that $r_1.\sigma = r_2$ (*resp.* $s_1.\sigma = s_2$).

Finally, we need another operation on string triplets which allows to combine them together: the string concatenation of T^* is noted $\bullet$. Let $Concat$ be the operation of concatenation on labels which concatenates the three components in the linear order: for $r \in \Sigma$, $Concat(r) = r_s \bullet r_h \bullet r_c$.

We then have defined a word order structure which encodes specifier/complement/head relations and two operations (substitution and concatenation). These two operations will be counterparts in the phonological calculus of *merge* and *move*.

Labelled Proofs. Before exhibiting the rules of MCG, we need to define the concept of labelling on a subset of rules of the *Minimalist Logic* ($\backslash_e$, $/_e$, $\otimes_e$ and $\sqsubset$).

For a given MCG G, let a *G-background* be $x : A$ with $x \in V$ and $A \in F$, or $\langle G_1; G_2 \rangle$ or else (G_1, G_2) with G_1 and G_2 some *G-backgrounds* which are defined on two disjoint sets of variables. G-backgrounds are series-parallel orders on subsets of $V \times F$. They are naturally extended to the entropy rule, noted $\sqsubset$. A *G-sequent* is a sequent of the form: $\Gamma \vdash_G (r_s, r_t, r_c) : B$ where Γ is a G-background, $B \in F$ and $(r_s, r_t, r_c) \in \Sigma$.

A *G-labelling* is a derivation of a G-sequent obtained with the following rules:

$$\frac{\langle s, A \rangle \in Lex}{\vdash_G (\epsilon, s, \epsilon) : A}\,[Lex]$$

$$\frac{x \in V}{x : A \vdash_G (\epsilon, x, \epsilon) : A}\,[axiom]$$

$$\frac{\Gamma \vdash_G r_1 : A \,/\, B \quad \Delta \vdash_G r_2 : B \quad Var(r_1) \cap Var(r_2) = \emptyset}{\langle \Gamma; \Delta \rangle \vdash_G (r_{1s}, r_{1t}, r_{1c} \bullet Concat(r_2)) : A}\,[/_e]$$

$$\frac{\Delta \vdash_G r_2 : B \quad \Gamma \vdash_G r_1 : B \setminus A \quad Var(r_1) \cap Var(r_2) = \emptyset}{\langle \Gamma; \Delta \rangle \vdash_G (Concat(r_2) \bullet r_{1s}, r_{1t}, r_{1c}) : A}\,[\backslash_e]$$

$$\frac{\Gamma \vdash_G r_1 : A \otimes B \quad \Delta[x : A, y : B] \vdash_G r_2 : C \quad Var(r_1) \cap Var(r_2) = \emptyset \quad A \in \mathrm{P}_2}{\Delta[\Gamma] \vdash_G r_2[Concat(r_1)/x, \epsilon/y] : C}\,[\otimes_e]$$

$$\frac{\Gamma \vdash_G r : A \quad \Gamma' \sqsubset \Gamma}{\Gamma' \vdash_G r : A}\,[\sqsubset]$$

Note that a G-labelling is a proof tree of the Minimalist Logic on which sequent hypotheses are decorated with variables and sequent conclusions are decorated with labels. Product elimination is applied with a substitution on labels and implication connectors with concatenation (a triplet is introduced in another one by concatenating its three components).

2.3 Lexicon

MCG encodes informations in the lexicon with types. They are defined over two sets, one of linguistic categories and the other of move features. Lexical items associate a label and a formula of PCL with respect to the following grammar:

$$\begin{array}{lll}
\mathrm{L} & ::= (\mathrm{B}) \,/\, \mathrm{P}_1 \mid \mathrm{C} \\
\mathrm{B} & ::= \mathrm{P}_1 \setminus (\mathrm{B}) \mid \mathrm{P}_2 \setminus (\mathrm{B}) \mid \mathrm{C} \mid \mathrm{D} \\
\mathrm{C} & ::= \mathrm{P}_2 \otimes (\mathrm{C}) \mid \mathrm{C}_1 \\
\mathrm{D} & ::= \mathrm{P}_2 \odot (\mathrm{D}) \mid \mathrm{C}_1 \\
\mathrm{C}_1 & ::= \mathrm{P}_1
\end{array}$$

where L is the starting non-terminal and P_1 and P_2 are atomic formulae belonging to set of features of the MCG (features which trigger *merge* or *move* rules).

Formulae of lexical items get started with a $/$ as the first connective, and continue with a sequence of $\backslash$. This corresponds to the sequence of selectors and licensors in MG lexical items. These are trigger rule features of MCGs. They give the concrete part of the derivation. A formula is ended with an atomic type, the category of the phrase, or a sequence of $\odot$ (which contains at least a specific type, which is also the main category).

For example, the following formula could be the one of an MCG entry: $(d \setminus h \setminus j \setminus k \setminus (a \otimes b \otimes c)) / m$, whereas this is not : $(d \setminus h \setminus j \setminus k \setminus (a \otimes b \otimes c)) / m / p$, because it has two $/$. These formulae have the following structure :

$$(c_m \setminus \ldots \setminus c_1 \setminus (b_1 \otimes \ldots \otimes b_n \otimes a))/d$$

with $a \in P_1$, $b_i \in P_2$, $c_j \in P$ and $d \in P_1$.

The morphism from MG lexicon to MCG ones is defined in [1].

2.4 Rules of MCG

In the same way as for MG, [22], rules of MCG are defined over two principles:

- combining two pieces of derivation: *merge*
- redefining internal relations in a derivation: *move*

As we have mentioned before, MCG is defined over a restriction of PCL: elimination of $/$ and $\backslash$, and $\otimes$. In the following, in order to distinguish relations in the sequence of hypotheses, a commutative relation will be marked with ',' and a non-one with ';'.

- *Merge* is the function which combines two pieces of proofs together and it needs an non-commutative relation to correctly encode relations among words. But in the same application, *merge* will also combine hypothesis of different proofs. And from a linguistic point of view, relations between these hypothesis should be commutative because there is no reason to block access to them. Then, *merge* combines an elimination of $\backslash$ or $/$ with the application of an entropy rule. For the same linguistic reasons as in MG, MCG use two different kinds of *merge* depending on the lexical/non-lexical status of the trigger.

  For the word order, *merge* is simply the concatenation of the string of one phrase in the label of the other one (depending of right/left relation):
  Lexical trigger:

$$\dfrac{\dfrac{\vdash (r_s, r_h, r_c) : A \,/\, B \qquad \Delta \vdash s : B}{\Delta \vdash (r_s, r_h, r_c \bullet Concat(s)) : A}\,[/_e]}{\Delta \vdash (r_s, r_h, r_c \bullet Concat(s)) : A}\,[entropy]$$

$$\Longrightarrow$$

$$\dfrac{\vdash (r_s, r_h, r_c) : A \,/\, B \quad \Delta \vdash s : B}{\Delta \vdash (r_s, r_h, r_c \bullet Concat(s)) : A}\,[mg]$$

A *merge* with a lexical item do not explicitly show the order between hypotheses. But here, the entropy rule is the replacement of a ; by a ,

Non-lexical trigger:

$$\dfrac{\dfrac{\dfrac{\Delta \vdash s : B \qquad \Gamma \vdash (r_s, r_h, r_c) : B \setminus A}{\Delta; \Gamma \vdash (Concat(s) \bullet r_s, r_h, r_c) : A}\,[\setminus_e]}{\Delta, \Gamma \vdash (Concat(s) \bullet r_s, r_h, r_c) : A}\,[entropy]}{\Longrightarrow \quad \dfrac{\Delta \vdash s : B \quad \Gamma \vdash (r_s, r_h, r_c) : B \setminus A}{\Delta, \Gamma \vdash (Concat(s) \bullet r_s, r_h, r_c) : A}\,[mg]}$$

– The encoding of *Move* in MCG is structurally different from the one in MG. Here, it assumes that a phrase is included in the proof if and only if all its hypotheses are. Then, we do not really reinterpret the local tree as in MG, but we directly produce the final derivation tree. In this way, a *move* is the discharge of hypotheses by a $\otimes$. For word order, the concatenation of the moved phrase is substituted in the position of the newest hypothesis:

$$\dfrac{\Gamma \vdash r_1 : A \otimes B \quad \Delta[u : A, v : B] \vdash r_2 : C}{\Delta[\Gamma] \vdash r_2[Concat(r_1)/u, \epsilon/v] : C}\,[mv]$$

Finally, to give account of [23], the framework is enriched with rules which modifies the position of the string in the label. There are two kinds of rules:

– *head movement* where the head of the merged element is concatenated in the final head. This implies four rules: two to distinguish left and right concatenation and two over the lexical status of the trigger of *merge*.
– *Affix hopping* where the head of the trigger is concatenated with the head of the merged element. In the same way, there are four rules to distinguish left from right concatenations and lexical status of the trigger.

We use a simple way to encode the different possibilities of merging with $<$ and $>$. Pointing to the connective indicates head-movement and outside defines affix hopping. The lexical status does not need to be represented. In the following of this paper, only head movement will be used, thus we give this four rules:

Head-Movement:

$$\dfrac{\Gamma \vdash (r_{spec}, r_{tete}, r_{comp}) : A /_< B \qquad \Delta \vdash s : B}{\Gamma, \Delta \vdash (r_{spec}, r_{tete} \bullet s_{tete}, r_{comp} \bullet Concat(s_{-tete})) : A}\,[mg]$$

$$\dfrac{\Gamma \vdash (r_{spec}, r_{tete}, r_{comp}) : A_> / B \qquad \Delta \vdash s : B}{\Gamma, \Delta \vdash (r_{spec}, s_{tete} \bullet r_{tete}, r_{comp} \bullet Concat(s_{-tete})) : A}\,[mg]$$

$$\dfrac{\Delta \vdash s : B \qquad \Gamma \vdash (r_{spec}, r_{tete}, r_{comp}) : B_> \setminus A}{\Delta, \Gamma \vdash (Concat(s_{-tete}) \bullet r_{spec}, s_{tete} \bullet r_{tete}, r_{comp}) : A}\,[mg]$$

$$\dfrac{\Delta \vdash s : B \qquad \Gamma \vdash (r_{spec}, r_{tete}, r_{comp}) : B \setminus_< A}{\Delta, \Gamma \vdash (Concat(s_{-tete}) \bullet r_{spec}, r_{tete} \bullet s_{tete}, r_{comp}) : A}\,[mg]$$

3 Phases

3.1 Encoding Phases in MCG

Following [7], Chomsky assumes that the analysis of a sentence is driven by the verb which goes by two specific states: the *phases VP* and *cP*. Note that neither *tP* nor the decomposition over simple verb form as it is used in usual MCG are *phases* (this is illustrated in figure 2). Moreover, Chomsky claims that syntactic islands are defined by *phases*. That is, the content of a *phase* must be moved to its left-hand side in order to let it accessible. This step of the *phase* is called a *transfer*.

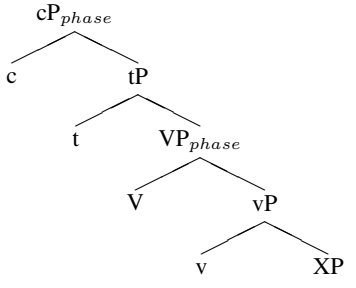


Fig. 2. Phases in verb structure

According to this structure, items on the right side of the *phase* can not be moved again. This syntactic island is called the Phase Impenetrability Condition (PIC). The definition of *phases* and the PIC are still under debate. Nevertheless, an interesting point for MCG is the simultaneous use of commutative and non-commutative properties of the framework to recognize them.

According to Chomsky, a *phase* is a node of the syntactic tree which triggers allowed moves. Because it is a node, this implies for MCG to combine two subproofs. It is not possible to simulate this with a single lexical item (which implies a terminal node). Also transfering is the realization of possible *moves* and the direct linking of hypotheses in a *cyclic move*.

Another argument in support of *phases* is in the semantic counterpart. This article does not present this second synchronized part of the calculus in MCG. Many different works claim that at particular step of the verb derivation some specific thematic roles could be assigned as in [10], [11] and [4]. [1] and [14] describe both arguments for this and the semantic tiers according to these assumptions. But, they do not include the idea of *phases*. Only remarks that simplification of derivation requires specific points in which continuation reductions must occur. We leave the presentation of consequences of *phases* on semantics to future works.

The analysis of a simple sentence derivation in standard MSG uses 4 items. In the following, the verb *read* will be used to illustrate this presentation for a simple active affirmative sentence.

1. The deep syntactic build of the verb: the verb and all its arguments (except the subject). $\vdash v \,/_< d$
2. Mode: introduces the subject category and, at least, the *accusative* case. $\vdash k \setminus d \setminus V \,/_< v$
3. Inflection: brings the inflection to the verb. $\vdash k \setminus t \,/_< V$
4. Comp: fully completes the analysis (a question mark, insert in relative clause, etc.) $\vdash c \,/\, t$

In order to keep control on the string associate to the verb, the derivation systematically uses head movement except *comp* which ends the derivation. We claim that *phases* can be encoded in MCG with a non-commutative order. The lexical realization of a *phase* item explicitly contains hypotheses in a non-commutative order.

$$\Delta_1, H_1; H_2 \Delta_2 \vdash A \tag{1}$$

This order makes a strong boundary in the derivation of the *phase*. It blocks *move* of elements in complement position to specifier. These are the only items that are lexically built with hypothesis different from the original definition of MCG [15], [1] and [2]. Using the $\odot_e$ rule of the PCL, the *phase* rule is defined as:

$$\frac{\Delta_s, \Delta_h, \Delta_c \vdash (s_s, s_h, s_c) : X \odot Y \quad \Gamma_s, X; Y, \Gamma_c \vdash (r_s, r_h, r_c) : Z}{\Gamma_s, \Delta_s, \Delta_h \vdash (r_s \bullet s_s, r_h, s_h \bullet s_c \bullet r_c) : Z} [phase]$$

The *phase* rule is the combination of a discharge of hypothesis in non-commutative order and a *transfer* step. This *transfer* is the realization of all possible *moves* and parts of *cyclic ones*. This new rule assembles several individual rules of PCL proof. It may be difficult to follow the derivation step by step. In the following, *phases* are given with more details. Thus, we note $[phase_1]$ the substitution part of the *phase* (the use of $\odot_e$ which combines two proofs) and $[phase_{trans}]$ moves which can be achieved after $[phase_1]$. The main condition to validate a *phase* is Δ_c and Γ_c are empty after $[phase_{trans}]$. This correspond to a phrase in complement position of a *phase* does not stand accessible. We note MCG$_{phase}$, MCG wit *phases*.

There is a direct consequence of this encoding of *phase* on the structure of the lexical item Hypotheses on its left-hand side of take the place of the complement of the head. Thus, all elements with which it should be combined must be in a specifier position. A more complex formula as in (1) will be built only with $\setminus$:

$$\Delta_1, H_1; H_2 \Delta_2 \vdash (s_s, s_h, s_c) B_n \setminus \ldots \setminus B_0 \setminus A \tag{2}$$

To take into account of the definition of the *phases* theory and the proposed encoding in MCG, a simple sentence will be divided into two *phases*: one with the *mode* and another with *comp*. Their lexical items are modified in this way into:

- mode: $\vdash k \setminus d \setminus V \,/\, v \Rightarrow V; v \vdash k \setminus d \setminus V$
- comp: $\vdash c \,/\, t \Rightarrow c; t \vdash c$

Hypotheses on the left-hand side of formulae receive a straightforward interpretation. They correspond to the conversion of a proof of a v to a proof of a V (or from a t to a c in the second one). Thus, we need to update the two other formulae in order to combine them with the two previous ones:

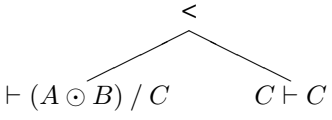
- verb: $\vdash v \,/\, d \Rightarrow \vdash (V \odot v) \,/\, d$
- inflection: $\vdash k \setminus t \,/\, V \Rightarrow \vdash k \setminus (c \odot t) \,/\, V$

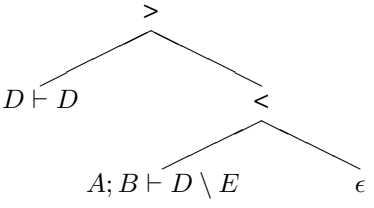
Note that the use of $\odot$ in a formula do not imply that this item is one of a *phase*. It means that they take an active part in the construction of the verb. Here, we have extended the main category of the item to the previous one, combined with the category of the following *phase* category with a $\odot$.

In fact, each head item drives a specific part of all elements are included under their relation to the head. But, the unloading of hypotheses (the realization of the *phase*) occurs later. Here, the logical account of the framework updates the derivation in a way that there is no direct intuition with the syntactic tree. We present a simple example of interpretation of a *phase* in a proof into a tree: on one hand the starting part of the derivation and on the other the proof which results in the *phase*:

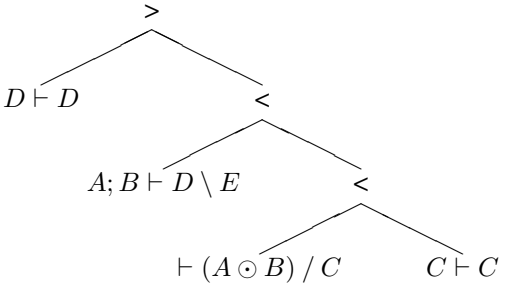
$$\frac{\vdash (A \odot B) \,/\, C \quad C \vdash C}{C \vdash (A \odot B)}\,[mg] \qquad \frac{A; B \vdash D \setminus E \quad D \vdash D}{D, A; B \vdash E}\,[mg]$$

Which correspond to the two following syntactic trees:





The syntactic tree of the *phase* is not a simple leaf but a tree with an empty position in which the other derivation is substitued:



And it is clear that all information derived in the second tree before the *phase* combination must be updated after. Finally, for the same reasons as for *merge*, Head Movement and Affix Hopping are needed through the *phase*. The same connectives are used with the same effects. One instance is:

$$\frac{\Delta_s, \Delta_h, \Delta_c \vdash (s_s, s_h, s_c) : X \odot_< Y \quad \Gamma_s, X; Y, \Gamma_c \vdash (r_s, r_h, r_c) : Z}{\Gamma_s \Delta_s, \Delta_h \vdash (r_s \bullet s_s, s_h \bullet r_h, s_c \bullet r_c) : Z}\,[phase]$$

where the head of the triplet is the concatenation of the two heads. The following section presents an application of this rules in a full analysis of a simple sentence.

3.2 Derivation of a Simple Sentence

To illustrate derivation with *phases*, this section presents the complete derivation of a simple sentence. However, extending the analysis to more complex syntactic structures has resulted in defining the lexical entries corresponding with the same principles. We present the analysis of the following example:

(1) The children read a book.

This simple sentence uses the affirmative form, the verb will take the four previous steps. In order to build noun phrase with MCG with generative theory, we combine a determiner $\vdash\ (k \otimes d)\ /\ n$ with a noun $\vdash\ n$. It produces a constituent of category d (for *determinal phrase*) which lacks the assignment of syntactic case (k). Then, the following lexicon is used:

$articles$	$\vdash (\epsilon, the, \epsilon) : k \otimes d\ /\ n$ $\vdash (\epsilon, a, \epsilon) : k \otimes d\ /\ n$
$noun$	$\vdash (\epsilon, chlidren, \epsilon) : n$ $\vdash (\epsilon, book, \epsilon) : n$
$verb$ $(mode)$ $(inflection)$ $(comp)$	$\vdash (\epsilon, read, \epsilon) : (V \odot v)_{<}\ /\ d$ $V; v \vdash (\epsilon, \epsilon, \epsilon) : k \setminus d \setminus V$ $\vdash (\epsilon, -, \epsilon) : k \setminus (c \odot t)\ /_{<}\ V$ $c; t \vdash (\epsilon, \epsilon, \epsilon) : c$

The derivation is a proof, so each main element (the head of each phrases) drives its subproof. The consequence is that the proof is built in several parts adjusted against each other by the discharge of hypotheses. This property makes the presentation more complex, but the search for proofs (*i.e.* the parsing) could be done in parallel. In the following, each verb step is presented. Note that the normalization of PCL from [3] ensures that we could combine each step one by one as presented here.

Step 1: In the first verb step of the derivation, the first position of the verb is saturated by a hypothesis of category d. It corresponds to the position of the main component that occupies the object position in the sentence. The object is not directly inserted in the derivation because all its features are not yet marked by hypotheses: the position of k (mark of the syntactic case assignment[3]) is missed. This is a departure from MG, in which all phrases are directly inserted in the derivation. Here, we only mark positions for insertion.

$$\frac{\vdash (\epsilon, read, \epsilon) : (V \odot v)_{<}\ /\ d \quad d \vdash (\epsilon, u, \epsilon) : d}{d \vdash (\epsilon, read, u) : (V \odot v)}\ [mg]$$

This is the end of the first verb step. The result must be inserted in another proof which contains hypothesis V and v. On one hand, the interpretation of the non-commutative relation between V and v in the type could be the saturation of all arguments of an element of type v to produce a V.

[3] In the semantic part of the calculus, this position is also used for the thematic role assignement.

Step 2: These hypotheses are in the proof driven by the second entry of the verb: *mode* which ends the first *phase*. First, we need to saturate all positions of *mode* with lexical hypothesis, and then we could:

1. combine the result with the first verb step,
2. introduce the object of the sentence with a *move* in the transfer part of the *phase*.

The lexical item of *mode* is merged with a hypothesis k and next a d:

$$\dfrac{d \vdash (\epsilon, w, \epsilon) : d \qquad \dfrac{k \vdash (\epsilon, v, \epsilon) : k \quad V; v \vdash (\epsilon, \epsilon, \epsilon) : k \setminus d \setminus V}{k, V; v \vdash (v, \epsilon, \epsilon) : d \setminus V}\,[mg]}{d, k, V; v \vdash (w\ v, \epsilon, \epsilon) : V}\,[mg]$$

At this point of the derivation, this result is combined with the first verb step with a $[phase]$ with head movement. Note that the string of the head of the discharged is concatenated to the string of the head position in order to keep all structural information over the verb accessible to the full derivation. The substitution part of the *phase* produces:

$$\dfrac{d \vdash (\epsilon, read, u) : (V \odot v)_< \quad d, k, V; v \vdash (w\ v, \epsilon, \epsilon) : V}{d, k, d \vdash (w\ v, read, u) : V}\,[phase_1]$$

Before ending the *phase*, we perform a *move* with all positions of the utterance's object (they are now in the proof). Then it could be introduced in the derivation with a transfer. In parallel, the determiner phrase is built with the two lexical entries "a" and "book" by a *merge*:

$$\dfrac{\vdash (\epsilon, a, \epsilon) : (k \otimes d) \,/\, n \quad \vdash (\epsilon, book, \epsilon) : n}{\vdash (\epsilon, a, book) : k \otimes d}\,[mg]$$

Thereby, it is discharged in the main proof. The choice of the d with which to carry out the unloading is not left by chance. The derivation must be continued with the one which empties the previous verb step with a *move*. This is exactly the interpretation of transfer part in [7].

$$\dfrac{\vdash (\epsilon, a, book) : k \otimes d \quad d, k, d \vdash (w\ v, read, u) : V}{d \vdash (w\ a\ book, read, \epsilon) : V}\,[phase_{trans}]$$

This *move* substitute in the newest variable as define in [1]. It is the full realization of the constituent. In this *phase*, non-commutativity and commutativity are both used. Non-commutativity in order to keep the structure of the verb and commutativity to unload hypotheses of the nominal phrase. This underlies the assumption that the order of the features of the noun could not be presupposed. This is reinforced by the analysis of questions where that object constituent undergoes one more *move* and then must be explicitly transfered from right to left part of the *phase*. Now, it is the end of the second verb step. The derivation continues by preparing the third one.

Step 3: This part of the derivation must be combined with the next lexical entry of the verb, the *inflection*. In this part of the verb step, it was merged with the previous result and next with a k hypothesis - the position of the subject case:

$$\frac{k \vdash (\epsilon, z, \epsilon) : k \qquad \dfrac{\vdash (\epsilon, -, \epsilon) : k \setminus (c \odot t) /_< V \quad d \vdash (w\ a\ book, read, \epsilon) : V}{d \vdash (\epsilon, read, w\ a\ book) : k \setminus (c \odot t)}\,[mg]}{k, d \vdash (z, read, w\ a\ book) : (c \odot t)}\,[mg]$$

This allows to discharge hypothesis about the subject constituent which it also build:

$$\frac{\vdash (\epsilon, the, \epsilon) : (k \otimes d) \,/\, n \quad \vdash (\epsilon, children, \epsilon) : n}{\vdash (\epsilon, the, children) : k \otimes d}\,[mg]$$

And unloaded:

$$\frac{\vdash (\epsilon, the, children) : k \otimes d \quad k, d \vdash (z, read, w\ a\ book) : (c \odot t)}{\vdash (the\ children, read, a\ book) : (c \odot t)}\,[mv]$$

Step 4: This example stands for a very simple sentence, then the last verb step corresponds only to the combination of the current bypass with the lexical entry *comp* by a *phase* with nothing in the transfer part:

$$\frac{\vdash (the\ children, read, a\ book) : (c \odot t) \quad c; t \vdash (\epsilon, \epsilon, \epsilon) : c}{\vdash (\epsilon, \epsilon, the\ children\ read\ a\ book) : c}\,[phase]$$

This ends the last *phase* and thus the derivation. The proof matches the string *The children read a book*. An important distinction with the previous versions of these grammars is in the use of lexical item without phonological part. Here, only the lexical items used in the *phase* process are necessary, but the structure of items for *phases* imposes a strict order in their pooling.

3.3 Question

In the previous example, the transfer part of the two *phases* is not really efficient. The first one introduced the object of the utterance and the second only ended the derivation. For questions, the $comp$ item is more complex because it introduces the last feature of the object. This time, its lexical item is : $c; t \vdash (\epsilon, \epsilon, \epsilon) : wh \setminus c$. And it is only afterward that a hypothesis wh is introduced that the object could be introduced in the derivation. But it means that in the previous *phase*, the constituent mark must be transferred from the left to the right part of the first *phase*. This is done with a *cyclic move*: the introduction of a new hypothesis $k \otimes d \vdash k \otimes d$, which explicitly connect the two hypotheses.

In the lexicon, only $comp$ is modified and an item for which is added :

$$which \vdash (\epsilon, \epsilon, \epsilon) : (wh \otimes (k \otimes d))$$

The derivation before the *phase* is still the same.

1. First step procedure:

$$\frac{\vdash (\epsilon, read, \epsilon) : (V \odot v)_< \,/\, d \quad d \vdash (\epsilon, u, \epsilon) : d}{d \vdash (\epsilon, read, u) : (V \odot v)} [mg]$$

2. Saturation of positions of $mode$:

$$\frac{d \vdash (\epsilon, w, \epsilon) : d \qquad \dfrac{k \vdash (\epsilon, v, \epsilon) : k \quad V; v \vdash (\epsilon, \epsilon, \epsilon) : k \setminus d \setminus V}{k, V; v \vdash (v, \epsilon, \epsilon) : d \setminus V} [mg]}{d, k, V; v \vdash (w\ v, \epsilon, \epsilon) : V} [mg]$$

3. Construction of the object constituent:

$$\frac{\vdash (\epsilon, which, \epsilon) : (wh \otimes (k \otimes d)) \,/\, n \quad \vdash (\epsilon, book, \epsilon) : n}{\vdash (\epsilon, which, book) : (wh \otimes (k \otimes d))} [mg]$$

We get all the necessary material to process the *phase*. Its first part combines:

$$\frac{d \vdash (\epsilon, read, u) : (V \odot v) \quad d, k, V; v \vdash (w\ v, \epsilon, \epsilon) : V}{d, k, d \vdash (w\ v, read, u) : V} [phase_1]$$

In the treatment of this utterance, the transfer part is not able to directly discharge the two hypotheses of the determiner phrase. A *cyclic move* is used in order to store the access to this element. At the same time, the *phase* move it on its left part:

$$\frac{k \otimes d \vdash (\epsilon, W, \epsilon) : k \otimes d \quad d, k, d \vdash (w\ v, read, u) : V}{d, k \otimes d \vdash (w\ W, read, \epsilon) : V} [phase_{trans}]$$

The derivation continues with the same third step and produces:

$$k \otimes d \vdash (the\ children, read, W) : (c \odot t)$$

Finally, before the last *phase*, the derivation introduces a wh hypothesis which will allow the *move* of object after the first part of the *phase* realization. The *move* of the transfer part is:

$$\frac{\vdash (\epsilon, which, book) : (wh \otimes (k \otimes d))\ wh, k \otimes d \vdash (y\, the\, children, read, W) : c}{\vdash (which\, book, \epsilon,\ the\, children\, read) : c} [phase_{trans}]$$

In this example, only the transfer in the first *phase*, which accounts for the *cyclic move* of the constituent allows to complete the derivation.

3.4 Blocked Derivation with PIC

A very important point in the definition of the *phase* rule is the fact that the complementizer part of hypothesis must be removed. This property encodes the Phase Impenetrability Condition. The previous one does not contain such problem because the transfer part of the first *phase* achieves all *moves* which empty complementizer hypotheses.

A simple example extracted from the previous one is the case where the k hypothesis in the second step of the verb is not included. Thus the derivation must failed because one hypothesis is away. The lexical entry corresponding to $mode$ is:

$$V; v \vdash (\epsilon, \epsilon, \epsilon) : d \setminus V$$

which produces a conclusion of a proof of type V with only d in the left hand side:

$$d, V; v \vdash (w\epsilon, \epsilon) : V$$

The result of the first part of the *phase* is:

$$\frac{d \vdash (\epsilon, read, u) : (V \odot v) \quad d, V; v \vdash (w\ v, \epsilon, \epsilon) : V}{d, d \vdash (w\ v, read, u) : V}\ [phase_1]$$

And the transfer part does not contribute to this step. The part of the Γ_c of the *phase* rule is not removed. The structure of the proof which blocks the derivation is the case where the constituent is in complementizer position of the head.

We would remark that derivations with *phases* immediately block the process unlike traditional MCG or MG which perform the full derivation before concluding that a specific feature stand at the end (and reject the derivation).

Even if this example is quite simple, it shows that the encoding of PIC directly uses properties of the MCG. Unlike the other constraints, we do not need to propose new rules. That insure to keep the same generative power for MCG_{phase}. The derivation strictly controls the structure and check internal relations.

4 Conclusion

The main aim of this paper is to introduce the concept of *phase* from minimalism into type logical grammars, simulating the generative theory of Chomsky which has been an open question since [7]. It involves the introduction of a new rule into the system, and highlights commutative and non-commutative relationships between elements of the parsing process. Moreover, this addition is not *ad hoc* as it allows full use of the properties of PCL underlying the formalism. This new rule is the composition of a substitution of hypotheses in commutative relations, followed by a transfer that is either the realization of a *move* became possible, or a *cyclic move*. This proposal also involves a new linguistic interpretation of *cyclic move*.

A full description of the system would require additional details [1]. However, we emphasize the role played by *phases* at the syntactic level to define *islands* where the encoding of PIC is simply the use of logical properties of the framework. Furthermore, we claim that the use of *phases* at semantic level corresponds to the introduction of thematic role predicates (variables related to the reification of formulas and substitution of variables). They also mark points in the semantic tiers where the context must be reduced. It plays a crucial role at the semantic level by marking reduction point for continuations.

The introduction of these *phases* confirms the use of a logical system simultaneously handling relations commutative and non-commutative at plays in linguistic analysis. Distributing the properties on each component used in this analysis can produce fine performances A remaining issue for this description is the formalization of the *phase* as reduction point at the semantic level that would reduce ambiguities of scope of quantifiers. In addition, the study of equivalence between the MCG with *phases* and MG remains an open question.

Acknowledgments. The author would like to express his gratitude to reviewers for their precise remarks, Corinna Anderson, Sai Qian and Sandrine Ribeau for their readings.

References

1. Amblard, M.: Calcul de représentations sémantiques et syntaxe générative: les grammaires minimalistes catégorielles. Ph.D. thesis, université de Bordeaux 1 (2007)
2. Amblard, M., Lecomte, A., Retore, C.: Categorial minimalist grammars: from generative syntax to logical forms. Linguistic Analysis 6(1-4), 273–308 (2010)
3. Amblard, M., Retore, C.: Natural deduction and normalisation for partially commutative linear logic and lambek calculus with product. In: Computation and Logic in the Real World, CiE 2007 (2007)
4. Baker, M.: Thematic Roles and Syntactic Structure. In: Haegeman, L. (ed.) Elements of Grammar, Handbook of Generative Syntax, pp. 73–137. Kluwer, Dordrecht (1997)
5. Chomsky, N.: Syntactic Structures. Mouton, The Hague (1957)
6. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
7. Chomsky, N.: Derivation by phase. ms. MIT, Cambridge (1999)
8. de Groote, P.: Partially commutative linear logic: sequent calculus and phase semantics. In: Abrusci, V.M., Casadio, C. (eds.) Third Roma Workshop: Proofs and Linguistics Categories – Applications of Logic to the analysis and implementation of Natural Language, pp. 199–208. CLUEB, Bologna (1996)
9. de Groote, P.: Towards abstract categorial grammars. In: ACL 2001 (2001)
10. Hale, K.: On argument structure and the lexical expression of syntactic relations. The View from Building, vol. 20. MIT Press, Ithaca (1993)
11. Kratzer, A.: External arguments. In: Benedicto, E., Runner, J. (eds.) Functional Projections. University of Massachussets, Amherst (1994)
12. Lambek, J.: The mathematics of sentence structures. American mathematical monthly (1958)
13. Lecomte, A.: Categorial grammar for minimalism. Language and Grammar: Studies in Mathematical Linguistics and Natural Language CSLI Lecture Notes, vol. 168, pp. 163–188 (2005)
14. Lecomte, A.: Semantics in minimalist-categorial grammars. Formal Grammar (2008)
15. Lecomte, A., Retoré, C.: Extending Lambek grammars: a logical account of minimalist grammars. In: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL 2001, pp. 354–361. ACL, Toulouse (2001)
16. Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, ch. 2, pp. 93–178. Elsevier, Amsterdam (1997)
17. Morrill, G.: Type logical grammar. Categorial Logic of Signs (1994)
18. Muskens, R.: Languages, lambdas and logic. Resource Sensitivity in Binding and Anaphora (2003)
19. Pollard, C.: Convergent grammars. Tech. rep., The Ohio State University (2007)
20. Retoré, C.: Pomset logic: a non-commutative extension of classical linear logic. In: de Groote, P., Hindley, J.R. (eds.) TLCA 1997. LNCS, vol. 1210, pp. 300–318. Springer, Heidelberg (1997)
21. Retoré, C.: A description of the non-sequential execution of petri nets in partially commutative linear logic. Logic Colloquium 99 Lecture Notes in Logic, pp. 152–181 (2004)
22. Stabler, E.: Derivational minimalism. LACL 1328 (1997)
23. Stabler, E.: Recognizing head movement. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, p. 245. Springer, Heidelberg (2001)
24. Steedman, M.: Combinatory grammars and parasitic gaps. In: Natural Language and Linguistic Theory, vol. 5 (1987)

Using Logic in the Generation of Referring Expressions

Carlos Areces[1], Santiago Figueira[2,⋆], and Daniel Gorín[3]

[1] INRIA Nancy, Grand Est, France
areces@loria.fr
[2] Departamento de Computación, FCEyN, UBA and CONICET, Argentina
[3] Departamento de Computación, FCEyN, UBA, Argentina
{santiago,dgorin}@dc.uba.fr

Abstract. The problem of generating referring expressions (GRE) is an important task in natural language generation. In this paper, we advocate for the use of logical languages in the output of the content determination phase (i.e., when the relevant features of the object to be referred are selected). Many different logics can be used for this and we argue that, for a particular application, the actual choice shall constitute a compromise between expressive power (how many objects can be distinguished), computational complexity (how difficult it is to determine the content) and realizability (how often will the selected content be realized to an idiomatic expression). We show that well-known results from the area of computational logic can then be transferred to GRE. Moreover, our approach is orthogonal to previous proposals and we illustrate this by generalizing well-known content-determination algorithms to make them parametric on the logic employed.

1 Generating Referring Expressions

The generation of referring expressions (GRE) –given a context and an element in that context generate a grammatically correct expression in a given natural language that uniquely represents the element– is a basic task in natural language generation, and one of active research (see [4,5,6,20,8] among others). Most of the work in this area is focused on the *content determination* problem (i.e., finding a collection of properties that singles out the target object from the remaining objects in the context) and leaves the actual *realization* (i.e., expressing a given content as a grammatically correct expression) to standard techniques[1].

However, there is yet no general agreement on the basic representation of both the input and the output to the problem; this is handled in a rather ad-hoc way by each new proposal instead.

Krahmer et al. [17] make the case for the use of *labeled directed graphs* in the context of this problem: graphs are abstract enough to express a large number of

⋆ S. Figueira was partially supported by CONICET (grant PIP 370) and UBA (grant UBACyT 20020090200116).

[1] For exceptions to this practice see, e.g., [16,21].

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 17–32, 2011.

domains and there are many attractive, and well-known algorithms for dealing with this type of structures. Indeed, these are nothing other than an alternative representation of relational models, typically used to provide semantics for formal languages like first and higher-order logics, modal logics, etc. Even valuations, the basic models of propositional logic, can be seen as a single-pointed labeled graph. It is not surprising then that they are well suited to the task.

In this article, we side with [17] and use labeled graphs as input, but we argue that an important notion has been left out when making this decision. Exactly because of their generality, graphs do not define, by themselves, a unique notion of *sameness*. When do we say that two nodes in the graphs can or cannot be referred uniquely in terms of their properties? This question only makes sense once we fix a certain level of expressiveness which determines when two graphs, or two elements in the same graph, are equivalent.

Expressiveness can be formalized using structural relations on graphs (isomorphisms, etc.) or, alternatively, logical languages. Both ways are presented in §2, where we also discuss how fixing a notion of expressiveness impacts on the number of instances of the GRE problem that have a solution; the computational complexity of the GRE algorithms involved; and the complexity of the surface realization problem. We then investigate the GRE problem in terms of different notions of expressiveness. We first explore in §3 how well-known algorithms from computational logic can be applied to GRE. This is a systematization of the approach of [1], and we are able to answer a complexity question that was left open there. In §4 we take the opposite route: we take the well-known GRE-algorithm of [17], identify its underlying expressivity and rewrite in term of other logics. We then show in §5 that both approaches can be combined and finally discuss in §6 the size of an RE relative to the expressiveness employed. We conclude in §7 with a short discussion and prospects for future work.

2 Measuring Expressive Power

Relational structures are very suitable for representing *situations* or *scenes*. A relational structure (also called "relational model") is a non-empty set of objects –the *domain*– together with a collection of relations, each with a fixed arity.

Formally, assume a fixed and finite (but otherwise arbitrary) vocabulary of n-ary relation symbols.[2] A relational model $\mathcal{M}$ is a tuple $\langle \Delta, \| \cdot \| \rangle$ where Δ is a nonempty set, and $\| \cdot \|$ is a suitable interpretation function, that is, $\|r\| \subseteq \Delta^n$ for every n-ary relation symbol r in the vocabulary. We say that $\mathcal{M}$ is *finite* whenever Δ is finite. The *size* of a model $\mathcal{M}$ is the sum $\#\Delta + \#\| \cdot \|$, where $\#\Delta$ is the cardinality of Δ and $\#\| \cdot \|$ is the sum of the sizes of all relations in $\| \cdot \|$.

Figure 1 below shows how we can represent a scene as a relational model. Intuitively, a, b and d are dogs, while c and e are cats; d is a small beagle; b and c are also small. We read $\mathit{sniffs}(d, e)$ as "d *is sniffing* e".

[2] Constants and function symbols can be represented as relations of adequate arity.

$$\begin{aligned}\Delta &= \{a, b, c, d, e\}\\ \|dog\| &= \{a, b, d\}\\ \|cat\| &= \{c, e\}\\ \|beagle\| &= \{d\}\\ \|small\| &= \{b, c, d\}\\ \|sniffs\| &= \{(a, a), (b, a), (c, b), (d, e), (e, d)\}\end{aligned}$$

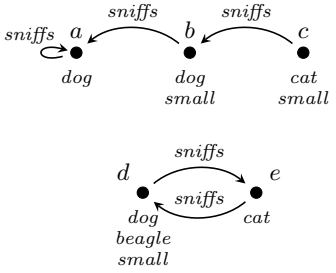


Fig. 1. Graph representation of scene $\mathcal{S}$

Logical languages are fit for the task of (formally) *describing* elements of a relational structure. Consider, e.g., the classical language of first-order logic (with equality), $\mathcal{FO}$, given by:

$$\top \mid x_i \not\approx x_j \mid r(\bar{x}) \mid \neg\gamma \mid \gamma \wedge \gamma' \mid \exists x_i.\gamma$$

where $\gamma, \gamma' \in \mathcal{FO}$, r is an n-ary relation symbol and $\bar{x}$ is an n-tuple of variables. As usual, $\gamma \vee \gamma'$ and $\forall x.\gamma$ are short for $\neg(\neg\gamma \wedge \neg\gamma')$ and $\neg\exists x.\neg\gamma$, respectively. Formulas of the form $\top$, $x_i \not\approx x_j$ and $r(\bar{x})$ are called *atoms*.[3] Given a relational model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ and a formula γ with free variables[4] among $x_1 \dots x_n$, we inductively define the *extension* or *interpretation* of γ as the set of n-tuples $\|\gamma\|^n \subseteq \Delta^n$ that satisfy:

$$\begin{aligned}\|\top\|^n &= \Delta^n & \|x_i \not\approx x_j\|^n &= \{\bar{a} \mid \bar{a} \in \Delta^n, a_i \neq a_j\}\\ \|\neg\delta\|^n &= \Delta^n \setminus \|\delta\|^n & \|r(x_{i_1} \dots x_{i_k})\|^n &= \{\bar{a} \mid \bar{a} \in \Delta^n, (a_{i_1} \dots a_{i_k}) \in \|r\|\}\\ \|\delta \wedge \theta\|^n &= \|\delta\|^n \cap \|\theta\|^n & \|\exists x_l.\delta\|^n &= \{\bar{a} \mid \bar{a}e \in \|\delta'\|^{n+1} \text{ for some } e\}\end{aligned}$$

where $1 \leq i, j, i_1, \dots, i_k \leq n$, $\bar{a} = (a_1 \dots a_n)$, $\bar{a}e = (a_1 \dots a_n, e)$ and δ' is obtained by replacing all occurrences of x_l in δ by x_{n+1}. When the cardinality of the tuples involved is known from the context we will just write $\|\gamma\|$ instead of $\|\gamma\|^n$.

With a language syntax and semantics in place, we can now formally define the problem of $\mathcal{L}$-GRE for a target set of elements T (we slightly adapt the definition in [1]):

$\mathcal{L}$-GRE Problem	
Input:	a model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ and a nonempty target set $T \subseteq \Delta$.
Output:	a formula $\varphi \in \mathcal{L}$ such that $\|\varphi\| = T$ if it exists, and $\perp$ otherwise.

When the output is not $\perp$, we say that φ is an *$\mathcal{L}$-referring expression ($\mathcal{L}$-RE) for T in $\mathcal{M}$*. Simply put then, the output of the $\mathcal{L}$-GRE problem is a formula of $\mathcal{L}$ whose interpretation in the input model is the target set, if such a formula

[3] For technical reasons, we include the inequality symbol $\not\approx$ as primitive. Equality can be defined using negation.

[4] W.l.o.g. we assume that no variable appears both free and bound, that no variable is bound twice, and that the index of bound variables in a formula increases from left to right.

exists. This definition applies also to the GRE for objects of the domain by taking a singleton set as target.

By using formulas with n free variables one could extend this definition to describe n-ary relations; but here we are only interested in describing subsets of the domain. Actually, we shall restrict our attention a little further:

Convention 1. We will only consider relational models with unary and binary relation symbols (i.e., labeled graphs). We will consistently use p for a unary relation symbol (and called it a *proposition*) and r for a binary relation symbol.

This convention captures the usual models of interest when describing scenes as the one presented in Figure 1. Accommodating relations of higher arity in our theoretical framework is easy, but it might affect computational complexity.

2.1 Choosing the Appropriate Language

Given a model $\mathcal{M}$, there might be an infinite number of formulas that uniquely describe a target (even formulas which are not logically equivalent might have the same interpretation once a model is fixed). Despite having the same interpretation in $\mathcal{M}$, they may be quite different with respect to other parameters.

As it is well known in the automated text generation community, different realizations of the same content might result in expressions which are more or less appropriate in a given context. Although, as we mentioned in the introduction, we will only address the content determination part (and not the surface realization part) of the GRE problem, we will argue that generating content using languages with different expressive power can have an impact in the posterior surface generation step.

Let us consider again the scene in Figure 1. Formulas γ_1–γ_4 shown in Table 1 are all such that γ_i uniquely describes b (i.e., $\|\gamma_i\| = \{b\}$) in model $\mathcal{S}$. Arguably, γ_1 can be easily realized as *"the small dog that sniffs a dog"*. Syntactically, γ_1 is characterized as a positive, conjunctive, existential formula (i.e., it contains no negation and uses only conjunction and existential quantification). Expressions with these characteristics are, by large, the most commonly found in corpora as those compiled in [22,9,7]. Formula γ_2, on the other hand, contains negation, disjunction and universal quantification. It could be realized as *"the small dog that only sniffs things that are not cats"* which sounds unnatural. Even a small change in the form of γ_2 makes it more palatable: rewrite it using $\exists$, $\neg$, and $\wedge$ to obtain *"the small dog that is not sniffing a cat"*. Similarly, γ_3 and γ_4 seem computationally harder to realize than γ_1: γ_3 contains an inequality (*"the dog sniffing another dog"*), while the quantified object appears in the first argument position in the binary relation in γ_4 (*"the dog that is sniffed by a small cat"*).

Summing up, we can ensure, already during the content determination phase, certain properties of the generated referring expression by paying attention to the formal language used in the representation. And we can do this, even before taking into account other fundamental linguistics aspects that will make certain realization preferable like saliency, the cognitive capacity of the hearer (can she recognize a *beagle* from another kind of dog?), etc.

Table 1. Alternative descriptions for object b in the model shown in Figure 1

$$\gamma_1 : dog(x) \wedge small(x) \wedge \exists y.(sniffs(x,y) \wedge dog(y))$$
$$\gamma_2 : dog(x) \wedge small(x) \wedge \forall y.(\neg cat(y) \vee \neg sniffs(x,y))$$
$$\gamma_3 : dog(x) \wedge \exists y.(x \not\approx y \wedge dog(y) \wedge sniffs(x,y))$$
$$\gamma_4 : dog(x) \wedge \exists y.(cat(y) \wedge small(y) \wedge sniffs(y,x))$$

As a concrete example, let $\mathcal{FO}^-$ be the fragment of $\mathcal{FO}$-formulas where the operator $\neg$ does not occur (but notice that atoms $x_i \not\approx x_j$ are permitted). By restricting content determination to $\mathcal{FO}^-$, we ensure that formulas like γ_2 will not be generated. If we ban $\not\approx$ from the language, γ_3 is precluded.

The fact that the representation language used has an impact on content determination is obvious, but it has not received the attention it deserves. Areces et al. [1] use different description logics (a family of formal languages used in knowledge representation, see [2]) to classify, and give a formal framework to previous work on GRE. Let us quickly introduce some of these languages as we will be mentioning them in future sections. Using description logics instead of $\mathcal{FO}$ fragments is just a notational issue, as most description logics can be seen as implicit fragments of $\mathcal{FO}$. For example, the language of the description logic $\mathcal{ALC}$, syntactically defined as the set of formulas,

$$\top \mid p \mid \neg\gamma \mid \gamma \wedge \gamma' \mid \exists r.\gamma$$

(where p is a propositional symbol, r a binary relational symbol, and $\gamma, \gamma' \in \mathcal{ALC}$) corresponds to a syntactic fragment of $\mathcal{FO}$ without $\not\approx$, as shown by the standard translation τ_x:

$$\tau_{x_i}(\top) = \top \qquad \tau_{x_i}(\gamma_1 \wedge \gamma_2) = \tau_{x_i}(\gamma_1) \wedge \tau_{x_i}(\gamma_2)$$
$$\tau_{x_i}(p) = p(x_i) \qquad \tau_{x_i}(\exists r.\gamma) = \exists x_{i+1}.(r(x_i, x_{i+1}) \wedge \tau_{x_{i+1}}(\gamma))$$
$$\tau_{x_i}(\neg\gamma) = \neg\tau_{x_i}(\gamma)$$

Indeed, given a relational model $\mathcal{M}$, the extension of an $\mathcal{ALC}$ formula φ in $\mathcal{M}$ exactly coincides with the extension of $\tau_{x_1}(\varphi)$ (see, e.g., [2]). Thanks to this result, for any formula φ of $\mathcal{ALC}$ and its sublanguages we can define $\|\varphi\| = \|\tau_{x_1}(\varphi)\|$. Coming back to our previous example, by restricting content generation to $\mathcal{ALC}$ formulas (or equivalently, the corresponding fragment of $\mathcal{FO}$) we would avoid formulas like γ_3 (no equality) and γ_4 (quantified element appears always in second argument position).

Generation is discussed in [1] in terms of different description logics like $\mathcal{ALC}$ and $\mathcal{EL}$ ($\mathcal{ALC}$ without negation). We will extend the results in that paper, considering for instance $\mathcal{EL}^+$ ($\mathcal{ALC}$ with negation allowed only in front of unary relations) but, more generally, we take a model theoretic approach and argue that the primary question is not whether one should use one or other (description) logic for content generation, but rather which are the *semantic differences* one cares about. This determines the required logical formalism but also impacts on both the content determination and the surface realization problems. Each

logical language can be seen as a compromise between expressiveness, realizability and computational complexity. The appropriate selection for a particular GRE task should depend on the actual context.

2.2 Defining *Sameness*

Intuitively, given a logical language $\mathcal{L}$ we say that an object u in a model $\mathcal{M}_1$ is similar in $\mathcal{L}$ to an object v in a model $\mathcal{M}_2$ whenever all $\mathcal{L}$-formulas satisfied by u are also satisfied by v. Formally, let $\mathcal{M}_1 = \langle \Delta_1, \|\cdot\|_1 \rangle$ and $\mathcal{M}_2 = \langle \Delta_2, \|\cdot\|_2 \rangle$ be two relational models with $u \in \Delta_1$ and $v \in \Delta_2$; we follow the terminology of [1] and say that u *is* $\mathcal{L}$*-similar to* v (notation $u \overset{\mathcal{L}}{\rightsquigarrow} v$) whenever $u \in \|\gamma\|_1$ implies $v \in \|\gamma\|_2$, for every $\gamma \in \mathcal{L}$. It is easy to show that $\mathcal{L}$-similarity is reflexive for all $\mathcal{L}$, and symmetric for languages that contain negation.

Observe that $\mathcal{L}$-similarity captures the notion of 'identifiability in $\mathcal{L}$'. If we take $\mathcal{M}_1$ and $\mathcal{M}_2$ to be the same model, then an object u in the model can be uniquely identified using $\mathcal{L}$ if there is no object v different from u such that $u \overset{\mathcal{L}}{\rightsquigarrow} v$. In other words, if there are two objects u and v in a model $\mathcal{M}$ such that $u \overset{\mathcal{L}}{\rightsquigarrow} v$, then the $\mathcal{L}$-GRE problem with input $\mathcal{M}$ and target $T = \{u\}$ will not succeed since for all formulas $\gamma \in L$ we have $\{u, v\} \subseteq \|\gamma\| \neq \{u\}$.

The notion of $\mathcal{L}$-similarity then, gives us a handle on the $\mathcal{L}$-GRE problem. Moreover, we can recast this definition in a structural way, so that we do not need to consider infinitely many $\mathcal{L}$-formulas to decide whether u is $\mathcal{L}$-similar to v. We can reinterpret $\mathcal{L}$-similarity in terms of standard model-theoretic notions like isomorphisms or bisimulations which describe structural properties of the model, instead. Given two models $\langle \Delta_1, \|\cdot\|_1 \rangle$ and $\langle \Delta_2, \|\cdot\|_2 \rangle$, consider the following properties of a binary relation $\sim \subseteq \Delta_1 \times \Delta_2$ (cf. Convention 1):

ATOM_L: If $u_1 \sim u_2$, then $u_1 \in \|p\|_1 \Rightarrow u_2 \in \|p\|_2$
ATOM_R: If $u_1 \sim u_2$, then $u_2 \in \|p\|_2 \Rightarrow u_1 \in \|p\|_1$
REL_L : If $u_1 \sim u_2$ and $(u_1, v_1) \in \|r\|_1$, then $\exists v_2$ s.t. $v_1 \sim v_2$ and $(u_2, v_2) \in \|r\|_2$
REL_R : If $u_1 \sim u_2$ and $(u_2, v_2) \in \|r\|_2$, then $\exists v_1$ s.t. $u_1 \sim v_1$ and $(u_1, v_1) \in \|r\|_1$
INJ_L : $\sim$ is an injective function (when restricted to its domain)
INJ_R : $\sim^{-1}$ is an injective function (when restricted to its domain)

We will say that a non-empty binary relation $\sim$ is an $\mathcal{L}$*-simulation* when it satisfies the properties indicated in Table 2. For example, a non-empty binary relation that satisfies ATOM_L, and REL_L is an $\mathcal{EL}$-simulation, as indicated in row 4 of Table 2. Moreover, we will say that an object v $\mathcal{L}$*-simulates* u (notation $u \underline{\overset{\mathcal{L}}{\rightarrow}} v$) if there is a relation $\sim$ satisfying the corresponding properties such that $u \sim v$. The following is a fundamental model-theoretic result:

Theorem 1. *If* $\mathcal{M}_1 = \langle \Delta_1, \|\cdot\|_1 \rangle$ *and* $\mathcal{M}_2 = \langle \Delta_2, \|\cdot\|_2 \rangle$ *are finite models,* $u \in \Delta_1$ *and* $v \in \Delta_2$*, then* $u \overset{\mathcal{L}}{\rightsquigarrow} v$ *iff* $u \underline{\overset{\mathcal{L}}{\rightarrow}} v$ *(for* $\mathcal{L} \in \{\mathcal{FO}, \mathcal{FO}^-, \mathcal{ALC}, \mathcal{EL}, \mathcal{EL}^+\}$*).*

Proof. Some results are well-known: $\underline{\overset{\mathcal{FO}}{\rightarrow}}$ is isomorphism on labeled graphs [11]; $\underline{\overset{\mathcal{ALC}}{\rightarrow}}$ corresponds to the notion of bisimulation [3, Def. 2.16]; $\underline{\overset{\mathcal{EL}}{\rightarrow}}$ is a simulation as defined in [3, Def. 2.77]. The remaining cases are simple variations of these.

Table 2. $\mathcal{L}$-simulations for several logical languages $\mathcal{L}$

$\mathcal{L}$	ATOM$_L$	ATOM$_R$	REL$_L$	REL$_R$	INJ$_L$	INJ$_R$
$\mathcal{FO}$	×	×	×	×	×	×
$\mathcal{FO}^-$	×		×		×	
$\mathcal{ALC}$	×	×	×	×		
$\mathcal{EL}$	×		×			
$\mathcal{EL}^+$	×	×	×			

Therefore, on finite models[5] simulations capture exactly the notion of similarity. The right to left implication does not hold in general on infinite models.

$\mathcal{L}$-simulations allow us to determine, in an effective way, when an object is indistinguishable from another in a given model with respect to $\mathcal{L}$.

For example, we can verify that $a \underline{\overset{\mathcal{EL}}{\rightarrow}} b$ in the model of Figure 1 (the relation $\sim = \{(a,a),(a,b)\}$ satisfies ATOM$_L$ and REL$_L$). Using Theorem 1 we conclude that there is no $\mathcal{EL}$-description for a, since for any $\mathcal{EL}$-formula γ, if $a \in \|\gamma\|$, then $b \in \|\gamma\|$. Observe that $b \underline{\overset{\mathcal{EL}}{\not\rightarrow}} a$, since (again applying Theorem 1), $b \in \|small(x)\|$ but $a \notin \|small(x)\|$. If one chooses a language richer than $\mathcal{EL}$, such as $\mathcal{EL}^+$, one may be able to describe a: take, for instance the $\mathcal{EL}^+$-formula $dog(x) \wedge \neg small(x)$.

As we will discuss in the next section, simulation gives us an efficient, computationally feasible approach to the $\mathcal{L}$-GRE problem. Algorithms to compute many kinds of $\mathcal{L}$-simulations are well known (see, [15,18,14,10]), and for many languages (e.g., $\mathcal{ALC}$, $\mathcal{ALC}$ with inverse relations, $\mathcal{EL}^+$ and $\mathcal{EL}$) they run in polynomial time (on the other hand, no polynomial algorithm for $\mathcal{FO}$- or $\mathcal{FO}^-$-simulation is known and even the exact complexity of the problem in these cases is open [13]).

3 GRE via Simulator Sets

In this section we will discuss how to solve the $\mathcal{L}$-GRE problem using simulation. Given a model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$, Theorem 1 tells us that if two distinct elements u and v in Δ are such that $u \underline{\overset{\mathcal{L}}{\rightarrow}} v$ then every $\mathcal{L}$-formula that is true at u is also true at v. Hence there is no formula in $\mathcal{L}$ that can uniquely refer to u. From this perspective, knowing whether the model contains an element that is $\mathcal{L}$-similar but distinct from u is equivalent to decide whether there exists an $\mathcal{L}$-RE for u.

Assume a fixed language $\mathcal{L}$ and a model $\mathcal{M}$. Suppose we want to refer to an element u in the domain of $\mathcal{M}$. We would like to compute the *simulator set* of u defined as $sim_{\mathcal{L}}^{\mathcal{M}}(u) = \{v \in \Delta \mid u \underline{\overset{\mathcal{L}}{\rightarrow}} v\}$. When the model $\mathcal{M}$ is clear from the context, we just write $sim_{\mathcal{L}}$. If $sim_{\mathcal{L}}^{\mathcal{M}}(u)$ is not the singleton $\{u\}$, the $\mathcal{L}$-GRE problem with target $\{u\}$ in $\mathcal{M}$ will fail.

An algorithm is given in [14] to compute $sim_{\mathcal{EL}^+}(v)$ for each element v of a given finite model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ in time $O(\#\Delta \times \#\|\cdot\|)$. Intuitively, this algorithm defines $S(v)$ as a set of candidates for simulating v and successively refines it by removing those which fail to simulate v. In the end, $S(v) = sim_{\mathcal{EL}^+}(v)$.

[5] Finiteness is not the weakest hypothesis, but it is enough for our development.

The algorithm can be adapted to compute $sim_{\mathcal{L}}$ for many other languages $\mathcal{L}$. In particular, we can use it to compute $sim_{\mathcal{EL}}$ in polynomial time which will give us the basic algorithm for establishing an upper bound to the complexity of the $\mathcal{EL}$-GRE problem –this will answer an open question of [1]. The pseudo-code is shown in Algorithm 1, which uses the following notation: $\mathcal{P}$ is a fixed set of unary relation symbols, for $v \in \Delta$, let $P(v) = \{p \in \mathcal{P} \mid v \in \|p\|\}$ and let also $suc_r(v) = \{u \in \Delta \mid (v,u) \in \|r\|\}$ for r a binary relation symbol.

Algorithm 1. Computing $\mathcal{EL}$-similarity

input : a finite model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$
output: $\forall v \in \Delta$, the simulator set $sim^{\mathcal{M}}_{\mathcal{EL}}(v) = S(v)$

foreach $v \in \Delta$ **do**
  $S(v) := \{u \in \Delta \mid P(v) \subseteq P(u)\}$
while $\exists r, u, v, w : v \in suc_r(u), w \in S(u), suc_r(w) \cap S(v) = \emptyset$ **do**
  $S(u) := S(u) \setminus \{w\}$

The algorithm is fairly straightforward. We initialize $S(v)$ with the set of all elements $u \in \Delta$ such that $P(v) \subseteq P(u)$, i.e., the set of all elements satisfying at least the same unary relations as v (this guarantees that property ATOM_L holds). At each step, if there are three elements u, v and w such that for some relation r, $(u,v) \in \|r\|$, $w \in S(u)$ (i.e., w is a candidate to simulate u) but $suc_r(w) \cap S(v) = \emptyset$ (there is no element w' such that $(w,w') \in \|r\|$ and $w' \in S(v)$) then clearly condition REL_L is not satisfied under the supposition that $sim_{\mathcal{EL}} = S$. S is 'too big' because w cannot simulate u. Hence w is removed from $S(u)$.

Algorithm 1 will only tell us whether an $\mathcal{EL}$-RE for an element u exists (that is, whether $sim_{\mathcal{EL}}(u) = \{u\}$ or not). It does not compute an $\mathcal{EL}$-formula φ that uniquely refers to v. But we can adapt it to obtain such a formula. Algorithm 1's main strategy to compute simulations is to successively refine an over-approximation of the simulator sets. The "reason" behind each refinement can be encoded using an $\mathcal{EL}$-formula. Using this insight, one can transform an algorithm that computes $\mathcal{L}$-simulator sets with a similar strategy, into one that additionally computes an $\mathcal{L}$-RE for each set.

Algorithm 2 shows a transformed version of Algorithm 1 following this principle. The idea is that each node $v \in \Delta$ is now tagged with a formula $F(v)$ of $\mathcal{EL}$. The formulas $F(v)$ are updated along the execution of the loop, whose invariant ensures that $v \in \|F(v)\|$ and $\|F(u)\| \subseteq S(u)$ hold for all $u, v \in \Delta$.

Initially $F(v)$ is the conjunction of all the unary relations that satisfy v (if there is none, then $F(v) = \top$). Each time the algorithm finds elements r, u, v, w such that $(u,v) \in \|r\|$, $w \in S(u)$ and $suc_r(w) \cap S(v) = \emptyset$, it updates $F(u)$ to $F(u) \wedge \exists r.F(v)$. Again this new formula φ is in $\mathcal{EL}$ and it can be shown that $v \in \|\varphi\|$ and $w \notin \|\varphi\|$, hence witnessing that $v \overset{\mathcal{EL}}{\rightsquigarrow} w$ is false.

Algorithm 2 can be easily modified to calculate the $\mathcal{EL}^+$-RE of each simulator set $sim_{\mathcal{EL}^+}$ by adjusting the initialization: replace $\subseteq$ by $=$ in the initialization of $S(v)$ and initialize $F(v)$ as $\bigwedge \big(P(v) \cup \overline{P}(v)\big)$, where $\overline{P}(v) = \{\neg p \mid v \notin \|p\|\}$.

Algorithm 2. Computing $\mathcal{EL}$-similarity and $\mathcal{EL}$-RE

input : a finite model $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$
output: $\forall v \in \Delta$, a formula $F(v) \in \mathcal{EL}$, and the simulator set $S(v)$ such that
$\|F(v)\| = S(v) = sim_{\mathcal{EL}}(v)$

foreach $v \in \Delta$ **do**
  $S(v) := \{u \in \Delta \mid P(v) \subseteq P(u)\}$;
  $F(v) := \bigwedge P(v)$;

while $\exists r, u, v, w : v \in suc_r(u), w \in S(u), suc_r(w) \cap S(v) = \emptyset$ **do**
  invariant $\forall u, v : \|F(u)\| \subseteq S(u) \wedge v \in \|F(v)\|$
  $S(u) := S(u) \setminus \{w\}$;
  if $\exists r.F(v)$ *is not a conjunct of* $F(u)$ **then**
    $F(u) := F(u) \wedge \exists r.F(v)$;

With a naive implementation Algorithm 2 executes in time $O(\#\Delta^3 \times \#\|\cdot\|^2)$ providing a polynomial solution to the $\mathcal{EL}$ and $\mathcal{EL}^+$-GRE problems. Algorithm 1 can be transformed to run with a lower complexity as in shown in [14]; moreover this version of the algorithm can be adapted to compute $\mathcal{EL}$- and $\mathcal{EL}^+$-RE for an arbitrary subset of the domain of $\langle \Delta, \|\cdot\| \rangle$ in $O(\#\Delta \times \#\|\cdot\|)$ steps. We shall skip the details.

Theorem 2. *The $\mathcal{EL}$ and $\mathcal{EL}^+$-GRE problems over $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ have complexity $O(\#\Delta \times \#\|\cdot\|)$.*

Theorem 2 answers a question left open in [1]: the $\mathcal{EL}$-GRE problem can be solved in polynomial time. Note, however, that this result assumes a convenient representation of formulas like, for example, directed acyclic graphs, to ensure that each step of the formula construction can be done in $O(1)$. In §6 we will take a closer look at the issue and its relation to the size of the smallest $\mathcal{L}$-RE.

Algorithm 2 was obtained by adding formula annotations to a standard '$\mathcal{EL}$-simulation-minimization' algorithm. Given an $\mathcal{L}$-simulation-minimization, we can typically adapt it in an analogous way to obtain an $\mathcal{L}$-GRE algorithm. The obtained algorithm computes $\mathcal{L}$-REs for *every* element of the domain simultaneously. This will make it particularly suitable for applications with static domains requiring references to many objects. Moreover, the algorithm can be adapted to dynamic domains by using techniques used to recompute simulations (see [19]), so that only those RE that were affected by a change in the domain need to be recomputed.

We have not addressed so far other relevant issues of the GRE problem besides computational complexity. In particular, Algorithm 2 pays no attention to the use of preferences among relations when generating an RE (i.e., preferring the selection of certain attributes over others, when possible). While there is room for improvement (e.g., making a weighted choice instead of the non-deterministic choice when choosing elements in the main loop of the algorithm), support for preferences is not one of the strong points of this family of algorithms. We consider algorithms with strong support for preferences in the following section.

4 GRE via Building Simulated Models

Krahmer et al. [17] introduce an algorithm for content determination based on the computation of *subgraph isomorphisms*. It is heavily regulated by cost functions and is therefore apt to implement different preferences. In fact, they show that using suitable cost functions it can simulate most of the previous proposals. Their algorithm takes as input a labeled directed graph G and a node e and returns, if possible, a connected subgraph H of G, containing e and enough edges to distinguish e from the other nodes.

In this section we will identify its underlying notion of expressiveness and will extend it to accommodate other notions. To keep the terminology of [17], in what follows we may alternatively talk of *labeled graphs* instead of relational models. The reader should observe that they are essentially the same mathematical object, but notice that in [17], propositions are encoded using looping binary relations (e.g., they write $dog(e, e)$ instead of $dog(e)$).

The main ideas of their algorithm can be intuitively summarized as follows. Given two labeled graphs H and G, and vertices v of H and w of G, we say that the pair (v, H) *refers* to the pair (w, G) iff H is connected and H can be "placed over" G in such a way that: 1) v is placed over w; 2) each node of H is placed over a node of G with at least the same unary predicates (but perhaps more); and 3) each edge from H is placed over an edge with the same label. Furthermore, (v, H) *uniquely refers* to (w, G) if (v, H) refers to (w, G) and there is no vertex $w' \neq w$ in G such that (v, H) refers to (w', G). The formal notion of a labeled graph being "placed over" another one is that of *subgraph isomorphism*: $H = \langle \Delta_H, \|\cdot\|_H \rangle$ can be placed over G iff there is a labeled subgraph (i.e., a graph obtained from G by possibly deleting certain nodes, edges, and propositions from some nodes) $G' = \langle \Delta_{G'}, \|\cdot\|_{G'} \rangle$ of G such that H is *isomorphic* to G', which means that there is a bijection $f : \Delta_H \to \Delta_{G'}$ such that for all vertices $u, v \in \Delta_H$, $u \in \|p\|_H$ iff $f(u) \in \|p\|_{G'}$ and $(u, v) \in \|r\|_H$ iff $(f(u), f(v)) \in \|r\|_{G'}$.

As an example, consider the relational model depicted in Figure 1 as a labeled graph G, and let us discuss the pairs of nodes and connected subgraphs (v, H) shown in Figure 2. Clearly, (i) refers to the pair (w, G) for any node $w \in \{a, b, d\}$; (ii) refers to (w, G) for $w \in \{b, d\}$; and both (iii) and (iv) uniquely refer to (b, G). Notice that (i)–(iv) can be respectively realized as "*a dog*", "*a dog that sniffs something*", "*a small dog that sniffs a dog*" (cf. γ_1 in Table 1) and "*the dog that is sniffed by a small cat*" (cf. γ_4 in Table 1).

It is important to emphasize that there is a substantial difference between the algorithm presented in [17] and the one we discussed in the previous sections:

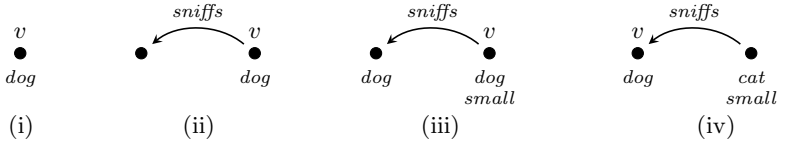


Fig. 2. Some connected subgraphs (v, H) of scene $\mathcal{S}$ in Figure 1

while the input is a labeled graph G and a target node v, the output is, in this case (and unlike the definition of $\mathcal{L}$-GRE problem presented in §2 where the output is a *formula*), the cheapest (with respect to some, previously specified cost function) connected *subgraph* H of G which uniquely refers to (v, G) if there is such H, and $\bot$ otherwise.

We will not deal with cost functions here; it is enough to know that a cost function is a monotonic function that assigns to each subgraph of a scene graph a non-negative number which expresses the goodness of a subgraph –e.g. in Figure 2, one may tune the cost function so that (iii) is cheaper than (iv), and hence (iii) will be preferred over (iv).

For reasons of space we will not introduce here the detailed algorithm proposed in [17]. Roughly, it is a straightforward branch and bound algorithm that systematically tries all relevant subgraphs H of G by starting with the subgraph containing only vertex v and expanding it recursively by trying to add edges from G that are adjacent to the subgraph H constructed up to that point. In the terminology of [17] a *distractor* is a node of G different from v that is also referred by H. The algorithm ensures that a subgraph uniquely refers to the target v when it has no distractors. Reached this point we have a new candidate for the solution, but there can be other cheaper solution so the search process continues until the cheapest solution is detected. Cost functions are used to guide the search process and to give preference to some solutions over others.

Here is the key link between the graph-based method of [17] and our logical-oriented perspective: on finite relational models, subgraph isomorphism corresponds to $\mathcal{FO}^-$-simulations, in the sense that given two nodes u, v of G, there is a subgraph isomorphic to G via f, containing u and v, and such that $f(u) = v$ iff $u \underline{\overset{\mathcal{FO}^-}{\rightarrow}} v$. Having made explicit this notion of sameness and, with it, the logical language associated to it, we can proceed to generalize the algorithm to make it work for other languages, and to adapt it in order to output a formula instead of a graph. This is shown in Algorithms 3 and 4.

Algorithm 3. $\texttt{makeRE}_{\mathcal{L}}(v)$

input : an implicit finite $G = \langle \Delta_G, \|\cdot\| \rangle$ and $v \in \Delta_G$
output: an $\mathcal{L}$-RE for v in G if there is one, or else $\bot$

$H := \langle \{v\}, \emptyset \rangle$;
$f := \{v \mapsto v\}$;
$H' := \texttt{find}_{\mathcal{L}}\ (v, \bot, H, f)$;
return $\texttt{buildF}_{\mathcal{L}}\ (H', v)$;

Algorithm 4. $\texttt{find}_{\mathcal{L}}(v, best, H, f)$

if $best \neq \bot \wedge \texttt{cost}(best) \leq \texttt{cost}(H)$ **then**
  return $best$
$distractors := \{n \mid n \in \Delta_G, n \neq v, v \underline{\overset{\mathcal{L}}{\rightarrow}} n\}$;
if $distractors = \emptyset$ **then**
  return H
foreach $\langle H', f' \rangle \in \texttt{extend}_{\mathcal{L}}(H, f)$ **do**
  $I := \texttt{find}_{\mathcal{L}}\ (v, best, H', f')$;
  if $best = \bot \vee \texttt{cost}(I) \leq \texttt{cost}(best)$ **then**
    $best := I$
return $best$;

These algorithms are parametric on $\mathcal{L}$; to make them concrete, one needs to provide appropriate versions of $\texttt{buildF}_{\mathcal{L}}$ and $\texttt{extend}_{\mathcal{L}}$. The former transforms the computed *graph* which uniquely refers to the target v into an $\mathcal{L}$-RE *formula* for v; the latter tells us how to extend H at each step of the main loop of

Algorithm 4. Note that, unlike the presentation of [17], $\texttt{makeRE}_{\mathcal{L}}$ computes not only a graph H but also an $\mathcal{L}$-simulation f. In order to make the discussion of the differences with the original algorithm simpler, we analyze next the case $\mathcal{L} = \mathcal{FO}^-$ and $\mathcal{L} = \mathcal{EL}$.

The case of $\mathcal{FO}^-$. From the computed cheapest isomorphic subgraph H' one can easily build an $\mathcal{FO}^-$-formula that uniquely describes the target v, as is shown in Algorithm 5. Observe that if $\mathcal{FO}$-simulations were used instead, we would have to include also which unary and binary relations *do not hold* in H'.

Algorithm 5. $\texttt{buildF}_{\mathcal{FO}^-}(H', v)$

let $H' = \langle\{a_1 \dots a_n\}, \|\cdot\|\rangle, v = a_1$;

$$\gamma := \bigwedge_{a_i \neq a_j} (x_i \not\approx x_j) \wedge \bigwedge_{(a_i,a_j) \in \|r\|} r(x_i, x_j) \wedge \bigwedge_{a_i \in \|p\|} p(x_i)$$

return $\exists x_2 \dots \exists x_n.\gamma$;

Algorithm 6. $\texttt{extend}_{\mathcal{FO}^-}(H, f)$

$A := \{H{+}p(u) \mid u \in \Delta_H,$
$\quad u \in \|p\|_G \setminus \|p\|_H\}$;
$B := \{H{+}r(u,v) \mid u \in \Delta_H,$
$\{(u,v),(v,u)\} \cap \|r\|_G \setminus \|r\|_H \neq \emptyset\}$;
return $(A \cup B) \times \{id\}$;

Regarding the function which extends the given graph in all possible ways (Algorithm 6), since H is a subgraph of G, f is the trivial identity function $id(x) = x$. We will see the need for f when discussing the case of less expressive logics like $\mathcal{EL}$. In $\texttt{extend}_{\mathcal{FO}^-}$ we follow the notation of [17] and write, for a relational model $G = \langle\Delta, \|\cdot\|\rangle$, $G + p(u)$ to denote the model $\langle\Delta \cup \{u\}, \|\cdot\|'\rangle$ such that $\|p\|' = \|p\| \cup \{u\}$ and $\|q\|' = \|q\|$ when $q \neq p$. Similarly, $G + r(u,v)$ denotes the model $\langle\Delta \cup \{u,v\}, \|\cdot\|'\rangle$ such that $\|r\|' = \|r\| \cup \{(u,v)\}$ and $\|q\|' = \|q\|$ when $q \neq r$. It is clear, then, that this function is returning all the *extensions* of H by adding a missing attribute or relation to H, just like is done in the original algorithm.

The case of $\mathcal{EL}$. Observe that $\texttt{find}_{\mathcal{EL}}$ uses an $\mathcal{EL}$-simulation, and any $\mathcal{FO}^-$-simulation is an $\mathcal{EL}$-simulation. One could, in principle, just use $\texttt{extend}_{\mathcal{FO}^-}$ also for $\mathcal{EL}$. If we do this, the result of $\texttt{find}_{\mathcal{EL}}$ will be a subgraph H of G such that for every $\mathcal{EL}$-simulation $\sim$, $u \sim v$ iff $u = v$. The problem is that this subgraph H may contain cycles and, as it is well known, $\mathcal{EL}$ (even $\mathcal{ALC}$) are incapable to distinguish a cycle from its *unraveling*[6]. Hence, although subgraph isomorphism get along with $\mathcal{FO}^-$, it is too strong to deal with $\mathcal{EL}$.

A well-known result establishes that every relational model $\mathcal{M}$ is equivalent, with respect to $\mathcal{EL}$-formulas,[7] to the unraveling of $\mathcal{M}$. That is, any model and its unraveling satisfy exactly the same $\mathcal{EL}$-formulas. Moreover, the unraveling of $\mathcal{M}$ is always a tree, and as we show in Algorithm 7, it is straightforward to extract a suitable $\mathcal{EL}$-formula from a tree.

Therefore, we need $\texttt{extend}_{\mathcal{EL}}$ to return all the possible "extensions" of H. Now "extension" does not mean to be a subgraph of the original graph G anymore.

[6] Informally, the unraveling of G, is a new graph, whose points are paths of G from a given starting node. That is, transition sequences in G are explicitly represented as nodes in the unraveled model. See [3] for a formal definition.

[7] Actually, the result holds even for $\mathcal{ALC}$-formulas.

We do this by either adding a new proposition or a new edge that is present in the unraveling of G but not in H. This is shown in Algorithm 8.

Algorithm 7. $\texttt{buildF}_{\mathcal{EL}}(H', v)$

requires H' to be a tree
$\gamma := \{\exists r.\texttt{buildF}_{\mathcal{EL}}(H', u) \mid (v, u) \in \|r\|\}$;
return $(\bigwedge \gamma) \wedge (\bigwedge_{v \in \|p\|} p)$;

Algorithm 8. $\texttt{extend}_{\mathcal{EL}}(H, f)$

$A :=$
  $\{\langle H + p(u), f\rangle \mid u \in \Delta_H, u \in \|p\|_G \setminus \|p\|_H\}$;
$B := \emptyset$;
foreach $u \in \Delta_G$ **do**
  foreach $u_H \in \Delta_H / (f(u_h), u) \in \|r\|_G$ **do**
    if $\forall v : (u_H, v) \in \|r\|_H \Rightarrow f(v) \neq u$ **then**
      $n :=$ *new node*;
      $B := B \cup$
        $\{\langle H + r(u_H, n), f \cup \{n \mapsto u\}\rangle\}$;
return $A \cup B$;

Observe that the behavior of $\texttt{find}_{\mathcal{EL}}$ is quite sensible to the `cost` function employed. For instance, on cyclic models, a `cost` function that does not guarantee the unraveling is explored in a breadth-first way may lead to non-termination (since $\texttt{find}_{\mathcal{EL}}$ may loop exploring an infinite branch).

As a final note on complexity, although the set of $\mathcal{EL}$-distractors may be computed more efficiently than $\mathcal{FO}^-$-distractors (since $\mathcal{EL}$-distractors can be computed in polynomial time, and computing $\mathcal{FO}^-$-distractors seems to require a solution to the subgraph isomorphism problem which NP-complete), we cannot conclude that $\texttt{find}_{\mathcal{EL}}$ is more efficient than $\texttt{find}_{\mathcal{FO}^-}$ in general: the model built in the first case may be exponentially larger –it is an unraveling, after all. We will come back to this in §6.

5 Combining GRE Methods

An appealing feature of formulating the GRE problem modulo expressivity is that one can devise general strategies that combine $\mathcal{L}$-GRE algorithms. We illustrate this with an example.

The algorithms based on $\mathcal{L}$-simulator sets like the ones in §3 simultaneously compute referring expressions for every object in the domain, and do this for many logics in polynomial time. This is an interesting property when one anticipates the need of referring to a large number of elements. However, this family of algorithms is not as flexible in terms of implementing preferences as those we introduced in §4 –though some flexibility can be obtained by using cost functions for selecting u, v and w in the main loop of Algorithm 2 instead of the non-deterministic choices.

There is a simple way to obtain an algorithm that is a compromise between these two techniques. Let A_1 and A_2 be two procedures that solve the $\mathcal{L}$-GRE problem based on the techniques of §3 and §4, respectively. One can first compute an $\mathcal{L}$-RE for every possible object using A_1 and then (lazily) replace the

calculated RE for u with $A_2(u)$ whenever the former does not conform to some predefined criterion. This is correct but we do better, taking advantage of the equivalence classes obtained using A_1.

Since A_1 computes, for a given $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$, the set $sim(u)$ for every $u \in \Delta$, one can build in polynomial time, using the output of A_1, the model $\mathcal{M}_{\mathcal{L}} = \langle \{[u] \mid u \in \Delta\}, \|\cdot\|_{\mathcal{L}} \rangle$, such that: $[u] = \{v \mid u \underline{\overset{\mathcal{L}}{\rightarrow}} v \text{ and } v \underline{\overset{\mathcal{L}}{\rightarrow}} u\}$ and $\|r\|_{\mathcal{L}} = \{([u_1] \dots [u_n]) \mid (u_1 \dots u_n) \in \|r\|\}$. $\mathcal{M}_{\mathcal{L}}$ is known as *the $\mathcal{L}$-minimization of $\mathcal{M}$*. By a straightforward induction on γ one can verify that $(u_1 \dots u_n) \in \|\gamma\|$ iff $([u_1] \dots [u_n]) \in \|\gamma\|_{\mathcal{L}}$ and this implies that γ is an $\mathcal{L}$-RE for u in $\mathcal{M}$ iff it is an $\mathcal{L}$-RE for $[u]$ in $\mathcal{M}_{\mathcal{L}}$.

If $\mathcal{M}$ has a large number of indistinguishable elements (using $\mathcal{L}$), then $\mathcal{M}_{\mathcal{L}}$ will be much smaller than $\mathcal{M}$. Since the computational complexity of A_2 depends on the size of $\mathcal{M}$, for very large scenes, one should compute $A_2([u])$ instead.

6 On the Size of Referring Expressions

The expressive power of a language $\mathcal{L}$ determines if there is an $\mathcal{L}$-RE for an element u. It also influences the *size* of the *shortest* $\mathcal{L}$-RE (when they exist). Intuitively, with more expressive power we are able to 'see' more differences and therefore have more resources at hand to build a shorter formula.

A natural question is, then, whether we can characterize the relative size of the $\mathcal{L}$-REs for a given $\mathcal{L}$. That is, if we can give (tight) upper bounds for the size of the shortest $\mathcal{L}$-REs for the elements of an arbitrary model $\mathcal{M}$, as a function of the size of $\mathcal{M}$.

For the case of one of the most expressive logics considered in this article, $\mathcal{FO}^-$, the answer follows from algorithm $\texttt{makeRE}_{\mathcal{FO}^-}$ in §4. Indeed, if an $\mathcal{FO}^-$-RE exists, it is computed by $\texttt{buildF}_{\mathcal{FO}^-}$ from a model H that is not bigger than the input model. It is easy to see that this formula is linear in the size of H and, therefore the size of any $\mathcal{FO}^-$-RE is $O(\#\Delta + \#\|\cdot\|)$. It is not hard to see that this upper bound holds for $\mathcal{FO}$-REs too.

One is tempted to conclude from Theorem 2 that the size of the shortest $\mathcal{EL}$-RE is $O(\#\Delta \times \#\|\cdot\|)$, but there is a pitfall. Theorem 2 assumes that formulas are represented as a DAG and it guarantees that this DAG is polynomial in the size of the input model. One can easily reconstruct (the syntax tree of) the formula from the DAG, but this, in principle, may lead to a exponential blow-up –the result will be an exponentially larger formula, but composed of only a polynomial number of different subformulas. As the following example shows, it is indeed possible to obtain an $\mathcal{EL}$-formula that is exponentially larger when expanding the DAG representation generated by Algorithm 2.

Example 1. Consider a language with only one binary relation r, and let $\mathcal{M} = \langle \Delta, \|\cdot\| \rangle$ where $\Delta = \{1, 2, \dots, n\}$ and $(i, j) \in \|r\|$ iff $i < j$. Algorithm 2 initializes $F(j) = \top$ for all $j \in \Delta$. Suppose the following choices in the execution: For $i = 1, \dots, n-1$, iterate $n - i$ times picking $v = w = n - i + 1$ and successively $u = n - i, \dots, 1$. It can be shown that each time a formula $F(j)$ is updated, it

changes from φ to $\varphi \wedge \exists r.\varphi$ and hence it doubles its size. Since $F(1)$ is updated $n-1$ many times, the size of $F(1)$ is greater than 2^n.

The large $\mathcal{EL}$-RE of Example 1 is due to an unfortunate (non-deterministic) choice of elements. Example 2 shows that another execution leads to a quadratic RE (but notice the shortest one is linear: $(\exists r)^{(n-1)}.\top$).

Example 2. Suppose now that in the first $n-1$ iterations we successively choose $v = w = n - i$ and $u = v - 1$ for $i = 0 \ldots n-2$. It can be seen that for further convenient choices, $F(1)$ is of size $O(n^2)$.

But is it always possible to obtain an $\mathcal{EL}$-RE of size polynomial in the size of the input model, when we represent a formula as a string, and not as a DAG? In [12] it is shown that the answer is 'no': for $\mathcal{L} \in \{\mathcal{ALC}, \mathcal{EL}, \mathcal{EL}^+\}$, the lower bound for the length of the $\mathcal{L}$-RE is exponential in the size of the input model[8], and this lower bound is tight.

7 Conclusions

The content determination phase during the generation of referring expressions identifies which 'properties' will be used to refer to a given target object or set of objects. What is considered as a 'property' is specified in different ways by each of the many algorithms for content determination existing in the literature. In this article, we put forward that this issue can be addressed by deciding when two elements should be considered to be equal, that is, by deciding which discriminatory power we want to use. Formally, the discriminatory power we want to use in a particular case can be specified syntactically by choosing a particular formal language, or semantically, by choosing a suitable notion of simulation. It is irrelevant whether we choose first the language (and obtain the associated notion of simulation afterwards) or vice versa.

We maintain that having both at hand is extremely useful. Obviously, the formal language will come handy as representation language for the output to the content determination problem. But perhaps more importantly, once we have fixed the expressivity we want to use, we can rely on model theoretical results defining the adequate notion of sameness underlying each language, which indicates what can and cannot be said (as we discussed in §2). Moreover, we can transfer general results from the well-developed fields of computational logics and graph theory as we discuss in §3 and §4, where we generalized known algorithms into *families* of GRE algorithms for different logical languages.

An explicit notion of expressiveness also provides a cleaner interface, either between the content determination and surface realization modules or between two collaborating content determination modules. An instance of the latter was exhibited in §5.

[8] More precisely, there are infinite models $G_1, G_2, \ldots$ such that for every i, the size of G_i is linear in i but the size of the minimum RE for some element in G_i is bounded from below by a function which is exponential on i.

As a future line of research, one may want to avoid sticking to a fixed $\mathcal{L}$ but instead favor an incremental approach in which features of a more expressive language $\mathcal{L}_1$ are used only when $\mathcal{L}_0$ is not enough to distinguish certain element.

References

1. Areces, C., Koller, A., Striegnitz, K.: Referring expressions as formulas of description logic. In: Proc. of the 5th INLG, Salt Fork, OH, USA (2008)
2. Baader, F., McGuiness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, implementation and applications. Cambridge University Press, Cambridge (2003)
3. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)
4. Dale, R.: Cooking up referring expressions. In: Proc. of the 27th ACL (1989)
5. Dale, R., Haddock, N.: Generating referring expressions involving relations. In: Proc. of the 5th EACL (1991)
6. Dale, R., Reiter, E.: Computational interpretations of the Gricean maxims in the generation of referring expressions. Cognitive Science 19 (1995)
7. Dale, R., Viethen, J.: Referring expression generation through attribute-based heuristics. In: Proc. of the 12th ENLG Workshop, pp. 58–65 (2009)
8. van Deemter, K.: Generating referring expressions: Boolean extensions of the incremental algorithm. Computational Linguistics 28(1), 37–52 (2002)
9. van Deemter, K., van der Sluis, I., Gatt, A.: Building a semantically transparent corpus for the generation of referring expressions. In: Proc. of the 4th INLG (2006)
10. Dovier, A., Piazza, C., Policriti, A.: An efficient algorithm for computing bisimulation equivalence. Theor. Comput. Sci. 311, 221–256 (2004)
11. Ebbinghaus, H., Flum, J., Thomas, W.: Mathematical Logic. Springer, Heidelberg (1996)
12. Figueira, S., Gorín, D.: On the size of shortest modal descriptions. Advances in Modal Logic 8, 114–132 (2010)
13. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. Freeman, New York (1979)
14. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proc. of 36th Annual Symposium on Foundations of Computer Science, pp. 453–462. IEEE Computer Society Press, Los Alamitos (1995)
15. Hopcroft, J.: An $nlog(n)$ algorithm for minimizing states in a finite automaton. In: Kohave, Z. (ed.) Theory of Machines and Computations. Academic Press, London (1971)
16. Horacek, H.: An algorithm for generating referential descriptions with flexible interfaces. In: Proc. of the 35th ACL, pp. 206–213 (1997)
17. Krahmer, E., van Erk, S., Verleg, A.: Graph-based generation of referring expressions. Computational Linguistics 29(1) (2003)
18. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM J. Comput. 16(6), 973–989 (1987)
19. Saha, D.: An incremental bisimulation algorithm. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 204–215. Springer, Heidelberg (2007)
20. Stone, M.: On identifying sets. In: Proc. of the 1st INLG (2000)
21. Stone, M., Webber, B.: Textual economy through close coupling of syntax and semantics. In: Proc. of the 9th INLG Workshop, pp. 178–187 (1998)
22. Viethen, J., Dale, R.: Algorithms for generating referring expressions: Do they do what people do? In: Proc. of the 4th INLG (2006)

Polarized Classical Non-associative Lambek Calculus and Formal Semantics

Arno Bastenhof

Utrecht University

Abstract. While initially motivated for studying natural language syntax, the intuitionistic bias underlying traditional Lambek calculi renders them particularly suitable to a Montagovian formal semantics through the Curry-Howard correspondence. Several recent proposals, however, have departed from the intuitionistic tradition, seeking instead to formulate 'classical' Lambek calculi. We show that this classical turn need not come at the cost of the tight connection with formal semantics, concentrating on De Groote and Lamarche's Classical Non-Associative Lambek calculus (**CNL**). Our work is founded in Girard's and Andreoli's research into polarities and focused proofs, suggesting the definition of *polarized* **CNL**, its connection to De Groote and Lamarche's original proposal explicated through the use of phase spaces. We conclude with a discussion of related literature, particularly Moortgat's Lambek-Grishin calculus.

1 Introduction

Categorial grammars in the Lambek tradition seek a proof-theoretic explanation of natural language syntax: syntactic categories are formulas and derivations are proofs. Typically, one observes an intuitionistic bias towards single conclusion sequents, the existence of a Montagovian formal semantics thereby being immediate through the Curry-Howard correspondence. Recent years, however, have seen proposals for 'classical Lambek calculi', retaining the resource sensitivity of the traditional systems, while dispensing with their intuitionistic asymmetry in favor of an involutive negation. This article proposes a formal semantics for such systems, concentrating on Classical Non-Associative Lambek calculus (**CNL**, [7]), arguably the most resource sensitive among its kin.

We proceed as follows. §2 briefly recapitulates the basics of **CNL**. In §3, we define *polarized* **CNL** ($\mathbf{CNL}^{pol}$), drawing from Girard's research into polarities and Andreoli's work on focusing ([5], [1]). Roughly, a dichotomy between positive and negative formulas is uncovered, serving as a guide to Cut-free proof search in avoiding 'don't care' non-determinism. A Montagovian formal semantics is defined for $\mathbf{CNL}^{pol}$, and briefly motivated through simple analyses of (subject) relativization and quantifier scope ambiguities. We ensure, in §4, that provability in **CNL** and $\mathbf{CNL}^{pol}$ coincide, using an argument involving phase spaces. §5 concludes with a discussion of several related topics.[1]

[1] Throughout this article, in referring to a previously stated definition (lemma, theorem, corollary, figure) n, we often use the abbreviation D.n (L.n, T.n, C.n, F.n).

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 33–48, 2011.

$$\frac{}{A, A^{\perp} \vdash} \; Ax \qquad \frac{\Gamma, \Delta \vdash}{\Delta, \Gamma \vdash} \; dp^1 \qquad \frac{\Gamma \bullet \Delta, \Theta \vdash}{\Gamma, \Delta \bullet \Theta \vdash} \; dp^2$$

$$\frac{\Delta, A \vdash \quad \Gamma, A^{\perp} \vdash}{\Gamma, \Delta \vdash} \; Cut \qquad \frac{\Gamma, A \bullet B \vdash}{\Gamma, A \otimes B \vdash} \otimes \qquad \frac{\Gamma, A \vdash \quad \Delta, B \vdash}{\Delta \bullet \Gamma, A \oplus B \vdash} \oplus$$

Fig. 1. CNL defined. Double inference lines indicate applicability in both directions.

2 Classical Non-associative Lambek Calculus

We briefly recapitulate De Groote and Lamarche's sequent calculus for **CNL**, permitting ourselves a few deviations in notation as motivated below.

Definition 1. Formulas in **CNL** are generated from positive and negative atoms $p, \bar{p}$ using the multiplicative disjunction $\oplus$ (*par*) and -conjunction $\otimes$ (*tensor*):

$$A, B ::= p \mid \bar{p} \mid (A \otimes B) \mid (A \oplus B)$$

Classicality is affirmed through *linear negation* $\cdot^{\perp}$:

$$p^{\perp} = \bar{p} \qquad \bar{p}^{\perp} = p$$
$$(A \otimes B)^{\perp} = B^{\perp} \oplus A^{\perp} \qquad (A \oplus B)^{\perp} = B^{\perp} \otimes A^{\perp}$$

Definition 2. *Structures* Γ, Δ combine formulas into binary-branching trees:

$$\Gamma, \Delta ::= A \mid (\Gamma \bullet \Delta)$$

Definition 3. Figure 1 defines the derivability judgement $\Gamma, \Delta \vdash$, pairing structures Γ, Δ into a *left-sided sequent*. The rules (dp^2) and (dp^1) remind of both Yetter's cyclicity ([17]), as well als Belnap's display postulates ([2]).

While De Groote and Lamarche employ the more common right-sided sequents, the left-sided notation facilitates a more transparent correspondence with the intuitionistic sequents used in Montagovian semantics, as shown in §3.2.

3 Polarized CNL

The current section develops a variation on **CNL** permitting a correspondence between Cut-free derivations and linear λ-terms in long β-normal form. Linguistic applications focus on the relation to a Montagovian-style formal semantics.

3.1 Motivation

Our revision of **CNL** is founded in the research on polarities in logic, initiated in the early nineties through the independent works of Andreoli ([1]) and Girard ([5]). The former's efforts concerned the elimination of inessential non-determinism in naïve backward chaining proof search, while Girard instead

sought a constructive reading of classical proofs, bypassing Lafont's critical pairs. Both ultimately settled on modified sequent calculi carrying explicit partitionings of formulas correlated with proof-theortic behavior. Downplayed to a strictly multiplicative setting, formulas are of either positive or negative *polarity*, depending on whether or not their inferences are always invertible, preserving the provability of their conclusion inside their premises. Top-down proof-search then prioritizes the application of invertible inferences, while enforcing maximal chains of non-invertible inferences applied to subformulas of the same initial formula.

Girard's constructivization of classical logic inspired a novel intuitionistic translation, developed further in [16], [18]. Roughly, the introduction of double negations is made contingent upon the polarity of the formula being translated, thus achieving parsimony. We adapt Girard's translation to **CNL** by developing a polarized variant thereof, explicitly distinguishing between positive and negative formulas. Through a Curry-Howard mapping, we obtain a compositional semantics along the lines of [11], illustrated by a few simple case analyses.

3.2 Polarization and Derivational Semantics

Definition 4. Formulas in *polarized* **CNL** (henceforth $\mathbf{CNL}^{pol}$) are inherently positive or negative, with *shifts* $\uparrow, \downarrow$ ([6], §5.3) establishing communication:

$$\begin{array}{c} P, Q ::= p \mid (P \otimes Q) \mid (\downarrow N) \\ M, N ::= \bar{p} \mid (M \oplus N) \mid (\uparrow P) \end{array}$$

Linear negation $\cdot^{\perp}$ is revised accordingly, satisfying $P^{\perp\perp} = P$ and $N^{\perp\perp} = N$:

$$\begin{array}{cc} p^{\perp} = \bar{p} & \bar{p}^{\perp} = p \\ (P \otimes Q)^{\perp} = Q^{\perp} \oplus P^{\perp} & (M \oplus N)^{\perp} = N^{\perp} \otimes M^{\perp} \\ (\downarrow N)^{\perp} = \uparrow N^{\perp} & (\uparrow P)^{\perp} = \downarrow P^{\perp} \end{array}$$

Remark 5. In practice, we assume $\uparrow, \downarrow$ to bind more strongly than $\otimes, \oplus$, and drop brackets accordingly. Thus, $P \otimes \uparrow Q^{\perp}$ abbreviates $(P \otimes (\uparrow Q^{\perp}))$.

The linguistic explanation of Lambek calculi emphasizes a reading of formulas as syntactic categories. Their inductive structure facilitates a tight correspondence with *semantic types*, the formulas of an intuitionistic calculus used to describe the range of possible denotations for linguistic expressions.

Definition 6. For the purpose of describing the semantic operations associated with derivability in $\mathbf{CNL}^{pol}$, we will use the following notion of (semantic) type:

$$\tau, \sigma ::= p \mid \perp \mid (\tau \otimes \sigma) \mid \neg\tau$$

We understand all positive atoms p of $\mathbf{CNL}^{pol}$ to have been inherited, and to have been augmented by a distinguished atom $\perp$. Compound types include multiplicative products $(\tau \otimes \sigma)$ (overloading notation) and negations $\neg\tau$. Intuitively, the latter may be understood by linear negations $\tau \multimap \perp$ with $\perp$ as result.

Definition 7. Formulas P, N are mapped into types $\sigma^{+}(P)$, $\sigma^{-}(N)$, as follows:

$$\frac{}{\tau^x \vdash x : \tau}\ Ax$$

$$\frac{\Gamma \vdash s : \neg\tau \quad \Delta \vdash t : \tau}{\Gamma, \Delta \vdash (s\ t) : \bot}\ \neg E \qquad \frac{\Gamma, \tau^x \vdash s : \bot}{\Gamma \vdash \lambda x^\tau s : \neg\tau}\ \neg I$$

$$\frac{\Delta \vdash t : \sigma_1 \otimes \sigma_2 \quad \Gamma, \sigma_1^x, \sigma_2^y \vdash s : \tau}{\Gamma, \Delta \vdash (\mathbf{case}\ t\ \mathbf{of}\ \langle x^{\sigma_1}, y^{\sigma_2}\rangle s) : \tau}\ \otimes E \qquad \frac{\Gamma \vdash s : \tau \quad \Delta \vdash t : \sigma}{\Gamma, \Delta \vdash \langle s, t\rangle : \tau \otimes \sigma}\ \otimes I$$

$$\begin{aligned}
(\lambda x s\ t) &\to_\beta s[t/x] \\
\mathbf{case}\ \langle t_1, t_2\rangle\ \mathbf{of}\ \langle x, y\rangle s &\to_\beta s[t_1/x, t_2/y] \\
(\mathbf{case}\ t\ \mathbf{of}\ \langle x, y\rangle s\ s') &\to_c \mathbf{case}\ t\ \mathbf{of}\ \langle x, y\rangle (s\ s') \\
\mathbf{case}\ (\mathbf{case}\ t\ \mathbf{of}\ \langle x, y\rangle s)\ \mathbf{of}\ \langle u, v\rangle s' &\to_c \mathbf{case}\ t\ \mathbf{of}\ \langle x, y\rangle (\mathbf{case}\ s\ \mathbf{of}\ \langle u, v\rangle s')
\end{aligned}$$

Fig. 2. Target language: typing rules and reductions. The c-conversions correspond to the commutative conversions of Prawitz ([14]), while the 'case' notation follows that of Wadler ([15]).

$$\begin{aligned}
\sigma^+(p) &= p & \sigma^-(\bar{p}) &= p \\
\sigma^+(P \otimes Q) &= \sigma^+(P) \otimes \sigma^+(Q) & \sigma^-(M \oplus N) &= \sigma^-(N) \otimes \sigma^-(M) \\
\sigma^+(\downarrow N) &= \neg\sigma^-(N) & \sigma^-(\uparrow P) &= \neg\sigma^+(P)
\end{aligned}$$

If the analogy of a semantic type as capturing a possible 'kind' of denotation is to be further pursued, we require a means of describing its inhabitants.

Definition 8. F.2 defines a calculus of *linear λ-terms*, named $\mathbf{LP}_{\otimes,\neg}$ (or simply **LP**) in reference to the permutative Lambek calculus. Subject of derivability are sequents $\Gamma \vdash s : \tau$, establishing inhabitation of s in τ relative to a *context* Γ (again overloading notation): a multiset $\{\tau_1^{x_1}, \ldots, \tau_n^{x_n}\}$ of type assignments $\tau_1, \ldots, \tau_n$ to the free variables $x_1, \ldots, x_n$ in M. The braces $\{,\}$ are often ommitted and we loosely write Γ, Δ for multiset union. Finally, sequents are to satisfy the linearity constraint that each variable in Γ is to occur free in s exactly once.

In practice, we sometimes abbreviate $\lambda z(\mathbf{case}\ z\ \mathbf{of}\ \langle x, y\rangle s)$ by paired abstraction $\lambda\langle x, y\rangle s$, not to be taken as a primitive constructor.

Definition 9. The structures Π, Σ of $\mathbf{CNL}^{pol}$ are built using solely positive formulas, annotated by **LP**'s variables.

$$\Pi, \Sigma ::= P^x \mid (\Pi \bullet \Sigma)$$

Structures Π map into **LP** contexts $\sigma(\Pi)$ by interpreting P^x as a type assignment $\{\sigma^+(P)^x\}$ and collapsing the sole structural connective $\cdot \bullet \cdot$ into set union.

Definition 10. Figure 3 defines derivability judgements $\Pi, \Sigma \vdash s$ and $\Pi \vdash s : N$ for $\mathbf{CNL}^{pol}$, defined by mutual induction. Both carry an **LP** term s, while N is said to inhabit the *stoup* in the latter case (adopting terminology of [5]). In practice, we will often abbreviate sequences of (dp^2) and (dp^1) by (dp).

Sequents $\Pi, \Sigma \vdash s$ may be understood to implicitly carry $\bot$ as the type of s. More specifically, a straightforward induction proves

$$\frac{}{p^x \vdash x : \bar{p}}\ Ax \qquad \frac{\Pi, \Sigma \vdash s}{\Sigma, \Pi \vdash s}\ dp^1 \qquad \frac{\Pi \bullet \Sigma, \Upsilon \vdash s}{\Pi, \Sigma \bullet \Upsilon \vdash s}\ dp^2$$

$$\frac{\Pi \vdash s : N}{\Pi, \downarrow N^x \vdash (x\ s)}\ \downarrow \qquad \frac{\Pi, P^y \bullet Q^z \vdash s}{\Pi, P \otimes Q^x \vdash \mathbf{case}\ x\ \mathbf{of}\ \langle y, z\rangle s}\ \otimes$$

$$\frac{\Pi, P^x \vdash s}{\Pi \vdash \lambda x.s : \uparrow P}\ \uparrow \qquad \frac{\Pi \vdash s : M \quad \Sigma \vdash t : N}{\Sigma \bullet \Pi \vdash \langle t, s\rangle : M \oplus N}\ \oplus$$

Fig. 3. Sequent rules for $\mathbf{CNL}^{pol}$

Lemma 1. *$\Pi, \Sigma \vdash s$ and $\Pi \vdash s : N$ imply, respectively, $\sigma(\Pi), \sigma(\Sigma) \vdash s : \bot$ and $\sigma(\Pi) \vdash s : \sigma^-(N)$ in **LP**.*

Lemma 2. *If $\Pi, \Sigma \vdash s$ or $\Pi \vdash s : N$, then s is in β-normal form.*

Example 1. If formulas are categories, then structures are the binary branching trees oft encountered in linguistic analysis, while proofs are syntactic derivations. We illustrate this correspondence through the following sample expressions, involving subject relativization (1) and quantifier scope ambiguities (2).

(1) (the) lemma that confuses Hilbert
(2) A lemma confuses everyone.

To model these data, we use atomic propositions s, np and n, categorizing sentences, noun phrases and nouns respectively. Our lexicon assigns the following (positive) categories to the words involved:

Word	Category	Lambek category
Hilbert	np	np
everyone	$\downarrow(\uparrow np \oplus \bar{s}) \otimes s$	$(np/s) \otimes s$
a	$\downarrow(\uparrow np \oplus \bar{n})$	np/n
lemma	n	n
confuses	$\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np})$	$(np\backslash s)/np$
that	$\downarrow((\bar{n} \oplus \uparrow n) \oplus \uparrow(\downarrow \bar{s} \otimes np))$	$(n\backslash n)/(np\backslash s)$

For reference purposes, we included the closest corresponding traditional Lambek formulas, with forward and backward implications A/B and $B\backslash A$ understood as $A \oplus B^\perp$ and $B^\perp \oplus A$ respectively.[2] While the category for 'everyone' may appear unconventional, the key idea is to have it include a subformula of the form $\downarrow N$, allowing $(\downarrow)$ to fix its scope. Its contractibility to np is what further sets it apart from the commonly used traditional category $s/(np\backslash s)$. Figures 5 and 6 provide derivations for (1) and (2), using F.4's derivation of transitive clauses as a macro.

The terms derived in Figures 5 and 6 constitute the denotations of the expressions (1) and (2), *parameterized* over the denotations of the words contained therein via the use of free variables. Note in particular the derivation of distinct terms for (2), corresponding to the two available scopal readings.

[2] [7] demonstrates **CNL** embeds (intuitionistic) non-associative Lambek calculus.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{s^u \vdash u : \bar{s}}\,Ax}{s^u, \downarrow \bar{s}^z \vdash (z\ u)}\downarrow}{\downarrow \bar{s}^z, s^u \vdash (z\ u)}\,dp^1 \qquad \dfrac{}{np^x \vdash x : \bar{np}}\,Ax}{\dfrac{\downarrow \bar{s}^z \vdash \lambda u(z\ u) : \uparrow s}{\downarrow \bar{s}^z \bullet np^x \vdash \langle \lambda u(z\ u), x\rangle : \bar{np} \oplus \uparrow s}\uparrow}\oplus \qquad \dfrac{}{np^y \vdash y : \bar{np}}\,Ax}{np^y \bullet (\downarrow \bar{s}^z \bullet np^x) \vdash \langle y, \langle \lambda u(z\ u), x\rangle\rangle : (\bar{np} \oplus \uparrow s) \oplus \bar{np}}\oplus}{np^y \bullet (\downarrow \bar{s}^z \bullet np^x), \downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np})^v \vdash (v\ \langle y, \langle \lambda u(z\ u), x\rangle\rangle)}\downarrow$$

Fig. 4. A 'macro' for deriving transitive clauses

3.3 Lexical Semantics

The previous section left us with the task of instantiating the lexical parameters feeding the denotations found in Figures 5 and 6. In doing so, we will admit ourselves some leverage in the strictness of the resource management regime adopted thus far: while the 'derivational' semantics of §3.2 was meant to reflect the linearity of the source $\mathbf{CNL}^{pol}$, the lexical semantics, constituting our means of referring to the world around us, need not be so restricted.

Put in more precise terms, in filling the lexical gap left by §3.2, we permit access to the full simply-typed λ-calculus, augmented by the logical constants of first-order predicate logic. To start with, the type language is revised thus:

$$\tau, \sigma \ ::= \ e \mid t \mid (\sigma \to \tau) \mid (\sigma \times \tau)$$

We have adopted base types e and t, interpreting, respectively, a fixed set of 'entities', or discourse referents, and the Boolean truth values. Complex types are either implications or products, the latter allowing the formation of pairs $\langle s, t\rangle$ and the left- and right projections $\pi_1(s)$ and $\pi_2(s)$. In updating our lexical entries, we associate words of category P with terms of type $\sigma^+(P)$, 'delinearized' by systematically replacing $\bot$, $(\tau \otimes \sigma)$ and $\neg\tau$ with t, $\tau \times \sigma$ and $\tau \to t$ respectively:

Word	Category	Denotations
Hilbert	np	**hilbert**
everyone	$\downarrow(\uparrow np \oplus \bar{s}) \otimes s$	$\langle \lambda\langle p, P\rangle \forall y((\mathbf{person}\ y) \Rightarrow (P\ y)) \wedge p, \top\rangle$
a	$\downarrow(\uparrow np \oplus \bar{n})$	$\lambda\langle Q, P\rangle \exists x((Q\ x) \wedge (P\ x))$
lemma	n	**lemma**
confuses	$\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np})$	$\lambda\langle x, \langle q, y\rangle\rangle (q\ ((\mathbf{confuse}\ y)\ x))$
that	$\downarrow((\bar{n} \oplus \uparrow n) \oplus \uparrow(\downarrow \bar{s} \otimes np))$	$\lambda\langle R, \langle Z, Q\rangle\rangle (Z\ \lambda z((Q\ z) \wedge (R\ \langle \lambda p\ p, z\rangle)))$

Here, $\lambda\langle y, z\rangle t$ abbreviates $\lambda x^{\tau\times\sigma} t[\pi_1(x)/y, \pi_2(x)/z]$. Note the use of nonlogical constants **hilbert**, **person**, **lemma** and **confuse**, of types e, $e \to t$ ($\times 2$) and $e \to (e \to t)$ respectively. Like types, the linear terms of §3.2 carry over straightforwardly: just replace case analyses **case** s **of** $\langle x, y\rangle t$ with projections $t[\pi_1(s)/x, \pi_2(s)/y]$. Substitution of the above lexical denotations for the appropriate variables then yields the following terms:

(1) $(3\ \lambda z((\mathbf{lemma}\ z) \wedge ((\mathbf{confuse\ hilbert})\ z)))$
(2a) $\forall y((\mathbf{person}\ y) \Rightarrow \exists x((\mathbf{lemma}\ x) \wedge (3\ ((\mathbf{confuses}\ y)\ x)))) \wedge \top)$
(2b) $\exists x((\mathbf{lemma}\ x) \wedge (\forall y((\mathbf{person}\ y) \Rightarrow (3\ ((\mathbf{confuses}\ y)\ x))) \wedge \top))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{n \vdash \bar{n}}\,Ax}{n, \downarrow \bar{n} \vdash}\downarrow}{\downarrow \bar{n}, n \vdash}dp \quad \dfrac{}{n \vdash \bar{n}}\,Ax}{\downarrow \bar{n} \vdash \uparrow n}\uparrow}{\downarrow \bar{n} \bullet n \vdash \bar{n} \oplus \uparrow n}\oplus \quad \dfrac{}{np \bullet (\downarrow \bar{s} \bullet np), \downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \vdash}\downarrow}{\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np, \downarrow \bar{s} \bullet np \vdash}dp}{\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np, \downarrow \bar{s} \otimes np \vdash}\otimes}{\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np \vdash \uparrow(\downarrow \bar{s} \otimes np)}\uparrow}{(\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np) \bullet (\downarrow \bar{n} \bullet n) \vdash (\bar{n} \oplus \uparrow n) \oplus \uparrow(\downarrow \bar{s} \otimes np)}\oplus}{(\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np) \bullet (\downarrow \bar{n} \bullet n), \downarrow((\bar{n} \oplus \uparrow n) \oplus \uparrow(\downarrow \bar{s} \otimes np)) \vdash}\downarrow}{n \bullet (\downarrow((\bar{n} \oplus \uparrow n) \oplus \uparrow(\downarrow \bar{s} \otimes np)) \bullet (\downarrow((\bar{np} \oplus \uparrow s) \oplus \bar{np}) \bullet np)), \downarrow \bar{n} \vdash}dp$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{2 \vdash 2}\,Ax}{2, 3 \vdash (3\ 2)}\downarrow}{3, 2 \vdash (3\ 2)}dp \quad \dfrac{}{1 \vdash 1}\,Ax}{3 \vdash \lambda 2(3\ 2)}\uparrow}{3 \bullet 1 \vdash \langle \lambda 2(3\ 2), 1\rangle}\oplus \quad \dfrac{}{7 \bullet (6 \bullet 4), 8 \vdash (8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle)}\downarrow}{8 \bullet 7, 6 \bullet 4 \vdash (8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle)}dp}{8 \bullet 7, 9 \vdash \textbf{case}\ 9\ \textbf{of}\ \langle 6, 4\rangle(8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle)}\otimes}{8 \bullet 7 \vdash \lambda\langle 6, 4\rangle(8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle)}\uparrow}{(8 \bullet 7) \bullet (3 \bullet 1) \vdash \langle \lambda\langle 6, 4\rangle(8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle), \langle \lambda 2(3\ 2), 1\rangle\rangle}\oplus}{(8 \bullet 7) \bullet (3 \bullet 1), 10 \vdash (10\ \langle \lambda\langle 6, 4\rangle(8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle), \langle \lambda 2(3\ 2), 1\rangle\rangle)}\downarrow}{1 \bullet (10 \bullet (8 \bullet 7)), 3 \vdash (10\ \langle \lambda\langle 6, 4\rangle(8\ \langle 7, \langle \lambda 5(6\ 5), 4\rangle\rangle), \langle \lambda 2(3\ 2), 1\rangle\rangle)}dp$$

Fig. 5. Derivation illustrating subject relativization. Reasons of space dictate the spreading of the syntactic and semantic components over separate derivation trees. Furthermore, due to the number of variables involved in the semantics, we use numbers as names (not to be confused with De Bruijn notation).

In each case, we are left with a free variable 3, corresponding to the category of the whole expression. Also, for (2a) and (2b), note $\phi \wedge \top$ is logically equivalent with ϕ. Alternatively, one may use the more complex category $\downarrow(\uparrow np \oplus \uparrow s) \otimes \downarrow \bar{s}$ for 'everyone', with denotation $\lambda\langle q, P\rangle \forall y(q\ ((\textbf{person}\ y) \Rightarrow (P\ y))), \lambda p\ p\rangle$.

4 Comparing Provability in CNL^{pol} and CNL

We defined polarized **CNL** and briefly demonstrated its application to formal semantics. Suffice it to assure that **CNL** and **CNL**^{pol} coincide on provability. One direction is straightforward: a derivation in **CNL**^{pol} is easily rewritten into one of **CNL** by dropping the shifts. Conversely, we witness a decoration of **CNL** formulae with shifts $\uparrow, \downarrow$, generating no (sub)formulas of the form $\downarrow\uparrow P$ or $\uparrow\downarrow N$, s.t. provability in **CNL** implies provability in **CNL**^{pol} w.r.t. said decoration.

The bulk of this section is devoted to proving closure of **CNL**^{pol} under Cut, using an argument involving phase spaces ([13]). Said models are defined in §4.1, together with proofs of soundness and completeness. §4.2 uses the results of §4.1 to prove Cut admissibility, and provides the decoration by which we demonstrate the completeness of **CNL**^{pol} w.r.t. **CNL**. In what is to follow, we will only be concerned with provability, hence omit any mention of term annotations.

$$np^4 \bullet (\downarrow \bar{s}^3 \bullet np^1), \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \vdash (5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) \quad F.4$$

$$(\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet np^4) \bullet \downarrow \bar{s}^3, np^1 \vdash (5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) \quad dp$$

$$(\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet np^4) \bullet \downarrow \bar{s}^3 \vdash \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) : \uparrow np \quad \uparrow \qquad n^6 \vdash 6 : \bar{n} \quad Ax$$

$$n^6 \bullet ((\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet np^4) \bullet \downarrow \bar{s}^3) \vdash \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle : \uparrow np \oplus \bar{n} \quad \oplus$$

$$n^6 \bullet ((\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet np^4) \bullet \downarrow \bar{s}^3), \downarrow(\uparrow np \oplus \bar{n})^7 \vdash (7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) \quad \downarrow$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, np^4 \vdash (7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) \quad dp$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \vdash \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) : \uparrow np \quad \uparrow \qquad s^8 \vdash 8 : \bar{s} \quad Ax$$

$$s^8 \bullet ((\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5) \vdash \langle 8, \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle : \uparrow np \oplus \bar{s} \quad \oplus$$

$$s^8 \bullet ((\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5), \downarrow(\uparrow np \oplus \bar{s})^9 \vdash (9\ \langle 8, \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad \downarrow$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, \downarrow(\uparrow np \oplus \bar{s})^9 \bullet s^8 \vdash (9\ \langle 8, \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad dp$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10} \vdash \textbf{case}\ 10\ \textbf{of}\ \langle 9, 8\rangle(9\ \langle 8, \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad \otimes$$

$$(\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6) \bullet (\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10}), \downarrow \bar{s}^3 \vdash \textbf{case}\ 10\ \textbf{of}\ \langle 9, 8\rangle(9\ \langle 8, \lambda 4(7\ \langle 6, \lambda 1(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad dp$$

$$np^4 \bullet (\downarrow \bar{s}^3 \bullet np^1), \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \vdash (5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) \quad F.4$$

$$(\downarrow \bar{s}^3 \bullet np^1) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, np^4 \vdash (5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) \quad dp$$

$$(\downarrow \bar{s}^3 \bullet np^1) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \vdash \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle) : \uparrow np \quad \uparrow \qquad s^8 \vdash 8 : \bar{s} \quad Ax$$

$$s^8 \bullet ((\downarrow \bar{s}^3 \bullet np^1) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5) \vdash \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle : \uparrow np \oplus \bar{s} \quad \oplus$$

$$s^8 \bullet ((\downarrow \bar{s}^3 \bullet np^1) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5), \downarrow(\uparrow np \oplus \bar{s})^9 \vdash (9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) \quad \downarrow$$

$$(\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10}) \bullet \downarrow \bar{s}^3, np^1 \vdash (9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) \quad dp$$

$$(\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10}) \bullet \downarrow \bar{s}^3 \vdash \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle) : \uparrow np \quad \uparrow \qquad n^6 \vdash 6 : \bar{n} \quad Ax$$

$$n^6 \bullet ((\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s})^9 \bullet s^8) \bullet \downarrow \bar{s}^3) \vdash \langle 6, \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle : \uparrow np \oplus \bar{n} \quad \oplus$$

$$n^6 \bullet ((\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s})^9 \bullet s^8) \bullet \downarrow \bar{s}^3), \downarrow(\uparrow np \oplus \bar{n})^7 \vdash (7\ \langle 6, \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad \downarrow$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, \downarrow(\uparrow np \oplus \bar{s})^9 \bullet s^8 \vdash (7\ \langle 6, \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad dp$$

$$(\downarrow \bar{s}^3 \bullet (\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6)) \bullet \downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5, \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10} \vdash \textbf{case}\ 10\ \textbf{of}\ \langle 9, 8\rangle(7\ \langle 6, \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad \otimes$$

$$(\downarrow(\uparrow np \oplus \bar{n})^7 \bullet n^6) \bullet (\downarrow((\overline{np} \oplus \uparrow s) \oplus \overline{np})^5 \bullet \downarrow(\uparrow np \oplus \bar{s}) \otimes s^{10}), \downarrow \bar{s}^3 \vdash \textbf{case}\ 10\ \textbf{of}\ \langle 9, 8\rangle(7\ \langle 6, \lambda 1(9\ \langle 8, \lambda 4(5\ \langle 4, \langle \lambda 2(3\ 2), 1\rangle\rangle)\rangle)\rangle) \quad dp$$

Fig. 6. Derivations illustrating quantifier scope ambiguities

4.1 Phase Spaces

Definition 11. A *phase space* is a 3-tuple $\langle P, \bullet, \bot \rangle$ where:

1. P is a non-empty set of *phases* closed under $\bullet : P \times P \to P$.
2. $\bot \subseteq P \times P$ s.t. $\forall x, y, z \in P$,

$$\langle x, y \rangle \in \bot \Rightarrow \langle y, x \rangle \in \bot$$
$$\langle x \bullet y, z \rangle \in \bot \Leftrightarrow \langle x, y \bullet z \rangle \in \bot$$

As usual, we often identify a phase space by its carrier set P. Define $\cdot^\bot : \mathscr{P}(P) \to \mathscr{P}(P)$ by $A \mapsto \{x \mid (\forall y \in A)(\langle x, y \rangle \in \bot)\}$.

Lemma 3. *Given P, one easily shows $A \subseteq B^\bot$ iff $B \subseteq A^\bot$ $(A, B \in \mathscr{P}(P))$. In other words, $\cdot^\bot$ is a Galois connection, and hence $\cdot^{\bot\bot}$ a closure operator.*

Formulas will be interpreted by *facts*: subsets $A \subseteq P$ s.t. $A = A^{\bot\bot}$.

Definition 12. A *model* consists of a phase space P and a valuation v taking positive atoms p into facts. v extends to maps $v^+(\cdot)$ and $v^-(\cdot)$, defined by mutual induction and acting on arbitrary positive and negative formulas respectively:

$$v^+(p) := v(p) \qquad v^-(\bar{p}) := v(p)$$
$$v^+(P \otimes Q) := v^+(P) \times v^+(Q) \qquad v^-(M \oplus N) := v^-(N) \times v^-(M)$$
$$v^+(\downarrow N) := v^-(N)^\bot \qquad v^-(\uparrow P) := v^+(P)^\bot$$

Here, $\times : \mathscr{P}(P) \times \mathscr{P}(P) \to \mathscr{P}(P)$ is defined $A \times B := \{x \bullet y \mid x \in A^\bot,\ y \in B^\bot\}^\bot$. Since Lemma 3 implies $A \subseteq A^{\bot\bot}$ and $A^{\bot\bot\bot} \subseteq A^\bot$ for $A \subseteq P$, this set is a fact.

Lemma 4. *$v^+(P) = v^-(\uparrow P)^\bot$ and $v^-(N) = v^+(\downarrow N)^\bot$ for any N, P.*

Proof. Immediate, since the sets involved are facts.

Lemma 5. *For any N, P, $v^+(P) = v^-(P^\bot)$ and (dually) $v^-(N) = v^+(N^\bot)$.*

Proof. By a straightforward inductive argument.

Definition 13. To state soundness and completeness, we interpret a structure Π by dual formulas Π^+ and Π^-, in the sense that $\Pi^{+\bot} = \Pi^-$ and $\Pi^{-\bot} = \Pi^+$:

$$P^+ = P \qquad P^- = P^\bot$$
$$(\Pi \bullet \Sigma)^+ = \Pi^+ \otimes \Sigma^+ \qquad (\Pi \bullet \Sigma)^- = \Sigma^- \oplus \Pi^-$$

Lemma 6. *We have the following equivalences:*

$$v^-(\uparrow \Pi^+) \subseteq v^-(\Sigma^-) \Leftrightarrow v^-(\uparrow \Sigma^+) \subseteq v^-(\Pi^-)$$
$$\Leftrightarrow v^+(\downarrow \Sigma^-) \subseteq v^+(\Pi^+) \Leftrightarrow v^+(\downarrow \Pi^-) \subseteq v^+(\Sigma^+)$$

Proof. Recalling $\Upsilon^{+\bot} = \Upsilon^-$ and $\Upsilon^{-\bot} = \Upsilon^+$ for arbitrary Υ, we have

$$v^+(\downarrow \Sigma^-) \subseteq v^+(\Pi^+) \quad \text{iff} \quad v^+(\Pi^+)^\bot \subseteq v^+(\downarrow \Sigma^-)^\bot \quad \text{(Lemma 3)}$$
$$\text{iff} \quad v^-(\uparrow \Pi^+) \subseteq v^-(\Sigma^-) \quad \text{(Lemma 4)}$$

and

$$\begin{array}{lll} v^+(\downarrow \Sigma^-) \subseteq v^+(\Pi^+) & \text{iff} & v^+(\Pi^+)^\perp \subseteq v^+(\downarrow \Sigma^-)^\perp \quad \text{(Lemma 3)} \\ & \text{iff} & v^-(\Pi^-)^\perp \subseteq v^-(\uparrow \Sigma^+)^\perp \quad \text{(Lemma 5)} \\ & \text{iff} & v^+(\downarrow \Pi^-) \subseteq v^+(\Sigma^+) \quad \text{(Lemma 4)} \end{array}$$

and similarly $v^-(\uparrow \Pi^+) \subseteq v^-(\Sigma^-)$ iff $v^-(\uparrow \Sigma^+) \subseteq v^-(\Pi^-)$.

Theorem 14. *All phase models satisfy the following implications:*

$$\begin{array}{rcl} \Pi, \Sigma \vdash & \Longrightarrow & v^+(\downarrow \Pi^-) \subseteq v^+(\Sigma^+) \\ \Pi \vdash N & \Longrightarrow & v^-(N) \subseteq v^-(\Pi^-) \end{array}$$

Proof. By induction on the derivation witnessing $\Pi, \Sigma \vdash$ or $\Pi \vdash N$, freely making use of L.6. The only nontrivial cases are $(\oplus)$ and (dp^2).

Case $(\oplus)$**.** Since $v^-(\Pi^-)^\perp \subseteq v^-(M)^\perp$ and $v^-(\Sigma^-)^\perp \subseteq v^-(N)^\perp$ by the induction hypothesis and L.3, we have

$$\{y \bullet x \mid x \in v^-(\Pi^-)^\perp,\ y \in v^-(\Sigma^-)^\perp\} \subseteq \{y \bullet x \mid x \in v^-(M)^\perp,\ y \in v^-(N)^\perp\}$$

with another application of L.3 ensuring $v^-(M \oplus N) \subseteq v^-(\Pi^- \oplus \Sigma^-)$.

Case (dp^2)**.** We check one direction, establishing

$$\frac{((v^+(\Sigma^+) \times v^+(\Upsilon^+))^\perp =)\ v^-(\uparrow(\Sigma \bullet \Upsilon)^+) \subseteq v^-(\Pi^-)}{v^-(\uparrow \Upsilon^+) \subseteq v^-((\Pi \bullet \Sigma)^-)\ (= \{x \bullet y \mid x \in v^-(\Pi^-)^\perp,\ y \in v^-(\Sigma^-)^\perp\}^\perp)}$$

applying L.6 twice to obtain the desired result. Thus, it suffices to prove $\langle z, x \bullet y\rangle \in \perp$ on the assumptions (a) $z \in v^-(\uparrow \Upsilon^+)$; (b) $x \in v^-(\Pi^-)^\perp$; and (c) $y \in v^-(\Sigma^-)^\perp$. By applying L.3 on the induction hypothesis, (b) implies $x \in v^+(\Sigma^+) \times v^+(\Upsilon^+) = \{y \bullet z \mid y \in v^+(\Sigma^+)^\perp,\ z \in v^+(\Upsilon^+)^\perp\}^\perp$. By (a), (c) and Lemmas 4 and 5, $y \in v^+(\Sigma^+)^\perp$ and $z \in v^+(\Upsilon^+)^\perp$, so that $\langle x, y \bullet z\rangle \in \perp$, iff $\langle z, x \bullet y\rangle \in \perp$.

Definition 15. Completeness will be established w.r.t. the *syntactic (phase) model*, defined by taking the structures Π as phases, setting $\langle \Pi, \Sigma\rangle \in \perp$ iff $\Pi, \Sigma \vdash$ and letting $v(p) = \{p\}^\perp = \{\Pi \mid \Pi, p \vdash\}$.

The following is our central lemma, resembling results of Okada ([13]) and Herbelin and Lee ([10]) for linear and intuitionistic logic respectively.

Lemma 7. *For arbitrary P, N, Π, Σ, the syntactic model satisfies:*

(i) $\Pi \in v^-(N)$ *implies* $\Pi, N^\perp \vdash$
(ii) $(\forall \Pi)(\Pi \vdash N \Rightarrow \Pi, \Sigma \vdash)$ *implies* $\Sigma \in v^-(N)$
(iii) $\Pi \in v^+(P)$ *implies* $\Pi, P \vdash$
(iv) $(\forall \Pi)(\Pi \vdash P^\perp \Rightarrow \Pi, \Sigma \vdash)$ *implies* $\Sigma \in v^+(P)$

Proof. First, note that if $\Pi \vdash N$ ($\Pi \vdash P^\perp$), then also $\Pi, \downarrow N \vdash$ ($\Pi, \downarrow P^\perp \vdash$) by applying ($\downarrow$). Consequently, (ii) and (iv) imply, respectively, $\downarrow N \in v^-(N)$ and $\downarrow P^\perp \in v^+(P)$. In practice, when invoking the induction hypothesis for (ii) or (iv), we often immediately instantiate them by the latter consequences. To prove (i)-(iv), we proceed by a simultaneous induction on P, N. We suffice by checking the cases $\bar{p}$, $\uparrow P$ and $M \oplus N$, the remaining p, $\downarrow N$ and $P \otimes Q$ being entirely dual.

Case $\bar{p}$. We show (i) and (ii).

(i) Since $v^-(\bar{p}) = \{p\}^\perp$, $\Pi \in v^-(p)$ implies $\Pi, p \vdash$ by definition.
(ii) $\Pi \vdash \bar{p}$ iff $\Pi = p$, and $p, \Sigma \vdash$ again implies $\Sigma \in v^-(\bar{p})$ by definition.

Case $\uparrow P$. We show (i) and (ii).

(i) Suppose $\Pi \in v^-(\uparrow P) = v^+(P)^\perp$. By IH(iv), $\downarrow P^\perp \in v^+(P)$, so that $\langle \Pi, \downarrow P^\perp \rangle \in \perp$, and hence $\Pi, \downarrow P^\perp \vdash$ by definition of $\perp$.
(ii) We show $\Sigma \in v^-(\uparrow P) = v^+(P)^\perp$, assuming (a) $\Pi \vdash \uparrow P$ implies $\Pi, \Sigma \vdash$ for all Π. Letting (b) $\Upsilon \in v^+(P)$, it suffices to ensure $\Sigma, \Upsilon \vdash$. IH(iii) and (b) imply $\Upsilon, P \vdash$, hence $\Upsilon \vdash \uparrow P$ by ($\uparrow$). Thus, $\Upsilon, \Sigma \vdash$ by (a), and we apply (dp^1).

Case $M \oplus N$. We show (i) and (ii).

(i) Let (a) $\Pi \in v^-(M \oplus N)$. We show $\Pi, N^\perp \bullet M^\perp \vdash$, implying $\Pi, N^\perp \otimes M^\perp \vdash$ by ($\otimes$). By (a), it suffices to ensure $M^\perp \in v^-(M)^\perp$ and $N^\perp \in v^-(N)^\perp$. I.e., we must ascertain $\Sigma, M^\perp \vdash$ and $\Upsilon, N^\perp \vdash$ on the assumptions $\Sigma \in v^-(M)$ and $\Upsilon \in v^-(N)$, but these are immediate consequences of IH(i) and (dp^2).
(ii) The following hypotheses will be used:

(a) $\Pi \vdash M \oplus N$ implies $\Pi, \Sigma \vdash$ for all Π
(b) $\Upsilon_1 \in v^-(M)^\perp$
(c) $\Upsilon_2 \in v^-(N)^\perp$
(d) $(\forall \Pi_1)(\Pi_1 \vdash M \Rightarrow \Pi_1, \Sigma \bullet \Upsilon_2 \vdash)$ implies $\Sigma \bullet \Upsilon_2 \in v^-(M)$
(e) $\Pi_1 \vdash M$
(f) $(\forall \Pi_2)(\Pi_2 \vdash N \Rightarrow \Pi_2, \Pi_1 \bullet \Sigma \vdash)$ implies $\Pi_1 \bullet \Sigma \in v^-(M)$
(g) $\Pi_2 \vdash N$

Assuming (a), we show $\Sigma \in v^-(M \oplus N)$, iff $\Sigma, \Upsilon \vdash$ for all $\Upsilon \in \{\Upsilon_2 \bullet \Upsilon_1 \mid \Upsilon_1 \in v^-(M)^\perp, \Upsilon_2 \in v^-(N)^\perp\}$. So assume (b), (c). Since $\Sigma, \Upsilon_2 \bullet \Upsilon_1 \vdash$ iff $\Sigma \bullet \Upsilon_2, \Upsilon_1 \vdash$ by (dp^2), it suffices by (b) to prove $\Sigma \bullet \Upsilon_2 \in v^-(M)$. By (d), i.e., IH(ii), we need only prove $\Pi_1, \Sigma \bullet \Upsilon_2 \vdash$ on the assumption (e), iff $\Pi_1 \bullet \Sigma, \Upsilon_2 \vdash$ by (dp^2). Applying (c), we must show $\Pi_1 \bullet \Sigma \in v^-(M)$, which follows from (f), i.e., IH(ii), if we can prove $\Pi_2, \Pi_1 \bullet \Sigma \vdash$ on the assumption (g). By (dp^2) and (a), this follows from $\Pi_2 \bullet \Pi_1 \vdash M \oplus N$, witnessed by (e), (g) and ($\oplus$).

Lemma 8. *We have the following implications:*

(i) $v^+(\Sigma^+) \subseteq \{\Pi \mid \Pi, \Sigma \vdash\}$ *implies* $\Sigma \in v^-(\uparrow \Sigma^+)$
(ii) $v^-(\Sigma^-) \subseteq \{\Pi \mid \Pi, \Sigma \vdash\}$ *implies* $\Sigma \in v^+(\downarrow \Sigma^-)$

Proof. We show (i), with (ii) being dual:

$$\begin{array}{lll} & v^+(\Sigma^+) \subseteq \{\Pi \mid \Pi, \Sigma \vdash\} & \\ \Leftrightarrow & \{\Pi \mid \Pi, \Sigma \vdash\}^\perp \subseteq v^+(\Sigma^+)^\perp & \text{(L.3)} \\ \Leftrightarrow & \{\Upsilon \mid (\forall \Pi)(\Pi, \Sigma \vdash \Rightarrow \Upsilon, \Pi \vdash)\} \subseteq v^-(\uparrow \Sigma^+) & \text{(def.)} \end{array}$$

And evidently $\Sigma \in \{\Upsilon \mid (\forall \Pi)(\Pi, \Sigma \vdash \Rightarrow \Upsilon, \Pi \vdash)\}$.

Lemma 9. *We have (i) $\Pi \in v^+(\Sigma^+)$ implies $\Pi, \Sigma \vdash$; and (ii) $\Pi \in v^-(\Sigma^-)$ implies $\Pi, \Sigma \vdash$ for arbitrary Σ, Π.*

Proof. By induction on Σ. The base case reduces to (i) and (iii) of L.7. For $\Sigma = \Sigma_1 \bullet \Sigma_2$, we prove (i), with (ii) being similar. So let $\Pi \in v^+(\Sigma_1^+ \otimes \Sigma_2^+)$. The desired result follows if we can show $\Sigma_1 \in v^+(\Sigma_1^+)^\perp = v^-(\uparrow \Sigma_1^+)$ and $\Sigma_2 \in v^+(\Sigma_2^+)^\perp = v^-(\uparrow \Sigma_2^+)$. But this holds by virtue of IH(i) and L.8(i).

Theorem 16. *If $v^+(\downarrow \Pi^-) \subseteq v^+(\Sigma^+)$ in the syntactic model, then $\Pi, \Sigma \vdash$.*

Proof. By L.8&9, $\Pi \in v^+(\downarrow \Pi^-)$, hence $\Pi \in v^+(\Sigma^+)$, so $\Pi, \Sigma \vdash$ by L.9.

4.2 Cut Admissibility and Completeness w.r.t. CNL

We proceed with the completeness proof for derivability in $\mathbf{CNL}^{pol}$ w.r.t. **CNL**, to be witnessed by the following decoration of **CNL** formulae with shifts.

Definition 17. For A a formula of **CNL**, let $\epsilon(A) = +$ if A is of the form p or $B \otimes C$, and $\epsilon(A) = -$ otherwise. We translate A into a formula $\updownarrow(A)$ of $\mathbf{CNL}^{pol}$ with no subformulas of the form $\downarrow\uparrow P$ or $\uparrow\downarrow N$. In the base cases, $\updownarrow(p) = p$, $\updownarrow(\bar{p}) = \bar{p}$, while for complex formulae,

$\epsilon(A)$	$\epsilon(B)$	$\updownarrow(A \otimes B)$	$\updownarrow(A \oplus B)$
$+$	$+$	$\updownarrow(A) \otimes \updownarrow(B)$	$\uparrow\updownarrow(A) \oplus \uparrow\updownarrow(B)$
$+$	$-$	$\updownarrow(A) \otimes \downarrow\updownarrow(B)$	$\uparrow\updownarrow(A) \oplus \updownarrow(B)$
$-$	$+$	$\downarrow\updownarrow(A) \otimes \updownarrow(B)$	$\updownarrow(A) \oplus \uparrow\updownarrow(B)$
$-$	$-$	$\downarrow\updownarrow(A) \otimes \downarrow\updownarrow(B)$	$\updownarrow(A) \oplus \updownarrow(B)$

Definition 18. The map $\updownarrow(\cdot)$ is extended to the level of structures Γ as follows:

$$A \mapsto \begin{cases} \updownarrow(A) & \text{if } \epsilon(A) = + \\ \downarrow\updownarrow(A) & \text{if } \epsilon(A) = - \end{cases} \qquad \updownarrow(\Gamma \bullet \Delta) = \updownarrow(\Gamma) \bullet \updownarrow(\Delta)$$

Our intention is to show that $\Gamma, \Delta \vdash$ implies $\updownarrow(\Gamma), \updownarrow(\Delta) \vdash$.

Lemma 10. *The following rules are admissible for polarized **CNL**:*

$$\frac{}{\downarrow N, N^\perp \vdash} \qquad \frac{\Sigma, P \vdash \quad \Pi, \downarrow P^\perp \vdash}{\Pi, \Sigma \vdash}$$

Proof. L.7(ii) implies $\downarrow N \in v^-(N)$, so that $\downarrow N, N^\perp \vdash$ by L.7(i). Now suppose $\Pi, \downarrow P^\perp \vdash$ and $\Sigma, P \vdash$. By T.14, $v^+(\downarrow \Pi^-) \subseteq v^+(\downarrow P^\perp)$ and $v^+(\downarrow P^\perp) \subseteq v^+(\Sigma^+)$, hence $v^+(\downarrow \Pi^-) \subseteq v^+(\Sigma^+)$ so that $\Pi, \Sigma \vdash$ by T.16.

Lemma 11. *Let* $(\cdot)^\bullet$ *take positive formulas into structures, as follows:*

$$(p)^\bullet = p; \quad (\downarrow N)^\bullet = \downarrow N; \quad (P \otimes Q)^\bullet = (P)^\bullet \bullet (Q)^\bullet$$

then (i) $\Pi, (P)^\bullet \vdash$ *implies* $\Pi, P \vdash$*; and (ii)* $(N^\perp)^\bullet \vdash N$*, for any* P, N*.*

Proof. We prove (i) by induction on P. If $P = p$ or $P = \downarrow N$, the desired result is immediate. If $P = P_1 \otimes P_2$, proceed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi, (P_1)^\bullet \bullet (P_2)^\bullet \vdash}{(P_2)^\bullet \bullet \Pi, (P_1)^\bullet \vdash}\, dp^1, dp^2, dp^1}{\Pi \bullet P_1, (P_2)^\bullet \vdash}\, IH, dp^2, dp^1}{\Pi, P_1 \bullet P_2 \vdash}\, IH, dp^2}{\Pi, P_1 \otimes P_2 \vdash}\, \otimes$$

Similarly, we prove (ii) by induction on N:

$$\dfrac{}{p \vdash \bar{p}}\, Ax \qquad \dfrac{\dfrac{}{\downarrow P^\perp, P \vdash}\, L10}{\downarrow P^\perp \vdash \uparrow P}\, \uparrow \qquad \dfrac{\dfrac{}{(N_1^\perp)^\bullet \vdash N_1}\, IH \quad \dfrac{}{(N_2^\perp)^\bullet \vdash N_2}\, IH}{(N_2^\perp)^\bullet \bullet (N_1^\perp)^\bullet \vdash N_1 \oplus N_2}\, \oplus$$

$$(N = \bar{p}) \qquad (N = \uparrow P) \qquad (N = N_1 \oplus N_2)$$

Theorem 19. $\Gamma, \Delta \vdash$ *in* ***CNL*** *implies* $\updownarrow(\Gamma), \updownarrow(\Delta) \vdash$ *in* $\boldsymbol{CNL}^{pol}$*.*

Proof. We proceed by induction on the derivation establishing $\Gamma, \Delta \vdash$. The cases (Ax) and (Cut) follow immediately from C.10, possibly with some applications of (dp^1) depending on the value of $\epsilon(A)$. The case $(\otimes)$ is equally trivial, translating to an application of $(\otimes)$ in $\mathbf{CNL}^{pol}$. Thus, we are left to check

Case $(\oplus)$**.** Suppose $\Gamma, A \vdash$ and $\Delta, B \vdash$. Considering all possible values for $\epsilon(A)$ and $\epsilon(B)$, we have four subcases to check in total. As a typical case, we pick $\epsilon(A) = -$ and $\epsilon(B) = +$. Thus, by induction hypothesis, $\updownarrow(\Gamma), \downarrow\updownarrow(A) \vdash$ and $\updownarrow(\Delta), \updownarrow(B) \vdash$. We construct a derivation of $\updownarrow(\Delta) \bullet \updownarrow(\Gamma), \downarrow(A \oplus \uparrow B) \vdash$ as follows:

$$\dfrac{\updownarrow(\Delta), \updownarrow(B) \vdash \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\updownarrow(A)^\perp)^\bullet \vdash A}\, L11(ii) \quad \dfrac{\dfrac{}{\downarrow\updownarrow(B)^\perp, B \vdash}\, L10}{\downarrow\updownarrow(B)^\perp \vdash \downarrow B}\, \uparrow}{\downarrow\updownarrow(B)^\perp \bullet (\updownarrow(A)^\perp)^\bullet \vdash A \oplus \uparrow B}\, \oplus}{\downarrow\updownarrow(B)^\perp \bullet (\updownarrow(A)^\perp)^\bullet, \downarrow(A \oplus \uparrow B) \vdash}\, \downarrow}{\downarrow(A \oplus \uparrow B) \bullet \downarrow\updownarrow(B)^\perp, (\updownarrow(A)^\perp)^\bullet \vdash}\, dp^1, dp^2}{\downarrow(A \oplus \uparrow B) \bullet \downarrow\updownarrow(B)^\perp, \updownarrow(A)^\perp \vdash}\, L11(i) \qquad \updownarrow(\Gamma), \downarrow\updownarrow(A) \vdash}{\updownarrow(\Gamma), \downarrow(A \oplus \uparrow B) \bullet \downarrow\updownarrow(B)^\perp \vdash}\, L10}{\updownarrow(\Gamma) \bullet \downarrow(A \oplus \uparrow B), \downarrow\updownarrow(B)^\perp \vdash}\, dp^2}{\updownarrow(\Gamma) \bullet \downarrow(A \oplus \uparrow B), \updownarrow(\Delta) \vdash}\, L10}{\updownarrow(\Delta) \bullet \updownarrow(\Gamma), \downarrow(A \oplus \uparrow B) \vdash}\, dp^1, dp^2$$

5 Related Topics

We consider some related topics and directions for future research.

5.1 Focused Proof Search

Though not made explicit in their choice of terminology, Hepple ([9]) and Hendriks ([8], Ch.4) were the first to study focused proof search within the Lambek calculus, with the aim of eliminating spurious ambiguities from Cut-free derivations. Strictly speaking, $\mathbf{CNL}^{pol}$'s derivations do not fully conform to Andreoli's specifications, as the latter enforces the eager application of invertible inferences. In the lower derivation of F.6, however, nothing prevents us from postponing the application of $(\otimes)$ until right before the second application of $(\oplus)$ (adopting a top-down view). For the moment we have ignored such refinements, seeing as they have no bearing on the matter of finding a formal semantics.

5.2 The Lambek-Grishin Calculus

The ideas expressed in this article are not limited to **CNL**. To illustrate, we briefly discuss a proposal similar to that of De Groote and Lamarche, namely the *Lambek-Grishin calculus* of Moortgat and associates (**LG**, [12]). We sketch a polarized reinterpretation of **LG** (henceforth $\mathbf{LG}^{pol}$), adopting a syntax highlighting the similarities with **CNL**, though rather different from Moortgat's. Compared to **CNL**, **LG**'s formulae include explicit connectives for implications $/, \backslash$ and (the dual) subtractions $\oslash, \oslash$.

$$\begin{aligned} P, Q &::= p \mid (P \otimes Q) \mid (P \oslash M) \mid (M \oslash P) \mid (\downarrow N) \\ M, N &::= \bar{p} \mid (M \oplus N) \mid (P \backslash M) \mid (M / P) \mid (\uparrow P) \end{aligned}$$

Moortgat's original (two-sided) account of **LG** lacks negative atoms, added here for the purpose of allowing classical negation to be defined:

$$\begin{array}{rlrl} p^{\perp} &= \bar{p} & \bar{p}^{\perp} &= p \\ (P \otimes Q)^{\perp} &= Q^{\perp} \oplus P^{\perp} & (M \oplus N)^{\perp} &= N^{\perp} \otimes M^{\perp} \\ (P \oslash M)^{\perp} &= M^{\perp} \backslash P^{\perp} & (P \backslash M)^{\perp} &= M^{\perp} \oslash P^{\perp} \\ (M \oslash P)^{\perp} &= P^{\perp} / M^{\perp} & (M / P)^{\perp} &= P^{\perp} \oslash M^{\perp} \\ (\downarrow N)^{\perp} &= \uparrow N^{\perp} & (\uparrow P)^{\perp} &= \downarrow P^{\perp} \end{array}$$

The category-type correspondence is adapted straightforwardly:

$$\begin{array}{rcl@{\qquad}rcl} \sigma^{+}(p) &=& p & \sigma^{-}(\bar{p}) &=& p \\ \sigma^{+}(P \otimes Q) &=& \sigma^{+}(P) \otimes \sigma^{+}(Q) & \sigma^{-}(M \oplus N) &=& \sigma^{-}(N) \otimes \sigma^{-}(M) \\ \sigma^{+}(P \oslash M) &=& \sigma^{+}(P) \otimes \sigma^{-}(M) & \sigma^{-}(P \backslash M) &=& \sigma^{-}(M) \otimes \sigma^{+}(P) \\ \sigma^{+}(M \oslash P) &=& \sigma^{-}(M) \otimes \sigma^{+}(P) & \sigma^{-}(M / P) &=& \sigma^{+}(P) \otimes \sigma^{-}(M) \\ \sigma^{+}(\downarrow N) &=& \neg\, \sigma^{-}(N) & \sigma^{-}(\uparrow P) &=& \neg\, \sigma^{+}(P) \end{array}$$

The extended logical vocabulary is reflected in the definition of structures:

$$\Pi, \Sigma \ ::= \ P \mid (\Pi \bullet \Sigma) \mid (\Pi \bullet\!\!- \Sigma) \mid (\Sigma -\!\!\bullet \Pi)$$

Finally, sequents, as before, are of the form $\Pi, \Sigma \vdash s$ or $\Pi \vdash s : N$, and made subject to the following inference rules:

$$\frac{}{p^x \vdash x : \bar{p}}\ Ax \qquad \frac{\Pi \vdash s : N}{\Pi, \downarrow N^x \vdash (x\ s)}\ \downarrow \qquad \frac{\Pi, P^x \vdash s}{\Pi \vdash \lambda x.s : \uparrow P}\ \uparrow$$

$$\frac{\Gamma, \Delta \vdash s}{\Delta, \Gamma \vdash s}\ dp^1 \qquad \frac{\Pi, P^y \bullet Q^z \vdash s}{\Pi, P \otimes Q^x \vdash \mathbf{case}\ x\ \mathbf{of}\ \langle y, z\rangle s}\ \otimes \qquad \frac{\Pi \vdash s : M \quad \Sigma \vdash t : N}{\Sigma \bullet \Pi \vdash \langle s, t\rangle : M \oplus N}\ \oplus$$

$$\frac{\Pi \bullet \Sigma, \Upsilon \vdash s}{\Pi, \Sigma \multimap \Upsilon \vdash s}\ dp^2 \qquad \frac{\Pi, P^y \bullet\!\!- M^{\perp z} \vdash s}{\Pi, P \oslash M^x \vdash \mathbf{case}\ x\ \mathbf{of}\ \langle y, z\rangle s}\ \oslash \qquad \frac{\Pi \vdash s : M \quad \Sigma \vdash t : P^\perp}{\Pi \bullet\!\!- \Sigma \vdash \langle s, t\rangle : P \backslash M}\ \backslash$$

$$\frac{\Pi \bullet\!\!- \Sigma, \Upsilon \vdash s}{\Pi, \Sigma \bullet \Upsilon \vdash s}\ dp^2 \qquad \frac{\Pi, M^{\perp y} \multimap\!\!\bullet P^z \vdash s}{\Pi, M \oslash P^x \vdash \mathbf{case}\ x\ \mathbf{of}\ \langle y, z\rangle s}\ \oslash \qquad \frac{\Pi \vdash s : M \quad \Sigma \vdash t : P^\perp}{\Sigma \multimap \bullet \Pi \vdash \langle t, s\rangle : M / P}\ /$$

Completeness w.r.t. traditional (unpolarized) **LG** may again be established through use of phase spaces. Details are left to the reader's imagination.

A competing proposal for associating **LG** with a formal semantics is offered by Bernardi and Moortgat ([3]), who define dual call-by-name and call-by-value continuation-passing style translations into the implicational fragment of intuitionistic multiplicative linear logic. The lack of products in the target language results in semantic types often containing a larger number of double negations. While the terms used in lexical semantics thereby also grow in size, it does offer the grammar engineer more possibilities for fine-tuning his denotations.

5.3 Normalization by Evaluation

As a final remark, we point out the constructivity of §4.1's semantic completeness proof: rather than finding a countermodel for each unprovable sequent, we found a model wherein every truth translates into a proof. Explicating the algorithmic content underlying said proof, we might be able to show normalization for

$$\mathbf{CNL}^{pol} + \frac{\Sigma, P^y \vdash t \quad \Pi, P^{\perp x} \vdash s}{\Pi, \Sigma \vdash (\lambda xs\ \lambda yt)}\ Cut$$

In other words, rather than ignoring the term labeling and demonstrating closure under Cut for provability, like we did in §4, we might instead ascertain the latter property to hold of the *proofs* themselves as well. As a result, D.17 and T.19, when composed with L.1, may then be understood as a double negation translation for (unpolarized) **CNL**. Such an argument, when involving the algorithmic content underlying a demonstration of model-theoretic completeness, makes essential use of a formalization of the constructive metalanguage, say Martin-Löf type theory or the Calculus of Constructions, as clearly argued in [4]. We leave these issues for future research.

Acknowledgements. This work has benefited from discussions with Michael Moortgat and Vincent van Oostrom, as well as from the comments of three anonymous referees. All remaining errors are my own.

References

1. Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. Belnap, N.: Display logic. Journal of Philosophical Logic 11(4), 375–417 (1982)
3. Bernardi, R., Moortgat, M.: Continuation semantics for the Lambek-Grishin calculus. Information and Computation 208(5), 397–416 (2010)
4. Coquand, C.: From semantics to rules: a machine assisted analysis. In: Börger, E., Gurevich, Y., Meinke, K. (eds.) CSL 1993. LNCS, vol. 832, pp. 91–105. Springer, Heidelberg (1994)
5. Girard, J.Y.: A new constructive logic: classical logic. Mathematical Structures in Computer Science 1(3), 255–296 (1991)
6. Girard, J.Y.: On the meaning of logical rules II: multiplicatives and additives. In: Foundation of Secure Computation, pp. 183–212. IOS Press, Amsterdam (2000)
7. De Groote, P., Lamarche, F.: Classical non associative Lambek calculus. Studia Logica 71, 355–388 (2002)
8. Hendriks, H.: Studied flexibility. Categories and types in syntax and semantics. Ph.D. thesis, ILLC Amsterdam (1993)
9. Hepple, M.: Normal form theorem proving for the Lambek calculus. In: COLING, pp. 173–178 (1990)
10. Herbelin, H., Lee, G.: Forcing-based cut-elimination for gentzen-style intuitionistic sequent calculus. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS, vol. 5514, pp. 209–217. Springer, Heidelberg (2009)
11. Montague, R.: Universal grammar. Theoria 36(3), 373–398 (1970)
12. Moortgat, M.: Symmetric categorial grammar. Journal of Philosophical Logic 38(6), 681–710 (2009)
13. Okada, M.: A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. Theoretical Computer Science 281(1-2), 471–498 (2002)
14. Prawitz, D.: Natural Deduction. Dover Publications, New York (2006)
15. Wadler, P.: A taste of linear logic. In: Borzyszkowski, A., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 185–210. Springer, Heidelberg (1993)
16. Lafont, Y., Reus, B., Streichter, T.: Continuation semantics or expressing implication by negation. Technical Report 93-21, University of Munich (1993)
17. Yetter, D.: Quantales and (noncommutative) linear logic. Journal of Symbolic Logic 55(1), 41–64 (1990)
18. Zeilberger, N.: The logical basis of evaluation order and pattern-Matching. Ph.D. thesis, Carnegie Mellon University (2009)

The Product-Free Lambek-Grishin Calculus Is NP-Complete

Jeroen Bransen

Utrecht University, The Netherlands

Abstract. The Lambek-Grishin calculus **LG** is the symmetric extension of the non-associative Lambek calculus **NL**. In this paper we prove that the derivability problem for the product-free fragment of **LG** is NP-complete, thus improving on Bransen (2010) where this is shown for **LG** with product.

1 Introduction

In his 1958 and 1961 papers, Lambek formulated two versions of the Syntactic Calculus: in (Lambek, 1958), types are assigned to *strings*, which are then combined by an *associative* concatenation operation; in (Lambek, 1961), types are assigned to *phrases* (bracketed strings), and the composition operation is non-associative. These two versions of the Syntactic Calculus are known as as **L** and **NL** respectively.

In terms of recognizing capacity, both **NL** and **L** are strictly context-free. For **NL**, this was shown in Kandulski (1988). Pentus (1993b) showed that also all languages recognized by **L** are context-free: the more liberal associative composition operation does not bring extra recognizing power. It is well known that there are natural language patterns that require expressivity beyond context-free. The original versions of the Syntactic Calculus cannot handle such patterns.

Turning to computational complexity, de Groote (1999) showed that the derivability problem for **NL** belongs to `PTIME`. For **L**, it has been proved by Pentus (2003) that this problem is NP-complete and thus belongs to a class of hard problems. Savateev (2009) shows that the NP-completeness result already holds for the *product-free* fragment of **L**. Given the fact that the step from **NL** to **L** does not lead to recognition beyond context-free, the NP-completeness of **L** would seem to be a high price to pay.

Several extensions of the Syntactic Calculus have been proposed to obtain a higher expressivity. The subject of this paper is the the Lambek-Grishin calculus **LG** (Moortgat, 2007, 2009). **LG** is a *symmetric* extension of the nonassociative Lambek calculus **NL**. In addition to $\otimes, \backslash, /$ (product, left and right division), **LG** has dual operations $\oplus, \oslash, \oslash$ (coproduct, left and right difference). These two families are related by linear distributivity principles.

As for the generative power of **LG**, Moot (2007) gives an embedding of *Lexicalized Tree Adjoining Grammars* (LTAG) in **LG**. Melissen (2009) shows that all

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 49–63, 2011.

languages which are the intersection of a context-free language and the permutation closure of a context-free language are recognizable in **LG**. This places the lower bound for **LG** recognition beyond LTAG. The upper bound is still open.

We have shown in (Bransen, 2010) that the derivability problem for **LG** is NP-complete. The present paper improves on this result by proving that NP-completeness also holds for the *product-free* fragment of **LG**, which is less expressive. Similar to what Pentus did for **L**, we establish our result by means of a reduction from SAT. In the case of the construction for **LG**, the distributivity principles for the interaction between the $\otimes$ and $\oplus$ families play a key role.

2 Lambek-Grishin Calculus

We define the formula language of the product-free fragment of **LG** as follows.

Let *Var* be a set of *primitive types*, we use lowercase letters to refer to an element of *Var*. Let *formulas* be constructed using primitive types and the binary connectives $/$, $\backslash$, $\oslash$ and $\otimes$ as follows:

$$A, B ::= p \mid A/B \mid B\backslash A \mid A \oslash B \mid B \otimes A$$

The sets of *input* and *output structures* are constructed using formulas and the binary structural connectives $\cdot \otimes \cdot$, $\cdot/\cdot$, $\cdot\backslash\cdot$, $\cdot \oplus \cdot$, $\cdot \oslash \cdot$ and $\cdot \otimes \cdot$ as follows:

$$\text{(input)} \qquad X, Y ::= A \mid X \cdot \otimes \cdot Y \mid X \cdot \oslash \cdot P \mid P \cdot \otimes \cdot X$$

$$\text{(output)} \qquad P, Q ::= A \mid P \cdot \oplus \cdot Q \mid P \cdot / \cdot X \mid X \cdot \backslash \cdot P$$

The *sequents* of the calculus are of the form $X \to P$, and as usual we write $\vdash_{LG_{\text{PF}}} X \to P$ to indicate that the sequent $X \to P$ is derivable in the product-free fragment of **LG**. The axioms and inference rules are presented in Figure 1, where we use the *display logic* from (Goré, 1998), but with different symbols for the *structural connectives*. For clarity, we note that the structural product is often written as a comma, or omitted entirely.

It has been proven by Moortgat (2007) that we have *Cut admissibility* for **LG**. This means that for every derivation using the *Cut*-rule, there exists a corresponding derivation that is *cut-free*. Therefore we will assume that the Cut-rule is not needed anywhere in a derivation.

3 Preliminaries

3.1 Derivation Length

We will first show that for every derivable sequent there exists a cut-free derivation that is polynomial in the length of the sequent. The length of a sequent φ, denoted as $|\varphi|$, is defined as the number of (formula and structural) connectives used to construct this sequent. A subscript will be used to indicate that we count only certain connectives, for example $|\varphi|_{\oslash}$.

$$\frac{}{p \to p}\ Ax \qquad \frac{X \to A \quad A \to P}{X \to P}\ Cut$$

$$\frac{Y \to X \cdot \backslash \cdot P}{X \cdot \otimes \cdot Y \to P}\ r \qquad \frac{X \cdot \oslash \cdot Q \to P}{X \to P \cdot \oplus \cdot Q}\ dr$$

$$\frac{}{X \to P \cdot / \cdot Y}\ r \qquad \frac{}{P \cdot \otimes \cdot X \to Q}\ dr$$

(a) Display rules

$$\frac{X \cdot \otimes \cdot Y \to P \cdot \oplus \cdot Q}{X \cdot \oslash \cdot Q \to P \cdot / \cdot Y}\ d \oslash /$$

$$\frac{X \cdot \otimes \cdot Y \to P \cdot \oplus \cdot Q}{Y \cdot \oslash \cdot Q \to X \cdot \backslash \cdot P}\ d \oslash \backslash$$

$$\frac{X \cdot \otimes \cdot Y \to P \cdot \oplus \cdot Q}{P \cdot \otimes \cdot X \to Q \cdot / \cdot Y}\ d \otimes /$$

$$\frac{X \cdot \otimes \cdot Y \to P \cdot \oplus \cdot Q}{P \cdot \otimes \cdot Y \to X \cdot \backslash \cdot Q}\ d \otimes \backslash$$

(b) Distributivity rules (Grishin's interaction principles)

$$\frac{X \to A \cdot / \cdot B}{X \to A/B}\ /R \qquad \frac{B \cdot \otimes \cdot A \to P}{B \otimes A \to P}\ \otimes L$$

$$\frac{X \to B \cdot \backslash \cdot A}{X \to B \backslash A}\ \backslash R \qquad \frac{A \cdot \oslash \cdot B \to P}{A \oslash B \to P}\ \oslash L$$

$$\frac{X \to A \qquad B \to P}{B/A \to P \cdot / \cdot X}\ /L \qquad \frac{X \to B \qquad A \to P}{P \cdot \otimes \cdot X \to A \otimes B}\ \otimes R$$

$$\frac{X \to A \qquad B \to P}{A \backslash B \to X \cdot \backslash \cdot P}\ \backslash L \qquad \frac{X \to B \qquad A \to P}{X \cdot \oslash \cdot P \to B \oslash A}\ \oslash R$$

(c) Logical rules

Fig. 1. The product-free Lambek-Grishin calculus inference rules

Lemma 1. *If* $\vdash_{LG_{PF}} \varphi$ *there exists a derivation with at most* $|\varphi|$ logical rules.

Proof. If $\vdash_{LG_{PF}} \varphi$ then there exists a cut-free derivation for φ. Because every logical rule removes one logical connective and there are no rules that introduce logical connectives, this derivation contains exactly $|\varphi|_{\{/,\backslash,\oslash,\otimes\}}$ logical rules. □

Lemma 2. *If* $\vdash_{LG_{PF}} \varphi$ *there exists a derivation with at most* $\frac{1}{4}|\varphi|^2$ Grishin interactions.

Proof. Let us take a look at the Grishin interaction principles. First of all, it is not hard to see that the interactions are irreversible. Also note that the interactions happen between the families of input connectives $\{\cdot \otimes \cdot, \cdot/\cdot, \cdot\backslash\cdot\}$ and output

connectives $\{\cdot \oplus \cdot, \cdot \oslash \cdot, \cdot \obslash \cdot\}$ and that the Grishin interaction principles are the only rules of inference that apply to both families. So, to any pair of input and output connectives, at most one Grishin interaction principle can be applied.

If $\vdash_{LG_{\text{PF}}} \varphi$, let Π be a cut-free derivation of φ. The maximum number of possible Grishin interactions in Π is reached when a Grishin interaction is applied on every pair of one input and one output connective. Thus, the maximum number of Grishin interactions in Π is $|\varphi|_{\{\cdot\otimes\cdot,\cdot/\cdot,\cdot\backslash\cdot\}} \cdot |\varphi|_{\{\cdot\oplus\cdot,\cdot\oslash\cdot,\cdot\obslash\cdot\}}$.

By definition, $|\varphi|_{\{\cdot\otimes\cdot,\cdot/\cdot,\cdot\backslash\cdot\}} + |\varphi|_{\{\cdot\oplus\cdot,\cdot\oslash\cdot,\cdot\obslash\cdot\}} = |\varphi|$, so the maximum value of $|\varphi|_{\{\cdot\otimes\cdot,\cdot/\cdot,\cdot\backslash\cdot\}} \cdot |\varphi|_{\{\cdot\oplus\cdot,\cdot\oslash\cdot,\cdot\obslash\cdot\}}$ is reached when $|\varphi|_{\{\cdot\otimes\cdot,\cdot/\cdot,\cdot\backslash\cdot\}} = |\varphi|_{\{\cdot\oplus\cdot,\cdot\oslash\cdot,\cdot\obslash\cdot\}} = \frac{|\varphi|}{2}$. Then the total number of Grishin interactions in Π is $\frac{|\varphi|}{2} \cdot \frac{|\varphi|}{2} = \frac{1}{4}|\varphi|^2$, so any cut-free derivation of φ will contain at most $\frac{1}{4}|\varphi|^2$ Grishin interactions. □

Lemma 3. *In a derivation of sequent φ at most $2|\varphi|$ display rules are needed to display any of the structural parts.*

Proof. A structural part in sequent φ is nested under at most $|\varphi|$ structural connectives. For each of these connectives, one or two r or dr rules can display the desired part, after which the next connective is visible. Thus, at most $2|\varphi|$ display rules are needed to display any of the structural parts.

Lemma 4. *If $\vdash_{LG_{\text{PF}}} \varphi$ there exists a cut-free derivation of length $O(|\varphi|^3)$.*

Proof. From Lemma 1 and Lemma 2 we know that there exists a derivation with at most $|\varphi|$ logical rules and $\frac{1}{4}|\varphi|^2$ Grishin interactions. Thus, the derivation consists of $|\varphi| + \frac{1}{4}|\varphi|^2$ rules, with between each pair of consecutive rules the display rules. From Lemma 3 we know that at most $2|\varphi|$ display rules are needed to display any of the structural parts. So, at most $2|\varphi| \cdot (|\varphi| + \frac{1}{4}|\varphi|^2) = 2|\varphi|^2 + \frac{1}{2}|\varphi|^3$ derivation steps are needed in the shortest possible cut-free derivation for this sequent, and this is in $O(|\varphi|^3)$. □

3.2 Additional Notations

Let us first introduce some additional notations to make the proofs shorter and more readable.

Let us call an input structure X which does not contain any structural connectives except for $\cdot \otimes \cdot$ a *$\otimes$-structure*. A $\otimes$-structure can be seen as a binary tree with $\cdot \otimes \cdot$ in the internal nodes and formulas in the leafs. Formally we define $\otimes$-structures U and V as:

$$U, V ::= A \mid U \cdot \otimes \cdot V$$

We define $X[]$ and $P[]$ as the input and output structures X and P with a hole in one of their leafs. Formally:

$$X[] ::= [] \mid X[] \cdot \otimes \cdot Y \mid Y \cdot \otimes \cdot X[] \mid X[] \cdot \oslash \cdot Q \mid Y \cdot \oslash \cdot P[] \mid Q \cdot \obslash \cdot X[] \mid P[] \cdot \obslash \cdot Y$$

$$P[] ::= [] \mid P[] \cdot \oplus \cdot Q \mid Q \cdot \oplus \cdot P[] \mid P[] \cdot / \cdot Y \mid Q \cdot / \cdot X[] \mid Y \cdot \backslash \cdot P[] \mid X[] \cdot \backslash \cdot Q$$

This notation is similar to the one of de Groote (1999) but with structures. If $X[]$ is a structure with a hole, we write $X[Y]$ for $X[]$ with its hole filled with structure Y. We will write $X^{\otimes}[]$ for a $\otimes$-structure with a hole.

Furthermore, we extend the definition of hole to formulas, and define $A[]$ as a *formula* A with a hole in it, similarly to what we did for structures. Hence, by $A[B]$ we mean the formula $A[]$ with its hole filled by formula B.

In order to distinguish between input and output polarity formulas, we write $A^{\bullet}$ for a formula with *input* polarity and A° for a formula with *output* polarity. The polarity indicates whether a formula can be displayed on the lefthand-side of the turnstyle (input) or the righthand-side (output). Note that for structures this is already implicitly defined by using X and Y for input polarity and P and Q for output polarity. This can be extended to formulas in a similar way, and we will use this notation only in cases where the polarity is not clear from the context.

3.3 Derived Rules of Inference

Now we will show and prove some derived rules of inference of the product-free fragment of **LG**.

If $\vdash_{LG_{\text{PF}}} A \to B$ and we want to derive $X^{\otimes}[A] \to P$, we can *replace* A by B in $X^{\otimes}[]$.

Lemma 5. *We have the inference rule below:*

$$\frac{A \to B \quad X^{\otimes}[B] \to P}{X^{\otimes}[A] \to P}\ \mathit{Repl}$$

Proof. We consider three cases:

1. If $X^{\otimes}[A] = A$, it is simply the cut-rule:

$$\frac{A \to B \quad B \to P}{A \to P}\ \mathit{Cut}$$

2. If $X^{\otimes}[A] = Y^{\otimes}[A] \cdot \otimes \cdot V$, we can move V to the righthand-side and use induction to prove the sequent:

$$\frac{A \to B \quad \dfrac{Y^{\otimes}[B] \cdot \otimes \cdot V \to P}{Y^{\otimes}[B] \to P \cdot / \cdot V}\ r}{\dfrac{Y^{\otimes}[A] \to P \cdot / \cdot V}{Y^{\otimes}[A] \cdot \otimes \cdot V \to P}\ r}\ \mathit{Repl}$$

3. If $X^{\otimes}[A] = U \cdot \otimes \cdot Y^{\otimes}[A]$, we can move U to the righthand-side and use induction to prove the sequent:

$$\frac{A \to B \quad \dfrac{U \cdot \otimes \cdot Y^{\otimes}[B] \to P}{Y^{\otimes}[B] \to U \cdot \backslash \cdot P}\ r}{\dfrac{Y^{\otimes}[A] \to U \cdot \backslash \cdot P}{U \cdot \otimes \cdot Y^{\otimes}[A] \to P}\ r}\ \mathit{Repl}$$

□

If we want to derive $X^{\otimes}[A \oslash B] \to P$, then we can *move* the expression $\oslash B$ out of the $\otimes$-structure.

Lemma 6. *We have the inference rule below:*

$$\frac{X^{\otimes}[A] \cdot \oslash \cdot B \to P}{X^{\otimes}[A \oslash B] \to P}\ Move$$

Proof. Let $C = A \oslash B$. We consider three cases:

1. If $X^{\otimes}[C] = C$, then this is simply the $\oslash L$-rule:

$$\frac{A \cdot \oslash \cdot B \to Y}{C \to Y}\ \oslash L$$

2. If $X^{\otimes}[C] = Y^{\otimes}[C] \cdot \otimes \cdot V$, we can move V to the righthand-side and use induction together with the Grishin interaction principles to prove the sequent:

$$\cfrac{\cfrac{\cfrac{\cfrac{(Y^{\otimes}[A] \cdot \otimes \cdot V) \cdot \oslash \cdot B \to P}{Y^{\otimes}[A] \cdot \otimes \cdot V \to P \cdot \oplus \cdot B}\ dr}{Y^{\otimes}[A] \cdot \oslash \cdot B \to P \cdot / \cdot V}\ d\oslash /}{Y^{\otimes}[C] \to P \cdot / \cdot V}\ Move}{Y^{\otimes}[C] \cdot \otimes \cdot V \to P}\ r$$

3. If $X^{\otimes}[C] = U \cdot \otimes \cdot Y^{\otimes}[C]$, we can move U to the righthand-side and use induction together with the Grishin interaction principles to prove the sequent:

$$\cfrac{\cfrac{\cfrac{\cfrac{(U \cdot \otimes \cdot Y^{\otimes}[A]) \cdot \oslash \cdot B \to P}{U \cdot \otimes \cdot Y^{\otimes}[A] \to P \cdot \oplus \cdot B}\ dr}{Y^{\otimes}[A] \cdot \oslash \cdot B \to U \cdot \backslash \cdot P}\ d\oslash \backslash}{Y^{\otimes}[C] \to U \cdot \backslash \cdot P}\ Move}{U \cdot \otimes \cdot Y^{\otimes}[C] \to P}\ r$$

□

3.4 Type Similarity

The type simililarity relation $\sim$, introduced by Lambek (1958), is the reflexive transitive symmetric closure of the derivability relation. Formally we define this as:

Definition 1. $A \sim B$ iff *there exists a sequence* $C_1 \ldots C_n (1 \leq i \leq n)$ *such that* $C_1 = A$, $C_n = B$ *and* $C_i \to C_{i+1}$ *or* $C_{i+1} \to C_i$ *for all* $1 \leq i < n$.

It was proved by Lambek for **L** that $A \sim B$ *iff* one of the following equivalent statements holds (the so-called *diamond property*):

$$\exists C \text{ such that } A \to C \text{ and } B \to C \quad \text{(join)}$$

$$\exists D \text{ such that } D \to A \text{ and } D \to B \quad \text{(meet)}$$

Although this diamond-property also holds for **LG**, this is unfortunately not true for the product-free case. However, in this paper we will use a property that is closely related to the diamond-property to create a choice for the truthvalue of a variable.

Definition 2. *If $A \sim B$ and C is the* join *type of A and B that is $A \to C$ and $B \to C$, we define $A \overset{C}{\sqcap} B = (A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B)$ as the meet type of A and B.*

The formula equivalent of this structure is also the solution given by Lambek (1958) for the associative system **L**, which is in fact the shortest solution for the non-associative system **NL** (Foret, 2003).

Lemma 7. *If $A \sim B$ with join-type C and $\vdash_{LG_{PF}} A \to P$ or $\vdash_{LG_{PF}} B \to P$, then we also have $\vdash_{LG_{PF}} A \overset{C}{\sqcap} B \to P$. We can write this as a derived rule of inference:*

$$\frac{A \to P \quad or \quad B \to P}{A \overset{C}{\sqcap} B \to P}\ Meet$$

Proof

1. If $A \to P$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \to C \quad C \to C}{C/C \to C \cdot / \cdot C}\,/L}{C/C \to C/C}\,/R \qquad B \to C}{(C/C)\backslash B \to (C/C) \cdot \backslash \cdot C}\,\backslash L}{(C/C)\backslash B \to (C/C)\backslash C}\,\backslash R \qquad A \to P}{A/((C/C)\backslash C) \to P \cdot / \cdot ((C/C)\backslash B)}\,/L}{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P}\,r$$

2. If $B \to P$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{C \to C \quad C \to C}{C/C \to C \cdot / \cdot C}\,/L}{(C/C) \cdot \otimes \cdot C \to C}\,r}{C \to (C/C) \cdot \backslash \cdot C}\,r \qquad C \to (C/C)\backslash C}{A \to C}\,\backslash R}{A/((C/C)\backslash C) \to C \cdot / \cdot C}\,/L}{A/((C/C)\backslash C) \to C/C}\,/R \qquad B \to P}{(C/C)\backslash B \to (A/((C/C)\backslash C)) \cdot \backslash \cdot P}\,\backslash L}{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P}\,r$$

□

The following lemma is the key lemma of this paper, and its use will become clear to the reader in the construction of Section 4. For an intuition of its use, refer to subsection 4.1.

Lemma 8. *If* $\vdash_{LG_{\text{PF}}} A \overset{C}{\sqcap} B \to P$ *then* $\vdash_{LG_{\text{PF}}} A \to P$ *or* $\vdash_{LG_{\text{PF}}} B \to P$.

Proof. By case analysis on the derivation we will show that *if* $\vdash_{LG_{\text{PF}}} (A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P$, *then* $\vdash_{LG_{\text{PF}}} A \to P$ or $\vdash_{LG_{\text{PF}}} B \to P$. We will look at the derivations in a top-down way, and we will do case analysis on the rules that are applied to the sequent. We will show that in all cases we could change the derivation in such way that the meet-type $A \overset{C}{\sqcap} B$ in the conclusion could have immediately been replaced by either A or B.

The first case is where a logical rule is applied on the lefthand-side of the sequent. At a certain point in the derivation, possibly when P is an atom, one of the following two rules must be applied:

1. The $/L$ rule, in this case first the r rule is applied so that we have $\vdash_{LG_{\text{PF}}} A/((C/C)\backslash C) \to P \cdot / \cdot ((C/C)\backslash B)$. Now if the $/L$ rule is applied, we must have that $\vdash_{LG_{\text{PF}}} A \to P$.
2. The $\backslash L$ rule, in this case first the r rule is applied so that we have $\vdash_{LG_{\text{PF}}} (C/C)\backslash B \to (A/((C/C)\backslash C)) \cdot \backslash \cdot P$. Now if the $\backslash L$ rule is applied, we must have that $\vdash_{LG_{\text{PF}}} B \to P$.

The second case happens when a logical rule is applied on the righthand-side of the sequent. Let $\delta = \{r, dr, d\oslash/, d\oslash\backslash, d\oslash/, d\oslash\backslash\}$ and let δ^* indicate a (possibly empty) sequence of structural residuation steps and Grishin interactions. For example for the $\oslash R$ rule there are two possibilities:

– The lefthand-side ends up in the first premisse of the $\oslash R$ rule:

$$\dfrac{\dfrac{\dfrac{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P''[A']}{P'[(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B)] \to A'}\ \delta^* \qquad B' \to Q}{P'[(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B)] \cdot \oslash \cdot Q \to A' \oslash B'}\ \oslash R}{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P[A' \oslash B']}\ \delta^*$$

In order to be able to apply the $\oslash R$ rule, we need to have a formula of the form $A' \oslash B'$ on the righthand-side. In the first step all structural rules are applied to display this formula in the righthand-side, and we assume that in the lefthand-side the meet-type ends up in the first structural part (inside a structure with the remaining parts from P that we call P'). After the $\oslash R$ rule has been applied, we can again display our meet-type in the lefthand-side of the formula by moving all other structural parts from P' back to the righthand-side (P'').

In this case it must be that $\vdash_{LG_{\text{PF}}} (A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P''[A']$, so from this lemma we know that in this case also $\vdash_{LG_{\text{PF}}} A \to P''[A']$ or $\vdash_{LG_{\text{PF}}} B \to P''[A']$. In the case that $\vdash_{LG_{\text{PF}}} A \to P''[A']$, we can show that $\vdash_{LG_{\text{PF}}} A \to P[A' \oslash B']$ as follows:

$$\dfrac{\dfrac{\dfrac{A \to P''[A']}{P'[A] \to A'}\ \delta^* \qquad B' \to Q}{P'[A] \cdot \oslash \cdot Q \to A' \oslash B'}\ \oslash R}{A \to P[A' \oslash B']}\ \delta^*$$

The case for B is similar.

– The lefthand-side ends up in the second premisse of the $\oslash R$ rule:

$$\dfrac{\dfrac{Q \to A' \quad \dfrac{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P''[B']}{B' \to P'[(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B)]}\,\delta^*}{Q \cdot \oslash \cdot P'[(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B)] \to A' \oslash B'}\,\oslash R}{(A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P[A' \oslash B']}\,\delta^*$$

This case is similar to the other case, except that the meet-type ends up in the other premisse. Note that, although in this case it is temporarily moved to the righthand-side, the meet-type will still be in an input polarity position and can therefore be displayed in the lefthand-side again.

In this case it must be that $\vdash_{LG_{\text{PF}}} (A/((C/C)\backslash C)) \cdot \otimes \cdot ((C/C)\backslash B) \to P''[B']$, and from this lemma we know that in this case also $\vdash_{LG_{\text{PF}}} A \to P''[B']$ or $\vdash_{LG_{\text{PF}}} B \to P''[B']$. In the case that $\vdash_{LG_{\text{PF}}} A \to P''[B']$, we can show that $\vdash_{LG_{\text{PF}}} A \to P[A' \oslash B']$ as follows:

$$\dfrac{\dfrac{Q \to A' \quad \dfrac{A \to P''[B']}{B' \to P'[A]}\,\delta^*}{Q \cdot \oslash \cdot P'[A] \to A' \oslash B'}\,\oslash R}{A \to P[A' \oslash B']}\,\delta^*$$

The case for B is similar.

The cases for the other logical rules are similar. □

4 Reduction from SAT to LG

For the product-free fragment of **L** it has been proven by Savateev (2009) that the derivability problem is also NP-complete. This is a remarkable result that does not follow directly from the fact that the derivability problem for **L** itself is NP-complete. It is known that removing the product restricts the calculus in an essential way, for example the diamond property does not hold in its original form in the product-free fragment (Pentus, 1993a). As the diamond property was used in the original proof by Pentus (2003), it could have been the case that the reduction from SAT was not possible in the product-free fragment. However, as Savateev proved this is not the case, and the product-free fragment of **L** is also NP-complete.

In this section we will show that we can reduce a Boolean formula in conjunctive normal form to a sequent of the *product-free* fragment of the Lambek-Grishin calculus. The size of the resulting sequent will be polynomial in the size of the Boolean formula. Let $\varphi = c_1 \wedge \ldots \wedge c_n$ be a Boolean formula in conjunctive normal form with clauses $c_1 \ldots c_n$ and variables $x_1 \ldots x_m$ and for all $1 \leq j \leq m$ let $\neg_0 x_j$ stand for the literal $\neg x_j$ and $\neg_1 x_j$ stand for the literal x_j.

Let p_i (for $1 \leq i \leq n$) be distinct primitive types from *Var*. We define the following families of types:

$$E^i_j(t) \leftrightharpoons \begin{cases} p_i/(p_i \backslash p_i) & \text{if } \neg_t x_j \text{ appears in clause } c_i \\ p_i & \text{otherwise} \end{cases} \quad \begin{array}{l} 1 \leq i \leq n,\ 1 \leq j \leq m \\ \text{and } t \in \{0,1\} \end{array}$$

$$E_j(t) \leftrightharpoons ((((s/s) \oslash E^1_j(t)) \oslash \ldots) \oslash E^{n-1}_j(t)) \oslash E^n_j(t) \quad 1 \leq j \leq m \text{ and } t \in \{0,1\}$$

$$H \leftrightharpoons (((((s/s) \oslash p_1) \oslash p_2) \oslash \ldots) \oslash p_{n-1}) \oslash p_n$$

$$F^0_j \leftrightharpoons E_j(0)/((H/H)\backslash H) \quad 1 \leq j \leq m$$

$$F^1_j \leftrightharpoons (H/H)\backslash E_j(1) \quad 1 \leq j \leq m$$

$$G^0_m \leftrightharpoons s$$

$$G^i_0 \leftrightharpoons G^{i-1}_m \quad 1 \leq i \leq n$$

$$G^i_j \leftrightharpoons G^i_{j-1} \oslash p_i \quad 1 \leq i \leq n \text{ and } 1 \leq j < m$$

$$G^i_m \leftrightharpoons G^i_{m-1} \oslash (p_i/(p_i \backslash p_i)) \quad 1 \leq i \leq n$$

Let $\bar{\varphi} = (F^0_1 \cdot \otimes \cdot F^1_1) \cdot \otimes \cdot (\ldots ((F^0_{m-1} \cdot \otimes \cdot F^1_{m-1}) \cdot \otimes \cdot ((F^0_m \cdot \otimes \cdot F^1_m) \cdot \otimes \cdot s))) \to G^n_m$ be the product-free **LG** sequent[1] corresponding to the Boolean formula φ. We now claim that $\vDash \varphi$ *if and only if* $\vdash_{LG_{\text{PF}}} \bar{\varphi}$.

4.1 Intuition

For each variable we use a meet-type to create a choice for its truthvalue. Remark that if we have $\vdash_{LG_{\text{PF}}} A \to C$ and $\vdash_{LG_{\text{PF}}} B \to C$ then, for $D_1 = A/((C/C)\backslash C)$ and $D_2 = (C/C)\backslash B$, we have $\vdash_{LG_{\text{PF}}} D_1 \cdot \otimes \cdot D_2 \to A$ and $\vdash_{LG_{\text{PF}}} D_1 \cdot \otimes \cdot D_2 \to B$.

In the construction we now have F^0_j and F^1_j as parts of the meet-type. We can replace each of these by either $E_j(1)$ or $E_j(0)$, so that we have $E_1(t_1) \cdot \otimes \cdot (E_2(t_2) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s)))) \to G^n_m$ for some $\langle t_1, t_2 \ldots t_{m-1}, t_m \rangle \in \{0,1\}^m$.

Let us call a formula of the form $((((A \oslash B_1) \oslash B_2) \oslash \ldots) \oslash B_{n-1}) \oslash B_n$ a $\oslash$-*stack* with n items, and let us call B_n the topmost item of this stack. A is considered to be the base of this $\oslash$-stack.

The sequent $E_1(t_1) \cdot \otimes \cdot (E_2(t_2) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s)))) \to G^n_m$ consists of m $\oslash$-stacks in the lefthand-side $(E_1(t_1) \ldots E_m(t_m))$, each containing one item for every clause. On the righthand-side there is also a $\oslash$-stack, with $n \cdot m$ items. Because of the logical $\oslash$ connective on the righthand-side, there has to be a matching item on the lefthand-side for every item from the righthand-side. By means of the Grishin interactions, every topmost item from a $\oslash$-stack in the lefthand-side can directly match with the item on the righthand-side, as the lefthand-side is an $\otimes$-structure.

Now the $\oslash$-stack on the righthand-side will be in order of decreasing clause number. For each clause it first contains the type $p_i/(p_i \backslash p_i)$, so in the lefthand-side there has to be at least one item $p_i/(p_i \backslash p_i)$ (meaning that there is a variable satisfying clause i). Then, there are $m-1$ types p_i to remove the rest of the p_i or $p_i/(p_i \backslash p_i)$ in the lefthand-side. Note that $\vdash_{LG_{\text{PF}}} p_i \to p_i/(p_i \backslash p_i)$.

[1] Note that the products in this formula are only structural products, which are usually omitted.

Finally, if all items from the stacks could be removed, there was an assignment with for each clause a variable satisfying it, so we are left with $(s/s)\cdot\otimes\cdot((s/s)\cdot\otimes\cdot...((s/s)\cdot\otimes\cdot s))\rightarrow s$ which is obviously derivable. If it was not satisfyable, there was at least 1 index i for which there was no $p_i/(p_i\backslash p_i)$ in the lefthand-side, so the sequent is not derivable.

4.2 Only-If Part

We will now prove that *if* $\vDash\varphi$, *then* $\vdash_{LG_{\text{PF}}}\bar{\varphi}$. We now assume that $\vDash\varphi$, so there is an assignment $\langle t_1,\ldots,t_m\rangle\in\{0,1\}^m$ that satisfies φ.

Lemma 9. *For* $1\leq i\leq n$, $1\leq j\leq m$ *and* $t\in\{0,1\}$ *we have* $\vdash_{LG_{\text{PF}}}p_i\rightarrow E^i_j(t)$.

Proof. We consider two cases:

1. If $E^i_j(t)=p_i$ this is simply the axiom rule.
2. If $E^i_j(t)=p_i/(p_i\backslash p_i)$ we can prove it as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{p_i\rightarrow p_i\quad p_i\rightarrow p_i}{p_i\backslash p_i\rightarrow p_i\cdot\backslash\cdot p_i}\,\backslash L}{p_i\cdot\otimes\cdot(p_i\backslash p_i)\rightarrow p_i}\,r}{p_i\rightarrow p_i\cdot/\cdot(p_i\backslash p_i)}\,r}{p_i\rightarrow p_i/(p_i\backslash p_i)}\,/R$$

□

Lemma 10. *For* $1\leq j\leq m$ *and* $t\in\{0,1\}$ *we have* $\vdash_{LG_{\text{PF}}}E_j(t)\rightarrow H$.

Proof. We can apply the $\oslash R$ rule n times together with Lemma 9 to prove this. □

Lemma 11. *For* $1\leq j\leq m$ *we have* $\vdash_{LG_{\text{PF}}}F^0_j\cdot\otimes\cdot F^1_j\rightarrow E_j(t_j)$.

Proof. From Lemma 10 we know that H_j is the join type of $E_j(0)$ and $E_j(1)$. Notice that $F^0_j\cdot\otimes\cdot F^1_j=E_j(0)\overset{H}{\sqcap}E_j(1)$, so from Lemma 7 we know that in this case also $\vdash_{LG_{\text{PF}}}F^0_j\cdot\otimes\cdot F^1_j\rightarrow E_j(t_j)$. □

We can replace each $F^0_j\cdot\otimes\cdot F^1_j$ in $\bar{\varphi}$ by $E_j(t_j)$.

Lemma 12. *If* $\vdash_{LG_{\text{PF}}}E_1(t_1)\cdot\otimes\cdot(\ldots(E_{m-1}(t_{m-1})\cdot\otimes\cdot(E_m(t_m)\cdot\otimes\cdot s)))\rightarrow G^n_m$ *then* $\vdash_{LG_{\text{PF}}}(F^0_1\cdot\otimes\cdot F^1_1)\cdot\otimes\cdot(\ldots((F^0_{m-1}\cdot\otimes\cdot F^1_{m-1})\cdot\otimes\cdot((F^0_m\cdot\otimes\cdot F^1_m)\cdot\otimes\cdot s)))\rightarrow G^n_m$.

Proof. This can be proven by applying Lemma 11 in combination with Lemma 5 m times. □

Lemma 13. *In* $E_1(t_1)\cdot\otimes\cdot(\ldots(E_{m-1}(t_{m-1})\cdot\otimes\cdot(E_m(t_m)\cdot\otimes\cdot s)))\rightarrow G^n_m$, *there is at least one occurrence of the expression* $\oslash(p_i/(p_i\backslash p_i))$ *in the lefthand-side for every* $1\leq i\leq n$.

Proof. This sequence of $E_1(t_1), \ldots, E_m(t_m)$ represent the truthvalue of all variables, and because this is a satisfying assignment, for all i there is at least one index k such that $\neg_{t_k} x_k$ appears in clause i. By definition we have that $E^i_k(t_k) = p_i/(p_i \backslash p_i)$. □

Lemma 14. $\vdash_{LG_{PF}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \rightarrow G^n_m$.

Proof. We prove this by induction on the length of G^n_m. By definition we have that $G^n_m = G^n_{m-1} \oslash (p_n/(p_n \backslash p_n))$, and from Lemma 13 we know that the expression $\oslash(p_n/(p_n \backslash p_n))$ occurs on the lefthand-side as outermost part of some $E_k(t_k)$, so by using Lemma 6 we can move this expression to the outside of the lefthand-side after which we can apply the $\oslash R$ rule.

Now on the righthand-side we have G^n_{m-1}, which consists of G^{n-1}_m surrounded by $m-1$ occurrences of the expression $\oslash p_n$. In the lefthand-side there are $m-1$ occurrences of $\oslash E^n_j(t_j)$, for every $1 \leq j \leq m$ $(j \neq k)$. Using the fact from Lemma 9 that $\vdash_{LG_{PF}} E^n_j(t_j) \rightarrow p_n$, we can again use Lemma 6 and the $\oslash R$ rule to remove all these expressions from the left- and righthand-side.

The sequent that remains is of exactly the same form, but for $n-1$ instead of n clauses. The same reasoning applies on this sequent, so we can repeat this process n times. Then, the $\oslash R$ rule has been applied $n \cdot m$ times in total, and the sequent will be of the form $(s/s) \cdot \otimes \cdot ((s/s) \cdot \otimes \cdot \ldots ((s/s) \cdot \otimes \cdot s))) \rightarrow s$. This can easily be derived, so it must be the case that also $\vdash_{LG_{PF}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \rightarrow G^n_m$. □

Lemma 15. *If* $\vDash \varphi$*, then* $\vdash_{LG_{PF}} \bar{\varphi}$*,*

Proof. From Lemma 14 we know that $\vdash_{LG_{PF}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \rightarrow G^n_m$, and using Lemma 12 we know that in this case also $\vdash_{LG_{PF}} (F^0_1 \cdot \otimes \cdot F^1_1) \cdot \otimes \cdot (\ldots ((F^0_{m-1} \cdot \otimes \cdot F^1_{m-1}) \cdot \otimes \cdot ((F^0_m \cdot \otimes \cdot F^1_m) \cdot \otimes \cdot s))) \rightarrow G^n_m$. □

4.3 If Part

For the if part we will need to prove that *if* $\vdash_{LG_{PF}} \bar{\varphi}$, *then* $\vDash \varphi$. Let us now assume that $\vdash_{LG_{PF}} \bar{\varphi}$.

Lemma 16. *If* $\vdash_{LG_{PF}} X \rightarrow P \oslash Y$*, then there exist a* Q *such that* Q *is part of* X *(possibly inside a formula in* X*) and* $\vdash_{LG_{PF}} Y \rightarrow Q$*.*

Proof. The only rule that matches a $\oslash$ in the righthand-side is the $\oslash R$ rule. Because this rule needs a $\cdot \oslash \cdot$ connective in the lefthand-side, we know that if $\vdash_{LG_{PF}} X \rightarrow P \oslash Y$ it must be the case that we can turn X into $X' \cdot \oslash \cdot Q$ such that $\vdash_{LG_{PF}} X' \rightarrow P$ and $\vdash_{LG_{PF}} Y \rightarrow Q$. □

Lemma 17. *If* $\vdash_{LG_{PF}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \rightarrow G^n_m$*, then there is an occurrence of* $p_i/(p_i \backslash p_i)$ *on the lefthand-side at least once for all* $1 \leq i \leq n$*.*

Proof. G^n_m by definition contains an occurrence of the expression $\oslash(p_i/(p_i\backslash p_i))$ for all $1 \leq i \leq n$. From Lemma 16 we know that somewhere in the lefthand-side we need an occurrence of a structure Q such that $\vdash_{LG_{\text{PF}}} p_i/(p_i\backslash p_i) \to Q$. From the construction it is obvious that the only possible type for Q is in this case $p_i/(p_i\backslash p_i)$. □

Lemma 18. *If* $\vdash_{LG_{\text{PF}}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \to G^n_m$, *then* $\langle t_1, \ldots, t_{m-1}, t_m \rangle$ *is a satisfying assignment for the CNF formula.*

Proof. From Lemma 17 we know that there is an occurrence of $p_i/(p_i\backslash p_i)$ in the lefthand-side of the formula for all $1 \leq i \leq n$. From the definition we know that for each i there is an index j such that $E^i_j(t_j) = p_i/(p_i\backslash p_i)$, and this means that $\neg_{t_j} x_j$ appears in clause i, so if for every $1 \leq i \leq n$ there is an occurrence of $p_i/(p_i\backslash p_i)$ in the lefthand-side then all clauses are satisfied. Hence, this choice of $t_1 \ldots t_m$ is a satisfying assignment. □

Lemma 19. *For* $1 \leq j \leq m$ *and* $\vdash_{LG_{\text{PF}}} X^{\otimes}[F^0_j \cdot \otimes \cdot F^1_j] \to G^n_m$ *we have* $\vdash_{LG_{\text{PF}}} X^{\otimes}[E_j(0)] \to G^n_m$ *or* $\vdash_{LG_{\text{PF}}} X^{\otimes}[E_j(1)] \to G^n_m$.

Proof. We know that $X^{\otimes}[F^0_j \cdot \otimes \cdot F^1_j]$ is a $\otimes$-structure, so we can apply the r rule several times to move all but the structure $F^0_j \cdot \otimes \cdot F^1_j$ to the righthand-side. We then have that $\vdash_{LG_{\text{PF}}} F^0_j \cdot \otimes \cdot F^1_j \to \ldots \cdot \backslash \cdot G^n_m \cdot / \cdot \ldots$. As remarked in Lemma 11 this is exactly the meet-type from Lemma 8 with a structural $\cdot \otimes \cdot$, so we know that now have that $\vdash_{LG_{\text{PF}}} E_j(0) \to \ldots \cdot \backslash \cdot G^n_m \cdot / \cdot \ldots$ or $\vdash_{LG_{\text{PF}}} E_j(1) \to \ldots \cdot \backslash \cdot G^n_m \cdot / \cdot \ldots$. Finally we can apply the r rule again to move all parts back to the lefthand-side, to show that $\vdash_{LG_{\text{PF}}} X^{\otimes}[E_j(0)] \to G^n_m$ or $\vdash_{LG_{\text{PF}}} X^{\otimes}[E_j(1)] \to G^n_m$.

Note that $F^0_j \cdot \otimes \cdot F^1_j$ provides an *explicit switch*, which means that the truth-value of a variable can only be changed in all clauses simultanously. □

Lemma 20. *If* $\vdash_{LG_{\text{PF}}} \bar{\varphi}$, *then* $\vDash \varphi$.

Proof. By Lemma 19, if $\vdash_{LG_{\text{PF}}} (F^0_1 \cdot \otimes \cdot F^1_1) \cdot \otimes \cdot (\ldots ((F^0_{m-1} \cdot \otimes \cdot F^1_{m-1}) \cdot \otimes \cdot ((F^0_m \cdot \otimes \cdot F^1_m) \cdot \otimes \cdot s))) \to G^n_m$, then also $\vdash_{LG_{\text{PF}}} E_1(t_1) \cdot \otimes \cdot (\ldots (E_{m-1}(t_{m-1}) \cdot \otimes \cdot (E_m(t_m) \cdot \otimes \cdot s))) \to G^n_m$ for some $\langle t_1, \ldots, t_{m-1}, t_m \rangle \in \{0,1\}^m$. From Lemma 18 we know that this is a satisfying assignment for φ, so if we assume that $\vdash_{LG_{\text{PF}}} \bar{\varphi}$, then $\vDash \varphi$. □

4.4 Conclusion

Theorem 1. *The product-free fragment of* ***LG*** *is NP-complete.*

Proof. From Lemma 4 we know that for every derivable sequent there exists a derivation that is of polynomial length, so the derivability problem for (the product-free fragment of) **LG** is in NP. From Lemma 15 and Lemma 20 we can conclude that we can reduce SAT to the product-free fragment of **LG**. Because SAT is a known NP-hard problem (Garey and Johnson, 1979), and our reduction is polynomial, we can conclude that derivability for the product-free fragment **LG** is also NP-hard.

Combining these two facts we conclude that the derivability problem for the product-free fragment of **LG**, like the derivability problem for **LG**, is NP-complete. □

5 Discussion and Future Work

In this paper we have shown the NP-completeness of the product-free fragment of **LG**. Does this result disqualify **LG** as a formalism for practical use in computational linguistics? We don't think this conclusion necessarily follows.

Pentus (2010) shows that the derivability problem for Lambek Grammars, i.e. **L** with a fixed lexicon, and thus bounded order, belongs to `PTIME`. This means that, although the general derivability problem for **L** is NP-complete, for the fixed lexicon situation there is a polynomial algorithm that can be used in practice. Whether a similar situation obtains for **LG** is a question that requires further research.

Here are some preliminary ideas. It is not hard to see that the SAT reduction used in this paper can not be encoded in a (possibly product-free) lexicon for **LG**, as the number of distinct primitive types already depends on the length of the Boolean formula. It is interesting to notice that the construction from (Melissen, 2009) for encoding the intersection of a context-free grammar with all permutations of another context-free grammar in **LG** still holds for **LG** with a fixed lexicon. Furthermore, as the construction does not use products, it even holds for the product-free fragment of **LG** with a fixed lexicon. This implies that also with a fixed lexicon, the generative power of **LG** lies beyond `LTAG`.

References

Bransen, J.: The Lambek-Grishin calculus is NP-complete. In: Proceedings 15th Conference on Formal Grammar, Copenhagen (2010) (to appear)

de Groote, P.: The Non-associative Lambek Calculus with Product in Polynomial Time. In: Murray, N.V. (ed.) TABLEAUX 1999. LNCS (LNAI), vol. 1617, pp. 128–139. Springer, Heidelberg (1999)

Foret, A.: On the computation of joins for non associative Lambek categorial grammars. In: Proceedings of the 17th International Workshop on Unification (UNIF 2003), Valencia, Spain, June 8-9 (2003)

Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)

Goré, R.: Substructural logics on display. Logic Jnl IGPL 6(3), 451–504 (1998)

Kandulski, M.: The non-associative Lambek calculus. In: Categorial Grammar. Linguistic and Literary Studies in Eastern Europe (LLSEE), vol. 25, pp. 141–151 (1988)

Lambek, J.: The Mathematics of Sentence Structure. American Mathematical Monthly 65, 154–170 (1958)

Lambek, J.: On the calculus of syntactic types. In: Structure of Language and Its Mathematical Aspects, pp. 166–178 (1961)

Melissen, M.: The Generative Capacity of the Lambek–Grishin Calculus: A New Lower Bound. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) Formal Grammar. LNCS, vol. 5591, pp. 118–132. Springer, Heidelberg (2011)

Moortgat, M.: Symmetries in Natural Language Syntax and Semantics: The Lambek-Grishin Calculus. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 264–284. Springer, Heidelberg (2007)

Moortgat, M.: Symmetric categorial grammar. Journal of Philosophical Logic 38(6), 681–710 (2009)

Moot, R.: Proof nets for display logic. CoRR, abs/0711.2444 (2007)

Pentus, M.: The conjoinability relation in Lambek calculus and linear logic. ILLC Prepublication Series ML–93–03, Institute for Logic, Language and Computation, University of Amsterdam (1993a)

Pentus, M.: Lambek grammars are context free. In: Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science, pp. 429–433. IEEE Computer Society Press, Los Alamitos (1993b)

Pentus, M.: Lambek calculus is NP-complete. CUNY Ph.D. Program in Computer Science Technical Report TR–2003005, CUNY Graduate Center, New York (2003)

Pentus, M.: A Polynomial-Time Algorithm for Lambek Grammars of Bounded Order. Linguistic Analysis 36, 441–472 (2010)

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)

Copredication, Quantification and Frames

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 propose a record type theoretical account of cases of copredication which have motivated the introduction of dot types in the Generative Lexicon ([2,1]). We will suggest that using record types gives us a general account of dot types and also makes a connection between copredication and the use of hypothetical contexts in a record type theoretic analysis of dynamic generalized quantifiers. We propose a view of lexical innovation which draws both on Pustejovsky's original work on the Generative Lexicon ([27]) and the notion of resource in [23,15]. We will also address issues relating to counting objects in terms of their aspects which are raised in [1].

Keywords: copredication, generalized quantifiers, type theory, records, frame semantics.

1 Introduction

[2,1] propose a type theoretical approach to account for examples which had motivated Pustejovsky [27] to introduce dot types. The intuitive idea behind dot types is that they are compositions of two types which nevertheless allow the two individual types to be recovered. Consider the examples in (1), based on examples originally given in [27] and discussed extensively in [1].[1]

(1)
a. The lunch was delicious
b. The lunch took forever

delicious is a predicate that is normally used of food but not events. The word "normally" is important here since I think it points to an inherent variability in natural language semantics. I will talk of *innovative* uses of words and phrases and mark them with i as in (2b).

(2)
a. The blancmange was delicious
b. ^{i}IFK Göteborg's last game was delicious

[1] I believe that such examples were first reported in the literature by McCawley ([25]) who discussed examples such as *John has memorized the score of the Ninth Symphony* and *The score of the Ninth Symphony is lying on the piano.*

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 64–79, 2011.

The i in (2b) should not be taken as representing any absolute judgement. A use might be innovative for one or more of the participants in a dialogue on first mention but similar uses subsequently in the same dialogue will presumably not be innovative. If one encounters similar uses in enough other dialogues (in particular with different dialogue participants) then the usage will become part of a speaker's general resources and such uses will no longer be regarded as innovative even the first time they are encountered in a dialogue. The task of a linguistic theory is not so much to determine what is innovative but to determine what mechanisms are available to a speaker in order to produce or interpret a use which is innovative with respect to the current state of their individual linguistic knowledge. The Generative Lexicon plays a central role in this as does the notion of resource developed in [16,22,23,15]. This work points to a shift in our view grammar and semantics, a move from Montague's ([26]) dictum "English as a formal language" to "English as a toolbox for constructing formal languages".

In those cases where (2b) is innovative, Pustejovsky would say that *game* and/or *delicious* are being coerced to another type in order to get an interpretation. Similarly *take forever* is a predicate which normally holds of events and not of food.

(3)
a. ^{i}The blancmange took forever
b. IFK's last game took forever

Again we can certainly get intepretations out of (3a) but it involves some kind of coercion. It is not the actual food that takes forever but something like the eating of it, the preparing of it or waiting for it. Here the coercions are so standard that it might seem that the i is less justified. In the case of *lunch* we do not feel that coercion is necessary. (Coercion readings are, of course, available. *The lunch took forever* can mean, for example, that the preparation of the lunch took forever.) This word seems equally easy to interpret as representing food or as representing an event. Pustejovsky therefore associates it with a dot type *Food·Event*. This is meant to represent that it will simultaneously behave as something of type *Food* and something of type *Event* and accept predicates of food and predicates of events.

A natural view might be that this is simply a case of polysemy, that is, that *lunch* is ambiguous between a food interpretation and an event interpretation. If this were the case it would be hard to see what the motivation of the dot type would be. There would simply be two types one associated with each meaning of the word. However, [2,1] discuss in detail cases of copredication where one occurrence of the word simultaneously has both interpretations. Compare (4a) with (4b) (both examples from Asher and Pustejovsky).

(4)
a. The lunch was delicious but took forever
b. $^{!}$The bank specializes in IPO's and is being quickly eroded by the river

The word *bank* is ambiguous between meaning a financial institution and the ground at the side of a river. It appears that a single occurrence of the word cannot be used in both senses. However, no such problems seem to occur with *lunch*. It seems that in some way natural language manages to treat lunch as a single object which simultaneously has an event and food aspect. Note, however, that it is possible to get the word *bank* to have both meanings simultaneously in examples like (5).

(5) ^{i}Sam went to the wrong bank

This could be used to report a misunderstanding about which kind of bank was being referred to. (The point of jokes very often rests on such ambiguities. See [29] for examples.) I would suggest that for most speakers in normal circumstances such a reading involves an innovative interpretation of *bank* obtained by taking the join or disjunction of two meanings associated with the phonology of *bank* in their current lexical resource. Such readings are, however, not available for *lunch*.

(6) Kim told me about the wrong lunch

(6) cannot be used to say that Kim told me about the event (in which I was not interested) rather than the food, something that might be expressed by (7).

(7) Kim told me the wrong things about the lunch. (I was interested in the food, not the conversation.)

In this paper I will propose an alternative treatment to that given by Asher and Pustejovsky using type theory with records in the variant TTR that I have developed in [7,8,9,13] based on work in Martin-Löf type theory [31,5,6,17]. I will propose (section 2) that records give us a credible model of what Asher ([1]) calls "thick individuals" following Kratzer's ([21]) use of the notion of "thick particular". We will, however, relate our notion to frames ([18,19,30]). I will suggest (section 3) that using record types not only gives us a simple and intuitive account of dot types but also makes an important connection between copredication and the use of hypothetical contexts in dynamic generalized quantifiers proposed in [7]. I will show (section 4) how this enables us to give an account of copredication based on frames. I will suggest (section 5) that the structured objects provided by record types will make it easier to define the kinds of coercions we need in order to handle the kind of innovative uses that we have marked with i in the examples above. Finally (section 6) I will show that this analysis will allow us to give an account of a puzzle concerning how we count the number of books on a shelf which is discussed in [1].

2 Frames and Thick Particulars

In [12,13] we propose the use of frames, modelled as records, as a way of accounting for the Partee puzzle regarding temperature and rising. Here we will

use the same idea to analyze copredication. Consider some particular lunch. As a "thin particular" (see [21]) we might consider it to be of type *Ind*, the type of individuals. One could certainly call into question whether it is an individual in the same sense as physical objects like people or tables. It certainly seems very different. Lunches do have a physical aspect in terms of the food served or eaten. They also have an event aspect which takes place in a space-time location. We will take account of this difference in terms of the "thick particulars" which we call lunches. The standard view of thick particulars is that they are objects together with all their properties. We will, however, use frames which pick out certain apparently cognitively important properties that individuals can have. If you want, you can think of them as "moderately thick particulars". Exactly which objects correspond to such "moderately thick particulars" and exactly how thick they are (i.e. how may fields they contain) we imagine depends on the experience and knowledge of particular agents and what they have come to regard as defining aspects associated with types. We imagine our frames being embedded in the kind of learning theory sketched in [23,15].

For the sake of this paper we will think of lunch-frames as being of the type in (8).

(8)
$$\begin{bmatrix} \text{x} & : \textit{Ind} \\ \text{event} & : \textit{Event} \\ \text{food} & : \textit{Food} \\ \text{c}_{\text{lunch}} & : \text{lunch\_ev\_fd(x, event, food)} \end{bmatrix}$$

A frame (that is, a record) which is of this type will have at least four fields (it may have more) labelled by 'x', 'event', 'food' and 'c_{lunch}'. In both records and record types there will be at most one field with any particular label. For the record to be of the type in (8) the objects in the fields have to have the types indicated in the record type. Thus the field labelled 'x' has to contain an individual, the field labelled 'event' has to contain an event, and the field labelled 'food' has to contain some food. The fourth field represented in (8) might appear to contain something other than a type. Intuitively, it represents a constraint (hence the 'c' in the label) that the individual in the record is a lunch with the event aspect represented in the event-field and the food aspect represented in the food-field. 'lunch_ev_fd(x, event, food)' is regarded as representing a type of situation or state of affairs where this constraint holds. This type is also special in that it is a *dependent* type. That is, precisely which type this represents depends on the objects in the 'x', 'event' and 'food' fields of the record you are considering as a candidate for being of the type in (8). Technical details are developed in [8,13] where such dependent types are treated as pairs of functions and sequences of path-names indicating where the arguments of the function are to be found in the record. Both records and record types are treated as sets of fields (that is, pairs of labels and objects, in the case of records, and types, in the case of record types). This means that the order of the fields is not significant.

(8) is much closer to Asher's ([1]) proposal than the type proposed in [11] given in (9)

$$(9) \quad \begin{bmatrix} \mathrm{x} & : \mathit{Ind} \\ \mathrm{c}_1 & : \mathrm{food(x)} \\ \mathrm{c}_2 & : \mathrm{event(x)} \end{bmatrix}$$

This claims that there is an individual (i.e. a "thin particular") which is both food and an event. Asher argues against this and his analysis of dot types such as *Food·Event* does not have this consequence. While I would take issue with the detail of Asher's arguments against this analysis, there is a clear advantage to (8) in that it gives us food and event objects which can be separately counted. This will be important in section 6 where we consider the problem of counting books discussed in [1] and first raised by Tim Fernando in discussion.

We will designate the type in (8) by *LunchFrame*. How should this type figure in the content of an utterance of the common noun *lunch*? Common noun contents are of type (10)

$$(10) \quad \left[\mathrm{x}{:}\mathit{Ind}\right] \rightarrow \mathit{RecType}$$

that is, the type of functions from records of type $\left[\mathrm{x}{:}\mathit{Ind}\right]$ to record types. A function of this type will map a frame (a record with a field for an individual labelled 'x') to a record type. Record types do duty (among other things) as intuitive "propositions". Thus a "proposition" corresponding to *a lunch is served* could be (11)

$$(11) \quad \begin{bmatrix} \mathrm{x} & : \mathit{Ind} \\ \mathrm{c}_{\mathrm{lunch}} & : \mathrm{lunch(x)} \\ \mathrm{c}_{\mathrm{served}} & : \mathrm{served(x)} \end{bmatrix}$$

If this type is non-empty it has to be the case that there is an individual which is a lunch and which is served. The type being non-empty corresponds to the "proposition" being "true". "Falsity" corresponds to the type being empty. The type (10) corresponds to an intensional version of Montague's type $\langle e, t\rangle$ in that it is the type of functions that map individuals (represented as "thick particulars" or frames) to "propositions" modelled as record types (rather than truth-values as in Montague's original). We will refer to (10) as *Ppty*, short for *property*.

Our standard strategy for interpreting common nouns N with a corresponding predicate N' is shown in (12).

$$(12) \quad \lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right]\left(\left[\mathrm{c}_{N'}{:}N'(r.\mathrm{x})\right]\right)$$

This is an interpretation where N' is predicated of the individual in the x-field or r (represented by 'r.x'). In the case of nouns which are associated with a frame type, like *lunch* according to our analysis, we need to make the interpretation exploit the frame type as in (13)

(13) $\lambda r{:}[\mathrm{x}{:}\mathit{Ind}](\begin{bmatrix}\mathrm{event}{:}\mathit{Event}\\ \mathrm{food}{:}\mathit{Food}\\ \mathrm{c_{lunch}}{:}\mathrm{lunch\_ev\_fd}(r.\mathrm{x},\ \mathrm{event},\ \mathrm{food})\end{bmatrix})$

This proposal diverges from the earlier proposal in [11] which was that the interpretation of *lunch* should be

(14) $\lambda r{:}\begin{bmatrix}\mathrm{x}:\mathit{Ind}\\ \mathrm{c_1}{:}\mathrm{food}(\mathrm{x})\\ \mathrm{c_2}{:}\mathrm{event}(\mathrm{x})\end{bmatrix}([\mathrm{c_3}{:}\mathrm{lunch}(r.\mathrm{x})])$

This has the consequence that *lunch* can only be predicated of objects that are both food and events, since the characterization of the domain of the function gives us a presupposition that the argument of *lunch* is such an object. This seems to make it difficult handle cases of negation such as (15).

(15) I went to a meeting. It wasn't a lunch. (So I'm hungry now.)

3 Dynamic Generalized Quantifiers

Following the proposal for the treatment of generalized quantifiers in [7] we will treat determiners such as *a(n)* as predicates which hold between two properties, that is functions of type *Ppty*. Thus *a donkey runs* could be represented as (16).

(16) $\begin{bmatrix}\mathrm{c_{exists}} : \mathrm{exists}(\lambda r{:}[\mathrm{x}{:}\mathit{Ind}]([\mathrm{c_{donkey}}{:}\mathrm{donkey}(r.\mathrm{x})]),\\ \qquad\qquad \lambda r{:}[\mathrm{x}{:}\mathit{Ind}]([\mathrm{c_{run}}{:}\mathrm{run}(r.\mathrm{x})]))\end{bmatrix}$

This is the non-dynamic treatment of generalized quantification. The strategy is to say that a type

(17) $q(P_1, P_2)$

is non-empty just in case the corresponding classical generalized quantifier relation, $q*$, as given for example in [3], holds between the set of objects which have P_1 and the set of objects which have P_2. We make this precise as follows. If T is a type, then the *extension* of T, $[\check{}T]$, is to be the set of objects which are of type T, given in (18).

(18) $\{a \mid a : T\}$

If P is of type *Ppty*, then the property-extension or *P-extension* of P, $[\downarrow P]$, is the set of objects which have the property P, given in (19).

(19) $\{a \mid \exists r[r :[\mathrm{x}{:}\mathit{Ind}] \wedge r.\mathrm{x} = a \wedge [\check{}P(r)] \neq \emptyset]\}$

So now we can say

(20) $q(P_1, P_2)$ is a non-empty type just in case $q*$ holds between $[\downarrow P_1]$ and $[\downarrow P_2]$

For example if q is 'exists' then the type will be non-empty just in case (21) holds.

(21) $[\downarrow P_1] \cap [\downarrow P_2] \neq \emptyset$.

In our treatment of dynamic quantifiers in [7] we required quantifier predicates q to be polymorphic in order to be able to pass information from the first argument of q to domain type of the second argument. The idea is that in the interpretation of a sentence like *every farmer who owns a donkey likes it* the second argument of 'every' corresponding to *likes it* will have its domain restricted to farmers who own a donkey as in (22).

(22) $\lambda r : \begin{bmatrix} \mathrm{x} & : \mathit{Ind} \\ \mathrm{c_{farmer}} & : \mathrm{farmer(x)} \\ \mathrm{y} & : \mathit{Ind} \\ \mathrm{c_{donkey}} & : \mathrm{donkey(y)} \\ \mathrm{c_{own}} & : \mathrm{own(x,y)} \end{bmatrix} (\left[\mathrm{c_{like}}{:}\mathrm{like}(r.\mathrm{x}, r.\mathrm{y})\right])$

We will use the notation $X \sqsubseteq T$ to represent a variable over types which are subtypes of the record type T. The definition of subtype is a little involved due to the fact that we have dependencies. See [13] for a precise definition. We redefine the type *Ppty* to be that given in (23).

(23) $(X \sqsubseteq \left[\mathrm{x}{:}\mathit{Ind}\right]) \rightarrow \mathit{RecType}$

that is, the type of a function whose domain type is some subtype of $\left[\mathrm{x}{:}\mathit{Ind}\right]$ and which yields a record type for any object of that subtype.

In order to achieve dynamic quantification we introduce a notion of fixed point type and use the fixed point type of the first argument to q to restrict the domain type of the second argument to q. If $P{:}\mathit{Ppty}$, we say that a is a *fixed point for* P just in case $a : P(a)$. Not all properties will have fixed points, but for the examples we are discussing here it is straightforward to compute what the type of the fixed points should be. Consider the property P of being a farmer who owns a donkey in (24).

(24) $\lambda r : \left[\mathrm{x}{:}\mathit{Ind}\ \right] (\begin{bmatrix} \mathrm{c_{farmer}} & : \mathrm{farmer}(r.\mathrm{x}) \\ \mathrm{y} & : \mathit{Ind} \\ \mathrm{c_{donkey}} & : \mathrm{donkey(y)} \\ \mathrm{c_{own}} & : \mathrm{own}(r.\mathrm{x},\mathrm{y}) \end{bmatrix})$

The fixed points of P will be all and only the records of the type in (25).

(25)
$$\begin{bmatrix} \mathrm{x} & : \mathit{Ind} \\ \mathrm{c_{farmer}} & : \mathrm{farmer(x)} \\ \mathrm{y} & : \mathit{Ind} \\ \mathrm{c_{donkey}} & : \mathrm{donkey(y)} \\ \mathrm{c_{own}} & : \mathrm{own(x,y)} \end{bmatrix}$$

We will call this the *fixed point type of* P. Intuitively, the fixed point type is obtained by extending the type of the domain with the dependent type that characterises its range. For any property P which has fixed points, we will use $\mathcal{F}(\mathcal{P})$ to represent the fixed point type of P.[2]

This notion of fixed point type is exploited in order to make the first argument of the quantifier provide a *hypothetical context* for the second argument. The second argument becomes a function which requires as argument (i.e. context) a record which is of the fixed point type of the first argument. We call it hypothetical because it does not require that there be such a context. It just characterises the domain of the function.[3] The function which is the second argument is *restricted* by the fixed point type of the first argument. The restriction of a function by a type is defined in (26).

(26) If F is a function $\lambda v : T_1(\phi)$, then the *restriction of* F *by the type* T_2 is $\lambda v : T_1 \barwedge T_2(\phi)$

Here $T_1 \barwedge T_2$ represents the merge of T_1 and T_2 as defined in [13]. It is like standard meets of types (represented by $T_1 \wedge T_2$ where $a : T_1 \wedge T_2$ iff $a : T_1$ and $a : T_2$) except that in the case of record types it merges them together into a single record type using an operation closely related to graph unification.

We can now redefine what it means for a quantifier relation q to hold between two properties (the first of which has fixed points) using dynamic quantification as in (27).

(27) $q(P_1, P_2)$ is a non-empty type just in case $q*$ holds between $[\downarrow P_1]$ and $[\downarrow P_2 \upharpoonright \mathcal{F}(P_1)]$

4 Treating Copredication

Allowing generalized quantifier predicates to be polymorphic enables us to add additional constraints on the domains of functions corresponding to verb-phrases. For example, we require that records which are arguments to *take forever* in

[2] In formulating this precisely we need to make sure that the domain and range types of P do not have any labels in common.

[3] A similar analysis of generalized quantifiers exploiting contexts in type theory is given in Fernando (2001)

addition to introducing an individual also introduce an event aspect to the object.

(28) $\lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{event}{:}\mathit{Event}\end{bmatrix}(\left[\text{c}_{\text{forever}}{:}\text{take\_forever\_ev}(r.\text{x},r.\text{event})\right])$

Similarly *be delicious* will require that its subject has a food aspect.

(29) $\lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{food}{:}\mathit{Food}\end{bmatrix}(\left[\text{c}_{\text{delicious}}{:}\text{be\_delicious\_fd}(r.\text{x},r.\text{food})\right])$

The conjunction *be delicious and take forever* needs to require that its subject has both a food and an event aspect.[4]

(30) $\lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{food} & :\mathit{Food}\\ \text{event}{:}\mathit{Event}\end{bmatrix}(\begin{bmatrix}\text{c}_{\text{delicious}}{:}\text{be\_delicious\_fd}(r.\text{x},r.\text{food})\\ \text{c}_{\text{forever}} \quad :\text{take\_forever\_ev}(r.\text{x},r.\text{event})\end{bmatrix})$

Notice that the fixed point types of these properties are candidates for frame types. Similarly, nouns will correspond to properties whose fixed point types are candidates for frame types.

(31)

a. $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right](\begin{bmatrix}\text{food} & :\mathit{Food}\\ \text{c}_{\text{blancmange}} & :\text{blancmange}(r.\text{x},\text{food})\end{bmatrix})$

b. $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right](\begin{bmatrix}\text{event} & :\mathit{Event}\\ \text{c}_{\text{game}} & :\text{game}(r.\text{x},\text{event})\end{bmatrix})$

c. $\lambda r{:}\left[\text{x}{:}\mathit{Ind}\right](\begin{bmatrix}\text{event}{:}\mathit{Event}\\ \text{food}{:}\mathit{Food}\\ \text{c}_{\text{lunch}}{:}\text{lunch\_ev\_fd}(r.\text{x},\ \text{event},\ \text{food})\end{bmatrix})$

[4] This is actually a simplification of the compositional interpretation. Since we allow meets of types we can define the meet of two functions representing dependent types (i.e. mapping objects to types). Thus

$$\lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{c}_1{:}\text{food}(\text{x})\end{bmatrix}(\left[\text{c}_3{:}\text{be\_delicious}(r.\text{x})\right]) \wedge \lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{c}_2{:}\text{event}(\text{x})\end{bmatrix}(\left[\text{c}_4{:}\text{take\_forever}(r.\text{x})\right])$$

is to be

$$\lambda r{:}\begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{c}_1{:}\text{food}(\text{x})\end{bmatrix} \wedge \begin{bmatrix}\text{x} & :\mathit{Ind}\\ \text{c}_2{:}\text{event}(\text{x})\end{bmatrix}(\left[\text{c}_3{:}\text{be\_delicious}(r.\text{x})\right] \wedge \left[\text{c}_4{:}\text{take\_forever}(r.\text{x})\right])$$

which is equivalent to (though not identical with) (30). Since there is no requirement in our type theory that $T_1 \wedge T_2$ is identical with $T_2 \wedge T_1$ (although they are equivalent in that they will always have the same extension) we could in principle have the means to distinguish examples mentioned by one of the reviewers such as: *John read and burnt the book* as opposed to **John burnt and read the book*. However, I suspect that the best way to treat these examples is in terms of Gricean principles concerning narration, *cf.* Grice's example *John got into bed and took off his trousers*.

Lunch objects are thus required to have both a food and an event aspect.[5]

Following the theory of dynamic quantifiers sketched in section 3, the function which is the first argument to the quantifier predicated is used to further restrict the domain of the second function. Thus *the game took forever* would actually be interpreted in terms of (32).

$$(32)\quad \mathrm{the}(\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right](\begin{bmatrix}\mathrm{event} : \mathit{Event}\\ \mathrm{c_{game}} : \mathrm{game\_ev}(r.\mathrm{x},\mathrm{event})\end{bmatrix}),\ \lambda r{:}\begin{bmatrix}\mathrm{x} & :\mathit{Ind}\\ \mathrm{event}{:}\mathit{Event}\\ \mathrm{c_{game}}{:}\mathrm{game\_ev}(\mathrm{x},\mathrm{event})\end{bmatrix}(\left[\mathrm{c_{forever}}{:}\mathrm{took\_forever\_ev}(r.\mathrm{x},r.\mathrm{event})\right]))$$

Thus the first argument to the quantifier predicate provides a context in which the second argument is interpreted by restricting the domain of the second argument. Normally the information passed on by the first argument is a subtype of the domain type of the original second argument, taken, we assume from the currently relevant lexical resource, in this case represented in (28).

With *lunch* we can use either verb-phrases requiring food or events since in either case the function corresponding to *lunch* will provide us with a subtype of the domain type of the second argument to the quantifier.

5 Treating Lexical Innovation

If we say that the game is delicious, we obtain the dynamic quantification in (33).

$$(33)\quad \mathrm{the}(\lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right](\begin{bmatrix}\mathrm{event}{:}\mathit{Event}\\ \mathrm{c_{game}}{:}\mathrm{game\_ev}(r.\mathrm{x},\mathrm{event})\end{bmatrix}),\ \lambda r{:}\begin{bmatrix}\mathrm{x} & :\mathit{Ind}\\ \mathrm{event}{:}\mathit{Event}\\ \mathrm{c_{game}}{:}\mathrm{game\_ev}(\mathrm{x},\mathrm{event})\\ \mathrm{food} & :\mathit{Food}\end{bmatrix}(\left[\mathrm{c_{delicious}}{:}\mathrm{be\_delicious\_fd}(r.\mathrm{x},r.\mathrm{food})\right]))$$

Here the fixed point type passed to the second argument from the first argument is *not* a subtype of the domain type for *delicious* in the lexical resource as represented in (29), since it does not provide the food-aspect required by the second

[5] There is perhaps an issue of whether the additional contraints on nouns should be treated as presuppositions as they are in [11]. Certainly the sentence *That pink thing isn't blancmange* doesn't seem to require that what is referred to by *that pink thing* is food. But then perhaps the difference between the predicate represented by *be blancmange* and that represented by *blancmange* is that the former only introduces an entailment about food whereas the latter introduces a presupposition. If that is so then we could go back to treating these aspect constraints as presuppositions. However, since we are constructing the fixed point type to provide the restriction on the second argument it does not matter which of the two options we take for present purposes.

argument. In this case we say that the use of *(is) delicious* is *innovative* with respect to the lexical resource. Note that the relative definition of innovation with respect to a resource is important since we can derive innovation with respect to a dialogue partner from this – dialogue partners may have different lexical resources available. Note that (33), although it follows our standard treatment of dynamic quantification, is not the most natural interpretation of the sentence. This is because given normal assumptions about resources we have available it is not easy to construe what a food aspect of a game would be. Thus it is more natural to coerce the interpretation of *delicious* to be a predicate of the event aspect of the game, so the hearer is left with the puzzle of how to figure out a meaning for delicious that relates to events. This coercion would thus involve a modification of the lexical resource ([29]).

This suggests a rather different view of types constructed from type-theoretical predicates than was suggested in constructive type theory which was created in order to reason about mathematical objects. Ranta ([28], pp. 53ff) has an illuminating discussion of non-mathematical propositions in type theory in which he suggests that an appropriate proof object for Davidson's sentence *Amundsen flew over the North Pole* would be a Davidsonian event of Amundsen flying over the North Pole. A problem he points to is that the kind of predicates which natural language requires are not sets in the constructive mathematical sense. He mentions predicates like *man* and *tree*. The problem seems magnified when we consider a predicate like *delicious* with is both vague (there is no obvious exact boundary between the delicious and the non-delicious) and subjective (what is delicious for me may not be so for you). He suggests three ways of approaching this problem: (i) work with *types* rather than sets, (ii) develop techniques of *approximation* and (iii) study delimited *models* of language use. It seems to me that all three of these should be developed in an adequate type-theoretical approach to natural language. It seems important that we recognize that human beings reason with *inexact* types for which they do not have (or indeed there may not even exist) clearly defined procedures for determining whether objects belong to those types or not. Such types may be refined by successive approximations as a result of being exposed to natural language utterances in a variety of situations. Finally, we have the ability to "tie down" certain expressions in a given restricted domain, "for the purposes of discussion" or at least to know what certain complex expressions would mean under the assumption that exact interpretations were assigned to their constituent parts. Thus, when being confronted with an innovative use of *delicious* applied to some game, while the type itself is perfectly well-defined, we may not have a way of determining exactly which are the proof-objects which belong to this type. We may observe certain aspects of the situation to which it was applied and assume they are possibly relevant, awaiting future uses for further refinement. If we have reason to do so we may engage in clarification, e.g. *What do you mean "delicious"?*. It is important to note that this can often occasion a refinement in the original speaker's view of what they said. It is not the case that speakers always use types which are exact for them.

This inexactitude is an important feature of natural language. It appears to play an important role in making it adaptable to new situations and domains and a large part of this involves the generation of innovative uses of lexical items, often involving the creation of new meanings for words already available in a lexical resource. The fact that the interpretation for *delicious* is inexact, means that we can begin to consider what it means for a game to be delicious. Innovation seems much harder with expressions which have an exact interpretation such as *divisible by three*, *is a prime number*. It is very hard to imagine what it might mean for a game to be divisible by three. On the view of natural languages as formal languages inexactitude can appear to be a big challenge for semantics. On the view of natural languages as providing resources (i.e. a toolbox) for the construction of formal languages, we can regard a lexical resource as providing an inexact predicate which could be refined to an exact one in a given domain or application.

6 How Many Books are on the Shelf?

In [1] Asher discusses an example of counting books on a shelf, originally due to Tim Fernando. Suppose there are exactly two copies of *War and Peace*, two copies of *Ulysses*, and six copies of the Bible on a shelf. How many books are there on the shelf? If you are counting physical objects the answer is ten. If you are counting informational or textual objects the answer might be three. It depends a little whether, for example, the copies of *War and Peace* actually contain precisely the same text. Suppose one is the Russian original and the other is a Swedish translation. In one sense it is the same book but for certain purposes you might want to count it as two (for example if you are removing doublets from your library). Similarly you may have two different editions of the Bible containing the same basic text but with different annotations by different biblical scholars. Sometimes you will want to count that as one and sometimes as two. It depends on the purposes you have at hand.

If you look up *book* on the Berkeley FrameNet (`http://framenet.icsi.berkeley.edu`)[6] you will find that it has been assigned the following frame elements: *Author, Components, Genre, Text, Title, Topic*. This is rather different from the aspects of books that are normally discussed in the generative lexicon literature which Asher represent as the types Physical Object and Informational Object. Of the frame elements one could imagine counting books by *Text* and one might be tempted to use *Title*, though this will go wrong if there are two books with the same title. (Google books returns 9,950 entries for "Introduction to Chemistry" ...). It would seem useful to add a frame element for physical object to the frame. The *Text* frame to which book belongs mentions that a text can be a physical object and has a role for *Material* (such as vellum). We will take a property corresponding to book which uses a frame corresponding to the dot type that Asher discusses.

[6] Accessed 13th February, 2011.

$$(34)\quad \lambda r{:}\left[\mathrm{x}{:}\mathit{Ind}\right]\ \left(\begin{bmatrix}\text{physobj} & : \mathit{PhysObj}\\ \text{infobj} & : \mathit{InfObj}\\ \mathrm{c}_{\text{book}} & : \text{book\_ph\_inf}(r.\text{x},\text{physobj},\text{infobj})\end{bmatrix}\right)$$

Our standard way of computing the property extension of this property given in (19) would give us the set of objects in the x-field. However, given that we now have aspects in separate fields of our frames we could relativize our notion of property extension to labels in the frame, as in (35).

(35) The *P-extension of P relative to label* ℓ, $[\downarrow_\ell P]$, is
$\{a \mid \exists r[r:\left[\mathrm{x}{:}\mathit{Ind}\right] \wedge r.\ell = a \wedge \check{P}(r)] \neq \emptyset\}$

Choosing different labels will enable us to obtain property extensions of different cardinalities as desired.

7 Conclusion

We have suggested that a record type theoretical approach to semantics gives us an approach to different aspects of objects which have previously been treated by Asher and Pustejovsky as involving dot types. The approach points to a connection between dynamic generalized quantification and the coercion involved in innovative uses of linguistic predicates. We have suggested an approach to lexical innovation which draws on the ideas of resources presented in [16,22,23,15]. Finally, we have suggested a way in which the approach can be used to account for the different ways of counting objects that seem to be available to us depending on the purposes at hand.

The account we present here is very similar in spirit to that presented by Asher in [1]. The notion of dot type that Asher develops there seems very close to the notion of record type and the revisions that I have undertaken since the earlier version of my analysis (creating separate objects to represent aspects) moves the present approach much closer to Asher's proposal. One difference between our approaches is that for me record types are used throughout the TTR approach I want to support, e.g. as models of DRSs and feature structures and, as developed in Ginzburg's work [20], in the analysis of dialogue information states or gameboards. Thus the kind of phenomena analyzed in this paper do not require an extension of the type theory already required elsewhere in the theory. In contrast dot types do not play such a general role in Asher's theory. Thus I would like to propose that dot types can be usefully construed as record types.

Related Work: Type Theories Ancient and Modern

Pustejovsky's work on the generative lexicon and dot-types in particular has given rise to a considerable number of proposals exploiting type theories of various kinds. A very useful survey of this work is given in [4]. The proposed treatment of lexical semantics in [4] is similar to ours in that it attempts to enrich Montague's type theoretic approach rather than discard it. It differs, however, in that it eschews the use of record types.

A recent proposal by Luo [24] presents an approach in terms of coercive subtyping. As Luo points out, coercive subtyping is related to the use of record types where record projection can be used to model the coercions involved. Luo's approach is to replace Montague's type theoretical approach where, for example, nouns are interpreted in terms of predicates modelled as functions of type $\langle e, t\rangle$[7] with an approach based on what Luo calls "modern type theories" where nouns correspond to types. Our approach is rather to create a hybrid of Montague's approach and modern type theories where nouns correspond to predicates modelled as objects of type $[\mathrm{x}{:}\mathit{Ind}]{\to}\mathit{RecType}$. That is, we have, like Montague, a predicate modelled as a function, but the result of applying that function to an appropriate argument is a type as in the "modern type theory" approach. (My particular proposal is that this type should, furthermore, be a record type.) What I have given up is the notion of canonicity which Luo discusses. This notion is extremely important for type theoreticians who have grown up in a proof theoretic tradition. In place of canonicity I have a notion of model theoretic objects associated with types, which, to me, having grown up in a model theoretic tradition, seems an appropriate way to make sure that the system gets grounded with canonical objects. There are deep and perplexing issues of both technical and philosophical nature which arise in this discussion and I do not pretend to have a complete grasp of all of them. But I am convinced that the way to illumination is the formulation and comparison of precise proposals for alternative treatments of the linguistic data.

Acknowledgments. An earlier version of this paper was presented as [11]. It contains revised material from [10]. The current paper contains an significant revision of the analysis and an extension of the results in the light of more recent discussion, particularly by Asher. I am grateful for useful comments on earlier versions of this work to Nicholas Asher, Tim Fernando, Stuart Shieber, two anonymous reviewers for the GSLT internal conference and participants in the Language Technology seminar in Gothenburg. I would also like to thank three anonymous referees for LACL 2011 for insightful comments. This work was supported by Vetenskapsrådet projects 2002-4879 *Records, types and computational dialogue semantics*, `http://www.ling.gu.se/~cooper/records/`, 2005-4211 *Library-based Grammar Engineering*, `http://www.cs.chalmers.se/ãarne/GF/doc/vr.html` and 2009-1569 *Semantic analysis of interaction and coordination in dialogue*, `http://sites.google.com/site/saicdproject/`.

References

1. Asher, N.: Lexical Meaning in Context: A Web of Words. Cambridge University Press, Cambridge (2011)
2. Asher, N., Pustejovsky, J.: Word Meaning and Commonsense Metaphysics. In: Course Materials for Type Selection and the Semantics of Local Context, ESSLLI 2005 (2005)

[7] Or $\langle\langle s, e\rangle, t\rangle$ to be historically precise.

3. Barwise, J., Cooper, R.: Generalized quantifiers and natural language. Linguistics and Philosophy 4(2), 159–219 (1981)
4. Bassac, C., Mery, B., Retoré, C.: Towards a type-theoretical account of lexical semantics. Journal of Logic, Language and Information 19(2), 229–245 (2010)
5. Betarte, G.: Dependent Record Types and Algebraic Structures in Type Theory. Ph.D. thesis, Department of Computing Science, University of Gothenburg and Chalmers University of Technology (1998)
6. Betarte, G., Tasistro, A.: Extension of Martin-Löf's type theory with record types and subtyping. In: Sambin, G., Smith, J. (eds.) Twenty-Five Years of Constructive Type Theory. Oxford Logic Guides, vol. 36. Oxford University Press, Oxford (1998)
7. Cooper, R.: Dynamic generalised quantifiers and hypothetical contexts. In: Ursus Philosophicus, a festschrift for Björn Haglund. Department of Philosophy, University of Gothenburg (2004), http://www.phil.gu.se/posters/festskrift/
8. Cooper, R.: Austinian truth, attitudes and type theory. Research on Language and Computation 3, 333–362 (2005)
9. Cooper, R.: Records and record types in semantic theory. Journal of Logic and Computation 15(2), 99–112 (2005)
10. Cooper, R.: A record type theoretic account of copredication and dynamic generalized quantification. In: Kvantifikator för en Dag, Essays dedicated to Dag Westerståhl on his sixtieth birthday. Department of Philosophy, University of Gothenburg (2006), http://www.phil.gu.se/posters/festskrift3/
11. Cooper, R.: Copredication, dynamic generalized quantification and lexical innovation by coercion. In: Proceedings of GL 2007, Fourth International Workshop on Generative Approaches to the Lexicon (2007), http://www.ling.gu.se/~cooper/records/copredinnov.pdf
12. 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)
13. Cooper, R.: Type theory and semantics in flux. In: Kempson, R., Asher, N., Fernando, T. (eds.) Handbook of the Philosophy of Science, Philosophy of Linguistics. Elsevier BV, Amsterdam (fthc), general editors: Gabbay, D.M., Thagard, P., Woods, J.: https://sites.google.com/site/typetheorywithrecords/drafts/ddl-final.pdf
14. Cooper, R., Kempson, R. (eds.): Language in Flux: Dialogue Coordination, Language Variation, Change and Evolution, Communication, Mind and Language, vol. 1. College Publications, London (2008)
15. Cooper, R., Larsson, S.: Compositional and ontological semantics in learning from corrective feedback and explicit definition. In: Edlund, J., Gustafson, J., Hjalmarsson, A., Skantze, G. (eds.) Proceedings of DiaHolmia: 2009 Workshop on the Semantics and Pragmatics of Dialogue, pp. 59–66. Department of Speech, Music and Hearing, KTH (2009)
16. Cooper, R., Ranta, A.: Natural Languages as Collections of Resources. In: Cooper, R., Kempson R., (eds.) [14], pp. 109–120
17. Coquand, T., Pollack, R., Takeyama, M.: A logical framework with dependently typed records. Fundamenta Informaticae XX, 1–22 (2004)
18. Fillmore, C.J.: Frame semantics. In: Linguistics in the Morning Calm, pp. 111–137. Hanshin Publishing Co., Seoul (1982)
19. Fillmore, C.J.: Frames and the semantics of understanding. Quaderni di Semantica 6(2), 222–254 (1985)
20. Ginzburg, J.: The Interactive Stance: Meaning for Conversation. Oxford University Press, Oxford (fthc)

21. Kratzer, A.: An Investigation of the Lumps of Thought. Linguistics and Philosophy 12, 607–653 (1989)
22. Larsson, S.: Formalizing the dynamics of semantic systems in dialogue. In: Cooper, R., Kempson, R. (eds.) [14], p. 344
23. Larsson, S., Cooper, R.: Towards a formal view of corrective feedback. In: Alishahi, A., Poibeau, T., Villavicencio, A. (eds.) Proceedings of the Workshop on Cognitive Aspects of Computational Language Acquisition, pp. 1–9. EACL (2009)
24. Luo, Z.: Type-theoretical semantics with coercive subtyping. In: Proceedings of SALT 20. pp. 38–56 (2010)
25. McCawley, J.D.: The Role of Semantics in a Grammar. In: Bach, E., Harms, R.T. (eds.) Universals in Linguistic Theory. Holt, Rinehart and Winston (1968)
26. Montague, R.: Formal Philosophy: Selected Papers of Richard Montague. Yale University Press, New Haven (1974), ed. and with an introduction by Thomason, R.H.
27. Pustejovsky, J.: The Generative Lexicon. MIT Press, Cambridge (1995)
28. Ranta, A.: Type-Theoretical Grammar. Clarendon Press, Oxford (1994)
29. Ritchie, G.: The linguistic analysis of jokes. Routledge, London (2004)
30. Ruppenhofer, J., Ellsworth, M., Petruck, M.R., Johnson, C.R., Scheffczyk, J.: FrameNet II: Extended Theory and Practice (2006), http://framenet.icsi.berkeley.edu/index.php?option=com_wrapper&Itemid=126
31. Tasistro, A.: Substitution, record types and subtyping in type theory, with applications to the theory of programming. Ph.D. thesis, Department of Computing Science, University of Gothenburg and Chalmers University of Technology (1997)

On Dispersed and Choice Iteration in Incrementally Learnable Dependency Types

Denis Béchet[1], Alexandre Dikovsky[1], and Annie Foret[2]

[1] LINA UMR CNRS 6241, Université de Nantes, France
{Denis.Bechet,Alexandre.Dikovsky}@univ-nantes.fr
[2] IRISA, Université de Rennes 1, France
Annie.Foret@irisa.fr

Abstract. We study learnability of Categorial Dependency Grammars (CDG), a family of categorial grammars expressing all kinds of projective, discontinuous and repeatable dependencies. For these grammars, it is known that they are not learnable from dependency structures.

We propose two different ways of modelling the repeatable dependencies through iterated types and the two corresponding families of CDG which cannot distinguish between the dependencies repeatable at least K times and those repeatable any number of times. For both we show that they are incrementally learnable in the limit from dependency structures.

Keywords: Grammatical inference, Categorial grammar, Dependency grammar, Incremental learning, Iterated types.

1 Introduction

Languages generated by grammars in a class $\mathcal{G}$ are learnable if there is an algorithm A which, for every target grammar $G_T \in \mathcal{G}$ and every finite set σ of generated words, computes a hypothetical grammar $A(\sigma) \in \mathcal{G}$ in a way that:

(i) the sequence of languages generated by the grammars $A(\sigma)$ converges to the target language $L(G_T)$ and
(ii) this is true for any increasing enumeration of sub-languages $\sigma \subset L(G_T)$.

This concept due to E.M. Gold [10] is also called learning from strings. More generally, the hypothetical grammars may be computed from finite sets of structures defined by the target grammar. This kind of learning is called learning from structures. Both concepts were intensively studied (see the surveys in [1] and in [11]). In particular, it is known that any family of grammars generating all finite languages and at least one infinite language (as it is the case of all classical grammars) is not learnable from strings. At the same time, some interesting positive results were also obtained. In particular, k-rule string and term generating grammars are learnable from strings for every k [14] and k-rigid (i.e. assigning no more than k types per word) classical categorial grammars (CG) are learnable from the so called "function-argument" structures and also from strings [4,11].

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 80–95, 2011.

In our recent paper [2], we adapt this concept of learning to surface dependency structures (DS), i.e. graphs of named binary relations on words, called dependencies (see Fig. 1,2,4). Dependencies are asymmetric. When two words w_1, w_2 are related through dependency d, $w_1 \xrightarrow{d} w_2$, w_1 is called governor and w_2 is called subordinate. Dependencies may be projective, i.e. non-crossing, as in Fig. 1,4, or discontinuous like $clit-a-obj$, $clit-3d-obj$ in Fig. 2. Very importantly, the linguistic intuition behind a dependency name d is that it identifies **all** syntactic and distributional properties of the subordinate in the context of its governor. In more detail, it identifies its syntactic role (e.g., "subject" "direct object", "copula", "attribute", "circumstantial" etc.), its position with respect to the governor and its part of speech (POS). In principle, the words dependent through the same dependency are substitutable (see the quasi-Kunze property in [13]). This might explain why the dependency structure cannot be completely defined through constituent structure with head selection.

Grammars defining dependency relations directly, in conformity with the basic dependency structure principles (see [13]) must face the problem of expressing the so called repeatable dependencies. These dependencies satisfy specific conditions most clearly formulated by I. Mel'čuk in the form of the following Principle of repeatable dependencies (see [13]).

> Every dependency is either *repeatable* or *not repeatable*. If a dependency d is not repeatable, then no word may have two subordinates through d. If d is repeatable, then **any word** g which governs a subordinate word s through d may have **any number** of subordinates through d.

E.g., the verbs may have any number of subordinate circumstantials (but no more than one direct or indirect complement), the nouns may have any number of attributes and of modifiers (but no more than one determiner), etc.

We choose the Categorial Dependency Grammars (CDG) [7,5] as the grammars to be inferred from dependency structures because these grammars define DS directly, without any order restrictions and in particular, they express the repeatable dependencies through the so called "iterated" types in conformity with the Principle of repeatable dependencies. As it was shown in [3], the k-rigid CDG without iterated types are learnable from analogues of the function-argument structures (and from strings) as it is the case of the classical categorial grammars. At the same time, even rigid (i.e. 1-rigid) CDG with iterated types are not learnable from function-argument structures. Moreover, in [2] we show that they are not learnable from the DS themselves. This may be seen as a proof of unlearnability from dependency treebanks of dependency grammars which express dependency relations in accordance with the basic dependency structure principles (in particular with the Principle of repeatable dependencies). On the other hand, in strict conformity with this Principle, in [2] a subclass of CDG which cannot distinguish between the dependencies repeatable K (or more) times and those repeatable any number of times (the Principle sets $K = 2$) is defined. For

these CDG, called in [2] K-star revealing, it is proved that they are incrementally learnable from dependency structures.

It is significant that the Principle of repeatable dependencies is uncertain as it concerns the precedence order of the repeatable subordinates. Let us consider the fragment in Fig. 1 of a DS of the French sentence *Ils cherchaient pendant une semaine surtout dans les quartiers nord un des deux évadés en bloquant systématiquement les entrées - sorties* (*fr.* **They tracked for a week especially in the nord quarters one of the two fugitives systematically blocking entries and exits*). For instance, for $K = 3$, the dependency *circ* is repeatable or not depending on how are counted its occurrences: all together or separately on the left and on the right of the direct complement *évadés*.

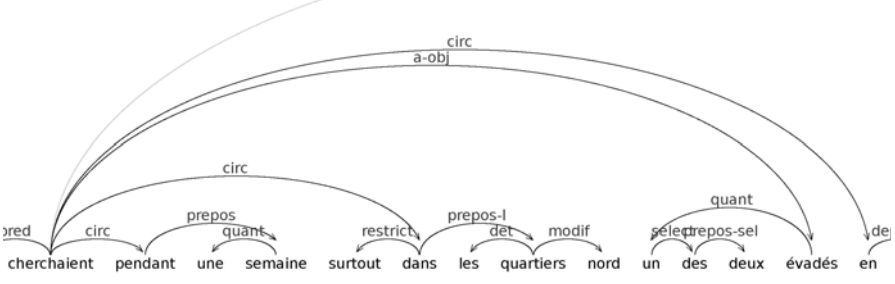


Fig. 1. Repeatable dependencies

In [2] is considered the simplest interpretation of `repeatable` dependencies as `consecutively repeatable`. This reading cannot be linguistically founded (even if the consecutively repeatable dependencies are the most frequent). In this paper two other readings of the repeatability are considered. One reading is maximally liberal and says that a subordinate through a repeatable dependency may be found anywhere on the left (or on the right) of the governor. We call such iteration `dispersed`. The other reading is closer to the consecutive one, but extends it with the disjunctive choice of repeatable dependencies which may occur in the same argument position. Respectively, we consider two extensions of the CDG: one with the dispersed iteration types (called `dispersed iteration` CDG) and the other with the choice iteration types (called `choice iteration` CDG). For both we consider the corresponding notion of K-star revealing: the `dispersed` K`-star revealing` and the `choice` K`-star revealing`. We show that both classes are incrementally learnable in the limit from dependency structures.

The plan of this paper is as follows. Section 2 introduces the background notions: Categorial Dependency Grammars, dispersed and choice iterations. Section 3, presents the notion of incremental learning in the limit. In Section 4, the condition of K-star revealing is adapted to dispersed iteration CDG and their incremental learnability from dependency structures is proved. Section 5 presents a similar result for choice iteration CDG.

2 Categorial Dependency Grammars with Extended Iteration Types

2.1 Categorial Dependency Grammars

`Categorial Dependency Grammars` (`CDG`) define projective dependency structures assigning to every word a set of first order types in which the argument subtypes determine the outgoing dependencies of the word and the head subtype determines its incoming dependency. They also define discontinuous dependencies through the so called potentials of the types, i.e. strings of `polarized valencies`. Every positive valency in the potential of a word's type determines the name and the direction of an outgoing dependency of the word, and every negative valency determines the name and the direction of the word's incoming dependency. The correspondence between the `dual` valencies (i.e. those having the same name and direction and the opposite signs) is established using general valency pairing principles such as **FA**: *Two dual valencies which are first available in the indicated direction may be paired.* In this way, the CDG define the dependency structures in the most direct and natural way and without any restrictions to the word order. Definitions, motivation, illustrations and properties of various classes of CDG may be found in [7,8,5,6].

Definition 1. *Let* $\mathbf{C}$ *be a set of* `dependency names` *and* $\mathbf{V}$ *be a set of* `valency names`. *The expressions of the form* $\swarrow v$, $\nwarrow v$, $\searrow v$, $\nearrow v$, *where* $v \in \mathbf{V}$, *are called* `polarized valencies`. $\nwarrow v$ *and* $\nearrow v$ *are* `positive`, $\swarrow v$ *and* $\searrow v$ *are* `negative`; $\nwarrow v$ *and* $\swarrow v$ *are* `left`, $\nearrow v$ *and* $\searrow v$ *are* `right`. *Two polarized valencies with the same valency name and orientation, but with the opposite signs are* `dual`.

An expression of one of the forms $\#(\swarrow v)$, $\#(\searrow v)$, $v \in \mathbf{V}$, *is called* `anchor type` *or just* `anchor`. *An expression of the form* d^* *where* $d \in \mathbf{C}$, *is called* `iterated dependency type`. *Anchor and iterated dependency types and dependency names are* `primitive types`.

An expression of the form $t = [l_m \backslash \ldots \backslash l_1 \backslash H / \ldots / r_1 \ldots / r_n]$ *in which* $m, n \geq 0$, $l_1, \ldots, l_m, r_1, \ldots, r_n$ *are primitive types and* H *is either a dependency name or an anchor type, is called* `basic dependency type`. $l_1, \ldots, l_m$ *and* $r_1, \ldots, r_n$ *are respectively left and right* `argument subtypes` *of* t. H *is called* `head subtype` *of* t *(or* `head type` *for short).*

A (possibly empty) string P *of polarized valencies is called* `potential`.

A `dependency type` *is an expression* B^P *where* B *is a basic dependency type and* P *is a potential.* $\mathbf{CAT}(\mathbf{C}, \mathbf{V})$ *denotes the set of all dependency types over* $\mathbf{C}$ *and* $\mathbf{V}$.

CDG are defined using the following calculus of dependency types [1] (with $C \in \mathbf{C}$, $H \in \mathbf{C}$ or an anchor, $V \in \mathbf{V}$, a basic type α and a residue of a basic type β):

$\mathbf{L^1}$. $H^{P_1}[H \backslash \beta]^{P_2} \vdash [\beta]^{P_1 P_2}$

$\mathbf{I^1}$. $C^{P_1}[C^* \backslash \beta]^{P_2} \vdash [C^* \backslash \beta]^{P_1 P_2}$

$\mathbf{\Omega^1}$. $[C^* \backslash \beta]^P \vdash [\beta]^P$

[1] We show left-oriented rules. The right-oriented are symmetrical.

D$^{\mathbf{l}}$. $\alpha^{P_1(\swarrow V)P(\nwarrow V)P_2} \vdash \alpha^{P_1PP_2}$, if the potential $(\swarrow V)P(\nwarrow V)$ satisfies the following pairing rule **FA** (*first available*):

$$\textbf{FA}: \quad P \text{ has no occurrences of } \swarrow V, \nwarrow V.$$

L$^{\mathbf{l}}$ is the classical elimination rule. Eliminating the argument subtype $H \neq \#(\alpha)$ it constructs the (`projective`) dependency H and concatenates the potentials. $H = \#(\alpha)$ creates the `anchor dependency`. **I**$^{\mathbf{l}}$ derives $k > 0$ instances of C. $\mathbf{\Omega}^{\mathbf{l}}$ serves for the case $k = 0$. **D**$^{\mathbf{l}}$ creates `discontinuous dependencies`. It pairs and eliminates dual valencies with name V satisfying the rule **FA** to create the discontinuous dependency V.

To compute the DS from proofs, these rules should be relativized with respect to the word positions in the sentence. To this end, when a type $B^{v_1 \cdots v_k}$ is assigned to the word in a position i, it is encoded using the `state` $(B, i)^{(v_1,i)\ldots(v_k,i)}$. The corresponding relativized state calculus is shown in [2]. In this calculus, for every proof ρ represented as a sequence of rule applications, one may define the DS `constructed` in this proof for a sentence x and written $DS_x(\rho)$.

Definition 2. *A* `categorial dependency grammar (CDG)` *is a system* $G = (W, \mathbf{C}, \mathbf{V}, S, \lambda)$, *where* W *is a finite set of words,* $\mathbf{C}$ *is a finite set of dependency names containing the selected name* S *(an axiom),* $\mathbf{V}$ *is a finite set of valency names, and* λ, *called* `lexicon`, *is a finite substitution on* W *such that* $\lambda(a) \subset \mathbf{CAT}(\mathbf{C}, \mathbf{V})$ *for each word* $a \in W$.

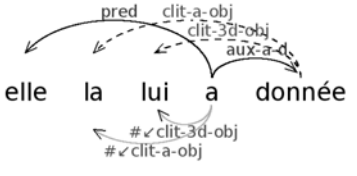
For a DS D *and a sentence* x, *let* $G(D, x)$ *denote the relation:*

"$D = DS_x(\rho)$, *where* ρ *is a proof of* $\Gamma \vdash S$ *for some* $\Gamma \in \lambda(x)$*".*

Then the `language` *generated by* G *is the set* $L(G) =_{df} \{w \mid \exists D\ G(D, w)\}$ *and the* `DS-language` *generated by* G *is the set* $\Delta(G) =_{df} \{D \mid \exists w\ G(D, w)\}$. $G_1 \equiv_s G_2$ *iff* $\Delta(G_1) = \Delta(G_2)$.

CDG are more expressive than CF-grammars (see [5,6]) and analyzed in polynomial time. In fact, they are equivalent to real time pushdown automata with independent counters [12]. Importantly, they express discontinuous DS in a direct and natural way. For instance, the DS in Fig. 2 is generated using the following type assignment:

$elle \mapsto [pred]$, $la \mapsto [\#(\swarrow clit\text{–}a\text{–}obj)]^{\swarrow clit\text{–}a\text{–}obj}$,
$lui \mapsto [\#(\swarrow clit\text{–}3d\text{–}obj)]^{\swarrow clit\text{–}3d\text{–}obj}$, $donnée \mapsto [aux\text{–}a\text{–}d]^{\nwarrow clit\text{–}3d\text{–}obj \nwarrow clit\text{–}a\text{–}obj}$,
$a \mapsto [\#(\swarrow clit - 3d - obj) \backslash \#(\swarrow clit - a - obj) \backslash pred \backslash S / aux - a - d]$



(*fr.* **she* $it_{\mathbf{g}=fem}$ *to him has given*)

Fig. 2. Non-projective dependency structure

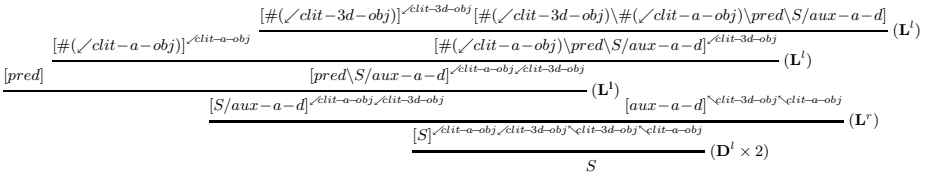
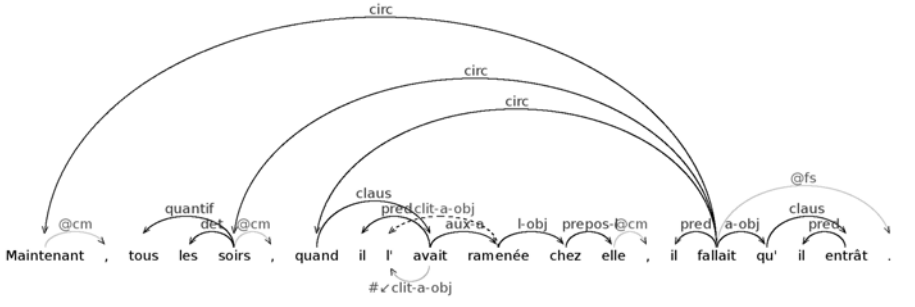
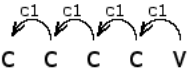


Fig. 3. Dependency structure correctness proof



(`fr`. **now all the evenings when he took her home he had to enter* [*M.Proust*])

Fig. 4. Iterated circumstantial dependency

(see the proof in Fig. 3). The iterated types also allow one to naturally express repeatable dependencies satisfying the Principle of repeatable dependencies. E.g., the repeatable circumstantial dependency *circ* in Fig. 4 may be determined by the type $[pred\backslash circ^*\backslash S/a-obj]$ assigned to the verb *fallait* (*had to*). One can see that such repeatable dependencies are of the kind we call "consecutive" in the Introduction. Iteration-less CDG cannot define such DS. Indeed, the assignments $a \mapsto [\alpha\backslash d]$ and $b \mapsto [d\backslash\beta]$ derive for ab the dependency $a \xleftarrow{d} b$. Therefore, the assignments $v \mapsto [c1\backslash S]$, $c \mapsto [c1\backslash c1], [c1]$ will derive for *ccccv* the sequenced (not iterated) dependencies as in the DS 

2.2 Dispersed and Choice Iterations

We will consider two different models of repeatable dependencies. One of them, called `dispersed iteration`, represents the case where the subordinates through a repeatable dependency may occur in any position on the left (respectively, on the right) of the governor. The other one, called `choice iteration`, will represent the case where the subordinates through one of several repeatable dependencies may occur in one and the same argument position. To define these models, we extend the primitive types with two new primitives: `dispersed iteration`

$\{d_1^*, \ldots, d_k^*\}$ and `choice iteration` $(d_1|\ldots|d_k)^*$, where $d_1, \ldots, d_k$ are dependency names.[2] Respectively we obtain two kinds of extended types.

Definition 3. *1. We call* `dispersed iteration types` *the expressions* B^P *in which* P *is a potential,* $B = [\alpha_1 \backslash L_m \backslash \ldots \backslash L_1 \backslash H / \ldots / R_1 \ldots / R_n / \alpha_2]$, $L_m, \ldots L_1$, H, $R_1 \ldots, R_n$ *are not iterated primitive types and* α_1, α_2 *are dispersed iterations (possibly empty, i.e.* $k = 0$*).*[3]
2. We call `choice iteration types` *the expressions* B^P *where* P *is a potential,* $B = [L_m \backslash \ldots \backslash L_1 \backslash H / \ldots / R_1 \ldots / R_n]$, H *is a not iterated primitive type and* $L_m, \ldots L_1$, $R_1 \ldots, R_n$ *are choice iterations or not iterated primitive types.*
3. Grammars using only dispersed iteration types are called `dispersed iteration CDG`*, those using only choice iteration types are called* `choice iteration CDG`*.*

Here are the respective extensions of the CDG calculus:

1. Choice iteration rules:
IC$^{\mathbf{l}}$. $C^{P_1}[(\alpha_1|C|\alpha_2)^* \backslash \beta]^{P_2} \vdash [(\alpha_1|C|\alpha_2)^* \backslash \beta]^{P_1 P_2}$.
ΩC$^{\mathbf{l}}$. $[(\alpha_1|C|\alpha_2)^* \backslash \beta]^P \vdash [\beta]^P$
LC$^{\mathbf{l}}$ and **DC$^{\mathbf{l}}$** as **L$^{\mathbf{l}}$** and **D$^{\mathbf{l}}$** in the CDG calculus.
2. Dispersed iteration rules:
LD$^{\mathbf{l}}$. $H^{P_1}[\{\alpha\} \backslash H \backslash \beta / \{\gamma\}]^{P_2} \vdash [\{\alpha\} \backslash \beta / \{\gamma\}]^{P_1 P_2}$
ID$^{\mathbf{l}}$. $C^{P_1}[\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^{P_2} \vdash [\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^{P_1 P_2}$
ΩD$^{\mathbf{l}}$. $[\{\alpha_1, C^*, \alpha_2\} \backslash \beta / \{\gamma\}]^P \vdash [\{\alpha_1, \alpha_2\} \backslash \beta / \{\gamma\}]^P$
DD$^{\mathbf{l}}$ as **D$^{\mathbf{l}}$** in the CDG calculus.
The order of elements in dispersed and choice iterations is irrelevant.

It is not difficult to simulate the dispersed iteration CDG through choice iteration CDG. Both are analyzed in polynomial time. As it concerns their weak generative power, both are conservative extensions of the CDG.

3 Incremental Learning

Learning. With every grammar $G \in \mathcal{C}$ is related an `observation set` $\Phi(G)$ of G. This may be the generated language $L(G)$ or an image of the constituent or dependency structures generated by G. Below we call `training sequence` for G an enumeration of $\Phi(G)$. An algorithm A is an `inference algorithm` for $\mathcal{C}$ if, for every grammar $G \in \mathcal{C}$, A applies to its training sequences σ of $\Phi(G)$ and, for every initial subsequence $\sigma[i] = \{s_1, \ldots, s_i\}$ of σ, it returns a `hypothetical grammar` $A(\sigma[i]) \in \mathcal{C}$. A `learns` a `target grammar` $G \in \mathcal{C}$ if on any training sequence σ for G A stabilizes on a grammar $\mathcal{A}(\sigma[T]) \equiv G$.[4] The grammar $\lim_{i \to \infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ returned at the stabilization step is the `limit grammar`. A `learns` $\mathcal{C}$ if it learns every grammar in $\mathcal{C}$. $\mathcal{C}$ is `learnable` if there is an inference algorithm learning $\mathcal{C}$.

[2] Both are used in the flat type expressions of the `compacted` CDG in [9] designed for large scale wide scope grammars.
[3] We suppose that $[\{\} \backslash \beta] = [\beta]$.
[4] $\mathcal{A}$ `stabilizes` on σ on step T means that T is the minimal number t for which there is no $t_1 > t$ such that $\mathcal{A}(\sigma[t_1]) \neq \mathcal{A}(\sigma[t])$.

Incremental Learning. Selecting a partial order $\preceq_{\mathcal{C}}$ on the grammars of a class $\mathcal{C}$ compatible with the inclusion of observation sets ($G \preceq_{\mathcal{C}} G' \Rightarrow \Phi(G) \subseteq \Phi(G')$), we can define the following notion of incremental learning algorithm on $\mathcal{C}$.

Definition 4. *Let $\mathcal{A}$ be an inference algorithm for $\mathcal{C}$ and σ be a training sequence for a grammar G.*
1. $\mathcal{A}$ is `monotonic` *on σ if $\mathcal{A}(\sigma[i]) \preceq_{\mathcal{C}} \mathcal{A}(\sigma[j])$ for all $i \leq j$.*
2. $\mathcal{A}$ is `faithful` *on σ if $\Phi(\mathcal{A}(\sigma[i])) \subseteq \Phi(G)$ for all i.*
3. $\mathcal{A}$ is `expansive` *(or* `consistent`*) on σ if $\sigma[i] \subseteq \Phi(\mathcal{A}(\sigma[i]))$ for all i.*
For $G_1, G_2 \in \mathcal{C}$, $G_1 \equiv_s G_2$ iff $\Phi(G_1) = \Phi(G_2)$.

Theorem 1. *Let σ be a training sequence for a grammar G. If an inference algorithm $\mathcal{A}$ is monotonic, faithful, and expansive on σ, and if $\mathcal{A}$ stabilizes on σ then $\lim_{i\to\infty} \mathcal{A}(\sigma[i]) \equiv_s G$.*

Proof. Indeed, stabilization implies that $\lim_{i\to\infty} \mathcal{A}(\sigma[i]) = \mathcal{A}(\sigma[T])$ for some T. Then $\Phi(\mathcal{A}(\sigma[T])) \subseteq \Phi(G)$ because of faithfulness. At the same time, by expansiveness and monotonicity, $\Phi(G) = \sigma = \bigcup_{i=1}^{\infty} \sigma[i] \subseteq \bigcup_{i=1}^{\infty} \Phi(\mathcal{A}(\sigma[i])) \subseteq \bigcup_{i=1}^{T} \Phi(\mathcal{A}(\sigma[i]))$ $\subseteq \Phi(\mathcal{A}(\sigma[T]))$.

4 Incremental Learning of Dispersed Iteration

In paper [2], we present an incremental learning algorithm for K-**star revealing** CDG which do not distinguish between the dependencies consecutively repeated at least K times and those consecutively repeated any number of times.

Below we change the definition of K-**star revealing** in order to adapt it to the dispersed iteration. We use $\Delta(G)$ as the observation set $\Phi(G)$. So the limit grammar will be `strongly` equivalent to the target grammar G. The notion of incrementality we use is based on a partial "flexibility" order $\preceq_{disp}$ on dispersed iteration CDG. Basically, this PO corresponds to grammar expansion in the sense that $G_1 \preceq_{disp} G_2$ means that G_2 defines no less dependency structures than G_1 and at least as precise dependency structures as G_1. It is the reflexive-transitive closure of the following preorder $<_{disp}$.

Definition 5. *1. All occurrences of a dependency name d on the left can be replaced by a single left dispersed iteration of d:*
$[\{fl_1^*, \ldots, fl_p^*\} \backslash l_m \backslash \cdots \backslash d \backslash l_i \backslash \cdots \backslash d \backslash \cdots \backslash l_1 \backslash g / r_1 \cdots / r_n / \{fr_1^*, \ldots, fr_q^*\}]^P$
$<_{disp}$ $[\{fl_1^*, \ldots, fl_p^*, d^*\} \backslash l_m \backslash \cdots \backslash l_1 \backslash g / r_1 \cdots / r_n / \{fr_1^*, \ldots, fr_q^*\}]^P$.
2. Symmetrically, all occurrences of a dependency name d on the right can be replaced by a single right dispersed iteration of d.
3. $\tau <_{disp} \tau'$ for sets of types τ, τ', if either:
(i) $\tau' = \tau \cup \{t\}$ for a type $t \notin \tau$ or
(ii) $\tau = \tau_0 \cup \{t'\}$ and $\tau' = \tau_0 \cup \{t''\}$
for a set of types τ_0 and some types t', t'' such that $t' <_{disp} t''$.

4. $\lambda <_{disp} \lambda'$ *for two type assignments* λ *and* λ', *if* $\lambda(w') <_{disp} \lambda'(w')$ *for a word* w' *and* $\lambda(w) = \lambda'(w)$ *for all words* $w \neq w'$.
5. $\preceq_{disp}$ *is the PO which is the reflexive-transitive closure of the preorder* $<_{disp}$.

It is not difficult to prove that the expressive power of CDG monotonically grows with respect to this PO.

Proposition 1. *Let* G_1 *and* G_2 *be two CDG such that* $G_1 \preceq_{disp} G_2$. *Then* $\Delta(G_1) \subseteq \Delta(G_2)$ *and* $\mathcal{L}(G_1) \subseteq \mathcal{L}(G_2)$.

Below we adapt to the dispersed iteration the basic definitions from [2].

Definition 6
`Vicinity`*: Let* D *be a DS in which an occurrence of a word* w *has :*
the incoming local dependency h *(or the axiom* S*), the left projective dependencies or anchors* $l_k, \ldots, l_1$ *(in this order), the right projective dependencies or anchors* $r_1, \ldots, r_m$ *(in this order), and discontinuous dependencies* $p_1(d_1), \ldots, p_n(d_n)$, *where* $p_1, \ldots, p_n$ *are polarities and* $d_1, \ldots, d_n \in \mathbf{V}$ *are valency names. Then the* `vicinity` *of* w *in* D *is the type*

$$V(w, D) = [l_1 \backslash \ldots \backslash l_k \backslash h / r_m / \ldots / r_1]^P,$$

in which P *is a permutation of* $p_1(d_1), \ldots, p_n(d_n)$ *in a standard lexicographical order, for instance, compatible with the polarity order* $\nwarrow < \searrow < \swarrow < \nearrow$.

Multiple occurrences of a dependency name d in a vicinity $V(w, D)$ correspond to a dispersed iteration $\{f_1^*, \ldots, f_p^*, d^*\}$ in the type assigned to w in a proof of D. For instance, in the DS in Fig. 4 the vicinity of the participle *ramenée* is $[aux - a / l - obj]^{\nwarrow clit-a-obj}$. In this DS, $[pred \backslash circ \backslash circ \backslash circ \backslash S / a - obj]$ is the vicinity of the verb *fallait*. This vicinity may be provided by the type assignment $fallait \mapsto [\{circ^*\} \backslash pred \backslash S / a - obj]$.

Definition 7. *Let* $K > 1$ *be an integer. We define a CDG* $\mathcal{C}^K_{disp}(G)$, *the* `dispersed` K`-star-generalization` *of* G, *by recursively adding for every word* w *and every dependency name* d *the types*

$$[\{fl_1^*, \ldots, fl_p^*, d^*\} \backslash l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \ldots, fr_q^*\}]^P$$

and

$$[\{fl_1^*, \ldots, fl_p^*\} \backslash l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \ldots, fr_q^*\}]^P$$

when w *has a type assignment* $w \mapsto t$, *where*

$t = [\{fl_1^*, \ldots, fl_p^*\} \backslash l_m \backslash \cdots d \backslash l_i \backslash \cdots d \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n / \{fr_1^*, \ldots, fr_q^*\}]^P$,
and t *has at least* K **occurrences of** d *as left arguments. Symmetrically, we also add the corresponding types if the* K *occurrences of* d *appear in the right part of* t.

For instance, with $K = 2$, for the type $[\{x^*\} \backslash a \backslash b \backslash a \backslash S / a]$, will be added the type $[\{a^*, x^*\} \backslash b \backslash S / a]$. The size of $\mathcal{C}^K_{disp}(G)$ can be exponential with respect to the size of G.

Definition 8. *Let* $K > 1$ *be an integer. CDG* G *is* **dispersed** K**-star revealing** *if* $\mathcal{C}^K_{disp}(G) \equiv_s G$

Algorithm $\mathbf{TGE}^{(K)}_{disp}$ (type-generalize-expand):
Input: $\sigma[i]$ (σ being a training sequence).
Output: CDG $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$.
let $G_H = (W_H, \mathbf{C}_H, \mathbf{V}_H, S, \lambda_H)$ where
$W_H := \emptyset$; $\mathbf{C}_H := \{S\}$; $\lambda_H := \emptyset$; $k := 0$
(loop) **for** $i \geq 0$ //Infinite loop on σ
let $\sigma[i+1] = \sigma[i] \cdot D$;
let $(x, E) = D$;
(loop) **for every** $w \in x$
$W_H := W_H \cup \{w\}$;
let $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$
let $LT := \{l_1\} \cup \cdots \cup \{l_m\}$
let $LF := \{d : d \in LT, card(\{i : 1 \leq i \leq m, l_i = d\}) \geq K\}$
let $RT := \{r_1\} \cup \cdots \cup \{r_n\}$
let $RF := \{d : d \in RT, card(\{i : 1 \leq i \leq n, r_i = d\}) \geq K\}$
let $t_w := [\{lf_1^*, \ldots, lf_p^*\} \backslash l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'} / \{rf_1^*, \ldots, rf_q^*\}]^P$
where $\{lf_1, \ldots, lf_p\} = LF$, $\{rf_1, \ldots, rf_q\} = RF$,
where $l'_{m'}, \ldots, l'_1$ is the sublist of $l_m, \ldots, l_1$ without elements in LF
where $r'_1, \ldots, r'_{n'}$ is the sublist of $r_1, \ldots, r_n$ without elements in RF.
$\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // `expansion`
end end

Fig. 5. *Inference algorithm* $\mathbf{TGE}^{(K)}_{disp}$

For instance, if $G(t)$ is the CDG $A \mapsto [a], B \mapsto [b], C \mapsto t$, where t is a type, then we can prove that:

- $G([\{a^*\}\backslash b \backslash S / a / b])$, $G([a \backslash b \backslash S])$ and $G([a \backslash b \backslash S / \{a, b\}])$ are all the three dispersed 2-star revealing,
- neither of $G([a \backslash a \backslash S])$, $G([a \backslash b \backslash a \backslash S])$, $G([a \backslash S / b / b])$ is dispersed 2-star revealing.

By Definition 7, we may suppose that no subtype may have K left or K right occurrences in a type of a dispersed K-star revealing grammar.

Theorem 2. *The class* $\mathcal{CDG}^{K \to *}_{disp}$ *of dispersed* K*-star revealing CDG is (incrementally) learnable from DS.*

A proof of this theorem is shown in the Appendix. Algorithm $\mathbf{TGE}^{(K)}_{disp}$ learning the dispersed K-star revealing CDG is shown in Fig. 5.

Remark 1. This algorithm transforms every observed words' vicinity v in DS belonging to $\sigma[i]$ into the least type t such that $v \preceq_{disp} t$. The grammar $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$ is computed in linear time with respect to $|\sigma[i]|$. This algorithm may be easily transformed into a square time algorithm computing in the limit the minimal length grammar among all dispersed K-star revealing grammars strongly equivalent to the target grammar.

5 Incremental Learning of Choice Iteration

In this section, we show an algorithm incrementally learning choice iteration CDG. For the rest of this section, every iterated type d^* is considered as the choice of a unique iterated type (written $(d)^*$). Above this, the lists of types in choice iterations will be considered as sets of types. Thus $(a|b)^* = (b|a)^* = (a|b|a)^*$. The incremental algorithm of [2] may diverge when applied to a choice iteration. For instance, for the CDG $A \mapsto [a]$, $B \mapsto [b]$, $C \mapsto [S/(a|b)^*]$, this algorithm will compute for C all the iteration-less types corresponding to all possible alternations of a and b: $[S]$, $[S/a]$, $[S/b]$ $[S/a/b]$, $[S/b/a]$, $[S/a/b/a]$, $[S/b/a/b]$, etc. It will also compute all possible alternations of a^* and b^*: $[S]$, $[S/a^*]$, $[S/b^*]$ $[S/a^*/b^*]$, $[S/b^*/a^*]$, $[S/a^*/b^*/a^*]$, $[S/b^*/a^*/b^*]$, etc. Thus, the algorithm will diverge. By the way, the algorithm $\mathbf{TGE}^{(K)}_{disp}$ of the preceding section will converge on this CDG and compute a grammar strongly equivalent to the following one: $A \mapsto [a]$, $B \mapsto [b]$, $C \mapsto [S/\{a^*, b^*\}]$ (equivalent to the original CDG). Unfortunately, $\mathbf{TGE}^{(K)}_{disp}$ will not converge for all choice iteration CDG. So we will look for generalizations specific for the choice iteration. In particular, one should not distinguish between several *consecutive* choice iterations and their *union*. For instance, x^* followed by $(y|z)^*$ should be "equivalent" to $(x|y|z)^*$. When applied to the vicinity $[S/a/c/b/b/a]$, in the case of $K = 2$, the learning algorithm should generalize it to $[S/a^*/c/b^*/a^*]$ and then the segment $/b^*/a^*$ should be replaced by a single choice iteration giving the type $[S/a^*/c/(a|b)^*]$.

Incremental learning of choice iteration is based on a PO $\preceq_{ch}$ on CDG, which is the reflexive-transitive closure of the following preorder $<_{ch}$.

Definition 9

1. *All occurrences of a dependency name d on the left (or on the right) can be replaced by a single iteration of d^*:*
   $[l_m \backslash \cdots \backslash d \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P <_{ch} [l_m \backslash \cdots \backslash d^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   (similar on the right).
2. *Two consecutive choice iterations can be merged:*
   $[l_m \backslash \cdots \backslash (a_1|\cdots|a_p)^* \backslash (b_1|\cdots|b_q)^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   $<_{ch} [l_m \backslash \cdots \backslash (a_1|\cdots|a_p|b_1|\cdots|b_q)^* \backslash \cdots l_1 \backslash g / r_1 \cdots / r_n]^P$
   (similar on the right).
3. *Same as 3–5 in Definition 5.*

Again, it is not difficult to prove that the expressive power of choice iteration CDG monotonically grows with respect to $\preceq_{ch}$.

Definition 10. *Let $K > 1$ be an integer. We define a CDG $\mathcal{C}^K_{ch}(G)$, the* `choice K-star-generalization` *of G, by recursively adding for every word w and every dependency name d:*

1. *the type $[l_m \backslash \cdots \backslash d^* \backslash \cdots \backslash d^* \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$ (every occurrence of d on the left is replaced by d^*) when w has a type assignment $w \mapsto t$, where $t = [l_m \backslash \cdots \backslash d \backslash \cdots \backslash d \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$, and d appears at least K times in $l_m, \ldots, d, \ldots, l_1$ or d appears at least one time in a choice iteration of $l_m, \ldots, l_1$,*

2. *the type* $[l_m\backslash\cdots\backslash(x_1|\cdots|x_p|y_1|\cdots|y_q)^*\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P$ *when* w *has a type assignment* $w \mapsto t$*, where*
$$t = [l_m\backslash\cdots\backslash(x_1|\cdots|x_p)^*\backslash(y_1|\cdots|y_q)^*\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P,$$
3. *the type* $[l_m\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P$ *when* w *has a type assignment* $w \mapsto t$*, where* $t = [l_m\backslash\cdots\backslash d^*\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P$,
4. *the type* $[l_m\backslash\cdots\backslash(x_1|\cdots|x_p)^*\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P$ *when* w $w \mapsto t$*, where* $t = [l_m\backslash\cdots\backslash(d|x_1|\cdots|x_p)^*\backslash\cdots\backslash l_1\backslash h/r_1/\cdots/r_n]^P$ *and* $p > 0$.

Symmetrically, are also added the corresponding types for the right part of t.

For instance, when $K = 2$, for the type $[b\backslash a\backslash b\backslash a\backslash S/a]$, will be added the type $[b^*\backslash a^*\backslash b^*\backslash a^*\backslash S/a]$. Then will also be added $[(a|b)^*\backslash b^*\backslash a^*\backslash S/a], [b^*\backslash(a|b)^*\backslash a^*\backslash S/a]$, $\ldots$, $[(a|b)^*\backslash S/a]$. Finally, the single iterations a^* or b^* will be erased or $(a|b)^*$ will be transformed into a^* or b^*. The size of $\mathcal{C}_{ch}^K(G)$ can be exponential with respect to the size of G.

Definition 11. *Let* $K > 1$ *be an integer. CDG* G *is* **choice** K**-star revealing** *if* $\mathcal{C}_{ch}^K(G) \equiv_s G$

Remark 2. The generalization induced by points (3) and (4) does not introduce new DS even when G is not **choice** K**-star revealing**. These points serve only to simplify the proof of learnability.

For instance, if $G(t)$ is the CDG with the assignments: $A \mapsto [a], B \mapsto [b], C \mapsto t$, where t is a type, then we can prove that:

- $G([a^*\backslash b\backslash S/a/b])$ and $G([a^*\backslash b\backslash a^*\backslash S])$ are both choice 2-star revealing,
- $G([a\backslash a\backslash S])$, $G([a\backslash b\backslash a\backslash S])$ and $G([a^*\backslash b^*\backslash S])$ are not choice 2-star revealing.

Now, without loss of generality we may suppose that in every type of a choice K-star revealing grammar, a subtype d is used at most $K-1$ times on the left and $K-1$ on the right. Besides this, there shouldn't be two consecutive choice iterations.

Theorem 3. *The class* $\mathcal{CDG}_{ch}^{K\to *}$ *of choice* K*-star revealing CDG is (incrementally) learnable from DS.*

A proof of this theorem is shown in the Appendix.

Remark 3. In fact, a similar proof shows that the CDG in $\mathcal{CDG}_{disp}^{K\to *}$ and $\mathcal{CDG}_{ch}^{K\to *}$ with potentials of a uniformly bounded length are learnable from strings in the sense of Gold for every K. Meanwhile, Theorems 2 and 3 are stronger because they provide incremental learning algorithms.

Algorithm $\mathbf{TGE}_{ch}^{(K)}$ learning the choice K-star revealing CDG is shown in Fig. 6.

Remark 4. This algorithm transforms every observed words' vicinity v in DS belonging to $\sigma[i]$ into the least type t such that $v \preceq_{ch} t$. The grammar $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$ is computed in linear time with respect to $|\sigma[i]|$. This algorithm too may be transformed into a square time algorithm computing in the limit the minimal length grammar among all choice K-star revealing grammars strongly equivalent to the target grammar.

Algorithm $\mathbf{TGE}_{ch}^{(K)}$ (type-generalize-expand):
Input: $\sigma[i]$ (σ being a training sequence).
Output: CDG $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$.
let $G_H = (W_H, \mathbf{C}_H, \mathbf{V}_H, S, \lambda_H)$ where
$W_H := \emptyset$; $\mathbf{C}_H := \{S\}$; $\lambda_H := \emptyset$; $k := 0$
(loop) **for** $i \geq 0$ //Infinite loop on σ
let $\sigma[i+1] = \sigma[i] \cdot D$;
let $(x, E) = D$;
(loop) **for every** $w \in x$
$W_H := W_H \cup \{w\}$;
let $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$
let $LT := \{l_1\} \cup \cdots \cup \{l_m\}$
let $LI := \{d : d \in LT, card(\{i : 1 \leq i \leq m, l_i = d\}) \geq K\}$
let $RT := \{r_1\} \cup \cdots \cup \{r_n\}$
let $RI := \{d : d \in RT, card(\{i : 1 \leq i \leq n, r_i = d\}) \geq K\}$
for $1 \leq i \leq m$, **let** $l'_i = l_i$ **if** $l_i \notin LI$ **else** $l'_i = l^*_i$
for $1 \leq i \leq n$, **let** $r'_i = r_i$ **if** $r_i \notin RI$ **else** $r'_i = r^*_i$
let $t'_w := [l'_m \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_n]^P$ // (end of) step I
(loop) **while** $\exists i$ such that $l'_i = (x_1 | \cdots | x_p)^*$ and $l'_{i+1} = (y_1 | \cdots | y_q)^*$
$l'_i := (x_1 | \cdots | x_p | y_1 | \cdots | y_q)^*$ and remove l'_{i+1}
(loop) **while** $\exists i$ such that $r'_i = (x_1 | \cdots | x_p)^*$ and $r'_{i+1} = (y_1 | \cdots | y_q)^*$
$r'_i := (x_1 | \cdots | x_p | y_1 | \cdots | y_q)^*$ and remove r'_{i+1}
let $t_w = [l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'}]$ // (end of) step II
$\lambda_H(w) := \lambda_H(w) \cup \{t_w\}$; // expansion
end end

Fig. 6. *Inference algorithm* $\mathbf{TGE}_{ch}^{(K)}$

6 Conclusion

In this paper, we propose two different ways of defining repeatable dependencies through iteration: the dispersed iteration and the choice iteration. Both conform to the linguistic Principle of repeatable dependencies. We adapt the K-star-revealing condition in [2] to these new models and show that in both cases the CDG satisfying the K-star-revealing are incrementally learnable in the limit from dependency structures. This kind of grammatical inference directly corresponds to the problem of deterministic extraction of a dependency grammar from a dependency treebank.

In this paper, the dispersed and the choice iterations are considered separately. Considered together, they should be related as follows:

$$[l_1 \backslash \cdots \backslash l_m \backslash (d_1 | \cdots | d_k)^* \backslash l_{(m+1)} \backslash \cdots \backslash l_n \backslash \beta]^P \preceq [\{d_1^*, \ldots, d_k^*\} \backslash l_1 \backslash \cdots \backslash l_m \backslash l_{(m+1)} \backslash \cdots \backslash l_n \backslash \beta]^P.$$

Both constructions are necessary in application CDG. For instance, in English, the type $[(modif|attr)^* \backslash det \backslash obj / claus]$ is admissible for nouns in the syntactic role of a complement, whereas the type $[\{modif^*, attr^*\} \backslash det \backslash obj / claus]$ is not: it allows modifiers and attributes to precede the noun's determinant. At the same time, the type $[pred \backslash circ^* \backslash S / iobj / dobj / \{circ^*\}]$ is admissible for main ditransitive verbs. It allows circumstantials of a verb to occur in any position with respect to its direct and indirect complements. The next step should be to study the learnability of CDG using both primitives.

References

1. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)
2. Béchet, D., Dikovsky, A., Foret, A.: Two models of learning iterated dependencies. In: Proc. of the 15th Conference on Formal Grammar (FG 2010), Copenhagen, Denmark. LNCS (2010) (to appear), http://www.angl.hu-berlin.de/FG10/fg10_list_of_papers
3. Béchet, D., Dikovsky, A., Foret, A., Moreau, E.: On learning discontinuous dependencies from positive data. In: Proc. of the 9th Intern. Conf. Formal Grammar (FG 2004), Nancy, France, pp. 1–16 (2004)
4. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49, 431–454 (1990)
5. Dekhtyar, M., Dikovsky, A.: Generalized categorial dependency grammars. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 230–255. Springer, Heidelberg (2008)
6. Dekhtyar, M., Dikovsky, A., Karlov, B.: Iterated dependencies and kleene iteration. In: Proc. of the 15th Conference on Formal Grammar (FG 2010), Copenhagen, Denmark. LNCS (2010) (to appear), http://www.angl.hu-berlin.de/FG10/fg10_list_of_papers
7. Dikovsky, A.: Dependencies as categories. In: Recent Advances in Dependency Grammars, COLING 2004, pp. 90–97 (2004)
8. Dikovsky, A.: Multimodal categorial dependency grammars. In: Proc. of the 12th Conference on Formal Grammar, Dublin, Ireland, pp. 1–12 (2007)
9. Dikovsky, A.: Towards wide coverage categorial dependency grammars. In: Proc. of the ESSLLI 2009 Workshop on Parsing with Categorial Grammars. Book of Abstracts, Bordeaux, France (2009)
10. Gold, E.M.: Language identification in the limit. Information and Control 10, 447–474 (1967)
11. Kanazawa, M.: Learnable classes of categorial grammars. Studies in Logic, Language and Information, FoLLI & CSLI (1998)
12. Karlov, B.N.: Normal forms and automata for categorial dependency grammars. Vestnik Tverskogo Gosudarstvennogo Universiteta (Annals of Tver State University). Series: Applied Mathematics 35 (95), 23–43 (2008) (in russ.)
13. Mel'čuk, I.: Dependency Syntax. SUNY Press, Albany, NY (1988)
14. Shinohara, T.: Inductive inference of monotonic formal systems from positive data. New Generation Computing 8(4), 371–384 (1991)

A Appendix

Proof of Theorem 2

Lemma 1. *Let σ be a training sequence for a dispersed K-star revealing CDG G. Then for all i :* $\mathbf{TGE}^{(K)}_{disp}(\sigma[i]) \preceq_{disp} \mathcal{C}^K_{disp}(G)$.

Proof. As G is dispersed K-star revealing, G and $\mathcal{C}^K_{disp}(G)$ are strongly equivalent. For each DS D in $\Delta(G)$ there is a type assignment in $\mathcal{C}^K_{disp}(G)$ to the words appearing in D, which "conforms to" D. More precisely, for a DS D, a word w and vicinity $V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P$, we choose the smallest type $t_{w,disp}$ among the

types assigned to w in $\mathcal{C}^K_{disp}(G)$, which yield the same DS D in $\Delta(\mathcal{C}^K_{disp}(G))$. $t_{w,disp}$ may be represented as:

$$[\{lf^*_1, \ldots, lf^*_p\}\backslash l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'} / \{rf^*_1, \ldots, rf^*_q\}]^{P'}$$

where P' is a permutation of P.
Let t_w denote the type computed by the algorithm for w in D.
We show that for this type $t_w \preceq_{disp} t_{w,disp}$ (modulo a potentials permutation).
- For each left dispersed iteration d^* in t_w, d occurs at least K times as a left argument in $V(w, D)$. Since $t_{w,disp}$ is minimal for D, d^* must be a member of left dispersed iterations.
- Every dependency name d' occurs less than K times as a left argument of t_w, and less than K times as a left argument in $V(w, D)$. In $t_{w,disp}$, either d' appears the same number of times as a left argument in the same relative positions, or d'^* is a member of left dispersed iterations.
This proves that $t_w \preceq_{disp} t_{w,disp}$.

Lemma 2. *The inference algorithm* $\mathbf{TGE}^{(K)}_{disp}$ *is monotonic, faithful and expansive on every training sequence* σ *of a dispersed K-star revealing CDG.*

Proof. By definition, the algorithm $\mathbf{TGE}^{(K)}_{disp}$ is `monotonic` (the lexicon is always extended). It is `expansive` because for $\sigma[i]$, we add types to the grammar that are based on the vicinities of the words of $\sigma[i]$. Thus, $\sigma[i] \subseteq \Delta(\mathbf{TGE}^{(K)}_{disp}(\sigma[i]))$. To prove that $\mathbf{TGE}^{(K)}_{disp}$ is `faithful` for $\sigma[i]$ of $\Delta(G) = \Delta(\mathcal{C}^K(G))$, we have to remark that $\mathbf{TGE}^{(K)}_{disp}(\sigma[i]) \preceq_{disp} \mathcal{C}^K_{disp}(G)$, using lemma 1.

Lemma 3. *The inference algorithm* $\mathbf{TGE}^{(K)}_{disp}$ *stabilizes on every training sequence* σ *of a dispersed K-star revealing CDG.*

Proof. G and $\mathcal{C}^K_{disp}(G)$ have a finite number of types and of dependency names. $\mathbf{TGE}^{(K)}_{disp}(\sigma[i])$ involves a subset of these names, and the same dependency name may be used as argument at most $K-1$ times on the left, and $K-1$ times on the right; there is a finite number of sets of dispersed types; therefore the algorithm must stabilize.

Proof of Theorem 3

Lemma 4. *Let* σ *be a training sequence for a choice K-star revealing CDG* G*. Then for all* i *:* $\mathbf{TGE}^{(K)}_{ch}(\sigma[i]) \preceq_{ch} \mathcal{C}^K_{ch}(G)$.

Proof. As G is choice K-star revealing, G and $\mathcal{C}^K_{ch}(G)$ are strongly equivalent. For each DS D in $\Delta(G)$ there is a type assignment in $\mathcal{C}^K_{ch}(G)$ to the words appearing in D, which "conforms to" D.

More precisely, for a DS D, a word w and its vicinity

$$V(w, D) = [l_m \backslash \cdots \backslash l_1 \backslash h / r_1 / \cdots / r_n]^P,$$

we choose the smallest type $t_{w,ch}$ among the types assigned to w in $\mathcal{C}^K_{ch}(G)$, which yield the same DS D in $\Delta(\mathcal{C}^K_{ch}(G))$ and to which the rules (1) and (2) in Definition 10 are not applied. $t_{w,disp}$ may be represented as:

$$[l'_{m'} \backslash \cdots \backslash l'_1 \backslash h / r'_1 / \cdots / r'_{n'}]^{P'}$$

where P' is a permutation of P.

Let t'_w and t_w denote the types computed by the algorithm for w in D at the end of step I and, respectively, of step II.

We show that $t'_w \preceq_{ch} t_{w,ch}$ and $t_w \preceq_{ch} t_{w,ch}$.
- For each left iteration d^* in t'_w, d occurs at least K times as a left argument in $V(w, D)$. Since $t_{w,ch}$ is non-extensible by the rules (1) and (2), there is a minimal length choice iteration in it containing d. If d^* in t'_w corresponds in t_w to a choice iteration $C = (d|...)^*$, then C corresponds to the largest sublist of iterations in t'_w including this d^*. Therefore, it must appear in $t_{w,ch}$ in a choice iteration larger or equal to C.
- Every dependency name d which is a left argument subtype in t'_w or in t_w, occurs less than K times as a left argument in t'_w and t_w and less than K times as a left dependency in $V(w, D)$. Therefore, d appears in $t_{w,ch}$ the same number of times as a left argument subtype or in a minimal length choice iterations including d in the same relative positions.
Thus $t_w \preceq_{ch} t_{w,ch}$.

Lemma 5. *The inference algorithm* $\mathbf{TGE}_{ch}^{(K)}$ *is monotonic, faithful and expansive on every training sequence* σ *of a choice* K*-star revealing CDG.*

Proof. By definition, the algorithm $\mathbf{TGE}_{ch}^{(K)}$ is `monotonic` (the lexicon is always extended). It is `expansive` because for $\sigma[i]$, the types added to the grammar are calculated from and are compatible with the vicinities of the words of $\sigma[i]$. Thus, $\sigma[i] \subseteq \Delta(\mathbf{TGE}_{ch}^{(K)}(\sigma[i]))$. To prove that $\mathbf{TGE}_{ch}^{(K)}$ is `faithful` for $\sigma[i]$ of $\Delta(G) = \Delta(\mathcal{C}_{ch}^{K}(G))$, it suffices to remark that by lemma 4, $\mathbf{TGE}_{ch}^{(K)}(\sigma[i]) \preceq_{ch} \mathcal{C}_{ch}^{K}(G)$.

Lemma 6. *The inference algorithm* $\mathbf{TGE}_{ch}^{(K)}$ *stabilizes on every training sequence* σ *of a choice* K*-star revealing CDG.*

Proof. There is a finite set Σ_{ch} of types and of dependency names in $\mathcal{C}_{ch}^{K}(G)$. The grammar $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$ uses a subset of this set. Let t denote a type assigned by the lexicon of $\mathbf{TGE}_{ch}^{(K)}(\sigma[i])$. The size of the multiset of non-repeatable dependency names occurring in t is smaller than $2K \times |\Sigma_{ch}|$ since a dependency name may be used as an argument subtype at most $K-1$ times on the left, and at most $K-1$ times on the right. Any choice iteration has at most $|\Sigma_{ch}|$ elements so the number of choice iteration is finite. No choice iteration may occur next to another choice iteration. Therefore the number of types is finite and the algorithm must stabilize.

Closure Properties of Minimalist Derivation Tree Languages

Thomas Graf

Department of Linguistics
University of California, Los Angeles
tgraf@ucla.edu
http://tgraf.bol.ucla.edu

Abstract. Recently, the question has been raised whether the derivation tree languages of Minimalist grammars (MGs; [14, 16]) are closed under intersection with regular tree languages [4, 5]. Using a variation of a proof technique devised by Thatcher [17], I show that even though closure under intersection does not obtain, it holds for every MG and regular tree language that their intersection is identical to the derivation tree language of some MG *modulo* category labels. It immediately follows that the same closure property holds with respect to union, relative complement, and certain kinds of linear transductions. Moreover, enriching MGs with the ability to put regular constraints on the shape of their derivation trees does not increase the formalism's weak generative capacity. This makes it straightforward to implement numerous linguistically motivated constraints on the Move operation.

Keywords: Minimalist Grammars, Derivation Tree Languages, Closure Properties, Regular Tree Languages, Derivational Constraints.

Introduction

Minimalist grammars (MGs) were introduced in [14] as a formalism inspired by Chomsky's Minimalist Program [1]. Over the years, MGs have been enriched with various tools from the syntactic literature (e.g. phases and persistent features [15]), most of which do not increase the framework's weak generative capacity. One recent extension was proposed in my own work ([4, 5]; I will refer to these papers in the third person): the addition of reference-set constraints, which introduce a notion of optimality to the system. Graf proposes to model these constraints by linear tree transductions mapping the set of derivation trees of an MG to its subset of optimal derivation trees. He concludes that while the specific constraints he implements do not increase the weak generative capacity of MGs, the result carries over to arbitrary reference-set constraints definable by linear tree transductions only if the class of derivation tree languages is closed under intersection with regular tree languages — an open problem.

In this paper, I show that even though Minimalist derivation tree languages are not closed under intersection with regular tree languages, it holds for every Minimalist derivation tree language and regular tree language that their

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 96–111, 2011.

intersection is a projection of a Minimalist derivation tree language. I call this property *p-closure* under intersection with regular tree languages. While this p-closure result is already sufficient for Graf's purposes, it turns out that Minimalist derivation tree languages enjoy several more p-closure properties that make them appear very similar to regular tree languages. The p-closure properties of MGs also entail that their weak generative capacity is unaffected by the addition of regular control, which proves useful in the implementation of syntactic constraints such as islandhood, phases and Relativized Minimality.

The paper is laid out as follows: The preliminaries section covers, besides the definition of MGs, mundane topics such as tree languages and tree automata and the new yet crucial notion of p-closure. I then proceed to define Minimalist derivation tree languages and establish some of their basic properties (Sec. 2). This is followed up by a detailed investigation of their p-closure properties in Sec. 3, the greatest part of which is devoted to showing p-closure under intersection with regular tree languages. In the last part of this paper, I define MGs with regular control as a formalism with the same weak generative capacity as standard MGs, and I sketch some potential linguistic applications.

1 Preliminaries and Notation

In this section, I briefly introduce tree languages, tree automata, Minimalist grammars, and the notion of p-closure, which will be of great importance in this paper. As usual, $\mathbb{N}$ denotes the set of non-negative integers. A *tree domain* is a finite subset D of $\mathbb{N}^*$ such that, for $w \in \mathbb{N}^*$ and $j \in \mathbb{N}$, $wj \in D$ implies both $w \in D$ and $wi \in D$ for all $i < j$. Every $n \in D$ is called a *node*. Given nodes $m, n \in D$, m *immediately dominates* n iff $n = mi$, $i \in \mathbb{N}$. In this case we also say m is the mother of n, or conversely, n is a daughter of m. The transitive closure of the immediate dominance relation is called *dominance*. A node that does not dominate any other nodes is a *leaf*, and the unique node that isn't dominated by any nodes is called the *root*.

Now let Σ be a *ranked* alphabet, i.e. every $\sigma \in \Sigma$ has a unique non-negative *rank* (*arity*); $\Sigma^{(n)}$ is the set of all n-ary symbols in Σ. A Σ-tree is a pair $T := \langle D, \ell \rangle$, where D is a tree domain and $\ell : D \to \Sigma$ is a function assigning each node n a *label* drawn from Σ such that $\ell(n) \in \Sigma^{(d)}$ iff n has d daughters. Usually the alphabet will not be indicated in writing when it is irrelevant or can be inferred from the context. Sometimes trees will be given in functional notation such that $f(t_1, \ldots, t_n)$ is the tree where the root node is labeled f and immediately dominates trees $t_1, \ldots, t_n$. I denote by T_Σ the set of all trees such that for $n \geq 0$, $f(t_1, \ldots, t_n)$ is in T_Σ iff $f \in \Sigma^{(n)}$ and $t_i \in T_\Sigma$, $1 \leq i \leq n$. A *tree language* is some subset of T_Σ. It is *regular* (a *recognizable set*) iff it is recognized by a deterministic bottom-up tree automaton.

Definition 1. *A* deterministic bottom-up tree automaton *is a* 4*-tuple* $A := \langle \Sigma, Q, F, \delta \rangle$*, where*

- Σ *is a ranked alphabet,*
- Q *is a finite set of states (i.e. of unary symbols* $q \notin \Sigma$*),*

– $F \subseteq Q$ *is the set of final states,*
– $\delta\colon \left(\bigcup_{n\geq 0} Q^n \times \Sigma^{(n)}\right) \to Q$ *is the transition function.*

In the remainder of this paper, I will refer to deterministic bottom-up tree automata simply as (tree) automata. The transition function of a tree automaton can be extended to entire trees in the usual manner. A tree T, then, is *recognized* (*accepted*) by A iff $\delta(T) \in F$, and the language recognized by A is $L(A) := \{T \mid \delta(T) \in F\}$. At several points in the paper, I also mention tree transducers. All the reader needs to know about them is that they are the tree equivalent of string transducers, i.e. they rewrite input trees as output trees.

Given a tree T, a *treelet* t of T is a continuous substructure of T, that is to say, there is no node that does not belong to t yet both dominates a node of t and is dominated by a node of t. In the special case where t contains either all the nodes of T dominated by some node of t or none of them, we call t a *subtree* of T. A *projection* of a tree language is its image under a function $\hat{f}$ that is the pointwise extension of a surjective map $f : \Sigma \to \Omega$ between alphabets to tree languages. Thus projections are a particular kind of relabeling.

Definition 2 (P-Closure). *Given a class of languages $\mathcal{L}$ and an operation O, $\mathcal{L}$ is* p-closed *under O iff the result of applying O to some $L \in \mathcal{L}$ is a projection of some $L' \in \mathcal{L}$.*

We now turn to the definition of MGs, which mostly follows the chain-based exposition of [16] except that I allow for multiple final categories. This small extension has no effect on expressivity, as will be explained after the MG apparatus has been introduced.

Definition 3. *A* Minimalist grammar *is a 6-tuple $G := \langle \Sigma, Feat, F, Types, Lex, Op\rangle$, where*

– $\Sigma \neq \emptyset$ *is the alphabet,*
– *Feat is the union of a non-empty set base of* basic features *(also called* category features*) and its prefixed variants $\{=f \mid f \in base\}$, $\{+f \mid f \in base\}$, $\{-f \mid f \in base\}$ of* selector*,* licensor*, and* licensee *features, respectively,*
– $F \subseteq base$ *is a set of final categories,*
– $Types := \{::, :\}$ *distinguishes* lexical *from* derived *expressions,*
– *the lexicon Lex is a finite subset of $\Sigma^* \times \{::\} \times Feat^*$,*
– *and Op is the set of generating functions to be defined below.*

A chain *is a triple in $\Sigma^* \times Types \times Feat^*$, and C denotes the set of all chains (whence $Lex \subset C$). Non-empty sequences of chains will be referred to as* expressions*, the set of which is called E. I will usually drop the tuple brackets of chains and lexical items, but not those of expressions (the exception being depictions of derivation trees and the definitions of merge and move below).*

The set Op of generating functions consists of the operations merge and move. The operation merge: $(E \times E) \to E$ is the union of the following three functions, for $s, t \in \Sigma^*$, $\cdot \in Types$, $f \in base$, $\gamma \in Feat^*$, $\delta \in Feat^+$, and chains $\alpha_1, \ldots, \alpha_k$, $\iota_1, \ldots, \iota_k$, $0 \leq k, l$:

$$\frac{s :: =f\gamma \qquad t \cdot f, \iota_1, \ldots, \iota_k}{st : \gamma, \iota_1, \ldots, \iota_k}\ merge1$$

$$\frac{s : =f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f, \iota_1, \ldots, \iota_l}{ts : \gamma, \alpha_1, \ldots, \alpha_k, \iota_1, \ldots, \iota_l}\ merge2$$

$$\frac{s \cdot =f\gamma, \alpha_1, \ldots, \alpha_k \qquad t \cdot f\delta, \iota_1, \ldots, \iota_l}{s : \gamma, \alpha_1, \ldots, \alpha_k, t : \delta, \iota_1, \ldots, \iota_l}\ merge3$$

As the domains of all three functions are disjoint, their union is a function, too. Given one of the configurations above, one also says that s selects t.

The operation move: $E \to E$ *is the union of the two functions below, with the notation as above and the further assumption that all chains satisfy the Shortest Move Constraint (SMC), according to which no two chains in the domain of move display the same licensee feature* $-f$ *as their first feature.*

$$\frac{s : +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f, \alpha_{i+1}, \ldots, \alpha_k}{ts : \gamma, \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \alpha_k}\ move1$$

$$\frac{s : +f\gamma, \alpha_1, \ldots, \alpha_{i-1}, t : -f\delta, \alpha_{i+1}, \ldots, \alpha_k}{s : \gamma, \alpha_1, \ldots, \alpha_{i-1}, t : \delta, \alpha_{i+1}, \ldots, \alpha_k}\ move2$$

The language $L(G)$ *generated by* G *is the string component of the subset of the closure of the lexicon under the generating functions that contains all and only those expressions consisting of a single chain whose feature component is a single final category:* $L(G) := \{\sigma \mid \langle \sigma \cdot c \rangle \in closure(Lex, Op), \cdot \in Types, c \in F\}$.

Example 1. Let G be an MG defined by $F := \{c\}$ and the lexicon Lex below:

a :: a b :: =a =a +k a c :: =a c
a :: a −k

Two derivation trees of G are depicted in Fig. 1. □

As noted before, I slightly relax the MG formalism by allowing multiple final categories instead of just one. This is an innocent move. If a relaxed MG has final categories $c_1, \ldots, c_n$, we can turn it into a canonical MG by restricting the set of final categories to some new category c and introducing n new lexical items of the form $\varepsilon :: =c_i\ c$, where $1 \leq i \leq n$ and ε designates the empty string. The two grammars have virtually identical derivation tree languages with the only difference being the merger of a phonetically null c-head as the last step of every derivation for the canonical variant.

As for their weak generative capacity, MGs were shown in [6, 11, 12] to generate multiple context-free string languages, whence they constitute a mildly context-sensitive grammar formalism in the sense of [7]. The set of derivation trees of an MG, however, is a regular tree language and there is an effective procedure for obtaining the derived trees from their derivation trees — this holds even of the strictly more powerful class of MGs with unbounded copying [9, 10]. This is my main reason for considering derivation trees rather than derived trees (besides the central role of derivation trees in Graf's work): in contrast to the latter, they provide a unified, finite-state perspective on both MG variants; however, derived trees are investigated by Kobele in this volume, and he, too, proves closure under intersection with regular tree languages.

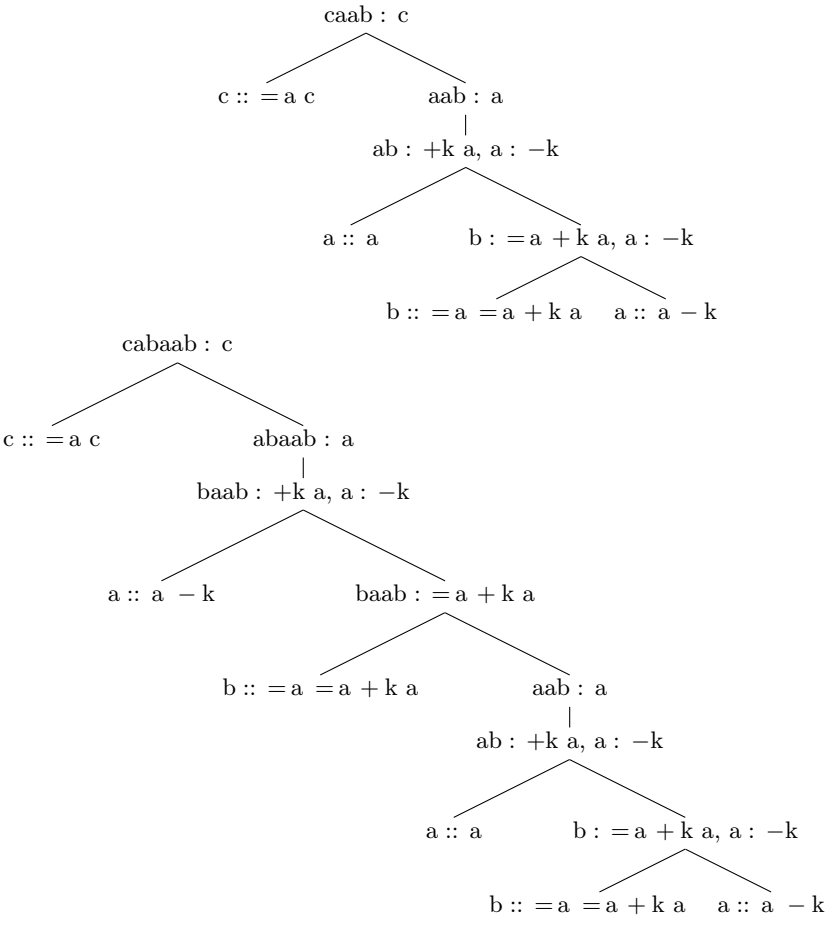


Fig. 1. Two derivation trees of the MG from example 1

2 Minimalist Derivation Tree Languages

The presentation of derivation trees in the MG literature varies somewhat with respect to what kind of labels are assigned to the interior nodes. The more common variant [cf. 16] is what I call string-annotated derivation trees, which were already encountered by the reader in the example at the end of the preliminaries section.

Definition 4 (String-annotated Derivation Tree Language). *Given an MG* $G := \langle \Sigma, \mathit{Feat}, F, \mathit{Types}, \mathit{Lex}, \mathit{Op} \rangle$*, its* string-annotated derivation tree language $\mathrm{sder}(G)$ *is the largest subset of* T_E *satisfying the following conditions:*

- *For every leaf node* n*,* $\ell(n) = \langle l \rangle$*,* $l \in \mathit{Lex}$*.*
- *For every binary branching node* n *immediately dominating* n' *and* n''*, it holds that* $\mathit{merge}(n', n'')$ *is defined and* $\ell(n) = \mathit{merge}(n', n'')$*.*

- *For every unary branching node n immediately dominating n', $move(n')$ is defined and $\ell(n) = move(n')$.*
- *For n the root node, $\ell(n) = \langle \sigma : c \rangle$, where $\sigma \in \Sigma^*$ and $c \in F$.*

String-annotated derivation tree languages aren't particularly interesting from the perspective of formal language theory as they are defined over an infinite alphabet ($L(G)$ is infinite in the general case, and so is E). Hence all work focusing on formal aspects of the derivation trees themselves [cf. 10] assume that the labels of the interior nodes indicate only which operations have taken place rather than the outputs of these operations at the respective stages of the derivations. This kind of derivation tree I refer to as Minimalist derivation tree.[1]

Definition 5 (Minimalist Derivation Tree Language). *Given an MG G, its* Minimalist derivation tree language $\mathrm{mder}(G)$ *is the set of trees obtained from* $\mathrm{sder}(G)$ *by the map μ relabeling all interior nodes by the corresponding operation:*

- $\mu(\langle l \rangle) = \langle l \rangle$, *where* $l \in Lex$
- $\mu(e(e_1, \ldots, e_n)) = op(\mu(e_1), \ldots, \mu(e_n))$, *where* $e, e_1, \ldots, e_n \in E$, $n \geq 1$, *and* op *is the unique operation in* Op *such that* $op(e_1, \ldots, e_n) = e$

Note that since the domains of all $op \in Op$ are pairwise disjoint, μ is indeed well-defined and a function. Also, whenever *merge* and *move* are used as labels, I will abbreviate them in the remainder of this paper by M and O, respectively.

It is fairly easy to see that Minimalist derivation tree languages are regular (consider the deterministic bottom-up tree automaton whose states correspond to the feature components of the labels of $\mathrm{sder}(G)$). In fact, they are a proper subset of the regular tree languages and are not closed under intersection with them.

Theorem 1. *The intersection of a Minimalist derivation tree language and a regular tree language may fail to be a Minimalist derivation tree language.*

Proof. Consider once again the Minimalist Grammar from example 1. Let $L_E \subseteq T_{Lex \cup \{M,O\}}$ contain all trees that have an even number of nodes, and only those. Then the result of intersecting the Minimalist derivation tree language of G with L_E is not a Minimalist derivation tree language. Among other things, it contains the Minimalist variant of the top tree in Fig. 1 on the facing page (i.e. with internal nodes replaced by M and O) but not the bottom one, yet they are built from the same lexical items and end in a final category, whence either both of them are in $closure(Lex, Op)$, or neither is. □

Corollary 1. *The image of a Minimalist derivation tree language under a linear transduction may fail to be a Minimalist derivation tree language, even if domain and co-domain of the transduction are identical.*

Proof. The intersection of two regular tree languages L and R is equivalent to the image of L under the diagonal of R, which is a linear transduction. □

[1] While it may not be in good style to have a technical term coalesce with a more colloquial one, the homophony is meant to highlight that I regard them as the canonical version of derivation trees for MGs.

3 P-Closure Properties

Theorem 1 and Cor. 1 notwithstanding, it does not take much ingenuity to realize that the counting condition in the example above can be enforced through the category and selection features of the feature calculus.

Example 2. Consider the lexical item $\mathrm{a} :: {=}\mathrm{a}\ {=}\mathrm{a} + \mathrm{k}\ \mathrm{a}$, and suppose that the derivation tree assembled so far contains an even number of nodes. The first selection feature causes the lexical item itself to be added to the derivation tree, plus the insertion of a Merge node. Thus the number of nodes in the derivation tree has increased by 2, which means that it is still even. Assume that at the next step a single lexical item is merged, increasing the count once more by 2. Then movement is triggered by the movement licensor feature, leading to the addition of only one node, so the tree now contains an odd number of nodes. At this point we only have to ensure that no lexical item of category c may be merged next in the derivation. The reason is that this would increase the counter only by 2, wherefore the number of nodes in the derivation tree would still be odd (and thus illicit), but nonetheless the derivation would be deemed well-formed, since we merged a lexical item of category c. In order to represent the arithmetic in the feature calculus, then, we have to replace $\mathrm{a} :: {=}\mathrm{a}\ {=}\mathrm{a} + \mathrm{k}\ \mathrm{a}$ by $\mathrm{a} :: {=}\mathrm{a_e}\ {=}\mathrm{a_o} + \mathrm{k}\ \mathrm{a_o}$ and $\mathrm{c} :: {=}\mathrm{a}\ \mathrm{c}$ by $\mathrm{c} :: {=}\mathrm{a_e}\ \mathrm{c}$. The fully refined grammar is given below.

$$
\begin{array}{lll}
\mathrm{a} :: \mathrm{a_o} & \mathrm{b} :: {=}\mathrm{a_o}\ {=}\mathrm{a_o} + \mathrm{k}\ \mathrm{a_e} & \mathrm{c} :: {=}\mathrm{a_e}\ \mathrm{c} \\
\mathrm{a} :: \mathrm{a_o} - \mathrm{k} & \mathrm{b} :: {=}\mathrm{a_o}\ {=}\mathrm{a_e} + \mathrm{k}\ \mathrm{a_o} & \\
 & \mathrm{b} :: {=}\mathrm{a_e}\ {=}\mathrm{a_o} + \mathrm{k}\ \mathrm{a_o} & \\
 & \mathrm{b} :: {=}\mathrm{a_e}\ {=}\mathrm{a_e} + \mathrm{k}\ \mathrm{a_e} &
\end{array}
$$

□

The strategy in the example above can easily be generalized to intersection with arbitrary regular sets by a slight modification of the technique employed by Thatcher in [17]. Thatcher realized that we may view the states of a tree automaton as an alphabet which the automaton "adds" to the original node labels. Therefore, one can simplify any recognizable set R over alphabet Σ to a degree where it can be described as the derivation tree language of a context-free grammar (ignoring the distinction between terminals and non-terminals), even though the class of the latter is properly included in the class of the former. One does so by first subscripting the symbols in Σ with the states of the canonical automaton accepting R, and subsequently recasting the transition rules in terms of rewriting rules — a transition $\sigma(q_1, \ldots, q_n) \to q$ corresponds to the set $\{\sigma_q \to \tau_{1,q_1}, \ldots, \tau_{n,q_n} \mid \sigma(\tau_1, \ldots, \tau_n)$ is a subtree of some tree of R and each τ_i is assigned state q_i in some $T \in R$, $1 \leq i \leq n\}$. Thatcher's strategy, however, cannot be used if the alphabet of our trees is fixed, as is the case with Minimalist derivation trees. Internal nodes have to be labeled by M or O, and adding subscripts to these symbols takes us out of the class of Minimalist derivation trees. Crucially, though, the internal nodes of a derivation tree are tied to the leaf nodes in a very peculiar way: every internal node denotes an operation, and this operation has to be triggered by a feature of some lexical item. So

while we may not suffix the states of an automaton to the internal node labels of a Minimalist derivation tree, we can still make them explicit by incorporating them into the feature calculus.

The first step in sharpening this rough sketch is the introduction of *slices*.

Definition 6 (Slice). *Given a Minimalist derivation tree* $T := \langle D, \ell \rangle$ *and lexical item* l *occurring in* T*, the* slice of l *is the pair* $\text{slice}(l) := \langle S, \ell \rangle$*,* $S \subseteq D$*, defined as follows:*

- $l \in S$,
- *if node* $n \in D$ *immediately dominates a node* $s \in S$*, then* $n \in S$ *iff the operation denoted by* $\ell(n)$ *erased a selector or licensor feature on* l*.*

The unique $n \in S$ *that isn't dominated by any* $n' \in S$ *is called the* slice root *of* l*.*

Intuitively, $\text{slice}(l)$ marks the subpart of the derivation that l has control over by virtue of its selector and licensor features. The next two lemmas jointly establish that for any derivation tree T, the set $\{\text{slice}(l) \mid l \text{ a lexical item in } T\}$ is a partition of T.

Lemma 1. *For every Minimalist derivation tree* $T := \langle D, \ell \rangle$ *and lexical item* l *in* T*,* $\text{slice}(l) := \langle S, \ell \rangle$ *is a unary branching treelet.*

Proof. That $\text{slice}(l)$ is unary branching follows immediately from the definition. So we only have to show that there is no node $n \notin S$ that both dominates and is dominated by nodes in $\text{slice}(l)$. Since the selector and licensor features of a lexical item l cannot be manipulated by any $o \in Op$ after l has already been selected by another lexical item, all these features of l have to occur before its category feature (which is unique). It is also clear that every licensee feature has to follow the category feature (*move* must be triggered by a licensor feature on some lexical item that is higher than l in the derivation tree, and only the category feature allows l to be selected by another lexical item so that the derivation can reach this higher point). Thus it holds for every lexical item that its feature sequence is an element of $\{=f, +f \mid f \in base\}^* \times base \times \{-f \mid f \in base\}^*$, proving the claim above. □

Lemma 2. *Given a Minimalist derivation tree* T*, every node of* T *belongs to some slice.*

Proof. Trivial. □

With these basic facts established, we turn to the algorithm that upon being given an MG $G := \langle \Sigma, Feat_G, F_G, Types, Lex_G, Op \rangle$ and regular language R will compute an MG G' such that $\text{mder}(G) \cap R$ is a projection of $\text{mder}(G')$. We begin by constructing the canonical automaton A_R for R (note that A_R is deterministic). In the next step, we pick a tree $T \in R \cap \text{mder}(G)$ and suffix each node n of T with the state q that A_R assigns to n when recognizing T (since A_R is deterministic, q is unique). After the second step has been applied to all members of $R \cap \text{mder}(G)$, the result will be a set with $R \cap \text{mder}(G)$ as its projection. So far, then, our procedure does not deviate at all from Thatcher's.

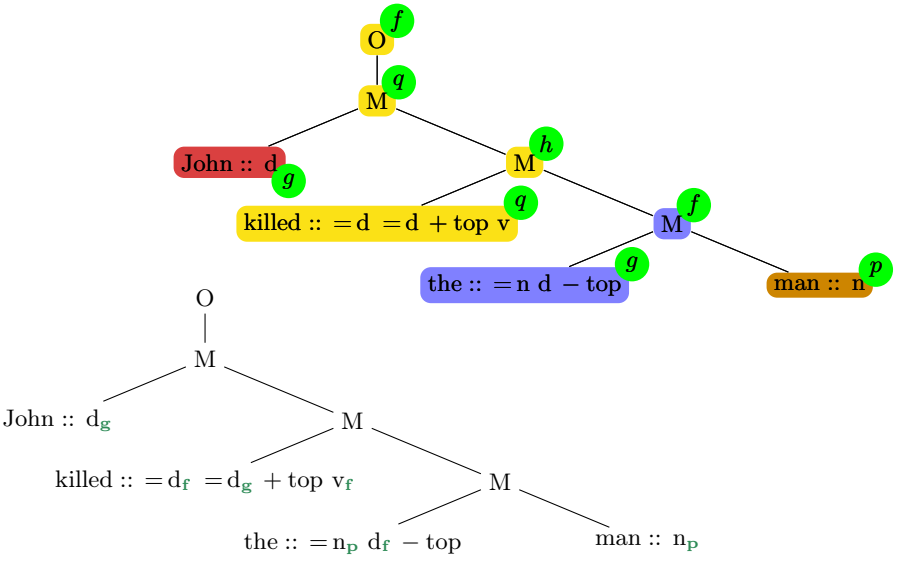


Fig. 2. Derivation tree of MG G with states of A_R (top), and corresponding derivation tree of G' with refined categories (bottom)

But now we have to move the state subscripts from the internal nodes into the lexical items. We do so again in two steps (before proceeding any further, though, the reader may want to take a look at the simplified example in Fig. 2 to get a better intuition for the procedure). For the first step, we look at each lexical item and add the subscripted state of its slice root to its category feature — keep in mind that a lexical item has exactly one such feature. In the second step, we have to refine the selection features accordingly. Given the definition of slices, it is easy to see that the $(n+1)^{\text{th}}$ node o of slice(l) (counting from l toward the slice root) denotes the operation that erased the n^{th} feature f of l, $n \geq 1$. If $\ell(o) = M$ (in which case f is a selector feature), determine the category c_q of the lexical item l' whose slice root is immediately dominated by o and replace f by $=c_q$. Repeat this procedure for all selector features of all lexical items in all trees.[2] Call the set of these lexical items with refined category and selector features $Lex_{G'}$ (which is still finite due to the restriction of A_R to finitely many states). The desired MG is $G' := \langle \Sigma, Feat_{G'}, F_{G'}, Types, Lex_{G'}, Op \rangle$, where

[2] Two things are worth mentioning here. First, there seems to be no way around restricting the selector features of lexical items, as is witnessed by example 2, where a lexical item of category a_o may not select two lexical items of the same category, and one of category a_e may not select two with differing categories. Second, it suffices for the construction to consider but a finite number of configurations, so a computational implementation of the procedure is still feasible. This follows from the finiteness of the lexicon and the index of the Nerode partition induced by the automaton.

- $Feat_{G'} := \{f_q, =f_q \mid f \in base_G,\ q \text{ a state of } A_R\} \cup \{-f, +f \mid f \in base_G\}$, and
- $F_{G'} := \{c_q \mid c \in F_G,\ q \text{ a final state of } A_R\}$

Lemma 1 and 2 jointly guarantee that the procedure above is well-defined. In order to prove its correctness, though, we first need one more property of slices.

Lemma 3. *Let $T := \langle D, \ell\rangle$ be a derivation tree, $\mathbb{S} := \{\text{slice}(l) \mid l = \ell(n)$ for some leaf $n \in D\}$, and $\mathbb{S} \upharpoonright \text{slice}(l) := \{\text{slice}(l') \in \mathbb{S} \mid l'$ was selected by $l\}$. Let $\overrightarrow{s} := \langle s_1, \ldots, s_n\rangle$ be a sequence of $s \in \mathbb{S}$ such that*

- $s_1 \in \mathbb{S}$
- *for all* $1 \leq i < n$, $s_{i+1} \in \mathbb{S} \upharpoonright s_i$
- $\mathbb{S} \upharpoonright s_n = \emptyset$

For every Minimalist derivation tree, there is at least one such $\overrightarrow{s}$, and for every choice of $\overrightarrow{s}$, $s_n := \langle S, \ell\rangle$ is a slice with $|S| = 1$.

Proof. The first half of the claim is trivial. As for the second one, if $\mathbb{S} \upharpoonright s_n$ is empty, the lexical item l such that $s_n := \text{slice}(l)$ has no selector features. But then it has no licensor features either, because these must precede the category feature, which is the only way l has left to enter the derivation. □

With this unsurprising yet important fact established, we finally turn to the correctness of the procedure, splitting the claim into two lemmas for the sake of readability. Note that I use π to denote the projection that strips away the state suffixes and thus turns $Lex_{G'}$ into Lex_G again.

Lemma 4. $\pi(\text{mder}(G')) \subseteq R \cap \text{mder}(G)$

Proof. Assume towards a contradiction $\pi(\text{mder}(G')) \not\subseteq R \cap \text{mder}(G)$. Then there has to be some tree $T \in \pi(\text{mder}(G'))$ such that $T \notin R \cap \text{mder}(G)$. But $\text{mder}(G)$ cannot be a proper subset of $\pi(\text{mder}(G'))$, so it has to be the case that $\text{mder}(G) \ni T \notin R$. Thus, when recognizing T, A_R assigns the root of T some non-final state q'. Suppose w.l.o.g. that the root node of T belongs to $\text{slice}(l)$ for some lexical item l of final category c_q. According to our procedure, c_q is a final category iff c is a final category of G and q is a final state of A_R. So if $T \in \pi(\text{mder}(G'))$, there has to be some $T' \in \text{mder}(G) \cap R$ such that in both trees the root node is also the root of $\text{slice}(l)$ (otherwise A_R never reached a final state in the slice root of l, whence l isn't of category c_q, a contradiction). Now since A_R is deterministic, the only way for it not to reach state q at the slice root of l in T is if the state it assigns to the slice root of some lexical item l' selected by l differs from the subscript of the corresponding selector feature of l. But the same reasoning applies to l' as well, so that we progress further down the tree until we encounter a lexical item that selects a lexical item l'' whose slice is of size 1 (by Lem. 3). So the automaton must have assigned the slice root of l'' a state different from the subscript of the category feature of l''. But A_R is bottom-up and deterministic, wherefore it always assigns the same state to l''. Contradiction. □

Lemma 5. $\pi(\text{mder}(G')) \supseteq R \cap \text{mder}(G)$

Proof. Assume once again towards a contradiction $\pi(\text{mder}(G')) \not\supseteq R \cap \text{mder}(G)$. Then there has to be some tree $T \in R \cap \text{mder}(G)$ such that $T \notin \pi(\text{mder}(G'))$. Since the lexicons of G and G' are identical modulo state-subscripts, the only option is that A_R and G' disagree with respect to states, at which point the reasoning of the previous proof applies unaltered. □

Theorem 2. *The class of Minimalist derivation tree languages over Σ, Feat is p-closed under intersection with regular tree languages.*

Corollary 2. *The class of Minimalist derivation tree languages over Σ, Feat is p-closed under intersection and relative complement.*

Proof. Since every Minimalist derivation tree language is regular, p-closure under intersection follows immediately from Thm. 2. Given two Minimalist derivation tree languages L and M, $R := L - M$ is a regular language, so $L - M = L \cap R$ is a projection of some Minimalist derivation tree language. □

Note that p-closure under relative complement does not imply p-closure under complement with respect to the class of Minimalist derivation tree languages over Σ, *Feat*, as for each grammar over this signature there exists another grammar whose derivation tree language is a proper superset of the former's. However, when we restrict our attention to the class of all MG whose lexicon is a subset of some finite set *Lex* over Σ, *Feat*, there will be one Minimalist derivation tree language that subsumes all others and p-closure under relative complement implies p-closure under complement as desired (which in turns implies p-closure under union).

Corollary 3. *Let Lex be some finite subset of $\Sigma^* \times \{::\} \times Feat^*$. Then the class $\{\text{mder}(G) \mid G$ an MG with $Lex_G \subseteq Lex\}$ is p-closed under complement and union.*

P-closure also extends to linear tree transductions whose co-domain is a Minimalist derivation tree language. This is of immediate relevance to Graf's tree transducer model of reference-set computation, because most reference-set constraints are conceived of as filters, that is to say, they map each Minimalist derivation tree language into a subset of itself.

Corollary 4. *Given a linear transduction τ with some Minimalist derivation tree language L as its co-domain, it holds for every regular tree language R that its image under τ is a projection of some Minimalist derivation tree language.*[3]

Proof. Follows from Thm. 2 and the fact that the range of a linear transduction applied to a regular tree language is regular. □

[3] My thanks go to an anonymous reviewer for pointing out that the corollary as it was originally stated was overly restrictive.

In connection with Graf's transducer approach, we also observe that the procedure as it is currently defined may lead to a significant (albeit still linear) blow-up in the size of the lexicon. The implication for Graf's work is that a grammar with reference-set constraints may be notably more succinct than one without them, even if both define the same tree languages *modulo* state subscripts. But since it might be the case that the procedure given here can still be improved upon, this has to remain a conjecture for now.

Conjecture 1. Given a lexicon *Lex* and $n \geq 0$, let $Lex^{(n)} := \{l \in Lex \mid l \text{ has exactly } n \text{ selector features}\}$. Now if there is no $m > k$ such that $Lex_G^{(m)} \neq \emptyset$, then in the worst case

$$|Lex_{G'}| = \sum_{i=0}^{k} \left(|Lex_G^{(i)}| \cdot |Q|^{i+1} \right)$$

4 Minimalist Grammars with Regular Control

P-closure under intersection also opens up new ways of incorporating constraints into MGs. Constraints have proven difficult to study in MGs, and their effects on the machinery are somewhat unpredictable; for instance, MGs with the SMC and the so-called Specifier Island Constraint (SPIC) are weaker than MGs that feature only the SMC, whereas MGs that lack the SMC yet have the SPIC generate type-0 languages [3]. But adopting the perspective of model-theoretic syntax [13], we may view constraints as defining formal languages. Thanks to Thm. 2, then, MGs can be augmented by any finite number of constraints defining regular tree languages without increasing their weak generative capacity. In fact, even the strong generative capacity of MGs is mostly unaffected, as the procedure outlined above only relies on refining category features, which — in the derived tree — surface only on the head of the highest phrase.

Definition 7 (MGs with Regular Control). *A* Minimalist Grammar with Regular Control *(MG^{RC}) is a 7-tuple $G := \langle \Sigma, Feat, F, Types, Lex, Op, \mathcal{R} \rangle$, where*

- *Σ, Feat, F, Types, Lex, and Op are defined as usual,*
- *and $\mathcal{R}$ is a finite set of regular tree languages.*

The language generated by G is the set $L(G) := \{\sigma \mid \langle \sigma \cdot c \rangle \in closure(Lex, Op), \cdot \in Types, c \in F,$ and $\langle \sigma \cdot c \rangle$ is the label of the root of some tree $T \in \mathrm{sder}(G)$ such that $\mu(T) \in \mathrm{mder}(G) \cap \bigcap_{R \in \mathcal{R}} R\}$.

Theorem 3. $\mathrm{MG} \equiv \mathrm{MG}^{\mathrm{RC}}$

Given the prominence of constraints in the syntactic literature, it is hardly surprising that there are numerous applications for regular control. The most obvious one are intervention conditions on movement such as the one illustrated in (1) below.

(1) a. Who$_i$ did John say that Bill adores t_i?
b. ?/* Who$_i$ did John doubt whether Bill adores t_i?

Without further precautions, a MG that derives (1a) will also derive (1b) as movement is restricted only by the feature calculus, not the shape of the phonological strings. A regular language can easily militate against such locality violations. Recall that the states of an automaton recognizing a Minimalist derivation tree can be taken to represent the feature components of the string-annotated derivation trees. In order to block (1b), then, one may proceed as follows. Let p and q be the states that the standard automaton would assign to *whether* and *Bill hates* t_i, respectively (we take these states to literally be sequences of feature sequences). Now introduce a new state p^{wh} that the automaton assigns to *whether* instead of p such that $p^{wh} \times q \times M$ is undefined only if q contains no sequence containing a movement licensee features, in which case $p^{wh} \times q \times M = p \times q \times M$.

It is easy to see that this strategy can be extended to intervention conditions in general, most of which require the automaton to check the shape of entire phrases rather than a single word. Two well-known examples are the Complex NP Constraint, which blocks extraction from a CP that is the complement of an NP (or rather, DP in contemporary analyses), and the Subject Constraint, which rules out movement originating from inside a DP in subject position.

(2) * Who$_i$ did John reject the claim that the lobbyists bribed t_i?

(3) a. Where$_i$ is it likely that John went t_i?
b. * Where$_i$ is that John went t_i likely?

A combination of both types of movement constraint is the *that*-trace effect: in general, a wh-word can be extracted out of a CP whose complementizer is *that*, but not if it is the subject of the clause.

(4) a. What$_i$ did you say that John ate t_i?
b. * Who$_i$ did you say that t_i ate my burrito?

Here the automaton has to be sensitive to both the nature of the complementizer and the structural properties of the domain from which the wh-word was extracted. The sensitivity to *that* is analogous to the *whether*-example, while the distinction between subjects and objects mirrors the domain condition of the Subject Constraint.

Further examples of linguistic locality constraints that can be captured this way are the Coordinate Structure Constraint, the Left Branch Condition, and phases (introduced in [2] and implemented for MGs in [15]). Many of the principles formalized in [13] can also be adapted for MGs, although the change from derived trees to derivation trees will require some slight revisions in certain cases, in particular binding and control, which rely on a notion of c-command that might prove tricky to capture on a derivational level.

Through the use of constraints we can also reduce the number of movement steps in our grammars. In early Minimalism [1], satisfying feature dependencies between non-adjacent phrases invariably required movement, an assumption inherited by MGs. In such a setup, subject-verb agreement, say, is assumed to be an effect of the subject moving into the specifier of the TP and checking its person features. But other instances of agreement, e.g. between determiners or

adjectives on the one hand and nouns on the other, are rather cumbersome to handle this way. This brought about a major revision of the feature calculus in order to make feature checking apply a distance in certain cases [2]. As long as we do not allow unbounded nesting and crossing of the checking paths defined by this operation, regular constraints can yield the same effect by associating every lexical item with "pseudo-features" that encode properties not pertinent to movement. For instance, the Icelandic adjective *rauðan* 'red', which is masculine, singular, accusative, and strongly inflected, could be assigned the corresponding pseudo-features, which in turn also have to be present on the noun the adjective combines with. A more interesting case is long-distance subject-verb agreement in English expletive constructions, as the phenomenon is noticeably more difficult to capture by manual refinement of categories.

(5) a. * There seems to John to be several men in the garden.
b. There seem to John to be several men in the garden.

But the gerrymandering of the feature calculus need not stop here. We may also employ regular constraints to incorporate a restricted version of pied-piping. Pied-piping refers to the phenomenon that a constituent containing some element with a movement licensee feature seems to be stuck to it for the purposes of movement.

(6) a. [Which famous linguist]$_i$ did Robert write a book [about t_i]?
b. [About which famous linguist]$_i$ did Robert write a book t_i?

In the syntactic literature this is often analyzed as the movement licensee feature of the DP percolating upwards into the PP. Unfortunately, enriching MGs with such a feature percolation mechanism allows them to generate any recursively enumerable language [8]. But at least for the example above, only a very limited kind of feature percolation is required: it is sufficient to allow both *about* and *which* to carry a movement licensee feature as long as we ensure that the variant of *about* with such a feature must merge with a DP such that the determiner of said DP does not carry the same feature, but could in principle (i.e. there is an entry in the lexicon with the same phonological string and the same category as the determiner that also carries the relevant feature). It is easy to see that this constraint can be enforced by regular means.

Dynamic restrictions on the distribution of features also allows us to work around certain shortcomings of the SMC. The SMC — albeit essential for keeping Minimalist derivation trees within the confines of regular tree languages — comes with its fair share of linguistic inadequacies, in particular with respect to wh-movement. Since English allows for wh-phrases to stay *in situ*, every wh-phrase in an MG must come in two variants, one with a wh-licensee feature, and one without it. But given this duality, nothing prevents superiority violations like the one in (7b) (for the sake of simplicity, only wh-movement is indicated by traces).

(7) a. Who$_i$ t_i prefers what?
b. * What$_i$ does who prefer?

The ungrammatical (7b) can be derived because *who* need not carry a wh-licensee feature, in which case the MG will treat it like any other DP. Consequently, nothing prevents *what* from carrying a wh-licensee feature, so the movement step is licit. Instances of overgeneration like this can be blocked if one takes a hint from our implementation of pied-piping: the automaton has to verify that no node along the movement path could potentially carry the same feature. In (7b) above, this condition is violated because *who* could be a carrier of the wh-licensee feature, and so the automaton rejects the derivation tree. Note that a similar strategy can be used if two identical features have to be distinguished due to the SMC yet at the same we want to capture specific locality restrictions they impose on each other (e.g. *who* being assigned feature wh_1 and *what* feature wh_2 in a multiple wh-movement language). Admittedly the notion "node along the movement path" has to be carefully worked out and may turn out to be rather complex in grammars with massive remnant movement. Overall, though, it seems that this approach goes a long way towards an MG implementation of Relativized Minimality as envisioned in [15], with the added benefit that the restrictions imposed at the level of derivation trees also carry over to the strictly more powerful mechanism of MGs with copying [9].

Conclusion

I defined Minimalist derivation tree languages and showed that they are p-closed under intersection with regular tree languages, intersection, complement, relative complement, union, and linear tree transductions whose co-domain is a Minimalist derivation tree language. From these closure properties it follows immediately that enriching MGs with regular control does not increase their weak generative capacity. The result has numerous linguistic applications, in particular regarding locality conditions on movement and reference-set constraints [cf. 4, 5].

Acknowledgments. For their motivational comments and helpful criticism, I am greatly indebted to Ed Stabler, Ed Keenan, Jens Michaelis, Uwe Mönnich, the three anonymous reviewers, and the attendees of the UCLA Mathematical Linguistics Circle. The research reported herein was supported by a DOC-fellowship of the Austrian Academy of Sciences.

References

[1] Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
[2] Chomsky, N.: Derivation by phase. In: Kenstowicz, M.J. (ed.) Ken Hale: A Life in Language, pp. 1–52. MIT Press, Cambridge (2001)
[3] Gärtner, H.M., Michaelis, J.: Some remarks on locality conditions and minimalist grammars. In: Sauerland, U., Gärtner, H.M. (eds.) Interfaces + Recursion = Language? Chomsky's Minimalism and the View from Syntax-Semantics, pp. 161–196. Mouton de Gruyter, Berlin (2007)

[4] Graf, T.: Reference-set constraints as linear tree transductions via controlled optimality systems. In: Proceedings of the 15th Conference on Formal Grammar (2010) (to appear)
[5] Graf, T.: A tree transducer model of reference-set computation. UCLA Working Papers in Linguistics 15, article 4 (2010)
[6] 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)
[7] Joshi, A.: 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, Cambridge (1985)
[8] Kobele, G.M.: Features moving madly: A formal perspective on feature percolation in the minimalist program. Research on Language and Computation 3(4), 391–410 (2005)
[9] Kobele, G.M.: Generating Copies: An Investigation into Structural Identity in Language and Grammar. Ph.D. thesis, UCLA (2006)
[10] 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)
[11] Michaelis, J.: Derivational minimalism is mildly context-sensitive. In: Moortgat, M. (ed.) LACL 1998. LNCS (LNAI), vol. 2014, pp. 179–198. Springer, Heidelberg (2001)
[12] 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)
[13] Rogers, J.: A Descriptive Approach to Language-Theoretic Complexity. CSLI, Stanford (1998)
[14] Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
[15] Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–643. Oxford University Press, Oxford (2011)
[16] Stabler, E.P., Keenan, E.: Structural similarity. Theoretical Computer Science 293, 345–363 (2003)
[17] Thatcher, J.W.: Characterizing derivation trees for context-free grammars through a generalization of finite automata theory. Journal of Computer and System Sciences 1, 317–322 (1967)

Well-Nestedness Properly Subsumes Strict Derivational Minimalism*

Makoto Kanazawa[1], Jens Michaelis[2], Sylvain Salvati[3], and Ryo Yoshinaka[4]

[1] National Institute of Informatics, Tokyo, Japan
[2] Bielefeld University, Bielefeld, Germany
[3] INRIA Bordeaux – Sud-Ouest, Talence, France
[4] Japan Science and Technology Agency, ERATO MINATO Project, Sapporo, Japan

Abstract. *Minimalist grammars (MGs)* constitute a mildly context-sensitive formalism when being equipped with a particular *locality condition (LC)*, the *shortest move condition*. In this format MGs define the same class of derivable string languages as *multiple context-free grammars (MCFGs)*. Adding another LC to MGs, the *specifier island condition (SPIC)*, results in a proper subclass of derivable languages. It is rather straightforward to see this class is embedded within the class of languages derivable by some *well-nested MCFG (MCFG$_{wn}$)*. In this paper we show that the embedding is even proper. We partially do so adapting the methods used in [13] to characterize the separation of MCFG$_{\text{wn}}$-languages from MCFG-languages by means of a "simple copying" theorem. The separation of strict derivational minimalism from well-nested MCFGs is then characterized by means of a "simple reverse copying" theorem. Since for MGs, *well-nestedness* seems to be a rather *ad hoc* restriction, whereas for MCFGs, this holds regarding the SPIC, our result may suggest we are concerned here with a structural difference between MGs and MCFGs which cannot immediately be overcome in a non-stipulated manner.

1 Introduction

Inspired by the work originating in [1], the formal type of a *minimalist grammar (MG)* has been introduced in [28] as an attempt at a rigorous algebraic formalization of the corresponding perspectives adopted within the linguistic framework of transformational grammar. MGs have been shown to be capable of integrating, if needed, a variety of arguably "odd" items from the syntactician's toolbox such as *head movement* [28,30], *(strict) remnant movement* [28,29], *affix hopping* [30], *copy-movement* [14] and *relativized minimality* [31], to mention some.

Interestingly, the formal MG-setting can also be seen as having anticipated some of the crucial developments and changes within the theoretical setting of the minimalist branch of generative grammar since the mid of the 1990s (see e.g.

* This work has essentially been carried out within the joint research project "Open Problems on Multiple Context-Free Grammars" funded by the National Institute of Informatics, Tokyo, Japan.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 112–128, 2011.

[2,3]). Maybe the most prominent deviance from at least the original linguistic setting was that MGs never incorporated any so-called *transderivational constraints*. However, *locality conditions (LCs)* applying to the *move*-operator have always been of decisive nature within the formal MG-framework. A particular LC, the *shortest move condition (SMC)*, played a crucial role in showing that each MG satisfying the definition in [28], and thus, obeying the SMC can be constructively transformed into a *multiple context-free grammar (MCFG)* in the sense of [27] deriving the same string language. The construction presented in [18] has not only proven the corresponding MG-class to be mildly context-sensitive in the sense of [11], but also led to a succinct, "chain-based" reformulation of MGs reducing them to their "bare essentials," cf. [32]. By means of this reformulation, MGs can be straightforwardly interpreted as a proper subtype of MCFGs. In particular, all corresponding MCFGs are of rank 2, i.e., the righthand side of each rule consists of at most two nonterminals. Nevertheless, in terms of derivable string languages the generative power of MCFGs is not reduced as shown independently in [10] and [20].

In particular building on the work in [16], in [29] a revised MG-type has been introduced. Throughout that paper this type is not distinguished by name from the type introduced earlier in [28], although beside the SMC, the revised version implicitly implements a second LC, which has been explicitly referred to as *specifier island condition (SPIC)* in [5] and later work. Closely in keeping with further theoretical linguistic considerations, in [29] also a particular type of a *strict minimalist grammar (SMG)* has been introduced, implementing the SPIC with somewhat more "strictness," and leading to *heavy pied-piping* constructions. In [19,21] it has been shown that the SMG-class and the MG-class of revised type define identical classes of derivable languages. From this point of view we consider the MG-class of revised type an instance of *strict derivational minimalism*.

With emphasis on particular linguistic aspects, the combinatory power of the SMC and the SPIC within the MG-framework has been discussed in [6], formal results have been proven in different other places: we already mentioned [18] and [10,20] as the sources showing that the class of MCFGs and the class of MGs obeying the SMC, but not necessarily the SPIC give rise to the same class of derivable string languages. [15] proves MGs obeying the SPIC, but not necessarily the SMC to be Turing complete. [26] shows that the decision problem for MGs neither obeying the SPIC nor the SMC is as hard the the one for proof search in *multiplicative exponential linear logic (MELL)* as introduced in [8].[1]

In [19,21] it is shown that MGs obeying both the SMC and the SPIC derive the same class of string language as a particular subtype of MCFGs of rank 2, referred to in [19,21] as the type of an $MCFG_{1,2}$. Here, we refer to this subtype as the type of a *monadic branching MCFG* ($MCFG_{mb}$). It plays the central role in our paper: an $\mathrm{MCFG_{mb}}$ is an MCFG of rank 2 such that for each rule with two nonterminals appearing on the righthand side it holds that from the first nonterminal only simple strings of terminals can be derived, i.e., only

[1] The latter result provides a negative answer to the question, whether all languages generated by such an MG are *semilinear*?

1-tuples of terminal strings can be derived from the first nonterminal instead of k-tuples for an arbitrary, but fixed $k \geq 1$ as in the general MCFG-case. In fact, it can be shown that the $\mathrm{MCFG}_{\mathrm{mb}}$-class constitutes a proper subclass of the full MCFG-class also in terms of derivable string languages, cf. [22]. Figure 1 is summing up the complexity results mentioned so far concerning MGs and the interaction of SMC and SPIC, where $\mathcal{MCFL}$ and $\mathcal{MCFL}_{\mathrm{mb}}$ denote the classes of derivable string languages determined the MCFG-class and the $\mathrm{MCFG}_{\mathrm{mb}}$-class, respectively.

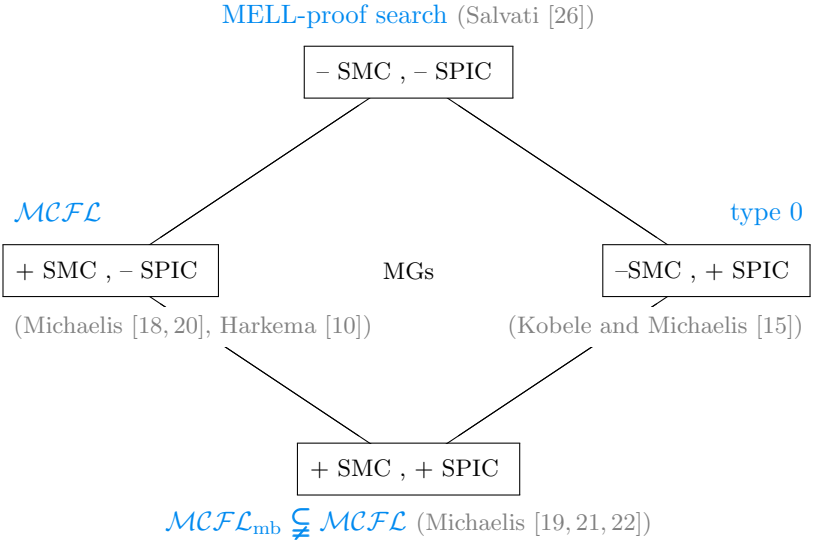


Fig. 1. The interaction of the SMC and the SPIC with the MG-framework

The proof presented in [22] to separate $\mathcal{MCFL}_{\mathrm{mb}}$ from $\mathcal{MCFL}$ builds on the inclusion of $\mathcal{MCFL}_{\mathrm{mb}}$ within $\mathcal{MCFL}_{\mathrm{wn}}$, the latter being the class of string languages derivable by some *well-nested MCFG (MCFG$_{wn}$)*. $\mathcal{MCFL}_{\mathrm{wn}}$ in its turn constitutes a proper subclass of $\mathcal{MCFL}$. As pointed out, e.g., in [23], the latter was (at least implicitly) known for quite a while. [13], however, crucially presents a separation theorem relying on arguments on "simple copying" which were not available in that form before. Whether the inclusion of $\mathcal{MCFL}_{\mathrm{mb}}$ within $\mathcal{MCFL}_{\mathrm{wn}}$ is proper, has, to the best of our knowledge, been generally open so far. We show here that the answer is positive. We partially do so adapting the methods used in [13]. The separation of $\mathcal{MCFL}_{\mathrm{mb}}$ from $\mathcal{MCFL}_{\mathrm{wn}}$, and thus, of strict derivational minimalism from well-nested MCFGs, is then characterized by means of a "simple reverse copying" theorem.

2 Multiple Context-Free Grammars

A *ranked alphabet* is a finite set Δ of the form $\Delta = \bigcup_{k \geq 0} \Delta^{(k)}$, where $\langle \Delta^{(k)} \mid k \geq 0 \rangle$ is an indexed family of pairwise disjoint sets. The set of *trees (over Δ)*, $\mathcal{T}(\Delta)$,

is built up recursively in the following way: If for some $k \geq 0$, we have $d \in \Delta^{(k)}$ and $T_1, \ldots, T_k \in \mathcal{T}(\Delta)$ then $(dT_1 \cdots T_k) \in \mathcal{T}(\Delta)$. In writing trees, we adopt the abbreviatory convention of dropping the outermost parentheses. Note that in case $k = 0$ the string $T_1 \cdots T_k$ is the empty string, ε, Therefore we generally omit the parentheses in this case.

Let N and Σ be a ranked and an unranked alphabet, respectively, with $N^{(0)} = \emptyset$, and assume $X = \{x_i \mid i \geq 0\}$ to be a countably infinite set of variables ranging over Σ^*. A *rule over* $\langle N, \Sigma\rangle$ (or, simply a *rule*, if $\langle N, \Sigma\rangle$ is understood from context) is an expression of the form

$$B_0(\alpha_1, \ldots, \alpha_{k_0}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, B_n(x_{n,1}, \ldots, x_{n,k_n}) \tag{1}$$

for some $n \geq 0$ and $k_i \geq 1$ for $i \in [0, n]$ such that for $i \in [0, n]$, $B_i \in N^{(k_i)}$, and such that for $i \in [1, k_0]$, α_i is a string over $\Sigma \cup \{x_{i,j} \mid i \in [1, n], j \in [1, k_i]\}$, where $\{x_{i,j} \mid i \in [1, n], j \in [1, k_i]\}$ is a set of pairwise distinct variables from X. In addition, each $x_{i,j}$ occurs at most once in $\alpha_1 \cdots \alpha_{k_0}$, the concatenation of all α_i "from left to right." In case $n = 0$ such a rule is *terminating*, otherwise it is *non-terminating*.

Definition 1. A *multiple context-free grammar (MCFG)*, G, is a quadruple $\langle N, \Sigma, P, S\rangle$, where N is a ranked alphabet of *nonterminals* with $N^{(0)} = \emptyset$, where Σ is an unranked alphabet of *terminals*, where P is a finite set of rules over $\langle N, \Sigma\rangle$, and where $S \in N^{(1)}$.

Let $G = \langle N, \Sigma, P, S\rangle$ be an MCFG.

For $k_0 \geq 1$, and corresponding $B_0 \in N^{(k_0)}$ and $w_1, \ldots, w_{k_0} \in \Sigma^*$, we write $\vdash_G B_0(w_1, \ldots, w_{k_0})$ to mean that $B_0(w_1, \ldots, w_{k_0})$ is *derivable (in G)* according to the following inference scheme:

$$\frac{\vdash_G B_1(w_{1,1}, \ldots, w_{1,k_1}) \ \ldots \ \vdash_G B_n(w_{n,1}, \ldots, w_{n,k_n})}{\vdash_G B_0(\alpha_1, \ldots, \alpha_{k_0})\sigma} \tag{2}$$

where $B_0(\alpha_1, \ldots, \alpha_{k_0}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, B_n(x_{n,1}, \ldots, x_{n,k_n})$ is a rule in P according to (1), where $w_{i,j} \in \Sigma^*$, and where σ is the substitution which maps each variable $x_{i,j}$ to $w_{i,j}$.

The *language derivable by* G, $L(G)$, is the set $\{w \in \Sigma^* \mid \vdash_G S(w)\}$.

Definition 2. A *multiple context-free language (MCFL)* is a set (of strings), L, such that there is an MCFG, G, with $L(G) = L$.

Let $G = \langle N, \Sigma, P, S\rangle$ be an MCFG.

If $A \in N^{(k)}$ for some $k \geq 1$ then k is the *arity of* A, denoted by $arity(A)$. The *dimension of* G is defined as the maximum of $\{arity(A) \mid A \in N\}$.

If $B_0(\alpha_1, \ldots, \alpha_{k_0}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, B_n(x_{n,1}, \ldots, x_{n,k_n})$ is some rule $p \in P$ according to (1) then the number n is the *rank of* p, denoted $rank(p)$. Thus, p is terminating iff $rank(p) = 0$. The *rank of* G, denoted $rank(G)$, is defined as the maximum of $\{rank(p) \mid p \in P\}$.

For $m, r \geq 1$, an *m-MCFG(r)* is an MCFG, G, of dimension at most m and rank at most r. An *m-MCFL(r)* is an MCFL, L, such that there is an m-MCFG(r), G, with $L(G) = L$ for some $m \geq 1$ and $r \geq 1$.

We denote by $m\text{-}\mathcal{MCFG}(r)$ the class of all MCFGs of dimension at most m and rank at most r, and by $m\text{-}\mathcal{MCFL}(r)$ the class of all MCFLs generated by some m-MCFG(r).

We let $m\text{-}\mathcal{MCFG}$, $\mathcal{MCFG}(r)$, $\mathcal{MCFG}$, $m\text{-}\mathcal{MCFL}$, $\mathcal{MCFL}(r)$ and $\mathcal{MCFL}$ denote the classes $\bigcup_{r\geq 1} m\text{-}\mathcal{MCFG}(r)$, $\bigcup_{m\geq 1} m\text{-}\mathcal{MCFG}(r)$, $\bigcup_{m,r\geq 1} m\text{-}\mathcal{MCFG}(r)$, $\bigcup_{r\geq 1} m\text{-}\mathcal{MCFL}(r)$, $\bigcup_{m\geq 1} m\text{-}\mathcal{MCFL}(r)$ and $\bigcup_{m,r\geq 1} m\text{-}\mathcal{MCFL}(r)$, respectively.

Theorem 1. *Let $L = \{w\#w^R \mid w \in L_0\}$ for some set of strings L_0.*

(i) If for some $m, r \geq 1$, $L_0 \in m\text{-}\mathcal{MCFL}(r)$ then $L \in 2m\text{-}\mathcal{MCFL}(r)$.
(ii) If for some $m, r \geq 1$, $L \in m\text{-}\mathcal{MCFL}(r)$ then $L_0 \in m\text{-}\mathcal{MCFL}(r)$.

Proof (sketch). Let $m, r \geq 1$. (i): constructing an $2m$-MCFG(r) G with $L(G) = L$, from a given m-MCFG(r) G_0 with $L(G_0) = L_0$, is straightforward. (ii): the class of $m\text{-}\mathcal{MCFL}(r)$ is closed under rational transductions. □

Let $G = \langle N, \Sigma, P, S\rangle$ be an MCFG.

In order to be able to talk about derivation trees of derivable facts, we will identify P with a ranked alphabet Δ_P relying on a bijection $f : P \to \Delta_P$ such that for each $p \in P$, $f(p) \in \Delta^{(n)}$ iff $rank(p) = n$. *Derivation trees* are trees over the ranked alphabet Δ_P. *Derivation trees contexts* are trees over the ranked alphabet $\Delta_P(Y)$, where $\Delta_P(Y)^{(n)} = \Delta_P^{(n)}$ for $n \geq 1$, and $\Delta_P(Y)^{(0)} = \Delta_P^{(0)} \cup Y$ with $Y = \{y_i \mid i \geq 0\}$ being a countably infinite set of variables disjoint from Δ_P. The following inference system associates derivation trees with derivable facts and derivation tree contexts with facts derivable from some premises:

$$\frac{}{y : B(x'_1, \ldots, x'_k) \vdash y : B(x'_1, \ldots, x'_k)} \tag{3}$$

$$\frac{\Gamma_1 \vdash_G T_1 : B_1(\beta_{1,1}, \ldots, \beta_{1,k_1}) \ \ldots \ \Gamma_n \vdash_G T_n : B_n(\beta_{n,1}, \ldots, \beta_{n,k_n})}{\Gamma_1, \ldots, \Gamma_n \vdash_G pT_1 \cdots T_n : B_0(\alpha_1, \ldots, \alpha_{k_0})\sigma} \tag{4}$$

In the first scheme, (3), it holds that $y \in Y$, $B \in N^{(k)}$ for some $k \geq 1$ and $x'_i \in X$. In the second scheme, (4), it holds that p is a rule from P of the form

$$B_0(\alpha_1, \ldots, \alpha_{k_0}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, B_n(x_{n,1}, \ldots, x_{n,k_n})$$

according to (1), $\beta_{i,j} \in (\Sigma \cup X)^*$, and σ is a substitution mapping each $x_{i,j}$ to $\beta_{i,j}$. Each Γ_i is a finite sequence of premises of the form $z : C(x'_1, \ldots, x'_k)$ with $z \in Y$, and with $C \in N^{(k)}$ for some $k \geq 1$ and $x'_i \in X$. It is also understood that Γ_i and Γ_j do not share any variables if $i \neq j$. Each T_i is a derivation tree context over $\Delta_P(Y)$. For $k \geq 1$, $A \in N^{(k)}$ and $w_i \in \Sigma^*$ for $i \in [1, k]$, it clearly holds that $\vdash_G A(w_1, \ldots, w_k)$ iff $\vdash_G T : A(w_1, \ldots, w_k)$ for some derivation tree T over Δ_P.

For each $k \geq 1$, $A \in N^{(k)}$ is *useful* if there are $w_i \in \Sigma^*$ for $i \in [1,k]$ such that $\vdash_G A(w_1,\ldots,w_k)$, and if there are $y \in Y$, $x'_i \in X$ for $i \in [1,k]$, $\alpha \in (\Sigma \cup X)^*$ and some derivation tree T over $\Delta_P(Y)$ such that $y : A(x'_1,\ldots,x'_k) \vdash_G T : S(\alpha)$. $A \in N^{(k)}$ is *useless*, if it is not useful.

Let $B_0(\alpha_1,\ldots,\alpha_{k_0}) \leftarrow B_1(x_{1,1},\ldots,x_{1,k_1}),\ldots,B_n(x_{n,1},\ldots,x_{n,k_n})$ be some rule $p \in P$ according to (1).

- p is *non-deleting* if for $i \in [1,n]$ and $j \in [1,k_i]$, $x_{i,j}$ occurs in $\alpha_1 \cdots \alpha_{k_0}$. (5)
- p is *non-permuting* if for $i \in [1,n]$ and $j,k \in [1,k_i]$, $j < k$ implies that the occurence (if any) of $x_{i,j}$ in $\alpha_1 \cdots \alpha_{k_0}$ precedes the occurence (if any) of $x_{i,k}$ in $\alpha_1 \cdots \alpha_{k_0}$. (6)
- p is *well-nested* if it is non-deleting and non-permuting, and for every $i,i' \in [1,n]$ with $i \neq i'$, $j \in [1,k_i-1]$ and $j' \in [1,k_{i'}-1]$, it additionally satisfies:
$$\alpha_1 \cdots \alpha_{k_0} \notin (\Sigma \cup X)^* x_{i,j} (\Sigma \cup X)^* x_{i',j'} (\Sigma \cup X)^* x_{i,j+1} (\Sigma \cup X)^* x_{i',j'+1} (\Sigma \cup X)^*. \quad (7)$$
- p is *monadic branching* if $n \leq 2$, and $n = 2$ implies $k_1 = 1$. (8)

Note that, if each $p \in P$ is non-deleting then G is a *linear context-free rewriting system (LCFRS)* in the sense of [33]; if each $p \in P$ is non-permuting then G is an MCFG in *monotone function form* in the sense of [19]; and if each $p \in P$ is non-deleting and non-permuting then G is an *ordered simple RCG* in the sense of [34] as well as a *monotone LCFRS* in the sense of [17].

Definition 3. An MCFG $G = \langle N, \Sigma, P, S\rangle$ is *well-nested*, if each rule $p \in P$ is well-nested in the sense of (7).

Definition 4. An MCFG $G = \langle N, \Sigma, P, S\rangle$ is *monadic branching*, if each rule $p \in P$ is monadic branching in the sense of (8).

We attach the subscripts "wn" and/or "mb" to "MCFG" and "MCFL" in order to refer to a well-nested and/or monadic branching MCFG and MCFL of corresponding type. More concretely, we write "MCFG_x," "m-MCFG_x," "$\mathrm{MCFG}_x(r)$" and "m-$\mathrm{MCFG}_x(r)$" as well as "MCFL_x," "m-MCFL_x," "$\mathrm{MCFL}_x(r)$" and "m-$\mathrm{MCFL}_x(r)$" with x being of the form "wn", "mb" or "wn,mb."

We likewise do so with regard to "$\mathcal{MCFG}$" and "$\mathcal{MCFL}$" and the corresponding (sub-)classes of MCFGs and MCFLs.

Corollary 1. *For $m \geq 1$, m-$\mathcal{MCFL}(1) = m$-$\mathcal{MCFL}_{mb}(1) = m$-$\mathcal{MCFL}_{wn}(1)$.*

Proof. For $m \geq 1$, m-$\mathcal{MCFL}(1)$ and m-$\mathcal{MCFL}_{\mathrm{mb}}(1)$ are identical by definition. The identity of m-$\mathcal{MCFL}_{\mathrm{mb}}(1)$ and m-$\mathcal{MCFL}_{\mathrm{wn}}(1)$ is a special case of Proposition 1, because we have m-$\mathcal{MCFL}_{\mathrm{wn,mb}}(1)$=$m$-$\mathcal{MCFL}_{\mathrm{wn}}(1)$. □

Theorem 2. $\mathcal{MCFL} = \mathcal{MCFL}(2)$

Theorem 3. *For* $m \geq 1$, $m\text{-}\mathcal{MCFL}_{wn} = m\text{-}\mathcal{MCFL}_{wn}(2)$.

Theorem 2 is a corollary of Theorem 11 of [24]. Theorem 3 is Lemma 5 of [13].

Theorem 4. *For any* $m, r \geq 1$ *let* G *be an* m*-MCFG*(r).

(i) There is a non-deleting m*-MCFG*(r), G', *such that* $L(G) = L(G')$.
(ii) If $G \in \mathcal{MCFG}_{mb}$ *then* G' *from (i) can also be chosen from* $\mathcal{MCFG}_{mb}$.

Proof. This is Corollary 2.2.10 of [19] which essentially follows from both Lemma 2.2 and its concrete proof in [27]. □

Proposition 1. *For* $m \geq 1$, $r \in [1, 2]$, $m\text{-}\mathcal{MCFL}_{mb}(r) = m\text{-}\mathcal{MCFL}_{wn,mb}(r)$.

Proof. Let $G \in m\text{-}\mathcal{MCFG}_{\mathrm{mb}}(r)$ for some $m \geq 1$ and $r \in [1, 2]$. By (ii) of the last theorem we can w.l.o.g. assume G to be non-deleting. Now, transform G into its non-permuting "closure", i.e. a non-deleting and non-permuting m-$\mathrm{MCFG}_{\mathrm{mb}}(r)$, G', deriving the same language (cf. Construction 2.4.3 and Corollary 2.4.4 in [19]). Since each rule in G' is not only non-deleting and non-permuting, but also monadic branching, well-nestedness of such a rule holds straightforwardly. □

Proposition 2. $\mathcal{MCFL}(1) \subsetneqq \mathcal{MCFL}_{mb}$.

Proof. $\mathcal{MCFL}(1) \subseteq \mathcal{MCFL}_{\mathrm{mb}}$ is an immediate consequence of the corresponding definitions. Because of

- $\mathcal{MCFL}(1) = \mathcal{ET0L}_{\mathrm{fin}}$ and $\mathcal{ET0L}_{\mathrm{fin}} \subseteq \mathcal{EDT0L}$, cf. [4] and [24],[2]
- $\mathcal{CFL} - \mathcal{EDT0L} \neq \emptyset$, cf. [4], and
- $\mathcal{CFL} = 1\text{-}\mathcal{MCFL}(2)$ and $1\text{-}\mathcal{MCFL}(2) \subseteq \mathcal{MCFL}_{\mathrm{mb}}$

even proper inclusion of $\mathcal{MCFL}(1)$ within $\mathcal{MCFL}_{\mathrm{mb}}$ holds. □

3 Separating $\mathcal{MCFL}_{\mathrm{wn}}$ from $\mathcal{MCFL}$

In this section we briefly recapitulate the main results from [13], in order to emphasize the analogies and differences to the way of separating $\mathcal{MCFL}_{\mathrm{mb}}$ from $\mathcal{MCFL}_{\mathrm{wn}}$ presented in the next section. The first theorem is Theorem 8 of [13].

Theorem 5 (copying theorem for $\mathcal{MCFL}_{\mathrm{wn}}$). *If for some set of strings* L_0, $L = \{w\#w \mid w \in L_0\}$ *holds then for each* $m \geq 1$, *the following are equivalent:*

(i) $L \in m\text{-}\mathcal{MCFL}_{wn}$. *(ii)* $L \in m\text{-}\mathcal{MCFL}(1)$.

Corollary 2. *If for some set of strings* L_0, $L = \{w\#w \mid w \in L_0\}$ *holds then the following are equivalent:*

(i) $L \in \mathcal{MCFL}_{wn}$. *(ii)* $L \in \mathcal{MCFL}(1)$. *(iii)* $L_0 \in \mathcal{MCFL}(1)$.

[2] $\mathcal{CFL}$ denotes the class of all context-free languages. For definitions of the language classes $\mathcal{EDT0L}$ and $\mathcal{ET0L}_{\mathrm{fin}}$ as well as their origins see [4].

This is Corollary 9 of [13]. It is proven there relying on two theorems, namely, the one presented here as Theorem 5 and the equivalent version of our Theorem 1 taking into account the language $\{w\#w \mid w \in L_0\}$ instead of $\{w\#w^R \mid w \in L_0\}$.

Theorem 6 (separation theorem for $\mathcal{MCFL}_{\mathbf{wn}}$). $\mathcal{MCFL}_{wn} \subsetneqq \mathcal{MCFL}$.

This is Corollary 10 of [13] following from the last corollary combined with the facts that $\mathcal{CFL} - \mathcal{MCFL}(1) \neq \emptyset$, and that for $L_0 \in \mathcal{CFL}$, $\{w\#w \mid w \in L_0\} \in \mathcal{MCFL}$.

4 Separating $\mathcal{MCFL}_{\mathbf{mb}}$ from $\mathcal{MCFL}_{\mathbf{wn}}$

We start by presenting an analog to the copying theorem for $\mathcal{MCFL}_{\text{wn}}$.

Theorem 7 (reverse copying theorem for $\mathcal{MCFL}_{\mathbf{mb}}$). *If for some set of strings L_0, $L = \{w\#w^R \mid w \in L_0\}$ holds then for each $m \geq 1$, (i') implies (ii'):*

(i') $L \in m\text{-}\mathcal{MCFL}_{mb}$.
(ii') $L \in m+1\text{-}\mathcal{MCFL}_{wn}(1)$ *and* $L_0 \in m+1\text{-}\mathcal{MCFL}_{wn}(1)$.

Corollary 3. *If for some set of strings L_0, $L = \{w\#w^R \mid w \in L_0\}$ holds then the following are equivalent:*

(i) $L \in \mathcal{MCFL}_{mb}$. *(ii)* $L \in \mathcal{MCFL}(1)$. *(iii)* $L_0 \in \mathcal{MCFL}(1)$.

Proof. "(iii)⇒(ii)": special case of Theorem 1. "(ii)⇒(i)": cf. Proposition 2. "(i)⇒(iii)": this is a corollary of Theorem 7. □

Lemma 1. *For each $L_0 \in \mathcal{CFL}$, $L = \{w\#w^R \mid w \in L_0\} \in 2\text{-}\mathcal{MCFL}_{wn}$.*

Proof. Starting, e.g., with a CFG in Chomsky normal form generating L_0, the construction of an 2-MCFG$_{\text{wn}}(2)$ generating L is straightforward. □

Theorem 8 (separation theorem for $\mathcal{MCFL}_{\mathbf{mb}}$). $\mathcal{MCFL}_{mb} \subsetneqq \mathcal{MCFL}_{wn}$.

Proof. Choose existing $L_0 \in \mathcal{CFL} - \mathcal{MCFL}(1)$. Then, by Theorem 7 and 1, $L = \{w\#w^R \mid w \in L_0\} \in 2\text{-}\mathcal{MCFL}_{\text{wn}} - \mathcal{MCFL}_{\text{mb}}$. □

The remaining part of this section is devoted to a detailed description of the crucial points underlying a proof of Theorem 7.

Proof (sketch) of Theorem 7. For some $m \geq 1$, let $L \in m\text{-}\mathcal{MCFL}_{\text{mb}}$. When having shown that $L \in m+1\text{-}\mathcal{MCFL}_{\text{wn}}(1)$ holds, $L_0 \in m+1\text{-}\mathcal{MCFL}_{\text{wn}}(1)$ follows from Theorem 1 and Corollary 1(ii).

Let $G = \langle N, \Sigma \cup \{\#\}, P, S\rangle$ be an m-MCFG$_{\text{mb}}$ with $L(G) = L$. W.l.o.g. G is well-nested by Proposition 1, thus, in particular, each $p \in P$ is non-deleting. Moreover, we can w.l.o.g. assume that each $A \in N$ is useful and derives an infinite set of tuples of strings over Σ, i.e., $\{\langle w_1, \ldots, w_k\rangle \in (\Sigma^*)^k \mid \vdash_G A(w_1, \ldots, w_k)\}$ is infinite for $k = \mathit{arity}(A)$. Trivially, Σ can be chosen such that $\Sigma \cap \{\#\} = \emptyset$.

Depending on G, in (21)-(24) we construct a $G' \in m+1\text{-}\mathcal{MCFG}_{\mathrm{wn}}(1)$ with $L(G') = L$. Before doing so, the crucial properties of G virtually employed by G' are carefully revealed step by step, and the presented technical details providing a precise characterization of those properties are summed up in Fig. 4. In a nutshell, we are concerned with the following situation as to G:

> if $\vdash_G T : S(\widehat{w}\,\#\,\widehat{w}^R)$ for some derivation tree T over Δ_P and $\widehat{w} \in \Sigma^*$, then looking at T from a bottom-up perspective, the unique instance of "#" appearing in the derived string $\widehat{w}\#\widehat{w}^R$ is successively passed on upward from the leftmost leaf of the tree to the root, i.e. along the tree's leftmost path, and within no other node of the tree any instance of # is created or manipulated in another way. (9)

Consider $p \in P$ with $rank(p) = 2$. Because it is monadic branching, p is of the form $B_0(\alpha_1, \ldots, \alpha_{k_0}) \leftarrow B_1(x_{1,1}, \ldots, x_{1,k_1}), \ldots, B_n(x_{n,1}, \ldots, x_{n,k_n})$ according to (1) such that $n = 2$ and $k_1 = 1$. We set $A = B_0$, $B = B_1$ and $C = B_2$, and also $k = k_0$, $l = k_2$, $x'_0 = x_{1,1}$ and $x'_i = x_{2,i}$ for $i \in [1, k_2]$. Thus, p is of the form

$$A(\alpha_1, \ldots, \alpha_k) \leftarrow B(x'_0), C(x'_1, \ldots, x'_l) \tag{10}$$

Since A, B and C are useful, there are $v_i \in (\Sigma \cup \{\#\})^*$ for $i \in [1, k]$, $u_i \in (\Sigma \cup \{\#\})^*$ for $i \in [0, l]$ and derivation trees T_B and T_C over Δ_P, and there are $y \in Y$, $x''_i \in X$ for $i \in [1, k]$, $\alpha \in (\Sigma \cup \{x''_i \mid i \in [1, k]\})^*$ and a derivation tree context $\widetilde{T}_S$ over $\Delta_P(Y)$ such that

$$\vdash_G pT_BT_C : A(v_1, \ldots, v_k) \quad \text{and} \quad y : A(x''_1, \ldots, x''_k) \vdash_G \widetilde{T}_S : S(\alpha) \tag{11}$$

and

$$\vdash_G T_B : B(u_0) \quad \text{and} \quad \vdash_G T_C : C(u_1, \ldots, u_l) \tag{12}$$

We will crucially show that (16) and (18) and, therefore, (20) hold, i.e., we will show a) that u_0 contains exactly one instance of #, while for $i \in [1, l]$, u_i does not contain any such instance, b) that $l > 1$ and c) that, therefore, in case $k > 1$, A cannot appear itself on the righthand side of any strictly binary rule from P. These properties essentially imply (9).

a) Since by choice of G each rule is non-deleting, and because each $\widetilde{w} \in L(G)$ is of the form $w\#w^R$ for some $w \in \Sigma^*$, from (11) and (12), it follows that

> $u_i \in \Sigma^*\{\#, \varepsilon\}\Sigma^*$ holds for each $i \in [0, l]$, but $u_i \in \Sigma^*\{\#\}\Sigma^*$ is true for at most one $i \in [0, l]$, (13)

and, in particular, there exist a unique $j_0 \in [1, k]$ and $v, \overline{v} \in (\Sigma \cup \{\#\})^*$ with

$$v_{j_0} = v\, u_0\, \overline{v} \tag{14}$$

Suppose, $u_0 \in \Sigma^*$. Then again, because of (11) and (12), and since by choice of G each of its rules is non-deleting, there are $w_1, w_2 \in \Sigma^*$ such that, w.l.o.g.,

$$\vdash_G S(w_1u_0w_2\#(w_1u_0w_2)^R) \quad \text{and thus,} \quad \vdash_G S(w_1u'w_2\#(w_1u_0w_2)^R) \tag{15}$$

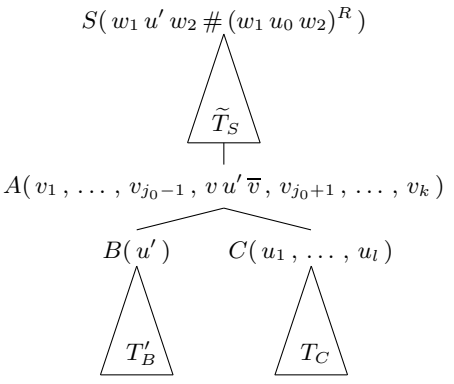


Fig. 2. Derivation tree according to (11)-(15)

whenever $\vdash_G T'_B : B(u')$ for some $u' \in \Sigma^*$ and some derivation tree T'_B over Δ_P.

Figure 2 depicts the situation as fixed in (11)-(15). However, having chosen G such that, in particular, the nonterminal B derives an infinite set of strings over Σ, (15) yields a contradiction to the fact that each element in $L(G)$ is of the form $w\#w$ for some $w \in \Sigma^*$. In other words, in combination with (13), we have

$$u_0 \in \Sigma^*\{\#\}\Sigma^* \quad \text{and} \quad u_i \in \Sigma^* \text{ for } i \in [1, l] \tag{16}$$

b) Suppose, $l = 1$. Then, we can again derive a contradiction. We can do so analogously to the case resulting from the assumption that $u_0 \in \Sigma^*$: because $u_1 \in \Sigma^*$ by (16), we can conclude that there are $w_1, w_2 \in \Sigma^*$ such that, w.l.o.g.,

$$\vdash_G S(w_1u_1w_2\#(w_1u_1w_2)^R) \quad \text{and thus,} \quad \vdash_G S(w_1u'w_2\#(w_1u_1w_2)^R) \tag{17}$$

whenever $\vdash_G C(u')$ for some $u' \in \Sigma^*$. By choice of G, the nonterminal C derives an infinite set of strings over Σ, and therefore the assumption $l = 1$ allows us to derive strings from S which are not in $L(G)$. Thus, it must hold that

$$l > 1 \tag{18}$$

c) Let T_S be the derivation tree over Δ_P which results from substituting the variable $y \in Y$ within the derivation context $\widetilde{T}_S$ over $\Delta_P(Y)$ by pT_BT_C. Recall, once more, that each rule of G is non-deleting. Thus, from (11)-(14) and (16) it, moreover, follows that there are $u, \overline{u}, w, \overline{w} \in \Sigma^*$ such that

$$u_0 = u\,\#\,\overline{u} \quad , \quad v_i \in \Sigma^* \text{ for } i \neq j_0 \quad \text{and} \quad \vdash_G T_S : S(w\, v\, u\,\#\,\overline{u}\,\overline{v}\,\overline{w}) \tag{19}$$

The situation as fixed in (11)-(14), (16) and (19) is displayed in Fig. 3. Taking into account the above considerations on the nonterminal C, in particular, the properties expressed in (16) and (18), it becomes clear that in case $k > 1$,

$$A \text{ cannot appear on the righthand side of any } p' \in P \text{ with } \mathit{rank}(p') = 2. \tag{20}$$

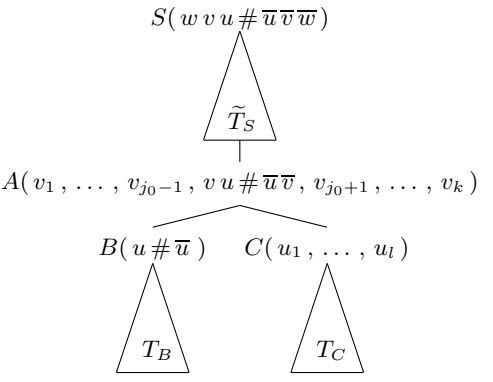


Fig. 3. Derivation tree T_S and intermediately derived "objects."

If A did so, $L(G)$ would, contradicting its definition, include a string consisting of more than one instance of #. Recall that $v_{j_0} \in \Sigma^*\{\#\}\Sigma^*$ by (14) and (16).

Thus, T_S and wvu are in fact respective instances of a derivation tree T and a string $\widehat{w}$ in the sense of the above "nutshell" (9). More concretely, for some $m(S) \geq 0$ there is a finite sequence of derivation tree contexts over $\Delta_P(Y)$, $\langle V_j \rangle_{0 \leq j \leq m(S)}$, such that V_0 is a tree over Δ_P with no occurrence of variables, and such that for $j \in [1, m(S)]$, V_j is a tree over $\Delta_P(\{y_j\})$ with exactly on instance of y_j occurring in V_j.

Furthermore, if $W_0 = V_0$, and if for $j \in [1, m(S)]$, W_j is the result of substituting y_j within V_j by W_{j-1} then $W_{m(S)} = T_S$.

Each V_j is built up in the following way:

- There are particular numbers $n(j) = n \geq 0$ and $s_i \geq 0$ for $i \in [0, n]$.
- There are nonterminals $B^{(i)} \in N$ and $C^{(i,i')} \in N$ for $i \in [0, n]$ and $i' \in [0, s_i]$ with $arity(B^{(i)}) = 1$ for $i \in [1, n]$, $arity(C^{(0,0)}) = 1$, and $C^{(0,s_0)} = S$ if $s_0 = 0$.
- We let
$$\begin{aligned} r_i &:= arity(B^{(i)}) \quad \text{for} \quad i \in [0, n] \\ l(i, i') &:= arity(C^{(i,i')}) \quad \text{for} \quad i \in [0, n]\ ,\ i' \in [0, s_i] \end{aligned}$$
- Then, there is a set of pairwise distinct variables
$$X_j = \{x_{i''}^{(i)}, x_{i'''}^{(i,i')} \mid i \in [0, n], i'' \in [1, r_i], i' \in [0, s_i], i''' \in [1, l(i, i')]\} \subseteq X$$

and there are

$$\begin{aligned} \alpha_{i''}^{(i)} &\in (\Sigma \cup \{\#\} \cup X_j)^* \quad \text{for} \quad i \in [0, n]\ ,\ i'' \in [0, r_i] \\ \alpha_{i''}^{(0)} &\in (\Sigma \cup \{\#\})^* \quad \text{for} \quad i'' \in [0, r_0] \quad \text{in case} \quad n = 0 \\ \beta_{i'''}^{(0,i')} &\in (\Sigma \cup \{\#\} \cup X_j)^* \quad \text{for} \quad i' \in [0, s_0]\ ,\ i''' \in [1, l(0, i')] \\ \beta_{i'''}^{(i,i')} &\in (\Sigma \cup X_j)^* \quad \text{for} \quad i \in [1, n]\ ,\ i' \in [0, s_i - 1]\ ,\ i''' \in [1, l(i, i')] \\ u_{i'''}^{(i,s_i)} &\in \Sigma^* \quad \text{for} \quad i \in [1, n]\ ,\ i''' \in [1, l(i, s_i)] \end{aligned}$$

- such that for $i \in [0, n-1]$, there are non-terminating rules from P of the form

$$p^{(i)} = B^{(i)}(\alpha_1^{(i)}, \ldots, \alpha_{r_i}^{(i)}) \leftarrow B^{(i+1)}(x_1^{(i+1)}), C^{(i+1,0)}(x_1^{(i+1,0)}, \ldots, x_{l(i+1,0)}^{(i+1,0)})$$

and such that in case $n = 0$,[3] there is a terminating rule from P of the form

$$p^{(0)} = B^{(0)}(\alpha_1^{(0)}, \ldots, \alpha_{r_0}^{(0)}) \leftarrow$$

- Furthermore, for $i \in [1, n]$, there are terminating rules from P of the form

$$q^{(i,s_i)} = C^{(i,s_i)}(u_1^{(i,s_i)}, \ldots, u_{l(i,s_i)}^{(i,s_i)}) \leftarrow$$

$$\text{while} \quad q^{(0,s_0)} = p^{(0)} \quad , \quad \text{implying that} \quad C^{(0,s_0)} = B^{(0)} \quad , \quad \text{and}$$

for $i \in [0, n]$, $i' \in [0, s_i - 1]$, there are unary branching rules from P of the form

$$q^{(i,i')} = C^{(i,i')}(\beta_1^{(i,i')}, \ldots, \beta_{l(i,i')}^{(i,i')}) \leftarrow C^{(i,i'+1)}(x_1^{(i,i'+1)}, \ldots, x_{l(i,i'+1)}^{(i,i'+1)})$$

- For $i \in [1, n]$, we now define derivation trees over Δ_P by

$$T^{(i,s_i)} := q^{(i,s_i)} \quad \text{and} \quad T^{(i,i')} := q^{(i,i')} T^{(i,i'+1)} \quad \text{for} \quad i' \in [0, s_i - 1]$$

and for $y_j \in Y$, we define derivation tree contexts over $\Delta_P(\{y_j\})$ by

$$\begin{aligned} U^{(n-1)} &:= p^{(n-1)} y_j T^{(n,0)} && \text{in case} \quad n > 0 \\ U^{(i)} &:= p^{(i)} U^{(i+1)} T^{(i+1,0)} && \text{for} \quad i \in [0, n-2] \\ U^{(0)} &:= p^{(0)} && \text{in case} \quad n = 0 \end{aligned}$$

Finally, we set

$$T^{(0,s_0)} := U^{(0)} \quad , \quad T^{(0,i')} := q^{(0,i')} T^{(0,i'+1)} \quad \text{for} \quad i' \in [0, s_0 - 1] \quad \text{and} \quad V_j := T^{(0,0)}$$

- Thus,

$$y_j : B^{(n)}(x_1^{(n)}) \vdash_G V_j : C^{(0,0)}(\beta_>) \quad \text{if} \quad n > 0 \quad \text{and} \quad \vdash_G V_j : C^{(0,0)}(\beta_=) \quad \text{if} \quad n = 0$$

for some $\beta_> \in (\Sigma \cup \{\#, x_1^{(n)}\})^*$ if $n > 0$, and $\beta_= \in (\Sigma \cup \{\#\})^*$ if $n = 0$. Recall that $arity(C^{(0,0)}) = 1$ in general, and that $arity(B^{(n)}) = 1$ in case $n > 0$.

Figure 4 aims at making the formal setting as it regards the derivation tree context V_j somewhat more accessible. Note that for $i \in [0, n-1]$, the respective calculation of the contributions of $B^{(i+1)}$ and $C^{(i+1,0)}$ to $B^{(i)}$ are independent of each other. Crucially, from a bottom-up perspective, the calculation of the contribution of $B^{(i+1)}$ can be done first and can be stored in a buffer, while calculating the contribution of C^{i+1}. In terms of the arity of a nonterminal the buffer size needed is 1, since for each $i \in [0, n-1]$, $B^{(i+1)}$ has arity 1. Exactly this property is used below in order to define the $m+1$-MCFG(1), G', based on the given m-MCFG$_{\text{mb}}$, G, with $L(G') = L(G)$: in terms of the transformed grammar, the subderivation trees $T(i, 0)$ for $i \in [1, n]$, i.e. the "ⓘ-parts" of

[3] Note that $n = 0$ implies $\{p^{(i)} \mid i \in [0, n-1]\} = \emptyset$.

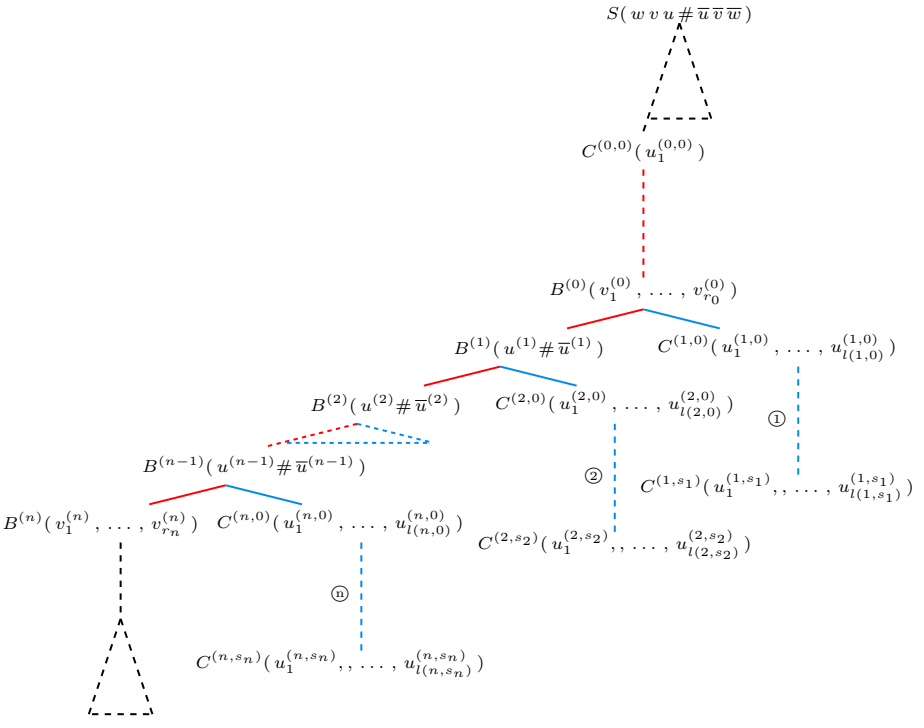


Fig. 4. Typical configuration within derivation tree T_S corresponding to derivation tree context V_j over $\Delta_P(\{y_j\})$ for arbitrary $j \in [0, m(S)]$

the original derivation tree context V_j (cf. Fig. 4), become integral parts of the leftmost path resulting in a completely unary branching derivation tree (cf. Fig. 5).

For $\{[A/B] \mid A, B \in N\}$ being a set of pairwise distinct new symbols, define now $G' = \langle N', \Sigma, P', S\rangle \in m+1\text{-}\mathcal{MCFG}(1)$ depending on G with $L(G') = L$.

- The set of nonterminals $N' = \bigcup_{k\geq 0} N'^{(k)}$ is defined by $N'^{(0)} = \emptyset$ and

$$N'^{(k+1)} = N^{(k+1)} \cup \{[A/B] \mid A \in N^{(k)}, B \in N^{(1)}\} \quad \text{for } k \geq 0 \tag{21}$$

- In order to define the rule set P', we distinguish three types of rules in P.

– A binary branching $p \in P$ is of the form $A(\alpha_1, \ldots, \alpha_k) \leftarrow B(x'_0), C(x'_1, \ldots, x'_l)$ in accordance with (10). For each such $p \in P$ we let

$$A(\alpha_1, \ldots, \alpha_k) \leftarrow [C/B](x'_0, x'_1, \ldots, x'_l) \in P' \tag{22}$$

– A unary branching $p \in P$ is of the form $A(\alpha_1, \ldots, \alpha_k) \leftarrow C(x'_1, \ldots, x'_l)$ according to (11), where $k = k_0$, $l = k_1$, $A = B_0$, $C = B_1$, $x'_i = x_{1,i}$ for $i \in [1, l]$. For each such $p \in P$, each $B \in N^{(1)}$, and some $x'_0 \in X - \{x'_i \mid i \in [1, l]\}$ let

$$p \in P' \quad \text{and} \quad [A/B](x'_0, \alpha_1, \ldots, \alpha_k) \leftarrow [C/B](x'_0, x'_1, \ldots, x'_l) \in P' \tag{23}$$

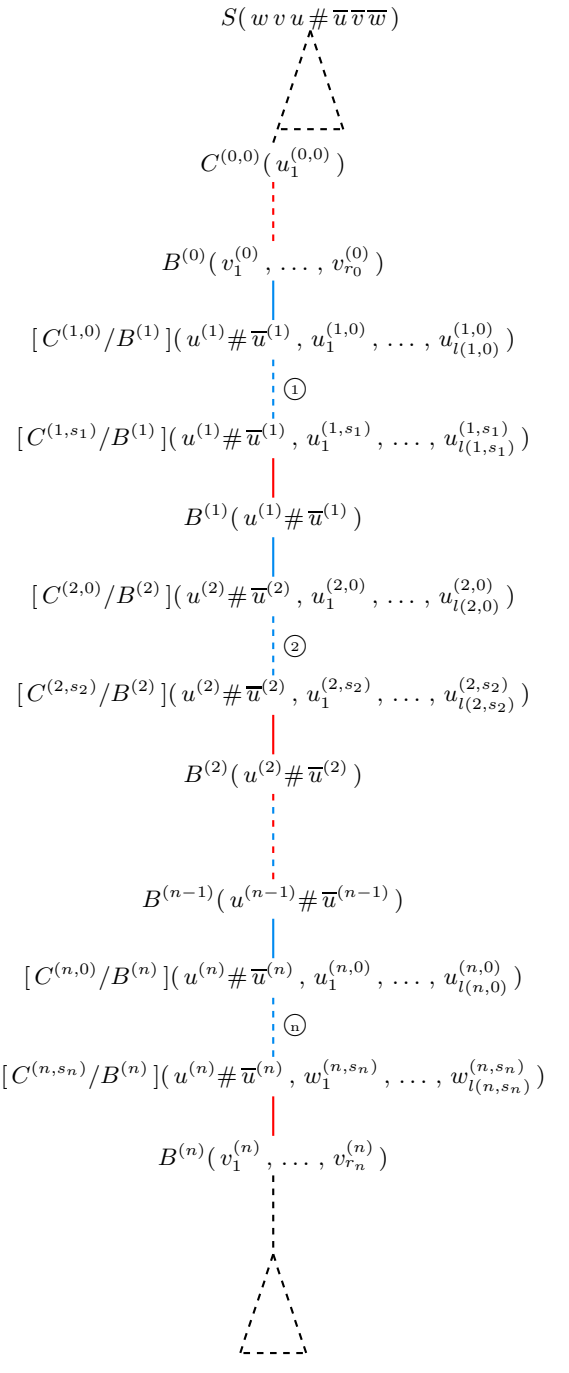


Fig. 5. Typical configuration within derivation tree of G' corresponding to the transformed derivation tree context V_j over $\Delta_P(\{y_j\})$ for arbitrary $j \in [0, m(S)]$

– A terminating $p \in P$ is of the form $A(w_1, \ldots, w_k) \leftarrow$ in accordance with (11), where $k = k_0$, $A = B_0$ and $w_i = \alpha_i$ for $i \in [1, k]$. For each such $p \in P$, each $B \in N^{(1)}$, and some $x'_0 \in X - \{x'_i \mid i \in [1, l]\}$ let

$$p \in P' \quad \text{and} \quad [A/B](x'_0, w_1, \ldots, w_k) \leftarrow B(x'_0) \in P' \tag{24}$$

An induction on the length of a derivation showed that $L(G') = L(G)$. Due to (20), in G' we do not have to "lift" by means of "$[\cdot/B]$" over the lefthand side of a binary branching rule from G, cf. (22). Rather, the "lifting" instantiated in (24) and inherited in (23) is validated in (22). Instead of giving more details we refer back to the considerations above and point to Fig. 4 and 5 depicting how a derivation tree context V_j is transformed to a corresponding one in terms of G'.

5 Conclusion

We have characterized the separation of monadic branching MCFGs, and thus, MGs obeying the shortest move condition (SMC) and the specifier island condition (SPIC), from well-nested MCFGs by means of a "simple reverse copying" theorem concerning the derivable languages. Solving a generally open problem, the result also provides a direct comparison to the separation of well-nested MCFGs from MCFGs, and thus, MGs only obeying the SMC, by means of an already known "simple copying" theorem.

The SPIC provides a rather canonical restriction within the MG-setting.[4] *Well-nestedness* provides a rather canonical restriction on MCFGs, or reversing the perspective, within the MCFG-framework well-nested MCFGs constitute a natural generalization of, e.g., *tree adjoining grammars*, the former crucially preserving the well-nestedness property of the latter.[5] Since on the other hand, in terms of MGs, *well-nestedness* seems to be a rather *ad hoc* restriction, whereas for MCFGs, this seems to hold with regard to the SPIC, our result may suggest that we are concerned here with a structural difference between MGs and MCFGs which cannot immediately be overcome in a non-stipulated manner.

References

1. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
2. Chomsky, N.: Derivation by phase. In: Kenstowicz, M. (ed.) Ken Hale. A Life in Language, pp. 1–52. MIT Press, Cambridge (2001)
3. Chomsky, N.: On phases. In: Freidin, R., Otero, C., Zubizaretta, M.L. (eds.) Foundational Issues in Linguistic Theory, pp. 133–166. MIT Press, Cambridge (2008)
4. Engelfriet, J., Rozenberg, G., Slutzki, G.: Tree transducers, L systems, and two-way machines. Journal of Computer and System Sciences 20, 150–202 (1980)

[4] Independently of possible, linguistically motivated objections, the SPIC might, e.g., be interpreted as a "generalized" *generalized left branch condition* in the sense of [7].

[5] It has even be argued that there are good reasons to think that well-nestedness should be an essential property of the concept of *mild context-sensitivity*, cf. [12].

5. Gärtner, H.-M., Michaelis, J.: A note on the complexity of constraint interaction: Locality conditions and minimalist grammars. In: Blache, P., Stabler, E., Busquets, J., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 114–130. Springer, Heidelberg (2005)
6. Gärtner, H.M., Michaelis, J.: Some remarks on locality conditions and minimalist grammars. In: Sauerland, U., Gärtner, H.M. (eds.) Interfaces + Recursion = Language?, pp. 161–195. Mouton de Gruyter, Berlin (2007)
7. Gazdar, G.: Unbounded dependencies and coordinate structure. Linguistic Inquiry 12, 155–184 (1981)
8. Girard, J.Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)
9. de Groote, P., Morrill, G., Retoré, C. (eds.): LACL 2001. LNCS (LNAI), vol. 2099. Springer, Heidelberg (2001)
10. Harkema, H.: A characterization of minimalist languages. In: de Groote, P. et al (eds.) [9], pp. 193–211
11. Joshi, A.K.: Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In: Dowty, D.R., Karttunen, L., Zwicky, A.M. (eds.) Natural Language Parsing, pp. 206–250. Cambridge University Press, New York (1985)
12. Kanazawa, M.: The convergence of well-nested mildly context-sensitive grammar formalisms (2009), invited talk held at FG-2009, Bordeaux
13. Kanazawa, M., Salvati, S.: The copying power of well-nested multiple context-free grammars. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 344–355. Springer, Heidelberg (2010)
14. Kobele, G.M.: Generating Copies. An investigation into structural identity in language and grammar. Ph.D. thesis, University of California, Los Angeles (2006)
15. Kobele, G.M., Michaelis, J.: Two type-0 variants of minimalist grammars. In: Rogers, J. (ed.) [25], pp. 81–91
16. Koopman, H., Szabolcsi, A.: Verbal Complexes. MIT Press, Cambridge (2000)
17. Kracht, M.: The Mathematics of Language. Mouton de Gruyter, Berlin (2003)
18. Michaelis, J.: Derivational minimalism is mildly context-sensitive. In: Moortgat, M. (ed.) LACL 1998. LNCS (LNAI), vol. 2014, pp. 179–198. Springer, Heidelberg (2001)
19. Michaelis, J.: On Formal Properties of Minimalist Grammars. Linguistics in Potsdam 13, Universitätsbibliothek, Publikationsstelle, Potsdam, Ph.D. thesis (2001)
20. Michaelis, J.: Transforming linear context-free rewriting systems into minimalist grammars. In: de Groote, et al (eds.) [9], pp. 228–244
21. Michaelis, J.: Observations on strict derivational minimalism. Electronic Notes in Theoretical Computer Science 53, 192–209 (2004)
22. Michaelis, J.: An additional observation on strict derivational minimalism. In: Rogers, J. (ed.) [25], pp. 101–111
23. Mönnich, U.: Some remarks on mildly context-sensitive copying. In: Hanneforth, T., Fanselow, G. (eds.) Language and Logos, pp. 367–389. Akad. Verlag, Berlin (2010)
24. Rambow, O., Satta, G.: Independent parallelism in finite copying parallel rewriting systems. Theoretical Computer Science 223, 87–120 (1999)
25. Rogers, J.: Proceedings of FG-MoL 2005. CSLI Publications, Stanford (2009)
26. Salvati, S.: Minimalist grammars in the light of logic. Research Report, INRIA Bordeaux (2011), http://hal.inria.fr/inria-00563807/en/
27. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)

28. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
29. Stabler, E.P.: Remnant movement and complexity. In: Bouma, G., Kruijff, G.J.M., Hinrichs, E., Oehrle, R.T. (eds.) Constraints and Resources in Natural Language Syntax and Semantics, pp. 299–326. CSLI Publications, Stanford (1999)
30. Stabler, E.P.: Recognizing head movement. In: de Groote, P., et al [9], pp. 245–260
31. Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 616–641. Oxford University Press, New York (2011)
32. Stabler, E.P., Keenan, E.L.: Structural similarity within and among languages. Theoretical Computer Science 293, 345–363 (2003)
33. Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: 25th Annual Meeting of the Association for Computational Linguistics, Stanford, CA, pp. 104–111. ACL (1987)
34. Villemonte de la Clergerie, É.: Parsing mcs languages with thread automata. In: Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms, Venezia, pp. 101–108 (2002)

Minimalist Tree Languages Are Closed Under Intersection with Recognizable Tree Languages

Gregory M. Kobele

University of Chicago
kobele@uchicago.edu

Abstract. Minimalist grammars are a mildly context-sensitive grammar framework within which analyses in mainstream chomskyian syntax can be faithfully represented. Here it is shown that both the derivation tree languages and derived tree languages of minimalist grammars are closed under intersection with regular tree languages. This allows us to conclude that taking into account the possibility of 'semantic crashes' in the standard approach to interpreting minimalist structures does not alter the strong generative capacity of the formalism. In addition, the addition to minimalist grammars of complexity filters is easily shown using a similar proof method to not change the class of derived tree languages.

Minimalist grammars (in the sense of [1]) are a formalization of mainstream chomskyian syntax. In this paper I will show that both derived and derivation tree languages of minimalist grammars are closed under intersection with regular tree languages. The technique used in the proofs of this fact is similar to that of [2], where non-terminals of context-free derivation trees were paired with states of an automaton. While the closure of the derived tree languages under regular intersection can be seen to follow from that of the derivation tree languages (by virtue of the monadic second order relation between the two), the proof method extends immediately to cases of linguistic interest where the connection is not as obvious, as in the case of 'complexity filters' in the sense of [3]. The closure of derived tree languages under regular intersection guarantees that the kind of semantic interpretation performed in the minimalist literature [4], which makes use of only a finite domain of types, cannot in virtue of partiality (semantic 'crashes') lead to sets of semantically well-formed trees which could not be directly derived by some minimalist grammar.

The remainder of the paper is organized as follows. The next section introduces minimalist grammars, as well as some relevant notation. Section 2 contains the proofs of closure under intersection with regular tree languages of both derivation and derived tree languages of minimalist grammars. Consequences and extensions of linguistic relevance are discussed in section 3. Finally, section 4 concludes.

1 Formal Preliminaries

Given a finite set A, A^* denotes the set of all finite sequences of elements over A. The symbol ϵ denotes the empty sequence. A ranked alphabet is a finite set

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 129–144, 2011.

F together with a function $\mathbf{rank} : F \to \mathbb{N}$ mapping each symbol in F to a natural number indicating its arity. Given $f \in F$ with arity $n = \mathbf{rank}(f)$, I will sometimes write $f^{(n)}$ to denote f while indicating that it has arity n. The set $T(F)$ of terms over a ranked alphabet F is the smallest subset of F^* containing all $f^{(0)} \in F$, and such that whenever it contains $t_1, \ldots, t_n$, it contains $f^{(n)}t_1 \cdots t_n$ for each $f^{(n)} \in F$. In lieu of writing $f^{(n)}t_1 \cdots t_n$, I will insert parentheses and commas for readability, writing in its stead $f(t_1, \ldots, t_n)$. Regular subsets of terms over an alphabet F can be given in terms of bottom-up tree automata, which are tuples $A = \langle Q, (\delta_f)_{f \in F} \rangle$, where each δ_f is a $\mathbf{rank}(f)$-ary function over Q. An automaton $A = \langle Q, (\delta_f)_{f \in F} \rangle$ induces a function $A : T(F) \to Q$ in the following manner: $A(f^{(n)}(t_1, \ldots, t_n)) = \delta_f(A(t_1), \ldots, A(t_n))$. For each state $q \in Q$, the set of terms $A^{-1}(q) = \{t \in T_F : A(t) = q\}$ mapped by A to the state q is a regular term language. It will be convenient in the following to treat A as operating over $T(F \cup Q)$, where elements of Q are treated as nullary symbols. In this case, $A(f^{(n)}(s_1, \ldots, s_n)) = \delta_f(q_1, \ldots, q_n)$, where $q_i = s_i$ if $s_i \in Q$, and $A(s_i)$ otherwise.

A partial function f from A to B is a total function from A to $B \cup \{\star\}$. If $f : A \to B$ is a partial function, we say it is undefined at $a \in A$ if $f(a) = \star$. The everywhere undefined function is denoted $\emptyset$. Given partial functions $f, g : A \to B$, their union $f \oplus g$ is defined iff there is no $a \in A$ such that both f and g are defined at a and $f(a) \neq g(a)$. In this case $(f \oplus g)(a) = \mathbf{if}\ f(a) = \star\ \mathbf{then}\ g(a)\ \mathbf{else}\ f(a)$. Given partial $f : A \to B$ and $a \in A$, $(f_{/a})(b) = \mathbf{if}\ b = a\ \mathbf{then}\ \star\ \mathbf{else}\ f(b)$. Given a subset $B \subset A^*$ with the property that $aw, au \in B$ implies that $w = u$, B can be viewed as the partial function f_B from A to A^* such that $f_B(a) = aw$ if $aw \in B$ and is undefined otherwise. In particular, given $aw \in A^*$, $\{aw\} : A \to A^*$ is the partial function defined only at a.

1.1 Minimalist Grammars

A minimalist grammar is given by a four-tuple $G = \langle \Sigma, \mathbf{sel}, \mathbf{lic}, Lex \rangle$ where Σ is a finite set, **sel** and **lic** are finite sets of *selection* and *licensing* features respectively which determine a set $\mathbb{F} := \{=x, x, +y, -y : x \in \mathbf{sel}, y \in \mathbf{lic}\}$ of *features*, $Lex \subset \Sigma \times \mathbb{F}^*$ is a finite set of lexical items. Features of the form $=x$ are *selector* features, those of the form $+y$ are *licensor* features, and those of the form x ($-y$) are *selectee* (*licensee*) features. Treating elements of Σ as nullary symbols, we define a ranked alphabet $S := \Sigma \cup \{\mathtt{t}^{(0)}, \bullet^{(2)}\}$. A term $t \in T(S)$ is *headed* by its right-most leaf. Conversely, a term $t' \in T(S)$ is a *maximal projection* (of its head) in t iff either $t' = t$ or there is a unary context $C[x]$, and some $t'' \in T(S)$ such that $t = C[\bullet(t', t'')]$ – in other words, t' is the left-daughter of some node. The expressions $L(G)$ generated by a minimalist grammar G is the smallest subset of $(T(S) \times \mathbb{F}^+)^+$ which 1) contains Lex, and 2) is closed under the operations presented in inference rule format below. In the rules below, $m, n \leq |\mathbf{lic}|$, $1 \leq i, j \leq m, n$, $\phi_i, \psi_j \in \Sigma \times \mathbb{F}^+$, $\gamma, \delta \in \mathbb{F}^+$, and

$s_1, s_2 \in T(S)$.[1] In addition, we require in $move_1$ and $move_2$ that ϕ_i is the only pair whose first feature is $-c$.[2]

$$\frac{\langle s_1, {=}c\gamma\rangle, \phi_1, \ldots, \phi_m \qquad \langle s_2, c\rangle, \psi_1, \ldots, \psi_n}{\langle \bullet(s_2, s_1), \gamma\rangle, \phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_n}\ merge_1$$

$$\frac{\langle s_1, {=}c\gamma\rangle, \phi_1, \ldots, \phi_m \qquad \langle s_2, c\delta\rangle, \psi_1, \ldots, \psi_n}{\langle \bullet(\mathtt{t}, s_1), \gamma\rangle, \phi_1, \ldots, \phi_m, \langle s_2, \delta\rangle, \psi_1, \ldots, \psi_n}\ merge_2$$

$$\frac{\langle s_1, {+}c\gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \langle s_2, {-}c\rangle, \phi_{i+1}, \ldots, \phi_m}{\langle \bullet(s_2, s_1), \gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \phi_{i+1}, \ldots, \phi_m}\ move_1$$

$$\frac{\langle s_1, {+}c\gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \langle s_2, {-}c\delta\rangle, \phi_{i+1}, \ldots, \phi_m}{\langle \bullet(\mathtt{t}, s_1), \gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \langle s_2, \delta\rangle, \phi_{i+1}, \ldots, \phi_m}\ move_2$$

An element of $L(G)$ is *complete* iff it is of the form $\langle s, c\rangle$. An element of $T(S)$ is a derived (or *surface*) tree iff it is the first component of a complete expression. The derived tree language of G at selectee feature c is defined to be the set of tree components of complete expressions with feature c, $L_c(G) := \{s : \langle s, c\rangle \in L(G)\}$.

The string language of G at selectee feature c (written $S_c(G)$) is defined to be the image of $L_c(G)$ under the mapping $\mathsf{y} : T(S) \to (S_0 - \{\mathtt{t}\})^*$ which maps a tree to the string consisting of its leaves in left-to-right order, ignoring traces; $\mathsf{y}(\mathtt{t}) = \epsilon$, $\mathsf{y}(s^{(0)}) = s$, and $\mathsf{y}(\bullet(t_1, t_2)) = \mathsf{y}(t_1)^\frown\mathsf{y}(t_2)$.

The following obvious proposition shows that every maximal projection is headed by some leaf.

Proposition 1. *Let G and $x \in \mathbb{F}$ be arbitrary. Then for every $t \in L_x(G)$ there is a unique maximal decomposition* LexProj(t) *of t into lexicalized maximal projections of the form* $\mathrm{TCon}_s^n[t_1, \ldots, t_n]$*, where $\mathtt{t} \neq s^{(0)} \in S$, $t_1, \ldots, t_n \in T(S)$, and* TCon_s^n *is the n-ary context over $T(S)$ defined as follows:*

1. $\mathrm{TCon}_s^0 := s$
2. $\mathrm{TCon}_s^{n+1} := \bullet(x_{n+1}, \mathrm{TCon}_s^n)$

Proof. This follows from the observation that every occurance of a trace ($\mathtt{t}$) in the surface tree of a complete expression is already itself a maximal projection (by inspection of the operations), and thus that every maximal projection is 'lexicalized'. □

Treating elements of Lex as nullary symbols, we define a ranked alphabet $U := Lex \cup \{\mathbf{r}^{(2)}, \mathbf{v}^{(1)}\}$. The set of derivation (or *underlying*) terms over G is $D(G) := T(U)$. Given $d \in D(G)$, we write $\mathsf{ev}(d)$ to denote the expression which is the evaluation of d in $L(G)$, if it exists ($\mathsf{ev} : D(G) \to L(G)$ is the partial injective mapping $\mathsf{ev}(\ell) = \ell$, $\mathsf{ev}(\mathbf{v}(d)) = move(\mathsf{ev}(d))$, and $\mathsf{ev}(\mathbf{r}(d_1, d_2)) = merge(\mathsf{ev}(d_1), \mathsf{ev}(d_2))$.)

[1] In the following rules, all move and merge operations are 'to the left'. This simplification made for expository purposes does not affect the results obtained in this paper.

[2] This constraint is called the *Shortest Move Constraint* (SMC).

We identify the set of well-formed, or *convergent*, derivations with the language (at a particular state) of a bottom up tree automaton $A_G = \langle Q, (\delta_f)_{f \in F} \rangle$, which is defined next. Given Lex we define $\mathsf{suf}(Lex) = \{\eta : \exists \sigma \in \Sigma^*, \gamma \in \mathbb{F}^*.\ \langle \sigma, \gamma\eta \rangle \in Lex\}$ to be the set of suffixes of lexical feature sequences. The states of our automaton are pairs $\langle \eta, f \rangle$, where $\eta \in \mathsf{suf}(Lex)$ and $f : \{\text{-}y : y \in \mathbf{lic}\} \to \mathsf{suf}(Lex)$ is a partial function.[3] The functions $(\delta_f)_{f \in F}$ are defined as per the following.

- For each lexical item $\ell = \langle \sigma, \eta \rangle \in Lex$, $\delta_\ell = \langle \eta, \emptyset \rangle$.
- Given state $s = \langle \text{+}y\eta, f \rangle$ with $f(\text{-}y) = \text{-}y\gamma$, $\delta_{\mathbf{v}}$ is defined at s iff either
  1. $\gamma = \epsilon$, in which case $\delta_{\mathbf{v}}(s) = \langle \eta, f_{/\text{-}y} \rangle$
  2. $\gamma = \text{-}z\eta$ and f is undefined at $\text{-}z$, in which case $\delta_{\mathbf{v}}(s) = \langle \eta, f_{/\text{-}y} \oplus \{\gamma\} \rangle$
- Given states $s = \langle \text{=}x\eta, f \rangle$ and $s' = \langle x\eta', f' \rangle$ with $f \oplus f'$ defined, $\delta_{\mathbf{r}}$ is defined at the pair s, s' iff either
  1. $\eta' = \epsilon$, in which case $\delta_{\mathbf{r}}(s, s') = \langle \eta, f \oplus f' \rangle$
  2. $\eta' = \text{-}y\gamma$ and $f \oplus f'$ is undefined at $\text{-}y$, in which case $\delta_{\mathbf{r}}(s, s') = \langle \eta, (f \oplus f') \oplus \{\eta'\} \rangle$

A derivation $d \in D(G)$ is *saturated* iff there is some state $\langle x\eta, f \rangle$ beginning with a selectee feature such that $A_G(d) = \langle x\eta, f \rangle$. The convergent derivations at selectee feature x are defined to be those in $D_x(G) := A_G^{-1}(\langle x, \emptyset \rangle) = \{t \in T(U) : A_G(t) = \langle x, \emptyset \rangle\}$. It was shown in [5] that for each x, there is a finite copying top-down tree transducer with regular look-ahead which maps $D_x(G)$ to $L_x(G)$. This is a simple consequence of the facts that the inference rules above can be put into the format of a multiple regular tree grammar, that we can restrict our attention to a finite set of 'categories' (as given by $\mathsf{suf}(Lex)^{|\mathbf{lic}|+1}$ [6]), and that a multiple regular tree grammar can be presented in terms of a finite copying top-down tree transducer with regular look-ahead acting on a regular set [7].

The following facts about convergent derivations will prove useful in the next section.

Proposition 2. *For any G, and any selectee feature x, the following are true:*

1. *if $t \in D_x(G)$ then every leaf of t is of the form $\langle \sigma\ ,\ \eta c \gamma \rangle$ for $\eta \in \{\text{=}x, \text{+}y : x \in \textbf{\textit{sel}} \,\&\, y \in \textbf{\textit{lic}}\}^*$ and $\gamma \in \{\text{-}y : y \in \textbf{\textit{lic}}\}^*$*
2. *if $t \in D_x(G)$ then every leaf ℓ of t occurs in the n-ary context $\textsc{Con}_\ell(\eta)$, where n is the number of selector features ℓ contains, and η is the initial sequence of selector and licensor features of ℓ, and where $\textsc{Con}_\ell(\cdot)$ is defined as follows:*

   (a) $\textsc{Con}_\ell(\epsilon) = \ell$
   (b) $\textsc{Con}_\ell(\eta\ \text{+}y) = \boldsymbol{v}(\textsc{Con}_\ell(\eta))$
   (c) $\textsc{Con}_\ell(\eta\ \text{=}x) = \boldsymbol{r}(\textsc{Con}_\ell(\eta), x_i)$, *where η contains i selector features*

[3] The SMC condition on the domains of the movement operations allows us to disregard expressions with more than one component beginning with the same feature.

Proof. 1 is proven by inspection of the definition of the transition functions δ_f for the automaton A_G; a state $\langle \texttt{+}z\eta, f\rangle$ where $f(\texttt{-}z) = x\gamma$, with $x \notin \{\texttt{-}y : y \in \mathbf{lic}\}$ is not in the domain of any transition function, nor is a state $\langle \texttt{-}z\eta, f\rangle$.

2 is proven by the observations that $A_G(\mathbf{r}(\langle \eta, f\rangle, \langle \gamma, g\rangle))$ is defined only if η begins with some selector feature and γ with some selectee feature, and that $A_G(\mathbf{v}(\langle \eta, f\rangle))$ is defined only if η begins with some licensor feature. □

Proposition 2 justifies us in restricting our attention to just those lexica which contain lexical items of the form $\langle \sigma\ ,\ \eta c\gamma\rangle$, where η is a string of selector and licensor features, and γ a string of licensee features. In the following, for ℓ as above, we write $\textsc{Con}_\ell$ as an abbreviation for $\textsc{Con}_\ell(\eta)$.

The equivalent of part two of proposition 2 for surface trees can be proven.

Proposition 3. *Let G and selectee feature x be arbitrary. Then for every $t \in L_x(G)$, each non-trace leaf s in t occurs in a context $\textsc{TCon}_s^{|\eta|}$, for some lexical item $\langle s\ ,\ \eta c\gamma\rangle \in Lex$.*

Proof. Let G, x, t, and s be as in the statement of the proposition, and let $d \in D_x(G)$ be a derivation of t. As the operations of a minimalist grammar are linear, non-deleting, and introduce only $\texttt{t}$ and $\bullet$ syncategorimatically, we can identify for each leaf of t the lexical item in d from which it came. Let s be derived from an occurrence of the lexical item $\ell = \langle s\ ,\ \eta c\gamma\rangle$ in d. This item occurs by proposition 2 in the context $\textsc{Con}_\ell$, and so there are $d_1, \ldots, d_{|\eta|}$ such that $d' = \textsc{Con}_\ell[d_1, \ldots, d_{|\eta|}]$ is the occurrence of (the projection of ℓ) in d. By inspection of the operations, $\mathsf{ev}(d') = \langle r, c\gamma\rangle, \phi_1, \ldots, \phi_k$ is seen to be such that r instantiates the context $\textsc{TCon}_s^{|\eta|}$. As no further operations can modify r internally, r occurs in t. □

Proposition 3 implies that once we know the identity and numerosity of the lexical items $\ell_1, \ldots, \ell_n$ in a complete derivation, the derived tree is gotten by putting their respective $\textsc{TCon}_\ell$ together, along with a certain number of syncategorematic traces (equal to the total number of licensee features across the ℓ_i).

2 Languages

The basic idea of the construction in both proofs is that each attractor feature of a minimalist lexical item can require that the element whose feature it checks have a certain property. Furthermore, each attractee feature can indicate that its lexical item has a certain property. This is done by annotating a feature with a representation of a particular property. (Then $\texttt{=}x^p$ indicates that it is looking for an x with property p, and $\texttt{-}y^p$ indicates that it is a $\texttt{-}y$ which has property p.) In the cases we will be interested in, P will be the state set of a finite state automaton.

Given a finite set P of properties, we define a function $\langle\cdot\rangle_P$ which maps a feature type $f \in \mathbf{sel} \cup \mathbf{lic}$ to the set $\{f^p : p \in P\}$ of its P-variants. Abusing

notation, we write $\langle\cdot\rangle_P$ for its extension to features ($\langle *x\rangle_P := \{*x^p : p \in P\}$), to sequences ($\langle aw\rangle_P := \langle a\rangle_P \cdot \langle w\rangle_P$), and to sets ($\langle X\rangle_P := \bigcup_{x\in X}\langle x\rangle_P$). The set of P-variants of a lexical item ℓ is the set of lexical items whose feature sequences are P-variants of the feature sequence of ℓ ($\langle\langle\sigma\ ,\ \gamma\rangle\rangle_P := \{\langle\sigma\ ,\ \eta\rangle : \eta \in \langle\gamma\rangle_P\}$), and the P-variant of a grammar $G = \langle\Sigma, \mathbf{sel}, \mathbf{lic}, Lex\rangle$ is the grammar whose feature types, and lexicon are P-variants of G's ($\langle G\rangle_P := \langle\Sigma, \langle\mathbf{sel}\rangle_P, \langle\mathbf{lic}\rangle_P, \langle Lex\rangle_P\rangle$). The operation $\langle\cdot\rangle_P$ can be extended in the obvious way to derivation trees, mapping leaves (lexical items) to their sets of P-variants, and acting pointwise on non-leaf nodes ($\langle f(t_1, \ldots, t_n)\rangle_P := \{f(s_1, \ldots, s_n) : s_i \in \langle t_i\rangle_P \text{ for } 1 \leq i \leq n\}$).

Proposition 4. *For any G, any P, $x \in \mathbb{F}$ and $q \in P$, if $d \in D_{x^q}(\langle G\rangle_P)$, then $\langle d\rangle_P^{-1} \in D_x(G)$. If $|P| = 1$, then $d \in D_x(G)$ implies $\langle d\rangle_P \in D_{x^q}(\langle G\rangle_P)$.*

Proof. Let A_G and $A_{\langle G\rangle_P}$ be the automata over derivations of G and $\langle G\rangle_P$ described in the previous section. Viewed as a function from derivations to states, we show that $A_G = \langle\cdot\rangle_P^{-1} \circ A_{\langle G\rangle_P} \circ \langle\cdot\rangle_P$. First, we have that $\mathsf{suf}(Lex) = \langle\mathsf{suf}(\langle Lex\rangle_P)\rangle_P^{-1}$, and thus that the states of A_G and $A_{\langle G\rangle_P}$ are in a one-many correspondence (via $\langle\cdot\rangle_P$). By the definitions of $\delta_{\mathbf{v}}$ and $\delta_{\mathbf{r}}$, the transitions at non-leaf nodes of both machines partially commute with $\langle\cdot\rangle_P$, in the sense that if $\delta(s_1, \ldots, s_n)$ is defined (in $A_{\langle G\rangle_P}$), then $\delta(\langle s_1\rangle_P^{-1}, \ldots, \langle s_n\rangle_P^{-1})$ is defined (in A_G) and is equal to $\langle\delta(s_1, \ldots, s_n)\rangle_P^{-1}$. Finally, for every $\ell' \in \langle Lex\rangle_P$, $\langle\delta_{\ell'}\rangle_P^{-1} = \delta_{\langle\ell'\rangle_P^{-1}}$.

If $|P| = 1$, then the states are in a bijective correspondence, the transitions of both machines at non-leaf nodes commute with $\langle\cdot\rangle_P$, and $\langle\delta_\ell\rangle_P = \delta_{\langle\ell\rangle_P}$. □

Proposition 5. *For any G, any P, and any $x \in \mathbb{F}$, it holds for all $q \in P$ that $L_x(G) = L_{x^q}(\langle G\rangle_P)$.*

Proof. In the forward direction, let q and x be arbitrary, and let $t \in L_x(G)$, with derivation $d \in D_x(G)$. By proposition 4, $\langle d\rangle_{\{q\}} \in D_{x^q}(\langle G\rangle_P)$. As features can be renamed arbitrarily (as long as features that are supposed to match still do) without affecting the identity of the tree derived, $\langle d\rangle_{\{q\}}$ evaluates to $t \in L_{x^q}(\langle G\rangle_P)$.

In the reverse direction, let q and x be arbitrary and $t \in L_{x^q}(\langle G\rangle_P)$ with derivation $d \in D_{x^q}(\langle G\rangle_P)$. By proposition 4, $\langle d\rangle_P^{-1} \in D_x(G)$. By the reasoning just above, evaluating $\langle d\rangle_P^{-1}$ yields $t \in L_x(G)$. □

While propositions 4 and 5 show that marking up a lexicon does not change the derived trees, it *does* have the effect of multiplying derivations for any given derived object. By selectively removing lexical items from $\langle Lex\rangle_P$, we can *ipso facto* impose meanings on the property symbols.

Example 6. To require that every derivation include lexical item $\ell \in Lex$, we take $P = \{0, 1\}$, and consider the largest subset X of $\langle Lex\rangle_P$ which satisfies the following conditions:

1. if $\ell' \in X$ and $\ell' \in \langle\ell\rangle_P$ then the property associated with its category feature z is 1

2. if $\ell' \in X$ and $\ell' \notin \langle \ell \rangle_P$, then the property associated with its category feature z is the maximum of the properties associated with its selector features $=x_i$.

Then every well-formed derivation tree headed by a lexical item with a category feature with property 1 contains some ℓ' with $\ell' \in \langle \ell \rangle_P$, and conversely, those headed by lexical items with some category feature z^0 do not contain an ℓ.

This example can actually be viewed as a special case of a more general construction, which restricts our attention to just those derivations that satisfy some property defined by a regular tree automaton.

Proposition 7. *Let G be a minimalist grammar, x a feature, and L a regular subset of $T(U)$. There is a minimalist grammar G_L, a feature y, and a set P such that $\langle D_y(G_L) \rangle_P^{-1} = D_x(G) \cap L$.*

Proof. Let G, x, and L be arbitrary as in the statement of the proposition. Let $A = \langle Q, (\delta)_{f \in U} \rangle$ be a deterministic bottom-up tree automaton such that for some $r \in Q$, $A^{-1}(r) = L$. The feature y in the statement of the proposition will be identified with x^r. We will be interested in a particular subset $Lex_A \subseteq \langle Lex \rangle_Q$, in which the annotations on the features of lexical items reflect the behaviour of A on the well-formed derivation trees they occur in.

First, we define a variant of the Con function from proposition 2, a mapping $V : \langle Lex \rangle_Q \to T(U \cup Q)$ from annotated lexical items to terms over U and Q, in the following manner. Writing V_ℓ for $V(\ell)$, we define $V_\ell := \textsc{Con}_{\langle \ell \rangle_Q^{-1}}[q_1, \ldots, q_n]$, where $\ell = \langle \sigma \ , \ {=}x_1^{q_1} \eta_1 \cdots {=}x_n^{q_n} \eta_n c \gamma \rangle$, and for $1 \leq i \leq n$ $\eta_i \in \{{+}y : y \in \langle \mathbf{lic} \rangle_Q\}^*$. Intuitively, V_ℓ calls $\textsc{Con}_\ell$, and fills in the variables where an argument would be merged with the state that the automaton must be in when reading the subtree occuring in that position.

Now, let Lex_A be the smallest subset of $\langle Lex \rangle_Q$, such that it contains a lexical item $\ell = \langle \sigma \ , \ {*}x_1^{a_1} \cdots {*}x_n^{a_n} z^a {-}y_1^{b_1} \cdots {-}y_m^{b_m} \rangle \in Lex_Q$ iff $a = A(V_\ell)$. Intuitively, we are removing all the lexical items from $\langle Lex \rangle_Q$ whose feature annotations on selector and category features do not accurately reflect the behaviour of A on the contexts in which they occur. We prove that for $d \in D(G_L)$ a saturated derivation tree with first feature z^q, $A(\langle d \rangle_Q^{-1}) = q$ by induction on the heights of saturated derivation trees. For the base case, let $\ell = \langle \sigma \ , \ z {-}y_1 \cdots {-}y_m \rangle$, and let $A(\ell) = q$. Then Lex_A contains an item $\ell = \langle \sigma \ , \ z^a {-}y_1^{b_1} \cdots {-}y_m^{b_m} \rangle \in Lex_Q$ iff $a = A(V_\ell) = A(\langle \ell \rangle_Q^{-1}) = q$. Now assume that the proposition holds of saturated derivation trees up to height $n-1$, and let saturated $d \in D(G_L)$ have height n. Then $d = \textsc{Con}_{\langle \ell \rangle_Q^{-1}}[d_1, \ldots, d_k]$ for some $\ell = \langle \sigma \ , \ {*}x_1^{r_1} \cdots {*}x_j^{r_j} z^q \gamma \rangle \in Lex_A$, and saturated derivation trees $d_1, \ldots, d_k$. By the inductive hypothesis, $A(\langle d_i \rangle_Q^{-1}) = q_i$, where the first feature of d_i is $z_i^{q_i}$, for $1 \leq i \leq k$. Then $A(\textsc{Con}_{\langle \ell \rangle_Q^{-1}}[d_1, \ldots, d_k]) = A(\textsc{Con}_{\langle \ell \rangle_Q^{-1}}[q_1, \ldots, q_n]) = A(V_\ell) = q$.

Now we are in a position to prove the original proposition. Define $G_L := \langle \Sigma, \langle \mathbf{sel} \rangle_Q, \langle \mathbf{lic} \rangle_Q, Lex_A \rangle$, and let $t \in D_{x^r}(G_L)$. By proposition 4, $\langle t \rangle_Q^{-1} \in D_x(G)$. As we saw in the previous paragraph, $A(\langle t \rangle_Q^{-1}) = r$, and thus $\langle t \rangle_Q^{-1} \in L$. This establishes the forward containment ($\subseteq$). For the reverse direction ($\supseteq$), let $t \in$

$D_x(G) \cap L$, and let $t' \in \langle t \rangle_Q$ be such that every selector feature on every leaf in t is annotated with the state that A is in when it has read the corresponding argument to that feature, and every selectee feature on every leaf in t is annotated with the state that A is in when it has read the entire subtree of t which is the instantiation of the context of that leaf. Clearly, $t' \in D(G_L)$. As $t \in L$, $t' \in D_{x^r}(G_L)$, which concludes the proof. □

It is worth remarking about the constructed G_L in the proof of proposition 7 that its lexical items exhibit only certain dependencies amongst their features. In particular, if some lexical item $\ell \in Lex_A$ has as its i^{th} feature a licensee feature $-y^p$, then Lex_A contains every ℓ' like ℓ except that instead of $-y^p$, its i^{th} feature is $-y^q$, for some $q \in Q$. The same holds true of $+y$ features, but *not* in general for $=x$ and x features.

From a high-level perspective, proposition 7 shows that we may 'push' down into the features of the leaves the state-annotations on non-terminals in a derivation tree obtained from Thatcher's [2] construction of a context-free grammar from a bottom-up finite state tree automaton.

2.1 Closure under Intersection with Regular Sets

In the previous section we noted that when incorporating regular restrictions on derivations, we only needed to enforce dependencies among selection features. Intuitively, all of the information that an automaton traversing a derivation tree might care about is present already when two expressions are merged together, and so to 'keep track' of this information we only need to encode it on the features relevant for merger.

By placing constraints on the dependencies between licensing features, we allow information to be communicated about aspects of the derived objects, such as whether a particular movement is of an expression that will remain in its current position, or of one which will continue on (and leave behind a trace).

Proposition 8. *Let G be a minimalist grammar, and $L \subseteq T(S)$ a regular tree language. Then for any x there is a minimalist grammar G^L and a feature y such that $L_y(G^L) = L_x(G) \cap L$.*

Proof. Let G and L and x be arbitrary, and let $A = \langle Q, (\delta_s)_{s \in S} \rangle$ be a bottom-up finite tree automaton such that for some $r \in Q$, $A^{-1}(r) = L$. As before, we will identify the y in the statement of the theorem with x^r. Define the state $t = A(\mathtt{t})$ to be the state A is in when it has scanned a trace. As before, we will define a subset $Lex^A \subseteq \langle Lex \rangle_Q$ to be our lexicon. First we define a 'surface counterpart' of the V mapping from proposition 7, a function $R : T(S) \times Q^* \to T(S \cup Q)$ from derived tree – state sequence pairs to terms over S and Q, in the following manner. Writing $R_t(w)$ for $R(t, w)$, we define $R_t(w) := \textsc{Tcon}_t^{|w|}[w_1, \ldots, w_{|w|}]$, where $\textsc{Tcon}_t^n$ is as in the statement of proposition 1.

Now we define $Lex^A \subseteq \langle Lex \rangle_Q$ to be the smallest set such that it contains a lexical item $\langle \sigma \, , \, {*}x_1^{a_1} \cdots {*}x_n^{a_n} z^t {-}y_1^t \cdots {-}y_m^a \rangle \in \langle \ell \rangle_Q$ iff $a \in A(R_\sigma(a_1 \cdots a_n))$. The selector and licensor feature annotations are intended to indicate the state the

automaton is in when it scans the surface item in that position, and the selectee and licensee feature annotations provide information about what surface term appears in each of the positions associated with these features. Note that all but the last selectee and licensee features are annotated with a 'trace' – this is because an expression leaves behind a trace in all but its final position.

Now let $X \subseteq L(G^L)$ be the subset of $L(G^L)$ containing all and only expressions $\phi_0, \ldots, \phi_k$ which meet the conditions that

1. for $1 \leq i \leq k$, the last feature in each ϕ_i is annotated with the state that A is in when it scans the tree component of ϕ_i, all other features of each ϕ_i are annotated with t (the state A is in when it scans a trace)
2. $\phi_0 = \langle s, *x_1^{a_1} \cdots *x_j^{a_j} z^{b_0} -y_1^{b_1} \cdots -y_m^{b_m} \rangle$ is such that $A(R_s(a_1 \cdots a_j)) = b_m$ and $b_0, \ldots, b_{m-1} = t$

We show that X contains Lex^A and is closed under $merge_i$ and $move_i$, for $1 \leq i \leq 2$, and is therefore equal to $L(G^L)$.

- Let $\ell = \langle \sigma \ , *x_1^{a_1} \cdots *x_j^{a_j} z^{b_0} -y_1^{b_1} \cdots -y_m^{b_m} \rangle \in Lex^A$. Then by definition, $b_0 \ldots, b_{m-1} = t$, and $b_m = A(R_\sigma(a_1 \cdots a_j))$.
- Let $\phi_0, \ldots, \phi_k \in X$ be in the domain of the $move_1$ operation. Then $\phi_0 = \langle s, +x_1^{a_1} *x_2^{a_2} \cdots *x_j^{a_j} z^{b_0} -y_1^{b_1} \cdots -y_m^{b_m} \rangle$, and for some i, $\phi_i = \langle r, -x_1^{a_1} \rangle$, where by hypothesis $a_1 = A(r)$, and $b_m = A(R_s(a_1 \cdots a_j))$. Then $move_1(\phi_0, \ldots, \phi_k)$ is the expression $\phi'_0, \phi_1, \ldots, \phi_{i-1}, \phi_{i+1}, \ldots, \phi_k$, where ϕ'_0 has as its first component the term $\bullet(r, s)$, and second component the tail of the feature sequence of ϕ_0. By the definition of R, we see that $A(R_{\bullet(r,s)}(a_2 \cdots a_j)) = A(R_s(A(r)\ a_2 \cdots a_j)) = A(R_s(a_1\ a_2 \cdots a_j)) = b_m$.
- Let $\phi_0, \ldots, \phi_k \in X$ be in the domain of the $move_2$ operation. Then $\phi_0 = \langle s, +x_1^{a_1} *x_2^{a_2} \cdots *x_j^{a_j} z^{b_0} -y_1^{b_1} \cdots -y_m^{b_m} \rangle$, and $\phi_i = \langle r, -x_1^{a_1} -z_1^{c_1} \cdots -z_h^{c_h} \rangle$ for some i, where by hypothesis $a_1 = t$, $c_1, \ldots, c_{h-1} = t$, $c_h = A(r)$, and $b_m = A(R_s(a_1 \cdots a_j))$. Then the result of applying $move_2$ to $\phi_0, \ldots, \phi_k$ is the expression $\phi'_0, \phi_1, \ldots, \phi'_i, \ldots, \phi_k$, where ϕ'_0 has as its first component the term $\bullet(\mathtt{t}, s)$, and second component the tail of the feature sequence of ϕ_0, and $\phi'_i = \langle r, -z_1^{c_1} \cdots -z_h^{c_h} \rangle$. By the definition of R, we see that $A(R_{\bullet(\mathtt{t},s)}(a_2 \cdots a_j)) = A(R_s(t\ a_2 \cdots a_j)) = A(R_s(a_1\ a_2 \cdots a_j)) = b_m$.
- the proof for $merge_1$ is similar to that of $move_1$ and is omitted.
- the proof for $merge_2$ is similar to that of $move_2$ and is omitted.

Now let $G^L := \langle \Sigma, \langle \mathbf{sel} \rangle_Q, \langle \mathbf{lic} \rangle_Q, Lex^A \rangle$. In the forward direction, let $t \in L_{x^r}(G^L)$. By proposition 5 we have that $t \in L_x(G)$. As $t \in L_{x^r}(G^L)$, it holds that $\langle t, x^r \rangle \in L(G^L)$, which has properties 1 and 2 above, whence $t \in A^{-1}(r) = L$. In the reverse direction, let $t \in L_x(G) \cap L$ with derivation $d \in D_x(G)$. For each non-trace leaf s in t identify the lexical item ℓ_s in d from which it came (as per proposition 3), and associate s with its annotated surface context $R_s(A(t_1), \ldots, A(t_n))$, for $t_1, \ldots, t_n \in T(S)$ such that $\textsc{TCon}_s^n[t_1, \ldots, t_n] \in \textsc{LexProj}(t)$. Define $\ell'_s \in \langle Lex \rangle_Q$ to be like ℓ_s but where the selector and licensor features are annotated from left to right with $A(t_1)$ throught $A(t_n)$, the last feature with $A(R_s(A(t_1), \ldots, A(t_n)))$, and all remaining feautres with t. Clearly,

$\ell'_s \in Lex^A$, and the d' obtained by replacing all leaves in d in this manner is a derivation in $D(G^L)$. That it is a complete derivation of category x^r follows from its construction, whence $\mathsf{ev}(d') = \langle t, x^r\rangle$, and $t \in L_{x^r}(G^L)$, as desired. □

As an application of proposition 8, we show the known result that the string languages of minimalist grammars are closed under intersection with regular string languages. We do this by generating a finite state tree automaton recognizing all trees with yields in the given regular string language, which can then by 8 be 'intersected' with an arbitrary minimalist grammar.

Example 9. Let $M = \langle Q^M, q_0^M, \zeta, q_f^M\rangle$ be a regular *string* automaton, with state set Q^M, initial and final states q_0^M and q_f^M, respectively, and transition function $\zeta : Q^M \times \Sigma \to Q^M$. The set of trees over $T(S)$ whose yield is in $L(M)$ is given by the regular *nondeterministic* tree automaton $A_M = \langle Q^A, Q_f^A, (\delta_f)_{f\in\Sigma}\rangle$ (where Q_f^A is the set of final states) defined as follows:

1. $Q^A = \left[Q^M \to Q^M\right]$
2. $Q_f^A = \{q \in Q^A : q(q_0^M) = q_f^M\}$
3. For $\mathtt{t}^{(0)}$, $\delta_{\mathtt{t}}(q) = q$
4. For $\sigma \in \Sigma_0$, $\delta_\sigma(q) = \lambda x.\zeta(q(x), \sigma)$
5. For $\bullet^{(2)}$, $\delta_\bullet(q_1, q_2) = q_1 \circ q_2$

Define $L := \bigcup_{q\in Q_f^A} A_M^{-1}(q)$. Then for G a minimalist grammar and x a selectee feature, there is a feature y such that, $S_y(G^L)) = S_x(G) \cap L(M)$.

Example 9 can be viewed as a particular instance of a much more general fact.[4]

Proposition 10. ***(Courcelle [8])***

1. *The inverse image of a (MSO) definable set of structures under a (MSO) definable transduction is (MSO) definable*
2. *The composition of two (MSO) definable transductions is (MSO) definable.*

In particular, if $\phi : T(S) \to \Delta^*$ is an MSO definable spell-out mapping, associating trees with strings over Δ, and $L \subseteq \Delta^*$ is a regular string language of 'what might have been just said', then $\phi^{-1}[L] = \{t \in T(S) : \phi(t) \in L\}$ is by proposition 10 a regular tree language, and thus for any minimalist grammar G, we can construct another minimalist grammar $G^{\phi^{-1}[L]}$ which derives exactly those trees in $\phi^{-1}[L]$ which are derived by G. As an example, mirror-style head

[4] In fact, proposition 8 follows from propositions 7 and 10 as a corollary, as was pointed out by an anonymous reviewer. Let L be a regular tree language, and ϕ_L a MSO formula defining exactly L. Let G be an arbitrary minimalist grammar, and let Δ be the MSO definable transduction taking exactly $D_x(G)$ to $L_x(G)$. Then by proposition 10, $S = \Delta^{-1}[L]$ is a regular (i.e. MSO-definable) subset of $D_x(G)$ which contains a derivation tree d iff d is the derivation of some tree $t \in L$, and thus by proposition 7, G_S exists, and is such that for some y, $L_y(G_S) = L_x(G) \cap L$.

movement, as implemented by [9], can be treated in terms of a non-standard (but MSO definable) spell-out mapping from minimalist trees into strings.

More generally, given a minimalist grammar G, any sequence $\phi_1, \ldots, \phi_n$ of MSO-definable mappings $\phi_i : A_i \to A_{i+1}$, where $A_0 = L(G)$ and $A_{n+1} = \Delta^*$ can be 'undone', in the sense that for any MSO-definable subset $L \subseteq \Delta^*$, $(\phi_n \circ \cdots \circ \phi_1)^{-1}(L)$ is a regular subset of $L(G)$, and there is therefore a minimalist grammar $G^{(\phi_n \circ \cdots \circ \phi_1)^{-1}(L)}$ which generates exactly this regular subset.

Not only does this yield a guarantee of mild-context sensitivity, as the string languages generable from finitely many MSO transductions applied successively to a regular tree language is exactly the set of multiple context free languages (see [10]), this also gives a sort of 'parsing as intersection' result [11] for any extension of the basic framework of minimalist grammars by MSO means.

3 Applications

In this section I discuss three applications of propositions 7 and 8 above.[5] I begin by discussing conditions under which semantic type mismatches (and the consequent deviance of an utterance) can be incorporated into the grammar. Next I introduce Koopman's 'complexity filters', and show that they do not increase the tree generating power of minimalist grammars. Finally, I offer a formalization of distributed morphology, and show that it is weakly equivalent to minimalist grammars.

3.1 Semantics

Semantics is often couched in a typed system. In this kind of framework, a structure might be unable to be assigned a meaning due to a type mismatch, and thus a semantic interpretation function might be partial. In such a case the meaningful structures generated by a grammar might be a proper subset of the structures it generates. Does the possibility of semantic type mismatch render the problem of finding meaningful parses for strings computationally more difficult? This question is especially salient in the context of the extended standard theory [13] and its decendents, where semantic interpretation is of the derived structure, and not of the structure of the derivation.

Propositions 7 and 8 tell us that, if we can represent the property of being semantically well-formed in terms of a bottom-up tree automaton, then we can transform a minimalist grammar generating possibly semantically uninterpretable surface trees into one which directly generates only the subset of the first which are semantically interpretable. This will hold true whether we interpret surface structures (as do [4]) or derivation trees (as is done in [14]).

While it is not in general true that given a set of typed constants, there are only finitely many types derivable from them, the textbook of [4] (which is the *locus classicus* for semantics in the chomskyian vein) eschews more than application

[5] Another application which I do not discuss is to work on economy constraints in minimalist grammars [12].

$(\alpha\beta \oplus \alpha \Rightarrow \beta)$ and (sometimes) 'predicate modification' $(\alpha t \oplus \alpha t \Rightarrow \alpha t)$, closing any finite set of types under which results in a finite set of types.

3.2 Complexity Filters

[15] analyze verbal complexes in Dutch, German and Hungarian in a syntactic framework much like the one presented here. They propose to treat these languages as similar for the purposes of this construction, with the primary locus of cross-linguistic variation being that some configurations are filtered out in certain languages. In general, the approach of [15], following [16], is to postulate a great number of silent terminal nodes, and that syntactic terminals get moved about repeatedly in a great number of ways. While this sort of approach allows for the 'discovery' of similar structures underlying very different surface strings (and in different languages), it tends to overgenerate. 'Complexity filters', as proposed by [15] (and expanded upon in [17,3]), are a way of reigning in this overgeneration. Koopman proposes that each lexical item ℓ may impose requirements on the *size* of the surface expressions $t_1, \ldots, t_n$ which may occur in $\text{TC}\textsc{on}_\ell[t_1, \ldots, t_n]$. The size of a tree t_i is defined in terms of the length of and labels along the path from the most deeply embedded phonetically overt terminal in t_i to its root. As a concrete example, [3] proposes that the Germanic prenominal genitive is structurally identical to the English Saxon genitive. The well-known restrictions on the prenominal genitive in German (that, for example, coordinated structures cannot appear) are not due to a different syntax from the superficially similar English construction, but to a lexical idiosyncracy of the German *'s* counterpart, which prohibits DPs with more than a small amount of pronounced structure (predominantly proper names and pronouns) to appear in its specifier. This is as shown in figure 1 (from [17]).

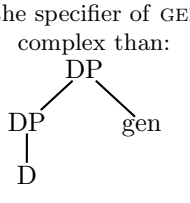


Fig. 1. Complexity filter associated with the German prenominal genitive

Importantly, all of the postulated complexity filters can be stated in terms of MSO logic over trees;[6] expressing that the length from the root to the most deeply embedded overt node is at most n is a simple universally quantified sentence to the effect that all nodes which are overt have depth less than or

[6] This is somewhat overkill, when compared to the attested complexity filters. The important thing is that the linguist appealing to complexity filters may propose *any* recognizable constraint, secure in the knowledge that he is not increasing the strong generative capacity of minimalist grammars.

equal to n. Conversely, *requiring* that a tree have a pronounced node deeper than some depth n is the negation of the previous statement.

To see that koopmanian complexity filters are admissible in the minimalist grammar formalism (in the sense that any MG with complexity filters may be reformulated as one with the same strong generative capacity without such), it is important to note first that these filters are not quite filters on the surface tree. As is indicated in figure 1, a complexity filter applies a regular filter to the surface trees in a particular geometric relationship to *a particular lexical item*, not just any head with a particular phonological form.

Toward a formalization, let $L_1, \ldots, L_n$ be the n complexity filters used in a minimalist grammar G. We will take each L_i $1 \leq i \leq n$ to define a regular tree language over $T(S)$, and will allow each selector and licensor feature of every lexical item to be associated with one of these filters. During a derivation, we check that the filters (if any) associated with a particular feature are respected. Below are presented two of the additional generating functions needed to take complexity filters associated with features into account.

$$\frac{\langle s_1, {=}c^L\gamma\rangle, \phi_1, \ldots, \phi_m \qquad \langle s_2, c\rangle, \psi_1, \ldots, \psi_n \qquad s_2 \in L}{\langle \bullet(s_2, s_1), \gamma\rangle, \phi_1, \ldots, \phi_m, \psi_1, \ldots, \psi_n} \; merge_1^{Filter}$$

$$\frac{\langle s_1, {+}c^L\gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \langle s_2, {-}c\delta\rangle, \phi_{i+1}, \ldots, \phi_m \qquad \mathtt{t} \in L}{\langle \bullet(\mathtt{t}, s_1), \gamma\rangle, \phi_1, \ldots, \phi_{i-1}, \langle s_2, \delta\rangle, \phi_{i+1}, \ldots, \phi_m} \; move_2^{Filter}$$

To recast this in 'standard' minimalist grammars, we will take a new lexicon $Lex^{Filter} \subseteq \langle Lex\rangle_Q$, where $Q := Q_1 \times \cdots \times Q_n \cup \{\mathbf{1}\}$ is the state set of the disjoint union of the product automaton $A_1 \otimes \cdots \otimes A_n$ of the automata A_i such that $A_i^{-1}(r_i) = L_i$, $1 \leq i \leq n$, and the 'unit' automaton $A_\mathbf{1}$ with one state recognizing all of $T(S)$. We require that each A_i is *total*, in that every δ_f^i is defined on every input. We want the feature annotations to respect the transitions of A, and so we are interested in the smallest subset X of $Lex^A \subseteq \langle Lex\rangle_Q$ meeting the following conditions, which we will call Lex^{Filter}:

1. if $\ell \in Lex$, then X contains every $\ell' \in Lex^A$ such that for n the number of selector and licensor features of ℓ, and for $1 \leq i \leq n$,
   - if the i^{th} feature of ℓ is associated with complexity filter L_j, then the i^{th} feature of ℓ' is annotated with some state $q \in Q_1 \times \cdots \times Q_{j-1} \times \{r_j\} \times Q_{j+1} \times \cdots \times Q_k$
   - if the i^{th} feature of ℓ is not associated with any complexity filter, then the i^{th} feature of ℓ' is annotated with some state $q \in Q - \{\mathbf{1}\}$
2. for each $\ell' \in X$ with no licensee features, X contains the lexical item ℓ'', which is identical to ℓ' except that its selectee feature is annotated with the state $\mathbf{1}$.

Crucially, the state annotation $\mathbf{1}$ does not appear on any selector or licensor feature. Note also that because we are drawing lexical items from Lex^A, the last feature of a lexical item is annotated with a state which accurately reflects its structure.

Proposition 11. *For G a minimalist grammar with complexity filters, and G^{Filter} the minimalist grammar with lexicon Lex^{Filter}, for every selectee feature x it holds that $L_x(G) = L_{x^1}(G^{Filter})$.*

The proof is similar to that of proposition 8 and is suppressed.

3.3 Distributed Morphology

Distributed Morphology is a theory of the relation between a surface tree and its yield developed in [18]. There are two particularly salient aspects of this theory that are relevant here. First, the surface tree may be deformed prior to computing its yield. These deformations take the following form (from [18]):

morphological merger: a head X may be lowered to the head Y of its complement, forming the complex $\bullet(X, Y)$
fusion: a node $\bullet(X, Y)$, where X, Y are heads, may be replaced by a new node Z
fission: a head Z may be replaced by the node $\bullet(X, Y)$
impoverishment: A head X may be replaced by a head Y
morpheme addition: A term $t \in L$ may be replaced by $\bullet(t, X)$

Clearly, each of the above operations is a regular tree transduction, for any particular selection of heads X, Y, Z, and regular tree language L.

The second salient aspect of the theory is that 'lexical insertion takes place *during* spell-out'. This means (essentially) that the lexicon is a function from Σ to feature sequences (i.e. each lexical item is associated with a unique name); we might as well identify Σ with an initial subset of the natural numbers.[7] The surface tree is then a hierarchical arrangement of natural numbers. Each natural number can be realized as one of a finite number of strings over Δ^*, depending on which of a finite number of regular contexts it occurs in. Thus, each lexical item is associated with an MSO-definable transduction that 'realizes' it.

A distributed morphology grammar consists of a minimalist grammar G, along with a finite sequence of transductions $\phi_1, \ldots, \phi_n$, where ϕ_i is either a lexical insertion transduction, or a transduction of one of the five types given above. A string $s \in \Delta^*$ belongs to $L_x(G, \boldsymbol{\phi})$ (the language of a grammar $G, \phi_1, \ldots, \phi_n$ at category x) iff s is the yield of $\phi_n \circ \cdots \circ \phi_1(t)$, for some $t \in L_x(G)$. As $L_x(G)$ is the image of a regular set ($D_x(G)$) under a (direction preserving) MSO-definable transduction [19], we have that $L_x(G, \boldsymbol{\phi})$ is a multiple context-free language for any G, $\boldsymbol{\phi}$, and x [20]. Moreover, as by proposition 10 we have that for any $s \in \Delta^*$, $(\phi_n \circ \cdots \circ \phi_1)^{-1}(s)$ is a regular tree language, we can directly define a minimalist grammar defining exactly the set $L_x(G) \cap (\phi_n \circ \cdots \circ \phi_1)^{-1}(s)$ by proposition 8, making in principle available a parsing-as-intersection view of recognition and parsing in distributed morphology.

[7] In 'reality', one associates with each lexical item a (non-recursive) attribute value matrix, and with a subset of the lexical items (the contentful lexical items) a unique name (DOG, CAT, etc). The operation of fusion is the unification of AVMs, fission is splitting one AVM into two, and impoverishment is the removal of certain attribute-value pairs from an AVM. This is not particularly important for the present purposes.

4 Conclusion

We have shown that both derived and derivation tree languages of minimalist grammars are closed under intersection with recognizable sets of trees. This is non-trivial, as neither the derivation tree languages nor the derived tree languages constitute a well-known family of languages (the derivation tree languages are a proper subset of the recognizable sets, and the derived tree languages are a proper subset of the multiple regular tree languages).

These closure results are immediately translatable into results of direct linguistic interest. We can view the linguistic results in terms of the admissibility of a certain kind of grammar factorization. This is clearest in the case of the example of semantic types. We have shown there to be no impact on strong generative capacity whether we treat semantic type information syntactically, giving us a direct characterization of the semantically well-typed grammatical sentences, or semantically, characterizing the semantically well-typed grammatical sentences in terms of a filter. Whereas it is common consensus that we need to perform the work of semantic type-checking *somewhere* in the natural language pipeline, the complexity filters of Koopman are more controversial. By showing them to be merely notational devices, we have demonstrated that any rational debate about their adoption into minimalist grammars must revolve around something other than syntax, or semantics (perhaps learning).

Courcelle's results give us access to a weak generative capacity preseving array of extensions to a basic theory, and in conjunction with the closure under intersection results, access to an in-principle parsing method. As however the size of an automaton recognizing a tree language is not bounded by any elementary function in the size of the smallest MSO formula defining that language, and the size of the minimalist grammar recognizing the intersection language is directly related the number of states of the automaton, the size of the grammar recognizing the intersection is potentially much larger than the original grammar.[8]

References

1. Stabler, E.P.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)
2. Thatcher, J.W.: Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. Journal of Computer and System Sciences 1(4), 317–322 (1967)
3. Koopman, H.: On restricting recursion and the form of syntactic representations. In: Speas, P., Roeper, T. (eds.) Recursion. Oxford University Press, Oxford (to appear)
4. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell Publishers, Malden (1998)
5. Kobele, G.M., Retoré, C., Salvati, S.: An automata theoretic approach to minimalism. In: Rogers, J., Kepser, S. (eds. Proceedings of the Workshop Model-Theoretic Syntax at 10, ESSLLI 2007, Dublin (2007)

[8] See [21] for some relevant discussion.

6. Michaelis, J.: On Formal Properties of Minimalist Grammars. PhD thesis, Universität Potsdam (2001)
7. Raoult, J.C.: Rational tree relations. Bulletin of the Belgian Mathematical Society 4, 149–176 (1997)
8. Courcelle, B.: Monadic second-order definable graph transductions: a survey. Theoretical Computer Science 126, 53–75 (1994)
9. Kobele, G.M.: Formalizing mirror theory. Grammars 5(3), 177–221 (2002)
10. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, vol. 3, pp. 125–213. Springer, Heidelberg (1997)
11. Lang, B.: Recognition can be harder than parsing. Computational Intelligence 10(4), 486–494 (1994)
12. Graf, T.: A tree transducer model of reference-set computation. UCLA Working Papers in Linguistics 15, 1–53 (2010)
13. Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge (1965)
14. Kobele, G.M.: Generating Copies: An investigation into structural identity in language and grammar. PhD thesis, University of California, Los Angeles (2006)
15. Koopman, H., Szabolcsi, A.: Verbal Complexes. MIT Press, Cambridge (2000)
16. Kayne, R.: The Antisymmetry of Syntax. MIT Press, Cambridge (1994)
17. Koopman, H.: Derivations and complexity filters. In: Alexiadou, A., Anagnostopoulou, E., Barbiers, S., Gärtner, H.M. (eds.) Dimensions of Movement: From features to remnants. Linguistik Aktuell/Linguistics Today, vol. 48, pp. 151–188. John Benjamins, Amsterdam (2002)
18. Halle, M., Marantz, A.: Distributed morphology and the pieces of inflection. In: Hale, K., Keyser, S.J. (eds.) The View from Building 20, pp. 111–176. MIT Press, Cambridge (1993)
19. Mönnich, U.: Minimalist syntax, multiple regular tree grammars and direction preserving tree transductions. In: Rogers, J., Kepser, S. (eds.) Proceedings of the Workshop Model-Theoretic Syntax at 10, ESSLLI 2007, Dublin (2007)
20. Engelfriet, J., Heyker, L.: The string generating power of context-free hypergraph grammars. Journal of Computer and System Sciences 43, 328–360 (1991)
21. Morawietz, F., Cornell, T.: The MSO logic-automaton connection in linguistics. In: Lecomte, A., Lamarche, F., Perrier, G. (eds.) LACL 1997. LNCS (LNAI), vol. 1582, pp. 112–131. Springer, Heidelberg (1999)

Do Dialogues Have Content?

Staffan Larsson

Department of Philosophy, Linguistics and Theory of Science
Gothenburg University
Box 200, SE405 30 Gothenburg, Sweden
sl@ling.gu.se

Abstract. In this paper, the notion of "the content of a dialogue" is shown to be problematic in light of the phenomena of semantic coordination in dialogue, and the associated notion of semantic plasticity – the ability of meanings to change as a result of language use. Specifically, it appears that any notion of content in dialogue based on classical model-theoretical semantics will be insufficient for capturing semantic plasticity. An alternative formal semantics, type theory with records (TTR) is briefly introduced and is show to be better equipped to deal with semantic coordination and plasticity. However, it is also argued that any account of content in dialogue which takes semantic coordination seriously will also need to consider the problems it raises for some concepts central to traditional notions of meaning, namely inference and truth.

1 Introduction

In this paper, the notion of "the content of a dialogue" is shown to be problematic in light of the phenomena of semantic coordination in dialogue, and the associated notion of semantic plasticity – the ability of meanings to change as a result of language use. In Section 1 a specific candidate account of "the content of a dialogue", SDRT, is discussed. Section 2 introduces the notions of semantic coordination and semantic plasticity, using examples of corrective feedback in first language acquisition. In Section 3, an alternative formal semantics, type theory with records (TTR) is briefly introduced and is show, in Section 4, to be better equipped to deal with semantic coordination and plasticity. Finally, Section 5 presents some general conclusions.

2 The Content of a Dialogue

Segmented Discourse Representation Theory [2] is an approach to discourse interpretation which is originally (Asher 1993) an extension of Hans Kamp's Discourse Representation Theory (DRT). It combines the insights of dynamic semantics on anaphora with a richer theory of discourse structure.

By supplying a language for representing the logical form of discourse and of dialogue, and assigning this language a dynamic semantic interpretation, SDRT

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 145–158, 2011.

offers a candidate account of what could be regarded as "the content of a dialogue". It is, simply, the SDRS (Segmented Discourse Representation Structure) resulting from parsing and integrating all the utterances in a dialogue, insofar as it can be be given a model-theoretic interpretation, and can be used for drawing inferences. A central aspect of SDRT is that it is based on a modal and dynamic variant of classical model-theoretic semantics.

There are a number of variants of classical model-theoretic semantics. In modal logic, the truth of a proposition is defined relative to a possible world. Montagovian intensional model theory adds a so-called intensional operator, and *property theory* [5] allows different properties with identical extensions. Despite the refinements in dealing with intensional vocabulary in these extensions, they all share a common property with classical model-theoretic semantics, namely a fundamental *extensionality*. Extensions of predicates are simply postulated in terms of the interpretation function F (in a more or less complicated way). The predicates are atoms, hence unanalysed and without structure.

Note that this paper is not arguing against SDRT *per se*; as a formal account of discourse and dialogue it has many merits. For example, it has recently been extended with a logically precise account of corrections and disagreement in dialogue [20], as part of a research agenda with the aim of increasing its coverage of grounding and other dialogue phenomena. The main reason we mention SDRT is to show that the idea of using model-theoretic semantics for dialogue is not far-fetched, but something which is more or less standard practice. However, one conclusion of the arguments put forward here is that *if* a theory like SDRT were to be extended to cover certain phenomena in dialogue which are currently beyond its intended scope (namely, semantic coordination and semantic plasticity), the classical model-theoretic semantics assigned to SDRSs may need to be replaced with something else.

3 The Challenge from Semantic Plasticity

3.1 Semantic Coordination and Semantic Plasticity

Here are a few examples of *corrective feedback*:

- C: That's a nice bear.
- D: Yes, it's a nice **panda**.
- C: Panda.

- Naomi: mittens
- Father: **gloves**.
- Naomi: gloves.
- Father: when they have fingers in them they are called gloves and when the fingers are all put together they are called mittens.

- Abe: I'm trying to tip this over, can you tip it over? Can you tip it over?
- Mother: Okay I'll **turn** it over for you.

– Adam: Mommy, where my plate?
– Mother: You mean your saucer?

The first one is made up, the others are quoted from various sources in [8] and [6]. In general, corrective feedback can be regarded as offering an alternative form to the one that the speaker used.

There is evidence that dialogue interaction plays an important role in establishing a shared language, not only in first (or second) language acquisition but also in the coordination of meaning in adult language, in historical language change, and in language evolution. For example, research on alignment shows that agents negotiate domain-specific microlanguages for the purposes of discussing the particular domain at hand [9,15,21,3,16,17]. We will use the term *semantic coordination* to refer to the process of interactively coordinating the meanings of linguistic expressions. Corrective feedback is one of several mechanisms for semantic coordination in dialogue; others include explicit definition and meaning accommodation [19,12,18]. Also, computational work on emergent vocabularies and category formation [22,4] has shown that it is possible to simulate part of the dynamics of language and meaning in computers.

On this general view, two agents do not need to share exactly the same linguistic resources (grammar, lexicon etc.) in order to be able to communicate, and an agent's linguistic resources can change during the course of a dialogue when she is confronted with a (for her) innovative use. It seems that speakers of natural languages are constantly in the process of creating new language to meet the needs of novel situations in which they find themselves. This fits with a view of meaning is essentially social. Even if speakers have their own takes on the language used for talking about the current activity, when a group of speakers have sufficiently similar takes on the meaning of an expression, the expression has meaning within that group.

To illustrate how meaning can change as a result of corrective feedback, here's one of several possible stories about what happens in the "panda" example above. C originally takes the meaning of "bear" to be something like "big fuzzy bear-shaped animal", where knowing the meaning of "bear-shaped" involves some capacity to recognize a certain physical shape. C has not hear of pandas, or has no clear idea what they are. On this particular occasion, C and D are at the zoo, looking at a panda (which is the shared focus of attention). When C sees the panda, she notices that it is bear-shaped and hence assumes that it is a bear. D's utterance indicates that "panda" is a (more) correct referring phrase for this animal, and since C (correctly) takes D to be better at English than C, C decides to adapt to this way of using language. C might reason according to the Pragmatic principle of contrast [7]: "Speakers take every difference in form to mark a difference in meaning." C has observed that D's use of "panda" refers to the object in shared focused of attention, and that its colour distinguishes it from previously observed bears. Based on this, C creates a meaning of "panda" as (roughly) a big, black-and-white bear-shaped animal. Accordingly, C also modifies her take on the meaning of "bear" to mean "a big, brown bear-shaped animal".

Of course, one may disagree with the analysis proposed here on numerous points. For our current purposes, however, the details are not so important. The important observation is that semantic coordination entails *semantic plasticity*: that the meaning of an expression may change over time. The research on alignment cited above shows that such changes may in fact happen during the course of a single dialogue.

The examples of corrective feedback given above are taken from first language acquisition, where the child detects innovative (for her) uses and adapts her take on the meaning accordingly. Semantic coordination in first language acquisition is a special case of semantic coordination in general, where there is a clear asymmetry between the agents involved with respect to expertise in the language being acquired. However, a working hypothesis is that the mechanisms for semantic coordination used in these situations are similar to those which are used when competent adult language users coordinate their language.

Part of knowing the meaning of a concrete term such as "bear" is being able to recognize bears and distinguish them from other animals. In the sketchy account of semantic learning in the panda example above, it was precisely this aspect of meaning which was being modified. One way of expressing this is that a "colour" feature was added to the meaning of "bear" with the value "brown". This aspect of meaning will here be referred to as *perceptual type*. The perceptual type associated with a concrete noun can be thought of as a classifier of sensory input as indicating presence (or absence) of a referent of that noun. Otherwise put, knowing the perceptual type of a concrete term allows an agent to identify referents of using its sensory apparatus.

Cognitively, perceptual types can be represented e.g. using feature-value pairs, neural networks, (other) statistical methods, or using some hybrid representation. The addition of a colour feature could be described as merely adding a feature-value pair to the perceptual type for "bear", but this is not the only option. For example, it might be that C in the panda example also updates her take on what counts as "bear-shaped". Because pandas chew tough bamboo stalks for nourishment, they have highly developed muscles around their jaw and large crushing molars which makes their heads very round in appearance. If C continues to have a single shape-detector for pandas and bears, she may eventually retrain her bear-shape classifier slightly to encompass this slight difference in appearance[1].

[1] The inclusion of perceptual types, regarded as embodied classifiers of sensory data, in meanings of linguistic expressions raises the problem how speakers can be coordinated with respect to something which potentially has rather different physiobiological realisations in each speaker (in terms of the structure of neural networks, for example). A quick and rough response to this question is that speakers are coordinated on a perceptual type insofar as they agree on their classifications of instances as belonging to that type or not, regardless of how the classifier is implemented in each speaker. On this view, meanings are neither purely abstract entities nor purely psychological entities; instead, they are social entities which have have both an abstract and a psychological side.

Apart from perceptual type, several other aspects of meanings could potentially be coordinated on, such as compositional semantics ([12]), ontological category ([14], [12]), and connotative meaning.

3.2 Semantic Plasticity and Inference

We have argued above that the phenomena of semantic coordination and semantic plasticity are fundamental aspects of the nature of linguistic meaning. Now, how does semantic coordination and plasticity affect inference and the notion of content of a dialogue?

Consider this (made up) example:

- C: "I like all animals on TV"
- C: "I saw a bear on TV yesterday"
- ...
- C: "Bear"
- D: "Yes it's a nice panda"
- C: "Panda"
  - C updates meaning for "panda" and "bear"

What content can be inferred from this dialogue? That C likes bears? That C likes pandas? The problem is that since C's take on the meaning of "bear" and "panda" has changed, we don't really know whether C saw a bear or a panda on TV.

Here is another example:

- C: "I have a pair of blue gloves"
- ...
- C: "Blue ball"
- D: "Yes it's a nice green ball"
- C: "Green ball"
  - C updates perceptual type (classifiers) for "blue" and "green"

This can be seen as a case of C and D coordinating their takes on the meaning of "green" (and "blue"). Again we may ask, what content can be inferred from this dialogue? That *C* has a pair of blue gloves? That C has a pair of green gloves? Well, that depends on which classifier (sensor, detector) triggers when C looks at the gloves in question. To make matters even worse, in a linguistic community there may be no clear-cut fact as to the correct name of a given colour. Different speakers may have different takes, and these may change during interactions and over time[2].

[2] One could imagine a wide variety of examples of semantic plasticity causing problems for inference. Above, we have focused on cases where speakers initially have different takes on the meaning of an expression, but eventually converge. A different case, no less relevant to the problems we are raising here, are cases where the meaning of an expression subtly changes during a dialogue, even though the speakers have at every instance quite similar takes on the meaning in question, and may perhaps not even notice the semantic changes taking place.

It appears that semantic plasticity causes serious problems for view of a dialogue as a conjunction of propositions from which inferences may be drawn using the rules of standard (first-order or higher order) logic. Since speaker's takes on the meanings of words may change during the dialogue, a use of an expression at time t may not have the same meaning as the use of that expression and time $t' > t$, and this may invalidate an inference from a proposition P uttered at t and a proposition P' uttered at t' to the conjunction $P\&P'$.

3.3 Semantic Plasticity and Classical Model-Theoretic Semantics

The solution to this problem, it may be thought, is to incorporate semantic plasticity in our model-theoretical semantics for dialogue. However, it is difficult to see how this could be done. Let's use the gloves example for illustration. We assume that the interpretation of the phrase "I have blue gloves" uttered by C involves using F to yield the referent (some object in the model A, let's call it a_{123}). Roughly, we have F(C's blue gloves) = a_{123}. Now, the meaning of the predicate "blue" is given as a set of objects in A, including a_{123}: F(blue) = {..., a_{123}, ...}.

Now, to update the meaning of "blue", what can we do? Well, the interpretation of "blue" is a set, so what we can do is to add elements to, or remove elements from F(blue). For example, as C's take on the meaning of "blue" changes as a result of D's utterance "Yes, it's a nice green ball", C should update her take on F so that F(blue) after the update no longer contains any green objects; it should instead contain exactly all blue objects in A[3].

This seems to require that C is able to compute, for all objects in the model A, whether they should be included in the new meaning of "blue"; i.e., for all objects known to C, whether they are blue (in the new sense). This in turn requires some method of deciding, for each element in A, whether to include it in $F(blue)$ or not. What would this be? Note that there is no "colour sample" to compare to, other than perhaps the objects in sight, e.g. the blue ball. It seems we need some generalisation or description to compare to, but we only have an extension! Furthermore, we need to apply this method to all elements of A. If used in this way, model theory seems to commit us to an unrealistic theory of conceptual learning.

4 An Alternative: TTR

What is needed to be able to account for semantic coordination and semantic plasticity in a formal semantic theory? Well, what seems to be missing in classical model-theoretical semantics is some *structured* representation of the *intensions* of linguistics expressions. In classical model theory, all we have is (minimally) structured representations of extensions, and as we have seen above this is not enough.

[3] "Blue" can also be analysed as a predicate modifier, i.e. a function from predicates to extensions: blue(glove)(g). All this does is say that "blue" can mean different things depending on the object it is ascribed to. However, we still end up with extensions, i.e. sets, which can only be modified by adding or deleting elements.

4.1 Representing Concepts Using TTR

One theory offering structured meaning representations is type theory with records (TTR) as characterized in [10,11] and elsewhere. An advantage of TTR is that it integrates logical techniques such as binding and the lambda-calculus into feature-structure like objects called record types. Thus we get more structure than in a traditional formal semantics and more logic than is available in traditional unification-based systems. The feature structure like properties are important for developing similarity metrics on meanings and for the straightforward definition of meanings modifications involving refinement and generalization. The logical aspects are important for relating our semantics to the model and proof theoretic tradition associated with compositional semantics. Below is an example of a record type:

$$\begin{bmatrix} \text{ref} & : \text{Ind} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \\ \text{shape} & : \text{shape(ref, BearShape)} \end{bmatrix}$$

A record of this type has to have fields with the same labels as those in the type. (It may also include additional fields not required by the type.) In place of the types which occur to the right of ':' in the record type, the record must contain an object of that type. Here is an example of a record of the above type:

$$\begin{bmatrix} \text{ref} & = a_{123} \\ \text{size} & = \text{sizesensorreading85} \\ \text{shape} & = \text{shapesensorreading62} \\ \text{colour} & = \text{coloursensorreadning78} \end{bmatrix}$$

Thus, for example, what occurs to the right of the '=' in the ref field of the record is an object of type *Ind*, that is, an individual. Types which are constructed with predicates like *size* and *shape* are sometimes referred to as "types of proof". The idea is that something of this type would be a proof that a given individual (the first argument) has a certain size or shape (the second argument). One can have different ideas of what kind of objects count as proofs. Here we are assuming that the proof-objects can be readings from sensors. We imagine that the mapping from sensor readings to types involves sampling of analogue data in a way that is not unsimilar to the digitization process involved, for example, in speech recognition.

Types constructed with predicates may also be *dependent*. This is represented by the fact that arguments to the predicate may be represented by labels used on the left of the ':' elsewhere in the record type. This means, for example, that in considering whether a record is of the record type, you will need to find a proof that the object which is in the ref-field of the record has the size represented by *MuchBiggerThanMe*. That is, this type depends on the value for the ref-field.

Some of our types will contain *manifest fields* [13] like the ref-field in the following type:

$$\begin{bmatrix} \text{ref}{=}a_{123} & : \text{Ind} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \\ \text{shape} & : \text{shape(ref, BearShape)} \end{bmatrix}$$

$\left[\text{ref}{=}a_{123}{:}\text{Ind}\right]$ is a convenient notation for $\left[\text{ref}{:}\text{Ind}_{a_{123}}\right]$ where $\text{Ind}_{a_{123}}$ is a *singleton type*. If $a : T$, then T_a is a singleton type and $b : T_a$ (i.e. b is of type T_a) iff $b = a$. Manifest fields allow us to progressively specify what values are required for the fields in a type.

An important notion in this kind of type theory is that of *subtype*. For example,

$$\begin{bmatrix} \text{ref} & : \text{Ind} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \end{bmatrix}$$

is a subtype of

$$\left[\text{ref} : \text{Ind}\right]$$

as is also

$$\left[\text{ref}{=}a_{123} : \text{Ind}\right]$$

The subtype relation corresponds to that of *subsumption* in typed feature structures. This gives us the ability to create type hierarchies corresponding to ontologies (in the sense, for example, of OWL). Such ontologies (coded in terms of record types) play an important role in our notion of resources available to an agent. In fact, modelling concepts in terms of record types commits us to a view of concepts which is very closely related to work on ontologies. But our view of the creation of local situation specific and domain related resources in addition to generic resources means that agents have access not to a single generic ontology but also situation specific and domain related ontologies. And, perhaps most important of all, the process of semantic coordination with an interlocutor can involve local *ad hoc* adjustment to an ontology. This plays an important role in characterizing the options open to an agent when confronted with an innovative utterance. We attempt to illustrate this below by working in more detail through a specific example.

4.2 A TTR Account of the "Panda" Example

Using TTR, we can now represent the dynamics of concepts in the example dialogues above. For example, in the "panda" example, we assume that, before D's utterance, C has a concept of "bear" as shown in Figure 1.

As mentioned above, we assume that C correctly understands D's utterance as offering "panda" as an alternative for "bear". Now, assuming that C has not observed the word "panda" before, C needs to create a panda-concept. Here C is confronted with a fundamental choice. Should a condition 'panda(ref)' be

$$\begin{bmatrix} \text{ref} & : \text{Ind} \\ \text{phys} & : \text{phys-obj(ref)} \\ \text{anim} & : \text{animate(ref)} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \\ \text{shape} & : \text{shape(ref, BearShape)} \\ \text{bear} & : \text{bear(ref)} \end{bmatrix}$$

Fig. 1. C's "bear" concept before the interaction

added to the concept in addition to the condition 'bear(ref)' making the panda concept be a subtype of the bear concept or should the panda condition replace the bear condition, making panda and bear sisters in the ontology? There is not enough evidence in this simple exchange to determine this.[4] We will choose to replace the bear condition with the panda condition. But there is more that must happen.

C has observed that the use of "panda" in s refers to the focused object a_{123}. Following the principle of contrast mentioned above, C takes "panda" to have a different meaning than "bear" in some respect other than that it is a panda as opposed to a bear, and looks for something about a_{123} which might distinguish it from previously observed bears. For example, the child might decide that it is the colour (black and white) that distinguishes it from previously observed bears (which have all been brown)[5]. C now creates a concept $[\text{panda}]^C$ of "panda" as shown in Figure 2.

$$\begin{bmatrix} \text{ref} & : \text{Ind} \\ \text{phys} & : \text{phys-obj(ref)} \\ \text{anim} & : \text{animate(ref)} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \\ \text{shape} & : \text{shape(ref, BearShape)} \\ \text{colour} & : \text{colour(ref, BlackAndWhite)} \\ \text{panda} & : \text{panda(ref)} \end{bmatrix}$$

Fig. 2. C's concept after integrating D's utterance

But now if colour is being used to distinguish between bears and pandas in situation s, C should create a refined bear concept, namely Figure 3 reflecting the hypothesis that bears are brown.

4.3 TTR and Semantic Plasticity

As we can see, TTR allows structured meanings which can be updated as a result of interaction in dialogue. Generally, we can think of (takes on) meanings

[4] And indeed many people can reach adulthood, the present author included, without being sure whether pandas are a kind of bear or not.

[5] This account relies on C having a memory of previously observed instances of a concept, in addition to the concept itself (which in the case of "bear" does not contain information about colour).

$$\begin{bmatrix} \text{ref} & : \text{Ind} \\ \text{phys} & : \text{phys-obj(ref)} \\ \text{anim} & : \text{animate(ref)} \\ \text{size} & : \text{size(ref, MuchBiggerThanMe)} \\ \text{shape} & : \text{shape(ref, BearShape)} \\ \text{colour} & : \text{colour(ref, Brown)} \\ \text{bear} & : \text{bear (ref)} \end{bmatrix}$$

Fig. 3. C's "bear" concept after integrating D's utterance

as types, or classifiers, of situations. All represented aspects of meaning may be modified, both "propositional" information, e.g. ontological semantics, and perceptual types (sensors) which may be updated (retrained). In contrast to classical model theory, we are not updating sets (extensions), but structured classifiers (intensions).

Another way of looking at the difference between TTR and classical model theory is to note that TTR assigns objects to types constructed from predicates ("propositions"), rather than to the predicates themselves. A fundamental type-theoretical intuition is that something of type $P(a)$ is whatever it is that counts as a proof object (or a *verification*) for the fact that a falls under the predicate P. In a model theory for TTR, the interpretation function F assigns a value to a type $P(a)$ instead of a predicate P.

Now, let's assume that a TTR model contains objects of two basic kinds: types constructed from predicates ("propositions"), and objects (proofs, verifications) of such types. Verifications may include sensor readings (e.g. output from a vision recognition system) taken to correspond to situations in the world. The semantic representation of a predicate (whether derived from a noun phrase, verb phrase, etc.) can now be structured (as a record type) and contain various kinds of information, including perceptual type. The compositional semantic representation of a sentence can be similarly structured.

In such a setting it seems useful to think of perceptual types as compositional. We thus take it that the perceptual type of a sentence can be composed from the perceptual types of the constituent expressions of the sentence:

- The perceptual type of "blue" is instantiated by perceptions of blue things
- The perceptual type for "ball" is instantiated by perceptions of balls
- The perceptual type for "blue ball" is instantiated by perceptions of blue balls
- ...
- The perceptual type for "Naomi has a blue ball" is instantiated by perceptions of situations where Naomi has a blue ball

Here, "perceptions" means "sensor readings from a perceptual systems". The idea is that when computing the meaning of a sentence, we construct a complex sensor from simpler sensors.

When modifying the meaning of an expression e (e.g. "blue") in TTR, we can do this by updating some part of the structured meaning of e, e.g. the perceptual

type. As mentioned, one kind of perceptual type is the sensor, which is assumed to give a positive reading when perceiving a certain kind of situation in the world (e.g. one involving gloves, or one involving Naomi having a blue ball). One way of updating a perceptual type, then, is as re-training a sensor. This seems more sensible than re-classifying all known objects according to an some unknown algorithm, and relies crucially on having a structured representation intensions of linguistic expressions.

4.4 Indeterminate Extensions

This view of intensional meaning as involving sensors interfacing with the real world has important consequences for how we think about extensions. The reason is that a re-trained sensor cannot immediately tell us which objects and situations will trigger it – the only way to find out is to apply it in various situations and see what happens. This means that there is no longer a definite extension of an expression apart from specific situations of language use. Instead, in each instance of use of an expression, the situation of use is classified as falling under the intension of the expression or not.

In general, especially in humans, a classifier does not operate in a vacuum and may be sensitive to a multitude of aspects of the perceived situation, including shared and individual goals, various social aspects of the situation, perceptual factors (light conditions etc), priming effects and more. Classification is thus a situated, complex and stochastic process. This is also true of classifiers involving "propositions", i.e. classifiers whose extensions are (sets of) situations. Whether a certain classifier ("proposition") classifies a situation ("is true") or not cannot be determined in the abstract, but only by applying the classifier to a situation[6]. Accepting this argument does not mean that the notion of truth becomes meaningless, but it does pose a serious problem which needs to be addressed in any formal theory taking semantic coordination and plasticity seriously.

The notion of type which we end up with if we want to our theory to cover the perceptual aspects of meanings of linguistic expression is in (at least) one respect very different from the notion of type in traditional mathematical type theory, where the objects of any type (i.e. extensions) are always assumed to be known. Traditional mathematical type theory is similar in this respect to classical model-theoretic semantics. However, type theory appears to be more open to removing

[6] This does not mean that perception and truth are conflated. If C and her friend E were at the zoo and they both referred to the panda as a bear, it would still be a panda. However, instead of saying that they were wrong, one could instead say that they (inadvertedly) established a local semantic convention of not making any distinction between pandas and bears. Given this convention, they were perfectly right to call the panda a bear. In cases of expressions referring to natural kinds such animal species, it is of course fairly clear that there is an external authority deciding the "proper" convention, e.g. the community of zoologists. However, in other cases, such as in the case of colour terms, it is much less clear that there is any such authority. As mentioned above, there may be no clear-cut fact as to the correct name of a given colour, and different speakers may have different takes, which may change during interactions and over time.

this assumption than model theory. Treating types as first-class objects, which can be manipulated and used as classifiers of novel situations, means that type theory is not as dependent as model theory on having determinate extensions of expressions.

4.5 Model-Theoretic Sensor Semantics?

But wait a minute – can't we apply the "sensor" trick in classical model-theoretic semantics? Can perceptual semantics be modelled in classical model-theoretic semantics? One idea is to let the model be a collection of sensor readings for a robot C:

- F(C's blue gloves) =glovesensor(345)
- $F(blue) = \{\ldots,$ glovesensor(345), $\ldots\}$

The idea is that whenever C's blue gloves are within C's field of vision, glovesensor(345) is active, and whenever a blue object is within C's field of vision, one of the sensors readings in F(blue) is active. Again, however, there is no separate sensor for the colour blue; instead it is given purely extensional definition as a set of sensors, one for each object falling under the general concept.

To update the meaning of "blue", we can add and remove elements of $F(blue)$ so that $F(blue)$ after the update contains exactly those sensor readings which are active when blue (according to the new meaning) objects are in A's field of vision. This requires that C is able to compute, for all sensor readings in A, whether they should be included in the new meaning of "blue"; i.e., for all objects known to C, whether they are blue (in the new sense). This in turn requires some method of deciding, for each sensor reading in A, whether to include it in $F(blue)$ or not. Again, we can ask: What would this be? There is still no "colour sample" to compare to, other than perhaps the objects in sight, e.g. the blue ball. It seems we need some generalisation or description to compare to, but we only have an extension. Again, model theory seems to commit us to an unrealistic theory of conceptual learning.

5 Conclusion

The phenomena of semantic coordination and semantic plasticity pose some general requirements on any formal theory purporting to account for them. Firstly, any such theory needs to treat intensions of linguistic expressions as first-class objects, i.e., it needs something equivalent to types. Secondly, these types need to be structured. The reason is that accounting for semantic plasticity in a way that makes sense requires the possibility of modifying intensions, and only structured objects can be modified. Thirdly, as some meanings involve classifying situations in the world based on perceptions thereof, and since classification of real-world situations is (in general) a complex stochastic process, any formal theory of semantic plasticity needs to allow for a fundamental indeterminacy of extensions of linguistic expressions.

While SDRT currently relies on classical model-theoretic semantics, it also has some features which may be useful if one would extend it to account for semantic plasticity. It insists on the utility of a level of representation between language and model, namely the language of SDRSs. Also, SDRSs are structured meaning representations. It would perhaps be possible to recast SDRT in a type-theoretic framework, thereby making it better equipped to deal with semantic plasticity. Recent work by Asher [1] on a type-theoretic account of word meaning is encouraging in this respect.

To sum up: If we go for the model-theoretic view of meaning, we will (it seems) have great difficulty accounting for semantic plasticity. TTR can account for semantic plasticity, but this seems to complicate the idea that the truth value of every proposition is at every point in time determinate (true or false). Furthermore, inferences involving "propositions" (types constructed from predicates) containing predicates whose meaning has changed during the dialogue are not always reliable. So semantic coordination and semantic plasticity seems to undermine both model-theoretic interpretation and the ability to draw inferences from whole dialogues (seen as a conjunction of propositions). So – do dialogues still have content?

Acknowledgments. This research was supported by The Swedish Bank Tercentenary Foundation Project P2007/0717, Semantic Coordination in Dialogue. I would like to thank Robin Cooper for useful discussion.

References

1. Asher, N.: Lexical Meaning in Context. Cambridge University Press, Cambridge (2010), http://www.cambridge.org/
2. Asher, N., Lascarides, A.: Logics of Conversation. Cambridge University Press, Cambridge (2003), http://www.cambridge.org/
3. Brennan, S.E., Clark, H.H.: Conceptual pacts and lexical choice in conversation. Journal of Experimental Psychology: Learning, Memory and Cognition 22, 482–493 (1996)
4. Briscoe, E.J. (ed.): Linguistic Evolution through Language Acquisition: Formal and Computational Models. Cambridge University Press, Cambridge (2002), http://www.isrl.uiuc.edu/~amag/langev/paper/briscoe2002editedbook.html
5. Chierchia, G., Turner, R.: Semantics and property theory. Linguistics and Philosophy 11(3) (1988)
6. Clark, E.V.: Young children's uptake of new words in conversation. Language in Society 36, 157–182 (2007)
7. Clark, E.: The lexicon in acquisition. Cambridge University Press, Cambridge (1993)
8. Clark, E.V., Wong, A.D.W.: Pragmatic directions about language use: Offers of words and relations. Language in Society 31, 181–212 (2002)
9. Clark, H.H., Wilkes-Gibbs, D.: Refering as a collaborative process. Cognition 22, 1–39 (1986)
10. Cooper, R.: Austinian truth, attitudes and type theory. Research on Language and Computation 3, 333–362 (2005)

11. Cooper, R.: Type theory with records and unification-based grammar. In: Hamm, F., Kepser, S. (eds.) Logics for Linguistic Structures. Mouton de Gruyter (2008), http://www.ling.gu.se/~cooper/records/ttrhpsg.pdf
12. Cooper, R., Larsson, S.: Compositional and ontological semantics in learning from corrective feedback and explicit definition. In: Edlund, J., Gustafson, J., Hjalmarsson, A., Skantze, G. (eds.) Proceedings of DiaHolmia, 2009 Workshop on the Semantics and Pragmatics of Dialogue (2009)
13. Coquand, T., Pollack, R., Takeyama, M.: A logical framework with dependently typed records. Fundamenta Informaticae XX, 1–22 (2004)
14. van Diggelen, J., Beun, R.J., Dignum, F., van Eijk, R.M., Meyer, J.J.: Ontology negotiation in heterogeneous multi-agent systems: The anemone system. Applied Ontology 2, 267–303 (2007)
15. Garrod, S.C., Anderson, A.: Saying what you mean in dialogue: a study in conceptual and semantic co-ordination. Cognition 27, 181–218 (1987)
16. Healey, P.: Expertise or expertese?: The emergence of task-oriented sub-languages. In: Shafto, M., Langley, P. (eds.) Proceedings of the 19th Annual Conference of the Cognitive Science Society, pp. 301–306 (1997)
17. Larsson, S.: Coordinating on ad-hoc semantic systems in dialogue. In: Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue (2007)
18. Larsson, S.: Formalizing the dynamics of semantic systems in dialogue. In: Cooper, R., Kempson, R. (eds.) Language in Flux - Dialogue Coordination, Language Variation, Change and Evolution. College Publications, London (2008)
19. Larsson, S., Cooper, R.: Towards a formal view of corrective feedback. In: Alishahi, A., Poibeau, T., Villavicencio, A. (eds.) Proceedings of the Workshop on Cognitive Aspects of Computational Language Acquisition, EACL, pp. 1–9 (2009)
20. Lascarides, A.: Agreement, disputes and commitments in dialogue. Journal of Semantics 26, 109–158 (2009)
21. Pickering, M.J., Garrod, S.: Toward a mechanistic psychology of dialogue. Behavioral and Brain Sciences 27(02), 169–226 (2004)
22. Steels, L., Belpaeme, T.: Coordinating perceptually grounded categories through language: A case study for colour. Behavioral and Brain Sciences 28(4), 469–489 (2005), http://www.isrl.uiuc.edu/~amag/langev/paper/steels_BBS_color.html, target Paper, discussion 489-529

Contextual Analysis of Word Meanings in Type-Theoretical Semantics

Zhaohui Luo*

Dept of Computer Science, Royal Holloway, Univ of London
Egham, Surrey TW20 0EX, U.K.
zhaohui@cs.rhul.ac.uk

Abstract. Word meanings are context sensitive and may change in different situations. In this paper, we consider how contexts and the associated contextual meanings of words may be represented in type-theoretical semantics, the formal semantics based on modern type theories. It is shown, in particular, that the framework of coercive subtyping provides various useful tools in the representation.

1 Introduction

Word meanings are context sensitive. In any lexical semantics, it is important to spell out how different word meanings may be disambiguated. In this paper, we make proposals on how some word meanings may be represented in type-theoretical semantics [34,21], the formal semantics based on modern type theories, and how disambiguation can be done automatedly based on the representation method. In particular, it is shown that coercive subtyping [20] provides important tools for such a representation in type theory.

We start by dealing with homonymy, when the meanings of a homonym (e.g., 'run') can be disambiguated by their typings in the type-theoretical semantics. In such cases, *overloading* provides a suitable representation mechanism for sense enumeration. We show how coercive subtyping supports the overloading of word meanings and the automated sense selection.

As word meanings may change from context to context, some uses are only meaningful in certain contexts, not in others. Since subtyping relations are crucial in representing such informal contexts, we should extend the formal notion of context (in type theory and other logical systems) to incorporate the assumption of subtyping relations. We formally introduce coercion contexts and show how they may be used in contextual analysis.

The meanings of some homonyms may not be distinguished by their typings. For example, in the type-theoretical semantics, common nouns (e.g., 'bank') are interpreted as types; therefore, the semantic typings of CNs are the same (different types do not have different 'typings') and cannot be used in disambiguation.

* This work is partially supported by the research grant F/07-537/AJ of the Leverhulme Trust in U.K.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 159–174, 2011.

For such disambiguation, we need to introduce *local coercions* (i.e., coercion contexts for terms) so that semantic interpretations can be given as intended.

There are some proposals that words should be given complex and structured meanings so that contributions to meaning generation can be made effectively (cf., Pustejovsky's work on Generative Lexicon Theory [31] and the opposing views of lexical atomism [12,32]). Although the author does not take a philosophical stand on this, it is worth stating that structured lexical entries have the potential to contribute to a computational treatment of the semantics in, for instance, building an inference engine based on a formal semantics. We shall study how structured lexical meanings of CNs may be represented by Σ-types and show that such representations are consistent with the successful treatment of copredication in type-theoretical semantics [21].

For all of the proposals in this paper, experiments have been done in the proof assistant Coq [9], which implements a modern type theory. This may serve to verify the proposals, but more importantly, it can be seen as the first step towards computer-assisted linguistic reasoning with proof engines that implement the type-theoretical semantics.

In §2, we first give an introduction to type-theoretical semantics. The representation of sense enumeration and selection by coercive subtyping is described in §3. Coercion contexts and local coercions, and their uses in contextual analysis, are studied in §4. In §5, we consider how some of the structured lexical entries may be represented as Σ-types in the context of studying copredication. Finally, we briefly describe our Coq experiments in §6, followed by the conclusion where some future work is discussed.

2 Type-Theoretical Semantics

By a *type-theoretical semantics*, we mean a formal semantics of natural languages based on modern type theories such as Martin-Löf's predicative type theory [25,29] and the impredicative type theory UTT [18]. Such a semantics is in the tradition of the Montague grammar [27] but the powerful type structures in a modern type theory provide new useful mechanisms for formal semantics of various linguistic features, some of which have been found difficult to describe in the Montagovian setting.

In this section, we give a brief overview of the basics of type-theoretical semantics, partly to lay down the background and partly to set up notations and terminologies.

2.1 A Brief Overview

The Montague grammar is based on Church's simple type theory [6], which is a *single-sorted* logic. In Montague grammar, there is a universal type e of entities: a common noun is interpreted as a function of type $e \to t$ and a verb or an adjective as a function of type $(e \to t) \to (e \to t)$, where t is the type of truth values.

In contrast, a modern type theory can be considered as a *many-sorted* logical system, where there are many sorts called *types* that may be used to stand for the domains to be represented. These types include:

- the *propositional types* (or logical propositions – see §2.2),
- the *inductive types* such as the type of natural numbers and Σ-types of dependent pairs (see the latter in §2.3), and
- other more advanced type constructions such as *type universes* (see §2.5).

Because of this many-sortedness, it is natural to interpret the noun phrases as types. Here are several basic interpretation principles one may adopt in a type-theoretical semantics [34]:

- Common nouns are interpreted as types. For instance, the CNs 'man' and 'human' can be interpreted as types $[\![man]\!]$ and $[\![human]\!]$, respectively.
- An adjective is interpreted as a predicate over the type that interprets the domain of the adjective. For instance, 'handsome' may be interpreted as a predicate $[\![handsome]\!] : [\![man]\!] \to Prop$, where $Prop$ is the type of logical propositions (see §2.2).
- Modified CNs are interpreted as Σ-types (see §2.3 for more details).

Furthermore, the framework of coercive subtyping provides us additional richer means for type-theoretical semantics [20,21] (see §2.4).

2.2 Embedded Logic

A modern type theory has an *embedded logic* (or *internal logic*) based on the propositions-as-types principle [10,14]. For example, in Martin-Löf's predicative type theory, the logical proposition $A\&B$ corresponds to the product type $A \times B$ (a special case of Σ-type – see below) and a pair of a proof of A and a proof of B corresponds to an object of the product type. Similarly, this correspondence extends to other logical operators: the logical implication ($\supset$) corresponds to the function types ($\to$), the universal quantifier ($\forall$) to the dependent Π-types, etc.

For Martin-Löf's type theory, the embedded logic is first-order and, for impredicative type theories such as ECC/UTT [18], the embedded logics are second-order or higher-order, where there is a type $Prop$ of logical propositions. Formally, $Prop$ is a totality and one can quantify over it to form other propositions (and this process is regarded as 'circular' by predicativists [11] or 'impredicative', in the technical jargon).[1]

In this paper, we shall use $Prop$ in linguistic interpretations. In a type-theoretical semantics, an assertive sentence is interpreted as a proposition of type $Prop$ and a verb or an adjective as a predicate of type $A \to Prop$, where A is the domain whose objects the verb or adjective can be meaningfully applied to. For instance, consider the following sentence:

[1] $Prop$ is very much like the type t in the simple type theory. The main difference is that, in modern type theories, we have explicit proof terms of logical propositions.

(1) John is handsome.

With $[\![John]\!] : [\![man]\!]$ and $[\![handsome]\!] : [\![man]\!] \to Prop$, the above sentence (1) is interpreted as proposition $[\![handsome]\!]([\![John]\!])$ of type $Prop$.

2.3 Dependent Types

Modern type theories contain *dependent types*. For instance, When A is a type and P is a predicate over A, $\Pi x{:}A.P(x)$ is the dependent function type that, in the embedded logic, stands for the universally quantified proposition $\forall x{:}A.P(x)$. Π-types degenerates to the function type $A \to B$ in the non-dependent case.

Another example of dependent type is the so-called Σ-types. If A is a type and B is an A-indexed family of types, then $\Sigma(A, B)$, or sometimes written as $\Sigma x{:}A.B(x)$, is a type, consisting of pairs (a, b) such that a is of type A and b is of type $B(a)$. When $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type degenerates into product type $A \times B$ of non-dependent pairs. Σ-types (and product types) are associated projection operations π_1 and π_2 so that $\pi_1(a, b) = a$ and $\pi_2(a, b) = b$, for every (a, b) of type $\Sigma(A, B)$ or $A \times B$.

In a type-theoretical semantics, modified common nouns are interpreted as Σ-types. For instance, $\Sigma([\![man]\!], [\![handsome]\!])$ is the type of handsome men (or more precisely, of those men together with proofs that they are handsome).

Notations for Σ-types. A nested Σ-type can be seen as a type of tuples/modules. The following notations will be adopted (in §5). We shall use

$$\left\{ \begin{array}{l} x_1 : A_1 \\ \ldots \\ x_n : A_n \end{array} \right\}$$

to stand for the Σ-type $\Sigma x_1 : A_1 \Sigma x_2 : A_2 \ \ldots \ \Sigma x_{n-1} : A_{n-1}.\ A_n$. For example,

$$\left\{ \begin{array}{l} name \ \ : [\![man]\!] \\ hproof : [\![handsome]\!](name) \end{array} \right\} \quad \text{stands for} \quad \Sigma([\![man]\!], [\![handsome]\!]).$$

Coq implementation of Σ-types. In the proof assistant Coq [9], Σ-types are implemented by means of the 'record' mechanism, which provide many automatic tools, including the following:

- If the record mechanism is used, the projection operators of the Σ-types are automatedly generated as globally defined terms, named by the 'labels' such as *name* and *hproof* in the above example.
- A projection operator may be indicated as a coercion. For instance, if we want to declare the first projection of the above Σ-type as a coercion, we may simply write `name :> Man` when defining the Σ-type. (cf., the definition of the dot-type `PhyInfo` in Appendix B.2.)

These automatic tools have greatly helped our experiments in Coq on the type-theoretical semantics, as reported in §6 and Appendix B.

2.4 Coercive Subtyping

Coercive subtyping [19,20] is an adequate theory of subtyping for modern type theories. In computer science, coercive subtyping has been implemented in many proof assistants such as Coq [9,35], Lego [23,4], Matita [26] and Plastic [5], and used effectively in interactive theorem proving. It has also been applied to type-theoretical semantics [21].

The basic idea of coercive subtyping is to consider subtyping as an abbreviation mechanism: A is a (proper) subtype of B ($A < B$) if there is a unique implicit coercion c from type A to type B and, if so, an object a of type A can be used in any context $\mathfrak{C}_B[\_]$ that expects an object of type B: $\mathfrak{C}_B[a]$ is legal (well-typed) and equal to $\mathfrak{C}_B[c(a)]$.

For instance, one may introduce $[\![man]\!] < [\![human]\!]$. Then, if we assume that $[\![John]\!] : [\![man]\!]$ and $[\![shout]\!] : [\![human]\!] \to Prop$, the interpretation (3) of (2) is well-typed:

(2) John shouts.

(3) $[\![shout]\!]([\![John]\!])$

according to the rule of coercive subtyping, because $[\![man]\!] < [\![human]\!]$.

2.5 Universes

Other more advanced features in a modern type theory are useful in developing the theory of type-theoretical semantics. For example, one may collect (the names of) some types into a type called a *universe* [25]. Introducing universes can be considered as a reflection principle: such a universe reflects those types whose names are its objects.

In type-theoretical semantics, universes can be introduced to help semantic interpretations. For instance, one may consider the universe CN : $Type$ of all common noun interpretations and, for each type A that interprets a common noun, there is a name $\overline{A}$ in CN. For example,

$$\overline{[\![man]\!]} : \text{CN} \quad \text{and} \quad T_{\text{CN}}(\overline{[\![man]\!]}) = [\![man]\!] .$$

In practice, we do not distinguish a type in CN and its name by omitting the overlines and the operator T_{CN} by simply writing, for instance, $[\![man]\!]$.

Type universes can be used in semantic interpretations. For instance, the universe CN can be used to give semantic interpretations to adverbs.[2] An adverb modifies a verb (an adjective) to result in a verb (adjective) phrase.[3] Since, in a type-theoretical semantics, verbs and adjectives are interpreted as predicates over a variety of domains (rather than over a single domain as in the Montagovian

[2] As far as the author is aware, there are no proposals in the literature on how adverbs should be interpreted in a type-theoretical semantics based on modern type theories.

[3] There are other adverbs. For example, an adverb may modify sentences to result in new sentences and, as in Montague grammar [27], such adverbs are interpreted as functions from $Prop$ to $Prop$.

setting), adverbs such as 'loudly' in 'John talked loudly' and 'simply' in 'That idea is simply ridiculous' would be interpreted as of type

$$\Pi A : \text{CN}.\ (A \to Prop) \to (A \to Prop).$$

For instance, for $[\![talk]\!] : [\![human]\!] \to Prop$, the following phrase (4) can be interpreted as (5), which is of type $[\![human]\!] \to Prop$:

(4) talk loudly

(5) $[\![loudly]\!]([\![human]\!], [\![talk]\!])$

Such interpretations of adverbs are experimented in Coq as, for example, shown in an example at the end of Appendix B.1.

3 Sense Selection via Overloading

In this paper, we shall consider how word meanings can be formalised in a type-theoretical semantics based on modern type theories. It is important to emphasise that the formal presentation of word meanings should naturally give rise to *automated disambiguation* in contexts so that, when considered in an interpretation of sentences or even larger linguistic entities where the concerned word occurs, the correct meaning will be automatedly selected.

In this section, we first start with the simple cases of homonymy and show that, when the meanings of a homonym can be differentiated by means of typing, its sense enumeration can be represented with coercive subtyping.

3.1 Sense Enumeration

A word may be homonymous with several unrelated meanings.[4] For instance, the word 'run' can be used in the following two sentences with different meanings.

(6) John runs quickly.

(7) John runs a bank.

In a type-theoretical semantics, we may have the following two different meanings of 'run', corresponding to the above uses:

(8) $[\![run]\!]_1 : [\![human]\!] \to Prop$

(9) $[\![run]\!]_2 : [\![human]\!] \to [\![institution]\!] \to Prop$

To represent the sense selection model, we need a mechanism that allows *automated selection* of the correct meaning when a sentence is interpreted. For instance, $[\![run]\!]_2$ in (9) should be selected automatedly when (7) is interpreted.

[4] Sense enumeration lexicons have been discussed, e.g., by Pustejovsky [31].

3.2 Simple Sense Selection via Overloading Based on Coercive Subtyping

In type theory, simple cases in the sense selection model can be represented by means of *overloading* (or *ad hoc polymorphism*) [36]. Intuitively, in the above example, the word 'run' is *overloaded* in the sense that it is associated with more than one meaning. Overloading can be supported by coercive subtyping [20,4], as explained below.

Assume that w be an arbitrary homonym with different meanings $[\![w]\!]_i : A_i$ $(i = 1, ..., n)$, where $A_j \neq A_k$ if $j \neq k$. Let $\mathbf{1}_w : Type$ be the inductive unit type with only one object $w : \mathbf{1}_w$ (see Appendix A for the formal details of the unit type). Then, the meaning of w is represented as the coercions $c_i : \mathbf{1}_w \to A_i$ $(i = 1, ..., n)$, defined as:

$$c_i(w) = [\![w]\!]_i : A_i.$$

For instance, the above word 'run' has two meanings $[\![run]\!]_1$ and $[\![run]\!]_2$ (in (8) and (9) in the previous subsection). The sense selection model for these two meanings of 'run' is given by the following two coercions:[5]

$$c_1(run) = [\![run]\!]_1 \quad \text{and} \quad c_2(run) = [\![run]\!]_2 .$$

This has the effect that, for example, in any context $\mathcal{C}_1[run]$ that requires an object of type $[\![human]\!] \to Prop$, we have

$$\mathcal{C}_1[run] = \mathcal{C}_1[c_1(run)] = \mathcal{C}_1[[\![run]\!]_1],$$

and, in any context $\mathcal{C}_2[run]$ that requires an object of type $[\![human]\!] \to [\![institution]\!] \to Prop$, we have

$$\mathcal{C}_2[run] = \mathcal{C}_2[c_2(run)] = \mathcal{C}_2[[\![run]\!]_2].$$

Therefore, through automated insertions of coercions, the sentences (6) and (7) will both be interpreted correctly as expected.

Remark 1. For some homonyms, their different meanings may have the same type and therefore cannot be differentiated by typing. For instance, in a type-theoretical semantics, common nouns are interpreted as types and, therefore, the disambiguation of a homonymous common noun (e.g., 'bank') may depend on further linguistic information (e.g., in the case of 'bank', it may refer to some financial matters). One can resort to local coercions, as to be discussed in §4.2, to give formal interpretations in such situations. □

We have experimented in the proof assistant Coq [9] on sense selection based on the unit type and coercive subtyping. See Appendix A for unit types and Appendix B.1 for an example of homonymy.

[5] If other meanings of 'run' are considered, further coercions are defined accordingly.

4 Representation of Contexts: Coercion Contexts and Local Coercions

Word meanings are context sensitive. For instance, as discussed in the above section, a homonym has different meanings when used in different sentences, or in different *sentential contexts*. In other circumstances, the contexts are not sentential; what they describe can be either a special situation or a specific background. Many usages are only meaningful in such special situations or *local contexts* in which, for instance, the meanings of some words change.

Example 1 (reference transfer). *Consider the following utterance (cf., [30]):*

(10) The ham sandwich shouts.

Assuming that the act of shouting requires that the argument be human, it is obvious that sentence (10) is not well-formed, unless it is uttered by somebody in some special extralinguistic context (e.g., by a waiter in a café to refer to a person who has ordered a ham sandwich). □

4.1 Coercion Contexts in Type Theory

In a type-theoretical semantics, such local contexts can be described by means of the formal notion of context in type theory. Traditionally, a context in type theory is of the form

$$x_1 : A_1,\ ...,\ x_n : A_n$$

where A_i is either a data type, in which case x_i is assumed to be an object of that type, or a logical proposition, in which case the proposition A_i is assumed to be true and x_i a proof of A_i. For example, one may have the following context:

$$m : [\![man]\!],\ hproof : [\![handsome]\!](m)$$

which assumes, in layman's terms, that 'm is a man' and 'm is handsome' (with 'hproof' being a proof).

The formal notion of context can be extended by coercion declarations or subtyping assumptions, as proposed in [21]. A *coercion context* is a context whose entries may be of the form $A <_c B$ as well as the usual form $x : A$. For instance, the following context may be used to describe the special circumstances in a café:

(11) ..., $[\![ham\ sandwich]\!] < [\![human]\!]$, ...

where the subtyping assumption says that a ham sandwich can be coerced into a person (i.e., the person who has ordered a ham sandwich). In a context such as (11), the above sentence (10) can be interpreted satisfactorily as intended.

Formally, we have:

$$\frac{\Gamma \vdash A : Type \quad \Gamma \vdash B : Type \quad \Gamma \vdash c : (A)B}{\Gamma,\ A <_c B\ \ valid} \qquad \frac{\Gamma,\ A <_c B,\ \Gamma'\ \ valid}{\Gamma,\ A <_c B,\ \Gamma' \vdash A <_c B : Type}$$

where $(A)B$ is the functional kind from A to B in the logical framework (see Chapter 9 of [18] for formal details.) In other words, coercions can now be introduced in contexts and they are only valid 'locally' in the context where they are introduced. For example, sentence (10) can be reasonably interpreted in a context which contains the subtyping as shown in (11) and, otherwise, it cannot.

Remark 2. (coherent context) Please note that validity of a context is not enough anymore for it to be legal. One needs to make sure that the context is *coherent*, in the sense that the declared coercions in the context do not lead to more than one coercion between two types. Since it requires some formal backgrounds to be treated more concisely, its details are omitted here. □

Example 2. *Consider the following example (adapted from [24]):*

(12) Every linguist drinks a glass.

Let's assume that 'drink' be interpreted as:

$$[\![drink]\!] : [\![animated]\!] \to [\![liquid]\!] \to Prop.$$

Now, since not every container contains drinks, there should be a special context in which the above sentence (12) can be interpreted. The coercion context should contain the following subtyping relations:

$$[\![glass]\!] < [\![beverage]\!], \quad [\![beverage]\!] < [\![liquid]\!], \quad [\![linguist]\!] < [\![animated]\!]$$

Then, in a coercion context with the above, (12) can be interpreted as:

(13) $\forall l : [\![linguist]\!] \,.\, \exists g : [\![glass]\!] \,.\, [\![drink]\!](l, g).$

In the coercion context, (13) is well-typed. □

4.2 Local Coercions in Terms

Consider the following phrases that use the homonym 'bank':

(14) the bank of the river

(15) the richest bank in the city

From the previous subsection, we know that (14) and (15) can be interpreted in the following contexts (16) and (17), respectively:

(16) ... $[\![bank]\!] < [\![riverside]\!]$, ...

(17) ... $[\![bank]\!] < [\![financial\ institution]\!]$, ...

Now, what if we want to use the word 'bank' twice with these two different meanings in the same text (e.g., in a text where (14) is followed not far by

(15))? Although one is not forbidden to introduce a context that contains both subtyping assumptions:

(18) ... $[\![bank]\!] < [\![riverside]\!]$, ... $[\![bank]\!] < [\![financial\ institution]\!]$, ...

it may not be known which subtyping relation should be used in which sentence (unless some extra information is available). Automatic selection fails. Sometimes, the situation is even worse: it may be impossible to assume all the subtyping relations because it would lead to incoherence.

Such a problem can be solved by introducing *local coercions* – coercions that are only effective locally for some terms (expressions in type theory). Coercions may be introduced into terms by the following rule:

$$\frac{\Gamma,\ A <_c B \vdash J}{\Gamma \vdash \mathbf{coercion}\ A <_c B\ \mathbf{in}\ J}$$

where J is any of the following four forms of judgement:

$$k : K, \quad k = k' : K, \quad K\ kind, \quad \text{and} \quad K = K'.$$

For instance, with $J \equiv k : K$, we have

$$\frac{\Gamma,\ A <_c B \vdash k : K}{\Gamma \vdash \mathbf{coercion}\ A <_c B\ \mathbf{in}\ k : K}$$

Intuitively, the coercions declared locally are only effective in the expressions in the scope of the keyword **in**. For instance, for

$$\mathbf{coercion}\ [\![bank]\!] < [\![financial\ institution]\!]\ \mathbf{in}\ e,$$

the subtyping relation between $[\![bank]\!]$ and $[\![financial\ institution]\!]$ is only effective in the expression e, not outside e.

The key word **coercion** distributes through the components of J. For example, the following two judgements are identified:

$$\mathbf{coercion}\ A <_c B\ \mathbf{in}\ (k : K)$$
$$(\mathbf{coercion}\ A <_c B\ \mathbf{in}\ k)\ :\ (\mathbf{coercion}\ A <_c B\ \mathbf{in}\ K)$$

The introduction of local coercions broadens the scope of interpretation in applying the above techniques. For example, assuming that (14) and (15) have interpretations $[\![(14)]\!]$ under $[\![bank]\!] < [\![riverside]\!]$ and $[\![(15)]\!]$ under $[\![bank]\!] < [\![financial\ institution]\!]$, respectively, then the following two terms give their semantics and can be used together with no problem:

(19) **coercion** $Bank < Riverside$ **in** $[\![(14)]\!]$

(20) **coercion** $Bank < FinancialInst$ **in** $[\![(15)]\!]$

Since the coercions only take effects in the relevant expressions, the intended semantics gets represented correctly.

5 Structured Lexical Entries: Copredication and Beyond

Lexical entries can either be atomic, as advocated by Fodor and Lepore's lexical atomism [12,32], or complex and structured as proposed and studied by Pustejovsky and others [31]. According to lexical atomism, the basic words such as 'book' may only be properly interpreted in an atomic way, while Pustejovsky and others believe that they should be represented by means of rich and structured entities which, when combined with other linguistic entities in a sentence, make important contributions to meaning generations.

In this paper, we shall not argue for or against whether the lexicon should be generative and structured, but only to investigate, if words should be represented as complex and structured entities, how it can be done in a type-theoretical semantics. We want to add that, if one can represent word meanings successfully as structured entities that contribute to meaning generation, it will make substantial contribution to natural language processing in practice.

In this section, we shall first review how copredication (cf., [3]) can be captured in the type-theoretical semantics [21] and then show how complex and structured lexical entries, represented by Σ-types,[6] can be dealt with satisfactorily in this respect.

5.1 Copredication and Dot-Types

The dot-type and its use in lexical semantics were first proposed by Pustejovsky [31] and further studied by many others including, for example, Asher in the study of copredication [2,3]. In [21], the author has proposed a type-theoretic formal treatment of dot-types, with the help of coercive subtyping, and shown that the type-theoretical semantics with coercive subtyping gives a satisfactory treatment of copredication, among others.

Example 3 (copredication). *Let* PHY *and* INFO *be the types of physical objects and informational objects, respectively. One may consider the dot-type* PHY$\bullet$ INFO *as the type of the objects with both physical and informational aspects. A dot-type is a subtype of its constituent types:* PHY $\bullet$ INFO $<$ PHY *and* PHY $\bullet$ INFO $<$ INFO. *A book may be considered as having both physical and informational aspects, reflected as:*

$$(*) \qquad [\![book]\!] < \text{PHY} \bullet \text{INFO}.$$

Now, consider the following sentence:

(21) John picked up and mastered the book.

In a type-theoretical semantics, we may assume

$$[\![pickup]\!] : [\![human]\!] \to \text{PHY} \to Prop$$
$$[\![master]\!] : [\![human]\!] \to \text{INFO} \to Prop$$

[6] Cooper [8,7] has proposed to use dependent record kinds in Martin-Löf's type theory to represent lexical entries. The idea of using Σ-types to represent structured lexical entries was proposed, but not studied in any depth, in [22].

Because of the above subtyping relationship $(*)$ *(and contravariance of subtyping for the function types), we have*

$$\begin{aligned} [\![pickup]\!] &: [\![human]\!] \to \textsc{Phy} \to Prop \\ &< [\![human]\!] \to \textsc{Phy} \bullet \textsc{Info} \to Prop \\ &< [\![human]\!] \to [\![book]\!] \to Prop \end{aligned}$$

$$\begin{aligned} [\![master]\!] &: [\![human]\!] \to \textsc{Info} \to Prop \\ &< [\![human]\!] \to \textsc{Phy} \bullet \textsc{Info} \to Prop \\ &< [\![human]\!] \to [\![book]\!] \to Prop \end{aligned}$$

Therefore, $[\![pickup]\!]$ *and* $[\![master]\!]$ *can both be used in a context where terms of type* $[\![human]\!] \to [\![book]\!] \to Prop$ *are required and the interpretation of the sentence (21) can proceed as intended.* □

5.2 Generative Lexical Entries as Σ-types

When lexical entries are represented as complex and structured entities, copredication should be able to be treated in a similar way. For the above example, the key is to make sure that, if we interpret the word 'book' as a structured entity, the subtyping relation $(*)$ still holds.

Our proposal is that the basic common nouns such as 'book' be represented as Σ-types in type theory (see §2.3 for a brief introduction to Σ-types and the relevant notations). For example, the lexical entry for 'book' (p.15 of [33]) may be interpreted as the following Σ-type:

$$[\![book]\!] = \left\{ \begin{array}{ll} Arg & : \textsc{Phy} \bullet \textsc{Info} \\ Qualia : \left\{ \begin{array}{ll} Formal & : Hold(p_1(Arg), p_2(Arg)) \\ Telic & : R(Arg) \\ Agent & : \exists h{:}Human.W(h, Arg) \end{array} \right\} & \end{array} \right\}$$

where p_1 and p_2 are the associated projection operators for the dot-type $\textsc{Phy} \bullet \textsc{Info}$ (see [21] for formal details) and $R(x)$ and $W(h, x)$ informally stand for 'x to be read' and 'h wrote x', respectively.

If 'book' is given the above structured interpretation, is it true that the subtyping relation $(*)$ still holds? (i.e., do we still have $[\![book]\!] \leq \textsc{Phy} \bullet \textsc{Info}$?) The answer is yes, because we can take the first projection π_1 for Σ-types as a coercion. Therefore, the formal calculations in the previous section goes through.

6 Implementations in Coq

Type theories have been implemented in several proof assistants such as Agda [1] and Coq [9]. We have used Coq to experiment with the proposal as reported in this paper. Coq supports the use of coercions. Although it is not completely satisfactory, it can be used well to implement the examples. (We do not get into the details here.) Some of the Coq codes are given in Appendix B.

– In Appendix B.1, we give the Coq codes that implement the examples of homonyms as described in §3. Please note that, since Coq has certain restrictions on what coercions can be defined (esp. concerning function types), we have to use type-casting in order to make some coercion insertions work.
– In Appendix B.2, we give the Coq codes that interpret 'book' as a Σ-type (implemented in Coq as the 'record' mechanism), as in §5.2, and implement examples of copredication etc; in particular, it shows how the following sentences can be interpreted as intended:

(22) John burned a boring book.

(23) John picked up and mastered book B.

7 Conclusion

Lexical semantics must consider how to give appropriate meaning explanations to the words which have different meanings in different contexts. We have studied how this may be done in various situations in the type-theoretical semantics based on modern type theories. The proposals can all be implemented on the computer based on the automated selection mechanisms with the help of coercive subtyping, among others. It would be interesting to see how such a system can be built based on the type-theoretical semantics.

The study of meta-theory of coercion contexts and local coercions is beyond the scope of the current paper. Although we do not foresee key difficulties, a challenge may be concerned with the notion of 'coherent context' as mentioned in Remark 2 in §4.1. The meta-theoretic properties such as strong normalisation do not just concern valid contexts, but coherent contexts and this new phenomena may need new tools to be dealt with satisfactorily.

In this paper, we have used Σ-types to represent structured lexical entries of CNs. However, other words such as verbs should not be represented as types. In general, the structured semantics of a word may be represented as a pair

$$(A, \phi), \text{ where } A : \mathit{Type} \text{ and } \phi : A \to \mathit{Prop}.$$

Such constructions have been studied in the contexts of mathematical theories [16] and program specifications [17]. Similar constructions seem to be needed in the current context and further studies are called for in this respect.

Among related work, we should mention that by Francez et al [13], where the proof-theoretic semantics (cf., [15]) of a fragment of natural language has been studied. Although it is quite different from type-theoretic semantics, there is one thing in common: proof-theoretic ideas play a central role in such semantics, in contrast to the more dominant model-theoretic approaches, including the Montague semantics.

Finally, it may also be helpful to mention that, in natural language processing, there is a substantial amount of work on the word sense disambiguation (WSD) problem (see, for example, [28] for a recent survey) and to emphasise that this is quite different (e.g., in aims) from the work reported here.

References

1. The Agda proof assistant (2008), http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php?
2. Asher, N.: A type driven theory of predication with complex types. Fundamenta Infor. 84(2) (2008)
3. Asher, N.: Lexical Meaning in Context: A Web of Words, draft, CUP (2010)
4. Bailey, A.: The Machine-checked Literate Formalisation of Algebra in Type Theory. Ph.D. thesis, University of Manchester (1999)
5. Callaghan, P., Luo, Z.: An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning 27(1), 3–27 (2001)
6. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic 5(1) (1940)
7. Cooper, R.: Records and record types in semantic theory. J. Logic and Compututation 15(2) (2005)
8. Cooper, R.: Copredication, dynamic generalized quantification and lexical innovation by coercion. Proceedings of GL 2007, the Fourth International Workshop on Generative Approaches to the Lexicon (2007)
9. The Coq Development Team: The Coq Proof Assistant Reference Manual (Version 8.1), INRIA (2007)
10. Curry, H., Feys, R.: Combinatory Logic, vol. 1. North-Holland, Amsterdam (1958)
11. Feferman, S.: Predicativity. In: Shapiro, S. (ed.) The Oxford Handbook of Philosophy of Mathematics and Logic. Oxford Univ. Press, Oxford (2005)
12. Fodor, J.A., Lepore, E.: The emptiness of the lexicon: Reflections on james pustejovskys the generative lexicon. Linguistic Inquiry 29(2), 269–288 (1998)
13. Francez, N., Dyckhoff, R.: Proof-theoretic semantics for a natural language fragment. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10. LNCS, vol. 6149, pp. 56–71. Springer, Heidelberg (2010)
14. Howard, W.A.: The formulae-as-types notion of construction. In: Hindley, J., Seldin, J. (eds.) To H. B. Curry: Essays on Combinatory Logic. Academic Press, London (1980)
15. Kahle, R., Schroder-Heister, P. (eds.) Synthese, vol. 148(3) (2006)
16. Luo, Z.: A higher-order calculus and theory abstraction. Information and Computation 90(1) (1991)
17. Luo, Z.: Program specification and data refinement in type theory. Mathematical Structures in Computer Science 3(3) (1993)
18. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ. Press, Oxford (1994)
19. Luo, Z.: Coercive subtyping in type theory. In: van Dalen, D., Bezem, M. (eds.) CSL 1996. LNCS, vol. 1258. Springer, Heidelberg (1997)
20. Luo, Z.: Coercive subtyping. J. of Logic and Computation 9(1), 105–130 (1999)
21. Luo, Z.: Type-theoretical semantics with coercive subtyping. In: Semantics and Linguistic Theory 20 (SALT 20), Vancouver (2010)
22. Luo, Z., Callaghan, P.: Coercive subtyping and lexical semantics (extended abstract). In: Logical Aspects of Computational Linguistics (LACL 1998) (1998)
23. Luo, Z., Pollack, R.: LEGO Proof Development System: User's Manual. LFCS Report ECS-LFCS-92-211, Dept. of Computer Science, Univ. of Edinburgh (1992)
24. Marlet, R.: When the generative lexicon meets computational semantics. In: 4th Inter. Workshop on Generative Approaches to the Lexicon (2007)
25. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)

26. The Matita proof assistant (2008), http://matita.cs.unibo.it/
27. Montague, R.: Formal Philosophy. Yale University Press, New Haven (1974)
28. Navigli, R.: Word sense disambiguation: a survey. ACM Computing Surveys 41(2) (2009)
29. Nordström, B., Petersson, K., Smith, J.: Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press, Oxford (1990)
30. Nunberg, G.: Transfers of meaning. J. of Semantics 12(2) (1995)
31. Pustejovsky, J.: The Generative Lexicon. MIT, Cambridge (1995)
32. Pustejovsky, J.: Generativity and explanation in semantics: A reply to fodor and lepore. Linguistic Inquiry 29(2), 289–311 (1998)
33. Pustejovsky, J.: The semantics of lexical underspecification. Folia Linguistica (1998)
34. Ranta, A.: Type-Theoretical Grammar. Oxford University Press, Oxford (1994)
35. Saïbi, A.: Typing algorithm in type theory with inheritance. In: POPL 1997 (1997)
36. Strachey, C.: Fundamental concepts in programming languages. Higher-Order and Symbolic Computation 13(1-2) (2000), (Paper based on 1967 lectures)

A The Unit Type

A unit type is an inductive type that has only one object. It is one of the inductive types in Martin-Löf's type theory or UTT, whose formal details can be found in, for example, Chapter 9 of [18]. In this paper, we consider a class of unit types: a unit type $\mathbf{1}_w$ for each word w. These unit types can be introduced by means of the following rules

$$\frac{}{\mathbf{1}_w : Type} \quad \frac{}{w : \mathbf{1}_w} \quad \frac{C : (\mathbf{1}_w)Type \quad c : C(w) \quad z : \mathbf{1}_w}{\mathcal{E}_w(C, c, z) : C(z)}$$

together with the computation rule $\mathcal{E}_w(C, c, w) = c : C(w)$ stating that, when applied to the canonical object w, its elimination operator $\mathcal{E}_w$ computes to c. For Coq implementation, see Appendix B.1 for that of $\mathbf{1}_{run}$.

B Implementations in Coq

B.1 Homonymy in Coq

```
(* Lexical Semantics using Coq's records: simple homonymy *)

(* Categories of Sentences & CNs *)
Definition S := Prop.
Definition CN := Set.
Parameters Bank Institution Human Man : CN.
Parameter John : Man.
Axiom mh : Man->Human.  Coercion mh : Man >-> Human.
Axiom bi : Bank->Institution.  Coercion bi : Bank >-> Institution.

(* unit type for "run" *)
Inductive Onerun : Set := run.
Definition RSem1 := Human->S.
Definition RSem2 := Human->Institution->S.
Parameter run1 : RSem1.
```

```
Parameter run2 : RSem2.
Definition r1 (r:Onerun) : RSem1 := run1.  Coercion r1 : Onerun >-> RSem1.
Definition r2 (r:Onerun) : RSem2 := run2.  Coercion r2 : Onerun >-> RSem2.

(* John runs quickly *)
Parameter quickly : forall (A:CN), (A->S)->(A->S).
Definition john_runs_quickly := quickly Human (run:RSem1) John.
(* John runs a bank *)
Definition john_runs_a_bank := exists b:Bank, (run:RSem2) John b.
```

B.2 Copredications in Coq

```
(* Note: Coq's "record-types" is Sigma-types plus *)
(* auto-defns of projections & coercions *)

(* Categories of Sentences and CNs *)
Definition S := Prop.
Definition CN := Set.

Parameter Human Man : CN.
Axiom mh : Man->Human.
Coercion mh : Man >-> Human.

(* Phy dot Info *)
Parameter Phy Info : CN.
Record PhyInfo : CN := mkPhyInfo { phy :> Phy; info :> Info }.

(* Book as Sigma-type with PhyInfo & BookQualia *)
Parameter Hold : Phy->Info->Prop.
Parameter R : PhyInfo->Prop.
Parameter W : Human->PhyInfo->Prop.
Record BookQualia (A:PhyInfo) : Set :=
  mkBookQualia { Formal : Hold A A;
                 Telic  : R A;
                 Agent  : exists h:Human, W h A }.
Record Book : Set := mkBook { Arg :> PhyInfo; Qualia : BookQualia Arg }.

(* "John burned a boring book" *)
Parameter John : Man.
Parameter boring : Info->S.
Record BBook : CN := mkBBook { b :> Book; _ : boring b }.
Parameter burn : Human->Phy->S.
Definition John_burned_a_boring_book := exists b:BBook, (burn John b) : S.

(* copredication: "John picked up and mastered book B" *)
Parameter B : Book.
Parameter pickup : Human->Phy->S.
Parameter master : Human->Info->S.
Definition John_picked_up_and_mastered_book_B
:= and (pickup John B) (master John B).
```

Logic Programming of the Displacement Calculus

Glyn Morrill

Universitat Politècnica de Catalunya
morrill@lsi.upc.edu
http: //www-lsi.upc.edu/~morrill/

Abstract. The displacement calculus of Morrill, Valentín and Fadda (2011)[12] forms a foundation for type logical categorial grammar in which discontinuity is accommodated alongside continuity in a logic which is free of structural rules and which enjoys Cut-elimination, the subformula property, decidability, and the finite reading property. The calculus deploys a new kind of sequent calculus which we call hypersequent calculus in which types and configurations have not only external context but also internal context, in the case that they are discontinuous. In this paper we consider the logic programming of backward chaining hypersequent proof search for the displacement calculus. We show how focusing eliminates all spurious ambiguity in the fragment without antecedent tensors and we illustrate coding of the essential features of displacement. In this way we lay a basis for parsing/theorem proving for this calculus, which is being used and extended in a system CatLog currently under development.

Keywords: categorial grammar, discontinuity, focusing, parsing as deduction, sequent calculus.

1 Introduction

Morrill, Valentín and Fadda (2011)[12] exemplifies applications of the displacement calculus such as quantification, discontinuous idioms, cross-serial dependencies, gapping, parentheticals, and comparative subdeletion. In this paper we address logic programming of parsing/theorem proving for the calculus by backward chaining (hyper)sequent proof search. We see that all spurious ambiguity in the fragment without antecedent tensors is eliminated by *focusing* (Andreoli 1992[1]) — controlling the types to which successive rules of inference are allowed to apply. Logic programming of the focused proof search depends on predicates handling the discontinuity, i.e. internal contexts, of discontinuous types and configurations. In Section 2 we review the displacement calculus. In Section 3 we define the focusing strategy. In Section 4 we look at the logic programming. We conclude in Section 5. The appendix contains a cover grammar of the PTQ fragment and output generated by the Prolog implementation CatLog currently under development.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 175–189, 2011.

2 The Displacement Calculus

Let a *vocabulary* V be a set including a distinguished symbol 1 called the *separator*. We define the *sort* $\sigma(s)$ of a string s over the vocabulary as the number of separators it contains. We define the *sort domains* $L_i, i \in \mathcal{N}$, as follows:

$$L_i = \{s \in V^* |\ \sigma(s) = i\} \tag{1}$$

The concatenation of strings of sort i and j is a string of sort $i + j$, thus the operation $+$ of concatenation is of functionality $L_i, L_j \to L_{i+j}$. We consider also operations of first and last wrap. Where s is a string of sort at least one and t is a string, the *first wrap* $s \times_> t$ is the string that results from replacing the first (leftmost) separator in s by t; in the same way the *last wrap* $s \times_< t$ is the string that results from replacing the last (rightmost) separator in s by t. Clearly first and last wrap are of functionality $L_{i+1}, L_j \to L_{i+j}$.

In the displacement calculus we define connective families by residuation with respect to concatenation and first and last wrap. The types so-defined are sorted according to the functionality of these operations. Thus there are sorted types $\mathcal{F}_i, i \in \mathcal{N}$ as follows, where 0 is the empty string and $k \in \{>, <\}$.[1]

$$\begin{array}{rrll}
\mathcal{F}_j := \mathcal{F}_i \backslash \mathcal{F}_{i+j} & [A \backslash C] = \{s_2 |\ \forall s_1 \in [A], s_1 + s_2 \in [C]\} & \text{under} \\
\mathcal{F}_i := \mathcal{F}_{i+j} / \mathcal{F}_j & [C / B] = \{s_1 |\ \forall s_2 \in [B], s_1 + s_2 \in [C]\} & \text{over} \\
\mathcal{F}_{i+j} := \mathcal{F}_i \cdot \mathcal{F}_j & [A \cdot B] = \{s_1 + s_2 |\ s_1 \in [A]\ \&\ s_2 \in [B]\} & \text{product} \\
\mathcal{F}_0 := I & [I] = \{0\} & \text{product unit} \\
\mathcal{F}_j := \mathcal{F}_{i+1} \downarrow_k \mathcal{F}_{i+j} & [A \downarrow_k C] = \{s_2 |\ \forall s_1 \in [A], s_1 \times_k s_2 \in [C]\} & \text{infix} \\
\mathcal{F}_{i+1} := \mathcal{F}_{i+j} \uparrow_k \mathcal{F}_j & [C \uparrow_k B] = \{s_1 |\ \forall s_2 \in [B], s_1 \times_k s_2 \in [C]\} & \text{circumfix} \\
\mathcal{F}_{i+j} := \mathcal{F}_{i+1} \odot_k \mathcal{F}_j & [A \odot_k B] = \{s_1 \times_k s_2 |\ s_1 \in [A]\ \&\ s_2 \in [B]\} & \text{wrap} \\
\mathcal{F}_1 := J & [J] = \{1\} & \text{wrap unit}
\end{array} \tag{2}$$

Where A is a type, sA denotes its sort.

Sequent calculus for displacement, which we call hypersequent calculus, involves some novelties. The set $\mathcal{O}$ of *configurations* is defined as follows, where Λ is the metalinguistic empty string (product unit) and $[\,]$ is the metalinguistic separator (wrap unit):

$$\mathcal{O} ::= \Lambda \mid [\,] \mid \mathcal{F}_0 \mid \mathcal{F}_{i+1}\{\underbrace{\mathcal{O} : \ldots : \mathcal{O}}_{i+1\ \mathcal{O}\text{'s}}\} \mid \mathcal{O}, \mathcal{O} \tag{3}$$

The fourth clause refers to a discontinuous type of sort $i + 1$ intercalated by $i + 1$ configurations corresponding to the positions of its separators. The sort of a configuration is the number of (metalinguistic) separators it contains. Where Δ is a configuration of sort at least one and Γ is a configuration, the metalinguistic first wrap $\Delta|_>\Gamma$ is the configuration that results from replacing the first (leftmost) metalinguistic separator in Δ by Γ; in the same way, the metalinguistic last wrap $\Delta|_<\Gamma$ is the configuration that results from replacing the last (rightmost) metalinguistic separator in Δ by Γ.

[1] We may abbreviate $\{\downarrow_>, \odot_>, \uparrow_>\}$ as $\{\downarrow, \odot, \uparrow\}$.

Where A is a type, its *vector* $\overrightarrow{A}$ is defined by:

$$\overrightarrow{A} = \begin{cases} A & \text{if } sA = 0 \\ A\{\underbrace{[\,] : \ldots : [\,]}_{sA\ [\,]\text{'s}}\} & \text{if } sA > 0 \end{cases} \tag{4}$$

Where Δ is a configuration of sort i and $\Gamma_1, \ldots, \Gamma_i$ are configurations, the *fold* $\Delta \otimes \langle \Gamma_1, \ldots, \Gamma_i \rangle$ is the configuration that results from replacing the metalinguistic separators in Δ by $\Gamma_1, \ldots, \Gamma_i$ respectively left-to-right. The standard distinguished occurrence notation $\Delta(\Gamma)$ signifies an occurrence of subconfiguration Γ with context Δ. This context is external. In displacement calculus a distinguished occurrence can be discontinuous, i.e. have internal contexts also. We notate a distinguished potentially discontinuous *hyperoccurrence* of a subconfiguation Γ of sort i with external and internal context Δ as $\Delta\langle\Gamma\rangle$ which stands for $\Delta_0(\Gamma \otimes \langle \Delta_1, \ldots, \Delta_i \rangle)$ where Δ_0 is the external context and $\Delta_1, \ldots, \Delta_n$ are the internal contexts.

A (hyper)sequent $\Gamma \Rightarrow A$ comprises an antecedent configuration Γ and a succedent type A of the same sort. The (hyper)sequent calculus for displacement is as shown in Figure 1, where $k \in \{>, <\}$.

3 Focusing

For the Lambek calculus (Lambek 1958 [8]) the property of Cut-elimination, that every theorem has a Cut-free proof, establishes decidability because the Cut-free sequent search space is finite, but within this space there are still multiple equivalent proofs differing in inessential rule orderings, resulting in 'spurious' ambiguity and computational inefficiency. For example, a complex type identity axiom instance can alternate with a proof involving succesive left and right inferences:

$$N\backslash S \Rightarrow N\backslash S \quad vs. \quad \dfrac{\dfrac{N \Rightarrow N \qquad S \Rightarrow S}{N, N\backslash S \Rightarrow S}\backslash L}{N\backslash S \Rightarrow N\backslash S}\backslash R \tag{5}$$

Left and right inferences can sometimes permute, for example:

$$\dfrac{N \Rightarrow N \qquad \dfrac{N, N\backslash S \Rightarrow S}{N\backslash S \Rightarrow N\backslash S}\backslash R}{(N\backslash S)/N, N \Rightarrow N\backslash S}/L \quad vs. \quad \dfrac{\dfrac{N \Rightarrow N \qquad N, N\backslash S \Rightarrow S}{N, (N\backslash S)/N, N \Rightarrow S}/L}{(N\backslash S)/N, N \Rightarrow N\backslash S}\backslash R \tag{6}$$

And two left inferences can sometimes permute, for example:

$$\begin{array}{l} CN \Rightarrow CN \quad \dfrac{N \Rightarrow N \qquad S \Rightarrow S}{N, N\backslash S \Rightarrow S}\backslash L \\ (\quad N/CN, CN, N\backslash S \Rightarrow S \end{array} \quad \text{vs.} \quad \begin{array}{l} \dfrac{CN \Rightarrow CN \qquad N \Rightarrow N}{N/CN, CN \Rightarrow N}/L \qquad S \Rightarrow S \\ (\quad N/CN, CN, N\backslash S \Rightarrow S \end{array} \tag{7}$$

$$\frac{}{\overrightarrow{A} \Rightarrow A}\,id \qquad \frac{\Gamma \Rightarrow A \qquad \Delta\langle \overrightarrow{A} \rangle \Rightarrow B}{\Delta\langle \Gamma \rangle \Rightarrow B}\,Cut$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \Gamma, \overrightarrow{A \backslash C} \rangle \Rightarrow D}\,\backslash L \qquad \frac{\overrightarrow{A}, \Gamma \Rightarrow C}{\Gamma \Rightarrow A \backslash C}\,\backslash R$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \overrightarrow{C / B}, \Gamma \rangle \Rightarrow D}\,/L \qquad \frac{\Gamma, \overrightarrow{B} \Rightarrow C}{\Gamma \Rightarrow C / B}\,/R$$

$$\frac{\Delta\langle \overrightarrow{A}, \overrightarrow{B} \rangle \Rightarrow D}{\Delta\langle \overrightarrow{A \cdot B} \rangle \Rightarrow D}\,\cdot L \qquad \frac{\Gamma_1 \Rightarrow A \qquad \Gamma_2 \Rightarrow B}{\Gamma_1, \Gamma_2 \Rightarrow A \cdot B}\,\cdot R$$

$$\frac{\Delta\langle \Lambda \rangle \Rightarrow A}{\Delta\langle \overrightarrow{I} \rangle \Rightarrow A}\,IL \qquad \frac{}{\Lambda \Rightarrow I}\,IR$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \Gamma|_k \overrightarrow{A \downarrow_k C} \rangle \Rightarrow D}\,\downarrow_k L \qquad \frac{\overrightarrow{A}|_k \Gamma \Rightarrow C}{\Gamma \Rightarrow A \downarrow_k C}\,\downarrow_k R$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \overrightarrow{C \uparrow_k B}|_k \Gamma \rangle \Rightarrow D}\,\uparrow_k L \qquad \frac{\Gamma|_k \overrightarrow{B} \Rightarrow C}{\Gamma \Rightarrow C \uparrow_k B}\,\uparrow_k R$$

$$\frac{\Delta\langle \overrightarrow{A}|_k \overrightarrow{B} \rangle \Rightarrow D}{\Delta\langle \overrightarrow{A \odot_k B} \rangle \Rightarrow D}\,\odot_k L \qquad \frac{\Gamma_1 \Rightarrow A \qquad \Gamma_2 \Rightarrow B}{\Gamma_1|_k \Gamma_2 \Rightarrow A \odot_k B}\,\odot_k R$$

$$\frac{\Delta\langle [\,] \rangle \Rightarrow A}{\Delta\langle \overrightarrow{J} \rangle \Rightarrow A}\,JL \qquad \frac{}{[\,] \Rightarrow J}\,JR$$

Fig. 1. (Hyper)sequent calculus for displacement

To deal with this König (1989)[7], Hepple (1990, 1990)[5] [4] and Hendriks (1993)[3] developed normalization for Lambek calculus theorem-proving, in an approach equivalent to focusing in linear logic where inferences are organized into alternate phases for synchronous and asynchronous connectives. Consider the fragment of the Lambek calculus with only antecedent and succedent divisions and succedent product. Focusing normalization comprises the following strategy:

- The identity axiom id is restricted to atomic types.
- In trying to prove a sequent, right rules are applied first, until the succedent type is atomic.
- Once the succedent type is atomic, an antecedent type is chosen as the *focus*. Successive left division rules will be applied to this focused type, with its value subtype becoming the focus of the major premise in each application. (8)

This focusing technique eliminates all spurious ambiguity in the fragment (i.e. without antecedent products) and is complete (Hepple 1990[5], 1990[4] appendix for the divisions; Hendriks 1993[3]). The focusing strategy for this fragment of Lambek calculus can be represented by marking types with boxes as shown in Figure 2; a sequent $\Gamma \Rightarrow A$ is proved by proving $\Gamma \Rightarrow \boxed{A}$.[2]

$$\frac{}{\boxed{P} \Rightarrow P}\, id \qquad \frac{\Gamma(\boxed{A}) \Rightarrow P}{\Gamma(A) \Rightarrow \boxed{P}}\, F$$

$$\frac{\Gamma \Rightarrow \boxed{A} \qquad \Delta(\boxed{C}) \Rightarrow D}{\Delta(\Gamma, \boxed{A\backslash C}) \Rightarrow D}\, \backslash L \qquad \frac{A, \Gamma \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{A\backslash C}}\, \backslash R$$

$$\frac{\Gamma \Rightarrow \boxed{B} \qquad \Delta(\boxed{C}) \Rightarrow D}{\Delta(\boxed{C/B}, \Gamma) \Rightarrow D}\, /L \qquad \frac{\Gamma, B \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{C/B}}\, /R$$

$$\frac{\Gamma \Rightarrow \boxed{A} \qquad \Delta \Rightarrow \boxed{B}}{\Gamma, \Delta \Rightarrow \boxed{A{\cdot}B}}\, {\cdot}R$$

Fig. 2. Focusing for Lambek calculus

The hypersequent calculus for displacement shares the shape of the Lambek calculus in respect of residuation. Distinguished occurrences become distinguished hyperoccurrences and antecedent active types become vectors, but otherwise the rules for the continuous connectives of displacement calculus have exactly the same form as those of the Lambek calculus. And the rules for the discontinuous connectives in displacement calculus also have this form with the metalinguistic comma representing concatenation on which the continuous connectives hinge replaced by the defined metalinguistic operations of wrap on which the discontinuous connectives hinge. In all cases the pattern of residuated triples and object language/metalanguage interaction is the same in the respects that effect derivational equivalence. The main contribution of the present paper is the observation that, consequentially, the same focusing strategy that serves for Lambek calculus also serves for displacement calculus. For the fragment without antecedent tensors, focusing for the displacement calculus can be represented as shown in Figure 3.

[2] Note that a necessary condition for a focusing choice to lead to a proof is that the eventual value of the focus be the atomic succedent, because the last major premise in the chain of left division inferences must be an identity axiom. Hence focusing choices (rule F) not satisfying this condition can be discounted.

$$\frac{}{\boxed{\overrightarrow{P}} \Rightarrow P}\,id \qquad \frac{\Gamma\langle \boxed{\overrightarrow{A}} \rangle \Rightarrow P}{\Gamma\langle A \rangle \Rightarrow \boxed{\mathrm{P}}}\,F$$

$$\frac{\Gamma \Rightarrow \boxed{A} \qquad \Delta\langle \boxed{\overrightarrow{C}} \rangle \Rightarrow D}{\Delta\langle \Gamma, \boxed{\overrightarrow{A \backslash C}} \rangle \Rightarrow D}\backslash L \qquad \frac{\overrightarrow{A}, \Gamma \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{A \backslash C}}\backslash R$$

$$\frac{\Gamma \Rightarrow \boxed{B} \qquad \Delta\langle \boxed{\overrightarrow{C}} \rangle \Rightarrow D}{\Delta\langle \boxed{\overrightarrow{C / B}}, \Gamma \rangle \Rightarrow D}/L \qquad \frac{\Gamma, \overrightarrow{B} \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{C / B}}/R$$

$$\frac{\Gamma_1 \Rightarrow \boxed{A} \qquad \Gamma_2 \Rightarrow \boxed{B}}{\Gamma_1, \Gamma_2 \Rightarrow \boxed{A \cdot B}} \cdot R$$

$$\frac{\Gamma \Rightarrow \boxed{A} \qquad \Delta\langle \boxed{\overrightarrow{C}} \rangle \Rightarrow D}{\Delta\langle \Gamma |_k \boxed{\overrightarrow{A \downarrow_k C}} \rangle \Rightarrow D}\downarrow_k L \qquad \frac{\overrightarrow{A} |_k \Gamma \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{A \downarrow_k C}}\downarrow_k R$$

$$\frac{\Gamma \Rightarrow \boxed{B} \qquad \Delta\langle \boxed{\overrightarrow{C}} \rangle \Rightarrow D}{\Delta\langle \boxed{\overrightarrow{C \uparrow_k B}} |_k \Gamma \rangle \Rightarrow D}\uparrow_k L \qquad \frac{\Gamma |_k \overrightarrow{B} \Rightarrow \boxed{C}}{\Gamma \Rightarrow \boxed{C \uparrow_k B}}\uparrow_k R$$

$$\frac{\Gamma_1 \Rightarrow \boxed{A} \qquad \Gamma_2 \Rightarrow \boxed{B}}{\Gamma_1 |_k \Gamma_2 \Rightarrow \boxed{A \odot_k B}} \odot_k R$$

Fig. 3. Focusing for displacement calculus

4 Logic Programming

Logic programming of Lambek calculus sequent proof search is straightforward (Moortgat 1988[9] appendix). Sequent rules are coded as Horn clauses and configurations as lists of types, as illustrated in the following for /L, where for brevity we use pattern matching for concatenation $\oplus$:

$$\frac{\Gamma \Rightarrow A \qquad \Delta(C) \Rightarrow D}{\Delta(C/B, \Gamma) \Rightarrow D}/L \tag{9}$$

$$p(\Delta_1 \oplus [C/B|\Gamma] \oplus \Delta_2, D) \leftarrow \\ p(\Gamma, B), \\ p(\Delta_1 \oplus [C] \oplus \Delta_2, D). \tag{10}$$

Since complex instances of the identity axiom can always be decomposed, it is appropriate to restrict the identity axiom to atomic types:

$$\frac{}{P \Rightarrow P}\, id \tag{11}$$

$$p([P], P) \leftarrow \\ atomic(P). \tag{12}$$

The programming can be refined to implement the focusing technique by defining for a goal sequent a first phase in which right rules are applied, followed by choice of a focus and a second phase in which a left (division) rule may be applied to the focused type.[3] Such an application generates a subgoal at the first phase in the minor premise and a subgoal at the second phase in the major premise.[4]

Because the displacement calculus shares the pattern of the Lambek calculus, the general logic programming considerations carry over. However, in addition the matching of hyperoccurrences, with internal as well as external contexts, must be handled. Types in displacement configurations are either of sort zero A_0 or of sort at least one $A_{i+1}\{\Gamma_1 : \ldots : \Gamma_{i+1}\}$ and a type occurrence may be encoded as a term $l(A, Ls)$ where A is the type and Ls is a list of its internal context configurations (empty if $sA = 0$). If the type is focused we represent this by encoding it $f(A, Ls)$. A displacement configuration is a list of such terms and separators 1. Matching hypersequents against the displacement rules depends on the fold operation in terms of which hyperoccurrences are defined. This is coded as the following predicate $fold(+Sin, ?Sout, -\Gamma, -\Delta sin, ?\Delta sout, +\Gamma_1)$ which means that configuration Γ is of the sort given by the unary notation difference list $Sin - Sout$ and Γ_1 is the result of replacing in order the separators of Γ by the configurations given by the difference list $\Delta sin - \Delta sout$:

[3] As remarked in fn. 1, since in the Lambek calculus a focusing choice can only lead to a proof if the eventual value of the focused type is the atomic succedent of the goal sequent, focusing choices can be restricted to such cases without loss of completeness. This makes for considerable savings in the search space, because otherwise left division rules must test all factorizations of unprovable sequents. The same observation applies for the displacement calculus.

[4] A necessary property for a Lambek sequent to be provable is that it have the same number of positive (succedent) and negative (antecedent) type occurrences of each atomic type (the van Benthem count invariant; van Benthem 1991 [15]). This is so because positive and negative occurrences of the same atom must be matched in identity axioms, and because the rules are multiplicative in the sense of linear logic. Thus, the proof search for a sequent may be pruned if the sequent does not satisfy this invariant. Again this makes for considerable savings in the search space because otherwise left division rules must test all factorizations of unprovable sequents. The same observation applies for the displacement calculus.

$$
\begin{array}{l}
fold(S, S, [], \Delta s, \Delta s, []).\\[2ex]
fold([1|S], Sout, [1|\Gamma], [\Delta|\Delta s], \Delta sout, \Delta \oplus \Gamma_1) \leftarrow\\
\qquad fold(S, Sout, \Gamma, \Delta s, \Delta sout, \Gamma_1).\\[2ex]
fold(Sin, Sout, [l(A, Ls)|\Gamma], \Delta sin, \Delta sout, [l(A, Ls1)|\Gamma_1]) \leftarrow\\
\qquad foldlst(Sin, S, Ls, \Delta sin, \Delta s, Ls1),\\
\qquad fold(S, Sout, \Gamma, \Delta s, \Delta sout, \Gamma_1).\\[2ex]
foldlst(S, S, [], \Delta s, \Delta s, []).\\[2ex]
foldlst(Sin, Sout, [L|Ls], \Delta sin, \Delta sout, [L1|Ls1]) \leftarrow\\
\qquad fold(Sin, S, L, \Delta sin, \Delta s, L1),\\
\qquad foldlst(S, Sout, Ls, \Delta s, \Delta sout, Ls1).
\end{array}
\tag{13}
$$

To represent the logic programming of displacement rules we give successive notational transformations starting with the hypersequent calculus metalanguage and ending with Prolog. For /L this is as shown in Figure 4. Here

$$\frac{\Gamma \Rightarrow B \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \overrightarrow{C/B}, \Gamma \rangle \Rightarrow D}\, /L$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta_0(\overrightarrow{C} \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}{\Delta_0((\overrightarrow{C/B}, \Gamma) \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\, /L$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta_0(\overrightarrow{C} \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}{\Delta_0(\overrightarrow{C/B} \otimes \langle \Delta_1 : \ldots : \Delta_{sC-sB} \rangle, \Gamma \otimes \langle \Delta_{1+sC-sB}, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\, /L$$

$$\frac{\Gamma \Rightarrow B \qquad \Delta_0(C\{\Delta_1 : \ldots : \Delta_{sC}\}) \Rightarrow D}{\Delta_0(C/B\{\Delta_1 : \ldots : \Delta_{sC-sB}\}, \Gamma \otimes \langle \Delta_{1+sC-sB}, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\, /L$$

$$
\begin{array}{l}
pleft([f(C/B, Ls1)] \oplus \Gamma', [f(C, Ls1 \oplus Ls2)]) \leftarrow\\
\qquad ssort(B, SB),\\
\qquad fold(SB, [], \Gamma, Ls2, [], \Gamma'),\\
\qquad p(\Gamma, B).
\end{array}
$$

```
pleft([f(C/B, Ls1)|Gamma1], [f(C, Ls)]) :-
      ssort(B, SB),
      fold(SB, [], Gamma, Ls2, [], Gamma1),
      p(Gamma, B),
      append(Ls1, Ls2, Ls).
```

Fig. 4. Programming of /L

$$\frac{\Gamma, \overrightarrow{B} \Rightarrow C}{\Gamma \Rightarrow C/B}\,/R$$

$$\frac{\Gamma, B\{\underbrace{[] : \ldots : []}_{sB\ [\,]\text{'s}}\} \Rightarrow C}{\Gamma \Rightarrow C/B}\,/R$$

$$\begin{array}{l} p(\Gamma, C/B) \leftarrow \\ \quad vector(B, Bvec), \\ \quad p(\Gamma \oplus [Bvec], C). \end{array}$$

```
p(Gamma, C/B) :-
      vector(B, Bvec),
      append(Gamma, [Bvec], Gamma1),
      p(Gamma1, C).
```

Fig. 5. Programming of /R

$$\frac{\Gamma \Rightarrow A \qquad \Delta\langle \overrightarrow{C} \rangle \Rightarrow D}{\Delta\langle \Gamma|_> \overrightarrow{A\downarrow_> C} \rangle \Rightarrow D}\downarrow_> L$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta_0(\overrightarrow{C} \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}{\Delta_0((\Gamma|_> \overrightarrow{A\downarrow_> C}) \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\downarrow_> L$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta_0(\overrightarrow{C} \otimes \langle \Delta_1, \ldots, \Delta_{sC} \rangle) \Rightarrow D}{\Delta_0(\Gamma \otimes \langle \overrightarrow{A\downarrow_> C} \otimes \langle \Delta_1, \ldots, \Delta_{1+sC-sA} \rangle, \Delta_{2+sC-sA}, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\downarrow_> L$$

$$\frac{\Gamma \Rightarrow A \qquad \Delta_0(C\{\Delta_1, \ldots, \Delta_{sC}\}) \Rightarrow D}{\Delta_0(\Gamma \otimes \langle A\downarrow_> C\{\Delta_1, \ldots, \Delta_{1+sC-sA}\}, \Delta_{2+sC-sA}, \ldots, \Delta_{sC} \rangle) \Rightarrow D}\downarrow_> L$$

$$\begin{array}{l} pleft(\Gamma, [f(C, Ls1 \oplus Ls2)]) \leftarrow \\ \quad firstins(\Gamma_1, f(A\downarrow_> C, Ls1), \Gamma), \\ \quad ssort(A, SA), \\ \quad fold(SA, [], \Gamma_2, [[1]|Ls2], [], \Gamma_1), \\ \quad p(\Gamma_2, A). \end{array}$$

```
pleft(Gamma, [f(C, Ls1Ls2)]) :-
      firstins(Gamma1, f(A 'v<' C, Ls1), Gamma),
      ssort(A, SA),
      fold(SA, [], Gamma2, [[1]|Ls2], Gamma1),
      p(Gamma2, A),
      append(Ls1, Ls2, Ls1Ls2).
```

Fig. 6. Programming of $\downarrow_>$L

$$\frac{\overrightarrow{A}|_{>}\Gamma \Rightarrow C}{\Gamma \Rightarrow A{\downarrow_{>}}C}\downarrow_{>}R$$

$$\frac{A\{\underbrace{[]:[]:\ldots:[]}_{sA\ []\text{'s}}\}|_{>}\Gamma \Rightarrow C}{\Gamma \Rightarrow A{\downarrow_{>}}C}\downarrow_{>}R$$

$$\frac{A\{\Gamma:\underbrace{[]:\ldots:[]}_{sA-1\ []\text{'s}}\} \Rightarrow C}{\Gamma \Rightarrow A{\downarrow_{>}}C}\downarrow_{>}R$$

$$p(\Gamma, A{\downarrow_{>}}C) \leftarrow vector(A, l(A, [[1]|Ls])), p([l(A, [\Gamma|Ls])], C).$$

```
p(Gamma, A 'v<' C) :-
      vector(A, l(A, [[1]|Ls])),
      p([l(A, [Gamma|Ls])], C).
```

Fig. 7. Programming of $\downarrow_k$R

$pleft(+\Gamma, -\Gamma_1)$ means that a left rule replaces a subconfiguration[5] Γ in the conclusion by Γ_1 in the major premise. The transformation for /R is shown in Figure 5. Here $vector(+B, -Bvec)$ means that $Bvec$ is the vector of the type B. For $\downarrow_{>}L$ the programming transformations are as shown in Figure 6. Here $firstins(-\Gamma, +A, +\Gamma_1)$ means that Γ_1 is the result of replacing the first (leftmost) metalinguistic separator in Γ by type A. For $\downarrow_{>}$R the programming transformation is as shown in Figure 7. The other rules are implemented similarly on the basis of the same ideas.

5 Conclusion

The principles presented in this paper form the basis of a Prolog categorial parser/theorem prover CatLog under development.[6] CatLog deals with many connectives over and above the displacement connectives, but it is the displacement hypersequent calculus, which forms a new multiplicative basis for categorial logic, which forms its core. The inclusion in CatLog of standard Curry-Howard type logical categorial semantics is straightforward because the semantic reading

[5] An ordinary occurrence, not a hyperoccurrence.

[6] An earlier such program Catlog for generalized Lambek calculus based on sequent calculus normalization was developed in the first half of the 90s, but was not brought to term because of the absence of a satisfactory treatment of discontinuity. The introduction of the displacement calculus has allowed this line of investigation to resume.

of a derivation is a homomorphic image of the syntactic proof. This paper has addressed the main technical question for displacement calculus, and hence displacement categorial logic, of backward chaining sequent proof search in a logic programming parsing as deduction paradigm.

Acknowledgement. I thank three LACL'11 reviewers for valuable comments. The research reported in the present paper was supported by DGICYT project SESAAME-BAR (TIN2008-06582-C03-01).

References

1. Andreoli, J.M.: Logic programming with focusing in linear logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. Dowty, D.R., Wall, R.E., Peters, S.: Introduction to Montague Semantics. Synthese Language Library, vol. 11. D. Reidel, Dordrecht (1981)
3. Hendriks, H.: Studied flexibility. Categories and types in syntax and semantics. PhD thesis, Universiteit van Amsterdam, ILLC, Amsterdam (1993)
4. Hepple, M.: The Grammar and Processing of Order and Dependency. PhD thesis, University of Edinburgh (1990)
5. Hepple, M.: Normal form theorem proving for the Lambek calculus. In: Karlgren, H. (ed.) Proceedings of COLING, Stockholm (1990)
6. Jäger, G.: Anaphora and Type Logical Grammar. Trends in Logic – Studia Logica Library, vol. 24. Springer, Dordrecht (2005)
7. König, E.: Parsing as natural deduction. In: Proceedings of the Annual Meeting of the Association for Computational Linguistics, Vancouver (1989)
8. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958); reprinted in Buszkowski, W., Marciszewski, W., van Benthem, J. (eds.): Categorial Grammar. Linguistic & Literary Studies in Eastern Europe, vol. 25, pp. 153–172. John Benjamins, Amsterdam (1988)
9. Moortgat, M.: Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus. Foris, Dordrecht, PhD thesis, Universiteit van Amsterdam (1988)
10. Morrill, G.: Intensionality and Boundedness. Linguistics and Philosophy 13(6), 699–726 (1990)
11. Morrill, G., Valentín, O.: On Anaphora and the Binding Principles in Categorial Grammar. In: Dawar, A., de Queiroz, R. (eds.) WoLLIC 2010. LNCS, vol. 6188, pp. 176–190. Springer, Heidelberg (2010)
12. Morrill, G., Valentín, O., Fadda, M.: The Displacement Calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011), doi:10.1007/s10849-010-9129-2
13. Morrill, G.V.: Type Logical Grammar: Categorial Logic of Signs. Kluwer Academic Publishers, Dordrecht (1994)
14. Morrill, G.V.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press, Oxford (2010)
15. van Benthem, J.: Language in Action: Categories, Lambdas, and Dynamic Logic. Studies in Logic and the Foundations of Mathematics, vol. 130. North-Holland, Amsterdam (1991); revised student edition printed in 1995 by the MIT Press

Appendix

This appendix contains unedited LaTeX output of CatLog for a cover grammar of PTQ. A printout of the categorial lexicon is followed by the analyses computed for example sentences drawn from Dowty, Wall and Peters (1981, ch. 7)[2]. The categorial logic for the fragment extends displacement calculus with implicitly universally quantified first order term structure for features on atomic types (Morrill 1994, ch. 6[13]), the limited contraction for anaphora of Jäger 2005[6], a difference operator (cf. Morrill and Valentín 2010[11]) and modalities for intensionality (Morrill 1990[10], 1994[13] chs. 4 & 5, 2010[14] ch. 8) such that $\Box A$ is the type of expressions in intension of type A; $\blacksquare A$ represents such expressions in intension which are rigid designators.

a : $\blacksquare(((SA\uparrow\blacksquare NB)\downarrow SA)/CNB) : \lambda C\lambda D\exists E[(C\ E) \wedge (D\ E)]$
and : $\blacksquare((SA\backslash SA)/SA) : \lambda B\lambda C[C \wedge B]$
and : $\blacksquare(((NA\backslash SB)\backslash(NA\backslash SB))/(NA\backslash SB)) : \lambda C\lambda D\lambda E[(D\ E) \wedge (C\ E)]$
believes : $\Box((NA\backslash Sf)/CPthat) : believe$
bill : $\blacksquare Nm : b$
catch : $\Box((NA\backslash Sb)/NB) : catch$
doesnt : $\Box((NA\backslash s)/(NA\backslash s)) : \hat{}\lambda B\lambda C\neg(B\ C)$
eat : $\Box((NA\backslash Sb)/NB) : eat$
every : $\blacksquare(((SA\uparrow NB)\downarrow SA)/CNB) : \lambda C\lambda D\forall E[(C\ E) \rightarrow (D\ E)]$
finds : $\Box((NA\backslash Sf)/NB) : finds$
fish : $\Box CNn : fish$
he : $\blacksquare((\blacksquare SA|Nm)/(Nm\backslash SA)) : \lambda BB$
her : $\Box(\Box((s\uparrow Nf) - (J\bullet(Nf\backslash s)))\downarrow(\Box s|Nf)) : \lambda A\lambda B((A\ B)\ B)$
her : $\blacksquare(((((SA\uparrow Nf) - (J\bullet(Nf\backslash SA)))\uparrow\blacksquare Nf) - (J\bullet((Nf\backslash s)\uparrow Nf)))\downarrow_<(SA\uparrow\blacksquare Nf)) : \lambda B\lambda C((B\ C)\ C)$
in : $\blacksquare(((NA\backslash SB)\backslash(NA\backslash SB))/NC) : \lambda D\lambda E\lambda F((\check{}in\ D)\ (E\ F))$
is : $\blacksquare((NA\backslash Sf)/NB) : \lambda C\lambda D[D = C]$
it : $\blacksquare((SA\uparrow Nn)\downarrow(\blacksquare SA|Nn)) : \lambda BB$
it : $\blacksquare(((((SA\uparrow Nn) - (J\bullet(Nn\backslash SA)))\uparrow\blacksquare Nn) - (J\bullet((Nn\backslash SB)\uparrow Nn)))\downarrow_<(SA\uparrow\blacksquare Nn)) : \lambda C\lambda D((C\ D)\ D)$
john : $\blacksquare Nm : j$
loses : $\Box((NA\backslash Sf)/NB) : loses$
loves : $\Box((NA\backslash Sf)/NB) : loves$
man : $\Box CNm : man$
necessarily : $\blacksquare(SA/\Box SA) : nec$
or : $\blacksquare((SA\backslash SA)/SA) : \lambda B\lambda C[C \vee B]$
or : $\blacksquare(((NA\backslash Sf)\backslash(NA\backslash Sf))/(NA\backslash Sf)) : \lambda B\lambda C\lambda D[(C\ D) \vee (B\ D)]$
park : $\Box CNn : park$
seeks : $\blacksquare((NA\backslash Sf)/\Box(((NB\backslash SC)/ND)\backslash(NB\backslash SC))) : \lambda E\lambda F((\check{}tries\ \hat{}((\check{}E\ \check{}find)\ F))\ F)$
she : $\blacksquare((\blacksquare SA|Nf)/(Nf\backslash SA)) : \lambda BB$
slowly : $\Box(\Box(NA\backslash SB)\backslash(NA\backslash SB)) : slowly$
such+that : $\blacksquare((CNA\backslash CNA)/(Sf|NA)) : \lambda B\lambda C\lambda D[(C\ D) \wedge (B\ D)]$
talks : $\Box(NA\backslash Sf) : talk$
that : $\blacksquare(CPthat/\Box Sf) : \lambda AA$
the : $\blacksquare(NA/CNA) : \iota$
to : $\blacksquare((NA\backslash St)/(NA\backslash Sb)) : \lambda BB$
tries : $\blacksquare((NA\backslash Sf)/\Box(NA\backslash St)) : \lambda B\lambda C((\check{}tries\ \hat{}(\check{}B\ C))\ C)$
unicorn : $\Box CNn : unicorn$
walk : $\Box(NA\backslash Sb) : walk$
walks : $\Box(NA\backslash Sf) : walk$
woman : $\Box CNf : woman$

Example (7-16) involves a quantifier phrase in subject position. Here and always lexical types are modalized outermost because lexical meanings are intensions. Within its modality the type for the quantifier is a functor seeking a count noun to its right. (A feature variable transmits the gender value.) This yields a generalized quantifier type which will infix at the position of a nominal in a sentence, simulating term insertion S14. Example (7-32) involves subject

(7-16) **every+man+talks** : Sf

$\blacksquare(((SA{\uparrow}NB){\downarrow}SA)/CNB) : \lambda C\lambda D\forall E[(C\ E) \rightarrow (D\ E)], \Box CNm : man, \Box(NF\backslash Sf) : talk \Rightarrow Sf$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{CNm \Rightarrow CNm}}{\boxed{\Box CNm} \Rightarrow CNm}\Box L \quad \dfrac{\dfrac{\dfrac{\dfrac{}{Nm \Rightarrow Nm} \quad \dfrac{}{Sf \Rightarrow Sf}}{Nm, \boxed{Nm\backslash Sf} \Rightarrow Sf}\backslash L}{Nm, \boxed{\Box(Nm\backslash Sf)} \Rightarrow Sf}\Box L}{1, \Box(Nm\backslash Sf) \Rightarrow Sf{\uparrow}Nm}{\uparrow}R \quad \dfrac{}{Sf \Rightarrow Sf}}{\boxed{(Sf{\uparrow}Nm){\downarrow}Sf}, \Box(Nm\backslash Sf) \Rightarrow Sf}{\downarrow}L}{\boxed{((Sf{\uparrow}Nm){\downarrow}Sf)/CNm}, \Box CNm, \Box(Nm\backslash Sf) \Rightarrow Sf}/L}{\boxed{\blacksquare(((Sf{\uparrow}Nm){\downarrow}Sf)/CNm)}, \Box CNm, \Box(Nm\backslash Sf) \Rightarrow Sf}\blacksquare L$$

$\forall C[(\check{}man\ C) \rightarrow (\check{}talk\ C)]$

(7-32) **every+man+walks+or+talks** : Sf

$\blacksquare(((SA{\uparrow}NB){\downarrow}SA)/CNB) : \lambda C\lambda D\forall E[(C\ E) \rightarrow (D\ E)], \Box CNm : man, \Box(NF\backslash Sf) : walk, \blacksquare((SG\backslash SG)/SG) : \lambda H\lambda I[I \vee H], \Box(NJ\backslash Sf) : talk \Rightarrow Sf$

$\blacksquare(((SA{\uparrow}NB){\downarrow}SA)/CNB) : \lambda C\lambda D\forall E[(C\ E) \rightarrow (D\ E)], \Box CNm : man, \Box(NF\backslash Sf) : walk, \blacksquare(((NG\backslash Sf)\backslash(NG\backslash Sf))/(NG\backslash Sf)) : \lambda H\lambda I\lambda J[(I\ J) \vee (H\ J)], \Box(NK\backslash Sf) : talk \Rightarrow Sf$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{CNm \Rightarrow CNm}}{\boxed{\Box CNm} \Rightarrow CNm}\Box L \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Nm \Rightarrow Nm} \quad \dfrac{}{Sf \Rightarrow Sf}}{Nm, \boxed{Nm\backslash Sf} \Rightarrow Sf}\backslash L}{Nm, \boxed{\Box(Nm\backslash Sf)} \Rightarrow Sf}\Box L}{\Box(Nm\backslash Sf) \Rightarrow Nm\backslash Sf}\backslash R \quad \dfrac{\dfrac{\dfrac{\dfrac{}{Nm \Rightarrow Nm} \quad \dfrac{}{Sf \Rightarrow Sf}}{Nm, \boxed{Nm\backslash Sf} \Rightarrow Sf}\backslash L}{Nm, \Box(Nm\backslash Sf), \boxed{(Nm\backslash Sf)\backslash(Nm\backslash Sf)} \Rightarrow Sf}\backslash L}{Nm, \Box(Nm\backslash Sf), \boxed{((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)}, \Box(Nm\backslash Sf) \Rightarrow Sf}/L}{Nm, \Box(Nm\backslash Sf), \blacksquare(((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)), \Box(Nm\backslash Sf) \Rightarrow Sf}\blacksquare L}{1, \Box(Nm\backslash Sf), \blacksquare(((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)), \Box(Nm\backslash Sf) \Rightarrow Sf{\uparrow}Nm}{\uparrow}R \quad \dfrac{}{Sf \Rightarrow Sf}}{\boxed{(Sf{\uparrow}Nm){\downarrow}Sf}, \Box(Nm\backslash Sf), \blacksquare(((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)), \Box(Nm\backslash Sf) \Rightarrow Sf}{\downarrow}L}{\boxed{((Sf{\uparrow}Nm){\downarrow}Sf)/CNm}, \Box CNm, \Box(Nm\backslash Sf), \blacksquare(((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)), \Box(Nm\backslash Sf) \Rightarrow Sf}/L}{\boxed{\blacksquare(((Sf{\uparrow}Nm){\downarrow}Sf)/CNm)}, \Box CNm, \Box(Nm\backslash Sf), \blacksquare(((Nm\backslash Sf)\backslash(Nm\backslash Sf))/(Nm\backslash Sf)), \Box(Nm\backslash Sf) \Rightarrow Sf}\blacksquare L$$

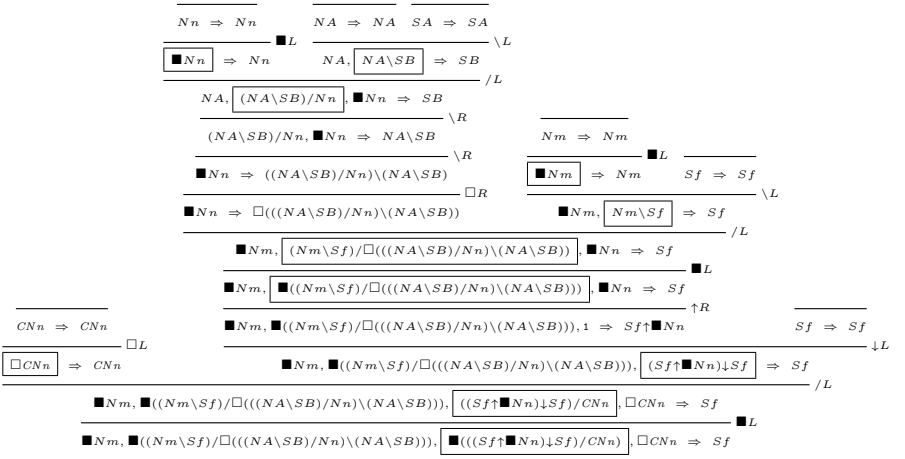
$\forall C[(\check{}man\ C) \rightarrow [(\check{}walk\ C) \vee (\check{}talk\ C)]]$

(7-60, 7-62) **john+seeks+a+unicorn** : Sf

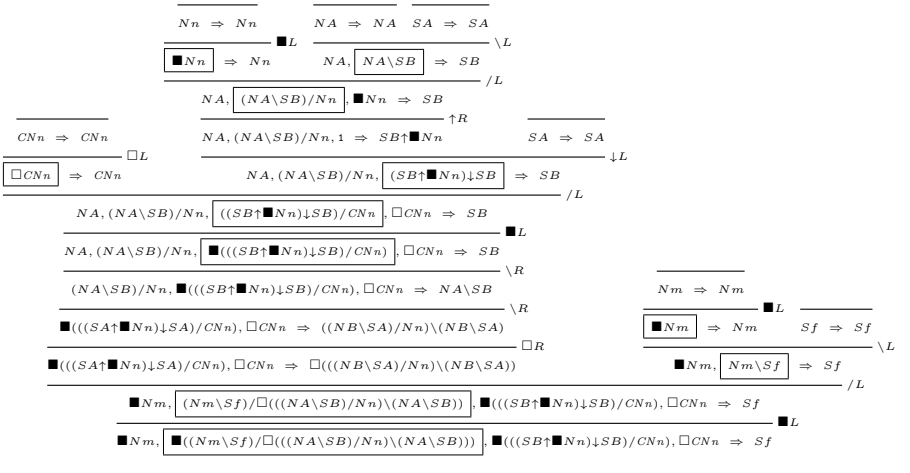
$\blacksquare Nm : j, \blacksquare((NA\backslash Sf)/\Box(((NB\backslash SC)/ND)\backslash(NB\backslash SC))) : \lambda E\lambda F((\check{}tries\ \hat{}((\check{}E\ \hat{}find)\ F))\ F), \blacksquare(((SG{\uparrow}\blacksquare NH){\downarrow}SG)/CNH) : \lambda I\lambda J\exists K[(I\ K) \wedge (J\ K)], \Box CNn : unicorn \Rightarrow Sf$

quantification and verb phrase coordination. Example (7-60, 7-62) involves the lifted intensional object transitive verb 'seek' which is synonymous with 'try to find'. The sentence has a specific reading in which there is a unicorn which John is trying to find, and a non-specific reading in which John is just trying to bring it about that he finds some, any, unicorn. The latter reading does not have existential commitment: it can be true without any unicorn actually existing. The two readings are obtained from derivations in which the indefinite object term-inserts outside or within the scope of the intensional verb. Example (7-76)

is the classic example of the interaction of the copula and indefinites in Montague grammar to assign *John is a man* a logical form equivalent to $(\check{}\,man\ j)$. Our grammar conserves this, with a lowered object type for the copula. This same lower extensional transitive verb type for *find* means that (7-98), by contrast with (7-60, 7-62), has only one reading, with existential commitment.



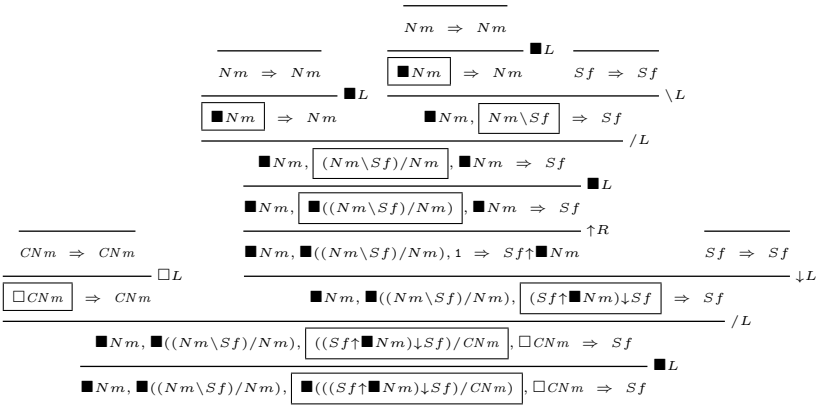
$\exists C[(\check{}\,unicorn\ C) \wedge ((\check{}\,tries\ \hat{}\,((\check{}\,find\ C)\ j))\ j)]$



$((\check{}\,tries\ \hat{}\,\exists G[(\check{}\,unicorn\ G) \wedge ((\check{}\,find\ G)\ j)])\ j)$

(7-76) **john+is+a+man** : Sf

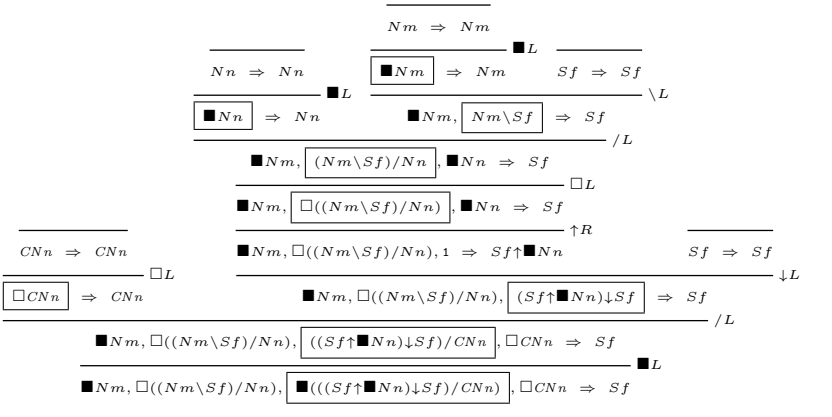
$\blacksquare Nm : j, \blacksquare((NA\backslash Sf)/NB) : \lambda C\lambda D[D = C], \blacksquare(((SE{\uparrow}\blacksquare NF){\downarrow}SE)/CNF) : \lambda G\lambda H\exists I[(G\ I) \wedge (H\ I)], \square CNm : man \Rightarrow Sf$



$\exists C[(\check{}\,man\ C) \wedge [j = C]]$

(7-98) **john+finds+a+unicorn** : Sf

$\blacksquare Nm : j, \square((NA\backslash Sf)/NB) : finds, \blacksquare(((SC\uparrow\blacksquare ND)\downarrow SC)/CND) : \lambda E\lambda F\exists G[(E\ G) \wedge (F\ G)], \square CNn : unicorn \Rightarrow Sf$



$\exists C[(\check{}\,unicorn\ C) \wedge ((\check{}\,finds\ C)\ j)]$

Space limitations prevent further illustration, but note for example that the type for the indefinite has a modal hypothetical subtype, which will allow an indefinite to take scope at a higher clause, but that the type for *every* has a nonmodal hypothetical subtype and will therefore only take scope in its local clause (Montague did not capture this distinction), and note that the accusative pronoun *her* has a type allowing it to take a preceding antecedent in its own clause or in an embedded clause by secondary wrap, but with use of the difference operator expressing the Principle B condition that the antecedent cannot be the subject (Montague did not capture this effect either).

Conditional Logic C_b and Its Tableau System

Yuri Ozaki and Daisuke Bekki

Ochanomizu University,
Graduate School of Humanities and Sciences
{ozaki.yuri,bekki}@is.ocha.ac.jp

Abstract. Conditional logic is a kind of modal logic for analyzing the truth conditions and inferences of conditional sentences in natural language. However, it has been pointed out in the literature that empirical problems plague all of the previously proposed conditional logics. Moreover, C_1 and C_2 are defined by imposing certain restrictions on their Kripke frames, and there exist no corresponding proof systems.

In order to solve these problems, we propose a new system of conditional logic, which we call C_b. C_b is an extension of C^+ through the addition of new rules on accessibility, and it has a corresponding tableau system. We show that C_b has empirical advantages over C_1 and C_2 as a model of inference in natural language, and compare it with other proof systems of conditional logic.

1 Introduction

1.1 Conditional Sentences in Natural Language and Classical Logic

The following inferences are valid in classical logic:

Antecedent strengthening: $A \supset B \vdash (A \wedge C) \supset B$
Transitivity: $A \supset B, B \supset C \vdash A \supset C$
Contraposition: $A \supset B \vdash \neg B \supset \neg A$

If we simply assume classical logic to explain the semantics of natural language, the formulae above give rise to the following infelicitous arguments [6].

(1) If it does not rain tomorrow we will go to the cricket. Hence, if it does not rain tomorrow and I am killed in a car accident tonight then we will go to the cricket.
(2) If the other candidates pull out, John will get the job. If John gets the job, the other candidates will be disappointed. Hence, if the other candidates pull out, they will be disappointed.
(3) If we take the car then it won't break down en route. Hence, if the car does break down en route, we didn't take it.

The reason for such infelicity is that obvious premises can be omitted in conditionals. For example, the first sentence in (1) introduces the condition "it does not rain" and as we usually do not think about the possibility that we may be killed in a car accident, we continue to assume that we will not to be killed in a car accident as an obvious premise. However, the second sentence of (1) has

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 190–204, 2011.

the opposite meaning in that the premise that is omitted as obvious itself leads to the invalid conclusion. In case (1), what we actually mean to say is:

(1') If it does not rain tomorrow and I am not killed in a car accident tonight, then we will go to the cricket tomorrow.

Similar comments can be made about the arguments in (2) and (3). In case (2), the actual wording should be "if John gets the job and the other candidates do not pull out, they will be disappointed." In case (3), the premise "we will take the car" is omitted and the sentence actually states the opposite, "we didn't take it", making case (3) incomprehensible. Thus, we see that we cannot correctly address conditionals of natural language with semantics based on classical logic.

Let us discuss case (1) in more detail. The correct sentence for (1) is "if it does not rain tomorrow then, other things being equal, we will go to the cricket". We can call "other things being equal" *ceteris paribus*. Conditional sentences include some notion of *ceteris paribus*, so G_A in the conditional sentence "if A and G_A, then B" can be referred to as a *ceteris paribus clause*, which depends on A. Therefore, in the example sentence, if A is "it does not rain tomorrow", then G_A includes the condition that we are not invaded by Martians. If A is "flying saucers arrive from Mars", it does not.

Thus, the notion of *ceteris paribus* is important in conditional sentences of natural language. Accordingly, some logic systems based on the notion of *ceteris paribus* have been proposed; however, they do not sufficiently represent truth conditions in conditional sentences. This paper, first, discusses a previously developed logic system for analyzing conditional sentences, and second, presents an extension of it as a new logic system.

2 Modal Tableau

Modal Tableau is a tableau system for modal logic. In this section, we briefly describe the version of modal tableau that we adopt from [6]. In modal tableaux, we put a natural number with each formula to designate a possible world in which the formula is assumed to be true.

$$\begin{array}{ccc} & A \vee B, i & \\ \swarrow & & \searrow \\ A, i & & B, i \end{array}$$

As the diagram above indicates, the tableau rules for truth functors are the same as those for classical logic except for the numbers for possible worlds. Four new rules are added for the modal operators.

$$\begin{array}{cccc} \neg\Box A, i & \neg\Diamond A, i & \Box A, i & \Diamond A, i \\ \downarrow & \downarrow & irj & \downarrow \\ \Diamond\neg A, i & \Box\neg A, i & \downarrow & irj \\ & & A, j & A, j \end{array}$$

In the rules above, r in irj represents the binary accessibility relation R between two worlds i, j in a Kripke frame $\langle W, R, \nu \rangle$ of modal logic. W is a

non-empty set and ν is a function that assigns a truth value to each formula, such that either $\nu_w(p) = 1$ or $\nu_w(p) = 0$. i and j are natural numbers, but j must be new and must not occur at any node above in the same branch.

The two rules on the right are deduced by the following interpretation in Kripke semantics for $\Box$ and $\Diamond$. For any world $w \in W$:

- $\nu_w(\Box A) = 1$ if, for all $w' \in W$ such that wRw', $\nu_{w'}(A) = 1$;
  $\nu_w(\Box A) = 0$ otherwise.
- $\nu_w(\Diamond A) = 1$ if, for some $w' \in W$ such that wRw', $\nu_{w'}(A) = 1$;
  $\nu_w(\Diamond A) = 0$ otherwise.

The two rules on the left can be explained by following proofs respectively.

In any world, w,

$$\begin{aligned}\nu_w(\neg\Box A) = 1 &\text{ iff } \nu_w(\Box A) = 0\\ &\text{iff it is not the case that, for all } w' \text{ such that } wRw',\ \nu_{w'}(A) = 1\\ &\text{iff for some } w' \text{ such that } wRw',\ \nu_{w'}(A) = 0\\ &\text{iff for some } w' \text{ such that } wRw',\ \nu_{w'}(\neg A) = 1\\ &\text{iff } \nu_w(\Diamond\neg A) = 1\end{aligned}$$

In any world, w,

$$\begin{aligned}\nu_w(\neg\Diamond A) = 1 &\text{ iff } \nu_w(\Diamond A) = 0\\ &\text{iff it is not the case that, for some } w' \text{ such that } wRw', \nu_{w'}(A) = 1\\ &\text{iff for all } w' \text{ such that } wRw',\ \nu_{w'}(A) = 0\\ &\text{iff for all } w' \text{ such that } wRw',\ \nu_{w'}(\neg A) = 1\\ &\text{iff } \nu_w(\Box\neg A) = 1\end{aligned}$$

The following is an example of a modal tableau for: $\vdash \Diamond(A \wedge B) \supset (\Diamond A \wedge \Diamond B)$

$$\begin{array}{c}
\neg(\Diamond(A \wedge B) \supset (\Diamond A \wedge \Diamond B)), 0\\
\downarrow\\
\Diamond(A \wedge B), 0\\
\neg(\Diamond A \wedge \Diamond B), 0\\
\swarrow \qquad\qquad \searrow\\
\begin{array}{c}\neg\Diamond A, 0\\ \downarrow\\ \Box\neg A, 0\\ \downarrow\\ 0r1\\ A \wedge B, 1\\ \downarrow\\ A, 1\\ B, 1\\ \downarrow\\ \neg A, 1\\ \times\end{array}
\qquad\qquad
\begin{array}{c}\neg\Diamond B, 0\\ \downarrow\\ \Box\neg B, 0\\ \downarrow\\ 0r1\\ A \wedge B, 1\\ \downarrow\\ A, 1\\ B, 1\\ \downarrow\\ \neg B, 1\\ \times\end{array}
\end{array}$$

As a proof of a theorem (rather than of deduction), the initial list has only one formula, $\neg(\Diamond(A \wedge B) \supset (\Diamond A \wedge \Diamond B)), 0$. The natural numbers for possible worlds start at 0. When the rule $\Diamond(A \wedge B), 0$ is applied, the new formula $A \wedge B$ is introduced for both branches and a new world number 1 is assigned to it. It causes no problem that the same world number is used in both branches as they do not interact with each other. When we judge whether a given branch closes or not in modal tableau, we must compare contradictory formulae which are assigned the same world number, as for $A, 1$ and $\neg A, 1$ in the example above.

3 Conditional Logic

In section 1.1, we explained the concept of *ceteris paribus* in the context of conditional sentences in natural language. The logic equipped with this concept is called *conditional logic* [3,4,8], which is a kind of modal logic held to be useful as a semantic framework of natural language.

3.1 Syntax of Conditional Logic

Let us write $A > B$ for a conditional with a *ceteris paribus* condition: "if A, then B". The syntax of conditional logic is defined by the following BNF grammar, where p is a propositional parameter:

$$\mathcal{F} ::= p \mid \neg\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \mathcal{F} \rightarrow \mathcal{F} \mid \mathcal{F} \leftrightarrow \mathcal{F} \mid \Box\mathcal{F} \mid \Diamond\mathcal{F} \mid \mathcal{F} > \mathcal{F}$$

3.2 Semantics of Conditional Logic

The Kripke frame of conditional logic consists of a quadruple: $\langle W, \{R_A : A \in \mathcal{F}\}, R, \nu\rangle$, where W is a non-empty set, ν is a function that assigns a truth value to each pair comprising a world, w, and a propositional parameter, p, the same as for modal logic. "In world w, p is true (or false)" is written as $\nu_w(p) = 1$ (or $\nu_w(p) = 0$). R is a binary relation on W which is reflexive, symmetrical and transitive. Each R_A is a binary relation on W for any formula A. Intuitively, $w_1 R_A w_2$ly means that w_2 is the same as w_1 except that A is true in w_2, which represents the *ceteris paribus* condition.

In the settings of conditional logic, $\Box$ and $\Diamond$ are treated as those of system K_ν [6].

- $\nu_w(\Box A) = 1$ if, for all $w' \in W$ such that wRw', $\nu_{w'}(A) = 1$; and 0 otherwise.
- For any world $w \in W$
  $\nu_w(\Diamond A) = 1$ if, for some $w' \in W$ such that wRw', $\nu_{w'}(A) = 1$; and 0 otherwise.

By means of the notions introduced above, the semantics of $A > B$ is defined as follows.

- $\nu_w(A > B) = 1$ iff for all w' such that $wR_A w'$, $\nu_{w'}(B) = 1$ and $\nu_{w'}(B) = 0$ otherwise.

Furthermore, in conditional logic, the following conception, $f_A(w)$, $[A]$, is added.

- $f_A(w) = \{x \in W : wR_Ax\}$
- $[A] = \{w : \nu_w(A) = 1\}$

Here, $f_A(w)$ is the set of worlds accessible to w under R_A. Also, R and $f_A(w)$ are interdefinable, since wR_Aw' iff $w' \in f_A(w)$. $[A]$ is a class of worlds where A is true, $\{w : \nu_w(A) = 1\}$.

With this conception, the semantics of $A > B$ can be simply defined as follows:

- $A > B$ is true in $w \Leftrightarrow f_A(w) \subseteq [B]$

4 Previous Study

It would seem that conditional logic has a close connection with the phenomenon of natural language, but there is as yet no logical system that can represent it. In this section, we briefly explain some logical systems that extend the conditional logic set out in section 3.

4.1 C^+

C^+ [1] is logic system which is an extension of conditional logic C by adding the following conditions on its Kripke frame.

1. $f_A(w) \subseteq [A]$
2. If $w \in [A]$, then $w \in f_A(w)$

C^+ has a tableau system that corresponds to the Kripke frame above. The following three rules are added to the tableau rules for C:

$$\begin{array}{cccc} A > B, i & \neg(A > B), i & & \\ ir_Aj & \downarrow & & \\ \downarrow & ir_Aj & \swarrow & \searrow \\ B, j & A, j & \neg A, i & A, i \\ & \neg B, j & & ir_Ai \end{array}$$

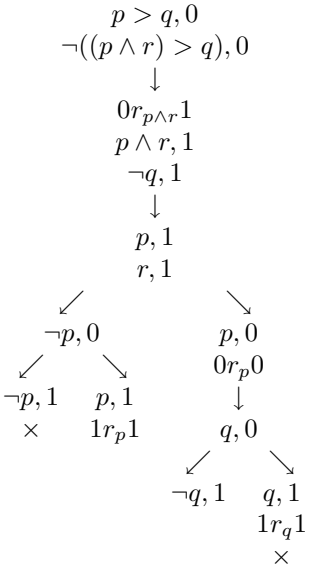
The difference from a modal tableau is that each formula has its own relation of accessible worlds like ir_Aj.

As an example for a tableau proof in C^+, we prove $A, A > B \vdash_{C^+} B$.

$$\begin{array}{cc} A, 0 & \\ A > B, 0 & \\ \neg B, 0 & \\ \swarrow & \searrow \\ \neg A, 0 & A, 0 \\ \times & 0r_A0 \\ & \downarrow \\ & B, 0 \\ & \times \end{array}$$

First, the rightmost rule of C^+ is applied, which yields $0r_A0$. Then, the leftmost rule can be applied to $A > B, 0$. This is closed by contradiction of $\neg B, 0$ and $B, 0$.

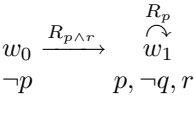
The tableau proof that follows is of $p > q \nvdash_{C^+} (p \wedge r) > q$ in C^+. This tableau proves that the inferences that result in the infelicitous semantics described in section 1.1 are not valid when implementing the concept of *ceteris paribus*.



Not all branches close completely, and for branches which do not do so, namely $p, 1$ and $\neg q, 1$, a similar formula already exists at the upper nodes. Thus, it seems to apply the same rule infinitely, indicating that this formula is indeed invalid.

Showing that the tree does not close does not mean that the formula is invalid, but indicates the possibility that it is invalid. One good way to *prove* the invalidity of the formula is to draw a counter-model. Counter-models can be read off from an open branch of a tableau in a natural way.

The counter-model of the formula above is as follows: $w_1 R_p w_1, w_0 R_{p\wedge r} w_1$ and $\nu_{w_0}(\neg p) = 1$ and $\nu_{w_1}(p) = \nu_{w_1}(\neg q) = \nu_{w_1}(r) = 1$. In regard to other formulae for A, the accessibility relation R_A is defined such that $f_A(w) = [A]$ for all w. Thus, the interpretation can be depicted as follows, from which we can check that the accessibility relation of worlds forms an infinite loop. This means that this tableau is never closed and $p > q \nvdash_{C^+} (p \wedge r) > q$ is proved.



Similarly, cases (2) and (3) in the 1.1 are invalid in C^+. Thus, by extending to C^+, we have solved the problem outlined in the 1.1 that classical logic is too weak as semantics in natural language.

4.2 S

In section 3.2, we said that $w_1 R_A w_2$ means that w_2 is the same as w_1 except that A is true in w_2. Thus, we need somehow to consider the notion of similarity between two worlds, in order to reflect the intuition behind the ceteris paribus condition. In C and C^+, however, the relation R_A by no means represents such notion of similarity.

S is an extension of C^+ through the addition of the following three conditions on a Kripke frame[1], which express a certain "similarity" between the two arguments of R_A.

3. If $[A] \neq \phi$, then $f_A(w) \neq \phi$
4. If $f_A(w) \subseteq [B]$ and $f_B(w) \subseteq [A]$, then $f_A(w) = f_B(w)$
5. If $f_A(w) \cap [B] \neq \phi$, then $f_{A \wedge B}(w) \subseteq f_A(w)$

As a result, for example, the inference $p > q, q > p \vdash (p > r) \equiv (q > r)$ is valid in S, but not in C^+.

Proof. Suppose that the premise is true in world w, i.e., $f_p(w) \subseteq [q]$ and $f_q(w) \subseteq [p]$. Then, by applying condition **4**, $f_p(w) = f_q(w)$. Hence, $f_p(w) \subseteq [r]$ iff $f_q(w) \subseteq [r]$, i.e., $(p > r)$is true in w iff $(q > r)$ is true in w. i.e., $(p > r) \equiv (q > r)$ is true in w.

Because S does not have a corresponding tableau system, the proof in S can rely on the semantic notions alone.

4.3 C_1, C_2

Although S is a stronger logic system than C^+, it is still weak as semantics of natural language. This prompted Stalnaker and Lewis to propose extensions[2] of S: C_1 [8] and C_2 [3,4]. Conditions **1** to **5** of S above are common conceptions of C_1 and C_2 [7].

C_1 and C_2 also make a difference through the addition of the following conditions. C_1 adds the following condition.

6. If $w \in [A]$ and $w' \in f_A(w)$, then $w = w'$

C_2, on the other hand, adds the following instead of 6.

7. If $x \in f_A(w)$ and $y \in f_A(w)$, then $x = y$

Both condition **6** and condition **7** concern the relationship between two worlds, but the difference between them is that **7** entails **6**, according to **2**. So we can say that C_2 is stronger than C_1. We can sort these systems by increasing strength as follows: $C^+ < S < C_1 < C_2$.

[1] The system S is defined as a common part of the conditional logics proposed by Stalnaker [8] and Lewis [3,4]. Following the convention in [6], we call it S.
[2] The names C_1 and C_2 are taken from [6].

However, C_1 and C_2 are not without problems [2]. For example, $A \wedge B \vdash A > B$ is one of the formulae that is valid in C_1 but not in S.

Suppose you go to a fake fortune-teller, who says that you will come into a large sum of money. And suppose that, purely by accident, you do. The statement "If the fortune-teller says that you will come into a large sum of money, you will" still, however, would appear to be false.

Similarly, $\vdash (A > B) \vee (A > \neg B)$ is an example that it is valid in C_2 but not in S. However, both of the following conditionals would appear to be false: "If it will either rain tomorrow or it won't, then it will rain tomorrow" and "If it will either rain tomorrow or it won't, then it won't rain tomorrow."

Thus, the two conceptions yield empirically wrong predictions, which are good illustrations that each logic system is too strong to be semantics of natural language. Given the above, although C_1 and C_2 are the strongest existing conditional logic, they do not seem to be suitable as a logic system for natural language.

5 Proposal: A New Conditional Logic C_b

In section 4, we pointed out the drawbacks with existing systems of conditional logic. However, viewing them purely from the perspective of the semantics of natural language, C^+ and S are too weak and C_1 and C_2 are too strong.

Moreover, S, C_1 and C_2 have no known tableau systems or any other proof systems. In developing a logic system for natural language, especially in the context of natural language processing, whether it enables us to *compute* the entailment relation between given sentences is an important feature, and is a feature that has not been achieved in previous studies with a few notable exceptions, such as [5].

Against such background, we propose here, as semantics of natural language, a logic system we call C_b that properly extends C^+ and has a corresponding tableau system.

One of the common problems with C^+, S, C_1 and C_2 is when $\wedge$ or $\vee$ appears on the accessibility relations: when such truth functions occur in the antecedents of formulae, we cannot apply any rules, and all inferences including these shall be invalid.

The system C_b is obtained by adding the following new conditions on the Kripke frame. It allows the nature that is inherited from C^+ to be preserved.

8. $\bigcup_{w' \in f_A(w)} f_B(w') \subseteq f_{A \wedge B}(w)$

9. $f_A(w) \subseteq f_{A \vee B}(w)$
$f_B(w) \subseteq f_{A \vee B}(w)$

C_b has a tableau system which is an extension of that of C^+ through the addition of the following three rules:

$$\begin{array}{ccc} ir_Aj & ir_Aj & ir_Aj \\ jr_Bk & \downarrow & \downarrow \\ \downarrow & ir_{A\vee B}j & ir_{B\vee A}j \\ ir_{A\wedge B}k & & \end{array}$$

The additional rules refer only to an accessibility relation not formulae. The left rule is for $\wedge$. For every ir_Aj and jr_Bk on the branch, we can derive $ir_{A\wedge B}k$. The right two rules are for $\vee$. For every ir_Aj on the branch, we can derive $ir_{A\vee B}j$ and $ir_{B\vee A}j$ for any B. In section 7 and section 8, we prove that this tableau system is sound and complete with respect to the Kripke semantics introduced above.

Note that any theorem of C^+ is a theorem of C_b. In the next section, we will verify what inferences are valid in C_b.

6 Empirical Verification

The following tableau is an example of a proof in C_b. We note, in passing, that all the following inferences are valid in C_b but not in C^+, S, C_1 or C_2.[3]

Ex.1 $(p \wedge q) > r \vdash p > (q > r)$

$$\begin{array}{c} (p \wedge q) > r, 0 \\ \neg(p > (q > r)), 0 \\ \downarrow \\ 0r_p1 \\ p, 1 \\ \neg(q > r), 1 \\ \downarrow \\ 1r_q2 \\ q, 2 \\ \neg r, 2 \\ \downarrow \\ 0r_{p\wedge q}2 \\ \downarrow \\ r, 2 \\ \times \end{array}$$

This is an example of applying the left rule proposed in section 5.

$0r_{p\wedge q}2$ is derived by applying the C_b rule to the second line $0r_p1$ and the third line $1r_q2$. This formula creates a sentence like the following.

"If the rain stops and the water temperature is more than $25°C$, then we can swim in the pool."
$\Rightarrow$"If the rain stops, then additionally, if the water temperature is more than $25°C$, we can swim in the pool."

[3] As discussed in section 4.1, we need to construct a counter-model for proving the invalidity of a formula, but we omit this here due to the space limitations.

Ex.2 $(p \wedge q) > r, p > q \vdash p > r$

$$
\begin{array}{c}
(p \wedge q) > r, 0 \\
p > q, 0 \\
\neg(p > r), 0 \\
\downarrow \\
0 r_p 1 \\
p, 1 \\
\neg r, 1 \\
\downarrow \\
q, 1 \\
\swarrow \quad \searrow \\
\neg q, 1 \quad q, 1 \\
\times \quad 1 r_q 1 \\
\downarrow \\
0 r_{p \wedge q} 1 \\
\downarrow \\
r, 1 \\
\times
\end{array}
$$

This inference is little changed from Ex. 1, and similarly to Ex. 1, it applies the left rule as in section 5 above.

This formula creates a sentences like the following:
"If A and B come, C will also come. And if A comes then B comes."
$\Rightarrow$ "If A comes then C comes."

Ex.3 $(p \vee q) > r \vdash (p > r) \wedge (q > r)$

$$
\begin{array}{cc}
(p \vee q) > r, 0 & \\
\neg((p > r) \wedge (q > r)), 0 & \\
\swarrow \quad \searrow & \\
\neg(p > r), 0 \quad \neg(q > r), 0 & \\
\downarrow \quad \downarrow & \\
0 r_p 1 \quad 0 r_q 2 & \\
p, 1 \quad q, 2 & \\
\neg r, 1 \quad \neg r, 2 & \\
\downarrow \quad \downarrow & \\
0 r_{p \vee q} 1 \quad 0 r_{p \vee q} 2 & \\
\downarrow \quad \downarrow & \\
r, 1 \quad r, 2 & \\
\times \quad \times &
\end{array}
$$

This is an example of applying the right rules proposed in section 5 above.

In order to apply the rule to the antecedent of the formula, it is necessary to account for the accessibility relation of $p \vee q$. Then, after $0 r_p 1$ and $0 r_q 2$ are

produced, they derive $0r_{p\vee q}1$ and $0r_{p\vee q}2$, respectively. Hence, closing becomes possible by applying the rule to the antecedent of the given formula.

This formula creates sentences like the following:
"If it rains or snows, the game will be cancelled."
⇒"If it rains, the game will be cancelled and if it snows, the game will be cancelled."

Ex.4 $p > q, (p \wedge \neg r) > \neg q \vdash p > r$

$$
\begin{array}{c}
p > q, 0 \\
(p \wedge \neg r) > \neg q, 0 \\
\neg(p > r), 0 \\
\downarrow \\
0r_p1 \\
p, 1 \\
\neg r, 1 \\
\downarrow \\
q, 1 \\
\swarrow \qquad \searrow \\
\neg\neg r, 1 \qquad \neg r, 1 \\
\downarrow \qquad\quad 1r_{\neg r}1 \\
r, 1 \qquad\quad \downarrow \\
\times \qquad\quad 0r_{p\wedge\neg r}1 \\
\qquad\quad \downarrow \\
\qquad\quad \neg q, 1 \\
\qquad\quad \times
\end{array}
$$

Although the above inference is valid in C_b, there is a problem in regard to its empirical validity. For example, the conditional corresponding to this formula seems to be false: 'If there are various drinks there, I will go. Even though there are various drinks, if there is no pizza, I won't go."
⇒?∗"If there are various drinks, there will be pizza."

Ex.5 $(p \wedge r) > (r \wedge q), p > r \vdash p > q$

$$
\begin{array}{c}
(p \wedge r) > (r \wedge q), 0 \\
p > r, 0 \\
\neg(p > q), 0 \\
\downarrow \\
0r_p1 \\
p, 1 \\
\neg q, 1 \\
\downarrow \\
r, 1 \\
\swarrow \qquad \searrow \\
\neg r, 1 \qquad r, 1 \\
\times \qquad\quad 1r_r1 \\
\qquad\quad \downarrow
\end{array}
$$

$$
\begin{array}{c}
0 r_{p\wedge r} 1 \\
\downarrow \\
r \wedge q, 1 \\
\downarrow \\
r, 1 \\
q, 1 \\
\times
\end{array}
$$

This inference is also valid in C_b, but we cannot find a conditional to correspond to it. In such instance, the system may be too strong. Therefore, in future research, we need to find a suitable limitation.

7 Soundness

Our proof of the soundness and completeness of C_b is based on the proof of soundness and completeness of C^+ given in [6].

Definition 1 (Faithfulness). *Let $I = \langle W, R, \nu \rangle$ be any Kripke interpretation, and b be any branch of a tableau. Then, I is faithful to b iff there is a map $g : \mathbb{N} \to W$ such that:*

1. *For every node A, i on b, A is true at $g(i)$ in I.*
2. *If $ir_A j$ is on b, $g(i) R_A g(j)$ in I.*

Lemma 2 (Soundness Lemma). *Let b be any branch of a tableau and $I = \langle w, R, \nu \rangle$ be any Kripke interpretation. If I is faithful to b and a tableau rule is applied to it, then it produces at least one extension b' such that I is faithful to b'.*

Proof. Since C^+ is proved to be sound with respect to its semantics given in section 4.1, we merely have to check the case for each rule of C_b.

The argument for the left tableau rule in section 5 is the following. Suppose there are $ir_p j$ and $jr_q k$ on a branch to which I is faithful and we apply the left rule and obtain $ir_{p\wedge q} k$. According to the definition of faithfulness, $g(i) r_p g(j)$ and $g(j) r_q g(k)$ are in I. Hence:

$$
\begin{array}{lll}
g(k) \in \{x \mid \exists w' \ (g(i) R_p w \wedge w R_q x)\} & & \\
\quad \equiv \bigcup_{w' \in f_p(w)} f_q(g(i)) & & \\
\quad \subseteq f_{p\wedge q}(g(i)) & & \text{(according to Condition } \mathbf{8} \text{ in section 5)} \\
\quad \equiv \{x \mid g(i) R_{p\wedge q} x\} & & \text{(according to the definition of } f_A(w)\text{)}
\end{array}
$$

Therefore, $g(i) R_{p\wedge q} g(k)$, which shows that I is faithful to this extension.

For the right rules, suppose that there is $ir_p j$ on a branch to which I is faithful and from applying the rule we obtain $ir_{p\vee q} j$. According to the definition of faithfulness, $g(i) r_p g(j)$ is in I. Hence:

$$
\begin{aligned}
g(j) \in & \{x \in W \mid g(i)R_p x\} \\
& \equiv f_p(g(i)) && \text{(according to the definition of } f_A(w)) \\
& \subseteq f_{p \vee q}(g(i)) && \text{(according to Condition } \mathbf{9} \text{ in section 5)} \\
& \equiv \{x \in W \mid g(i)R_{p \vee q} x\}
\end{aligned}
$$

Therefore, $g(i)R_{p \vee q}(j)$, which shows that I is faithful to this extension. The case for $ir_q j$ can be proved in the same way.

Theorem 3 (Soundness Theorem). *The tableau system of C_b is sound with respect to its semantics, i.e. for finite Σ, if $\Sigma \vdash A$ then $\Sigma \models A$.*

Proof. Suppose that $\Sigma \nvdash A$. Then we have the interpretation $I = \langle W, R, \nu \rangle$ that makes every formula in Σ true and A false, in some world w. Let $h : \mathbb{N} \to W$ be a map such that $h(0) = w$, which makes I faithful to the initial list. When we apply a rule to the list, there is at least one extension to which I is faithful, due to the Soundness Lemma. Thus, if the tableau is closed, there is at least one branch b for which the interpretation I is faithful, and there is a formula B such that both B and $\neg B$ are on b. This is impossible, however, because it means $\nu(B) = 1$ and $\nu(\neg B) = 1$. Therefore, the tableau must be open, i.e. $\Sigma \nvdash A$.

8 Completeness

Definition 4 (Induced Interpretation). *Let b be an open branch of a tableau. The interpretation $I = \langle W, R, \nu \rangle$ induced by b is defined as follows:*

- $W = \{w_i \mid i$ *occurs in* $b\}$
- *For any formula A:*
  - $w_i R_A w_j$ *iff* $ir_A j$ *is on b, if A occurs as the antecedent of a conditional or negated conditional at a node of b.*
  - $w_i R_A w_j$ *iff* $\nu_{w_j}(A) = 1$ *otherwise.*
- $\nu_{w_i}(A) = \begin{cases} 1 & \text{if } A, i \text{ occurs on } b \\ 0 & \text{if } \neg A, i \text{ occurs on } b \\ 1 \text{ or } 0 & \text{otherwise} \end{cases}$

Lemma 5 (Completeness Lemma). *Let b be any open complete branch of a tableau. Let $I = \langle W, R, \nu \rangle$ be the interpretation induced by b. Then:*

- *if A, i is on b, then $\nu_{w_i}(A) = 1$*
- *if $\neg A, i$ is on b, then $\nu_{w_i}(A) = 0$*

Proof. Since the syntax of C_b is the same as that of C and C^+, the Completeness Lemma can be proved in the same way [6].

Theorem 6 (Completeness Theorem). *The tableau system of C_b is complete with respect to its Kripke semantics: for finite Σ, if $\Sigma \models A$ then $\Sigma \vdash A$.*

Proof. Suppose that $\Sigma \nvdash A$. Given an open branch b of the tableau, the interpretation induced by b makes all the formulae in Σ true and A false in w_0, by the Completeness Lemma.

Now we must check that the induced interpretation satisfies conditions 1 and 2 of C^+, whose proof is given in [6], and conditions 8 and 9 of C_b given in section 5.

Suppose that b is a completed open branch. For any formula A, either A occurs on b as an antecedent or not. In the former case, the result holds according to the definition of R_A. In the latter case, let us check the two conditions of C_b in turn.

- For **8**, let w_x be any world. Suppose that there exists a world w_y such that $w_i R_A w_y$ and $w_y R_B w_x$. Then, according to the definition of induced interpretations, $ir_A y$ and $yr_B x$ occur on b. Since b is completed, $ir_{A\wedge B}x$ is also on b. Again, according to the definition of induced interpretations, $w_i R_{A\wedge B} w_x$ holds, as required.
- For **9**, let w_x be any world such that $w_i R_A w_x$. According to the definition of induced interpretations, $ir_A x$ occurs on b. Since b is completed, $ir_{A\vee B}x$ and $ir_{B\vee A}x$ occurs on b for any formula B. So, according to the definition of induced interpretations, $w_i R_{A\vee B} w_x$ and $w_i R_{B\vee A} w_x$ hold, as required.

Hence $\Sigma \nvdash A$.

9 Conclusion and Future Work

In this paper, we have explored in detail the problems of conditional sentences in natural language and proposed a new logic system C_b by extending existing conditional logics.

There are two advantages to our logic C_b: first, as a semantic theory of natural language, it is empirically more correct than preceding analyses; second, a tableau proof is available for C_b, which is sound and complete with respect to its Kripke semantics.

The following figure shows a comparison between S and C_b in regard to their valid inferences.

S \ C_b	valid	invalid
valid	$p > (q \wedge r) \vdash p > q$ $p > (p > q) \vdash p > q$	$p > q, q > r \vdash p > r$ $p > q, \neg(p > \neg r) \vdash (p \wedge r) > q$
invalid	$(p \vee q) > r \vdash (p > r) \wedge (q > r)$ $(p \wedge q) > r \vdash p > (q > r)$ $(p \wedge q) > r, p > q \vdash p > r$ $p > q, (p \wedge \neg r) > \neg q \vdash p > r$ $(p \wedge r) > (r \wedge q), p > r \vdash p > q$	$(p \wedge r) > (r \wedge q) \vdash (p > q) \vee (r > q)$

This figure indicates that inferences with conditional formulae that include $\wedge$ or $\vee$ in the antecedent are valid only in C_b, constituting what we believe to be a substantial extension.

For future work, further examination of the empirical validity of C_b is required. Cases such as Ex. 4 and Ex. 5 in section 6 may be problematic for the current version of C_b. Moreover, we should think about how to treat inferences with a conditional formula whose antecedent contains the negation symbol $\neg A$. The availability of an automatic proof and its implementation remain as topics for future work.

Acknowledgments. We wish to thank Eric McCready, Alastair Butler, Kei Yoshimoto and Kenichi Asai for their comments on an earlier version of this paper. We would also thank the reviewers of this paper for their useful and valuable comments. Daisuke Bekki is partially supported by Grant-in-Aid for Young Scientists (A), 22680013, 2010-203, from the Ministry of Education, Science, Sports and Culture, Japan.

References

1. Chellas, B.F.: Basic conditional logic. Journal of Philosophical Logic 4, 53–133 (1980)
2. Harper, W.L., Stalnaker, R., Pearce, G.: Ifs. Reidel, Dordrecht (1981)
3. Lewis, D.K.: Counterfactuals. Blackwell, Oxford (1973)
4. Lewis, D.K.: Counterfactuals and comparative possibility. Journal of Philosophical Logic 2 (1973)
5. Muskens, R.: A compositional discourse representation theory. In: The 9th Amsterdam Colloquium, pp. 467–486 (1993)
6. Priest, G.: An Introduction to Non-Classical Logic. Cambridge Uniersity Press, Cambridge (2008)
7. Read, S.: Thinking About Logic: An Introduction to the Philosopy of Logic. Oxford University Press, Oxford (1994)
8. Stalnaker, R.: A Theory of Conditionals, in Studies in Logical Theory. American Philosophical Quanterly Monograph Series, vol. 2. Basil Blackwell, Oxford (1968)

Are (Linguists') Propositions (Topos) Propositions?

Carl Pollard

Department of Linguistics, The Ohio State University,
1712 Neil Avenue, Columbus, Ohio 43210, USA
pollard@ling.ohio-state.edu

Abstract. Lambek([22]) proposed a categorial achitecture for natural language grammars, whereby syntax and semantics are modelled by a **biclosed monoidal category (bmc)** and a cartesian closed category (ccc) respectively, and semantic interpretation by a functor from syntax to semantics that preserves the biclosed monoidal structure; essentially this same architecture underlies the framework of abstract categorial grammar (ACG, de Groote [12]), except that the bmc is now symmetric, in keeping with the collapsing of Lambek's directional implications / and \ into the linear implication $\multimap$. At the same time, Lambek proposed that the semantic ccc bears the additional structure of a **topos**, and that the meanings of declarative sentences—linguist's propositions—can be identified with propositions in the sense of topos theory, i.e. morphisms from the terminal object 1 to the subobject classifier Ω. Here we show (1) that this proposal as it stands is untenable, and (2) that a serviceable framework results if a **preboolean algebra object** distinct from Ω is employed instead. Additionally we show that the resulting categorial structure provides 'for free', via Stone duality, an account of the relationship between fine-grained 'hyperintensional' semantics ([6],[33],[27],[28]) and the familiar coarse-grained intensional semantics of Carnap ([2]) and Montague ([26]).

Keywords: proposition, hyperintension, intension, topos, Stone duality, preboolean algebra.

1 Introduction

Montague ([26]) was first to systematically apply the methods of mathematical logic to the analysis of natural language (NL) meaning, and a great deal of the subsequent history of NL semantics has consisted of attempts to repair, improve, or elaborate on Montague semantics (MS). Lambek ([22]) made a significant though little-known contribution to this enterprise. Here I focus on one aspect of Lambek's contribution, namely his proposal that NL meanings be modelled by morphisms in a *topos*, and that, in particular, declarative sentence meanings (usually called propositions by linguists and philosophers) can be identified with the topos-theorist's propositions, i.e. morphisms from the terminal object 1 to the

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 205–218, 2011.

subobject classifier Ω.[1] Specifically, I will show that although Lambek's proposal as it stands is untenable, a serviceable framework results if a preboolean algebra object distinct from Ω is employed instead of Ω itself. Moroever, I will show that the resulting categorical structure provides 'for free', via the natural topos-theoretic generalization of Stone duality, a straightforward account of the relationship between fine-grained 'hyperintensional' semantics (Cresswell [6],[33],[27],[28]) and the familiar coarse-grained intensional semantics of Carnap ([2]) and Montague ([26]).) and the familiar coarse-grained intensional semantics inherited by mainstream linguistic semantics from Carnap ([2]) and Montague ([26]).

As is well known, Montague semantics is a synthesis (some would say a hodgepodge) of ideas from several sources. From Frege ([9]) came the ideas (1) that a (linguistic) expression has a **sense** (which does not depend on how things are) and a **reference** (which does); (2) that the sense of a declarative sentence is a **proposition** and the reference is that proposition's **truth value**; and (3) that the sense of an expression is a function of the senses of its syntactic constituents ('compositionality'). From Carnap ([2]) came the idea that meanings are **intensions**, functions that map each world to the reference at that world. And from Kripke ([17]) came the idea that worlds are not complete state descriptions (as per Carnap) or maximal consistent sets of propositions (as per earlier Kripke ([16])), but rather unanalyzed primitives.

Montague's semantic theory was written in an idiosyncratic higher-order language (IL), subsequently shown by Gallin ([10]) to be essentially equivalent to a version (Ty2) of the Church-Henkin ([5],[13]) simple theory of types with one additional basic type for worlds. (Hereafter we ignore IL and pretend that Montague semantics was written in Ty2 all along.) Moreover, this theory was only one component of a larger account of how strings of words come to express senses and (relative to a world) denote references. In this account, the relationship between strings and sense is mediated by an explicit (albeit primitive) categorial grammar (primitive in the sense of lacking hypothetical proof, along the lines of Ajdukiewicz ([3]) and Bar-Hillel ([4])) which recursively defines a set of ordered triples consisting of (1) a string, (2) a syntactic type, and (3) an intension. As Montague (p. 263) observes, such a grammar defines a function ('translation') from analysis trees (categorial grammar proofs) to intensions. It is this translation function which, in Lambek's categorical reformulation, is rendered as a biclosed monoidal functor.

Of particular importance for present purposes is Montague's system of semantic types. Besides the truth value type t (Henkin's o) provided by the logic, there are two basic types: e, the type of **entities** (Henkin's ι) and w (Montague's s), the type of **worlds**. In particular, the type (here called p) for propositions in the sense of declarative sentence meanings is not a basic type, but rather (in spirit following Carnap) is *defined* to be $\mathrm{w} \rightarrow \mathrm{t}$ (sets of worlds). Correspondingly, for a proposition to be **true** at a world is for the world to be a set-theoretic member of the proposition. In this scheme of things, the intensions for the NL 'logic

[1] For discussion of other aspects of Lambek's categorial semantics and relevant historical background, see Pollard [29].

words' are the expected boolean operations on propositions, e.g. (here $\rightsquigarrow$ is 'is translated as'):

$$\textit{and} \rightsquigarrow \lambda_{pqw}.(p\ w) \wedge (q\ w) : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$$
$$\textit{implies} \rightsquigarrow \lambda_{pqw}.(p\ w) \rightarrow (q\ w) : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$$

i.e. intersection and relative complement of sets of worlds; and the centrally important relation of NL semantics, **entailment**, is modelled by subset inclusion in $\mathrm{w} \rightarrow \mathrm{t}$:

$$\mathsf{entails} =_{\mathrm{def}} \lambda_{pq}.\forall_w.(p\ w) \rightarrow (q\ w) : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{t}$$

At first blush, this seems right because intuitively, for p to entail q is supposed to mean that, no matter how things are, if p is true with things that way, then so is q. But it has long been well known (even, in fact, to Montague) that there are problems with assuming that propositions are sets of worlds. One of the most obvious of these is the **logical omniscience** problem: since in MS entailment is modelled as the subset inclusion order in the set of sets of worlds, mutually entailing propositions must be equal; and therefore there can be only one necessary truth, namely the set of all worlds. So anybody who knows some necessary truth (e.g. that Sarah Palin is Sarah Palin) knows them all (e.g. the Riemann Hypothesis or its denial, whichever is true).

Logical omniscience is one aspect of a more general problem of **granularity**: MS meanings do not make sufficiently fine-grained distinctions to account for a wide range of robust entailment patterns. Another aspect of granularity is the problem of donkeys and asses. Thus, since *Chiquita is a donkey* and *Chiquita is an ass* express mutually entailing propositions (call them p and q), MS treats them as identical. But what if Pedro believes the first but not the second? Under the standard assumption that belief is a relation between entities and propositions, that is only possible if p and q are distinct. The moral of this little story is that, unlike the case of MS, the relation used to model natural-language entailment better not be antisymmetric. We will return to this in due course.

2 Lambek's Categorical Semantics

Lambek's contribution to semantics was to observe that, properly reconstructed, MS is anything but a hodge-podge of Frege, Kripke, Carnap, and Ajdukiewicz. In the categorical semantics (hereafter CS) that he proposed, Montague's primitive applicative categorial grammar is replaced first with Lambek's ([18]) syntactic calculus, and then ([22], p. 308) 'we raise [the] syntactic calculus to the level of a category by introducing appropriate equations'. The resulting category, called a a **biclosed monoidal category** or **mbc** ([19],[20]), stands in the relation to the syntactic calculus which generalizes the well-known relation in which a cartesian closed category (ccc) stands to positive intuitionistic propositional logic: the types become the objects of the categories, and the morphisms from A to B model (or perhaps better, reify) equivalence classes of proofs of B from A. Then the categorical counterpart of a Montagovian analysis tree (or of an equivalence class of

proofs in the syntactic calculus) for a linguistic expression of syntactic type A is a morphism from I to A, where I is the monoidal unit object of the bmc.

In place of Montague's semantic model, essentially a standard model (in the sense of [13]), CS employs a ccc, more specifically, a **topos**, roughly speaking, the categorical embodiment of a higher-order type theory, to which we return below. In the topos, the counterparts of Montagovian meanings (intensions) of semantic type B are **global elements** of B, i.e. morphisms from 1 to B where 1 is the terminal object of the topos.

And in place of Montague's translation function from analysis trees to meanings, CS has a functor (in the sense of category theory, not in the categorial-grammar sense of a linguistic expression that takes a syntactic argument) from the bmc to the topos which preserves the biclosed monoidal structure. This makes sense because a topos is a ccc, and a ccc in turn is a 'degenerate' bmc.[2] Here the object-level mapping of the functor correponds to Montague's function f from English syntactic categories to semantic types. And finally, the meaning of an expression (morphism in the bmc) is the topos morphism which its image by this functor at the level of morphisms.

But what is a topos? For present purposes, it suffices to think of a topos as simply a ccc endowed with enough additional structure to enable one to speak of the 'characteristic function' of a 'subset'. The most familiar topos is the category of sets, with sets as the objects and functions with specified codomains (not just specified domains) as the morphisms. In this case, for a set A, we can identify a subset B with the morphism m from B to A which is the embedding function mapping each element of B to 'itself thought of as an element of A'; while the characteristic function is a certain 'A-predicate' (morphism from A to the set $2 =_{\text{def}} \{0,1\}$), namely the (necessarily unique) function χ whose composition with m coincides with the function $\top_B$ from B to 2 which maps every every element of B to 1. Note here that $\top_B$ is itself a composition $\top \circ g$, where g is the unique function from B to the singleton set $1 =_{\text{def}} \{0\}$, and $\top$ is the function from 1 to 2 that maps 0 to 1.

The notion of a topos is the natural categorical generalization of this situation, with 'subset' replaced by the usual categorical notion of subobject (as an equivalence class of monics), 2 replaced by a distinguished object Ω, whose global elements are called **propositions**, and $\top$ replaced by a distinguished proposition true. (See e.g. Goldblatt ([11]) for detailed definitions and discussion.) For the ccc to count as a topos, it is required that true **classify subobjects**. What this means is that for each monic $m : B \rightarrow A$, there must be a unique **characteristic morphism** $\mathbf{char}(m) : A \rightarrow \Omega$. The precise definition of **char** [3] has as one important consequence that $\mathbf{char}(m) \circ m$ coincides with $\mathsf{true} \circ \mathbf{O}_B$, where $\mathbf{O}_B$ is

[2] One can think of this degeneration, which arises from the imposition of equations corresponding to the structural rules of permutation, contraction, and weakening, as signaling the transition from the 'resource sensitivity' of syntax, where order and repetition are significant, to the realm of propositional content where they are not.

[3] Technically: for χ to count as the char of m, the diagram equating $\chi \circ m$ with $\mathsf{true} \circ \mathbf{O}_B$ is required not merely to commute but moreover to be a pullback.

the unique morphism from B to the terminal object 1; here $\mathbf{0}_B$ and true are the respective categorical generalizations of the functions $\top$ and g above. A second consequence of the definitions of char and subobject classifier is that two monics have the same char iff they are equivalent as subobjects (in the sense of factoring through each other). And a third consequence, of particular importance for us, is that Ω itself is endowed with a natural structure as a heyting algebra object. (In fact, we could further require that the topos be **boolean**, in the sense that LEM (in the form $\vdash \forall_t t \vee \neg t$) hold in the internal language (see immediately below), but we do not insist on this. However, since natural language entailment is usually taken to be classical, we will impose the weaker condition that Ω be a boolean algebra object.) Toposes so defined generalize not only the category of sets, but also the notion of 'model of a higher order theory'; in fact, the (generalized) models of Henkin ([13]) are, up to isomorphism, just the toposes which are subcategories of the category of sets.

What in CS corresponds to Montague's IL (or, as we are presenting MS, to Ty2)? As Lambek and Scott ([23]) discuss at length, just as each ccc corresponds in a well-known and natural way (i.e. via a categorical equivalence) to a typed lambda calculus ([21]), so each topos corresponds to a certain higher-order intuitionistic type theory, called the **internal language** of the topos. Thus the move from a (mere) ccc to a topos generalizes the move that Church ([5]) made from (mere) lambda calculus to the simple theory of types, where equality is represented *internally* by a family of object-language constants $=_A$, as opposed to lambda-calculus term equivalence which is defined in the metalanguage.[4] And just as in the simple theory of types ([13]), all the usual intuitionistic (or, if desired, classical) connectives and quantifiers of higher order logic become definable in terms of (internal) equality.

3 Are Propositions Propositions?

Up to this point, it would appear that CS has to be judged a success: it shows that Montague was well on the way to having an elegant and natural theory of NL meaning and its connection with grammar. But what about the granularity problem? Does CS fare any better than MS did?

To answer this, we have to take into account an aspect of CS that we have not yet considered, namely Lambek's proposal to identify senses of declarative sentences—linguist's propositions—with topos propositions (morphisms from 1 to Ω). In this connection, recall that Ω is a boolean algebra object, the CS counterpart of Montague's denotation for the type $\mathrm{p} = \mathrm{w} \rightarrow \mathrm{t}$, namely the powerset of the set of worlds. In the Montague setting, as noted above, natural language entailment is modeled as subset inclusion of sets of worlds, which is just a special case of the order induced induced on a boolean algebra by its boolean structure, viz. $p \sqsubseteq p$ iff $p = p \sqcap q$. Since, in a topos with boolean Ω, the boolean

[4] Under this correspondence, the counterpart of the constant $=_A$ in the interal language of a topos is $\mathbf{char}(\delta_A)$, where $\delta_A =_{\mathrm{def}} A \rightarrow A \times A$ is the diagonal morphism $(\mathbf{id}_A, \mathbf{id}_A)$.

connectives of the internal language correspond to the boolean operations on Ω itself, the CS constant denoting natural language entailment must be subject to the axiom

$$\vdash \mathsf{entails} = \lambda_{pq}.(p \wedge q) = p : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{t}$$

or equivalently

$$\vdash \forall_{pq}.(p \ \mathsf{entails} \ q) = ((p \wedge q) = p)$$

Now one thing we can observe right away is that CS is afflicted by the granularity problem, for the same reason that MS is, namely that entailment is a (internal) order, and in particular antisymmetric. However, it is important to appreciate the precise manner in which CS fails to avoid the granularity problem. It is not *merely* the straightforward categorical generalization of the fact the entailment among propositions in MS is antisymmetric. In fact, things are far worse than that. To see why, recall that in MS, entailment and implication are distinct. In fact they are not even of the same type: as just noted, entails has type $\mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{t}$ (a binary *relation* on propositions), whereas implication was defined in MS as a binary *operation* on propositions, namely the relative complement operation on the powerset of the set of worlds:

$$\mathit{implies} \rightsquigarrow \lambda_{pqw}.(p\ w) \rightarrow (q\ w) : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$$

Now MS is far from perfect, but this distinction between entailment and implication is a feature, not a bug, of MS. In CS though, since Ω is the object whose global elements model linguistic propositions, it is with respect to the natural boolean structure on this object that the words *and* and *implies* have to be interpreted, namely as the meet $\wedge$ and relative complement $\rightarrow$ respectively. Unfortunately, it is a fact about toposes that $\rightarrow$ is internally definable in terms of $\wedge$ and object-language equality ([23], p. 133):

$$\mathrm{p} \rightarrow q =_{\mathrm{def}} (p \wedge q) = p$$

i.e. for any two propositions p and q, $p \rightarrow q$ is *the same morphism as* $(p \wedge q) = p$, so that the distinction between implication and entailment collapses! On a more intuitive level, this collapse comes about because topos propositions are being forced to do double duty as both the senses of declarative sentences ('linguistic propositions') and the corresponding references ('truth values'), which is reflected by the fact that global elements of Ω are often called 'truth values' rather than propositions.

One disastrous consequence of this collapse (one is enough) is what I refer to as the **total omniscience problem**. Let p, q, r, and s be the propositions expressed by the following four sentences:

1. Chiquita is a donkey
2. Frances is a mule.
3. Pedro knows Chiquita is a donkey.
4. Pedro knows Frances is a mule.

so that $p = (\mathsf{donkey\ chiquita})$, $q = (\mathsf{mule\ frances})$, $r = (\mathsf{know}\ p\ \mathsf{pedro})$, and $s = (\mathsf{know}\ q\ \mathsf{pedro})$. Then we can easily prove (internally):

$$\vdash (p \wedge q \wedge r) \rightarrow s$$

(Hint: $p \wedge q$ is defined as $\langle p, q\rangle = \langle \mathsf{true}, \mathsf{true}\rangle$, and so $\vdash (p \wedge q) \rightarrow (p = q)$.) More generally: anyone who knows some (possibly contingent) truth is *totally* omniscient (knows *every* truth, not just necessary ones but contingent ones also!

4 Preboolean Algebras

A way into overcoming this problem was suggested to me (separately) by Drew Moshier and Bill Lawvere (p.c. 2000), namely to use a boolean algebra object p distinct from Ω for sentence senses. This is reminiscent of Thomason's ([33]) intentional logic, with a basic type p distinct from t. Howard Gregory (p.c., 2001) reminded me that although this solves the *total* omniscience problem, it's no help with *logical* omniscience because entailment is still antisymmetric. However, there is a way forward: axiomatize p not as a *boolean* algebra, but rather as something this just barely misses a boolean algebra, by failing to be antisymmetric. Such things are called *pre*boolean algebras, or equivalently, **boolean categories**, though they tend not to get much respect.[5] In fact there is a familar structure of just this kind, namely the set of formulas of classical propositional logic preordered by logical consequence. Of course, nobody ever pays much attention to this algebra; it is usually just seen as a station on the way to constructing the Lindenbaum algebra by identifying equivalent formulas. And in the setting of propositional (or even first-order) logic, this is an entirely natural thing to do, given that substituting an equivalent ψ' for a subformula ψ of a given formula ϕ produces an equivalent formula ϕ', a simple consequence of the fact that the algebra operations are all functorial (i.e. either monotone or antitone in each argument). But things are different in the realm of linguistic propositions, where the typical propositional operators (expressed by verbs of propositional attitude) are not functorial. In that realm, as is widely appreciated (e.g. by advocates of impossible worlds, structured propositions, intentional logic, and hyperintensional semantics), there is ample cause to doubt that natural language entailment is antisymmetric.

To understand what a preboolean algebra is, recall that in any preordered set $\{A, \sqsubseteq\}$, there is an induced equivalence relation $\equiv$ given by

$$a \equiv b \text{ iff } a \sqsubseteq b \text{ and } b \sqsubseteq a$$

[5] Indeed, it is often noted that boolean categories are uninteresting because they are just boolean algebras, but this is only true up to categorical equivalence, since *pre*boolean algebras are just as good boolean categories as boolean algebras are; to put it another way, boolean algebras are the *skeletal* boolean categories (ones where the only isos are identities.)

Then a preboolean algebra is an algebra with operations just like those of a boolean algebra, except that all the usual equalities obtain only *up to equivalence (u.t.e.)*. To put it another way, a preboolean algebra is an algebra that yields a boolean algebra when quotiented by its own induced equivalence relation. The topos-internal counterpart of such a thing, an **internal boolean category**, or, equivalently, an pre-heyting algebra object that validates Double Negation Elimination u.t.e., is precisely what we will use to model the (linguists') propositions preordered by entailment. To this end, we first introduce the following constants:

Linguistic Propositional Connectives

a. $\vdash \mathsf{truth} : \mathrm{p}$
b. $\vdash \mathsf{falsity} : \mathrm{p}$
c. $\vdash \mathsf{not} : \mathrm{p} \rightarrow \mathrm{p}$
d. $\vdash \mathsf{and} : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$
e. $\vdash \mathsf{or} : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$
f. $\vdash \mathsf{implies} : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$
g. $\vdash \mathsf{entails} : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{t}$

Here not, and , or , and implies are the respective translations of the English sentential connectives *it is not the case that*, *and*, *or*, and (static, episodic) *if ... then*.

In the internal language, the preboolean axioms are as follows, where now we are using p, q, and r as variables of the type p of *linguistic* propositions, not of the type Ω of *topos* propositions, and where

$$p \equiv q =_{\mathrm{def}} (p \ \mathsf{entails}\ q) \wedge (q \ \mathsf{entails}\ p)$$

Axioms for Linguistic Entailment

a. $\vdash \forall_p.p \ \mathsf{entails}\ \mathsf{truth}$
b. $\vdash \forall_p.\mathsf{falsity}\ \mathsf{entails}\ p$
c. $\vdash \forall_{p,q}.(p \ \mathsf{and}\ q)\ \mathsf{entails}\ p$
d. $\vdash \forall_{p,q}.(p \ \mathsf{and}\ q)\ \mathsf{entails}\ q$
e. $\vdash \forall_{p,q,r}.((p \ \mathsf{entails}\ q) \wedge (p \ \mathsf{entails}\ r)) \rightarrow (p \ \mathsf{entails}\ (q \ \mathsf{and}\ r))$
f. $\vdash \forall_{p,q}.p \ \mathsf{entails}\ (p \ \mathsf{or}\ q)$
g. $\vdash \forall_{p,q}.q \ \mathsf{entails}\ (p \ \mathsf{or}\ q)$
h. $\vdash \forall_{p,q,r}.((p \ \mathsf{entails}\ r) \wedge (q \ \mathsf{entails}\ r)) \rightarrow ((p \ \mathsf{or}\ q) \ \mathsf{entails}\ r)$
i. $\vdash \forall_{p,q}.(p \ \mathsf{implies}\ q) \ \mathsf{and}\ p) \ \mathsf{entails}\ q$
j. $\vdash \forall_{p,q,r}.((r \ \mathsf{and}\ p) \ \mathsf{entails}\ q) \rightarrow (r \ \mathsf{entails}\ (p \ \mathsf{implies}\ q))$
k. $\vdash \forall_p.(\mathsf{not}\ p) \equiv (p \ \mathsf{implies}\ \mathsf{falsity})$
l. $\vdash \forall_p.(\mathsf{not}\ (\mathsf{not}\ p))\ \mathsf{entails}\ p$

And what about Ω? It hasn't gone anywhere, but it is no longer being asked to model the meanings of natural-language declarative sentences. Rather, it is used, just like Montague's t, for their reference (see Appendix). In recognition of that fact, we will rename it from Ω to t, and call its global elements **truth values**, reserving the term 'proposition' for the linguistic sentence meanings.

5 Background on Ultrafilters

Inside our semantic topos, we now have linguistic propositions (type p) and a preorder on them called entailment, but we have not yet related this preorder to the linguistic notion of entailment as that relation that obtains between (the natural-language sentences which express) two propositions just in case, no matter how things might be, if the first is true with things that way, then so is the second. These 'ways things might be' are what the unanalyzed primitive worlds of Montague ([26] (and Kripke ([17]) model. But there is an older tradition, exemplified by, *inter alia*, [15], [16], and [1], wherein worlds are constructed out of propositions, rather than the other way around, as ultrafilters (or equivalently, maximal consistent) sets of propositions. Pollard ([28]) sketches the logical underpinnings of a categorical approach along these lines, some features of which we will amplify and clarify in due course. Before that though, we review some basic notions about ultrafilters of (pre)boolean algebras.

Our point of departure is the Stone representation theorem (SRT) ([31], [32], [14]), which relates boolean algebras and their spectra. Here the **spectrum** $\mathbf{Spec}(B)$ of a boolean algebra B is its set of ultrafilters, topologized as follows. The open sets are taken to be the unions of the sets $\mathbf{C}_b$, where $b \in B$ and $\mathbf{C}_b$ is the set whose members are those ultrafilters to which b belongs. Then it turns out that the sets $\mathbf{C}_b$ are precisely the clopens (open-and-closed subsets) of $\mathbf{Spec}(B)$.[6] Then SRT can be stated as follows:

Theorem (Stone Representation Theorem)
For any boolean algebra B, the function mapping each $b \in B$ to $\mathbf{C}_b$ is an isomorphism onto a subalgebra of $\wp(\mathbf{Spec}(B))$.
We call this function the **Stone mapping** of B. To paraphrase, SRT says that the Stone mapping embeds B into a powerset algebra.

Obviously, this does not generalize to *pre*boolean algebras, since equivalent but unequal elements belong to the same ultrafilters. But what will eventually be of more interest to us is not SRT itself, but rather the main lemma involved in its proof, which *does* so generalize:

Lemma ((Pre)Boolean Algebras Have 'Enough' Ultrafilters)
If B is a (pre)boolean algebra with order $\sqsubseteq$, and $a, b \in B$ such that $a \not\sqsubseteq b$, then there is an ultrafilter F such that $a \in F$ but $b \notin F$.

This is a theorem of ZFC, but known to be weaker than Choice.

Now one thing that can be noted right away is that there is a mild reformulation of Montague semantics (MS), call it MS', wherein propositions are taken to be primitive and to form a boolean algebra P (so that entailment is still antisymmetric). If one goes that route, then SRT tells us that, as long as we identify worlds with ultrafilters, then we we get a natural one-to-one correspondence between propositions and (some, but not all) intensions (functions from

[6] Moreover, $\mathbf{Spec}(B)$ itself can be shown to be a **Stone space**, i.e. a compact Hausdorff space where every open is a union of clopens. Up to homeomorphism, every Stone space arises in this way.

worlds to truth values). The key difference between MS and MS' is that in MS, *every* set of worlds counts as a proposition, whereas in MS', only those sets of worlds which are clopens of $\mathbf{Spec}(P)$ do.[7]

For die-hard Montagovians, or those who, like Stalnaker ([30]), are willing to accept the consequences of having an antisymmetric entailment relation, MS' represents a tenable intermediate position between MS and full-blown hypertensionality. The advantage of MS' over MS is that, although it doesn't solve the granularity problem, it does fix some other, less well-known bugs of MS, of which we mention just one here. In MS, if we let W be the set of worlds, then there is a one-to-one correspondence between worlds and the *principal* ultrafilters over the boolean algebra $\wp(W)$ of propositions. But (assuming Choice) since there are infinitely many propositions, $\wp(W)$ must have a nonprincipal ultrafilter F. And so, F is a maximal consistent set of propositions which does not correspond to any 'way things might be' (i.e. member of W). This puzzle does not arise in MS', since there even the nonprincipal ultrafilters count as worlds.

6 Hyperintensional Categorical Semantics

We now turn to elaborating our topos semantics to countenance worlds and extensions. Worlds are straightforward, since the predicate $u : (\mathrm{p} \to \mathrm{t}) \to \mathrm{t}$ on sets of propositions of being an ultrafilter is internally definable ([28]): we can take the type w of worlds to be a subtype of $\mathrm{p} \to \mathrm{t}$ whose **char** is u.

Now we ask: does p have enough ultrafilters? Well, it is known that in any topos with choice, *any* preboolean algebra object has enough ultrafilters. But it is also known ([7]) that toposes with choice are boolean (i.e. $\vdash \forall_{t \in \mathrm{t}}.t \vee (\neg t)$), which we may not wish to insist on. In any case, the only preboolean object whose ultrafilters we care about is p, so instead we just directly require of p that it have enough ultrafilters. To that end, we use the general fact about toposes that every predicate has a **kernel**, i.e. for any morphism $f : A \to \mathrm{t}$, there exists an object B and a monic $\mu : B \rightarrowtail A$ such that $\mathbf{char}(\mu) = f$. Now let $\mu : \mathrm{w} \to \mathrm{p} \to \mathrm{t}$ be a kernel of u, let w be a variable of type w, and define

$$p@w \ =_{\mathrm{def}} \lambda_{pw}.\mu\ w\ p : \mathrm{p} \to \mathrm{w} \to \mathrm{t}$$

Informally speaking, this says of p that it is one of the *facts* of w. Now we are in a position to assert internally the axiom that p have enough ultrafilters:

$$\vdash \forall_{pq}.\neg(p \text{ entails } q) \to \exists_w.p@w \wedge \neg q@w$$

And from this follows a (very) weak internal form of Stone duality:

$$\vdash \forall_{p,q}.(\forall_w.(p@w) \leftrightarrow (q@w)) \to p \equiv q$$

[7] If P were finite, then MS and MS' would be indistinguishable, since then every set of worlds would be a clopen; but of course this is a non-starter.

That is: if two propositions are facts of the same worlds, then they are equivalent (but crucially, not necessarily equal). To put it another way, suppose we define the **intension** corresponding to a proposition p by

$$\mathbf{int}\ p =_{\mathrm{def}} \lambda_w.p@w : \mathrm{w} \to \mathrm{t}.$$

Note that **int** is precisely the internal counterpart of the Stone mapping, except that it needn't be monic. Note also that **int** p is much like a Carnapian intension ([2]), modulo the replacement of (syntactic!) 'complete state descriptions' by (semantic!) ultrafilters. Thus distinct propositions can correspond to one and the same intension. For this reason we call this framework a hyperintensional categorical semantics (HCS).

7 Conclusion

The main differences between HCS and MS can be summarized as follows:

- MS is written in the classical simple theory of types; HCS in the internal language of a (not necessarily classical) topos.
- In MS worlds are primitives and every set of worlds is a proposition; in HCS, propositions form a preboolean algebra and every ultrafilter is a world.
- In MS meanings are intensions; in HCS meanings are hyperintensions and their images under the generalized Stone mapping are intensions.
- The reason HCS is more fine-grained than MS is that the Stone mapping at type p isn't monic.

There is no shortage of issues remaining. Among them:

- Here we have required the subobject classifier to be boolean, but not required the topos itself to be boolean. Does it matter?
- Does the framework extend straightfowardly from static to dynamic semantics? (For some steps in this directions, see Martin and Pollard ([24], [25]).
- Here we simplified the semantics of noun phrases by assuming the extensional type correpsonding to e is e itself (i.e. there is no type distinction between i (individual concepts) and e (individuals/entities), as there is for Carnap and Montague. Is such a distinction needed, e.g. for Hesperus-Phosphorus-type problems, or can they be handled dynamically via discourse referents?
- How should fictional entities be treated, or actual ones which are fictional from the point of view of other worlds?
- Aside from granularity, does treating worlds as ultrafilters help with other problems in the analysis of modality, attitudes, and counterfactuality?

Appendix

For reference, we sketch here how to elaborate HCS in order to account for intensions and extensions at all meaning types (not just p).

(1) **Extensional and Intensional Types**

- We define the **hyperintensional** types to be p, e, and types obtained from these using the cartesian type constructors (1, $\times$, and $\rightarrow$).
- For each hyperintensional type A, the corresponding **extensional** type $\mathrm{Ext}(A)$ is defined as follows:

  $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$

  $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$

  $\mathrm{Ext}(1) = 1$

  $\mathrm{Ext}(A \times B) = \mathrm{Ext}(A) \times \mathrm{Ext}(B)$

  $\mathrm{Ext}(A \rightarrow B) = A \rightarrow \mathrm{Ext}(B)$
- the corresponding **intensional** type is $\mathrm{Int}(A) =_{\mathrm{def}} \mathrm{w} \rightarrow \mathrm{Ext}(A)$.

(2) **Extensions at Worlds**

- The **extension** of a hyperintension $a : A$ at a world w, written $a@w : \mathrm{Ext}(A)$, is defined as follows, where $*$ denotes the unique inhabitant of 1:
  - This was already defined for $A = \mathrm{p}$
  - $a@w = a$ for $A = \mathrm{e}$
  - $*@w = *$
  - $\langle a, b\rangle\, @w = \langle a@w, b@w\rangle$
  - $a@w = \lambda_x.(a\ x)@w$ for $A = B \rightarrow C$.

(3) **Intensions**

- the **intension** of a hyperintension $a : A$ is $\mathbf{int}\ a =_{\mathrm{def}} \lambda_w.a@w : \mathrm{Int}(A)$
- a hyperintension has a lower type than the corresponding intension
- $\mathbf{int}_{\mathrm{p}} : \mathrm{p} \rightarrow \mathrm{w} \rightarrow \mathrm{t}$ coincides with the internal Stone mapping already defined
- Hence the family of morphisms $\mathbf{int}_A$ amounts to a generalized Stone mapping at all hyperintensional types.

Acknowledgments. The research reported here arose in connection with unpublished joint work with Shalom Lappin and Chris Fox. I am indebted to Jim Lambek and Phil Scott for their help in clarifying the problematic nature of Lambek's original proposal, and to Drew Moshier and Bill Lawvere for suggesting (separately) the use of a boolean algebra object of 'linguistic propositions' distinct from the subobject classifier.

References

1. Adams, R.: Theories of actuality. Noûs 8, 211–231 (1974)
2. Carnap, R.: Meaning and Necessity. University of Chicago Press, Chicago (1947)
3. Ajdukiewicz, K.: Die syntaktische Konnexität. Studia Philosophica 1, 1–27 (1935); English translation in McCall, S. (ed.) Polish Logic, 1920-1939, 207–231. Oxford University Press, Oxford
4. Bar-Hillel, Y.: A quasi-arithmetical notation for syntactic description. Language 29, 47–58 (1953)
5. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)
6. Cresswell, M.J.: Hyperintensional logic. Studia Logica 34, 25–48 (1975)
7. Diaconescu, R.: Axiom of choice and complementation. Proc. Amer. Math. Soc. 51, 176–178 (1975)
8. Fox, C., Lappin, S., Pollard, C.: A higher-order fine-grained logic for intensional semantics. In: Proceedings of the Seventh International Symposium on Logic and Language, Pécs, Hungary, pp. 37–46 (2002)
9. Frege, G.: On sense and reference. In: Geach, P., Black, M. (eds.) Translations from the Philosophical Writings of Gottlob Frege, 3rd edn., pp. 56–78. Basil Blackwell, Oxford (1980)
10. Gallin, D.: Intensional and Higher Order Modal Logic. North-Holland, Amsterdam (1975)
11. Goldblatt, R.: Topoi: the categorial analysis of logic. North-Holland, Amsterdam (1983)
12. de Groote, P.: Toward abstract categorial grammars. In: Proceedings of the 39th Annual Meeting and 10th Conference of the European Chapter of the Association for Computational Linguistics, pp. 148–155 (2001)
13. Henkin, L.: Completeness in the theory of types. Journal of Symbolic Logic 15, 81–91 (1950)
14. Johnstone, P.: Stone Spaces. Cambridge University Press, Cambridge (1982)
15. Jónsson, B., Tarski, A.: Boolean algebras with operators, part 1. American Journal of Mathematics 73(4), 891–939 (1951)
16. Kripke, S.: A completeness theorem in modal logic. Journal of Symbolic Logic 24, 1–14 (1959)
17. Kripke, S.: Semantic analysis of modal logic I: normal modal propositional calculi. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9, 67–96 (1963)
18. Lambek, J.: The Mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
19. Lambek, J.: Deductive systems and categories I: Syntactic calculus and residuated categories. Mathematical Systems Theory 2, 287–318 (1968)
20. Lambek, J.: Deductive systems and categories II: Standard constructions and closed categories. In: Dold, A., Eckmann, B. (eds.) Category Theory, Homology Theory, and their Applications. Springer Lecture Notes in Mathematics, vol. 86, pp. 76–122 (1969)
21. Lambek, J.: From λ-calculus to cartesian closed categories. In: Hindley, J., Seldin, J. (eds.) To H.B. Curry: Essays on Combinatorial Logic, Lambda Calculus, and Formalism, pp. 375–402. Academic Press, New York (1980)
22. Lambek, J.: Categorial and categorical grammars. In: Oehrle, R., Bach, E., Wheeler, D. (eds.) Categorial Grammars and Natural Language Structures. Reidel, Dordrecht (1988)

23. Lambek, J., Scott, P.J.: Introduction to Higher Order Categorical Logic. Cambridge University Press, Cambridge (1986)
24. Martin, S., Pollard, C.: Dynamic hyperintensional semantics: enriching contexts for type-theoretic discourse analysis. In: 15th International Conference on Formal Grammar (FG 2010), Copenhagen. Springer Lecture Notes in Artificial Intelligence, (August 2010) (in press)
25. Martin, S., Pollard, C.: Under revision. A higher-order theory of presupposition. To appear in Special Issue of Studia Logica on Logic and Natural language
26. Montague, R.: The proper treatment of quantification in English. In: Thomason, R. (ed.) Formal Philosophy: Selected Papers of Richard Montague, pp. 247–270. Yale University Press, New Haven (1974)
27. Muskens, R.: Sense and the computation of reference. Linguistics and Philosophy 28(4), 473–504 (2005)
28. Pollard, C.: Hyperintensions. Journal of Logic and Computation 18(2), 257–282 (2008)
29. Pollard, C.: Remarks on categorical semantics of natural language. Invited talk presented at the Workshop on Logic, Categories, and Semantics, Bordeaux (November 2010),
http://www.ling.ohio-state.edu/~pollard/lcs/remarks.pdf
30. Stalnaker, R.: Inquiry. Bradford Books/MIT Press, Cambridge, MA (1984)
31. Stone, M.: The theory of representation for boolean algebras. Transactions of the American Mathematical Society 40, 37–111 (1936)
32. Stone, M.: Topological representation of distributive lattices and Brouwerian logics. Časopis pešt. mat. fys. 67, 1–25 (1937)
33. Thomason, R.: A model theory for propositional attitudes. Linguistics and Philosophy 4, 47–70 (1980)

Event in Compositional Dynamic Semantics

Sai Qian[1,2,3] and Maxime Amblard[1,2]

[1] LORIA & INRIA Nancy Grand-Est - BP 239 - 54506 Vandœuvre-lès-Nancy Cedex
[2] Nancy 2 University, 13 Rue du Marchal Ney - 54037 Nancy Cedex
[3] Nancy 1 University, 24-30 Rue Lionnois - BP 60120 | 54003 Nancy Cedex
{sai.qian,maxime.amblard}@loria.fr

Abstract. We present a framework which constructs an event-style discourse semantics. The discourse dynamics are encoded in continuation semantics and various rhetorical relations are embedded in the resulting interpretation of the framework. We assume discourse and sentence are distinct semantic objects, that play different roles in meaning evaluation. Moreover, two sets of composition functions, for handling different discourse relations, are introduced. The paper first gives the necessary background and motivation for event and dynamic semantics, then the framework with detailed examples will be introduced.

Keywords: Event, Dynamics, Continuation Semantics, DRT, Discourse Structure, Rhetorical Relation, Accessibility, λ-calculus.

1 Event Semantics

The idea of relating verbs to certain events or states can be found throughout the history of philosophy. For example, a simple sentence *John cries* can be referred to a *crying* action, in which *John* is the agent who carries out the action. However, there were no real theoretical foundations for semantic proposals based on events before [5]. In [5], Davidson explained that a certain class of verbs (action verbs) explicitly imply the existence of underlying events, thus there should be an implicit event argument in the linguistic realization of verbs.

For instance, traditional Montague Semantics provides *John cries* the following interpretation: $Cry(john)$, where Cry stands for a 1-place predicate denoting the *crying* event, and $john$ stands for the individual constant in the model. Davidson's theory assigns the same sentence another interpretation: $\exists e.Cry(e, john)$, where Cry becomes a 2-place predicate taking an event variable and its subject as arguments.

Later on, based on Davidson's theory of events, Parsons proposed the Neo-Davidsonian event semantics in [19]. In Parsons' proposal, several modifications were made. First, event participants were added in more detail via thematic roles; second, besides action verbs, state verbs were also associated with an abstract variable; furthermore, the concepts of holding and culmination for events, the decomposition of events into subevents, and the modification of subevents by adverbs were investigated.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 219–234, 2011.

As explained in [19], there are various reasons for considering event as an implicit argument in the semantic representation. Three of the most prominent ones being adverbial modifiers, perception verbs and explicit references to events.

Adverbial modifiers of a natural language sentence usually bear certain logical relations with each other. Two known properties are *permutation* and *drop*. Take Sentence (1) as an example.

(1) John buttered the toast slowly, deliberately, in the bathroom, with a knife, at midnight.

Permutation means the truth conditions of the sentence will not change if the order of modifiers are alternated, regardless of certain syntactic constraints; drop means if some modifiers are eliminated from the original context, the new sentence should always be logically entailed by the original one. In Parsons' theory, the above sentence is interpreted as $\exists e.(Butter(e) \wedge Subject(e, john) \wedge Object(e, toast) \wedge Slow(e) \wedge Deliberate(e) \wedge In(e, bathroom)...)$. A similar treatment for adjectival modifiers can also be found in the literature. Compared with several other semantic proposals, such as increasing the arity of verbs or higher order logic solutions, event is superior.

Aside from modifiers, perception verbs form another piece of evidence for applying event in semantic representations. As their name suggests, perception verbs are verbs that express certain perceptual aspects, such as *see*, *hear*, $feel$, and etc. The semantics of sentences that contain perception verbs are quite different from those whose sub-clauses are built with *that* construction. For instance, we can interpret *see* in three different ways[1]:

1. sb. **see** sb./sth.: $e \rightarrow e \rightarrow t$, e.g., *Mary sees John.*
2. sb. **see** some fact: $e \rightarrow t \rightarrow t$, e.g., *Mary sees that John flies.*
3. sb. **see** some event: $e \rightarrow v \rightarrow t$, e.g., *Mary sees John fly.*

As the example shows, the first *see* just means somebody sees somebody or something. The second *see* indicates that Mary sees a fact, the fact is *John flies*. Even if Mary sees it from TV or newspaper, the sentence is still valid. The third sentence, in contrast, is true only if Mary directly perceives the event of *John flying* with her own sight.

Furthermore, natural language discourses contain examples of various forms of explicit references (mostly the *it* anaphor) to events, for example, *John sang on his balcony at midnight.* ***It*** *was horrible.*

2 Dynamic Semantics and Discourse Relation

2.1 Dynamic Semantics

In the 1970s, based on the principle of compositionality, Richard Montague combined First Order Logic, λ-calculus, and type theory into the first formal natural

[1] "e" and "t" are the same as in traditional Montague Semantics, while "v" stands for a new type for event.

language semantic system, which could compositionally generate semantic representations. This framework was formalized in [16], [17], and [18]. By convention, it is named Montague Grammar (MG).

However, despite its huge influence in semantic theory, MG was designed to handle single sentence semantics. Later on, some linguistic phenomena, such as anaphora, donkey sentences, and presupposition projection began to draw people's attention from MG to other approaches, such as dynamic semantics, which has a finer-grained conception of meaning. By way of illustration, we can look at the following "donkey sentence", which MG fails to explain:

(2) a. A farmer$_1$ owns a donkey$_2$. He$_1$ beats it$_2$.
b. *Every farmer$_1$ owns a donkey$_2$. He$_?$ beats it$_?$.

In the traditional MG, the meaning of a sentence is represented as its truth conditions, that is the circumstances in which the sentence is true. However, in dynamic semantics, the meaning of a sentence is its context change potential. In other words, meaning is not a static concept any more, it is viewed as a function that always builds new information states out of the old ones by updating the current sentence. Some of the representative works, which emerged since the 1980s, include File Change Semantics [10], Discourse Representation Theory (DRT) [12], and Dynamic Predicate Logic (DPL) [6].

2.2 A New Approach to Dynamics

Recently in [8], de Groote introduced a new framework, which integrates a notion of context into MG, based only on Church's simply-typed λ-calculus. Thus the concept of discourse dynamicscan be embedded in traditional MG without any other specific definitions as is the case in other dynamic systems.

In DRT, the problem of extending quantifier scope is tackled by introducing sets of reference markers. These reference markers act both as existential quantifiers and free variables. Because of their special status, variable renaming is very important when combining DRT with MG. The framework in [8] is superior in the computational aspect because the variable renaming has already been solved with the simply typed λ-calculus. Further more, every new sentence is only processed under the environment of the previous context in DRT, but [8] proposed to evaluate a sentence based on both left and right contexts, which would be abstracted over its meaning.

In Church's simple type theory, there are only two atomic types: "ι", denoting the type of individual; "o", denoting the type of proposition[2]. The new approach adds one more atomic type "γ", to express the left contexts, thus the notion of dynamic context is realized. Consequently, as the right context could be interpreted as a proposition given its left context, its type should be $\gamma \to o$. For the same reason, the whole discourse could be interpreted as a proposition given both its left and right contexts. Assuming s and t is respectively the syntactic

[2] Here we follow the original denotation in [8], but actually there is no great difference between "ι", "o" (Church's denotation) and "e", "t" (Montague's denotation).

category for sentence and discourse, their semantic interpretations are:

$$[\![s]\!] = \gamma \to (\gamma \to o) \to o, [\![t]\!] = \gamma \to (\gamma \to o) \to o$$

In order to conjoin the meanings of sentences to obtain the composed meaning of a discourse, the following formula is proposed:

$$[\![D.S]\!] = \lambda e\phi.[\![D]\!]e(\lambda e'.[\![S]\!]e'\phi) \tag{1}$$

in which D is the preceding discourse and S is the sentence currently being processed. The updated context $D.S$ also possesses the same semantic type as D and S, it has the potential to update the context. Turning to DRT, if we assume "$x_1, x_2, \cdots$" are reference markers, and "$C_1, C_2, \cdots$" are conditions, the corresponding λ-term for a general DRS in the new framework should be:

$$\lambda e\phi.\exists x_1 \cdots x_n.C_1 \wedge \cdots C_m \wedge \phi e'$$[3]

To solve the problem of anaphoric reference, [8] introduced a special choice operator. The choice operator is represented by some oracles, such as $sel_{he}, sel_{she}, ...$ It takes the left context as argument and returns a resolved individual. In order to update the context, another operator "::" is introduced, which adds new accessible variables to the processed discourse. For instance, term "$a :: e$" actually is interpreted as "$\{a\} \bigcup e$" mathematically. In other words, we can view the list as the discourse referents in DRT.

Finally, let us look at a compositional treatment of Discourse (3) according to the above formalism. The detailed type and representation for each lexical entry is presented in the following table:

Word	Type	Semantic Interpretation
John/Mary	$(\iota \to [\![s]\!]) \to [\![s]\!]$	$\lambda\psi e\phi.\psi\mathbf{j/m}e(\lambda e.\phi(\mathbf{j/m} :: e))$
she		$\lambda\psi e\phi.\psi(sel_{she}e)e\phi$
kisses	$[\![np]\!] \to [\![np]\!] \to [\![s]\!]$	$\lambda os.s(\lambda x.o(\lambda ye\phi.Kiss(x,y) \wedge \phi e))$
smiles	$[\![np]\!] \to [\![s]\!]$	$\lambda s.s(\lambda xe\phi.Smile(x) \wedge \phi e)$

(3) John kisses Mary. She smiles.

$([\![kisses]\!][\![Mary]\!])[\![John]\!] \Rightarrow_\beta \lambda e\phi.Kiss(j,m) \wedge \phi(m :: j :: e)$
$[\![smile]\!][\![she]\!] \Rightarrow_\beta \lambda e\phi.Smile(sel_{she}(e)) \wedge \phi(e)$
$[\![D.S]\!] \Rightarrow_\beta \lambda e\phi.(Kiss(j,m) \wedge Smile(sel_{she}(j :: m :: e)) \wedge \phi(j :: m :: e))$

2.3 Discourse Relations and Discourse Structure

Since the emergence of dynamic semantics, people have been changing their opinion on the notion of meaning. Based on that, many researchers working on multiple-sentence semantics have studied an abstract and general concept:

[3] Here, "e'" is a left context made of "e" and the variables "$x_1, x_2, x_3 \cdots$". Its construction depends on the specific structure of the context, for more details see [8].

discourse structure, in other words, the rhetorical relations, or coherence relations ([11], [15], [1]). Representative theories include Rhetorical Structure Theory (RST) and Segmented Discourse Representation Theory (SDRT). The idea that an internal structure exists in discourse comes naturally. Intuitively, in order for a context to appear natural, its constituent sentences should bear a certain coherence with each other, namely discourse relations (DRs). That is also why it is not the case that any two random sentences can form a natural context.

It is still an open question to identify all existing DRs. But it is generally agreed there are two classes of DRs, namely the coordinating relations and the subordinating relations. The former includes relations like Narration, Background, Result, Parallel, Contrast, etc., while relations such as Elaboration, Topic, Explanation, and Precondition belong to the latter type. The distinction of two classes of DRs also has intuitive reasons.

For instance, the function of a sentence over its context could be to introduce a new topic or to support and explain a topic. Thus the former plays a subordinate role, and the latter plays a coordinate role together with those that function in the similar way (supporting or explaining). In addition, it is a even more complicated task to determine which DRs belong to which class. [3] provides some linguistic tests to handle this problem and analyzes some deeper distinctions between these two classes.

The reason that we introduced different types of DRs is because we can construct a more specific discourse hierarchy based on it. The hierarchy can aid in the resolution of some semantic or pragmatic phenomena like anaphora. The original theoretical foundation of this idea dates back to [20], which says that in a discourse hierarchy, only constituents at *accessible nodes* can be integrated into the updated discourse structure. By convention, a subordinating DR creates a vertical edge and coordinating DR a horizontal edge. The accessible nodes are all located on the right frontier in the hierarchy. This is also known as the *Right Frontier Constraint*. For instance, in Figure 1, $Event_1$, $Event_3$ and $Event_5$ are on the right frontier, so they stay accessible for further attachments. However, $Event_2$ and $Event_4$ are blocked, which indicates that variables in these two nodes cannot be referenced by future anaphora.

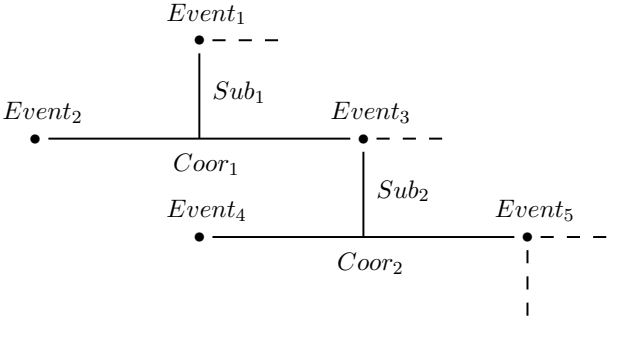


Fig. 1. Graph Structure Example

So far, we are clear about the fact that discourses do have structures. By comparing with other dynamic semantic treatments of phenomena such as pronouns and tense, we can identify advantages of using DRs. For further illustration, we use the example from [13]:

(4) a. John had a great evening last night.
b. He had a great meal.
c. He ate salmon.
d. He devoured lots of cheese.
e. He won a dancing competition.
f. *It was a beautiful pink.

Traditional dynamic semantic frameworks, such as DRT, will totally accept Discourse (4), because there is no universal quantification or negation to block any variable. The pronoun *it* in (4-f) can either refer back to *meal*, or *salmon*, or *cheese*, or *competition*. Normally *pink* will only be used to describe *salmon*, which is in the candidate list. However, sentence (4-f) does sound unnatural to English-speaking readers. Here discourse structure can help to explain. If we construct the discourse hierarchy according to different types of DRs introduced above, we obtain the graph in Figure 2.

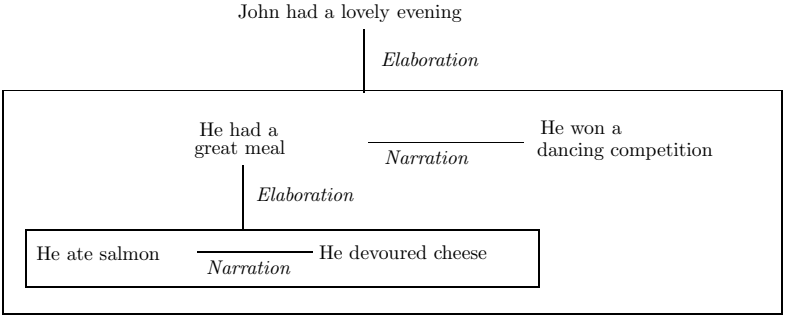


Fig. 2. Discourse Hierarchy of (4)

Thus, it is clear that (4-f) is not able to be attached to (4-c), where *salmon* is located. Relation between (4-c) and (4-d) is a Narration, which is of coordinating type, so (4-c) is blocked for further reference. In addition, many linguistic phenomena other than anaphora can be better explained with discourse structure and the right frontier constraint, such as presupposition projection, definite descriptions, and word sense disambiguation.

3 Event in Dynamic Semantics

So far, we have first presented the advantages of using events in semantic analysis over traditional MG, then the motivation for dynamics in discourse semantics,

and finally the need for DRs in more subtle semantic processing. In this section, we propose a framework that compositionally constructs event-style discourse semantics, with various DRs and the accessibility constraint (right frontier constraint) embedded. First, we explain how to build representations of single sentences. After that, the meaning construction for discourse, which is based on its component sentences, will be presented.

3.1 Event-Based Sentential Semantics

As we showed in Section 1, the implicit event argument helps to handle many linguistic phenomena, such as adverbial modifications, sentential anaphora (*it*) resolution. With the notion of thematic relations, the verb predicate will take only one event variable as argument, instead of multiple variables, each representing a thematic role relation. Most of the current theories only describe event semantics from a philosophical or pure linguistic point of view, and the corresponding semantic representations are provided without concrete computational constructions. That is what our proposal focuses on. Before we introduce our framework, some assumptions need to be specified.

Thematic roles have been used formally in literature since [9]. However, to determine how many thematic roles are necessary is still an open question. In addition, indicating exactly which part of a sentence correlates to which thematic role is also a difficult task. In our framework, we only consider the most elementary and the most widely accepted set of thematic roles. The roles and their corresponding syntactic categories are listed in the following table:

Thematic Role	**Syntactic Correspondence**
Agent	Subject
Theme	Direct object; subject of "*be*"
Goal	Indirect object, or with "*to*"
Benefactive	Indirect object, or with "for"
Instrument	Object of "*with*"; subject
Experiencer	Subject
Location/Time	With "*in*" or "*at*"

In [19], the author provides a template-based solution to construct semantic representations with events. Sentences are first classified into different cases based on their linguistic properties, such as *passive*, *perceptive*, *causative*, *inchoative*, etc. Then a unique template is assigned to each case; the number, types and positions of arguments are specifically designed for that template. In our proposal, we will also use templates, but a much simpler version. The templates only contain the most basic thematic roles for certain verbs. They are subject to modification and enhancement for more complicated cases. For instance, the template for the verb *smile* only contains one agent role, while the verb *kiss* contains both agent and theme roles.

Furthermore, our proposal will generalize the ontology of event variables. So to speak, at the current stage we make no distinction between *events*, *states* and

processes, for the sake of simplicity (their linguistic differences are described in [19]). So there will just be one unique variable, representing the underlying event, or state, or process, indicated by the verb. There is a simple example:

(5) John kisses Mary in the plaza.

Under event semantics, the semantic representation for Sentence (5) should be:

$$\exists e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza))^{4}.$$

In order to obtain the above representation compositionally, we use the following semantic entries for words in the lexicons:

$$[\![John]\!] = john \qquad [\![Mary]\!] = mary$$
$$[\![kiss]\!] = \lambda ose.(Kiss(e) \wedge Ag(e, s) \wedge Pat(e, o))$$
$$[\![in\_the\_plaza]\!] = \lambda Pe.(P(e) \wedge Loc(e, plaza))^{5}$$

Thus, by applying the above four entries to one another in a certain order[6], we can compute the semantic representation of (5) step by step:

1. $[\![kiss]\!][\![Mary]\!]$
   $\Rightarrow_\beta \lambda se.(Kiss(e) \wedge Ag(e, s) \wedge Pat(e, mary))$
2. $([\![kiss]\!][\![Mary]\!])[\![John]\!]$
   $\Rightarrow_\beta \lambda e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary))$
3. $[\![in\_the\_plaza]\!](([\![kiss]\!][\![Mary]\!])[\![John]\!])$
   $\Rightarrow_\beta \lambda Pe.(P(e) \wedge Loc(e, plaza))(\lambda e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary)))$
   $\Rightarrow_\beta \lambda e'.(\lambda e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary)))(e') \wedge Loc(e, plaza)$
   $\Rightarrow_\beta \lambda e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza))$

At this point, the event variable "e" is not yet instantiated as an existential quantifier. To realize that, we can simply design an EOS (End Of Sentence) operator[7], to which the partial sentence representation could be applied:

$$[\![EOS]\!] = \lambda P.\exists e.P(e)$$

As a result, the last step would be:

4. $[\![EOS]\!]([\![in\_the\_plaza]\!](([\![kiss]\!][\![Mary]\!])[\![John]\!]))$
   $\Rightarrow_\beta \lambda P.\exists e.P(e)(\lambda e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza)))$
   $\Rightarrow_\beta \exists e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza))$

In the above solution, the adverbial modifier *in the plaza* is handled in the manner that is traditional for intersective adjectives (e.g., *tall*, *red*). With a similar formalism, any number of intersective adverbial modifiers can be added

[4] *Ag* stands for Agent, *Pat* for Patient and *Loc* for Location.

[5] It is of course possible to break down the interpretation construction of "*in the plaza*" into a more detailed level by providing entries for each word, but we give the compound one for the whole PP just for simplification.

[6] The function-argument application can be obtained via shallow syntactic processing.

[7] This could be a comma, full stop, exclamation point, or any other punctuation marks.

to the event structure, as long as the corresponding lexical entry is provided for each modifier.

The event variable, which is embedded in the verb, is the greatest difference between our framework and MG. From a computational point of view, we need to pass the underlying event variable from the verb to other modifiers. As a result, we first use the λ-operator for the event variable in the verb interpretation, then the EOS to terminate the evaluation and instantiate the event variable with an existential quantifier. Another framework which compositionally obtain an event-style semantics is [4], which introduces the existential quantifier for the event at the beginning of interpretation.

3.2 Event-Based Discourse Semantics

In the previous part, we showed how to compute single sentence semantics with events. In this section we will combine event structure with dynamic semantics, extending our formalism to discourse.

As explained in Section 2.2, [8] expresses dynamics in MG by introducing the concept of left and right contexts. We adopt the idea, inserting the left and right contexts into our semantic representations. Thus we bestow upon our event-based formalism the potential to be updated as in other dynamic systems. To achieve this, we modify the lexical entries in the previous section as following:

$$[\![John]\!] = john \qquad [\![Mary]\!] = mary$$
$$[\![kiss]\!] = \lambda oseab.(Kiss(e) \wedge Ag(e,s) \wedge Pat(e,o) \wedge b(e :: a))$$
$$[\![in\_the\_plaza]\!] = \lambda Peab.(Peab \wedge Loc(e, plaza))$$

Here, in contrast to the notation used in [8], "a" stands for the left context and "b" stands for the right context. In our logical typing system, we use type "v" for the event variable, and type "α" for the left context. Types "e" and "t" have the same meaning as convention. In an additional departure from the formalism in [8], we assume the left context contains the accessibility information of previous event variables, instead of individual variables. That is why we keep using the original interpretations for $John$ and $Mary$, instead of inserting the constants "$john$" and "$mary$" in the left context list structure. However, the list constructor "::" does have a similar meaning. The only difference with the constructor in [8] is that our "::" takes an event variable and the left context as arguments, while the previous one takes an individual variable and the left context. Given the lexical entries above, the semantic representation with a dynamic potential for Sentence (5) becomes:

1. $[\![in\_the\_plaza]\!](([\![kiss]\!][\![Mary]\!])[\![John]\!])$
   $\Rightarrow_\beta \lambda eab.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza) \wedge b(e :: a))$

Its semantic type also changes from "$v \to t$" into "$v \to (\alpha \to (\alpha \to t) \to t)$"[8]. In order to terminate the semantic processing, we need a new EOS symbol:

$$[\![EOS]\!] = \lambda P.\exists e.PeAB$$

[8] "$\alpha \to t$" is the type for the right context, represented by variable "b".

Thus we obtain the final interpretation:

2. $[\![EOS]\!]([\![in\_the\_plaza]\!](([\![kiss]\!][\![Mary]\!])[\![John]\!]))$
   $\Rightarrow_\beta \exists e.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza) \wedge B(e :: A))$

The "A" and "B" in the EOS and above formula are not variables any more. They are just constants of type "α" and "$\alpha \rightarrow t$", respectively, which have the effect of freezing the left and right contexts. We can see that the new representation does not seem different from the previous version. That is, of course, because although we embed the dynamic potential into the entries, we are still evaluating single sentence semantics. The power of dynamics will not show up until the case becomes more complicated. So let us consider the following discourse:

(6) a. John kisses Mary in the plaza.
    b. She smiles.

To handle Example (6), we need to provide two more entries:

$$[\![she]\!] = \lambda Peab.P(Sel(a))eab$$
$$[\![smile]\!] = \lambda seab.(Smile(e) \wedge Ag(e, s) \wedge b(e :: a))$$

Inspired by [8], the interpretation of *she* is made by an external function: Sel. This function is supposed to work over a structured representation of the discourse: we claim that individual variables are defined in the scope of event variables. Thus the resolution of this anaphora must be first do by picking out an event variable, and, through this event, choose the correct individual variable following the previous.

We also apply a type-raising representation for NP (*she*), because we need to pass the selection function Sel for further processing. Similar type-raising version of $John$ and $Mary$ could also be constructed. After type raising, the only thing that needs to be changed is the order of argument application, and the resulting logic term will exactly be the same.

So, getting back to Discourse (6), we can first obtain the representations for (6-a) and (6-b) independently:

1. $[\![in\_the\_plaza]\!](([\![kiss]\!][\![Mary]\!])[\![John]\!])$
   $\Rightarrow_\beta \lambda eab.(Kiss(e) \wedge Ag(e, john) \wedge Pat(e, mary) \wedge Loc(e, plaza) \wedge b(e :: a))$
2. $[\![she]\!][\![smile]\!]$
   $\Rightarrow_\beta \lambda eab.(Smile(e) \wedge Ag(e, Sel(a)) \wedge b(e :: a))$

Now the problem is how to combine the two interpretations to yield the discourse semantics. [8] uses Formula 1 to merge sentence interpretations, which takes the previous discourse and the current sentence as input, returns a new piece of updated discourse. Same as DRT, there is no rhetorical relation involved in [8]. However, this paper goes one step further, aiming to encode the discourse structure and event accessibility relations between different sentences.

Hence, we make another assumption here: *discourse* and *sentence* are distinct semantic entities, they have different types and meaning evaluations. Every discourse contains certain rhetoric relations, while single sentences should be able

to be interpreted without those relations. That is because we consider the discourse structure as a production from sentence composition. Unlike in Formula 1, where discourse D and sentence S have exactly the same semantic properties, we assign them different types. Drawing from the above assumption, the event variables should be instantiated into existential quantifier in discourse only, while they are still of λ-forms in sentences. By way of illustration, the followings are the most general representations for sentence and discourse:

$$[\![S]\!] = \lambda eab.(Pred(e) \wedge ... \wedge ba)$$
$$[\![D]\!] = \lambda ab.\exists e_1 e_2 ...(Pred_1(e_1) \wedge Pred_2(e_2) \wedge ... \wedge Rel_1(e_i, e_j) \wedge Rel_2(e_m, e_n) \wedge ... \wedge ba')$$[9]

Please note that the interpretation for discourse does not only contain "a'", where accessibility relations are located; but also various rhetoric relations, represented by Rel_1, Rel_2 and so on. Those rhetoric relations, as we discussed in Section 2.3, can be classified into either subordinating or coordinating. They have completely different effects in shaping the discourse structure graphs, which determines the accessibility relations. Here we do not care about how many different discourse relations there are (such as Narration, Background, Elaboration, etc.), we just assume if there is a relation, it must belong to one of the two classes. And those rhetoric relations are added only during the meaning merging process. As a consequence, we propose two sets of composition functions, according to different types of DRs.

Subordinating Composition Functions. Based on the right frontier constraint, for those discourses and sentences which are connected by subordinating DRs, all accessible nodes in the previous discourse remain the same, meanwhile the new sentence will be inserted as accessible in the updated discourse. For example in Figure 3, when $Event_6$ is added into the discourse by a subordinating relation with $Event_5$, the current accessible nodes include $Event_1$, $Event_3$, $Event_5$ and $Event_6$. Hence, we introduce the composition functions for subordinating DRs as follows:

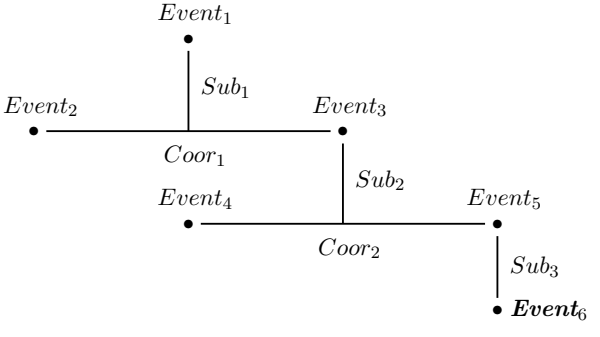


Fig. 3. Graph Structure with Subordinating Relations

[9] The left context "a'" in the representation is a complicated structure containing the event accessibility relation. There will be further examples showing how to create "a'" from "a" and other event variables.

$$[\![Sub_{Bas}]\!] = \lambda DSab.Da(\lambda a'.\exists e.(Sea'b)) \tag{2}$$

$$[\![Sub_{Adv}]\!] = \lambda DSab.Da(\lambda a'.\exists e.((Sea'b) \wedge Rel(Sel(a'), e)) \tag{3}$$

We suppose that every sentence needs to combine with a previous discourse to form a new discourse, also including the first sentence in the context. However, the first sentence could only be combined with an empty discourse:

$$[\![Empty]\!] = \lambda ab.ba$$

which in fact contains no context information at all, it is created just for computational reason. That's why we design two composition functions 2 and 3, namely the Sub_{Bas} and the Sub_{Adv}, to respectively handle the first sentence case and all other situations. Now we will construct the interpretation of (6), as an illustration for our composition functions. Suppose (6-a) and (6-b) hold a subordinating relation between each other[10], then in order to obtain the whole representation for (6), we first need to combine (6-a) with the empty discourse by Sub_{Bas}, then combine the result with (6-b) by Sub_{Adv}.

1. $[\![Sub_{Bas}]\!][\![Empty]\!][\![(6\text{-a})]\!]$[11]
   $\Rightarrow_\beta \lambda a_1 b_1.(\lambda a_3 b_3.b_3 a_3) a_1 (\lambda a_2.\exists e.(\lambda e' a_4 b_4.(Kiss(e') \wedge ... \wedge b_4(e' :: a_4)) e a_2 b_1))$
   $\Rightarrow_\beta \lambda a_1 b_1.(\lambda b_3.b_3 a_1)(\lambda a_2.\exists e.(Kiss(e) \wedge ... \wedge b_1(e :: a_2)))$
   $\Rightarrow_\beta \lambda a_1 b_1.\exists e.(Kiss(e) \wedge ... \wedge b_1(e :: a_1))$

This step does two things. First, it instantiates the event variable from (6-a) into an existential quantifier. In addition, it inserts the new event argument into the accessible list of the left context. Because the empty discourse does not contain any variable in its left context, the list construction is fairly simple, we just need a naive "push-in" operation.

2. $[\![Sub_{Adv}]\!]([\![Sub_{Bas}]\!][\![Empty]\!][\![(6\text{-a})]\!])[\![(6\text{-b})]\!]$
   $\Rightarrow_\beta \lambda a_1 b_1.(\lambda a_3 b_3.\exists e_1.(Kiss(e_1) \wedge ... \wedge b_3(e_1 :: a_3))) a_1 (\lambda a_2.\exists e.(((\lambda e_2 a_4 b_4.$
   $(Smile(e_2) \wedge ... \wedge b_4(e_2 :: a_4))) e a_2 b_1) \wedge Rel(Sel(a_2), e)))$
   $\Rightarrow_\beta \lambda a_1 b_1.(\lambda b_3.\exists e_1.(Kiss(e_1) \wedge ... \wedge b_3(e_1 :: a_1)))(\lambda a_2.\exists e.(Smile(e) \wedge ... \wedge b_1(e ::$
   $a_2) \wedge Rel(Sel(a_2), e)))$
   $\Rightarrow_\beta \lambda a_1 b_1.\exists e_1.(Kiss(e_1) \wedge ... \wedge \exists e.(Smile(e) \wedge ... \wedge b_1(e :: e_1 :: a_1) \wedge Rel(Sel(e_1 ::$
   $a_1), e)))$
   $= \lambda a_1 b_1.\exists e_1 e_2.(Kiss(e_1) \wedge ... \wedge Smile(e_2) \wedge ... \wedge b_1(e_2 :: e_1 :: a_1) \wedge Rel(Sel(e_1 ::$
   $a_1), e_2))$

Suppose the selection function Sel is able to pick the correct event variable out of the accessible list, then our desired DRs and accessibility relation would be successfully encoded in the final logic formula. There are two more remarks for

[10] Here is just an assumption, our system does not account how to determine the DRs, we only focus on encoding those relations.

[11] We omit some internal thematic structures for (6-a) just for a clear view of the logic terms. The same omission will also be carried out for (6-b).

the subordinating composition functions: 1. no new event variable is created during the meaning composition, but all event variables with the λ-operator will be instantiated as existential quantifiers; 2. the composing process will not change the accessibility condition in the previous discourse, only a new accessible node is added.

Coordinating Composition Functions. Again, let's first analyze the effect of coordinating DRs on accessibility structure. When a new node is added to an existing discourse with coordinating relation, a horizontal edge is built, as shown in Figure 4, $Event_6$ and $Event_5$ for example. At the same time, an abstract

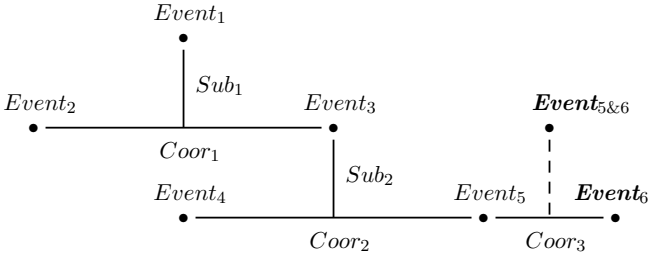


Fig. 4. Graph Structure with Coordinating Relations

variable node - $Event_{5\&6}$, is created. This is a distinct property compared to subordinating DRs. We need the new abstract node because in many cases the anaphora *it* could only be resolved with reference to a set of sentences connected with coordinating relations, as in:

(7) Mary stumbled her ankle. She twisted it. John did so too.

To see more examples, see [14].

Based on the above analysis, we propose the following composition functions:

$$[\![Coor_{Bas}]\!] = \lambda DSab.Da(\lambda a'.\exists e.(Sea'b)) \tag{4}$$

$$[\![Coor_{Adv}]\!] = \lambda DSab.\exists e_c.Da(\lambda a'.\exists e.(Se(e_c :: (Del(a')))b) \wedge Rel(Sel(a'), e, e_c)) \tag{5}$$

Notice that Formula 4 is identical to 2 because both basic composition functions are designed only to handle the first sentence case, in which we do not really need to distinguish from different DRs (there is even no DR at all). In contrast to Formula 3, the advanced subordinating function, there are three main differences in Formula 5. First, apart from instantiating the event variable of current sentence, another abstract event variable "e_c" is created. It is directly inserted into the accessible list because the new abstract node will always be at the right frontier in the updated discourse structure. Moreover, we introduce a new function *Del*, which takes the current accessible list as argument, and deletes those nodes which will no longer be accessible in the new discourse. It works in a similar way as the *Sel* function. Finally, the *Rel* function takes three

arguments, including the abstract variable. By doing this we can keep track of the relation between abstract variables and their component nodes.

Now let us use Discourse (6) again as an illustration. This time we assume the rhetoric relation between (6-a) and (6-b) is of a coordinating kind. Thus we will build its semantic representation with 4 and 5.

1. $[\![Coor_{Bas}]\!][\![Empty]\!][\![\text{(6-a)}]\!] = [\![Sub_{Bas}]\!][\![Empty]\!][\![\text{(6-a)}]\!]$
   $\Rightarrow_\beta \lambda a_1 b_1.\exists e.(Kiss(e) \wedge ... \wedge b_1(e :: a_1))$
2. $[\![Coor_{Adv}]\!]([\![Coor_{Bas}]\!][\![Empty]\!][\![\text{(6-a)}]\!])[\![\text{(6-b)}]\!]$
   $\Rightarrow_\beta \lambda S a_1 b_1.\exists e_c.(\lambda a_3 b_3.\exists e_1.(Kiss(e_1) \wedge ... \wedge b_3(e_1 :: a_3)))a_1(\lambda a_2.\exists e.((\lambda e_2 a_4 b_4.$
   $(Smile(e_2) \wedge ... \wedge b_4(e_2 :: a_4)))e(e_c :: (Del(a_2)))b_1) \wedge Rel(Sel(a_2), e, e_c))$
   $\Rightarrow_\beta \lambda S a_1 b_1.\exists e_c.(\lambda b_3.\exists e_1.(Kiss(e_1) \wedge ... \wedge b_3(e_1 :: a_1)))(\lambda a_2.\exists e.(Smile(e) \wedge ... \wedge$
   $b_1(e :: e_c :: (Del(a_2)))) \wedge Rel(Sel(a_2), e, e_c))$
   $\Rightarrow_\beta \lambda S a_1 b_1.\exists e_c e_1 e_2.(Kiss(e_1) \wedge ... \wedge Smile(e_2) \wedge ... \wedge b_1(e_2 :: e_c :: (Del(e_1 ::$
   $a_1))) \wedge Rel(Sel(e_1 :: a_1), e_2, e_c))$

As we can see from the final formula, the new event variable "e_2" and the abstract variable "e_c" are added into the accessible list. *Del* will then eliminate the inaccessible node "e_1", leaving only "e_2" and "e_c" on the right frontier.

To test the validity of the proposed system, we have implemented all the above calculus in the Abstract Categorial Grammar [7].

3.3 Comparison with Other Related Works

Recently there are some other semantic frameworks based on discourse structure, DRT and other dynamic concepts. For example in [2], the authors expressed SDRT in a non-representational way with dynamic logic. Similar to the formalism presented in our paper, they also use the continuation calculus from [8], where the concepts of left and right contexts are involved for introducing dynamics. However, there are some distinctions between our work and theirs.

First of all, we use an event-style semantics for meaning representation. Consequently, the basic construct of rhetorical relation in our framework is event, in contrast with the discourse constituent unit (DCU) in [2]. Event-based theory, as an independent branch of formal semantics, has been studied since a long time ago. Many lexical properties (mainly for verbs), such as tense and aspect, causative and inchoative, etc., have already been investigated in detail. By using event here, we can borrow many off-the-shelf results directly. Also, the DCUs, which are notated by π in other discourse literatures, are not as concretely defined as events. There are cases where a single DCU contains multiple events. For instance, "*John says he loves Mary. Mary does not believe it.*". Only with DCU, the resolution for *it* in the second sentence will cause ambiguity.

In addition, we and [2] make different assumptions over *discourse* and *sentence*. The same way as in [8], [2] views the discourse and sentence as identical semantic construct. However, as explained in Section 3.2, we do distinguish them as different objects. When encountering a single sentence, we should interpret it independently, without considering any discourse structure. While discourse

is not simply a naive composition of component sentences. It should be their physical merging with various DRs added.

Finally, the DRs originate from different sources in the two works. [2] uses key words as DR indicator. For example, in discourse "*A man walked in, then he coughed.*", [2] embeds the *Narration* relation in the interpretation of *then*. However, we believe that the DRs only be revealed when sentence and discourse are combined, they should not emerge in sentence interpretations. So DRs are presented in the composition functions (Formula 2, 3, 4 and 5) in our work.

4 Conclusion and Future Work

In this paper, we have represented the accessibility relations of natural language discourse within event semantics. This approach does not depend on any specific logic, all formulas are in the traditional MG style.

We decide to use event-based structure because it is able to handle sentential anaphora resolution (e.g., *it*), adverbial modifiers and other semantic phenomena. Also, applying dynamics to event semantics may largely extend its power, which was originally developed to treat single sentences. As we know, the accessibility among sentences in discourse depends on various types of DRs. However, these DRs are usually hard to determine. We assume all DRs be classified into two types: subordinating and coordinating. Also we obtain the accessibility relation with the right frontier constraint. Based on that, we encode these DRs and the updating potential for single sentences in a First Order Logic system.

In our approach, we differentiate discourse and sentence as two distinct semantic objects. The DRs are only added during the updating process, which is realized through the set of composition functions. This choice not only has computational, but also philosophical evidences.

In this paper, we only focus on representing the DRs and accessibility in logical forms, but how to determine these DRs, or whether the DRs have a more complicated effect than the right frontier constraint could be the subjects of future works. Further more, since we tried to construct the event structure compositionally, the scoping interaction among the new quantifiers (e.g., $\exists e_1 e_2 ...$) and previous existing ones (e.g., $\exists x_1 x_2 ...$) also needs further investigation.

References

1. Asher, N.: Reference to abstract objects in discourse. Springer, Heidelberg (1993)
2. Asher, N., Pogodalla, S.: Sdrt and continuation semantics. Proceedings of LENLS, Tokyo, Japan VII (2010)
3. Asher, N., Vieu, L.: Subordinating and coordinating discourse relations. Lingua 115(4), 591–610 (2005)
4. Bos, J.: Towards a large-scale formal semantic lexicon for text processing. In: Chiarcos, C., Eckart de Castilho, R., Stede, M. (eds.) Proceedings of the Biennal GSCL Conference From Form to Meaning: Processing Texts Automatically, 2009, pp. 3–14 (2009)

5. Davidson, D.: The logical form of action sentences. In: Rescher, N. (ed.) The Logic of Decision and Action. University of Pittsburgh Press, Pittsburgh (1967)
6. Groenendijk, J., Stokhof, M.: Dynamic predicate logic. Linguistics and Philosophy 14(1), 39–100 (1991)
7. de Groote, P.: Towards abstract categorial grammars. In: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, pp. 252–259. Association for Computational Linguistics (2001)
8. de Groote, P.: Towards a montagovian account of dynamics. In: Proceedings of Semantics and Linguistic Theory XVI (2006)
9. Gruber, J.S.: Studies in lexical relations. Ph.D. thesis, Massachusetts Institute of Technology. Dept. of Modern Languages (1965)
10. Heim, I.: File change semantics and the familiarity theory of definiteness. In: Bäuerle, R., Schwarze, C., von Stechow, A. (eds.) Meaning, Use, and Interpretation of Language, pp. 164–189. Walter de Gruyter, Berlin (1983)
11. Hobbs, J.R.: On the coherence and structure of discourse. CSLI, Center for the Study of Language and Information, US (1985)
12. Kamp, H.: A theory of truth and semantic representation. In: Groenendijk, J., Janssen, T., Stokhof, M. (eds.) Formal Methods in the Study of Language. Mathematical Centre Tracts, vol. 135, pp. 277–322. Mathematisch Centrum, Amsterdam (1981)
13. Lascarides, A., Asher, N.: Temporal interpretation, discourse relations and commonsense entailment. Linguistics and Philosophy 16(5), 437–493 (1993)
14. Lascarides, A., Asher, N.: Segmented discourse representation theory: Dynamic semantics with discourse structure. Computing Meaning, 87–124 (2007)
15. Mann, W., Thompson, S.: Rhetorical structure theory: Toward a functional theory of text organization. Text-Interdisciplinary Journal for the Study of Discourse 8(3), 243–281 (1988)
16. Montague, R.: English as A Formal Language. Linguaggi nella societae nella tecnica, 189–224 (1970)
17. Montague, R.: Universal Grammar. Theoria 36(3), 373–398 (1970)
18. Montague, R.: The proper treatment of quantification in ordinary english. In: Hintikka, J., Moravcsik, J., Suppes, P. (eds.) Approaches to Natural Language. Reidel, Dordrecht (1973)
19. Parsons, T.: Events in the Semantics of English: A Study in Subatomic Semantics. MIT Press, Cambridge (1991)
20. Polanyi, L.: A formal model of the structure of discourse. Journal of Pragmatics 12(5-6), 601–638 (1988)

Using Tree Transducers for Grammatical Inference

Noémie-Fleur Sandillon-Rezer[1,2,3] and Richard Moot[1,2,3]

[1] Université de Bordeaux
LaBRI, 351 cours de la libération
33400 Talence, France
[2] CNRS, esplanade des Arts et Métiers
33400 Talence, France
[3] SIGNES (INRIA Bordeaux SW), 351 cours de la libération
33400 Talence, France
{nfsr,moot}@labri.fr
http://www.labri.fr/perso/nfsr,
http://www.labri.fr/perso/moot

Abstract. We present a novel way of extracting a categorial grammar from annotated data. Using the sentences from the Paris VII annotated treebank [2] as our starting point, we use a tree transducer to convert the annotated trees from the corpus into categorial grammar derivations.

We describe both the formal aspects and the implementation of the tree transducer, which is a conservative extension of standard tree transducers allowing a compact specification of the transductions rules relevant for our purposes, and we discuss the specific set of transduction rules we use to convert the corpus into AB grammar derivation trees.

Evaluating the resulting tree transducer on the entire corpus, we find that it produces a treebank finds lexical entries for $90,0\%$ of the corpus, though it produces complete derivations for only 75% of all sentence in the corpus.

1 Introduction

The main goal of our current work is to extract a french categorial grammar from the annotated corpus of Paris VII.

In 1958, Lambek [16] introduced an algorithm to distinguish sentences from non-sentences. The basic principle of categorial grammar is that the lexicon assign a finite set of types[1] to words. Types are recursively defined from a set of basic types; in the current article the basic types we consider are s for sentence (eg. "Jean aime Marie"), np for noun phrase, (eg. "Jean" or "l'étudiant"), n for common noun (eg. "étudiant" or "livre"), pp for prepositional phrase (eg. "à Jean") and cs for complementized sentence (eg. "que Jean aime Marie"). For the recursive case, if A and B are types, then both A/B (which will give an expression of type A when combined with an expression of type B to its

[1] In the text, we will use the words "formula", "type" and "category" interchangeably.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 235–250, 2011.

right) and $B\backslash A$ (which will give an expression of type A when combined with an expression of type B to its left) are types as well. The derivation rules, which model the intuitions behind the types just given, are shown in Figure 1. The fragment of the Lambek calculus containing only these rules is often called AB.

$$\frac{A/B \quad B}{A}\,[/E] \qquad \frac{B \quad B\backslash A}{A}\,[\backslash E]$$

Fig. 1. The elimination rules for AB

As a simple example, consider a lexicon of Figure 2. which assigns the name "CSF" the category np and which assigns the past participle "créé" (created) the complex type $(np\backslash s)/np$; this complex type indicates that "créé" combines first with a noun phrase to its right (its object) to form an $np\backslash s$, which serves as argument to the auxiliary verb "a" to form an $np\backslash s$. We can use this lexicon to derive that "CSF a crée un journal" is a sentence, as shown in Figure 2. Each leaf in the derivation tree corresponds to a lexical entry (as indicated by a unary *Lex* rule), each local subtree corresponds to one of the two derivation rules of Figure 1 and the root of the tree is labeled s for sentence.

$$\dfrac{\dfrac{\dfrac{\text{CSF}}{np}\,[Lex] \qquad \dfrac{\dfrac{\text{a}}{(np\backslash s)/(np\backslash s)}\,[Lex] \qquad \dfrac{\dfrac{\text{créé}}{(np\backslash s)/np}\,[Lex] \qquad \dfrac{\dfrac{\text{un}}{np/n}\,[Lex] \qquad \dfrac{\text{journal}}{n}\,[Lex]}{np}\,[/E]}{np\backslash s}\,[/E]}{np\backslash s}\,[\backslash E]}{s}$$

Fig. 2. Derivation of "CSF a créé un journal"

The goal of this paper is to obtain derivation trees, just like the one shown in Figure 2 (though generally quite a bit more complicated!) for the sentences of the Paris VII corpus, which are linguistically annotated, but in a rather different format.

The rest of this paper is structured as follows. In the next section, we will talk about previous work on learning categorial grammars, then we will present the corpus we use as the basis for our grammar learning. In Section 4 we will introduce the generalized tree transducer which will be the formal tool for transforming the trees from the corpus into derivation trees for categorial grammar; Section 5 will give some details about its implementation and comment on the effectiveness of the current implementation for its designated task. The last section will conclude and discuss some of the possible extensions of the current work.

2 Learning Categorial Grammars

Work on learning categorial grammars can be more or less divided into two categories: a theoretical branch, which is placed within the framework of

identification in the limit introduced by Gold [10] and an applied branch, which is directed towards providing algorithms which produce categorial grammars on the basis of pre-existing annotated data.

The theoretical work answers important questions about the properties a class of grammars must have if we want a grammar learner to converge on the correct grammar. However, for practical applications, it has the following drawbacks:

- converting an input tree to a structure as required by the grammar induction algorithms of Buszkowski and Penn [4] and of Kanazawa [13] is not that much more difficult than converting directly to a proof tree[2]. Compare Figure 3, which represents the required input structure for the learning algorithms, with Figure 2: the only difference is that the arguments (the B formulas in each rule) are variables in Figure 3 and, as we will see in Section 4.2, the linguistic annotation generally gives enough clues to determine the correct instantiation of these variables.
- when we allow the lexicon to assign more than one formula to each word, the complexity of learning becomes NP-hard [8].

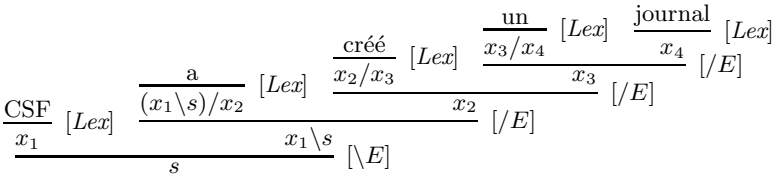


Fig. 3. Input trees for the learning algorithms of [4] and [13]

The more applied research, such as [11], [12], [19] and [20], uses special-purpose algorithms which apply only to the corpus in question, with little hope of reuse of tools for other corpora. An additional drawback of the use of special-purpose algorithms is that it is hard to prove the algorithm satisfies specific properties (besides inspection of the output).

Given the differences in annotation formats between corpora and the grammatical differences between languages, adapting a tool from one corpus to another will always be a labour-intensive enterprise. However, it is our hope that the use of the right kind of formal tool can allow one implementation, though with different set of parameters and rules files, to serve as the basis for corpus extraction over different types of corpora.

We believe that tree transducers can be such a tool. In Section 4, we will present a generalization of the top-down tree transducer, the G-transducer, and discuss some of its specific properties which make it an attractive tool for transforming linguistically annotated trees into AB derivations.

[2] Kanazawa discusses algorithms for categorial grammars from strings as well [13]. However, their high computational complexity excludes them from practical use.

3 Presentation of the Paris VII Corpus

As the starting point for our grammatical inference, we have used the Paris VII Corpus of around 12.000 annotated French sentences, which contains a selection of articles from the newspaper *Le Monde* from the period between December 1989 and January 1994.

The corpus has been annotated by the Paris VII Formal Linguistics laboratory "Laboratoire de Linguistique Formelle" [2], using planar trees[3] for the annotation. Figure 4 shows part of an annotation tree from the corpus.

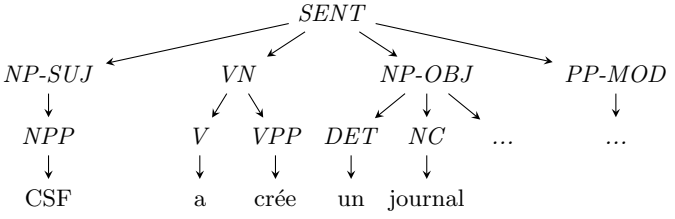


Fig. 4. Part of an annotation tree from the Paris VII treebank

The root is labeled *SENT*, for "sentence", and the leaves represent its words. Internal nodes are annotated with the following information, depending on whether or not they are preterminal nodes (ie. the parents of the leaves) [1]:

POS tags: the part-of-speech tags (POS) information is given for the preterminal nodes only, indicating the POS tag of its daughter. Examples of the POS tags used in the Paris VII corpus are: *NC* for the common nouns ("journal" in Figure 4), *V* for inflected verbs ("a" in the same figure), *DET* for the determiner ("un" in the figure).

Phrasal types: the other internal nodes are annotated with the syntactic category of the node. Examples are noun phrase (*NP*, "CSF" and "un journal" in Figure 4), verbal nucleus (*VN*, which is the verb cluster together with its clitics and adverbs occurring between the verbs of the cluster, in Figure 4 "a crée" is a verb cluster), adjectival phrase (*AP*)... In addition, the edge connecting the internal node to its parent can be annotated with the role the node fulfills within its parent category: so an *NP* inside a sentence (*SENT*) can be assigned roles like *–SUJ* to define the subject, *–OBJ* for an object, and *-MOD* for a sentential modifier. In the text, we will often simplify this by joining the node and edge label of a daughter node together. Thus, we will write *NP-OBJ* for a noun phrase which serves as object to its parent node (such as "un journal" in the example) and *NP-SUJ* for the subject noun phrase ("CSF").

[3] Trees where nodes have a variable number of daughters.

The node and edge labels give us important information about the structure of the sentence and we will use it to guide the transformation of the annotation trees into categorial derivations.

The goal of the current paper is to go from a tree like the one shown in Figure 4 to a tree like the one shown in Figure 2.

4 *G*-transducer

The main idea of this paper is to use a tree transducer for the automatic conversion of linguistically annotated trees into AB grammar derivations. Tree transducers have many applications in natural language processing (see [14] for an overview oriented towards statistical NLP and [7] for an general overview of tree transducers and their formal properties). However, to the best of our knowledge, they have not been applied to grammar extraction before. Since a tree transducer is an automaton translating trees to trees, and both our annotated input trees and AB derivations are simply trees, tree transducers seem to be a natural choice for the task at hand.

A transducer is just like an ordinary automaton, except that it writes to an output tape at the same time it reads symbols from its input tape. Where a finite-state transducer reads and writes strings, a tree transducer reads and writes trees. We want to apply our transducer to each of the trees in the corpus and obtain as output a derivation tree which is as close as possible to the original annotation. The output tree is a binary branching tree, where the internal nodes are assigned both a formula A and one of the labels [/] and [\] and its two daughters are assigned A/B and B (in the case of [/]) and B and $B\backslash A$ (in the case of [\]), corresponding to the rules $[/E]$ and $[\backslash E]$ from Figure 1. Note that this is the form required for many of the learning algorithms (see [4, 13] and Figure 3), so our transducers are compatible with Gold-style learning as well. However, we use additional rules (explained in 4.2) to assign formulas to the argument nodes B of each rule, which means that we do not have to deal with formulas containing variables (and subsequent unification to reduce the lexicon size).

In order to make writing the transduction rules more convenient, we introduce a slight generalization of the standard top-down tree transducer, which we will call the G-transducer. Instead of writing many different rules for only marginally different cases, these rule generalizations of the G-transducer allow us to apply rules to nodes of arbitrary arity and to have a form of restricted quantification over node labels. This extension is conservative: it does not augment the expressive power of the transducer, it only reduces to number of rules we have to write.

4.1 Formal Definition

Definition A generalized transducer is a tuple $\langle Q, q_i, Q_f, X, \delta \rangle$ where:

Q is the set of states.
q_i is the initial state, $q_i \in Q$.
Q_f is the set of final states, $Q_f \subseteq Q$.

X is the alphabet for reading and writing. $X = \{A \cup M \cup T\}$ with A for the corpus annotation alphabet (*SENT*, *NP*, *VN*, etc.), M for french words, and T for types (np/n, etc).

δ is the set of transduction rules, of the form
$q(f(x_1, \ldots, x_n)) \rightarrow u[q_1(t_1), \ldots q_p(t_p))]$ where for each $i \in [1, p]$, either $t_i \in \{x_1, \ldots, x_n\}$ or $t_i = f_i(x_{i,1}, \ldots x_{i,m})$ with $x_{i,j} \in \{x_1, \ldots, x_n\}$; $m \leq n - 1$.
$f \in A$ has an arity of n; $q, q_1, \ldots, q_n \in Q$ and $u \in T(r, X_n)$ (the set of subtrees with root $r \in A$ and with leaves X_n, where each for each $x_i \in X_n$, we have $x \in A$).
In our case, we use a sub-class of these rules:
$q(f(x_1, \ldots, x_n)) \rightarrow u[q_1(t_1), \ldots q_p(t_p)]$ where for all $i \in [1, p]$,

$$t_i \in \{x_1, \ldots x_n\} \text{ or } \begin{cases} q_i = q \text{ and} \\ t_i = f(x_{i,1}, \ldots x_{i,m}) \end{cases}$$

with $(x_{i,1}, \ldots x_{i,m})$ subsequences of $(x_1, \ldots, x_n)$. We will call these rules "recursive rules", since $q(f(\ldots))$ calls $q(f(\ldots))$.

Properties of the transducer. The G-transducer, as we will use it in what follows, has a number of properties which are shared by other tree transducers:

ε-free: there is no $\varepsilon-$rule in our transducer.
Linear: in the right-hand-side of each rule, each node is unique.
Complete (non-deleting): the nodes which appear in the left-hand-sid of the rule appear in the right-hand-side as well.
Deterministic: at each state and input, only a single transition applies.
Finite look-ahead: we allow each transduction rule have a complex tree as its left-hand-side, ie. f and the f_i can be complex trees with the indicated yield. This corresponds to having finite (as opposed to regular) look-ahead.[4]

Original features. The features which make the G-transducer original, and which, as we will show in Section 4.2, allow for a compact specification of the transduction rules, are the following.

Recursivity: A recursive rule applies to a node with a specified label but with an arbitrary number of daughters nodes. Figure 5 shows an example where the matched pattern consists of the rightmost daughters only (this is just for explanatory purposes, the pattern can occur on both sides as well, as shown in Figure 7). The parent node P has a number of daughters, daughter nodes 1 to n can be any sequence of nodes but node $n+1$ to $n+k$ match the specified pattern. In other words, the node matches the pattern of k nodes whatever the number of sister nodes occuring before it. Note that

[4] In the terminology used in [14], our transducers are a conservative extension of the **xRLND** transducers, where the prefix "**x**" indicates we allow our rules to have a complex tree instead of a node as their left hand side, **R** indicates "**R**oot to frontier" (ie. top-down) and the remaining letters indicate **L**inear, **N**on-deleting and **D**eterministic.

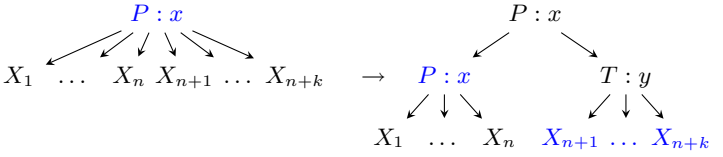


Fig. 5. The new P node has less daughters than before the transduction and the automaton will continue treating it in the same state and with the same output label as the parent node. This rule will generally be instantiated such that T is binary branching, and y is $x \backslash x$.

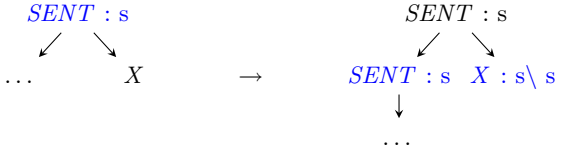


Fig. 6. The same rule will be applied for $X \in \{ADV, PP - MOD, AdP - MOD, ...\}$

Figure 6 and Figure 7 are also instances of this schema with $n = 1$ and $n = 2$ respectively. As shown in Figure 5, the transducer continues recursively to a new node P with n daughter nodes and can "restructure" nodes $n + 1$ to $n + k$ into a subtree T with output label y, with top-down descent to each of its daughter nodes.

Though these recursive rules generalize the standard definition of transduction rule in allowing rules to match nodes of arbitrary arity, we can, when given the maximum node arity of the input tree, convert a recursive rule into a number of "ordinary" transduction rules.

Parametrization: We allow rules with a restricted quantification over node labels. An example is shown in Figure 6. Here, the variable X can range over three node labels. Note that is equivalent to spelling out each of the different instantiations of X into its separate rule, but an important convenience when writing conversion rules.

Priority System: To avoid non-determinism, the rules are always applied in the same order (see Figure 7). The only disadvantage of this method is that it

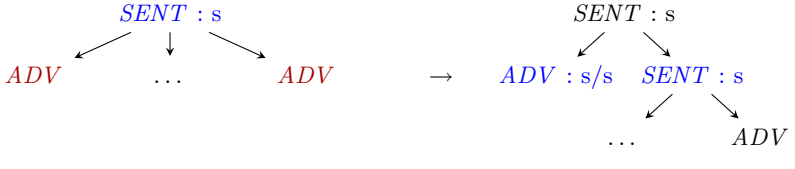


Fig. 7. When more than one rule can be applied, we always follow a predefined order

always gives wide scope to the same nodes. Though this is unfortunate, since it would bias a parser trained on the resulting trees, it makes no difference in the types assigned in the lexicon. In addition, it seems difficult to extend the transducer in such a way as to correctly and automatically deduce scope information which is not annotated in the corpus.

4.2 Transduction Rules

Even though the G-transducer allows us to specify rule schemata compactly, we still need to write a large number of transduction rules in order to get a reasonable coverage of the treebank. For the moment, we have focused on the most frequent combinations of mother node and daughters. The current implementation uses a total of 1297 transduction rules. In this section, we will schematically describe the ideas and design principles behind these transduction rules. We will return to the effectiveness of the transducer in Section 5.3.

For the transduction rules, we have tried to stay as close as possible to the "standard" way of analysing sentences and complex expressions in categorial grammars (see [16,22,18,24] for syntactic foundations, but also other extraction methods such as [11,12,19,20]). For example:

- Common nouns (*NC* in the corpus) will generally be assigned the type n.
- The noun phrases (*NP*) will generally be assigned the type np.
- The adjectives will generally have either type n/n or type $n\backslash n$ (French adjectives can occur both before and after the noun).
- The past participles and infinitives have the $np\backslash s$ type.
- etc.

In addition, the treatment of the main syntactic categories are treated as follows.

The *NP* nodes: The *NP* node itself can fulfil the role of subjet (*SUJ*), objet (*OBJ*), attribute of the subjet (*ATS*) or modifier (*MOD*[5]). *NP* nodes will be assigned np when appearing as *NP*, *NP-SUJ*, *NP-ATS*, *NP-OBJ* and x/x or $x\backslash x$ when appearing as *NP-MOD*.

The daughters of the NP node are treated as follows.

- *DET* will always be functor, and its type will be np/n.
- *NC*, the first *NC* node will generally be assigned the type n, and any later *NC* node the type $n\backslash n$.
- The adjectives (*ADJ* and *AP* nodes) are linked to the nearest *NC* node, and, whenever possible, with a modifier type. Most of time, this type will be n/n or $n\backslash n$ type (french modifiers can occur either before of after the common noun).

[5] *NP-MOD* is a noun phrase used adverbially as a modifier; examples would be "cette année" (this year) and "mardi 31 décembre" (tuesday the 31st of december). Its type will be x/x or $x\backslash x$, according to the type x of the sub-tree it modifies: that means s/s for a sentence-initial "Cette année," but $((np\backslash s)/np)\backslash((np\backslash s)/np)$ for a modifier occurring between a transitive verb and its object, and so on.

- Like the adjectives, the modifiers (**-MOD*) will be linked to the nearest subtree containing an *NC* node, and will be assigned one of the types n/n, $n\backslash n$, np/np or $np\backslash np$.

The *VN* nodes: contain the conjugated verb, one or more clitics[6], adverbs and any other verbs in the verb cluser.

- Adverbs and modifiers are always modifiers, and have x/x or $x\backslash x$ as type.
- The *VPP*, *VINF*, *VPinf* and *VPpart*[7] will have the type $np\backslash s$. This choice reflects the principle that these arguments contain an implicit subject[8].
- Subject clitics are always np arguments, modifier clitics are always modifiers, object clitics are arguments when they occur to the right of the verb and functors of type schema $X/(X/np)$ when they occur to its left, where X is the type of the verb without the clitic. In it simplest incarnation, this will be $X = np\backslash s$ and the clitic will have type $(np\backslash s)/((np\backslash s)/np)$. In other words, the clitic selects a transitive verb to its right in order to form an intransitive verb. Clitics are a difficult linguistic problem, and this choice follows the reasoning of [15] and [21] as close as is possible in an AB grammar.

The *SENT* nodes: The *SENT* node is the root node of each tree in the corpus. The generally principle is to move the different modifiers outside of the sentence domain, as shown in Figure 6, then reduce the verbal group and its arguments in one transduction rule.

- With the exception of the *Sint-MOD* node, sentence-initial and sentence-final adverbs and sentence modifiers will always have s/s or $s\backslash s$ type and will act as a modifier (see Figure 6).
- Subjects will have the type np.
- Objects, no matter what are their labels, will always be an argument of the verb.That means a *NP-OBJ* node will have the np type, a *Ssub-OBJ* will be typed cs, a *VPpart-OBJ* node will be $np\backslash s$, and so on.
- When a modifier or an adverb occurs next to a *VN* node, it will modify the verb (see Figure 8).
- If a *VN* node has a *VPP* as its rightmost daughter, and the *VN* node is followed by an object, a subject attribute or an object attribute, they will

[6] Clitics are words which are strongly linked to the verb; in general they occur directly before the verb without intervening material. Different clitic forms can fulfil different grammatical roles: subject, object, or modifiers.

[7] *VPP*: past participle; *VINF*: infinitive verb; *VPinf*: verb phrase group with an infinitive verb as its head; *VPpart*: verb phrase containing a past participle as its head.

[8] For the more linguistically oriented reader, this choice implements an Extended Projection Principle [5]

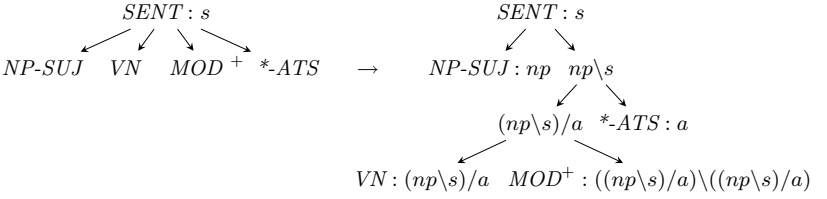


Fig. 8. $MD \in \{ADV, PP{-}MOD, AdP{-}MOD, ...\}$. If there is more than one modifier, we repeat the same scheme. The type of a depends on the ATS node: $NP\text{-}ATS = np$, $AP\text{-}ATS = PP\text{-}ATS = n\backslash n$, $Ssub\text{-}ATS = cs$, $VPinf\text{-}ATS = np\backslash s$. A similar rule applies if the subject in included in the VN subtree.

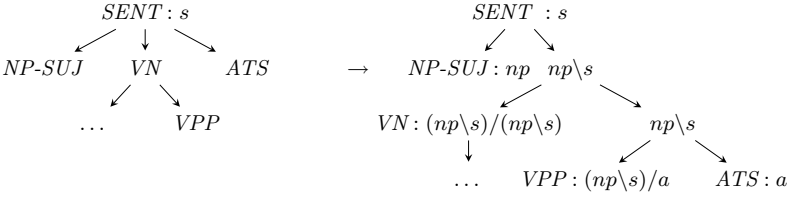


Fig. 9. The tree is rebracketed in such a way that the ATS node is an argument of the VPP node instead of its VN parent. Categories and types for ATS: $NP\text{-}ATS = np$, $AP\text{-}ATS = PP\text{-}ATS = n\backslash n$, $Ssub\text{-}ATS = cs$, $VPinf\text{-}ATS = np\backslash s$.

be arguments of the VPP node instead of the VN node[9] (see Figure 9; note that Figure 4 is an instance of this pattern as well).

The binary nodes: for theses nodes, we have to choose which of the daughters is the functor, and which is the argument. We apply the first of the following rules which is appropriate.

- An adverb or a **-MOD* node will always be modifier.
- A VN node will always be functor.
- A noun phrase not annotated as *NP-MOD* will always be argument.
- Inside a prepositional phrase PP, a P or $P + D$[10] will always be functor.

The output of one of our transducer for a given input tree from the corpus is a binary branching tree, where all nodes are assigned a type and where each local tree corresponds to an elimination rule. In other words, an AB derivation tree.

[9] The treebank generally annotates arguments to the past participle as arguments to the entire verb group. As shown in Figure 9 (and Figure 4 as well). The more natural analysis is for the auxiliary verb to select the past participle only and for the past participle to select the other arguments.

[10] An amalgam of a preposition and a determiner, like “des”, “du”, “au” and “aux”.

5 Implementation

We have implemented the transducer described in Section 4 in a robust system [23], which reads and writes trees in parenthesized format (as used by tools such as Tregex [17]) and which implements some additional consistency tests to verify that the transduction rules conform to the properties specified in Section 4 and that the output tree is a coherent proof tree (see [23] for the source code).

To complement the transducer, we have added a corpus corrector which repairs some frequent mistakes in such a way that the output tree can be handled without problems by the transducer. The two-step approach is summarized in Figure 10.

Fig. 10. Schematic representation of the tools

Though the resulting corpus of derivation trees do not constitute a parser, it contains the information necessary to induce a probabilistic tree automaton (or a probabilistic context-free grammar), which naturally produces a parse forest when intersected with an input string.

5.1 Corpus Corrector

Like any large corpus, the Paris VII treebank has a number of mistakes and inconsistencies. Rather than multiplying the number of transductions in order to handle these errors, we have decided to implement a separate corpus correction program, which corrects a number of errors which occur frequently enough to be a hindrance.

Correction of mistakes: whenever a word can have two functions in a sentence, sometimes the annotation is incorrect. This can happen, for example, when the past participle of a verb is identical to another verb form. In these cases, the annotation sometime labels them as an inflected verb form V instead of VPP. As a consequence, the transducer will erroneously apply the transduction corresponding to an idiomatic form (see Figure 11).

Another annotation mistake is linked to the ending punctuation: in theory, the $PONCT$ node should always be the daughter of $SENT$, in practice it is sometimes more deeply nested. In these cases, we move the punctuation to the outermost level in order to apply the normal transduction rules (the ending punctuation has the type $s\backslash s$).

Corrections of inconsistencies: Given that many people have annotated the corpus, sometimes the same grammatical construction are annotated differently from one sentence to another. As shown in Figure 12, coordinations are a case where the construction shown on the right-hand-side of the figure corresponds to the type $(np\backslash np)/np$ for the coordination CC (normally, a

word like “et” (and) or “ou” (or)). In the corpus, though coordinations often have the structure shown in Figure 12 on the right, we also frequently find it annotated as shown on the left (as well as some other variants). The correction phase converts these structures to the tree shown on the right, thereby producing a more consistent annotation.

Token conversions needed to apply the transducer: from the implementation, neither quotes nor colon are read by the program: they are seen as special characters. They are respectively replaced by \” and –*COL*–, so the transducer can treat them.

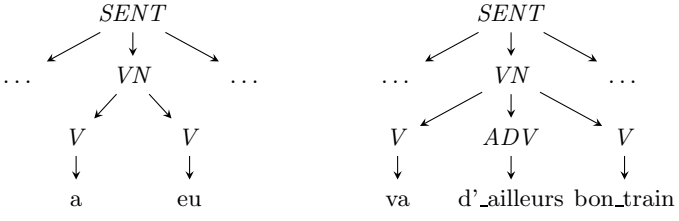


Fig. 11. Annotation mistake concerning the past participle, easily confused with the idiomatic form of “aller bon train”

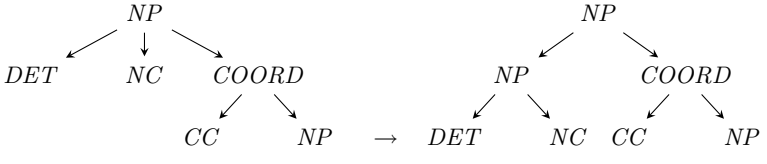
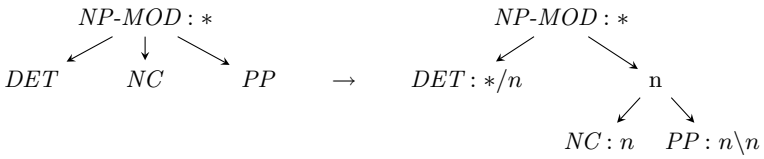


Fig. 12. In the left tree, it seems that the coordination conjoins an *NC* node and an *NP* node. The more natural analysis is shown on the right: the *DET* and *NC* nodes have a new *NP* node as parent, making this a standard *NP* coordination case. This correction permits to reduce the number of needed rules to treat coordinations.

5.2 Transducer

The implemented transducer takes a forest of parenthesized trees and a rule file as input, and gives a set of typed binary trees as output.

We have created a language to write the transduction rules. In this language, the rules are represented in a parenthesized form, as it is shown in Figure 13. This representation has been chosen to be close to the form of the trees. The wildcard type $*$ represents the type inherited from the treatment of the parent node; it will be globally substituted in the output pattern. The wildcard indicates that, in Figure 13, the *NP-MOD* can have any type x, depending on previous transduction rules that have been applied on the whole sentence and that the *DET* node, after transduction will have type x/n.




```
(rule
     (NP-MOD:* DET NC PP)
     (NP-MOD:* DET:*/n
          (:n NC:n PP:n\\n)))
```

Fig. 13. An example of one of the rules from our rules files, as used by the transducer. It is shown in its theoretical form (top) and in its parenthesized form (bottom).

When the trees are loaded, they are first verified for syntactic correctness (for example, we verify that the parentheses are well-bracketed). The rule file is verified too, and it should satisfy to the following three conditions:

- No information is lost (non-erasing transducer): the nodes included in the original pattern must appear in the replacement tree.
- Each rule should binarize the output tree: each rule which does not binarize the pattern it treats is reported. It avoids human mistakes when typing rules and it ensures that no node will need to be transduced a second time.
- Each type assignment should be coherent in its context: for each binary subtree, the type of the root must be derivable from the types of its daughters, that is each local subtree corresponds to one of the rules of Figure 1. This check ensures the transduction rules extract proofs.

Once rules and sentences are loaded, the transduction algorithm is very simple:

- Search for the matching rule.
- When we find it, the tree is transformed.
- Apply the same process to each daughter we need to transform.
- Forward the type if the node has only one daughter.

In addition to a typed binary trees forest, the transducer gives information about the number of rules applied and used, the untreated sentences and the configuration for which it has not found any rules.

5.3 Evaluation

Our current transducer generates a lexicon of 24.902 words (90, 0% of the complete corpus, which contains 27.589 words), even if it treats only 75% of the sentences completely: it uses 1297 rules to analyze 9380 sentences, which represent a greater coverage than our initial tranducer (372 rules to analyze 550 sentences). Some examples of our lexicon are shown in Figure 14. In addition to the lexicon, our transducer gives us derivation trees of the corpus of Paris VII and a second type of lexicon, more generalized, that only lists the different types

associated to preterminal nodes. This second lexicon is useful for assigning types to words which do not have enough occurrence in the training corpus (or which do not occur at all).

The generated lexicon is highly ambiguous: many frequent words have over a hundred different formulas assigned to them. For example, "et" (and), which occurs 3.882 times, has 159 distinct formulas in the lexicon and "lui" (him) occurs 181 times but has 45 distinct formulas in the lexicon.

The 25% of sentences we don't analyze contain structures which are complex and/or very rare: for the moment, we have focused on creating transduction rules which can be applied to multiple trees.

```
1079:au -  178:pp_a/n,147:(n\n)/n,  141:(s\s)/n,  92:(s/s)/n,
           84:pp/n,  83:(n\n)/n,  33:((np\s)\(np\s))/n, ...
1993:est-  380:(np\s)/np, 364:(np\s)/(n\n), 223:(np\s)/(np\s),
           161:(np\s)/pp, ...
8374:la-   7996:np/n, 94:(n\n)/n, 57:(s\s)/n, 43:(s/s)/n, ...
```

Fig. 14. Extract of the lexicon: the preposition-determiner "au" (to the), used 1079 times in the corpus, occurs 178 times as argument, 147 times as noun modifier, 141 times as sentence-final modifier and 92 times as sentence-initial modifier. The verb "est" (is), used 1993 times in the corpus, and its most frequent types in the analysis. As can be seen from the type assignments, it occurs most frequently as a transitive verb (380 occurrences), a copula verb (364 occurrences) and an auxiliary verb (223 occurrences). The determiner "la" (the) is used 8374 times in the corpus. The most frequent use is np/n, the usual type for a node *DET* (determiner) in a nominal phrase. The two next types are used in a modifier, the first one for a noun modifier and the second one for a sentence modifier, like"La semaine dernière ..." (Last week ...).

6 Conclusion and Future Work

We have introduced a generalization of the top-down tree transducer and shown how it can be used as a device for automated grammar extraction. We have described the transducer formally and talked about the way it is implemented. In addition, we have shown how the transduction language can be used to convert a large part of the Paris VII treebank into categorial derivations.

This work opens up several possibilities for future research. The most obvious is the addition of further transduction rules, with the goal of increasing the coverage towards the entire corpus and the comparison of our results with the semi-automatically extracted types which have been obtained in [20]; comparing the types obtained using both methods would be a useful way of validating both approaches.

Another possibility for future research is to use our forest of categorial derivations as input to the algorithm described by Marion and Besombes [3] with an added stochastic component (which would assign weights to each transition base roughly on how often this transition has been used in the corpus). This corresponds to using the forest we obtain as output of the transducer as a way of

estimating the weights of a probabilistic parser, in much the same way as has been done for CCG [6].

In addition, we want to make the types more detailed, distinguishing for example between verbs which take an infinitival group as an argument and those which take a past participle as an argument.

A final interesting line of research to follow would be to extend the current work on tree-to-tree transducers to tree-to-*graph* transducers [9]. This would allow us to move from the simple AB grammars towards the different types of Lambek grammars and their modern incarnations.

Our work is available at [23], under the GNU General Public License.

References

1. Abeillé, A., Clément, L.: Annotation morpho-syntaxique (2003), http://llf.linguist.jussieu.fr
2. Abeillé, A., Clément, L., Toussenel, F.: Building a treebank for french. Treebanks. Kluwer, Dordrecht (2003)
3. Besombes, J., Marion, J.: Learning tree languages from positive examples and membership queries. In: Ben-David, S., Case, J., Maruoka, A. (eds.) ALT 2004. LNCS (LNAI), vol. 3244, pp. 440–453. Springer, Heidelberg (2004)
4. BuszKowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49(4), 431–454 (1990), http://dx.doi.org/10.1007/BF00370157
5. Chomsky, N.: Lectures on government and binding (1981)
6. Clark, S., Curran, J.: Wide-coverage efficient statistical parsing with ccg and log-linear. Models, Computational Linguistics 33 (2007)
7. Comon, H., Dauchet, M., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (1997), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.125.6165
8. Costa-Florencio, C.: Consistent identification in the limit of any of the classes k-valued is np-hard. In: de Groote, P., Morrill, G., Retoré, C. (eds.) LACL 2001. LNCS (LNAI), vol. 2099, p. 125. Springer, Heidelberg (2001)
9. Engelfriet, J., Vogler, H.: The translation power of top-down tree-to-graph transducers. Journal of Computer and System Sciences 49(2) (1993)
10. Gold, E.M.: Language identification in the limit. Information and Control 10(5) (1967)
11. Hockenmaier, J.: Data and models for statistical parsing with combinatory categorial grammar (2003)
12. Hockenmaier, J.: Creating a ccgbank and a wide-coverage ccg lexicon for german. In: Proceedings of COLING/ACL, Sydney (2006)
13. Kanazawa, M.: Learnable Classes of Categorial Grammars. Center for the Study of Language and Information, Stanford University, Ventura Hall, 220 Panama Street, Stanford, CA 94305-4115 (1998); phone: 650-723-3084; e-mail: pubs@csli.stanford.edu; World Wide Web: http://csli-www.stanford.edu/publications/
14. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A. (ed.) CICLing 2005. LNCS, vol. 3406, pp. 1–24. Springer, Heidelberg (2005)

15. Kraak, E.: A deductive account of french object clitics. In: SYntax and Semantics, pp. 271–312 (1998)
16. Lambek, J.: The mathematics of sentence structure. The American Mathematical Monthly 65(3), 154–170 (1958), http://www.jstor.org/stable/2310058, article-type: primary_article / Full publication date: March 1958, Mathematical Association of America
17. Levy, R., Andrew, G.: Tregex and tsurgeon: tools for querying and manipulating tree data structures (2006), http://nlp.stanford.edu/software/tregex.shtml
18. Moortgat, M.: Categorial type logics. In: Handbook of Logic and Language, pp. 93–177 (1997), http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.5803
19. Moot, R.: Automated extraction of type-logical supertags from the spoken dutch corpus. In: Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach (2010)
20. Moot, R.: Semi-automated extraction of a wide-coverage type-logical grammar for french. In: Proceedings TALN 2010, Monreal (2010)
21. Moot, R., Retoré, C.: Les indices pronominaux du français dans les grammaires catégorielles. Lingvisticae Investigationes 29(1), 137–146 (2006)
22. Morrill, G.V.: Type Logical Grammar: Categorial Logic of Signs. Springer, Heidelberg (1994)
23. Sandillon-Rezer, N. (2011), http://www.labri.fr/perso/nfsr/
24. Steedman, M.: The syntactic process (200)

Distributional Learning of Abstract Categorial Grammars

Ryo Yoshinaka[1,⋆] and Makoto Kanazawa[2]

[1] ERATO MINATO Discrete Structure Manipulation System Project,
Japan Science and Technology Agency
ryoshinaka@erato.ist.hokudai.ac.jp
[2] National Institute of Informatics, Japan
kanazawa@nii.ac.jp

Abstract. Recent studies on grammatical inference have demonstrated the benefits of the learning strategy called "distributional learning" for context-free and multiple context-free languages. This paper gives a comprehensive view of distributional learning of "context-free" formalisms (roughly in the sense of Courcelle 1987) in terms of abstract categorial grammars, in which existing "context-free" formalisms can be encoded.

1 Introduction

Recent studies on grammatical inference have demonstrated how powerful "distributional learning" is for learning context-free languages and context-sensitive languages. Distributional learning algorithms exploit information on the contexts in which strings may occur to form a grammatical sentence. Classes of languages that admit distributional learning algorithms have some characterization on the distribution of strings in contexts. For example, Clark and Eyraud [1] show that *substitutable context-free languages* are efficiently learnable from positive data, where a language L is said to be substitutable if two strings are congruent whenever they share a context in which they may occur in L. In other words, a language L is substitutable if it holds that

$$u_1w_1v_1, u_1w_2v_1, u_2w_1v_2 \in L \implies u_2w_2v_2 \in L$$

for any substrings w_1,w_2 and contexts $\langle u_1, v_1\rangle$, $\langle u_2, v_2\rangle$. In multiple context-free grammars (MCFGs), a nonterminal determines a set of m-tuples of strings for some natural number m, hence the corresponding contexts will be $(m+1)$-tuples of strings. Accordingly distributional learning algorithms for MCFGs model and exploit the relation among tuples of strings [2,3,4]. Clark [5] discusses distributional learning of different formalisms with different learning strategies from a comprehensive viewpoint. His discussion covers a wide range of string languages, yet one can look at learning of tree languages from a similar vantage point. For example, a tree can be decomposed into a connected subgraph and

⋆ The author is concurrently working in Hokkaido University.

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 251–266, 2011.

the rest of the tree, in which the subgraph can be plugged. This way we exploit the distribution of a specific kind of subgraph in trees.

Abstract categorial grammars [6], ACGs for short, can be seen as a general framework in which many existing grammar formalisms can be naturally encoded. De Groote and Pogodalla [7,8] have discussed how ACGs encode CFGs, linear context-free rewriting systems (which are equivalent to MCFGs), tree adjoining grammars and linear context-free tree grammars, which are generically called *context-free formalisms* in the literature [9]. This paper aims to give a comprehensive view of distributional learning of context-free formalisms in terms of ACGs, regardless of the kind of language they generate: whether string languages, tree languages, or something else.

In Section 2, we review the simply typed lambda calculus and present a key lemma that enables a straightforward generalization of existing distributional learning algorithms. ACGs are defined in Section 3, where we briefly review de Groote and Pogodalla's encoding of existing formalisms in the ACG framework. Section 4 explains how to adapt the ideas of distributional learning to ACGs, and some concrete examples of learning algorithms are presented in Section 5. We conclude the paper with Section 6.

2 Simply Typed Lambda Calculus

Let A be a non-empty set of *atomic types*. The set $\mathscr{T}(A)$ of *types* built on A is defined as the smallest superset of A such that

- if $\alpha, \beta \in \mathscr{T}(A)$, then $(\alpha \to \beta) \in \mathscr{T}(A)$.

As usual, we suppress parentheses of types assuming right associativity, e.g., $\alpha \to \beta \to \gamma \to \delta$ means $(\alpha \to (\beta \to (\gamma \to \delta)))$. Moreover, we write $\alpha^3 \to \delta$ for $\alpha \to \alpha \to \alpha \to \delta$. Let $\mathrm{At}(\alpha)$ denote the set of atomic types that occur in α. The set of positions of positive (negative) occurrences of atomic type p is defined to be a finite language over $\{0,1\}$, as follows:

$$\begin{aligned}
\mathrm{occ}^+_p(p) &= \{\epsilon\},\\
\mathrm{occ}^+_p(q) &= \varnothing \quad \text{for all atomic } q \neq p,\\
\mathrm{occ}^+_p(\alpha \to \beta) &= 1\,\mathrm{occ}^-_p(\alpha) \cup 0\,\mathrm{occ}^+_p(\beta),\\
\mathrm{occ}^-_p(q) &= \varnothing \quad \text{for all atomic } q,\\
\mathrm{occ}^-_p(\alpha \to \beta) &= 1\,\mathrm{occ}^+_p(\alpha) \cup 0\,\mathrm{occ}^-_p(\beta)\ .
\end{aligned}$$

Let

$$\begin{aligned}
\mathrm{occ}^+(\alpha) = \bigcup_{p\in A} \mathrm{occ}^+_p(\alpha), \quad \mathrm{occ}^-(\alpha) = \bigcup_{p\in A} \mathrm{occ}^-_p(\alpha),\\
\mathrm{occ}(\alpha) = \mathrm{occ}^+(\alpha) \cup \mathrm{occ}^-(\alpha)\ .
\end{aligned}$$

The *size* of type α, denoted by $|\alpha|$, is the number of occurrences of atomic types in α.

For two sets A_0 and A_1 of atomic types, a *type substitution* η is a mapping from A_0 to $\mathscr{T}(A_1)$, which is homomorphically extended as $\eta(\alpha{\to}\beta) = \eta(\alpha){\to}\eta(\beta)$. We often use the suffix notation $\alpha\eta$ instead of $\eta(\alpha)$. A type substitution η is called a *type relabeling* if $p\eta$ is atomic for each atomic type p. Two types α_1 and α_2 are said to be *unifiable* if there is a type substitution η such that $\alpha_1\eta = \alpha_2\eta$. In such a case, η is called a *unifier* of α_1 and α_2. A *most general unifier* ρ is a unifier such that for any unifier η, there is a type substitution σ such that $\eta = \sigma \circ \rho$. If α_1 and α_2 are unifiable, $\mathrm{mgu}(\alpha_1, \alpha_2)$ denotes a most general unifier of α_1 and α_2. We assume that if $\rho = \mathrm{mgu}(\alpha_1, \alpha_2)$, we have $p\rho = p$ for all $p \notin \mathrm{At}(\alpha_1) \cup \mathrm{At}(\alpha_2)$.

Let X be a countably infinite set of *variables*. The set of *λ-terms* is defined to be the smallest set that fulfills the following conditions:

- for any $x \in X$, x is a λ-term,
- for any two λ-terms M and N, (MN) is a λ-term,
- for any λ-terms M, $(\lambda x.M)$ is also a λ-term.

We suppress parentheses of λ-terms in such a way that $\lambda xyz.MNPQ$ means $(\lambda x.(\lambda y.(\lambda z.(((MN)P)Q))))$, and so on. The notions of *subterms*, *free variables*, *β-normal form* etc., are defined as usual (see, e.g., [10]). We always identify two α-equivalent terms. When M β-reduces to N by zero or more steps, we write $M \twoheadrightarrow_\beta N$. The set of free variables of a λ-term M is denoted by $\mathrm{FV}(M)$.

We call M *affine* if for every subterm of M of the form NP, $\mathrm{FV}(N) \cap \mathrm{FV}(P) = \varnothing$. M is λI if for every subterm of M of the form $\lambda x.N$, $x \in \mathrm{FV}(N)$. M is *linear* if M is both affine and λI.

A *type environment* Γ is a finite partial mapping from X to $\mathscr{T}(A)$, which is usually denoted as a list $x_1 : \alpha_1, \ldots, x_n : \alpha_n$ of pairs of a variable x_i and the assigned type α_i. We define λ-terms and typing judgments on them by the following type assignment system:

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\mathrm{var})$$

$$\frac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ (\mathrm{app}) \qquad \frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}\ (\mathrm{abs})$$

If $\Gamma \vdash M : \alpha$, then (Γ, α) is a *typing* of M and M is an *inhabitant* of (Γ, α). A pair (Γ, α) is *inhabited* if it has an inhabitant.

If M has a typing, then there is a typing (Γ, α) such that for every typing (Γ', α') of M, there is a type substitution σ such that $\Gamma\sigma = \Gamma' \restriction \mathrm{FV}(M)$ and $\alpha\sigma = \alpha'$. Such a typing (Γ, α) is called a *principal typing* of M. Given a λ-term M, a principal typing of M can be computed in polynomial time (see [10]).

A deduction of $\Gamma \vdash M : \alpha$ is *η-long* if every judgment of the form $\Delta \vdash N : \beta{\to}\gamma$ that occurs in it is either the conclusion of an instance of (abs) or the left premise of an instance of (app). If $\Gamma \vdash M : \alpha$ has an η-long deduction, M is an *η-long inhabitant* of (Γ, α). We also say that M is an η-long inhabitant of α if $\Gamma = \varnothing$.

Lemma 1. *Suppose that M is a λI η-long inhabitant of (Γ, α). Let (Γ', α') be a principal typing of M. Then there is a type relabeling σ such that $\Gamma \restriction \mathrm{FV}(M) = \Gamma'\sigma$ and $\alpha = \alpha'\sigma$.*

Theorem 1 (Subject Reduction). *If $\Gamma \vdash M : \alpha$ and $M \twoheadrightarrow_\beta N$, then $\Gamma \vdash N : \alpha$.*

A β-reduction step that contracts a β-redex $(\lambda x.P)Q$ is called *non-erasing* if $x \in \mathrm{FV}(P)$, and *non-duplicating* if x occurs free at most once in P. A (many-step) β-reduction is non-erasing (non-duplicating) if it entirely consists of non-erasing (non-duplicating) β-reduction steps.

Theorem 2 (Subject Expansion). *If $M \twoheadrightarrow_\beta N$ by non-erasing and non-duplicating β-reduction and $\Gamma \vdash N : \alpha$, then $\Gamma \vdash M : \alpha$.*

Note that if M, N are linear, any β-reduction from MN is non-erasing and non-duplicating.

Let $\Gamma = \{x_1 : \alpha_1, \dots, x_n : \alpha_n\}$. For $p \in A$, define

$$\mathrm{occ}^+_p(\Gamma, \alpha_0) = \{\, (0, w) \mid w \in \mathrm{occ}^+_p(\alpha_0) \,\} \cup \{\, (i, w) \mid w \in \mathrm{occ}^-_p(\alpha_i), 1 \le i \le n \,\},$$
$$\mathrm{occ}^-_p(\Gamma, \alpha_0) = \{\, (0, w) \mid w \in \mathrm{occ}^-_p(\alpha_0) \,\} \cup \{\, (i, w) \mid w \in \mathrm{occ}^+_p(\alpha_i), 1 \le i \le n \,\}.$$

Let

$$\mathrm{occ}^+(\Gamma, \alpha_0) = \bigcup_{p \in A} \mathrm{occ}^+_p(\Gamma, \alpha), \quad \mathrm{occ}^-(\Gamma, \alpha_0) = \bigcup_{p \in A} \mathrm{occ}^-_p(\Gamma, \alpha),$$
$$\mathrm{occ}(\Gamma, \alpha_0) = \mathrm{occ}^+(\Gamma, \alpha_0) \cup \mathrm{occ}^-(\Gamma, \alpha_0)\ .$$

A pair (Γ, α) is *balanced* if $|\,\mathrm{occ}^+_p(\Gamma, \alpha)| \le 1$ and $|\,\mathrm{occ}^-_p(\Gamma, \alpha)| \le 1$ for every atomic type p. (Γ, α) is *strictly balanced* if $|\,\mathrm{occ}^+_p(\Gamma, \alpha)| = |\,\mathrm{occ}^-_p(\Gamma, \alpha)| \le 1$ for every atomic p.

Theorem 3 (see Hirokawa [11] for history). *If M is an affine λ-term, then M has a balanced principal typing. If M is a linear λ-term, then M has a strictly balanced principal typing.*

Theorem 4 ([11]). *If M is a β-normal inhabitant of a balanced pair (Γ, α), then M is affine.*

Theorem 5 (Coherence Theorem [12], see [13]). *If M and N are inhabitants of a balanced pair (Γ, α), then $M =_{\beta\eta} N$.*

Lemma 2. *There is a polynomial-time algorithm which takes a balanced pair (Γ, α) as input and determines whether it is inhabited, and if so, returns its unique η-long β-normal inhabitant.*

Proof. The algorithm given by Bunder [14] runs in polynomial time. □

Lemma 3. *Let α, β be fixed types. There is a polynomial $f(n)$ satisfying the following conditions: If P is a linear η-long β-normal inhabitant of $(\Gamma, \beta) = (\{z_1 : \gamma_1, \dots, z_l : \gamma_l\}, \beta)$, where $\mathrm{FV}(P) = \{z_1, \dots, z_l\}$, then there are at most $f(|\gamma_1| + \cdots + |\gamma_l|)$ pairs of λ-terms (Q, R) such that*

– *Q is a linear η-long β-normal inhabitant of (Γ, α),*
– *R is a linear η-long β-normal inhabitant of $(\Gamma, \alpha \to \beta)$, and*
– *$RQ \twoheadrightarrow_\beta P$.*

Moreover, given P and Γ as input, one can list all such pairs (Q, R) in polynomial time.

Proof. Let P be a linear η-long β-normal inhabitant of $(\Gamma, \beta) = (\{z_1 : \gamma_1, \ldots, z_l : \gamma_l\}, \beta)$, with $\mathrm{FV}(P) = \{z_1, \ldots, z_l\}$. Suppose that a pair of λ-terms (Q, R) satisfies the conditions of the theorem. Since R and Q are linear, $RQ \twoheadrightarrow_\beta P$ by non-erasing non-duplicating β-reduction. This implies that $(\mathrm{FV}(Q), \mathrm{FV}(R))$ is a partition of $\mathrm{FV}(P)$. By Lemma 1 and Theorem 3, Q and R have strictly balanced principal typings of the form (Γ_1, α_1) and $(\Gamma_2, \alpha_2 \to \beta_2)$, respectively, such that for some type relabelings σ_1 and σ_2, we have

$$\Gamma_1\sigma_1 \cup \Gamma_2\sigma_2 = \Gamma, \quad \alpha_1\sigma_1 = \alpha_2\sigma_2 = \alpha, \quad \beta_2\sigma_2 = \beta\ .$$

Let $\rho = \mathrm{mgu}(\alpha_1, \alpha_2)$. Then ρ must be a type relabeling. Without loss of generality, we may assume that the following conditions hold:

– (Γ_1, α_1) and $(\Gamma_2, \alpha_2 \to \beta_2)$ have no atomic type in common,
– for each $p \in \mathrm{At}(\alpha_1) \cup \mathrm{At}(\alpha_2)$, $p\rho$ is a "fresh" atomic type, i.e., an atomic type not occurring in (Γ_1, α_1) or $(\Gamma_2, \alpha_2 \to \beta_2)$.

Let $\alpha' = \alpha_1\rho$, $\beta' = \beta_2\rho$, and $\Gamma' = (\Gamma_1 \cup \Gamma_2)\rho$. Then (Γ', β') must be a principal typing of RQ. By the Subject Reduction and Expansion Theorems, this implies that (Γ', β') is a principal typing of P. This typing is also strictly balanced, by Theorem 3.

Note that $(\Gamma_1 \cup (\Gamma' \restriction \mathrm{FV}(R)), \alpha_1) = (\Gamma_1 \cup (\Gamma_2\rho), \alpha_1)$ is balanced, so by the Coherence Theorem, Q is its unique η-long β-normal inhabitant. This typing differs from (Γ', α) only in positions where atomic types in $\mathrm{At}(\alpha_1)$ occur, and so is completely determined, up to renaming of atomic types, by an injective mapping

$$h_1 \colon \mathrm{occ}(\alpha) \to \mathrm{occ}(\Gamma, \alpha)$$

such that

– $h_1(w) \in \mathrm{occ}^-(\Gamma, \alpha)$ for all $w \in \mathrm{occ}^+(\alpha)$,
– $h_1(w) \in \mathrm{occ}^+(\Gamma, \alpha)$ for all $w \in \mathrm{occ}^-(\alpha)$, and
– $h_1(w) = (0, v)$ implies $h_1(v) = (0, w)$.

There are less than $(|\gamma_1| + \cdots + |\gamma_l| + |\alpha|)^{|\alpha|}$ such mappings.

Similarly, the typing $((\Gamma' \restriction \mathrm{FV}(Q)) \cup \Gamma_2, \alpha_2 \to \beta_2) = ((\Gamma_1\rho) \cup \Gamma_2, \alpha_2 \to \beta_2)$, and hence R, is completely determined by an injective mapping $h_2 \colon \mathrm{occ}(\alpha) \to \mathrm{occ}(\Gamma, \alpha \to \beta)$ such that

– $h_2(w) \in \mathrm{occ}^+(\Gamma, \alpha \to \beta)$ for all $w \in \mathrm{occ}^+(\alpha)$,
– $h_2(w) \in \mathrm{occ}^-(\Gamma, \alpha \to \beta)$ for all $w \in \mathrm{occ}^-(\alpha)$, and
– $h_2(w) = (0, 1v)$ implies $h_1(v) = (0, 1w)$.

There are less than $(|\gamma_1| + \cdots + |\gamma_l| + |\alpha| + |\beta|)^{|\alpha|}$ such mappings.

We have proved that for some polynomial $f(n)$ determined by α and β, there are less than $f(|\gamma_1|+\cdots+|\gamma_l|)$ pairs of λ-terms (Q, R) satisfying the conditions of the lemma. To show that one can enumerate all such pairs in polynomial time, it suffices to show that, given injective mappings h_1, h_2, one can determine whether there is a corresponding pair (Q, R), and if so, output it in polynomial time. Let (Γ', β') be a principal typing of P, which must be strictly balanced by Theorem 3. By choosing a fresh atomic type for each pair of positions $\{(0, w), h_1(w)\}$ in $\mathrm{occ}(\Gamma', \alpha)$, the mapping h_1 determines a balanced pair

$$(\Gamma_1', \alpha_1)$$

that differs from (Γ', α) only in the positions in $\{\,(0, w) \mid w \in \mathrm{dom}(h_1)\,\} \cup \mathrm{rng}(h_1)$. Similarly, the mapping h_2 determines a balanced pair $(\Gamma_2', \alpha_2 \to \beta_2)$ that differs from $(\Gamma', \alpha \to \beta')$ only in the positions in $\{\,(0, 1w) \mid w \in \mathrm{dom}(h_2)\,\} \cup \mathrm{rng}(h_2)$, where $\mathrm{At}(\alpha_1) \cap \mathrm{At}(\alpha_2) = \varnothing$. By Lemma 2, one can, in polynomial time, determine whether each is inhabited by a linear λ-term, and if so, produce its unique linear η-long β-normal inhabitant. Let Q and R be the inhabitants of (Γ_1', α_1) and of $(\Gamma_2', \alpha_2 \to \beta)$, respectively, obtained this way. To see whether their application RQ β-reduces to P, first check whether $\mathrm{FV}(Q) \cap \mathrm{FV}(R) = \varnothing$, and if so, let

$$\Gamma_1 = \Gamma_1' \restriction \mathrm{FV}(Q), \quad \Gamma_2 = \Gamma_2' \restriction \mathrm{FV}(R)\ .$$

Since (Γ_1, α_1) and $(\Gamma_2, \alpha_2 \to \beta_2)$ are balanced, by Lemma 1 and Theorem 3, it is easy to see that (Γ_1, α_1) and $(\Gamma_2, \alpha_2 \to \beta_2)$ must be principal typings of Q and R, respectively. It follows that (Γ_1, α_1) and $(\Gamma_2, \alpha_2 \to \beta_2)$ are both strictly balanced and share no atomic type. Let $\rho = \mathrm{mgu}(\alpha_1, \alpha_2)$. Then we see that

$$(\Gamma_1 \cup \Gamma_2, \beta_2)\rho$$

is a principal typing of RQ, assuming as before that $p\rho$ is "fresh" for each $p \in \mathrm{At}(\alpha_1) \cup \mathrm{At}(\alpha_2)$. By the Coherence Theorem and the Subject Reduction and Expansion Theorems, $RQ \twoheadrightarrow_\beta P$ if and only if this typing is also a principal typing of P. This is so just in case $(\Gamma_1 \cup \Gamma_2, \beta_2)\rho$ and (Γ', β') are identical up to renaming of atomic types, which can be checked in polynomial time. □

3 Abstract Categorial Grammars

3.1 Definition

A *higher-order signature* Σ is a triple $\langle A, C, \tau\rangle$ where A is a finite non-empty set of atomic types, C is a finite set of constants, and τ is a function from C to $\mathscr{T}(A)$. λ-terms over Σ and their typing judgments are defined by the type assignment system introduced in the previous subsection enhanced with the axiom

$$\frac{}{\Gamma \vdash_\Sigma c : \tau(c)}\ (\mathrm{con})$$

for $c \in C$, where we use $\vdash_\Sigma$ to specify the signature Σ under which the typing judgement is given. The set of λ-terms over Σ is denoted by $\Lambda(\Sigma)$. We understand Σ_* to be $\langle A_*, C_*, \tau_* \rangle$ for any subscript $*$.

For two higher-order signatures Σ_0 and Σ_1, a *term substitution* θ is a mapping from C_0 to $\Lambda(\Sigma_1)$ such that $\mathrm{FV}(\theta(c)) = \varnothing$ for all $c \in C_0$. For two higher-order signatures Σ_0 and Σ_1, we say that a type substitution $\eta : A_0 \to \mathscr{T}(A_1)$ and a term substitution $\theta : C_0 \to \Lambda(\Sigma_1)$ are *compatible* iff $\vdash_{\Sigma_1} \theta(c) : \tau_0(c)\eta$ holds for all $c \in C_0$. A *lexicon* from Σ_0 to Σ_1 is a compatible pair of a type substitution and a term substitution. For a lexicon $\mathscr{L} = \langle \eta, \theta \rangle$, we define $\hat{\theta}$ as the homomorphic extension of θ that maps a variable to the same variable. Indeed, we always have $\vdash_{\Sigma_1} \hat{\theta}(M) : \alpha\eta$ if $\vdash_{\Sigma_0} M : \alpha$. We identify a lexicon $\mathscr{L} = \langle \eta, \theta \rangle$ with the functions η and $\hat{\theta}$. A lexicon $\mathscr{L}$ is said to be *linear* if it assigns a linear λ-term to every constant.

Definition 1 (de Groote [6]). An *abstract categorial grammar (*ACG*)* is a quadruple $\mathscr{G} = \langle \Sigma_0, \Sigma_1, \mathscr{L}, s \rangle$, where

- Σ_0 is a higher-order signature, called the *abstract vocabulary*,
- Σ_1 is a higher-order signature, called the *object vocabulary*,
- $\mathscr{L}$ is a linear lexicon from Σ_0 to Σ_1,
- $s \in A_0$ is called the *distinguished type*.

An ACG $\mathscr{G} = \langle \Sigma_0, \Sigma_1, \mathscr{L}, s \rangle$ generates two languages, the *abstract language* $\mathcal{A}(\mathscr{G})$ and the *object language* $\mathcal{O}(\mathscr{G})$, defined as

$$\mathcal{A}(\mathscr{G}) = \{\, |M|_\beta \mid \vdash_{\Sigma_0} M : s \,\},$$
$$\mathcal{O}(\mathscr{G}) = \{\, |\mathscr{L}(M)|_\beta \mid M \in \mathcal{A}(\mathscr{G}) \,\},$$

where $|P|_\beta$ denotes the β-normal form of P.

We call the triple $\langle c, \tau_0(c), \mathscr{L}(c) \rangle$ for $c \in C_0$ a *lexical entry*, and specify an ACG by giving the set of lexical entries and the distinguished type.

The abstract language can be thought of as a set of abstract grammatical structures, and the object language is regarded as the set of concrete forms obtained from these abstract structures and the lexicon. Thus, we simply speak of the language generated by an ACG to refer to its object language. The term *abstract categorial languages (*ACL*s)* means the object languages of ACGs. Elements of A_0 are called *abstract atomic types*.

Since ACGs are concerned with linear λ-terms, hereafter we consider only linear λ-terms and by a λ-term we mean a linear one. This paper is particularly concerned with *second-order* ACG*s*. A second-order ACG is an ACG whose abstract constants in C_0 have a type of the form $p_1 \to \ldots \to p_n \to q$ for some $p_1, \ldots, p_n, q \in A_0$. By an ACG we mean a second-order ACG.

3.2 Encoding of Context-Free Formalisms in ACGs

ACGs can be seen as a general framework in which different context-free formalisms can be encoded. We briefly review de Groote and Pogodalla's [8] encoding techniques for some context-free formalisms in ACGs. This paper does not

give formal definitions of those formalisms. For the details, the reader is referred to their paper [8].

Strings. For an alphabet T, let $\Sigma_T = \langle \{o\}, T, \tau \rangle$ where $\tau(a) = o \to o$ for all $a \in T$. We denote the "string type" $o \to o$ by str. A string $abcd$ over T is encoded by $\lambda z.a(b(c(dz)))$, which is denote as $/abcd/$. Then for any $w \in T^*$, we have $\vdash_{\Sigma_T} /w/ : \mathit{str}$. Two strings are concatenated by $\mathbf{B} = \lambda xyz.x(yz)$, that is, $\mathbf{B}/abc//de/ \twoheadrightarrow_\beta /abcde/$. For the sake of legibility, we use the infix notation $/abc/ + /de/$ to denote $\mathbf{B}/abc//de/$. Obviously $+$ is associative modulo β.

Trees. Trees are constructed over a ranked alphabet $T = \bigcup_{k \in \mathbb{N}} T^{(k)}$ for pairwise disjoint sets $T^{(0)}, T^{(1)}, \ldots$. A symbol $a \in T^{(k)}$ always has k children. For a finite ranked alphabet T, let $\Sigma_T = \langle \{o\}, T, \tau \rangle$ where $\tau(a) = o^k \to o$ if $a \in T^{(k)}$. Each tree over T is straightforwardly translated to a λ-term with no variables.

Context-Free Grammars. Let $G = \langle N, T, P, S \rangle$ be a CFG, where N is the set of nonterminal symbols, T is the set of terminal symbols, P is the set of production rules and $S \in N$ is the distinguished nonterminal. De Groote and Pogodalla construct an ACG $\mathscr{G}_G = \langle \Sigma_0, \Sigma_T, \mathscr{L}, S \rangle$ for G as follows. Let $A_0 = N$ and $\mathscr{L}(B) = \mathit{str}$ for all $B \in A_0$. For each rule $\pi \in P$ of the form $B \to u_0 B_1 u_1 \ldots B_n u_n$, where $B_i \in N$ and $u_i \in T^*$, $\mathscr{G}_G$ has the lexical entry

$$\langle c_\pi,\ B_1 \to \ldots \to B_n \to B,\ \lambda x_1 \ldots x_n./u_0 x_1 u_1 \ldots x_n u_n/ \rangle\ .$$

One can see that a nonterminal B generates $w \in T^*$ iff there is $M \in \Lambda(\Sigma_0)$ such that $\vdash_{\Sigma_0} M : B$ and $\mathscr{L}(M) \twoheadrightarrow_\beta /w/$.

Linear Context-Free Rewriting Systems. Nonterminals of an LCFRS are ranked and each nonterminal of rank m generates m-tuples of strings. We denote the "m-tuple type" $(\mathit{str}^m \to \mathit{str}) \to \mathit{str}$ by tpl_m. For example, a triple $\langle u, v, w \rangle$ is encoded as $\lambda y.y/u//v//w/$, which admits the typing judgement $\vdash_{\Sigma_T} \lambda y.y/u//v//w/ : \mathit{tpl}_3$. Let $G = \langle N, T, P, S \rangle$ be an LCFRS, where N is the set of ranked nonterminals, T is the set of terminals, P is the set of rules and $S \in N$ is the distinguished nonterminal of rank 1. The corresponding ACG $\mathscr{G}_G = \langle \Sigma_0, \Sigma_T, \mathscr{L}, s \rangle$ is defined as follows. Let $A_0 = N$ and $\mathscr{L}(B) = \mathit{tpl}_m$ for $B \in N$ of rank m. For each rule $\pi \in P$ of the form

$$B(w_1, \ldots, w_m) \to B_1(\boldsymbol{x}_1), \ldots, B_n(\boldsymbol{x}_n),$$

where $\boldsymbol{x}_i$ denotes the sequence of variables $x_{i,1}, \ldots, x_{i,m_i}$ and w_i is a sequence of terminals and variables, $\mathscr{G}_G$ has an abstract constant c_π of type $B_1 \to \ldots \to B_n \to B$ that is mapped by $\mathscr{L}$ to

$$\lambda x_1 \ldots x_n y.x_1(\lambda \boldsymbol{x}_1.x_2(\lambda \boldsymbol{x}_2. \ldots x_n(\lambda \boldsymbol{x}_n.y/w_1/ \ldots /w_m/) \ldots))\ .$$

One can see that a nonterminal B generates $\langle v_1, \ldots, v_m \rangle \in (T^*)^m$ iff there is $M \in \Lambda(\Sigma_0)$ such that $\vdash_{\Sigma_0} M : B$ and $\mathscr{L}(M) \twoheadrightarrow_\beta \lambda y.y/v_1/ \ldots /v_m/$. We have one more abstract constant f of type $S \to s$ that is mapped to $\mathscr{L}(f) = \lambda x.x(\lambda z.z)$, with which we have $\mathscr{L}(f)(\lambda y.y/w/) \twoheadrightarrow_\beta /w/$.

Linear Context-Free Tree Grammars. Nonterminals of an LCFTG define trees with a fixed number of holes on leaves. Let $G = \langle N, T, P, S\rangle$ be an LCFTG, where N is the set of ranked nonterminals, T is the set of ranked terminals, P is the set of rules and $S \in N$ is the distinguished nonterminal of rank 0. The corresponding ACG $\mathscr{G}_G = \langle \Sigma_0, \Sigma_T, \mathscr{L}, S\rangle$ is defined as follows. Let $A_0 = N$ and $\mathscr{L}(B) = o^m \to o$ for $B \in N$ of rank m. Suppose that a rule $\pi \in P$ is of the form $B(x_1, \ldots, x_m) \to t$, where B is of rank m, t is a tree with some leaves labeled with $x_1, \ldots, x_m$, and t has n occurrences of nonterminals $B_1, \ldots, B_n$. $\mathscr{G}_G$ has an abstract constant c_π of type $B_1 \to \ldots \to B_n \to B$ that is mapped by $\mathscr{L}$ to

$$\lambda y_1 \ldots y_n x_1 \ldots x_m . t[B_i := y_i]_{1 \le i \le n} \ .$$

One can see that a nonterminal B of rank m generates a tree t with variables $x_1, \ldots, x_m$ iff there is M such that $\vdash_{\Sigma_0} M : B$ and $\mathscr{L}(M) \twoheadrightarrow_\beta \lambda x_1 \ldots x_m . t$.

4 Distributional Learning of ACGs

Distributional learning of CFGs models and exploits the distribution of strings ($\in T^*$) in contexts ($\in T^* \times T^*$) in the language L of our learning target. The learner takes an example $w \in T^*$ and then decomposes w into a substring v and a context $\langle u_1, u_2\rangle$ such that $w = u_1 v u_2$. What we do here is to extract possible intermediate structures that could appear in a derivation process for $w \in L$ in a CFG G_* that generates L. In a derivation tree t of G_* that yields w, every subtree t' of t determines a substring v of w, and the remaining part of the tree, which is called a *tree context*, determines a context $\langle u_1, u_2\rangle$ such that $w = u_1 v u_2$. Hence, finding a right decomposition of $w \in L$ corresponds to finding a subtree of the derivation tree and its tree context. If one can find all the right decompositions and classify them according to the nonterminal rooting the sub-derivation tree, then the grammar G_* can be simulated.

One may associate derivation trees of a CFG with λ-terms in the abstract language of a second-order ACG. Indeed those λ-terms are just trees. The yielded structures of both "trees" are visible to the learner, though those trees themselves are not. Suppose that a learner is given an example $P \in \mathcal{O}(\mathscr{G}_*)$ from the target language, where there should be $M \in \mathcal{A}(\mathscr{G})$ such that $\mathscr{L}(M) \twoheadrightarrow_\beta P$. The learner cannot observe the term M, yet she would like to decompose P into Q and P' so that

- there are M', N such that $M = N[z := M']$ with $\vdash_{\Sigma_0} M' : p$ for some $p \in A_0$,
- $P' = \mathscr{L}(M')$, $Q = \mathscr{L}(N)$.

Clearly $Q[z := P'] \twoheadrightarrow_\beta P$ holds. We call P' an *intermediate substructure of* P. For technical convenience, hereafter we assume $\mathscr{L}(c)$ to be an η-long β-normal inhabitant of $\mathscr{L}(\tau_0(c))$ for any abstract constant $c \in C_0$.

Usually a learner is supposed to know what kind of object can be obtained during the derivation procedure as intermediate structures. In CFGs, they are just

strings, as nonterminals derive strings. Intermediate substructures of LCFRSs are m-tuples of strings derived by their nonterminals, hence the learner for LCFRSs, e.g. [2], extracts an m-tuple of strings from a string. This paper assumes that the learner knows the types of intermediate substructures of λ-terms in the object language. More formally, we assume a finite set $\Omega \subseteq \mathscr{T}(A_1)$ such that $\mathscr{L}(p) \in \Omega$ for any abstract atomic type $p \in A_0$ of the target grammar. When the learning target is ACGs that encode CFGs, Ω is just the singleton of the string type str. When the learning target is ACGs that encode LCFTGs, Ω consists of types of the form $o^m \to o$.

Definition 2. Let us fix an object vocabulary $\Sigma_1 = \langle A_1, C_1, \tau_1 \rangle$. For a finite set $\Omega \subseteq \mathscr{T}(A_1)$, a type $\sigma \in \Omega$ and a natural number m, we define a class $\mathbb{ACG}(\Omega, \sigma, m)$ to consist of second-order ACGs $\langle \Sigma_0, \Sigma_1, \mathscr{L}, s \rangle$ such that

- $\mathscr{L}(p) \in \Omega$ for every $p \in A_0$,
- $\mathscr{L}(s) = \sigma$ for the distinguished type s,
- $\tau_0(c) = p_1 \to \cdots \to p_k \to q$ implies $k \leq m$ for any $c \in C_0$.

For example, ACGs that encode CFGs in Chomsky normal form are all in $\mathbb{ACG}(\{\mathit{str}\}, \mathit{str}, 2)$. We assume that the class $\mathbb{ACG}(\Omega, \sigma, m)$ of ACGs is fixed. When P, Q and P' are η-long β-normal inhabitants of types σ, $\alpha \to \sigma$ and α, respectively, and $QP' \twoheadrightarrow_\beta P$, we call Q, P' and $\langle Q, P' \rangle$, respectively, an *α-context*, an *α-subterm* and an *α-decomposition of P*. For a set $\mathcal{D} \subseteq \Lambda(\Sigma_1)$ of η-long β-normal inhabitants of the type σ, let

$$\mathrm{Sub}_\alpha(\mathcal{D}) = \{\, P \mid P \text{ is an } \alpha\text{-subterm of some element of } \mathcal{D} \,\},$$
$$\mathrm{Con}_\alpha(\mathcal{D}) = \{\, Q \mid Q \text{ is an } \alpha\text{-context of some element of } \mathcal{D} \,\},$$
$$\mathrm{Sub}_\Omega(\mathcal{D}) = \bigcup_{\alpha \in \Omega} \mathrm{Sub}_\alpha(\mathcal{D}), \quad \mathrm{Con}_\Omega(\mathcal{D}) = \bigcup_{\alpha \in \Omega} \mathrm{Con}_\alpha(\mathcal{D})\ .$$

Note that $\mathrm{Con}_\sigma(\mathcal{D})$ always contains I_σ unless $\mathcal{D}$ is empty, where I_α denotes the η-long β-normal inhabitant of the type $\alpha \to \alpha$ which η-reduces to $\lambda z.z$. Lemma 3 implies that one can compute those in polynomial time in the size of $\mathcal{D}$.

Corollary 1. *Let us fix α and σ. For any P such that $\vdash_{\Sigma_1} P : \sigma$, one can enumerate all the α-decomposition of P in polynomial time in the size of P.*

Proof. For $P \in \Lambda(\Sigma_1)$ with m occurrences of constants $c_1, \ldots, c_m$, let $\Gamma = \{\, x_i : \tau_1(c_i) \mid 1 \leq i \leq m \,\}$, $\theta = [x_i := c_i]_{1 \leq i \leq m}$, and $\hat{P}$ be a constant-free λ-term such that $P = \hat{P}\theta$. For $\hat{P}'$ and $\hat{Q}$ such that $\Gamma \vdash \hat{P}' : \alpha$ and $\Gamma \vdash \hat{Q} : \alpha \to \sigma$, obtained by applying the procedure presented in Lemma 3 to $\hat{P}$, the pair $\langle \hat{Q}\theta, \hat{P}'\theta \rangle$ is an α-decomposition of P and nothing else can be an α-decomposition of P. □

Remark 1. When the object signature is a string signature, any CFG in Chomsky normal form is encoded by an ACG in $\mathbb{ACG}(\{\mathit{str}\}, \mathit{str}, 2)$, and every ACG in $\mathbb{ACG}(\{\mathit{str}\}, \mathit{str}, 2)$ can be regarded in a straightforward way as an encoding of a CFG. However, it is not the case for LCFRSs. Every LCFRS whose nonterminals have rank at most n and whose rules have at most m nonterminals on the

right hand side is encoded in ACGs in $\mathbb{ACG}(\{tpl_k \mid k \leq n\}, str, m)$, but the class $\mathbb{ACG}(\{tpl_k \mid k \leq n\}, str, m)$ contains ACGs that cannot be regarded as an encoding of any LCFRS by the standard interpretation by de Groote and Pogodalla. In order to apply our discussion to concrete formalisms that one wants to target specifically, the three parameters Ω, σ, m are not enough. One needs to determine a reasonable subclass of $\mathbb{ACG}(\Omega, \sigma, m)$ and to take subsets of $\mathrm{Sub}_\Omega(\mathcal{D})$ and $\mathrm{Con}_\Omega(\mathcal{D})$ according to the encoding of the target formalism. This paper largely ignores such details, but as long as one can decide in polynomial time whether a λ-term corresponds to some construct of the target formalism, which is quite a natural assumption, the discussion of this paper still applies to the learning of the formalism.

Proposition 1. *The membership problem for the class* $\mathbb{ACG}(\Omega, \sigma, m)$ *is uniformly decidable in polynomial time.*

Proof. Salvati [15] and Kanazawa [16] show that the membership problem for a fixed second-order ACG is decidable in polynomial time, where the degree of the polynomial is linear in $\max\{\, |\mathscr{L}(\tau_0(c))| \mid c \in C_0 \,\}$, which has a fixed bound in $\mathbb{ACG}(\Omega, \sigma, m)$. □

In the next section, we will demonstrate how existing distributional learning algorithms can be generalized in terms of ACGs.

5 Examples of Learning Algorithms

5.1 Substitutable ACGs

Among several properties on grammars/languages proposed so far for which distributional learning works, we first pick the strongest property, *substitutability*. Clark and Eyraud [1] and Yoshinaka [2] have shown that *substitutable* CFGs and MCFGs, respectively, are efficiently learnable from positive examples. Those algorithms are quite simple compared to other concrete algorithms of distributional learning.

The learning scheme we assume is identification in the limit from positive data. A learner $\mathcal{L}$ is given an infinite sequence of positive examples $P_1, P_2, \dots$ from the language $\mathcal{O}(\mathscr{G}_*)$ of a grammar $\mathscr{G}_*$ in the target class such that $\mathcal{O}(\mathscr{G}_*) = \{\, P_1, P_2, \dots \,\}$. Each time $\mathcal{L}$ takes a new example P_n, it outputs a conjecture $\mathscr{G}_n$. We say that $\mathcal{L}$ *identifies* $\mathscr{G}_*$ *in the limit from positive data* if for any sequence of positive examples, there is a point n_0 such that $\mathscr{G}_n = \mathscr{G}_{n_0}$ for all $n > n_0$ and $\mathcal{O}(\mathscr{G}_{n_0}) = \mathcal{O}(\mathscr{G}_*)$. This subsection presents an efficient algorithm that identifies all Ω-*substitutable* ACGs in $\mathbb{ACG}(\Omega, \sigma, m)$ in the limit from positive data.

Definition 3. An ACG $\mathscr{G}$ is Ω-*substitutable* if for every $\alpha \in \Omega$,

$$|Q_1 P_1|_\beta, |Q_1 P_2|_\beta, |Q_2 P_1|_\beta \in \mathcal{O}(\mathscr{G}_*) \text{ implies } |Q_2 P_2|_\beta \in \mathcal{O}(\mathscr{G}_*)$$

for any $Q_1, Q_2 \in \mathrm{Con}_\alpha(\mathcal{O}(\mathscr{G}_*))$ and $P_1, P_2 \in \mathrm{Sub}_\alpha(\mathcal{O}(\mathscr{G}_*))$. In this case, we say that P_1 and P_2 are α-*substitutable* in $\mathcal{O}(\mathscr{G}_*)$.

For a finite set $\mathcal{D} \subseteq \mathcal{O}(\mathscr{G}_*)$ of positive examples, our learner constructs an ACG $\mathscr{G}_\mathcal{D} = \langle \Sigma_\mathcal{D}, \Sigma_1, \mathscr{L}_\mathcal{D}, s\rangle$ as follows. First, we take Ω-subterms of elements of $\mathcal{D}$ to be abstract atomic types:

$$A_\mathcal{D} = \{\, [P^\alpha] \mid P \in \mathrm{Sub}_\alpha(\mathcal{D}) \text{ for } \alpha \in \Omega \,\} \cup \{s\} \ .$$

Each atomic type of the form $[P^\alpha]$ is mapped to $\mathscr{L}_\mathcal{D}([P^\alpha]) = \alpha$ and $\mathscr{L}_\mathcal{D}(s) = \sigma$. Corollary 1 ensures that one can compute $A_\mathcal{D}$ in polynomial time. Our grammar $\mathscr{G}_\mathcal{D}$ has lexical entries of the following three types.

A. $\langle \mathsf{A}_{(RP_1^{\alpha_1}\dots P_k^{\alpha_k})^\beta},\ [P_1^{\alpha_1}] \to \dots \to [P_k^{\alpha_k}] \to [P_0^\beta],\ R\rangle$
if $RP_1 \dots P_k \twoheadrightarrow_\beta P_0$ with $k \le m$,
B. $\langle \mathsf{B}_{P_1^\alpha, P_2^\alpha},\ [P_1^\alpha] \to [P_2^\alpha],\ I_\alpha\rangle$ if $|QP_1|_\beta, |QP_2|_\beta \in \mathcal{D}$ for some $Q \in \mathrm{Con}_\alpha(\mathcal{D})$,
C. $\langle \mathsf{C}_{P^\sigma},\ [P^\sigma] \to s,\ I_\sigma\rangle$ if $P \in \mathcal{D}$.

It is clear that lexical entries of Types B and C can be computed in polynomial time. It is also the case for Type A.

Lemma 4. *Fix Σ_1, Ω, and m. Let $\alpha_0, \dots, \alpha_k \in \Omega$, with $k \le m$. Given P_i such that $\vdash_{\Sigma_1} P_i{:}\alpha_i$ for $i = 0, \dots, k$, one can enumerate all Q such that $QP_1 \dots P_k \twoheadrightarrow_\beta P_0$ in polynomial time.*

Proof. By Corollary 1, one can enumerate all η-long β-normal inhabitant Q_k of $\alpha_k \to \alpha_0$ such that $Q_kP_k \twoheadrightarrow_\beta P_0$ in polynomial time.[1] Then for each Q_k, one can enumerate all Q_{k-1} such that $Q_{k-1}P_{k-1} \twoheadrightarrow_\beta Q_k$. By repeatedly applying Corollary 1, one can enumerate all sequences $Q_k, \dots, Q_1$ such that $Q_{i-1}P_{i-1} \twoheadrightarrow_\beta Q_i$ for $i = k, \dots, 2$ and Q_i is an η-long β-normal inhabitant of $\alpha_i \to \dots \to \alpha_k \to \alpha_0$ for $i = k, \dots, 1$ in polynomial time, because we have an upper bound m on k. We have $Q_1P_1 \dots P_k \twoheadrightarrow_\beta P_0$. □

Algorithm 1 shows our learner for Ω-substitutable ACGs in $\mathbb{ACG}(\Omega, \sigma, m)$. It is clear that the algorithm updates its conjecture $\mathscr{G}_n$ in polynomial time in $\sum_{1\le i\le n} \|P_i\|$ where $\|P\|$ denotes the size of P.

The correctness of the algorithm can be shown in the same manner as in [1,2].

Lemma 5. *If $\vdash_{\Sigma_\mathcal{D}} M : [P^\alpha]$, then P and $|\mathscr{L}_\mathcal{D}(M)|_\beta$ are α-substitutable in $\mathcal{O}(\mathscr{G}_*)$.*

Proof. We prove the lemma by induction on M.

Suppose that M has the form $M = \mathsf{A}_{(RP_1^{\alpha_1}\dots P_k^{\alpha_k})^\alpha} M_1 \dots M_k$, where we have $\vdash_{\Sigma_\mathcal{D}} M_i : [P_i^{\alpha_i}]$ for each $i = 1, \dots, k$ and $\mathscr{L}_\mathcal{D}(\mathsf{A}_{(RP_1^{\alpha_1}\dots P_k^{\alpha_k})^\alpha}) = R$. We note that the base case is given when $k = 0$. The presence of the abstract constant implies that there is an α-context $Q \in \mathrm{Con}_\alpha(\mathcal{D})$ such that

$$|QP|_\beta = |Q(RP_1 \dots P_k)|_\beta \in \mathcal{D} \subseteq \mathcal{O}(\mathscr{G}_*) \ .$$

[1] Since P_k is given in this case, we do not need to list all the decompositions of P_0. Only the latter half of the procedure presented in Lemma 3 suffices to get such Q_k.

Algorithm 1. Learning Ω-Substitutable ACGs

Data: A sequence of positive examples $P_1, P_2, \ldots$
Result: A sequence of ACGs $\mathscr{G}_1, \mathscr{G}_2, \ldots$
let $\hat{\mathscr{G}}$ be an ACG such that $\mathcal{O}(\hat{\mathscr{G}}) = \varnothing$;
for $n = 1, 2, \ldots$ **do**
  read the next example P_n;
  if $P_n \notin \mathcal{O}(\hat{\mathscr{G}})$ **then**
    let $\hat{\mathscr{G}} = \mathscr{G}_{\mathcal{D}}$ for $\mathcal{D} = \{P_1, \ldots, P_n\}$;
  end if
  output $\hat{\mathscr{G}}$ as $\mathscr{G}_n$;
end for

For $i = 1, \ldots, k$, let $P'_i = |\mathscr{L}_{\mathcal{D}}(M_i)|_\beta$ and

$$Q_i = |\lambda y_i.Q(RP'_1 \ldots P'_{i-1} Y_i P_{i+1} \ldots P_k)|_\beta$$

where Y_i denotes the η-long inhabitant of $(y_i{:}\alpha_i, \alpha_i)$ that η-reduces to y_i. Clearly Q_i is an α_i-context. By induction hypothesis, P_i and P'_i are α_i-substitutable. Then

$$\begin{aligned} |QP|_\beta = |Q(RP_1 \ldots P_k)|_\beta = |Q_1 P_1|_\beta \in \mathcal{O}(\mathscr{G}_*) \\ \Longrightarrow\ |Q_1 P'_1|_\beta = |Q_2 P_2|_\beta \in \mathcal{O}(\mathscr{G}_*) \ \Longrightarrow\ |Q_2 P'_2|_\beta = |Q_3 P_3|_\beta \in \mathcal{O}(\mathscr{G}_*) \\ \Longrightarrow\ \ldots \ \Longrightarrow\ |Q_k P'_k|_\beta = |Q(RP'_1 \ldots P'_k)|_\beta = |Q\mathscr{L}_{\mathcal{D}}(M)|_\beta \in \mathcal{O}(\mathscr{G}_*)\ . \end{aligned}$$

Therefore, P and $|\mathscr{L}_{\mathcal{D}}(M)|_\beta$ are α-substitutable for each other.

Suppose that M has the form $M = \mathsf{B}_{P_1^\alpha, P^\alpha} M_1$. The presence of the constant $\mathsf{B}_{P_1^\alpha, P^\alpha}$ implies that P_1 and P are α-substitutable. The induction hypothesis on $\vdash_{\Sigma_{\mathcal{D}}} M_1 : [P_1^\alpha]$ is that P_1 and $|\mathscr{L}_{\mathcal{D}}(M_1)|_\beta$ are also α-substitutable. Hence $|\mathscr{L}_{\mathcal{D}}(M)|_\beta = |I_\alpha \mathscr{L}_{\mathcal{D}}(M_1)|_\beta = |\mathscr{L}_{\mathcal{D}}(M_1)|_\beta$, P_1 and P are all α-substitutable. □

Corollary 2. *If* $\vdash_{\Sigma_{\mathcal{D}}} M : s$, *then* $|\mathscr{L}_{\mathcal{D}}(M)|_\beta \in \mathcal{O}(\mathscr{G}_*)$.

Proof. If $\vdash_{\Sigma_{\mathcal{D}}} M : s$, M must have the form $M = \mathsf{C}_{P^\sigma} M'$ where $\vdash_{\Sigma_{\mathcal{D}}} M' : [P^\sigma]$ and $P \in \mathcal{D}$. By Lemma 5, P and $|\mathscr{L}_{\mathcal{D}}(M')|_\beta = |\mathscr{L}_{\mathcal{D}}(M)|_\beta$ are σ-substitutable. In particular for the identity σ-context $I_\sigma \in \mathrm{Con}_\sigma(\mathcal{D})$, we have that $P = |I_\sigma P|_\beta \in \mathcal{O}(\mathscr{G}_*)$ implies $|\mathscr{L}_{\mathcal{D}}(M)|_\beta = |I_\sigma \mathscr{L}_{\mathcal{D}}(M')|_\beta \in \mathcal{O}(\mathscr{G}_*)$. □

Let the target grammar $\mathscr{G}_* = \langle \Sigma_0, \Sigma_1, \mathscr{L}_*, s \rangle$. For each atomic type $p \in A_0$, we let S_p and T_p be the smallest λ-terms such that $\vdash_{\Sigma_0} S_p : p$ and $z : p \vdash_{\Sigma_0} T_p : s$, respectively. Let $\mathcal{E}_* = \{\, T_p[z := cS_{p_1} \ldots S_{p_k}] \mid p \in A_0,\ c \in C_0,\ \tau_0(c) = p_1 \to \cdots \to p_k \to p \,\}$ and $\mathcal{D}_* = \{\, |\mathscr{L}_*(S)|_\beta \in \mathcal{O}(\mathscr{G}_*) \mid S \in \mathcal{E}_* \,\}$.

Lemma 6. *For any* $\mathcal{D} \supseteq \mathcal{D}_*$, *if* $\vdash_{\Sigma_0} N : p$ *with* $p \in A_0$, *there is* $M \in \Lambda(\Sigma_{\mathcal{D}})$ *such that* $\vdash_{\Sigma_{\mathcal{D}}} M : [|\mathscr{L}_*(S_p)|_\beta^{\mathscr{L}_*(p)}]$ *and* $\mathscr{L}_{\mathcal{D}}(M) =_\beta \mathscr{L}_*(N)$.

Proof. By induction on N. Suppose that $N = cN_1 \ldots N_k$ with $\tau_0(c) = p_1 \to \ldots p_k \to p$. Let $P_{p_i} = |\mathscr{L}(S_{p_i})|_\beta$ and $P_0 = |\mathscr{L}_*(cS_{p_1} \ldots S_{p_k})|_\beta$. By definition we

have $|\mathscr{L}_*(T_p[z := cS_{p_1} \dots S_{p_k}])|_\beta \in \mathcal{D}$ and thus we have a lexical entry

$$\left\langle \mathsf{A}_{\mathscr{L}_*(c)^{\mathscr{L}_*(\tau_0(c))} S_{p_1}^{\mathscr{L}_*(p_1)} \dots S_{p_k}^{\mathscr{L}_*(p_k)}},\ [P_{p_1}^{\mathscr{L}_*(p_1)}] \to \dots \to [P_{p_k}^{\mathscr{L}_*(p_k)}] \to [P_0^{\mathscr{L}_*(p)}],\ \mathscr{L}_*(c) \right\rangle.$$

By induction hypothesis, we have $M_i \in \Lambda(\Sigma_\mathcal{D})$ such that $\vdash_{\Sigma_\mathcal{D}} M_i : [P_{p_i}^{\mathscr{L}_*(p_i)}]$ and $\mathscr{L}_\mathcal{D}(M_i) =_\beta \mathscr{L}_*(N_i)$. Thus for $M' = \mathsf{A}_{\mathscr{L}_\mathcal{D}(c)^{\mathscr{L}(\tau_0(c))} S_{p_1}^{\mathscr{L}(p_1)} \dots S_{p_k}^{\mathscr{L}(p_k)}} M_1 \dots M_k$, we have $\vdash_{\Sigma_\mathcal{D}} M' : [P_0^{\mathscr{L}_*(p)}]$ and $\mathscr{L}_\mathcal{D}(M') =_\beta \mathscr{L}_*(N)$.

By $T_p[z := S_p], T_p[z := cS_{p_1} \dots S_{p_k}] \in \mathcal{E}_*$, we have $|\mathscr{L}_*(\lambda z.T_p)P_p|_\beta$, $|\mathscr{L}_*(\lambda z.T_p)P_0|_\beta \in \mathcal{D}$ for $P_p = |\mathscr{L}_*(S_p)|_\beta$. Hence the conjectured grammar has the lexical entry

$$\left\langle \mathsf{B}_{P_0^{\mathscr{L}_*(p)}, P_p^{\mathscr{L}_*(p)}},\ [P_0^{\mathscr{L}_*(p)}] \to [P_p^{\mathscr{L}_*(p)}],\ I_{\mathscr{L}_*(p)} \right\rangle$$

and the lemma holds for $M = \mathsf{B}_{P_0^{\mathscr{L}_*(p)}, P_p^{\mathscr{L}_*(p)}} M'$. □

Corollary 3. *$\mathcal{O}(\mathscr{G}_*) = \mathcal{O}(\mathscr{G}_\mathcal{D})$ if $\mathcal{D} \supseteq \mathcal{D}_*$.*

Therefore, once the learner gets a superset of $\mathcal{D}_*$, it always conjectures an ACG that generates the target language. We also remark that the set $\mathcal{D}_*$ is not too big. $\mathcal{D}_*$ consists of at most $|C_0|$ positive examples and each element in $\mathcal{D}_*$ has the smallest abstract derivation structure that involves a constant $c \in C_0$.

5.2 ACGs with the Finite Kernel Property

Clark et al. [17] have proposed an algorithm that learns CFGs with the *Finite Kernel Property* and their result is generalized by Yoshinaka [3] to MCFGs. The learning scheme they use is *identification in the limit from positive data and membership queries*, which is the same as the scheme in the previous subsection except that the learner can query an oracle whether an arbitrary object is in the learning target and receive an answer in constant time. In order to simplify the definition of the FKP, we allow each ACG to have multiple distinguished types $s_1, \dots, s_k$. These types must all be mapped to the same type by the lexicon.

Definition 4. We say that $\mathscr{G} \in \mathbb{ACG}(\Omega, \sigma, m)$ has the *Finite Kernel Property* (FKP) if for each abstract atomic type $p \in A_0$, there is an $\mathscr{L}(p)$-subterm P_p of an element of $\mathcal{O}(\mathscr{G})$ such that

$$\{\, Q \in \mathrm{Con}_{\mathscr{L}(p)}(\mathcal{O}(\mathscr{G})) \mid |QP_p|_\beta \in \mathcal{O}(\mathscr{G}) \,\} =$$
$$\{\, Q \in \mathrm{Con}_{\mathscr{L}(p)}(\mathcal{O}(\mathscr{G})) \mid |Q\mathscr{L}(M)|_\beta \in \mathcal{O}(\mathscr{G}) \text{ for all } M \text{ with } \vdash_{\Sigma_0} M : p \,\}\ .$$

Algorithm 2 describes our learner for ACGs with the FKP. We refrain from explaining the details of the algorithm and proving the correctness and the efficiency of it, as it is just a translation of existing algorithms, and a general idea on translation has been provided in the previous subsection. The definition of our conjecture $\mathscr{G}(\mathcal{S}, \mathcal{C}) = \langle \Sigma_{\mathcal{S},\mathcal{C}}, \Sigma_1, \mathscr{L}_{\mathcal{S},\mathcal{C}}, \{s\} \rangle$ is given as follows.

$$A_{\mathcal{S},\mathcal{C}} = \{\, [P^\alpha] \mid P \in \mathcal{S}_\alpha \text{ for } \alpha \in \Omega \,\} \cup \{s\},$$

Algorithm 2. Learning ACGs with the FKP

Data: A sequence of strings $P_1, P_2, \cdots \in \mathcal{O}(\mathscr{G}_*)$;
Result: A sequence of ACGs $\mathscr{G}_1, \mathscr{G}_2, \cdots \in \mathbb{ACG}(\Omega, \sigma, m)$
let $\mathcal{D} := \varnothing$; $\hat{\mathscr{G}} := \mathscr{G}(\varnothing, \varnothing)$;
for $n = 1, 2, \ldots$ **do**
  let $\mathcal{D} := \mathcal{D} \cup \{P_n\}$; $\mathcal{C}_\alpha := \mathrm{Con}_\alpha(\mathcal{D})$ for each $\alpha \in \Omega$;
  if $\mathcal{D} \nsubseteq \mathcal{O}(\hat{\mathscr{G}})$ **then**
    let $\mathcal{S}_\alpha := \mathrm{Sub}_\alpha(\mathcal{D})$ for each $\alpha \in \Omega$;
  end if
  output $\hat{\mathscr{G}} = \mathscr{G}(\bigcup_{\alpha\in\Omega} \mathcal{S}_\alpha, \bigcup_{\alpha\in\Omega} \mathcal{C}_\alpha)$ as $\mathscr{G}_n$;
end for

Each atomic type of the form $[P^\alpha]$ is mapped to $\mathscr{L}_{\mathcal{S},\mathcal{C}}([P^\alpha]) = \alpha$ and $\mathscr{L}_{\mathcal{S},\mathcal{C}}(s) = \sigma$. We have lexical entries of the following three types.

A. $\langle \mathsf{A}_{(RP_1^{\alpha_1}\ldots P_k^{\alpha_k})^\beta},\ [P_1^{\alpha_1}] \to \cdots \to [P_k^{\alpha_k}] \to [P_0^\beta],\ R\rangle$
   if $RP_1 \ldots P_k \twoheadrightarrow_\beta P_0$ with $k \le m$,
B. $\langle \mathsf{B}_{P_1^\alpha, P_2^\alpha},\ [P_1^\alpha] \to [P_2^\alpha],\ I_\alpha\rangle$
   if $|QP_2|_\beta \in \mathcal{O}(\mathscr{G}_*)$ implies $|QP_1|_\beta \in \mathcal{O}(\mathscr{G}_*)$ for all $Q \in \mathcal{C}_\alpha$,
C. $\langle \mathsf{C}_{P^\sigma},\ [P^\sigma] \to s,\ I_\sigma\rangle$ if $P \in \mathcal{D}$.

The only difference from the construction of conjectures in the case of learning Ω-substitutable ACGs is the condition for lexical entries of Type B. This is decided in polynomial time with the aid of the membership oracle.

Theorem 6. *Algorithm 2 identifies* ACG*s with the* FKP *in* $\mathbb{ACG}(\Omega, \sigma, m)$ *in the limit from positive data and membership queries.*

6 Discussions

This paper has demonstrated how substring–context relation should be generalized and explained in terms of ACGs. For the sake of the clarity, we parameterize only a set Ω of types to specify classes of our learning target. As discussed in Remark 1, however, it is easy to bring into our learning algorithm other parameters to pick out classes of ACGs that exactly match specific classes of string or tree grammars in well-known formalisms.

Among several results of distributional learning, we have picked two properties and learning algorithms and demonstrated how they should be explained in terms of ACGs. Other results are also easily translated by the same idea. For example, one can reasonably define *congruential* ACG*s* and give a learning algorithm for them under the same learning scheme as in the preceding work [18,4].

Acknowledgement

This work was supported in part by Mext Kakenshi 20700124 and by the NII joint research project "Open Problems on Multiple Context-Free Grammars".

References

1. Clark, A., Eyraud, R.: Polynomial identification in the limit of substitutable context-free languages. Journal of Machine Learning Research 8, 1725–1745 (2007)
2. Yoshinaka, R.: Learning mildly context-sensitive languages with multidimensional substitutability from positive data. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) ALT 2009. LNCS, vol. 5809, pp. 278–292. Springer, Heidelberg (2009)
3. Yoshinaka, R.: Polynomial-time identification of multiple context-free languages from positive data and membership queries [19], pp. 230–244
4. Yoshinaka, R., Clark, A.: Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In: Proceedings of the 15th Conference on Formal Grammar, Copenhagen, Denmark (2010)
5. Clark, A.: Towards general algorithms for grammatical inference. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS, vol. 6331, pp. 11–30. Springer, Heidelberg (2010)
6. de Groote, P.: Towards abstract categorial grammars. In: Association for Computational Linguistics, Proceedings of the Conference on 39th Annual Meeting and 10th Conference of the European Chapter, pp. 148–155 (2001)
7. de Groote, P.: Tree-adjoining grammars as abstract categorial grammars. In: TAG+6, Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks, Università di Venezia, pp. 145–150 (2002)
8. de Groote, P., Pogodalla, S.: On the expressive power of abstract categorial grammars: Representing context-free formalisms. Journal of Logic, Language and Information 13(4), 421–438 (2004)
9. Courcelle, B.: An axiomatic definition of context-free rewriting and its application to NLC graph grammars. Theoretical Computer Science 55(2-3), 141–181 (1987)
10. Hindley, J.R.: Basic Simple Type Theory. Cambridge University Press, Cambridge (1997)
11. Hirokawa, S.: Balanced formulas, BCK-minimal formulas and their proofs. In: Nerode, A., Taitslin, M.A. (eds.) LFCS 1992. LNCS, vol. 620, pp. 198–208. Springer, Heidelberg (1992)
12. Babaev, A., Soloviev, S.: A coherence theorem for canonical morphism in cartesian closed categories. Zapiski nauchnykh Seminarov Lenigradskogo Otdeleniya matematichskogo Instituta im. V.A. Steklova An SSSR 88, 3–29 (1979)
13. Mints, G.: A short introduction to intuitionistic logic. Kluwer Academic/Plenum Publishers, New York (2000)
14. Bunder, M.W.: Proof finding algorithms for implicational logics. Theoretical Computer Science 232(1-2), 165–186 (2000)
15. Salvati, S.: Problèmes de Filtrage et Problèmes d'analyse pour les Grammaires Catégorielles Abstraites. PhD thesis, L'Institut National Polytechnique de Lorraine (2005)
16. Kanazawa, M.: Parsing and generation as Datalog queries. In: Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, pp. 176–183 (2007)
17. Clark, A., Eyraud, R., Habrard, A.: Using contextual representations to efficiently learn context-free languages. Journal of Machine Learning Research 11, 2707–2744 (2010)
18. Clark, A.: Distributional learning of some context-free languages with a minimally adequate teacher [19], pp. 24–37
19. Sempere, J.M., García, P. (eds.): ICGI 2010. LNCS, vol. 6339. Springer, Heidelberg (2010)

Some Generalised Comparative Determiners

Richard Zuber

Rayé des cadres du CNRS, Paris, France
Richard.Zuber@linguist.jussieu.fr

Abstract. Functions denoted by specific comparative expressions called *generalised comparative determiners* are analysed. These expressions form verb arguments when applied to common nouns. They denote functions which take sets and a binary relation as argument and give a set as result. These functions are thus different from denotations of "ordinary" determiners. However, they do obey some similar constraints, properly generalised. It is shown that verbal arguments obtained from such generalised determiners extend the expressive power of NLs since functions that they denote are not just case extensions of type $\langle 1 \rangle$ quantifiers used to interpret "ordinary" determiner phrases.

1 Introduction

The basic meaning of the notion of determiner as established in modern formal linguistics is that of an expression which when applied to one (or more) common nouns (CNs) forms a noun phrase (NP) or rather a determiner phrase (DP). NPs are (syntactic) arguments of verb phrases and thus they can occur in various argumental positions, for instance as grammatical subjects, grammatical objects or various oblique objects. When a DP occurs in the direct object position (of a simple sentence of the form NP TVP DP) one can consider that the determiner which forms it denotes a function which takes a set (the denotation of the CN) and a binary relation (denotation of the TVP) and gives a set (the denotation of the TVP+DP) as result. In this paper I will discuss some functions of this type in cases where they are denoted not by "ordinary" determiners but by what can be called *generalised determiners*, that is expressions which take one or more common nouns as arguments and which give a verbal argument as result.

There are verbal arguments which cannot occur in every argumental position and thus are not strictly speaking NPs. Consider (1), (2) and (3):

(1) Leo washed himself.

(2) Leo washed himself and Lea.

(3) a. *Himself washed Leo.
b. *Himself and Lea washed them/themselves/Leo.

In (1) *himself* occurs in direct object position and is thus similar to an ordinary noun phrase. In addition it can be conjoined with an ordinary NP to make a Boolean complex NP which can occur in the direct object position, as shown

S. Pogodalla and J.-P. Prost (Eds.): LACL 2011, LNAI 6736, pp. 267–281, 2011.

in (2). However, neither *himself* alone nor its Boolean composition with an ordinary NP can occur in the subject position, as seen in (3a) and (3b).

The case of *himself* and of its Boolean compound indicated above is well known: it illustrates (nominal) anaphora. What is less well-known is the fact that there are expressions forming nominal anaphors in a way similar to the one in which "ordinary" DPs are formed: they can be formed by an application of an *anaphoric determiner* to a CN. To see this consider the following examples:

(4) Leo shaved every philosopher except himself.

(5) Lea admires most philosophers, including herself.

In (4) the DP-like expression *every philosopher except himself* and in (5) the DP-like expression *most philosophers, including herself* are (complex) nominal anaphors. What is important for us is that they are formed by the application of (complex) determiner-like expressions *every..., except himself* and *most..., including herself*, respectively, to the common noun *philosophers*. These expressions are called anaphoric determiners and, obviously, anaphoric determiners are generalised determiners.

It is possible to mention many differences between functions denoted by ordinary determiners and functions denoted by anaphoric determiners (Zuber 2010a). In particular it is useful to consider that anaphoric determiners denote functions from sets (denotations of common nouns) to functions from binary relations (denotations of transitive verbs) to sets (denotations of verb phrases). Or, equivalently, anaphoric determiners denote functions which take a set (or sets) and a binary relation as argument and give a set as result.

Some languages, for instance Scandinavian, Slavic or Latin, have possessive anaphoric determiners which are morphologically simple, in contraposition to those given in (4) and (5) . Slavic type anaphoric determiners and various properties of functions denoted by them are studied in Zuber 2010b and Zuber 2010c.

In his paper I am going to study another sub-class of generalised determiners, which I will call *comparative determiners*. I will in particular show that functions denoted by comparative determiners, though of the same type as functions denoted by anaphoric determiners, have different formal properties.

As an example of a comparative (generalised) determiner consider (6):

(6) Leo knows more philosophers than Lea.

The expression *more...than Lea* in (6) takes the common noun *philosophers* as argument and makes an expression which can be considered as "syntactic" argument of the verb *know*. In addition this argument cannot occur in the subject position. Consequently *more...than Lea* is a generalised determiner.

2 Formal Preliminaries

The following technical and notational preliminary will be useful. Since we are interested basically in comparatives and proportional quantifiers, both of which

are naturally interpreted only in finite universes, we will assume that our universe of discourse E is finite. Thus all sets (of individuals) considered are sub-sets of E. For any set A, $|A|$ is the cardinality of A and for any binary relation R the set aR is defined as follows: $aR = \{x : \langle a, x\rangle \in R\}$.

Since the functions I will discuss are ofen related to quantifiers let me recall some basic notions concerning generalised quantifiers. Functions from sets (sub-sets of E) to truth-values are type $\langle 1\rangle$ quantifiers. Functions from pairs of sets to truth-values or binary relations between sets are type $\langle 1, 1\rangle$ quantifiers. They are denotations of unary determiners. Type $\langle 1, 1, 1\rangle$ quantifiers are ternary relations between sets. They are denotations of binary determiners. In fact for some syntactic reasons, one can distinguish two sub-classes of type $\langle 1, 1, 1\rangle$ quantifiers: (1) quantifiers whose type is noted $\langle\langle 1, 1\rangle 1\rangle$ and (2) quantifiers whose type is noted $\langle 1\langle 1, 1\rangle\rangle$. The first class corresponds to denotations of binary determiners which take two nominal arguments (as in *More students than teachers danced*) and the second class corresponds to denotations of binary determiners which take two predicative arguments (as in *More students danced than sang*).

We will also use the following notation for functions from sets or relations to sets. A type $\langle 1, 2 : 1\rangle$ function is a function having a set and a binary relation as argument and giving a set as result. A type $\langle 1, 1, 2 : 1\rangle$ function is a function which takes two sets and one binary relation as arguments and gives a set as a result. Such functions are denoted by unary and binary generalised determiners.

We are interested in the interpretation of sentences of the form $NP\ TVP\ GDP$ where TVP is a transitive verb phrase and GDP is a generalised determiner phrase (an expression obtained by the application of a generalised determiner to a common noun). In such sentences NP is interpreted by a type $\langle 1\rangle$ quantifier, which is a set of sets, and $TVPs$ is interpreted by a binary relation. Concerning GDP there are two possibilities: if it is an "ordinary" DP, it is interpreted by a function which is an accusative extension of a type $\langle 1\rangle$ quantifier or, if it is not an DP it is interpreted by a function (from binary relations to sets) which is not an accusative extension of a type $\langle 1\rangle$ quantifier. An accusative extension Q_{acc} of a type $\langle 1\rangle$ quantifier Q is defined as follows (Keenan 1988):

Definition 1. *For each type* $\langle 1\rangle$ *quantifier* Q, $Q_{acc}(R) = \{a : Q(aR) = 1\}$.

Thus the accusative extension of a quantifier is a function from binary relations to sets induced by the quantifier in the way indicated in Definition 1. Accusative extensions of quantifiers permit one to compute directly denotations of verb phrases formed from transitive verb phrases and a noun phrase in the position of the direct object.

Accusative extensions of type $\langle 1\rangle$ quantifiers are specific type $\langle 2 : 1\rangle$ functions. They are specific because they satisfy the following accusative extension condition **AE** (Keenan and Westerstahl 1997):

Definition 2 (AE). *A type* $\langle 2 : 1\rangle$ *function* F *satisfies the* accusative extension *condition (**AE**) iff for* R *and* S *binary relations, and* $a, b \in E$, *if* $aR = bS$ *then* $a \in F(R)$ *iff* $b \in F(S)$.

Not all type $\langle 2:1\rangle$ functions satisfy **AE**. For instance the function $SELF$ defined as $SELF(R) = \{x : \langle x, x\rangle \in R\}$ which interprets the reflexive pronoun (as it occurs in (1) for instance) does not satisfy **AE**. Similarly the type $\langle 2:1\rangle$ function $F(R) = MORE_{l,Ph}(R) = \{x : |xR \cap Ph| > |lR \cap Ph|\}$ (where l is the individual referred to by *Lea* and Ph is the set of philosophers), which is denoted by the verbal argument *more philosophers than Lea* in (6), does not satisfy **AE**.

The two functions above satisfy conditions weaker than **AE**. The function $SELF$ satisfies the following anaphor condition **AC**:

Definition 3 (AC). *A type* $\langle 2:1\rangle$ *function* F *satisfies the* anaphor condition ***(AC)*** *iff for* R *and* S *binary relations, and* $a \in E$*, if* $aR = aS$ *then* $a \in F(R)$ *iff* $a \in F(S)$*.*

The **AC** condition, sometimes called *predicate invariance* (Keenan and Westerstahl 1997), is obviously weaker than **AE**. The function $MORE_{l,Ph}$ above satisfies another weakening of **AE**, the so-called *argument invariance* condition **AI** defined in definition 4:

Definition 4 (AI). *A type* $\langle 2:1\rangle$ *function* F *is* argument invariant ***(AI)*** *iff for any binary relation* R *and* $a, b \in E$*, if* $aR = bR$ *then* $a \in F(R)$ *iff* $b \in F(R)$*.*

The conditions **AE**, **AC** and **AI** concern type $\langle 2:1\rangle$ functions, considered here as being denoted by "full" verbal arguments. We are interested in denotations of generalised determiners, that is expressions forming verbal arguments when applied to common nouns. When such determiners are unary, that is when they apply to one common noun, they denote type $\langle 1, 2:1\rangle$ functions. The accusative extension condition for such functions is as follows:

Definition 5 (D1AE). *A type* $\langle 1, 2:1\rangle$ *function* F *satisfies the* accusative extension condition for unary determiners ***(D1AE)*** *iff for* R *and* S *binary relations,* $X \subseteq E$ *and* $a, b \in E$*, if* $aR \cap X = bS \cap X$ *then* $a \in F(X, R)$ *iff* $b \in F(X, S)$*.*

Denotations of ordinary determiners occurring in DPs which take direct object position satisfy **D1AE**. More precisely if D is a type $\langle 1, 1\rangle$ quantifier, then the function $F(X, R) = D(X)_{acc}(R)$ satisfies **D1AE**. Denotations of anaphoric determiners do not satisfy **D1AE**. For instance the function $F(X, R) = \{y : X \cap yR = \{y\}\}$ denoted by the anaphoric determiner *no... except himself/herself* does not satisfy **D1AE**.

Anaphoric functions satisfy the following condition (Zuber 2010c):

Definition 6 (D1AC). *A type* $\langle 1, 2:1\rangle$ *function* F *satisfies* anaphor condition for unary determiners ***(D1AC)*** *iff for* R *and* S *binary relations* $X \subseteq E$*, and* $a \in E$*, if* $aR \cap X = aS \cap X$ *then* $a \in F(X, R)$ *iff* $a \in F(X, S)$*.*

In this article we are interested in functions denoted by comparative generalised determiners. As we will see they do not satisfy **D1AC**. The condition which characterises such functions is as follows:

Definition 7 (D1AI). *A type* $\langle 1, 2:1\rangle$ *function* F *satisfies* argument invariance for unary determiners ***(D1AI)*** *iff for any binary relation* R*,* $X \subseteq E$ *and* $a, b \in E$*, if* $aR \cap X = bR \cap X$ *then* $a \in F(R)$ *iff* $b \in F(R)$*.*

The following property gives a justification of condition **D1AI**:

Proposition 1. *If the function F of type $\langle 1,2:1\rangle$ satisfies D1AI then the function G^A of type $\langle 2:1\rangle$ defined as $G^A(R) = F(A,R)$ satisfies AI.*

What proposition 1 informally says is that functions satisfying **D1AI** are those from which we get functions satisfying **AI** when fixing their set argument.

Functions from binary relations and sets to sets which satisfy **D1AI** have the following obvious property:

Proposition 2. *If a function F from binary relations to sets satisfies D1AI then for any $X, Y \subseteq E$ one has $F(X, E \times Y) = \emptyset$ or $F(X, E \times Y) = E$.*

The conditions specified above are related to the fact that "proper" anaphors, anaphoric determiners or comparative generalised determiners cannot occur in subject position. We have seen that such expressions are close, in some sense, to expressions denoting various quantifiers. As we will see functions denoted by generalised comparative determiners, in addition to property of argument invariance satisfy a natural generalisation of the conservativity property characteristic of "ordinary" determiners.

Conservativity and related properties (intersectivity, etc) are properties of quantifiers. It is possible, however to naturally generalise this notion to functions of the type studied here. We have the following definition (Zuber 2010a):

Definition 8. *A function F of type $\langle 1,2:1\rangle$ is* conservative *iff $F(X,R) = F(X,(E\times X)\cap R)$*

The following property gives plausibility to the above definition of generalised conservativity (Zuber 2010b):

Proposition 3. *Let D be a type $\langle 1,1\rangle$ quantifier and F a type $\langle 1,2:1\rangle$ function defined as: $F(X,R) = D(X)_{acc}(R)$. Then F is conservative iff D is conservative.*

It is easy to check that the anaphoric function denoted by the anaphoric determiner *no... except himself* and the comparative function denoted by the generalised determiner *more...than Lea* (both mentioned above) are conservative. As is well established conservative quantifiers have various important sub-classes, such as classes of intersective, cardinal, etc. quantifiers (cf. Keenan and Westerstahl 1997). It is also possible to generalise the notions of intersective, co-intersective and cardinal quantifiers in such a way that they apply to type $\langle 1,2:1\rangle$ functions. Thus we have the following definitions (Zuber 2010a):

Definition 9. *A type $\langle 1,2:1\rangle$ function is* intersective *(resp.* co-intersective*) iff $F(X_1,R_1) = F(X_2,R_2)$ whenever $(E\times X_1)\cap R_1 = (E\times X_2)\cap R_2$ (resp. $(E\times X_1)\cap R_1' = (E\times X_2)\cap R_2'$).*

The following proposition, similar to proposition 1, can be considered as justifying the above definition:

Proposition 4. *Let D be a type $\langle 1,1\rangle$ quantifier and F a type $\langle 1,2:1\rangle$ function defined as: $F(X,R) = D(X)_{acc}(R)$. Then F is intersective (resp. co-intersective) iff D is intersective (resp. co-intersective).*

It is easy to see that both functions mentioned above are intersective.

Concerning co-intersective functions it is easy to show that the function $EVERY(X)$-BUT-$SELF(R)$ defined in (7), and denoted by the anaphoric determiner *every... but himself*, is co-intersective:

(7) $EVERY(X)$-BUT-$SELF(R) = \{x : X \cap xR' = \{x\}\}$

It is also possible to generalise other sub-properties of conservativity. Consider so-called cardinal quantifiers. A type $\langle 1,1\rangle$ quantifier F is cardinal iff $F(X_1)(Y_1) = F(X_2)(Y_2)$ whenever $|X_1 \cap Y_1| = |X_2 \cap Y_2|$; numerals denote cardinal quantifiers. Type $\langle 1,2:1\rangle$ cardinal functions are defined as follows:

Definition 10. *A type* $\langle 1,2:1\rangle$ *function is* cardinal *iff* $F(X_1,R_1) = F(X_2,R_2)$ *whenever* $\forall y(|X_1 \cap yR_1| = |X_2 \cap yR_2|)$

Obviously, cardinal, intersective and cardinal functions are conservative.

In the next section we will discuss various examples of comparative generalised determiners which denote functions having one of the properties defined in definitions 8, 9 and 10.

3 Some Unary Comparative Determiners

In this section I will basically discuss numerical comparative (and superlative) generalised determiners, that is, semantically, quantifiers and more generally functions applying to sets and relations involving (numerical) comparisons of cardinalities of various sets. In that way the results will be clear and at the same time directly extensible to the case of adjectival comparison. However, I will discuss adjectival comparative (superlative) constructions only briefly since they give rise to generalised determiners only in a special case.

Let me first clarify the distinction between what have been sometimes called *comparative quantifiers* and comparative functions in which we are interested here and which are not comparative quantifiers (or their case extensions).

Numerical comparisons make it clear that various entities can be compared in comparative constructions. To show this, let me present two numerical comparative constructions in semantics of which binary (comparative) quantifiers but not comparative type $\langle 1,2:1\rangle$ functions are involved. In (8) we have comparative constructions involving what are now known as binary determiners (Keenan and Moss 1985, Beghelli 1994, Zuber 2009):

(8) a. More students than priests dance.
b. More students dance than talk.

In (8a) we have a binary determiner denoting a type $\langle\langle 1,1\rangle 1\rangle$ quantifier and in (8b) a binary determiner denoting a type $\langle 1\langle 1,1\rangle\rangle$ quantifier. Their semantics is given in (9) and (10) respectively:

(9) $MORE(S)THAN(P)(D) = |S \cap D| > |P \cap D|$

(10) $MORE(S)(D)THAN(T) = |S \cap D| > |S \cap T|$

We know that DPs formed with binary determiners occur in object position:

(11) Leo met more students than priests.

One can say that in (11) we have a binary generalised determiner which denotes a type $\langle 1, 1, 2:1\rangle$ function. But clearly this is not the case of a genuine generalised determiner: the DP it forms can occur in subject position (as in (8a), the function it denotes is just the accusative extension of the function given in (9)). Later on we will see that there are genuine generalised determiners which denote type $\langle 1, 1, 2:1\rangle$ functions.

Consider example (12), similar to the one in (6) used above to illustrate the case of genuine generalised determiners:

(12) Leo met more students than Lea.

When comparing (12) with (11) we see that different things are compared in these sentences: in (11) we compare properties that one individual has whereas in (12) we compare individuals which share one (non trivial) property.

In (12) we have a genuine generalised determiner, *more...than Lea*. It is genuine because the function it denotes is not an accusative extension of a type $\langle 1, 1\rangle$ quantifier. Informally this can be seen by the following reasoning in which condition **D1AE** is used. Suppose the set of Xs that Bill met is the same as the set of Xs that Leo met. It does not follow from this that the sentence *Bill met more Xs than Lea* has the same truth value as the sentence *Leo met more Xs than Lea*.

The function which is denoted by the generalised determiner *more...than Lea* is given in (13):

(13) $F(X, R) = MORE_l(X, R) = \{y : |yR \cap X| > |lR \cap X|\}$, where l is the individual referred to by *Lea*

It is obvious that the function in (13) satisfies the argument invariance for unary determiners. Indeed, suppose that (14a) holds. Then clearly (14b) holds as well:

(14) a. $aR \cap X = bR \cap X$
b. $a \in MORE_l(X, R)$ iff $b \in MORE_l(X, R)$

It is also easy to show that function $MORE_l$ is conservative. We have to show that $MORE_l(X, R) = MORE_l(X, (E \times X) \cap R)$. But this is obvious given the semantics in (13) and the fact that $a((E \times X) \cap R) = X \cap aR$.

In fact, the function $MORE_l$ has a stronger property than conservativity: it is cardinal (in the sense of definition 10 above). To show this we have to prove that (15) holds if (16) holds:

(15) $MORE_l(X_1, R_1) = MORE_l(X_2, R_2)$
(16) $\forall y(|X_1 \cap yR_1| = |X_2 \cap yR_2|)$

We have the following chain of equivalent statements: $x \in MORE_l(X_1, R_1)$ iff (given (13)) $|xR_1 \cap X_1| > |lR_1 \cap X_1|$ iff (given (16)) $|xR_2 \cap X_2| > |lR_2 \cap X_2|$ iff $x \in MORE_l(X_2, R_2)$. Thus $MORE_l$ is a cardinal type $\langle 1, 2 : 1\rangle$ function.

The above example suggests various questions concerning the number, the variety of patterns and the specificity of properties of generalised comparative determiners and of the functions they denote. For instance one would like to know, roughly speaking, whether there are many other patterns of generalised comparative determiners which denote cardinal functions, whether there are generalised determiners which denote intersective non cardinal functions, or which denote conservative non intersective functions, etc. of course such questions cannot be fully answered here and, in addition some of them should be made more precise. In what follows I will try to give partial answers to some of these questions.

Observe first that in the above example we can replace the expression *more* by *less* or *the same number (of)*. Obviously the generalised determiners thus obtained will still denote cardinal functions.

We can still make generalised determiners more complex by adding explicitly numerals to the determiners discussed above. Consider the example in (17). It is probably ambiguous with the readings indicated in (18a) and (18b):

(17) Leo read 5 more books than Lea.

(18) a. Leo read exactly 5 more books than Lea.
b. Leo read at least 5 more books than Lea.

In the above examples we have the generalised comparative determiners *5 more... than Lea* and *exactly 5 more... than Lea*. They denote type $\langle 1, 2 : 1\rangle$ functions which are instances of functions given in (19a) and (19b) respectively:

(19) a. $EXACTLY\text{-}MORE_{n,i}(X, R) = \{x : |xR \cap X| = |iR \cap X| + n\}$
b. $AT\text{-}LEAST\text{-}MORE_{n,i}(X, R) = \{x : |xR \cap X| > |iR \cap X| + n\}$

It is easy to show that functions in (19a) and (19b) are cardinal and satisfy condition **D1AI**. Moreover, the above examples show that there is an infinite number of generalised determiners denoting cardinal functions.

It is still possible to make comparative generalised determiners more complex by making Boolean compounds of them. Such Boolean complex determiners are given in the following examples:

(20) Leo read more books than Lea but less than 17.

(21) Leo read more books than Lea and more than Bill.

(22) Leo read less books than Lea but more than Bill.

Again, functions involved in the semantics of above examples are cardinal functions. This follows from the observation that functions discussed here form Boolean algebras, the fact that I will not comment in more detail.

Recall that cardinal functions are intersective ones. One may wonder whether there are generalised comparative determiners denoting intersective non cardinal functions. I think that a good example of such a determiner is *the same...as Lea*

as it occurs in (23) . The function denoted by this determiner is an instance of the function given in (24):

(23) Leo read the same books as Lea.

(24) $THE\text{-}SAME_i(X, R) = \{y : yR \cap X = iR \cap X\}$.

The function in (24) is intersective. We have to show that (26) follows from (25). This is true because (25) entails the two equalities given in (27):

(25) $(E \times X_1) \cap R_1 = (E \times X_2) \cap R_2$

(26) $\{y : yR_1 \cap X_1 = iR_1 \cap X_1\}$ iff $\{y : yR_2 \cap X_2 = iR_2 \cap X_2\}$

(27) (i) $yR_1 \cap X_1 = yR_2 \cap X_2$, (ii) $iR_1 \cap X_1 = iR_2 \cap X_2$

We get the needed result by replacing in (26) the equal parts indicated in (27).

It is also easy to show that function $THE\text{-}SAME_i$ given in (24) does not satisfy **D1AE** and that it satisfies **D1AI**.

In quite the same way one can show that the generalised determiner *different... than Lea*, as it occurs in (28), denotes an intersective comparative function:

(28) Leo knows different languages than Lea.

One can consider that (28) is ambiguous with possible readings in which the set of languages known by Leo and by Lea are either "just" different or they are disjoint. These two readings can be distinguished by two different functions $DIFFERENT_i$ indicated in (29a) and (29b) respectively:

(29) a. $DIFFERENT_i(X, R) = \{y : yR \cap X \neq iR \cap X\}$

b. $DIFFERENT_i(X, R) = \{y : yR \cap X \cap iR \cap = \emptyset\}$

Both functions, the one in (29a) and the one in (29b), are intersective.

All the examples of generalised determiners presented above contain a complementizer *than* or *as*. We discussed basically the cases when these complementizers are followed by a proper name. In fact this is not necessary: they can be followed by virtually any NP. Thus we can have generalised determiners like *more... than most students, the same... as some philosophers, different ... than the ten teachers*, etc. The presentation of semantics of such determiners necessitates the generalisation of the notation aR to the notation $Q_{nom}(R)$, where Q is a type $\langle 1 \rangle$ quantifier, a denotation of an NP. $Q_{nom}(R)$, the nominative extension of Q is defined as follows:

(30) $Q_{nom}(R) = \{x : Q(Rx) = 1\}$, where $Rx = \{y : \langle y, x \rangle \in R\}$

The nominal extension of a type $\langle 1 \rangle$ quantifier is different from its accusative extension. The difference can be illustrated by the following example. Let $Q = MOST(S)$. Then $MOST(S)_{nom}(R)$ is the set of objects to which most Ss are in the relation R whereas $MOST(S)_{acc}(R)$ is the set of objects which are in the relation R to most Ss.

To see the usefulness of the notation $Q_{nom}(R)$ consider the following example in (31). The generalised determiner *the same... as most students* occurring in (31) denotes the function $THE\text{-}SAME_{MOST(S)}$ given in (32) :

(31) Leo knows the same languages as most students.

(32) $THE\text{-}SAME_{MOST(S)}(X,R) = \{y : yR \cap X = MOST(S)_{nom}(R) \cap X\}$

The extension of the complements of *than* (or of *as*) enriches, though somewhat trivially, all the classes of generalised comparative determiners we have considered. There is, however one class of comparative functions formally defined in the preceding section, which do not seem to be denoted in natural languages: these are co-intersective functions.

Some superlative constructions also give rise to generalised comparative determiners. It is enough for our purpose to consider that superlatives correspond to specific conjunctions of comparatives: roughly *the oldest man* is the man who is older than m_1, and older than m_2... and older than m_n. So roughly speaking, to get a superlative we form a conjunction in which conjuncts are the complements of *then*. We will indicate implicitly such a conjunction by the expression *than anybody else* or *anything else*. Consider now (33a), its semantics in (33b) and its superlative counterpart in (34):

(33) a. More teachers danced than sang.
b. $MORE(T)(D)THAN(S) = |T \cap D| > |T \cap S|$

(34) More teachers danced than did anything else.

By applying the above idea of conjunction of comparatives we get (35):

(35) $MOST(T)(D) = 1$ iff $\forall X(X \cap D = \emptyset) \rightarrow |T \cap D| > |T \cap X|$

The equivalences in (36) and (37) are easy to be proved. From them and (35) follow the two equivalences in (38):

(36) For all sets X and Y, $|X \cap Y| > |X' \cap Y|$ iff $\forall Z(X \cap Z = \emptyset) \rightarrow |X \cap Y| > |Z \cap Y|$

(37) For any set X, Y, $|X| = |X \cap Y| + |X \cap Y'|$

(38) $MOST(T)(D) = 1$ iff $|T \cap D| > |T \cap D'|$ iff $2 \times |T \cap D| > |T|$

Thus the superlative associated with the comparative construction in which a binary determiner occurs is the "classical" determiner *most (of)* denoting $MOST$. It is not a genuine generalised determiner. However, a "generalised superlative determiner" can be associated with the comparative construction in (39). Its superlative counterpart is given in (40a) and its denotation is given in (40b):

(39) Leo read more books than Lea.

(40) a. Leo read more books than anybody else.
b. $NSUP(X,R) = \{x : \forall y(y \neq x \rightarrow |xR \cap X| > |yR \cap X|\}$

The function $NSUP$ is denoted by the following generalised determiners, supposedly equivalent, *the most, more than anybody else* or *the greatest number*

of. Interestingly, this function is not anaphoric: using proposition 2, one shows that it does not satisfy **D1AC**. Informally, suppose that Leo studies precisely those languages that he knows. It does not follow from this that *Leo studies the greatest number of languages* is equivalent to *Leo knows the greatest number of languages*.

To conclude this section I give an example of a non-numerical comparative and superlative constructions which contain a generalised determiner. Consider (41) which contains the determiner *an older... than Lea*: Sentence (41) has two readings: absolute, in (42a) and relative, in (42b):

(41) Leo hugged an older woman than Lea.

(42) a. Leo hugged a woman older than Lea.
b. Leo hugged a woman that was older than a woman hugged by Lea.

When the determiner is used in a DP in subject position, as in (43), the relative reading disappears:

(43) An older woman than Lea was dancing.

Any gradable adjective introduces a (total) relation; in the above case we have the relation O corresponding to *be older than*. Consequently in the semantics of (41) two type $\langle 1, 2:1\rangle$ functions are involved:

(44) a. $F_{O,l}(X,R) = \{x : \exists y(y \in xR \cap X \wedge \langle y, l\rangle \in O\}$
b. $F_{O,l}(X,R) = \{x : \exists y, z(y \in xR \cap X \wedge z \in lR \cap X \wedge \langle y, z\rangle \in O\}$

Only the function in (44b) is a genuine comparative type $\langle 1, 2:1\rangle$ function.

There may be some problems with the above example since it involves some unicity conditions. The absolute and relative readings are better seen in superlative constructions (the classical paper concerning this problem is Szabolcsi 1986: (45) is ambigous with the two meanings given respectively in (46a), absolute reading, and (46b), relative reading:

(45) Leo hugged the oldest woman

(46) a. Leo hugged a woman who was older than any other woman
b. Leo hugged a woman who was older than any other woman hugged by anybody else.

Clearly the different readings of the superlatives in (46a) and (46b) are related to the different comparatives from which they originate. Thus an explanation of these different readings is easy to conceive along the lines here proposed. In particular one observes that when the superlative occurs in subject position it can have only the absolute reading as in (47):

(47) The oldest woman lives in Japan.

The above observation suggests that superlatives with relative readings are not related to an accusative extension of (the denotation of) any noun phrase, and

thus in particular the superlative itself cannot be considered as an "ordinary" noun phrase denoting a type $\langle 1 \rangle$ quantifier. This is indeed the case since it does not satisfy the **AE**. We show this informally, just using English examples, by showing that the corresponding "comparative-anaphoric" form of superlative, the one given in (46b) does not satisfy **AE**. Suppose that (48) holds:

(48) The persons that Leo hugged are the same as those that Bill kissed.

In (48) we have an instance of the conditional part of the **AE** condition. One observes now that, given (48), the sentence in (49a) needs not hold the same truth value as the one in (49b):

(49) a. Leo hugged a woman who is older than any other woman hugged by anybody else
b. Bill kissed a woman who is older than any other woman kissed by anybody else.

On the other hand the function interpreting the absolute reading of the superlative does satisfy the **AE** condition : from (48) follows the identity of truth values between (50a) and (50b):

(50) a. Leo hugged a woman who is older than any other woman.
b. Bill kissed a woman who is older than any other woman.

The generalised determiner we have in (45) corresponds to the expression *the oldest* when it forms the superlative DP which occurs in the object position gives rise to the relative reading. It denotes the function in (51):

(51) $F_{O,a}(X, R) = \{x : \exists y(y \in xR \cap X \wedge \forall z(z \neq y \wedge z \in aR \cap X) \rightarrow \langle y, z \rangle O\}$

In the next section I discuss some binary comparative determiners.

4 Some Binary Generalised Determiners

Recall that natural languages have binary or even n-ary determiners, that is expressions which take two or n common nouns to form a DP. As Keenan and Moss 1985 noted, n-ary determiners can be easily obtained by the conjunctions of common nouns in the "syntactic" scope of an unary determiner:

(52) Most students, teachers and priests were sleeping.

This sentence probably means that most students and most teachers and most priests were sleeping and not that most individuals which are students, teachers and priests ("at the same time") were sleeping. Under this reading the determiner *most..and...and...* is a ternary determiner.

Deterniners taking many common nouns as arguments as illustrated by the above example have an obvious property: their denotations are Booleanly reducibly to a conjunction of denotations of unary determiners (see Keenan and Moss 1985). It has been observed, however that natural languages have also

binary determiners whose denotations are not reducible in that sense (Keenan and Moss 1985, Beghelli 1994). For instance, the quantifier denoted by the binary determiner in (11) above is not Booleanly reducible (cf. Beghelli 1994).

Of course binary determiners occurring in DPs in object position can be considered as generalised binary determiners: if D_2 is a type $\langle\langle 1,1\rangle 1\rangle$ quantifier then the function $F((X_1, X_2, R) = D_2(X_1, X_2)_{acc}(R)$ is a well-defined type $\langle 1,1,2:1\rangle$ function. In what follows I indicate, however, that natural languages have genuine generalised binary, or even n-ary, comparative determiners. Though the notion of Boolean reducibility of n-ary determiners will not be made more precise, it will be intuitively clear that such determiners can be either Boolean reducible or Boolean irreducible. Moreover, functions denoted by these determiners have similar properties to the functions denoted by unary determiners: they are "at least" conservative and satisfy the condition of argument invariance.

We need first to define various properties of functions denoted by binary determiners, similar to those which have functions denoted by unary generalised determiners. Such properties are well-defined for "ordinary" binary or n-ary determiners (cf. Keenan and Moss 1985, Beghelli 1994, Zuber 2005, Zuber 2009). I give here some such definitions for type $\langle 1,1,2:1\rangle$ functions. For conservativity we have the definition 11 and the proposition 5 (Zuber 2010a):

Definition 11. *A type* $\langle 1,1,2:1\rangle$ *function* F *is* conservative *iff for any* $X_1, X_2 \subseteq E$ *and any binary relations* R_1 *and* R_2, *if* $E \times X_1 \cap R_1 = E \times X_1 \cap R_2$ *and* $E \times X_2 \cap R_1 = E \times X_2 \cap R_2$ *then* $F(X_1, X_2, R_1) = F(X_1, X_2, R_2)$.

Proposition 5. *A type* $\langle 1,1,2:1\rangle$ *function* F *is conservative iff for any* $X_1, X_2 \subseteq E$ *and binary relation* R *one has* $F(X_1, X_2, R) = F(X_1, X_2, (E\times(X_1\cup X_2))\cap R)$.

Since many of the determiners we will present denote cardinal functions, here is the corresponding definition:

Definition 12. *A type* $\langle 1,1,2:1\rangle$ *function is* cardinal *iff* $F(X_1, Y_1, R_1) = F(X_2Y_2, R_2)$ *whenever* $\forall y(|X_1 \cap yR_1| = |X_2 \cap yR_2|)$ *and* $\forall x(|Y_1 \cap xR_1| = |Y_2 \cap xR_2|)$

Finally, the condition of argument invariance for type $\langle 1,1,2:1\rangle$ functions is formulated as follows:

Definition 13 (D2AI). *A function* F *of type* $\langle 1,1,2:1\rangle$ *satisfies* argument invariance condition for binary determiners ***(D2AI)*** *iff for any* $a, b \in E$, $X, Y \subseteq E$ *and* R *a binary relation, if* $a((E\times X)\cap R) = b((E\times X)\cap R)$ *and* $a((E\times Y)\cap R) = b((E \times Y) \cap R)$ *then* $a \in F(X, Y, R)$ *iff* $b \in F(X, Y, R)$.

Let us see now some examples of binary generalised determiners which denote conservative argument invariant functions. Consider sentence (53): one of its readings is given in (54a). In (53) we have a binary generalised determiner which denotes an instance of the type $\langle 1, 12:1\rangle$ function given in (54b):

(53) Leo read more books and articles than Lea.

(54) a. Leo read more books than Lea and more articles than Lea.

b. $MORE_{i,j}(X, Y, R) = \{y : |yR \cap X > |iR \cap X| \wedge |yR \cap Y > |jR \cap Y|\}$

It is easy to show that function $MORE_{i,j}$ is cardinal and thus conservative. It is also argument invariant. The fact that (54a) is equivalent to (54b) and that we have a conjunction in (53) shows that this function is Booleanly reducible. Furthermore, since the number of CNs which can occur as conjuncts in (54a) is not limited, example (54a) shows how to construct n-ary comparative determiners.

A binary generalised determiner denoting the non-reducible type $\langle 1,1,2:1\rangle$ function is given in (55a); the corresponding function is given in (55b) :

(55) a. Leo read more books than Lea articles.
b. $MORE_{2,i}(X,Y,R) = \{y : |yR \cap X| > |iR \cap Y|\}$

Using the method similar to the one used in connection with the example (24) above one shows that the function in (55b) is cardinal and argument invariant.

We get similar examples be replacing in the above examples *more* by *less* or *the same number of*. Similarly we can make some Boolean compounds as in (56), where we also have a generalised binary determiner denoting a cardinal function:

(56) Leo read more books than Lea articles but less than 17 altogether.

Consider finally (57a) and the function in (57b) which is denoted by the generalised determiner *three times more... than Lea...*:

(57) a. Leo read at least three times more books than Lea articles
b. $MORE_{3\times,i}(X,Y,R) = \{y : |yR \cap X| \geq 3 \times |iR \cap Y|\}$

The type $\langle 1,1,2:1\rangle$ function in (57b) is cardinal.

5 Conclusive Remarks

By analogy with anaphoric determiners I distinguished a subclass of generalised determiners called *comparative determiners*. A generalised determiner is an expression which when applied to one or more common nouns forms a generalised DP, that is an expression which can serve as argument of a verb phrase. A genuine generalised DP is an expression which cannot serve as the grammatical subject of a sentence. Anaphoric determiners (studied in Zuber 2010a, 2010b) are genuine generalised determiners because when applied to a common noun they form nominal anaphors which are verbal arguments which cannot occur in subject position. Since I was basically interested in the logical properties of comparative determiners, no attempt has been made to justify their category syntactically. From the logical point of view it appears that their denotations have many striking similarities with denotations of "ordinary" determiners: for instance, they are conservative in a naturally generalised sense. Furthermore, as with ordinary determiners, there are not only unary generalised determiners but also n-ary ones.

Semantically, comparative DPs, in the same way as nominal anaphor, cannot be interpreted by functions corresponding to generalised (type $\langle 1\rangle$) quantifiers denoted by ordinary NPs. Formally this amounts to saying that denotations

comparative DPs do not satisfy the specific invariant condition for type $\langle 1 \rangle$ quantifiers given in called accusative extension condition. They satisfy, however the strictly weaker condition given of argument invariance. Consequently one can say that comparative (generalised) DPs essentially augment the expressive power of, say, English, since the expressive power of English would be less than it is if the only noun phrases we need were ones interpretable as subjects of main clause intransitive verbs. The reason is that such DPs must be interpreted by functions from relations to sets which lie outside the class of generalised quantifiers as classically defined, that is type $\langle 2:1 \rangle$ functions which are not extensions of type $\langle 1 \rangle$ quantifiers.

It might be interesting to notice the analogy with nominal anaphors. Keenan (1987, 1988, 2007) shows that something similar is true because of the existence of nominal anaphors. More specifically, anaphors like *himself, herself* (considered as the second nominal argument of transitive verbs) also must be interpreted by functions which do not satisfy the **AE**, and thus the generalised type $\langle 1 \rangle$ quantifiers are not enough for their interpretation. Anaphoric functions interpreting nominal anaphors satisfy another weakening of the **AE**, the condition **AC**.

References

[1994]Beghelli, F.: Structured Quantifiers. In: Kanazawa, M., Piñon, C. (eds.) Dynamics, Polarity, and Quantification, pp. 119–145. CSLI Publications (1994)

[1987]Keenan, E.L.: Semantic Case Theory. In: Groenendijk, J., Stokhof, M. (eds.) Sixth Amsterdam Colloquium (1987)

[1988]Keenan, E.L.: On Semantics and the Binding Theory. In: Hawkins, J. (ed.) Explaining Language Universals, pp. 105–144. Blackwell, Malden (1988)

[2007]Keenan, E.L.: On the denotations of anaphors. Research on Language and Computation 5(1), 5–17 (2007)

[1985]Keenan, E.L., Moss, L.: Generalized quantifiers and the expressive power of natural language. In: van Benthem, J., ter Meulen, A. (eds.) Generalized Quantifiers, Foris, Dordrecht, pp. 73–124 (1985)

[1997]Keenan, E.L., Westerståhl, D.: Generalized Quantifiers in Linguistics and Logic. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 837–893. Elsevier, Amsterdam (1997)

[1986]Szabolcsi, A.: Comparative supelatives. In: MIT Working Papers in Linguistics, pp. 245–266 (1986)

[2005]Zuber, R.: More Algebras for Determiners. In: Blache, P., Stabler, E. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 347–362. Springer, Heidelberg (2005)

[2009]Zuber, R.: A semantic constraint on binary determiners. Linguistics and Philosophy 32, 95–114 (2009)

[2010a]Zuber, R.: Generalising Conservativity. In: Dawar, A., de Queiroz, R. (eds.) WoLLIC 2010. LNCS, vol. 6188, pp. 247–258. Springer, Heidelberg (2010a)

[2010b]Zuber, R.: Semantics of Slavic anaphoric possessive determiners. In: Proceedings of SALT 19 (2010b) forthcoming

[2010c]Zuber, R.: Semantic constraints on anaphoric determiners. Research on Language and Computation (2010c) forthcoming

Author Index