LNCS 7351

Denis Béchet
Alexander Dikovsky (Eds.)

Logical Aspects of Computational Linguistics

7th International Conference, LACL 2012
Nantes, France, July 2012
Proceedings

Springer

Lecture Notes in Computer Science 7351

FoLLI Publications on Logic, Language and Information

Subline of Lectures Notes in Computer Science

Denis Béchet   Alexander Dikovsky (Eds.)

Logical Aspects of Computational Linguistics

7th International Conference, LACL 2012
Nantes, France, July 2-4, 2012
Proceedings

Springer

Volume Editors

Denis Béchet
Alexander Dikovsky
Université de Nantes, LINA UMR 6241
2, rue de la Houssinière, BP 92208, 44322 Nantes Cedex 3, France
E-mail: {denis.bechet; alexandre.dikovsky}@univ-nantes.fr

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-31261-8 e-ISBN 978-3-642-31262-5
DOI 10.1007/978-3-642-31262-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012939839

CR Subject Classification (1998): I.2, F.4.1, F.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

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

Welcome to the proceedings of the 7th International Conference on Logical Aspects of Computational Linguistics, which was held July 2–4, 2012, in Nantes, France. The aim of LACL conferences is to bring together active researchers interested in all aspects concerning the use of logics in computational linguistics to discuss current research, new results, problems, and applications of both theoretical and practical nature.
LACL 2012 focused on its traditional topics:

- Type logical grammars (Lambek Grammars, Abstract Categorial Grammars, Combinatorial Categorial Grammars, Categorial Dependency Grammars) and other formal grammars closely related to them (Minimalist Grammars, extended TAG and other weakly context-sensitive grammars)
- Formal semantics of natural language (type and proof theoretical semantics, intensional model theoretic semantics, dynamic semantics, lexical semantics)
- Logical models of discourse and dialogue (game theoretic models, ludics)

Amongst 24 thoroughly triply refereed submitted papers, the Program Committee consisting of 33 colleagues listed here selected 15 high-quality contributions by authors from China, France, Germany, Italy, Japan, Poland, Russia, UK and USA. LACL 2012 included two invited talks: by M. Steedman (University of Edinburg) and A. Clark (Royal Holloway University of London) and a tutorial by C. Fouqueré and M. Quatrini (University Paris 13 and University of Aix Marseille II). Besides this, the technical program also included a System Demonstration session.

We would like to thank all authors who submitted papers, the four invited speakers and all conference participants. We are grateful to the members of the Program Committee and of the Demo Session Committee for their thorough efforts in reviewing and discussing submitted contributions with expertise and carefulness. We are also grateful to our institutional sponsors and supporters: the Association for Logic, Language and Information (FoLLI), CNRS, Ecole des Mines of Nantes, Laboratory of Informatics of Nantes (LINA), University of Nantes and its Faculty of Sciences and Technologies. We would also like to express our gratitude to the Organizing Committee and all the people who made this meeting possible.

April 2012

Denis Béchet
Alexander Dikovsky

Organization

Program Committee

Michele Abrusci	Università di Roma Tre, Italy
Nicholas Asher	IRIT/CNRS, France
Raffaella Bernardi	University of Trento, Italy
Philippe Blache	LPL/CNRS, Aix-en-Provence, France
Wojciech Buszkowski	Poznan University, Poland
Denis Béchet	LINA - University of Nantes, France
Phillipe De Groote	LORIA/INRIA, Nancy Grand Est, France
Michael Dekhtyar	University of Tver, Russia
Alexander Dikovsky	LINA CNRS UMR 6241, Université de Nantes, France
Markus Egg	Humboldt Universität Berlin, Germany
Annie Foret	University of Rennes 1, France
Nissim Francez	Technion, Haifa, Israel
Makoto Kanazawa	NII, Japan
Gregory M. Kobele	University of Chicago, USA
Marcus Kracht	University of Bielefeld, Germany
Alain Lecomte	Université Paris 8, France
Hans Leiss	Universität München, Germany
Michael Moortgat	Universiteit Utrecht, UiL OTS, The Netherlands
Richard Moot	LaBRI/CNRS, France
Glyn Morrill	Universitat Politècnica de Catalunya, Barcelona, Spain
Reinhard Muskens	Tilburg University, The Netherlands
Uwe Mönnich	Universität Tübingen, Germany
Gerald Penn	University of Toronto, Canada
Mati Pentus	Moscow State University, Russia
Sylvain Pogodalla	LORIA/INRIA, Nancy Grand Est, France
Carl Pollard	The Ohio State University, USA
Anne Preller	LIRMM/CNRS, France
Christian Retoré	LaBRI/CNRS, France
Sylvain Salvati	INRIA, Bordeaux Sud-Ouest, France
Chung-Chieh Shan	Rutgers, The State University of New Jersey, USA
Edward Stabler	University of California, Los Angeles, USA
Mark Steedman	University of Edinburgh, UK
Isabelle Tellier	Université Paris 3, France

Demo Session Committee

Denis Béchet	LINA - University of Nantes
Florian Boudin	LINA - University of Nantes
Alexander Dikovsky	LINA - University of Nantes
Nicolas Hernandez	LINA - University of Nantes

Organizing Committee

Ramadan Alfared	LINA - University of Nantes
Denis Béchet	LINA - University of Nantes
Florian Boudin	LINA - University of Nantes
Alexander Dikovsky	LINA - University of Nantes
Anna Even	LINA - University of Nantes
Nicolas Hernandez	LINA - University of Nantes
Ophélie Lacroix	LINA - University of Nantes
Annie Lardenois	LINA
Anne-Françoise Quin	LINA - CNRS

Table of Contents

Logical Grammars, Logical Theories

Alexander Clark

Department of Computer Science
Royal Holloway, University of London
Egham, TW20 0EX
United Kingdom
alexc@cs.rhul.ac.uk

Abstract. Residuated lattices form one of the theoretical backbones of the Lambek Calculus as the standard free models. They also appear in grammatical inference as the syntactic concept lattice, an algebraic structure canonically defined for every language L based on the lattice of all distributionally definable subsets of strings. Recent results show that it is possible to build representations, such as context-free grammars, based on these lattices, and that these representations will be efficiently learnable using distributional learning. In this paper we discuss the use of these syntactic concept lattices as models of Lambek grammars, and use the tools of algebraic logic to try to link the proof theoretic ideas of the Lambek calculus with the more algebraic approach taken in grammatical inference. We can then reconceive grammars of various types as equational theories of the syntactic concept lattice of the language. We then extend this naturally from models based on concatenation of strings, to ones based on concatenations of discontinuous strings, which takes us from context-free formalisms to mildly context sensitive formalisms (multiple context-free grammars) and Morrill's displacement calculus.

1 Introduction

Logic is concerned with proof; in the logical grammar tradition the role of proof is central and well understood [1]. But logic is also concerned with truth – and what does it mean for a grammar, such as a Lambek grammar, to be true? What can it be true of? This is a problem not of proof theory but of model theory, and while there has been a great deal of work on models for the Lambek calculus [2], these are models for the calculus, not for the grammars. These models are free – the only statements that are true in these models are things that are true of all languages – but this seems to be inappropriate – there are things that are true of some languages and not of others; true about English but not about French or Dyirbal.

A Lambek grammar consists only of a finite set of type assignments: (w, T) where w is a nonempty sequence, typically of length 1, and T is a type. Therefore the question of the truth of the grammar reduces to the question of the truth of the type assignments. The metatheory of Lambek grammar is explained in [3] and [4]. Several things are clear: first the types in the grammar are intended to

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 1–20, 2012.

refer to sets of strings in a given language, and the composite types $A \bullet B$, C/B, $A\backslash C$ and the associated calculus are motivated and justified by the algebraic properties of the associated operations on these sets. The following quotes from [4] lay out the metatheoretical assumptions. First

> Sets of strings of English words will be called *(syntactic) types.*

Secondly, that the operations are 'free':

> When applying this notation to a natural language . . . we are thinking of the . . . system . . . freely generated by the words of the language under concatenation.

Finally, that the starting point for constructing a grammar is to model an existing language. Lambek does not conceive of this as a learning problem, but as a modeling problem in the structuralist tradition. Rather than defining a grammar G *in vacuo* and then showing how that grammar defines a language L — the generative stance — he starts with a language L_*, and constructs a grammar $G(L_*)$, — the descriptive approach — using a non-algorithmic method:

> One can continue playing this game until every English word has been assigned a finite number of types.

One hopes that the grammar defined in this way will define a language which is equal to the original language L_*, a condition Lambek states as follows:

> One may consider the categorial grammar to be *adequate* provided it assigns type S to w if and only if the latter is a well-formed declarative sentence according to some other standard.

Lambek's specific proposals within this metatheoretical program include what are now called Lambek grammars, and later, pregroup grammars. Others have studied the relationship between types and sets of strings; specifically Buszkowski in [5], and the possibilities of developing the informal process of construction of a grammar into an algorithmically well-defined learning procedure [6,7].

In this paper we will take a different tack, motivated again by a focus on learnability. We are interested in identifying formalisms that are not only descriptively adequate but that also can be learned from the data in a principled way – in Chomskyan terms, in achieving explanatory adequacy. Defining the types as sets of strings seems a reasonable first step. We can in a similar vein consider nonterminal symbols in a context-free grammar to be sets of strings: namely the set of strings that can be derived from that nonterminal. For a CFG G, with a nonterminal N we define as usual $\mathcal{L}(G, N) = \{w \mid N \overset{*}{\Rightarrow}_G w\}$. A symbol NP which represents noun phrases can be considered as the set of strings that can be noun phrases; the start symbol S corresponds to the language L itself.

In Lambek grammars and other categorial grammars, we have the same correspondence, but it is more complex. We can associate with each type, primitive or complex, a set of strings: say $v(T)$, the *value* of that type. There is a complication because the types are no longer atomic elements but have some structure. So a type might be of the form A/B: in this case we can ask, what is the relationship between $v(A/B)$ and $v(A)$ and $v(B)$? In particular what is the relationship between $v(A/B)$ and $v(A)/v(B)$? Clearly this depends on how we define the function v.

More fundamentally, if the primitive types are sets of strings, then what sets of strings are they? We now have a good understanding of how learnability affects the answer to this question, at least in the context of CFGs. Learnable representations are almost invariably based on objective or empiricist representations where the structure of the representation is based on the structure of the data: the language itself. The research program is then to identify structures of various type in the data, and then define representations that exploit algebraic properties of this structure. The classical example of this is the canonical deterministic finite automaton being based on the Myhill-Nerode congruence, which gives rise to the theory of regular grammatical inference [8,9,10]. When learning CFGs, the sets of strings corresponding to each nonterminal must correspond to distributionally definable sets. In the most basic models we define them to be congruence classes [11] or the sets of strings that can occur in a given context [12]. The syntactic concept lattice (SCL) of a language [13] is defined by looking at the relation between substrings and contexts in languages. This has now given rise to several different learnability results and formalisms [14,15,16]. The SCL is a residuated lattice. These objects are well known as models of various types of substructural logics [17], and as one of the foundations of the Lambek calculus. Thus a natural question arises as to the precise nature of the relationship between Lambek grammar and grammars of various types based on the SCL. It might be possible, for example, to construct efficient learning algorithms for some subclasses of categorial grammars — [7] showed that some interesting classes are learnable, though [18] showed that there are computational problems with this approach. Conversely, it might be possible to adapt the elegant syntax-semantics interface of categorial and type-logical grammars to these learnable models.

There is an additional connection between Lambek grammars and distributional learning. Learning is in some sense the inverse of generation: learning is an inverse problem. The slash operators $/, \backslash$ are the left and right residuals (a type of inverse) of concatenation and thus have a crucial role in the inference process; they also correspond directly to distributional properties.

More precisely, if we have a set of strings X and a language L, the set L/X which we define as $\{u | \forall v \in X, uv \in L\}$ is exactly the set of strings that has a certain distributional property. Namely, using notation which we present later, it is the set of all strings that can occur in all of the contexts $\{\Box v | v \in X\}$. This is also something noted in [3] where Lambek explicitly defines the motivating operations in distributional terms, as we shall see later.

This paper examines the relationship between the structure of this lattice, and the structure of models of Lambek grammar. Our main claim is that we can better meet the metatheoretical objectives of Lambek's program by basing the grammars we use on the SCL, and that more generally we can look at grammars as being sets of equations that describe algebraic structures associated with the language.

2 Residuated Lattices

We will start with some standard definitions. A monoid is a simple associative algebraic structure, $\langle M, \circ, 1 \rangle$ where M is a set, $\circ$ is an associative binary operation and 1 is a left and right unit. The standard example is the set of all strings over a finite alphabet Σ under concatenation; which we write Σ^*, with the empty string denoted by λ. This is the free monoid generated by Σ. In what follows we will assume we have some fixed nonempty set Σ. A (formal) language is a subset of Σ^*.

A context is just an ordered pair of strings that we write $l\Box r$. l and r refer to left and right; $\Box$ is a symbol not in Σ. [1] We can combine a context $l\Box r$ with a string u with a wrapping operation that we write $\odot$: so $l\Box r \odot u$ is defined to be lur. We will sometimes write f for a context $l\Box r$. The empty context we write $\Box$. Clearly $\Box \odot x = x$.

We will fix a formal language $L \subseteq \Sigma^*$; this then defines a relation between contexts $l\Box r$ and strings w given by $l\Box r \sim_L w$ iff $lwr \in L$. The distribution of a string w is defined as $C_L(w) = \{l\Box r | lwr \in L\}$. We can define a congruence relation based on distributional equivalence with respect to a language L. We say $u \equiv_L v$ iff $C_L(u) = C_L(v)$. This is an equivalence relation and a congruence; we write $[u]_L = \{v \mid u \equiv_L v\}$. This gives us a second relevant example of a monoid: the *syntactic monoid* which is the quotient monoid $\Sigma^*/\equiv_L$, whose elements are the congruence classes and where the natural concatenation operation, $[u]_L \circ [v]_L = [uv]_L$, is well defined as this is a congruence. This is not free: this has a nontrivial algebraic structure that depends on the language; it can be a group, it can be Abelian, it can be finite or infinite. Indeed this is finite iff the language L is regular.

The second class of structures we are interested in are lattices; which are partially ordered sets with least upper bounds (join, written $\vee$) and greatest lower bounds (meet, written $\wedge$). The classic example is the lattice of all subsets of a given set, where meet is set intersection and join is set union. Given the meet operation we can define the partial order as $X \leq Y$ iff $X = X \wedge Y$.

A residuated lattice is a combination of these two structures — a monoid and a lattice — where the two structures must interact 'nicely'. Formally we say that it is a tuple $\langle M, \circ, 1, \wedge, \vee, /, \backslash \rangle$, where M is a set, $\circ, \wedge, \vee, /, \backslash$ are binary operations and 1 is a constant, such that $\langle M, \circ, 1 \rangle$ is a monoid, and $\langle M, \wedge, \vee \rangle$ is a lattice.

[1] In previous work we have written a context as an ordered pair (l, r). This causes confusion when we consider discontinuous strings, and so here we use this less ambiguous notation.

We can state the interaction requirement as the fact that the following three conditions are equivalent:

$$X \circ Y \leq Z \text{ iff } X \leq Z/Y \text{ iff } Y \leq X\backslash Z$$

This is a slightly stronger requirement than mere monotonicity. The classic example of this is the set of all languages over Σ^*, which we denote by 2^{Σ^*}, and where the lattice operations are set intersection and union, and where $\circ$ is defined to be simple concatenation $M \cdot N = \{uv|u \in M, v \in N\}$ and the residuals are defined naturally to be

$$M/N = \{u|\forall v \in N, uv \in M\}$$
$$N\backslash M = \{u|\forall v \in N, vu \in M\}$$

It is easy to verify that these definitions satisfy the axioms of residuated lattices. Note that of course $\circ$ is not commutative. This lattice is distributive, but in general these lattices are not. We will use for the rest of the paper this 'result on top' notation, for consistency with the literature in residuated lattices.

Given the definitions it is easy to see that $(X/Y) \circ Y \leq Y$. Similarly it is easy to establish the following identities, and many others, which will be very familiar to those who know the Lambek calculus.

Composition $(X/Y) \circ (Y/Z) \leq (X/Z)$ and $(Z\backslash Y) \circ (Y\backslash Z) \leq (Z\backslash X)$
Associativity $(Z\backslash X)/Y = Z\backslash(X/Y)$
Lifting $X \leq Y/(X\backslash Y)$ and $X \leq (Y/X)\backslash Y$.

However there are also identities that have no counterparts in the Lambek calculus. These are ones that use the operations $\wedge$ and $\vee$ such as for example the fact that $X \wedge Y \leq X$. These are however valid in the full Lambek calculus, augmented with appropriate inference rules for these connectives.

2.1 Syntactic Concept Lattice

Given a particular language, L, we have a particular monoid, the syntactic monoid of that language. In just the same way, there is a residuated lattice associated with the language that we call the syntactic concept lattice. This is the lattice of all distributionally definable subsets of strings in a language; we will denote this $\mathfrak{B}(L)$.

There are a number of equivalent ways of defining this; we will use a way which emphasizes the links with Galois connections and residuated maps. As noted above, we can consider the relation between strings and contexts defined by a language. Given this relation we can then define two maps from sets of strings to sets of contexts and vice-versa. These are traditionally called 'polar' maps and we will denote them both by the use of $'$. Intuitively, given a set of strings S, we can define the set of contexts S' to be the set of contexts that appear with every element of S. This is sometimes called the *distribution* of the set S in the language L.

$$S' = \{l\Box r \mid \forall w \in S \;\; lwr \in L\} \tag{1}$$

If S consists of a single string w, then S' is just the same as $C_L(w)$. If S contains two strings, then it will be the intersection of the distribution of those two strings. Clearly if $S \subseteq T$ then $S' \supseteq T'$ – as we make the set of strings larger, the intersection of their distributions will only decrease.

Dually, we can define for a set of contexts C the set of strings C' that occur with all of the elements of C

$$C' = \{w \mid \forall\, l\Box r \in C \;\; lwr \in L\} \tag{2}$$

Consider now for any set of strings S the set of strings S''. It is easy to verify that $S'' \supseteq S$, and furthermore that $(S'')'' = S''$ – in other words, this is a closure operation. We say that the set of strings S is *closed* iff $S = S''$.

Note that for any set of contexts C, C' is a closed set of strings. We can thus define a closed set of strings in two ways: either by picking a set of strings S and closing it to get S'' or by picking a set of contexts C which defines the set C'. We call the former method a *primal* method, and the latter a *dual* method.

It is easy to see that L itself is a closed set of strings. L' will contain the empty context $\Box$. As a result any string in L'' must occur in the context $\Box$ and is thus in L; therefore $L = L''$. Similarly Σ^* is always closed.

We can consider, for a language L, the set of all closed sets of strings of the language. Interestingly, this set is finite if and only if the language is regular. For example, the language $L = \Sigma^*$ only has one closed set of strings, Σ^*. The infinite regular language $L = (ab)^*$ has 7 closed sets of strings. These consist of the four sets $\emptyset, \{\lambda\}, L, \Sigma^*$ together with 3 other sets which are bL, La and bLa.

These closed sets form a lattice – the partial order is given by set inclusion and meet and join are defined as $S \wedge T = S \cap T$ and $S \vee T = (S \cup T)''$. We define a concatenation operation $\circ$ of two closed sets of strings as $S \circ T = (ST)''$; ST is not necessarily a closed set, and so to make the operation well defined we take the closure. Note the difference between this definition and 'normal' set concatenation $S \cdot T$. Given the language $(ab)^*$, if we concatenate the two closed sets of strings La and bL, we get the language $LabL = (ab)^+$. This is not closed: $((ab)^+)'' = (ab)^* = L$. So $La \circ bL = L$ which is not equal to $LabL$.

While the concatenation is different from concatenation in the lattice 2^{Σ^*}, there are two residual operations which are identical. Thus if S and T are closed then S/T and $S\backslash T$ are also closed. We can verify that the closed sets of strings with respect to a language L form a residuated lattice, which we denote $\mathfrak{B}(L)$. See [13] for further details [2]. Note that the natural map from $2^{\Sigma^*} \to \mathfrak{B}(L)$, given by $h_L(S) = S''$ is a homomorphism of $\circ, \vee$ and 1.

[2] In that paper we present the lattice as a collection of ordered pairs of sets of strings and sets of contexts. In this paper we will consider the mathematically equivalent formulation just as sets of strings.

3 Types Should Be Closed Sets of Strings

Let us now start to consider the relationship between the SCL, $\mathfrak{B}(L)$, and the system of types that is used by Lambek grammars. This can be framed in a number of different ways, mathematically, conceptually and empirically.

The empirical question is whether, in natural languages, the types correspond to closed sets of strings. Lambek clearly thought so: indeed we find in [3] the following statement:

> We shall assign type n to all expressions which can occur in any context in which all proper names can occur.

Let us look at this statement carefully. Suppose we have a language L and a set of strings N that is the set of proper names. The set of contexts in which all proper names can occur is just the set N' using the polar maps defined earlier. The set of expressions which can occur in any of N' is just N''. So the type n corresponds exactly to the closed set of strings N''; since the start symbol S corresponds to the language L, then all product free types will correspond to closed sets of strings.

The second way of looking at this is as a mathematical claim: is it the case that for all Lambek grammars, the types correspond to closed sets of strings? Here we can give a clear negative answer: it is easy to construct Lambek grammars where the types correspond to quite arbitrary sets of strings. For example consider the language $L = \{a^n b^n c \mid n > 0\}$. The closure of $\{ab\}$ is $\{a^n b^n \mid n > 0\}$, but one could write a grammar where there is a type which generates any finite or infinite context-free subset of that closed set.

Conceptually, we can make additional arguments in favour of this representational assumption. Given that we have exactly this structure why would we impose another structure on it? Using a Lambek grammar implicitly imposes a residuated lattice structure on the language [5]. On grounds of general parsimony, in the absence of a convincing reason to use a different structure, we should use a grammar which follows the existing objectively valid empirically based structure, rather than using some other more arbitrary one: a convincing reason would be if the restriction affected the weak or strong generative capacity of the formalism. We now consider this possibility.

3.1 Weak Generative Power

Lambek grammars can represent any context-free language. However if we add the restriction that the types must correspond to closed sets of strings, then it is natural to ask whether we lose any weak generative power. The somewhat surprising answer is that we do not.

Suppose we have a context-free language defined by a CFG, G, which we assume is in Greibach normal form. For each nonterminal N we define a set of strings which is the closure of the set of strings in the yield of N: $\{w | N \overset{*}{\Rightarrow}_G w\}''$. And for each terminal symbol a we define a type A which has value $v(A) = \{a\}''$.

For an element of $(V \cup \Sigma)^+$ we extend the type function v using $\circ$, in the obvious way: $v(a\beta) = v(a) \circ v(\beta)$ and $v(N\beta) = v(N) \circ v(\beta)$. We can then prove that if $N \rightarrow \alpha$ is a production then $v(N) \supseteq v(\alpha)$ [15]. We can therefore construct a categorial grammar in the standard way since if the rule is $N \rightarrow aM_1 \ldots M_k$ then $a \in v(N/M_k/\ldots/M_1)$, and so we can assign the type $N/M_k/\ldots/M_1$ to a. Any derivation of a string with respect to the CFG can be turned into a valid derivation with respect to the Lambek grammar, and, conversely, any string generated by the Lambek grammar will be in the language by the soundness of the Lambek calculus.

3.2 Strong Generative Capacity

In contrast, from the point of view of strong generative capacity, that is to say in terms of the sets of structural descriptions that our grammar assigns to strings, we do give up some modeling power. In particular we lose the ability to describe distributionally identical strings in different ways. More precisely, we can propose the following principle: If for two letters $a, b \in \Sigma$, the distribution of a is a subset of the distribution of b, then for any strings l, r the set of structural descriptions assigned to lar must be a subset of the set of structural descriptions assigned to lbr, modulo some relabeling to reflect the difference between a and b.

If, for example, two letters are distributionally identical, then in this model, they *must* be treated identically. In a normal Lambek grammar, or context-free grammar, we are free to treat them as being completely different. Consider the following example: a language L which has two different grammars which are weakly equivalent but assign different structural descriptions to the strings. To be concrete, suppose we have $(ab)^+$ which admits the two CFGs, G_1 which has productions $S_1 \rightarrow ab$, $S_1 \rightarrow abS$ and G_2 which has the productions $S_2 \rightarrow ab$ $S_2 \rightarrow aTb$, $T \rightarrow ba$, $T \rightarrow bS_2a$. Consider the language $(c|d)(ab)^+$. We could have a grammar then with two rules $S \rightarrow cS_1$ and $S \rightarrow dS_2$. This will assign different trees to strings starting with c and strings starting with d even though c and d are in this case completely distributionally identical. This sort of situation is ruled out by this model – if natural languages had this sort of behaviour then the type of distributionally motivated model that we propose here would be clearly incorrect. If on the other hand there is this type of connection between the distributional properties and the structural properties of the language, then this would indicate that we are on the right track.

3.3 Consequences

Let us now proceed on the assumption that we want the types in our grammar to correspond to closed sets of strings. We fix a target language L_*. We start by leaving the Lambek grammar formalism completely unchanged, and making the representational assumption that the types are closed sets of strings. We assume a finite set of primitive types **Pr**, which we extend to a set of product free types $\mathbf{Tp}(/, \backslash)$ using just the two connectives $/, \backslash$. We assume that we have a function v, a valuation, from these types to closed sets of strings, $v : \mathbf{Tp}(/, \backslash) \rightarrow \mathfrak{B}(L_*)$, which is defined on the primitive types and then extended recursively to the complex types using $v(R/T) = v(R)/v(T)$ and $v(R\backslash T) = v(R)\backslash v(T)$. We assume that one primitive type S is the start type such that $v(S) = L_*$.

Since the SCL is a residuated lattice, the Lambek calculus is sound. That is to say, if we have a sequent $\Gamma \rightarrow X$ that is derivable in the Lambek calculus then $v(\Gamma) \subseteq v(X)$ where we extend v to $\mathbf{Tp}^+$ using $\circ$. We then make a finite lexicon which is a finite assignment of types to elements of Σ, and we require that it satisfy the obvious compatibility condition that if a has type T then $a \in v(T)$. This is the condition that is called *correctness* in [5].

This gives us a Lambek grammar. Note that since the Lambek calculus is sound, and because of the compatibility condition the type assignment satisfies we have the following trivial lemma, whose proof we omit.

Lemma 1. *$L(G) \subseteq L_*$.*

Example 1. Suppose $L_* = \{a^n b^n | n > 0\}$. The closed sets of strings include $\{a\}, \{b\}$. We assign type A to $\{a\}$, B to $\{b\}$ and S to L. We then have the infinite set of derived types A/A, S/B etc. each of which refer to closed sets of strings. $v(A/A) = \{\lambda\}$, $v(S/B) = \{a^{i+1}b^i | i \geq 0\}$, $v((S/B)/S) = \{a\}$, and so on. Since $a \in v(S/B)$ and in $v((S/B)/S)$ we assign a the two types S/B and $(S/B)/S$ and to b we assign the type B. This gives us a Lambek grammar. Clearly $ab \in L(G)$ since $S/B, B$ is a valid type assignment and we can prove $S/B, B \rightarrow S$. Similarly $aabb \in L(G)$ through the type assignment $(S/B)/S, S/B, B, B$. Indeed $L(G) = L_*$.

This grammar is adequate in Lambek's sense – it defines the correct language. Of course, we needed to manually choose which primitive types we should use. In this case we only needed to use the 'natural' types – namely the closures of the singleton sets of elements of Σ, $A = \{a\}''$ and $B = \{b\}''$, and the special type S. These particularly simple types have a fixed valuation in a given language.

Contrast this to the approach taken by Buszkowski in [5]. There we start with a fixed grammar. For a primitive type T, $v(T)$ is defined to be the set of strings such that there is a valid type assignment so that we can derive T from that type assignment. Given this we then extend it to the set of all product free types. We can then ask whether the type assignment (a, T) satisfies $a \in v(T)$. Here we already have a semantics for the types before we start defining the grammar, and so it is easy to stipulate this correspondence as a condition.

4 Finite Representations of Closed Sets

In the preceding discussion we simply stipulated what the relevant closed sets of strings were. This glosses over an important problem – how to identify or refer to a closed set of strings. In the earlier example we started with some primitive types which referred to the sets $\{a\}''$ which are just the closed sets of strings defined by a single element of Σ, and the language itself, L, which is always a closed set. To avoid confusion, we will use a notational convention from mathematical logic and use $\dot{a}$ for the type (symbol in the metalanguage) to distinguish it from the symbol $a \in \Sigma$ in the object language. We will use only these 'natural' types as our primitive types: one type $\dot{a}$ for each element a of Σ and one special type $\dot{L}$, which refers to L, together with the extra type $\dot{\lambda}$, which denotes the closure of the empty string. In what follows we will eliminate all other primitive types, as they are in a sense arbitrary, and rely only on these more basic ones which have a well defined semantics.[3] We will also augment our set of connectives to include not just the two connectives $/, \backslash$, but also a product $\bullet$, and the two lattice operations $\vee, \wedge$. Our set of types $\mathbf{Tp}_\Sigma$ is thus the closure of these primitive types under the five binary connectives.

Given a language L, we recursively define the value of all types using a valuation v_L, a function from $\mathbf{Tp}_\Sigma \to \mathfrak{B}(L)$, which is defined in Table 1.

Table 1. Interpretation of the basic types and connectives

Primitive types	Lattice connectives	Monoid connectives
$v_L(\dot{a}) = \{a\}''$	$v_L(R \vee T) = (v_L(R) \cup v_L(T))''$	$v_L(R \bullet T) = (v_L(R) \cdot v_L(T))''$
$v_L(\dot{L}) = L$	$v_L(R \wedge T) = v_L(R) \cap v_L(T)$	$v_L(R/T) = v_L(R)/v_L(T)$
$v_L(\dot{\lambda}) = \{\lambda\}''$		$v_L(R\backslash T) = v_L(R)\backslash v_L(T)$

Note that for any term T, $v_L(T)$ is a closed set of strings, an element of $\mathfrak{B}(L)$.

We are interested in ways that we can use some simple finite expression to define, or refer to, a closed set of strings in a language. There are two different strategies we can use to define closed sets of strings. The first, which we call a *primal* approach, involves specifying a finite set of strings S; this will define the closed set S''. Alternatively, we can pick a finite set of contexts C and use this to define closed set C'. Using these two approaches we can express both the primal and dual approaches using just complex types formed from these 'natural' types. Suppose we have a single string of length n, $w = a_1 \dots a_n$. We will write $\dot{w}$ as an abbreviation for the complex type $\dot{w} = \dot{a_1} \bullet (\dot{a_2} \dots \dot{a_n})$. It is easy to see that $v(\dot{w}) = \{w\}''$, and so we can represent the set of strings $\{w\}''$ using types of the form $\dot{w}$. Similarly suppose we have a set of strings $X = \{u, v\}$. We can refer to the closed set of strings X'' using the type $\dot{X} = \dot{u} \vee \dot{v}$. Again it is easy to see that $v(\dot{X}) = X''$. Similarly for any finite set of strings X we define the associated term $\dot{X}$. There are of course many different terms, by associativity of concatenation and commutativity and associativity of union, which all denote the same closed set of strings – we assume that we fix some particular one according to some arbitrary convention.

In the dual situation we will have a finite set of contexts, C which defines a set of strings C'. Suppose $C = \{l \Box r\}$; we define $\dot{C}$ to be the complex type $(\dot{l}\backslash\dot{L})/\dot{r}$. Suppose $u \in C'$. This means that $lur \in L$. Suppose $l_2 \in \{l\}''$ and $r_2 \in \{r\}''$ then $l_2 u r_2 \in L$ so $u \in v(\dot{C})$. Indeed $v(\dot{C}) = C'$. If C has more than one element, then

[3] We do not consider here $\top$ and $\bot$ as they seem not to be necessary for syntactic description.

we will define a term using $\wedge$: If $C = \{l_1 \Box r_1, \ldots, l_k \Box r_k\}$, we define $\dot{C} = ((\dot{l_1} \backslash \dot{L})/\dot{r_1}) \wedge \ldots ((\dot{l_k} \backslash \dot{L})/\dot{r_k})$. Again it is straightforward to verify that $v(\dot{C}) = C'$.

Therefore in order to define closed sets of strings using either the primal or dual methods, we need only use the basic natural types that refer to the individual elements of Σ and the language itself. There is no need to use any additional arbitrary primitive types. This comes at a price – we cannot refer to all closed sets of strings in all languages using only finite terms of this form. This restricts the class of languages we can define in this way in a nontrivial way. Contrast this case with that of congruential languages [11], where the restriction to congruence classes causes some limitation in descriptive power, but it is easy to refer to each congruence class (simply by referring to any string in the class). Here, we lose no descriptive power, but cannot in some cases refer to the relevant classes of strings using only a finite description.

5 Grammars as Sets of Equations

Having eliminated all of the arbitrary symbols, we are left only with symbols that have a clear denotation in our language. We can therefore define statements using these symbols to say things about the language we are modeling that may be either true or false.

We have a set of types or terms, using the primitive symbols and the binary operations, that we call $\mathbf{Tp}_\Sigma$. Each term refers to a particular closed set of strings in a language using the valuation v_L.

We then add one relation symbol, equality, $\doteq$, again using the dot convention to distinguish this as a symbol in the metalanguage. We will consider equations of the form $R \doteq T$ for $R, T \in \mathbf{Tp}_\Sigma$.

Definition 1. *An equation $R \doteq T$ is true in the language L, which we write $L \models R \doteq T$, iff $v_L(R) = v_L(T)$*

Note that we can state inequalities using only the symbol $\doteq$ since $(T \wedge R) \doteq R$ is true iff $v_L(T) \subseteq v_L(R)$. We will use $S \dot{\subseteq} T$ as a shorthand for such an equation.

Every equality or inequality of this form is either true or false of a particular language. The statement $w \in L$ becomes the equation $\dot{w} \dot{\subseteq} \dot{L}$, in the sense that $L \models \dot{w} \dot{\subseteq} \dot{L}$ iff $w \in L$. The statement $u \equiv_L v$ becomes $\dot{u} \doteq \dot{v}$. Clearly there are variety of statements that can be framed in this way, from ones which, for CFGs are polynomially decidable such as the former, to those like the latter, which are undecidable for CFGs. The lexicon in a Lambek grammar consists of type assignments (a, T) where $a \in \Sigma$ and $T \in \mathbf{Tp}_\Sigma$ which in this formalism can be written as equations of the form $\dot{a} \dot{\subseteq} T$. We do not restrict ourselves only to this sort of equation but consider for the moment any kind of equation.

We can therefore view a grammar as a set of equations that are true of the language, or more strictly true of the syntactic concept lattice of the language. More formally, let $\mathcal{E}$ be a finite set of equation; we say that $L \models \mathcal{E}$ iff $L \models E$ for all $E \in \mathcal{E}$. If this is the case then we say that L is a model for $\mathcal{E}$. Note that there will always be at least one model: the language $L = \Sigma^*$ has a trivial SCL with only one element, and so it satisfies all sets of equations.

Example 2. Consider the grammar of Example 1. We have two nontrivial assignments of the word a: $\dot{L}/B$ and $(\dot{L}/B)/\dot{L}$. These two assignments correspond to the inequalities $\dot{a}\dot{\subseteq}\dot{L}/\dot{b}$ and $\dot{a}\dot{\subseteq}\dot{L}/\dot{b}/\dot{L}$. These correspond to the statements $\{a\}''\{b\}'' \subseteq L$ and $\{a\}''L\{b\}'' \subseteq L$, both of which are true for the language in question.

We can define a semantic notion of entailment: If every language L that satisfies all of the equations in $\mathcal{E}$ also satisfies another equation E then $\mathcal{E} \models E$. In other words, all models of $\mathcal{E}$ are also models of $\mathcal{E} \cup \{E\}$. We could define a language on the basis of this semantic entailment: $\{w \in \Sigma^* \mid \mathcal{E} \models \dot{w}\,\dot{\subseteq}\,\dot{L}\}$, but it is more convenient, we think, to use a syntactic notion of entailment using a syntactic calculus. We can code the axioms of the residuated lattices using equations and functions of various arities. We code the primitive types $\dot{a}, \dot{\lambda}, \dot{L}$ as constant symbols, and have a countable set of variables $x, y, z, \ldots$. For example associativity of the monoid is defined as: $x \bullet (y \bullet z) \doteq (x \bullet y) \bullet z$, using implicit universal quantification as in equational logic. We assume a finite equational basis for residuated lattices that we write **RL** and we write $\mathcal{E} \vdash E$ for the syntactic notion of entailment under Birkhoff's equational deductive system using $\mathcal{E}$ and **RL**. This gives us a slightly different definition that we will now use.

Definition 2. *A finite set of equations $\mathcal{E}$ defines a language over Σ, $\mathcal{L}(\mathcal{E}) = \{w \in \Sigma^* \mid \mathcal{E} \vdash \dot{w}\,\dot{\subseteq}\,\dot{L}\}$*

This does not entirely coincide with the definition of semantic entailment, in spite of Birkhoff's completeness theorem. This is because the notion of semantic entailment restricts the class of models to lattices that are the SCLs of languages, whereas the equations that we use to define residuated lattices allow all residuated lattices.

Many different formalisms can be cast in this light: they all have weak generative power properly included in the class of conjunctive grammars [19]; a context sensitive formalism that properly includes the CFGs. The use of terms with $\wedge$ takes us out of the class of CFLs [20].

An advantage of framing things in this way is that grammar construction may become a computationally tractable problem, because it can be decomposed into a collection of smaller problems. Each equation is either true or false with respect to the language being modeled. If a grammar overgenerates then this is because at least one equation in the grammar is false. The validity of each equation can be evaluated and tested independently of all others. This means that the learning algorithm only has to deal with a set of independent, local problems rather than one single global problem. As a result for these formalisms we can in general devise computationally efficient learning procedures [21,22,11]. However, given the undecidability of this logic in general, we will need to restrict the sorts of equations we use in order to exploit this property efficiently.

5.1 Lambek Grammars as Equations

We will start with Lambek grammars. We assume that all of the primitive types in the grammar can be defined using equations, as in Section 4, and that the grammar is adequate. If we assign the type T to a word a in a Lambek grammar, then this is equivalent to the inequality $\dot{a}\dot{\subseteq}T$. We assume that all of these type assignments are true. Indeed if we assign to a word a the two types R, T then this is equivalent to the single type assignment, $\dot{a} \dot{\subseteq} R \wedge T$ since if $a \in v(T)$ and $a \in v(R)$ then $a \in v(R) \cap v(T) = v(R \wedge T)$[20]. We can therefore code the lexicon of a Lambek grammar as a set of equations $\mathcal{E}$, one per element of Σ. We can then show that the language defined by the Lambek grammar is exactly equal to $\mathcal{L}(\mathcal{E})$. The two sets of derivations though are not entirely the same for reasons we will explain in the context of CFGs in the next section.

5.2 Context-Free Grammars

We can also frame context free grammars as a system of equations of this type. Suppose we have some CFG G in, for simplicity, Chomsky normal form. For a nonterminal N we can define the corresponding set of strings as the closure of the set of yields $\{w|N \overset{*}{\Rightarrow}_G w\}''$. We assume that for each nonterminal in G, the corresponding closed set of strings can be expressed as a finite term, which we will write as $\dot{N}$. So $v_L(\dot{N}) = \mathcal{L}(G, N)''$. Clearly S can be represented as the term $\dot{L}$. Then we can convert each production of the form $N \to PQ$ to an equation which states that the right hand side is a subset of the left hand side: $\dot{P} \bullet \dot{Q} \dot{\subseteq} \dot{N}$. Each production of the form $N \to a$ is an equation of the form $\dot{a} \dot{\subseteq} \dot{N}$, and an epsilon-production is of the form $\dot{\lambda} \dot{\subseteq} \dot{L}$. Therefore we can define for each production in the grammar an equation, giving a set of equations $\mathcal{E}_G$ such that $S \overset{*}{\Rightarrow}_G w$ iff $\mathcal{E}_G \vdash \dot{w} \dot{\subseteq} \dot{L}$

Note that the derivation process in the grammar is not exactly the same as the proof with respect to the set of equations. For example, if a nonterminal N is represented by a dual term, say $\dot{l}\backslash\dot{L}/\dot{r}$, and we have a production $N \to a$, then from $\dot{a} \dot{\subseteq} \dot{l}\backslash\dot{L}/\dot{r}$ we can deduce directly using the residuation operations that $\dot{l} \bullet \dot{a} \bullet \dot{r} \dot{\subseteq} \dot{L}$ i.e. that lar is in the language, even if, for example N is not reachable in the grammar. This does not affect the set of strings that are generated, since the equations are all true, and thus we will not overgenerate. Similarly, when we consider nonterminals that are defined primally, N might correspond to the closure of the finite set $\{w_1, \ldots, w_k\}$. In this case we will have a direct proof of $\dot{w}_i \dot{\subseteq} \dot{N}$, which might not correspond to a CFG derivation. Thus it might be desirable to restrict the class of derivations available to exclude these ones which though legal, fail to correspond to proper CFG derivations.

In both the case of CFGs and Lambek grammars, we cannot represent all context free languages in this way, since there are languages where the closed sets that we require cannot be defined using only finite terms of the types we consider here.

5.3 Finite Automata

We can however represent all regular languages using finite automata in this manner, since residual languages are closed sets. Recall that the residual languages are defined as $u^{-1}L = \{v|uv \in L\}$, for some $u \in \Sigma^*$. Consider a deterministic finite automaton with state set Q, initial state q_0, transition function δ and set of final states $F \subseteq Q$. For each state in $q \in Q$, assuming they are all reachable, we can find a string w_q such that $\delta(q_0, w_q) = q$; we stipulate that $w_{q_0} = \lambda$.

We can therefore represent a state q by the term $\dot{w}_q\backslash\dot{L}$; under this scheme the initial state is represented by $\dot{\lambda}\backslash\dot{L}$ which will have the same value as $\dot{L}$. If there is a transition from state q to state r labelled with a then this can be represented as the equation: $\dot{w}_q\backslash\dot{L}\dot{\supseteq}\dot{a} \bullet (\dot{w}_r\backslash\dot{L})$. If a state q is in F this is represented by the equation $\dot{w}_q\dot{\subseteq}\dot{L}$ or equivalently $\dot{\lambda}\dot{\subseteq}\dot{w}_{q_f}\backslash\dot{L}$. It is easy to verify that if a word w is accepted by the automaton then, if we denote $\delta(q_0, w) = q_f$, then $\mathcal{E} \vdash \dot{L}\dot{\supseteq}\dot{w} \bullet (\dot{w}_{q_f}\backslash\dot{L})$, and therefore, given that $q_f \in F$ that $\mathcal{E} \vdash \dot{w} \dot{\subseteq} \dot{L}$ iff the automaton accepts w.

5.4 Thue Systems

We can get some more insight by considering a particularly limited form of equation. In order to say that a string is in the language, we can use an equation of the form $\dot{w} \dot{\subseteq} \dot{L}$. Clearly if we only have equations of this type, we will not have anything other than a simple finite list of elements of the language. Let us add a second type of rule: equations of the form $\dot{u} \doteq \dot{v}$. This states that $\{u\}'' = \{v\}''$ which implies that $u \equiv_{L_*} v$. A finite set of equations of this type is exactly equivalent to a Thue system: the only connective we use is $\circ$ and the only relation is equality. Recall that a Thue system T is just a finite set of pairs of strings, that we will write as $u \leftrightarrow v$. For example $\{ab \leftrightarrow \lambda\}$ is a simple Thue system. This defines a congruence of Σ^*, which we write $\equiv_T$, as the symmetric transitive closure of $lur \equiv_T lvr$ if $u \leftrightarrow v$ is in T. This is clearly a congruence which is called the Thue congruence defined by T; one of the simplest and earliest form of rewriting systems. This is essentially the same as a finite presentation of a finitely generated monoid: we have a finite set Σ of generators, and a finite set of equations or relations over Σ. The combination of these two types of rules gives us a simple way of defining languages: for example the Dyck language can be defined using the two equations $\dot{a}\dot{b} \doteq \dot{\lambda}$ and $\dot{\lambda} \dot{\subseteq} \dot{L}$.

Note that the congruence defined by the system will not in general be exactly the same as the syntactic congruence of the language defined by the system. If the system of equations defines the language L then clearly if $\mathcal{E} \vdash \dot{u} \doteq v$ then $u \equiv_L v$, but not necessarily in reverse.

Example 3. Consider $\mathcal{E}$ to consist of $\dot{a}\dot{b}\dot{a}\dot{b} \doteq \dot{a}\dot{b}$ and $\dot{a}\dot{b} \dot{\subseteq} \dot{L}$. This defines the language $(ab)^+$ which is regular. This has a syntactic monoid where $aa \equiv_L bb$. But it is not the case that $\mathcal{E} \vdash \dot{a}\dot{a} \doteq \dot{b}\dot{b}$.

5.5 Distributional Lattice Grammars

Distributional lattice grammars (DLGs) are a learnable grammatical formalism explicitly based on a model of the SCL [23]; it is therefore straightforward to convert them into a set of equations that describe it. For reasons of space we will not present them in full; one of their advantages is that they can compactly represent an exponential number of equations using only a polynomial amount of data. The DLG models a partial lattice that considers only a finite set of contexts We assume that we have a set of k contexts, $F = \{l_1\Box r_1, \ldots, l_k\Box r_k\}$, which include the empty context $\Box$. We denote this partial lattice by $\mathfrak{B}(L, F)$; which consist only of the finite collection of closed sets of strings defined by subsets of F, together with a concatenation operation defined by

$$M \circ N = ((MN)' \cap F)'$$

We can finitely represent this partial lattice using the set of contexts, a sufficiently large set of substrings K, and some amount of information about L. In particular we need to know which elements of $F \odot KK$ are in L. There is a map f^* from $\mathfrak{B}(L, F) \to \mathfrak{B}(L)$, which embeds the finite collection of closed sets defined by subsets of F into the full lattice, which satisfies

$$f^*(M \wedge N) = f^*(M) \wedge f^*(N) \tag{3}$$

$$f^*(M \circ N) \supseteq f^*(M) \circ f^*(N) \tag{4}$$

These equations therefore justify the following conversion of relations in the finite lattice into statements about the SCL. We use subsets of F to define closed sets of strings; if $C \subseteq F$ is a set of contexts, we use $\dot{C}$ for the term that we use to refer to the closed set of strings C'. We then have the following types of equations, where X, Y, Z are subsets of F.

$$\begin{array}{rl} \dot{a} \,\dot{\subseteq}\, \dot{X} & \text{if } a \in X' \\ \dot{X} \bullet \dot{Y} \,\dot{\subseteq}\, \dot{Z} & \text{if } X'Y' \subseteq Z' \\ \dot{X} \wedge \dot{Y} \,\dot{\subseteq}\, \dot{Z} & \text{if } X' \cap Y' \subseteq Z' \end{array}$$

The DLG specifies a subset of these equations $\mathcal{E}$ and there is a simple cubic time algorithm for computing for any string w, the maximal set of these contexts C such that $\mathcal{E} \vdash \dot{w} \,\dot{\subseteq}\, \dot{C}$. Language membership is then determined by the presence of the empty context $\Box$, equivalent to $\dot{L}$ in the set C.

6 Discontinuity

So far we have considered continuous individual strings and models based on simple concatenation. There are three related systems: distributional learning based on the relation between contexts and strings; Lambek calculus and Lambek grammars which we can view as the logic of these structures, and context free grammars.

Given the inadequacies of context-free formalisms in general, it is natural to extend this approach to mildly context sensitive formalisms, which are directly or indirectly based on modeling *discontinuous strings*. The formalism most direct analogous to CFGs is the class of multiple context free grammars [24] (MCFGs) and we can think of the displacement calculus [1] as its logic. Distributional learning has also made the same transition [25]. For simplicity, we will just consider the generalisation to strings with one gap (two components); the generalisation to k components is straightforward but requires a more elaborate notation.

We define a 2-word to be a pair of strings (an element of $\Sigma^* \times \Sigma^*$) that we write (v_1, v_2). We define a 2-context to be a string with 2 gaps: which we can write $u_0 \Box u_1 \Box u_2$. We can combine this with a pair of strings to get a string: $u_0 \Box u_1 \Box u_2 \odot (v_1, v_2) = u_0 v_1 u_1 v_2 u_2$. Given a particular language, this defines a relation between 2-contexts and 2-words. We again define the polar maps given sets of 2-contexts C_2 and 2-words S_2.

$$C_2' = \{(v_1, v_2) | \forall f \in C_2, f \odot (v_1, v_2) \in L\}$$

$$S_2' = \{f | \forall (v_1, v_2) \in S_2, f \odot (v_1, v_2) \in L\}$$

Again we say that a set of 2-words, S_2 is closed if $S_2'' = S_2$. The set of all closed sets of 2-words in a language forms a lattice, which we call $\mathfrak{B}^2(L)$. These closed sets have an interesting structure, and are closely related, as one would expect, to the elements of $\mathfrak{B}(L)$, which we will now write $\mathfrak{B}^1(L)$. Indeed suppose X_2 is a closed set of 2-words (an element of $\mathfrak{B}^2(L)$). Suppose $X_2 \supseteq Y_1 \times Z_1$, for sets of strings Y_1, Z_1, then $X_2 \supseteq Y_1'' \times Z_1''$; in other words we can write any element of $\mathfrak{B}^2(L)$ as $\bigcup_i Y_1^i \times Z_1^i$, as a union of products of the order-1 lattice. Therefore there is a natural map from $\mathfrak{B}(L) \times \mathfrak{B}(L)$ to $\mathfrak{B}^2(L)$ given by $h(X,Y) = (X \times Y)''$. Indeed we can consider these elements to define a relation between two closed sets of strings: it may be the case that an element of $\mathfrak{B}^2(L)$ is equal to $X \times Y$ for some $X, Y \in \mathfrak{B}^1(L)$, in which case the relation is trivial and there is no gain in using the order 2 lattice. In the interesting cases of non-context-free languages, this relation in general will be infinite in the sense that it can only be defined as an infinite union of products. In linguistically interesting cases this relation will often correspond to the relation between a displaced constituent (e.g. a *wh*-phrase in English) and the constituent that it has been displaced out of.

We can define a number of concatenation operations, and their associated residuals. For two 2-words, there are a number of ways that they can be concatenated to form another 2-word or a word. For example, (u_1, v_1) and (u_2, v_2) could be combined to form $(u_1 u_2, v_1, v_2), (u_1, u_2 v_1 v_2), (u_1 u_2, v_2 v_1)$ and many others. For brevity we will only consider some of these. Given sets of 2-words X_2, Y_2, we can define a family of concatenation operations of which we just show two: $X_2 \oplus Y_2 = \{(u_1 u_2, v_1 v_2) | (u_1, v_1) \in X_2, (u_2, v_2) \in Y_2\}''$, $X_2 \ominus Y_2 = \{(u_1 u_2, v_2 v_1) | (u_1, v_1) \in X_2, (u_2, v_2) \in Y_2\}''$. We have taken the closure of the resulting set of multiwords, giving in these cases associative binary operations on $\mathfrak{B}^2(L)$. The operation $\ominus$ corresponds to the MCFG production $Z(u_1 v_1, v_2, u_2) := X(u_1, u_2), Y(v_1, v_2)$ in Horn clause notation. Indeed $\{(\lambda, \lambda)\}''$ is a left and right identity for both operations. Unsurprisingly we can again define the left and right residuals of the operations which will give us some more residuated structures.

These operations are

$$X_2/_{\ominus}Y_2 = \{(v_1, v_2) \mid \forall(u_1, u_2) \in Y_2(v_1u_1, u_2v_2) \in X_2\} \tag{5}$$

$$Y_2\backslash_{\ominus}X_2 = \{(v_1, v_2) \mid \forall(u_1, u_2) \in Y_2(u_1v_1, v_2u_2) \in X_2\} \tag{6}$$

The sets defined here are closed, and satisfy the equations $X_2 \ominus Y_2 \subseteq Z_2$ iff $X_2 \subseteq Z_2/_{\ominus}Y_2$ iff $Y_2 \subseteq X_2\backslash_{\ominus}Z_2$.

At this level of the hierarchy again we have the same metatheoretical issues. We can represent the tuples of strings directly using a MCFG; the same arguments that we used in the case of CFGs suggest that the nonterminals of dimension 2 should represent closed sets of 2-words. We can view the logic, in this case Morrill's discontinuous calculus, as the equational theory of this algebraic structure. Finally, these can be learned using an extension of the distributional techniques we discussed earlier [25,26] .

7 Discussion

The standard model theory for the Lambek calculus uses free models of various types. In particular we have the standard models: the free semigroup models 2^{Σ^+} (or 2^{Σ^*} if we allow empty antecedents). It is easy to verify that the Lambek calculus is sound with respect to these free models, and it can also be proved to be complete [2]. Morrill [1] says:

> Since language in time appears to satisfy the cancellation laws, the free semigroup models appear to be the most ontologically committed and therefore scientifically incisive models.

The cancellation laws are of the form $u \bullet v = u \bullet w$ implies $v = w$. Clearly if equality here just means equality of strings, then this law is true, since the strings are part of a free semigroup. But if = refers to linguistic/distributional equality, then this is not true: for example, words can be ambiguous in isolation and unambiguous in context. “the can” might be distributionally identical to “the jug” but “can” and “jug” are very different.

Lambek's stipulation that the algebraic structures must be free is thus, from our point of view partially correct. The strings are generated freely giving us the free monoid Σ^*; but the subsets of these are not – we do not have the free join semilattice but only the lattice of closed sets.

7.1 Proof Theory

In general the relation of syntactic entailment is undecidable. Post [27] showed that in general determining whether $x \equiv_T y$ is undecidable with respect to a Thue system. Nonetheless there are subclasses of these sets of equations which have inference procedures that are efficient for certain classes of conclusions. Recall that in order to parse, we are only interested in proving equations of the form $\dot{w}\dot{\subseteq}\dot{L}$.

Lexicalisation is one solution. By requiring all the equations to be of the form $\dot{a}\dot{\subseteq}T$, we can reduce the problem to one which is decidable since it is, approximately, the equational theory of residuated lattices. This solves that part of the problem. But this is only one solution. For suitable other classes of equations the required proofs can be solved efficiently: indeed in polynomial time, rather than the NP-hardness of the Lambek calculus.

7.2 Conclusion

The main claim of this paper is that since languages have an intrinsic residuated lattice structure, grammatical formalisms that use residuated lattices, such as Lambek grammars should be based on this structure. The crucial consequence of this is that we can then learn these grammars efficiently under various different learning paradigms. We deviate in two important respects from Lambek's metaprogram – we abandon the free models, and we may abandon lexicalisation. Lambek says:

> If we take such a program seriously we are not allowed to state a rule such as $N_s \subseteq S/(N\backslash S)$ which although plausible[4] is neither listed in the dictionary nor derivable from general principles. . . .

We might say that we are not concerned with rules that are 'plausible', but rather with rules that are true or correct in a given language. If they are true, and we can reason efficiently with them using a sound logical calculus, and in addition learn them, then there seems little reason to forbid them.

Pereira [28] in his review of Morrill's 1994 book on logical grammar says:

> Types and allowed type inferences are not arbitrary formal machinery but instead reflect precisely the combinatory possibilities of the underlying prosodic algebra.

Here our prosodic algebra is much simpler than the one Pereira is discussing, but the observation is very apt. Grammars are theories and languages (or algebras associated with them) are models. Substructural logics can be viewed as the equational theories of certain types of lattices; logical grammars then can be viewed as theories of syntactic concept lattices. We can view this as a principled misinterpretation of Chomsky's dictum [29],

> A grammar of the language L is essentially a theory of L.

Of course, Chomsky meant a *scientific* theory. We, in contrast, propose viewing a grammar as a finite set of equations whose deductive closure defines the language: as a logical theory in other words.

This reduces the arbitrariness of grammars – there is a single specific algebraic object that we are trying to model, be it the syntactic concept lattice, the syntactic monoid, or some multisorted algebra of discontinuous strings as in Section 6. Many different formalisms can be framed in this way, that depend on the

[4] N_s here refers to a mass noun like 'water'.

type of algebra that we are modeling. For suitable restricted sets of equations we can perform the relevant inferences efficiently.

Learnability has always been a central concern of modern linguistics, and the uniformity of the syntax semantics interface as conceived in the categorial grammar tradition is very appealing from this perspective. As Pereira says:

> ... uniformity must also go backwards, if the use and meaning of a sign is to be induced from its appearance with other signs of appropriate type.

By basing the logic on a well defined observable structure, we can learn the use cleanly; and perhaps ultimately the meaning as well.

Acknowledgments. I am grateful to Glynn Morrill, Shalom Lappin, Hans Leiss, Ryo Yoshinaka, Zhaohui Luo and Robin Adams for helpful discussions. Any errors are of course my own.

References

1. Morrill, G.: Categorial grammar: Logical syntax, semantics and processing. Oxford University Press (2011)
2. Pentus, M.: Models for the Lambek calculus. Annals of Pure and Applied Logic 75(1-2), 179–213 (1995)
3. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65(3), 154–170 (1958)
4. Lambek, J.: Categorial and categorical grammars. In: Oehrle, R.T., Bach, E., Wheeler, D. (eds.) Categorial Grammars and Natural Language Structures, vol. 32, pp. 297–317. D. Reidel (1988)
5. Buszkowski, W.: Compatibility of a categorial grammar with an associated category system. Mathematical Logic Quarterly 28(14-18), 229–238 (1982)
6. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49(4), 431–454 (1990)
7. Kanazawa, M.: Learnable classes of categorial grammars. PhD thesis, Stanford University (1994)
8. Angluin, D.: Inference of reversible languages. Journal of the ACM 29(3), 741–765 (1982)
9. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
10. de la Higuera, C.: Grammatical inference: learning automata and grammars. Cambridge University Press (2010)
11. Clark, A.: Distributional Learning of Some Context-Free Languages with a Minimally Adequate Teacher. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 24–37. Springer, Heidelberg (2010)
12. Shirakawa, H., Yokomori, T.: Polynomial-time MAT Learning of C-Deterministic Context-free Grammars. Transactions of the Information Processing Society of Japan 34, 380–390 (1993)
13. Clark, A.: A Learnable Representation for Syntax Using Residuated Lattices. In: de Groote, P., Egg, M., Kallmeyer, L. (eds.) FG 2009. LNCS (LNAI), vol. 5591, pp. 183–198. Springer, Heidelberg (2011)

14. Clark, A.: Learning Context Free Grammars with the Syntactic Concept Lattice. In: Sempere, J.M., García, P. (eds.) ICGI 2010. LNCS, vol. 6339, pp. 38–51. Springer, Heidelberg (2010)
15. Yoshinaka, R.: Towards Dual Approaches for Learning Context-Free Grammars Based on Syntactic Concept Lattices. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 429–440. Springer, Heidelberg (2011)
16. Yoshinaka, R.: Integration of the Dual Approaches in the Distributional Learning of Context-Free Grammars. In: Dediu, A.-H., Martín-Vide, C. (eds.) LATA 2012. LNCS, vol. 7183, pp. 538–550. Springer, Heidelberg (2012)
17. Galatos, N., Jipsen, P., Kowalski, T., Ono, H.: Residuated lattices: an algebraic glimpse at substructural logics. Elsevier (2007)
18. Costa Florêncio, C.: Learning categorial grammars. PhD thesis, Utrecht University (2003)
19. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
20. Kanazawa, M.: The Lambek calculus enriched with additional connectives. Journal of Logic, Language and Information 1(2), 141–171 (1992)
21. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
22. Clark, A.: Towards General Algorithms for Grammatical Inference. In: Hutter, M., Stephan, F., Vovk, V., Zeugmann, T. (eds.) ALT 2010. LNCS (LNAI), vol. 6331, pp. 11–30. Springer, Heidelberg (2010)
23. Clark, A.: Efficient, correct, unsupervised learning of context-sensitive languages. In: Proceedings of the Fourteenth Conference on Computational Natural Language Learning, Uppsala, Sweden, pp. 28–37. Association for Computational Linguistics (July 2010)
24. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88(2), 229 (1991)
25. Yoshinaka, R.: Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. Theoretical Computer Science 412(19), 1821–1831 (2011)
26. 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)
27. Post, E.: Recursive unsolvability of a problem of Thue. The Journal of Symbolic Logic 12(1), 1–11 (1947)
28. Pereira, F.: Review of Type logical grammar: categorial logic of signs by Glyn Morrill. Computational Linguistics 23(4), 629–635 (1997)
29. Chomsky, N.: Syntactic Structures. Mouton (1957)

Ludics and Natural Language: First Approaches*

Christophe Fouqueré[1] and Myriam Quatrini[2]

[1] LIPN, Université Paris 13 and CNRS,
christophe.fouquere@lipn.univ-paris13.fr
[2] IML, Université d'Aix-Marseille and CNRS
quatrini@iml.univ-mrs.fr

Abstract. Ludics is a rebuilding of Linear Logic from the sole concept of interaction on objects called designs, that abstract proofs. Works have been done these last years to reconsider the formalization of Natural Language: a dialogue may be viewed as an interaction between such abstractions of proofs. We give a few examples taken from dialogue modeling but also from semantics or speech acts to support this approach.

1 Introduction

Ludics is a rebuilding of Linear Logic from the sole concept of interaction on objects called designs that abstract proofs. Works have been done these last years to reconsider the formalization of Natural Language: a dialogue may be viewed as an interaction between such abstractions of proofs. Among the domains that have been explored, one may cite formal semantics [1] and pragmatism: speech acts [2] and figures of dialogue [3]. The approach follows what have been done initially for dialogues [4] where dialogues are modeled in proof theory using Ludics.

Ludics [5] is a logical framework developed by J.-Y. Girard around 2000. The motto underlying Ludics is: *interaction is a central concept in logic*. This motto may be considered as following the fruitful paradigm between logic and computer science, namely the Curry-Howard isomorphism. Recall that this isomorphism establishes a perfect correspondence between programs and their execution on one side, and formal proofs and cut elimination on the other side. A cut between conclusions of proofs enables an interaction between these proofs, when one of the formulas is the conclusion of one proof, whereas the other is a hypothesis of the other proof. Called modus ponens, it is the main ingredient of reasoning; one of the main results of proof theory states that it is always possible to *normalize* a proof, *i.e.* to transform a proof with cuts to an equivalent proof without cuts. The dynamics of computation may therefore be considered as the heart of logic.

After decades of work studying the properties of this dynamics and expanding its scope of relevance, Ludics reverses priorities of concepts. Traditionally, formulas and proofs are first set, the cut being one of the rules used in the definition of

* This work was completed with the support of the ANR project LOCI number 0212 02.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 21–44, 2012.

what is a correct proof. The cut elimination procedure completes the picture by adding a dynamics. Then, progressively, to ensure dynamic properties, formulas and proofs have been refined. Linear Logic [6] illustrates these changes: conjunction and disjunction connectives are each replaced by two versions, additive and multiplicative, and the framework of proofnets gives a geometrical presentation to proofs where the cut elimination property still appears as a property given a posteriori, *i.e.* as a reduction of such graphs. A contrario, in Ludics, cut, *i.e.* interaction, is a primitive concept. Neither formulas nor proofs are considered primitive but *designs*, whose sole purpose is that they carry on the interaction. In fact, interaction takes place between two designs as a step-by-step travel through two dual paths, one path in each design. A design is then nothing else but a set of potential paths where interaction may take place. Moreover, a design is essentially defined by its counter-designs: those with whom it interacts.

The concrete basic steps of interaction (called *actions* in Ludics) correspond to the basic steps of cut elimination. They allow for the exploration of a design as one may explore a formula through its main connective to its subformulas, and so on. However, a design abstracts from the notion of formula: a design has only loci, *i.e.* adresses, where interaction goes through. When a set of designs is given, the space where interaction with these designs may take place is also given. This space is defined by all the counter-designs of each design of this set. This gives an external viewpoint on designs, from which one may observe regularities. Formulas may then be retrieved as sets of designs closed relatively to these counter-designs. Indeed, exploring a set of designs reduces to exploring each design of this set. When it is closed, a set of designs describes all the ways to explore the object it represents, until its undecomposable elements. This has to be related to the concept of a formula, defined inductively by its connectives and subformulas that compose it until obtaining propositional variables or constants. Moreover, and this is an essential property of Ludics, some designs associated with a formula may be proofs (of this formula): those designs that satisfy suitable properties, among which, precisely, the fact that the exploration can always continue until a suitable term. So, truthness of a formula requires the existence of some proof belonging to the set of designs associated to this formula.

The formalization of Natural Language uses these “primitive” objects of the interaction, upstream of a reconstruction of logic. The fact that interaction is the fundamental concept of Ludics justifies its relevance to the study of dialogues, hence also to Natural Language. Remark that Ludics has also been used in the same spirit for formalizing web processes [7].

Section 2 is devoted to a presentation of Ludics. We describe first what is an action and what conditions should satisfy a tree of actions to be a design. We give the definition of interaction between nets of designs and give the main steps for rebuilding logic. In particular, we recall the main theorems that Ludics satisfies, *i.e.* internal completeness. Then we present a sequent calculus that mimics the structure of designs. Finally, we summarize works that have been done to extend Ludics.

Section 3 is an introduction to the formalization of dialogues in Ludics. We consider short dialogues to show how Ludics may be used. In particular, we express how to represent a divergent dialogue, and how to represent presuppositions. Works concerning the modeling of meaning are presented in section 4. Contrarily to standard semantics, the meaning of an utterance is given by the way this utterance may be justified, explored in dialogues. Hence a set of designs may be associated with an utterance. We show with one example that classical semantics may be retreived thanks to the reconstruction of logic available with Ludics. Finally we present a few schemas that may be used for representing simple speech acts.

2 Ludics: Basic Notions

In this section we present the basic concepts of Ludics. The reader may find thorough presentations of Ludics obviously in [5] but also in [8,9,10,11].

2.1 Actions, Designs, Interaction: An Informal Description

In Ludics, primitive elements of interaction are **actions**. Actions are polarized and appear as dual pairs: for each positive action (resp. negative) κ, there exists a dual negative action (resp. positive) $\overline{\kappa}$ and $\overline{\overline{\kappa}} = \kappa$. As in game theory, a **justification** relation between actions is given. Actions are organized in alternate sequences called **chronicles** in Ludics. These sequences should satisfy the following conditions: (i) a positive action can be either *initial* or *justified* by a negative action that precedes it in the sequence, (ii) a negative action, except the first which may be initial, is justified by the positive action immediately preceding. Following the metaphor of games, these alternate sequences of actions can be seen as *plays* that can be grouped to form *strategies* named designs in Ludics [12]. Thus, **designs** are sets of chronicles that may be interleaved to give rise to what is travelled during an interaction. Not all sets of chronicles can be designs: in particular a design should be organized as a forest, with only one root when the root is a positive action.

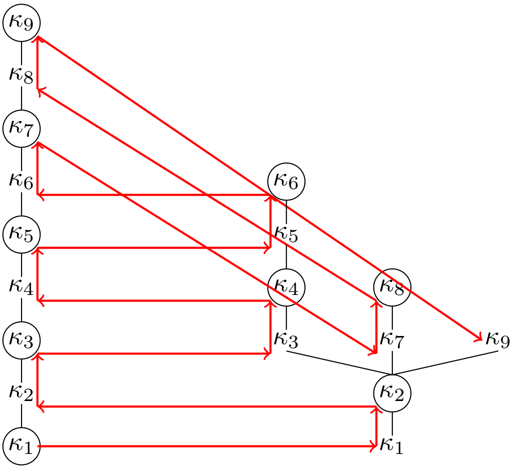
An **interaction** occurs between two designs when each of them contains a path dual of the other path (*e.g.* example 1 on the left). It is a travel through these two designs, which runs as follows:

- It starts with the object that contains as root a positive action (hence unique).
- Every time it goes through a positive action $\textcircled{\kappa}$ of one of the two designs, the travel continues in the other design on the negative action that is dual κ (when it exists), then continues on the only positive action that follows this negative action in the same design.

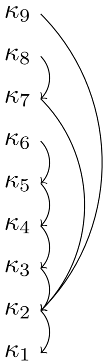
This process continues as long as positive actions that are visited have a dual negative action in the other design. If such a negative action is not found, the process *diverges*, *i.e.* fails.

The **trace** of an interaction is given by the sequence of pairs of dual actions followed during this interaction (*e.g.* example 1 on the right). Conversely, it is possible, from the trace of an interaction, to recover the minimal designs whose interaction gives this trace.

Example 1. In figure on the left, circled actions are positive. In figure on the right, the name of the action is noted in place of the pair of dual actions for easiness of reading, an arrow from action κ to action κ' denotes that κ is justified by κ'.



Interaction between two designs



Trace of the interaction with justification between actions.

2.2 Actions, Designs, Interaction: A Formal Presentation

The definition of actions in Ludics is precisely defined by means of the concept of *locus* or *address*. This concept plays a very special role in Ludics as the intended use consists in replacing logical formulas as designs should replace proofs.

Definition 1 (Action). *A* proper action κ *is a triple* (ϵ, ξ, I) *where:*
- $\epsilon \in \{+, -\}$ *is the* **polarity** *of* κ*,*
- *the finite sequence of integers* ξ *is the* **focus** *(the address) of* κ*,*
- *the finite set of integers* I *is the* **ramification** *of* κ*.*

Besides proper actions, there exists also a special positive action, the **daïmon***, noted* $\dagger$*.*

In the following, we note $\xi.i$ the sequence of integers ξ followed by the integer i. We note also $\overline{\epsilon}$ the dual polarity of ϵ, *e.g.* if $\epsilon = +$ then $\overline{\epsilon} = -$. Finally, if $I = \{i_1, i_2, \ldots, i_n\}$, then $\xi.I$ is the set of sequences of integers $\{\xi.i_1, \xi.i_2, \ldots, \xi.i_n\}$. The ramification of an action determines the loci that may be used as continuations during an interaction. Relations of *justification* and *duality* follow from the previous definition. An action $(\epsilon, \xi.i, J)$ is justified by the action $(\overline{\epsilon}, \xi, I)$ when $i \in I$. The dual action of the action (ϵ, ξ, I) is the action $(\overline{\epsilon}, \xi, I)$.

The main objects of Ludics are the designs, *i.e.* sets of *chronicles* satisfying specific conditions. A chronicle is a sequence of actions that satisfies also conditions to ensure in particular justification and linearity.

Definition 2 (Chronicle). *A* **chronicle** $\mathfrak{c}$ *is a non-empty, finite, alternate sequence of actions such that*

- Proper positive action: *A proper positive action is either justified,* i.e. *its focus is built from one of the previous actions in the sequence, or is called an initial action.*
- Negative action: *A negative action may be initial, in that case it is the first action of the chronicle. Otherwise, this action is justified by the positive action that precedes it immediately.*
- Linearity: *Actions have distinct foci.*
- Daïmon: *If the action daïmon is present, this action is the last one of the chronicle.*

Definition 3 (Coherence between chronicles). *Two chronicles* $\mathfrak{c}_1$ *and* $\mathfrak{c}_2$ *are* **coherent***,* i.e. $\mathfrak{c}_1 \frown \mathfrak{c}_2$*, when the following conditions are satisfied:*

- Comparability: *Either one chronicle extends the other, or they differ first on negative actions,* i.e. *if* $w\kappa_1 \frown w\kappa_2$ *then either* $\kappa_1 = \kappa_2$ *or* κ_1 *and* κ_2 *are negative actions.*
- Propagation: *If the two chronicles diverge on negative actions with distinct foci, then ulterior actions in one chronicle are distinct from ulterior actions in the other* i.e. *if* $w(-, \xi_1, I_1)w_1\sigma_1 \frown w(-, \xi_2, I_2)w_2\sigma_2$ *with* $\xi_1 \neq \xi_2$ *then* σ_1 *and* σ_2 *have distinct foci.*

A *base* is associated to each chronicle. A base is a sequent of loci denoted $\Gamma \vdash \Delta$ such that Δ is a finite set of loci and Γ contains at most one locus. Furthermore, loci of $\Gamma \cup \Delta$ are pairwise disjoint, *i.e.* there does not exist a locus that is a sublocus of another one. If Γ is empty, the base is *positive*, otherwise it is *negative*. A chronicle $\mathfrak{c}$ has a **base** $\Gamma \vdash \Delta$ as soon as either Γ is not empty and its locus is the focus of the first (negative) action of the chronicle, or Γ is empty and the first action of $\mathfrak{c}$ is positive and Δ contains all the foci of initial positive actions of the chronicle.

Definition 4 (Design, Net)

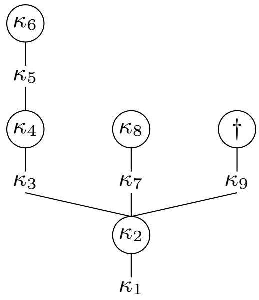
- *A* **design** $\mathfrak{D}$*, of base* $\Gamma \vdash \Delta$*, is a set of chronicles of base* $\Gamma \vdash \Delta$*, such that the following conditions are satisfied:*
  - Forest: *The set is prefix-closed.*
  - Coherence: *The set is a clique of chronicles,* i.e. $\forall \mathfrak{c}_1, \mathfrak{c}_2,\ \mathfrak{c}_1 \frown \mathfrak{c}_2$.
  - Positivity: *A chronicle without extension in* $\mathfrak{D}$ *ends by a positive action.*
  - Totality: $\mathfrak{D}$ *is not empty when the base is positive, in that case each chronicle begins with a (unique) positive action.*
- *A* **net** *is a finite set of designs of disjoint bases.*

Designs may also be presented in a sequent-like style: it is a tree whose nodes are sequents made of loci in place of formulas. An action specifies the relation between a node and its daughters: loci that are present in nodes are subloci of ancestors in the tree. The root of the tree contains initial loci that may be used in an interaction. As the framework is linear, a locus that is initial cannot be a sublocus of other initial loci.

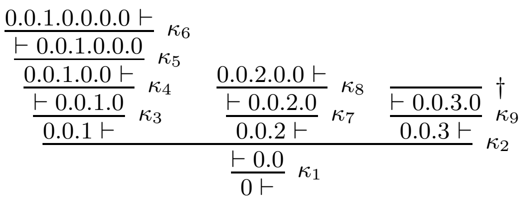
Definition 5 (Designs in a sequent-style)

- *A* **sequent** $\varGamma \vdash \varDelta$ *is given with two finite sets of addresses or loci* $\varGamma$ *and* $\varDelta$, i.e. *finite sequences of integers, such that* $\varGamma$ *contains at most one element and there is no address of* $\varGamma \cup \varDelta$ *is an initial sequence of another one.*
- *A* **sequent-design** *(simply called design), with base the sequent* $\varGamma \vdash \varDelta$ *is a tree of sequents built from the three following rules:*
  - Daimon
  $$\frac{}{\vdash \varDelta}\ \dagger$$
  - Positive rule
  $$\frac{\dots \quad \xi.i \vdash \varDelta_i \quad \dots}{\vdash \varDelta, \xi}\ (+, \xi, I)$$
  for $i \in I$, $\varDelta_i$ *are pairwise disjoint and included in* $\varDelta$.
  - Negative rule
  $$\frac{\dots \quad \vdash \xi.I, \varDelta_I \quad \dots}{\xi \vdash \varDelta}\ (-, \xi, \mathfrak{R})$$
  $\mathfrak{R}$ *is a set (that may be empty or infinite) of ramifications. For all* $I \in \mathfrak{R}$, $\varDelta_I$, *non necessarily disjoint, are included in* $\varDelta$.

Example 2 (example 1 ctd)



a design


$0.0.1.0.0.0.0 \vdash$ κ_6
$\vdash 0.0.1.0.0.0$ κ_5
$0.0.1.0.0 \vdash$ κ_4
$\vdash 0.0.1.0$ κ_3
$0.0.1 \vdash$
$0.0.2.0.0 \vdash$ κ_8
$\vdash 0.0.2.0$ κ_7
$0.0.2 \vdash$
$\dagger$
$\vdash 0.0.3.0$ κ_9
$0.0.3 \vdash$ κ_2
$\vdash 0.0$ κ_1
$0 \vdash$

in a sequent-style

We provide in subsection 2.3 a brief presentation of a sequent calculus related to Ludics. We already may give intuitions to interpret logically a design. In a bottom-up reading, a design corresponds to a proof search. To recover a proof in

Linear Logic, one tries to substitute formulas to addresses. The action specifies the principle generalized connective of the formula, combining connectives $\otimes$ and $\oplus$ for a positive action, or connectives $⅋$ and $\&$ for a negative action. Subformulas are found by iterating this operation on premisses of the rule. If this process ends in specific cases[1], without using the daïmon, then the result is an explicit proof of a formula.

Example 3

- The simplest design consists in a unique positive action, the daimon. It is noted $\mathfrak{Dai}_+$: $\dfrac{}{\vdash \ \Gamma}\dagger$. It is a positive design. We note later that it interacts with all negative designs of the same base: the action daïmon ends an interaction.
- The fax noted $\mathcal{F}ax_{\xi,\xi'}$ is a design recursively defined in the following way:

$$\dfrac{\dots \qquad \dfrac{\dots \qquad \overset{\mathcal{F}ax_{\xi'.i,\xi.i}}{\xi'.i \vdash \xi.i} \qquad \dots}{\vdash \xi.I,\xi'}\,(+,\xi',I) \qquad \dots}{\xi \vdash \xi'}\,(-,\xi,\mathcal{P}_f(\mathbb{N}))$$

The set of finite sets of integers is noted $\mathcal{P}_f(\mathbb{N})$. We note later that it allows for delocalizing a design from a locus ξ to another locus ξ'.

Interaction, *i.e.* cut elimination, is the process of normalization of specific nets of designs called *cut nets*. In a cut net, addresses that appear in the bases of designs are either distinct, or present once in the positive part of a base and once in the negative part of a base, such pairs are the *cuts* of the net. Hence, in a cut net, the graph made of bases and cuts is acyclic and connex. We give below the definition of interaction in case of a *closed* cut net, *i.e.* addresses in bases are all part of cuts. The reader may find a definition of interaction for general cut nets in [5]. Remark that in a closed cut net, there must exist a positive design, *i.e.* a design whose base has an empty left part. Such a design is called a *principal* design.

Definition 6 (Interaction for closed cut nets). *Let $\mathfrak{R}$ be a closed cut net, the result of the interaction is a design noted $[\![\mathfrak{R}]\!]$ and defined in the following way: let $\mathfrak{D}$ be the principal design of $\mathfrak{R}$, with first action κ,*

- *if κ is a daïmon, then $[\![\mathfrak{R}]\!] = \mathfrak{Dai}_+$,*
- *otherwise κ is a proper positive action $(+, \sigma, I)$ such that σ is part of a cut with another design, whose first action is $(-, \sigma, \mathcal{N})$ (where $\mathcal{N}$ aggregates the ramifications of the negative actions of same focus σ):*
  - *If $I \notin \mathcal{N}$, then interaction fails (diverges).*
  - *Otherwise, interaction continues with the connected part of the subdesigns we get with I and the remainder of $\mathfrak{R}$.*

Hence either an interaction of a closed cut net diverges, or it does not end, or the result is the design $\mathfrak{Dai}_+$.

[1] Empty sets of ramifications, empty ramifications, recursive configurations, ...

Example 4

- Consider the design $\mathfrak{D}$ of base $\vdash \xi$ and with unique action $(+, \dagger)$, and $\mathfrak{E}$ of base $\xi \vdash$, then $[\![\mathfrak{D}, \mathfrak{E}]\!] = \mathfrak{D}$.
- Let $\mathfrak{D}$ be a design of base $\vdash \xi$ then normalization with the fax $\mathcal{F}ax_{\xi,\xi'}$ is the design $\mathfrak{D}'$ we get from $\mathfrak{D}$ by substituting ξ with ξ' in each chronicle of $\mathfrak{D}$. For example, if we consider that the first action of $\mathfrak{D}$ is $(+, \xi, I)$, the interaction goes as follows:

$$\dfrac{\dfrac{\overset{\mathfrak{D}_1}{\xi.1 \vdash} \quad \ldots \quad \overset{\mathfrak{D}_n}{\xi.n \vdash}}{\vdash \xi}\,(+,\xi,I) \qquad \dfrac{\ldots \quad \dfrac{\ldots \quad \overset{\mathcal{F}ax_{\xi'.i,\xi.i}}{\xi'.i \vdash \xi.i} \quad \ldots}{\vdash \xi.I, \xi'}\,(+,\xi',I) \quad \ldots}{\xi \vdash \xi'}\,(-,\xi,\mathcal{P}_f(\mathbb{N}))}{}$$

After two steps, we get:

$$\dfrac{\overset{[\![\mathfrak{D}_1, \mathcal{F}ax_{\xi'_1,\xi_1}]\!]}{\xi'.1 \vdash} \qquad \ldots \qquad \overset{[\![\mathfrak{D}_n, \mathcal{F}ax_{\xi'_n,\xi_n}]\!]}{\xi'.n \vdash}}{\vdash \xi'}\,(+,\xi',I)$$

The process follows in a recursive manner.

Orthogonality is defined in the following way:

Definition 7 (Orthogonal, Behaviour)

- *Let* $\mathfrak{D}$ *be a design of base* $\xi \vdash \sigma_1, \ldots, \sigma_n$ *(resp.* $\vdash \sigma_1, \ldots, \sigma_n$*), the net of designs* $\mathfrak{R} = (\mathfrak{A}, \mathfrak{B}_1, \ldots, \mathfrak{B}_n)$ *(resp.* $\mathfrak{R} = (\mathfrak{B}_1, \ldots, \mathfrak{B}_n)$*), where* $\mathfrak{A}$ *has the base* $\vdash \xi$ *and* $\mathfrak{B}_i$ *has the base* $\sigma_i \vdash$*, belongs to the orthogonal of* $\mathfrak{D}$ *noted* $\mathfrak{D}^\perp$ *if* $[\![\mathfrak{D}, \mathfrak{R}]\!] = \mathfrak{D}\mathfrak{ai}_+$.
- *Let* E *be a set of designs of same base,* $\mathrm{E}^\perp = \bigcap_{\mathfrak{D} \in \mathrm{E}} \mathfrak{D}^\perp$.
- E *is a* behaviour *if* $\mathrm{E} = \mathrm{E}^{\perp\perp}$. *A behaviour is* positive *(resp.* negative*) if the base of its designs is positive (resp. negative).*

Among remarkable facts of Ludics, let us note that designs are completely defined by their interactions, as affirmed by the *separation theorem*: two designs $\mathfrak{D}$ and $\mathfrak{D}'$ of same base are different iff there exists a design $\mathfrak{E}$ such that the results of interaction $[\![\mathfrak{D}, \mathfrak{E}]\!]$ and $[\![\mathfrak{D}', \mathfrak{E}]\!]$ differ. Other main properties are the following ones [5]:

Theorem 1. *Normalization is stable by intersection, is associative and monotonous with respect to the order induced by duals:*

- *If* K *is not empty and for all* $k \in K$, $\mathfrak{R}_k \subset \mathfrak{R}$ *then* $[\![\bigcap_{k \in K} \mathfrak{R}_k]\!] = \bigcap_{k \in K} [\![\mathfrak{R}_k]\!]$.
- *Let* $\{\mathfrak{R}_0, \ldots, \mathfrak{R}_n\}$ *be a net of nets,* $[\![\mathfrak{R}_0 \cup \cdots \cup \mathfrak{R}_n]\!] = [\![[\![\mathfrak{R}_0]\!], \ldots, [\![\mathfrak{R}_n]\!]]\!]$.
- *If* $\mathfrak{D}_0^\perp \subset \mathfrak{E}_0^\perp, \ldots, \mathfrak{D}_n^\perp \subset \mathfrak{E}_n^\perp$ *then* $[\![\mathfrak{D}_0, \ldots, \mathfrak{D}_n]\!]^\perp \subset [\![\mathfrak{E}_0, \ldots, \mathfrak{E}_n]\!]^\perp$

Operations on behaviours are defined hereafter. Two behaviours are considered *disjoint* when the sets of ramifications of their first action are disjoint. Two designs, two behaviours of same polarity are *alien* when the intersection of the unions of the ramifications of their first actions is empty. Finally two positive behaviours $\mathbf{G}$ and $\mathbf{H}$ are *independent* when, if I and I' are two ramifications of first actions of designs in $\mathbf{G}$ (resp. J and J' in $\mathbf{H}$) such that $I \cup J = I' \cup J'$ then $I = I'$ and $J = J'$.

Definition 8

- *Let* $\mathbf{G}_k$ *be a family of behaviours of positive base pairwise disjoint,* $\bigoplus_k \mathbf{G}_k = (\bigcup_k \mathbf{G}_k)^{\perp\perp}$
- *Let* $\mathbf{G}_k$ *be a family of behaviours pairwise disjoint,* $\&_k \mathbf{G}_k = \bigcap_k \mathbf{G}_k$
- *Let* $\mathfrak{A}$ *and* $\mathfrak{B}$ *be two positive alien designs, one defines* $\mathfrak{A} \otimes \mathfrak{B}$ *in the following way: if* $\mathfrak{A}$ *or* $\mathfrak{B}$ *is* $\mathfrak{Dai}_+$*, then* $\mathfrak{A} \otimes \mathfrak{B} = \mathfrak{Dai}_+$*. Otherwise* $\mathfrak{A}$ *and* $\mathfrak{B}$ *have as first action respectively* $(+, \langle\rangle, I)$ *and* $(+, \langle\rangle, J)$*, let us define* $\mathfrak{A}'$ *by replacing in the chronicles of* $\mathfrak{A}$ *the first action* $(+, \langle\rangle, I)$ *by* $(+, \langle\rangle, I \cup J)$*, one defines in the same way* $\mathfrak{B}'$*, then* $\mathfrak{A} \otimes \mathfrak{B} = \mathfrak{A}' \cup \mathfrak{B}'$
- *Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be two positive alien behaviours,* $\mathbf{G} \otimes \mathbf{H} = \{\mathfrak{A} \otimes \mathfrak{B}\ ;\ \mathfrak{A} \in \mathbf{G}, \mathfrak{B} \in \mathbf{H}\}^{\perp\perp}$
- *Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be two negative alien behaviours,* $\mathbf{G} \parr \mathbf{H} = (\mathbf{H}^{\perp} \otimes \mathbf{G}^{\perp})^{\perp}$

Theorem 2

- *(internal additive completeness) Let* $K \neq \emptyset$*,* $\bigoplus_{k \in K} \mathbf{G}_k = \bigcup_{k \in K} \mathbf{G}_k$
- *A behaviour of positive base (resp. negative base) is always decomposable as a* $\bigoplus$ *(resp. a* $\&$*) of connected behaviours.*
- *(adjunction) Let* $\mathfrak{F}$*,* $\mathfrak{A}$*,* $\mathfrak{B}$ *be three designs,* $\mathfrak{F}$ *negative,* $\mathfrak{A}$ *and* $\mathfrak{B}$ *positive, there exists a unique negative design* $(\mathfrak{F})\mathfrak{A}$ *(that does not depend on* $\mathfrak{B}$*) such that* $[\![\mathfrak{F}, \mathfrak{A} \otimes \mathfrak{B}]\!] = [\![(\mathfrak{F})\mathfrak{A}, \mathfrak{B}]\!]$*.*
- *(internal multiplicative completeness) Let* $\mathbf{G}$ *and* $\mathbf{H}$ *be two independent positive behaviours, then* $\mathbf{G} \otimes \mathbf{H} = \{\mathfrak{A} \otimes \mathfrak{B}\ ;\ \mathfrak{A} \in \mathbf{G}, \mathfrak{B} \in \mathbf{H}\}$

Theorem 3 (Full soundness and completeness). *Ludics is fully sound and complete with respect to second-order multiplicative-additive Linear Logic.*

2.3 Back to a Sequent Calculus

Hypersequentialized calculi have been proposed for various fragments or the full Linear Logic (*e.g.* [13,14,1]). Designs may there be considered as (para)proofs of such calculi. The key ingredient to specify what kind of structure should be given to sequents comes from works on focalization in logical programming: Andreoli states in [15] that a proof in Linear Logic can be organized in such a way that decomposition of clustered deterministic (say negative, *i.e.* $\parr$ and &) and non-deterministic (say positive, *i.e.* $\otimes$ and $\oplus$) connectives in a formula alternate: designs may then be viewed as abstracting such concrete and focalized proofs, and taking into account infinity, *e.g.* η-expansion, and failures, *i.e.* at most one of two dual formulas has a proof. Furthermore, although every negative formulas[2] may be concurrently and immediately decomposed, one positive formula may be

[2] A negative formula has a negative connective as the main one.

chosen when a positive step occurs. Such a positive formula is the focus of the application of the rule. Following focalization, we can consider that a sequent contains at most one negative formula and we put its dual on the left part of the sequent (using implicitly De Morgan rules). Hence the right part of a sequent contains only positive formulas. A decomposition step, *i.e.* a bottom-up application of a rule in a sequent calculus, consists in

- either choosing a positive formula to be decomposed, to give rise to a set of (negative) subformulas, hence a set of sequents (one for each negative subformula),
- or decomposing the negative formula to give rise to a set of sets of (positive) subformulas, hence a set of sequents (one for each set of positive subformulas).

The calculus contains then three rules: one for the axiom, and one for the decomposition of each polarity, using synthetic connectives of various arity. A cut rule may be added that does not change the calculus. Positive formulas which are considered in such a sequent calculus are built from a set X of positive atoms according to the following schema:

$$F := X \mid (F^\perp \otimes \cdots \otimes F^\perp) \oplus \cdots \oplus (F^\perp \otimes \cdots \otimes F^\perp)$$

The rules are the following ones:

Axiom rule $\dfrac{}{x \vdash x, \Gamma}$ where $x \in$ **Cut rule** $\dfrac{B \vdash A, \Gamma \quad A \vdash \Delta}{B \vdash \Gamma, \Delta}$

X.

Negative rule **Positive rule**

$$\dfrac{\vdash A_{11}, \ldots, A_{1n_1}, \Gamma \quad \ldots \quad \vdash A_{p1}, \ldots, A_{pn_p}, \Gamma}{(A_{11}^\perp \otimes \cdots \otimes A_{1n_1}^\perp) \oplus \cdots \oplus (A_{p1}^\perp \otimes \cdots \otimes A_{pn_p}^\perp) \vdash \Gamma} \quad \dfrac{A_{i1} \vdash \Gamma_1 \quad \ldots \quad A_{in_i} \vdash \Gamma_p}{\vdash (A_{11}^\perp \otimes \cdots \otimes A_{1n_1}^\perp) \oplus \cdots \oplus (A_{p1}^\perp \otimes \cdots \otimes A_{pn_p}^\perp), \Gamma}$$

where $\cup \Gamma_k \subset \Gamma$ and for $k, l \in \{1, \ldots p\}$, $\Gamma_k \cap \Gamma_l = \emptyset$.

The complete decomposition of connectives of the same polarity may be broken by using *shift* connectives $\downarrow$ and $\uparrow$, that correspond to unary versions of the previous connectives. Let $\downarrow$ (resp. $\uparrow$) change the negative (resp. positive) polarity into the positive (resp. negative) one. Then two following rules may be added to the system:

Shift rules $\dfrac{A \vdash \Gamma}{\vdash \downarrow A^\perp, \Gamma} \qquad \dfrac{\vdash A, \Gamma}{\downarrow A^\perp \vdash \Gamma}$

Example 5. Let A, B and C be negative formulas, formulas $A \otimes B \otimes C$ and $A \otimes \uparrow (B \otimes C)$ are positive. Their decomposition using the previous rules are the following ones:

$$\dfrac{A^\perp \vdash \quad B^\perp \vdash \quad C^\perp \vdash}{\vdash A \otimes B \otimes C} \qquad \dfrac{A^\perp \vdash \quad \dfrac{\dfrac{B^\perp \vdash \quad C^\perp \vdash}{\vdash B \otimes C}}{\downarrow (B \otimes C)^\perp \vdash}}{A \otimes \uparrow (B \otimes C)}$$

2.4 Extensions of Ludics

Works have been done to extend Ludics beyond the multiplicative-additive fragment of Linear Logic. These works are of particular interest as linearity or sequentiality is no more a constraint. In current use of Ludics in Natural Language, this has not been considered. However, it may be useful in some cases.

First, *Ludics nets* were developed as a game model for concurrent interaction by F. Maurel and C. Faggian [16] and more thoroughly analyzed by P.-L. Curien and C. Faggian in [13]: in some way, Ludics nets are abstract proof-nets, hence graphs, in the same way as designs are abstract proof trees.

K. Terui proposed in [17] a reformulation of Ludics more suitable from a computational point of view. *c-designs* are built as Girard's designs except that (i) variables may occur as generic addresses (possibly in an infinite number), (ii) positive nodes may be explicit internal cuts or serve to model divergence of interaction, (iii) c-designs may not be linear. The syntax of c-designs is such that they may be more easily used for programming purposes, and a grammar allows for a finite specification of them by means of generators. K. Terui uses this model for characterizing word languages where the acceptance relation between a word and a grammar is expressed w.r. to orthogonality between c-designs. Its main results relate automata and c-designs:
- Girard's finite designs recognize exactly regular languages.
- Cuts are necessary (hence the use of c-designs) to recover the full expressive power of Turing machines.

M. Basaldella and C. Faggian in [12] extended Ludics to be able to deal with exponentials. Contrarily to the works of Terui, they only extend the language with *neutral* actions to represent exponentials. They obtain a full completeness result with respect to an hypersequentialized calculus for Linear Logic.

3 Dialogues and Argumentation

Ludics has been used in a series of papers for representing several aspects of Natural Language. This section is devoted to the representation of dialogues (*e.g.* [4,3]). Shortly speaking, a dialogue may be considered as the trace of the interaction between two designs, one for each locutor. In elementary cases, an utterance of a locutor is a positive action in her design that has to be present as a negative action in her interlocutor for the dialogue to continue[3]. Generally, an utterance is not as simple as a unique action: an utterance may be a complex sentence including several propositions, it may make use of presupposition, . . . , or it may correspond to a misunderstanding or an end of the dialogue. The formalization of dialogues in Ludics considers that an intervention in a dialogue conveys one or several **dialogue acts**, that Landragin [18] defines as "the minimal unit of communication in a dialogical context". A dialogue act is a communicational fact whose role is to fuel the dynamics and determine the shape of

[3] Note that the duality positive/negative does not correspond to a duality question/answer as in game semantics but to a duality production/reception in dialogue modeling.

the dialogue. It may be explicit or implicit, verbal or not (*e.g.* an acknowledgment given as a gesture). It may appear as one or more propositions, but also as part of a proposition (word, adverb, ...). It expresses an entitlement or a decision of the speaker, and also its acknowledgment by the addressee. In some sense, it is quite close to a speech act. However a speech act may correspond to several dialogue acts as shown in following examples. Dialogue acts are indeed more elementary than speech acts. They can be seen as the basic blocks from which one builds interpretation for dialogical interventions or even utterances. Formally, a dialogue act may be defined as an action in Ludics together with the expression that reveals the dialogue act in the intervention. Such an expression may be a proposition, a word (*e.g.* a single adverb, a noun), a prosodic feature, a non verbal sign (a nod, a shake, a slap, ...). In trivial cases, an intervention is a unique dialogue act. Otherwise a turn of speech has to be decomposed into sequences of dialogue acts, hence may correspond to a complex design. Note that the representation of an utterance in terms of dialogue acts is dependent of the context of the dialogue, and in particular of past interventions that occurred.

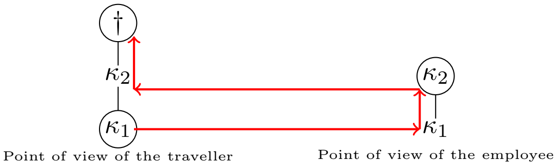
Let us give a small example of a dialogue, its rudimentary interpretation in terms of dialogue acts and the way interaction is done with Ludics.

Example 6. Let us consider the following example between a traveller T and an employee E:
– T: *What time does the next train to Paris leave?*
– E: *7:45 p.m.*
– T: *Thanks.*

In a first approximation, each intervention is interpreted as a unique dialogue act:

- dual actions $\kappa_1/\overline{\kappa_1} = (+/-, \xi, \{0\})$ and an expression that is the proposition "*What time does the next train to Paris leave?*"; ξ is a locus arbitrarily chosen on which this act is localized; this act offers a unique opening on which E may anchor her answer.
- dual actions $\kappa_2/\overline{\kappa_2} = (+/-, \xi.0, \emptyset)$ and an expression that is the proposition "*the next train to Paris leaves at 7:45 p.m.*" ; $\xi.0$ is the locus of this dialogue act, justified by κ_1; there is no opening created by this act as it is simply a given fact.
- an action $\kappa_3 = (+, \dagger)$ with the expression "*Thanks*"; with this dialogue act, T informs E that the dialogue went well and is finished.

Actions κ_1 and κ_2 are positive from the point of view of the locutor that produces them: positive for T for the first, for E for the second, and negative for the locutor that receives them. The action κ_3, positive, is produced by T. The interaction between the two (really simple) designs is depicted in the following figure. It converges as it ends with $\dagger$.



In the following subsections, we present first the approach by means of a simple dialogue extracted from a novel of C. Dickens. Then we illustrate specific forms of dialogues in terms of designs.

3.1 First Use in Dialogue

The following dialogue is extracted from the novel “David Copperfield” of C. Dickens.

Example 7. (C. Dickens, David Copperfield) The dialogue takes place between a coachman (**C**) and David (**D**), the coachman brings David to London:

C	I_1 :	You are going through, sir?
D	I_2 :	Yes, William. I am going to London. I shall go down into Suffolk afterwards.
C	I_3 :	Shooting, sir?
D	I_4 :	I don’t know.
C	I_5 :	Birds is got wery shy, I’m told
D	I_6 :	So I understand
C	I_7 :	Is Suffolk your county, sir?
D	I_8 :	Yes, Suffolk’s my county.
C	I_9 :	The dumplings is uncommon fine down there ...

As a first approximation, a dialogue act is associated with each utterance with actions $\kappa_1, \ldots, \kappa_9$ and respective espressions $I_1, \ldots, I_9$. The justification between actions is given by the figure on the right in example 1. The first action κ_1 is initial: with this dialogue act **C** initiates the dialogue. Actions κ_3, κ_7 and κ_9 are justified by the second intervention κ_2: in the three cases, the corresponding dialogue acts refer to ‘*Suffolk*’, entity present in the second intervention that may introduce topics concerning hunting, native soil, gastronomy, ...

Ludics allows for rebuilding the designs whose interaction produces as a trace this alternate sequence of interventions. The dialogue acts may be given with a positive polarity for the interventions of **C**: in that way, the dialogical interaction is represented with the point of view of **C**. Reversing the polarities, one gets the point of view of **D**. This produces two designs that interact and whose trace is the dialogue. In figures given in example 1, the design on the left is the point of view of **C**, and the design on the right is the point of view of **D**.

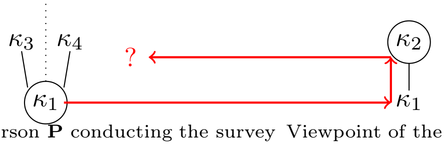
More generally, actions of Ludics are interpreted in terms of dialogue in the following way: a positive action in a design corresponds to an active role of the locutor when the design is her point of view. On the contrary, a negative action in the same design reports a passive role (*e.g.* receiving an intervention of her interlocutor). Viewing the dialogue as a trace makes explicit the fact that there are two points of view. Moreover it allows to observe the success or the failure of these two points of view, *i.e.* the fact that the dialogue may fail or end with a drop.

3.2 Divergent Dialogues, Presupposition in Dialogues

Dialogues may badly end because of misunderstandings, disagreements,... and it is necessary to be able to represent such situations. Ludics interaction distinguishes two cases. An interaction is convergent when it ends with a daimon, it is divergent when a positive focus has no dual counterpart. These two cases allow for interpreting two standard final situations in dialogues: either it finishes well or there is a misunderstanding between the two locutors. The following example is given by M. Chemillier [19] to illustrate the difficulties that arise when one wants to isolate the logical part of a dialogue in a field survey.

Example 8. A person **P** conducting a survey gives to a native the following informations: *All the Kpelle cultivate rice. Mister Smith does not cultivate rice.* The person **P** asks the following question to the native **N**:
- **P**: "Is Mister Smith a Kpelle?" (κ_1)
- **N**: "I do not know Mister Smith, I have never seen him." (κ_2)

Here we consider only the question and its answer that we represent each by a single dialogue act (resp. with actions κ_1 and κ_2). The person **P** expects a logically correct answer, *e.g. "No"*, with action κ_3. He plans also to receive an incorrect answer, *e.g. "Yes"* or *"It may be the case"*, with action κ_4. Hence the design for the person **P** may be either $\enclose{circle}{\kappa_1}$ followed by κ_3, or $\enclose{circle}{\kappa_1}$ followed by κ_4. The interaction in Ludics is given below: it is divergent as there is no negative action dual to κ_2 in what is expected in the design of **P**. The dialogue cannot continue.



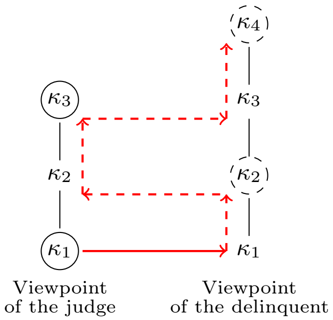
Viewpoint of the person **P** conducting the survey Viewpoint of the native **N**

The previous modeling is simplistic: a turn of speech may be more complex than a simple action. In fact, to each intervention may be associated a set of dialogue acts that complements the current design of the locutor. The result should still be a design. In particular, the first dialogue act should be anchored in some previous intervention: the dialogue continues normally if each person answers (in some way) a previous intervention. Furthermore, an intervention must be represented by a sequence of dialogue acts beginning and ending with a positive action: if the last action is not the daimon, the interlocutor may continue the dialogue. Dually, the design of the interlocutor is increased by what comes from the locutor. The result should be such that one of the chronicles ends with a negative action: the interlocutor initiates her own intervention by a dialogue act anchored on this negative action.

Dialogues with presuppositions are examples of such more complex situations. A presupposition is an implicit assertion concerning the world, whose validity is accepted in the dialogue.

Example 9. Let us consider this well known example due to Aristotle; a judge asks a young delinquent this question: *"Have you stopped beating your father?"*. Answering this question by '*Yes*' or '*No*' supposes that the answer of an implicit question is '*Yes*':
– *"Did you beat your father?"*
– *"Yes."*
– *"Have you stopped beating him?"*

The question asked by the judge, "*Have you stopped beating your father?*" displays three successive dialogue acts κ_1, κ_2 and κ_3 that refer respectively to interventions : "*Did you beat your father?*"; "*Yes*"; "*Have you stopped beating him?*". If the young adult continues the dialogue, he implicitly assumes that the interaction between the two (partial) designs do not diverge. Hence he accepts to justify his next intervention on the negative dialogue act dual to κ_3, this dialogue act being anchored on the positive dialogue act dual to κ_2: he assumes he would have answered "*Yes*" to the implicit question.



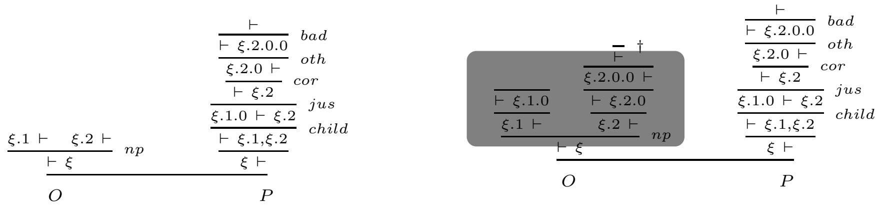
3.3 Argumentation

Argumentative dialogues show some peculiarities. In particular speech acts that compose them are limited in number: assertions, arguments, denial, concessions, . . . In addition, a notion of winnings appears, which seems specific to argumentative dialogues. In a dialogue whose aim is to exchange information, to share knowledge or a feeling, the question of whether one of the speakers wins is quite irrelevant. But to know who is right after a controversy is essential. An argumentative dialogue is distinguished from (common) dialogues in the sense that two arguments are opposed, each party having his thesis to defend or advance. The dialogue is then mainly an exchange of arguments and counter-arguments with the sole purpose that the dialogue ends when a thesis is considered winning. In *L'Art d'avoir toujours raison* [20], Schopenhauer means to define dialectics as "the art of winning controversies". Ideally, the easiest way to win a controversy is to have a strategy, winning against adversaries. This corresponds to a proof in Ludics. Several Schopenhauer's stratagems have been studied by Quatrini in [21], we reproduce below an example of *retorsio argumenti* or turning of the tables, by which your opponent's argument is turned against himself:

Example 1 *The opponent O declares, for instance, "So-and-so is a child, you must make allowance for him." The proponent P retorts, "Just because he is a child, I must correct him; otherwise he will persist in his bad habits."*

The controversy between P and O is analyzed in the following way:

- O justifies his implicit thesis (*"You might be wrong correcting this child"*) by two premisses: *"He is a child"* and *"you must make allowance for children"*. This is represented by a unique dialogue act with action $(+, \xi, \{1, 2\})$, and expression noted *np*. The ramification contains two elements as the argumentation has two premisses.
- P accepts this intervention with an action dual to the previous one. Then he replies: he explicitly concedes the first premisse and contradicts implicitly the second premisse (*"you must make allowance for him"* = *"you must correct him"*) by giving a counter-argument *"otherwise he will persist in his bad habits"*. This gives rise to several dialogue acts (see figure 1 on the left):
  - The first sentence: *"Just because he is a child"* contains a concession : $(+, \xi.1, \{0\})(-, \xi.1.0, \emptyset)$. The first action is expressed by *"he is a child"* (noted *child*) and the second action is expressed with the word *"Just"* (noted *jus*).
  - The intervention of P continues with a negation of the assertion of O. This is represented by an action $(+, \xi.2, \{0\})$ expressed with *"I must correct him"* (noted *cor*);
  - The counter-argumentation conveys two dialogue acts: the first one $(-, \xi.2.0, \{0\})$, expressed by *"otherwise"* (noted *oth*), lets P resuming his intervention, and the second one $(+, \xi.2.0.0, \emptyset)$ establishes his argument, expressed with *"he will persist in his bad habits"* (noté *bad*). The ramification of this last action is empty as this final argument cannot be contradicted.
- The situation after the opponent O accepts this intervention of P is depicted in figure 1 on the right, there are no more loci where O may continue the dialogue hence either he refuses it (divergence) or he "plays a daimon" (to keep the dialogue convergent). In any case, he looses.



Situation after the intervention of P

Situation after O accepts the intervention of P

Fig. 1. Schopenhauer - stratagem 26

4 Meaning in Ludics

Resting on the formalization of dialogues in Ludics as it is presented in the previous section, propositions have been done for modeling various aspects of Natural

Langage analysis, especially meaning. In this section we show how Ludics has been used for semantics and speech acts. Furthermore, we precise how inferential issues may be tackled with such a formalization. We illustrate these propositions by analysing further example 8 we reproduce below.

Example 10. (example 8 ctd)
- **P**: “All the Kpelle cultivate rice. Mister Smith does not cultivate rice. Is Mister Smith a Kpelle?”
- **N**: “I do not know Mister Smith, I have never seen him.”

4.1 Semantics of Utterances

In [1,4], Lecomte and Quatrini propose a conception of interactive meaning based on Ludics where the Ludics frame is at the same time used as a metaphor and as a formal device able to elaborate this metaphor. At a metaphoric level, a design being defined by its orthogonal[4], they postulate that the meaning of a sentence is given by its *dual* sentences. Moreover, they claim that Ludics offers also a framework to model the “meaning” of a sentence.

Precisely, the *dual* sentences of a given sentence u are utterances v which both:
- correctly interact with u: a speaker S claims the sentence u, her interlocutor expresses such an utterance v, and S is able to continue the dialogue.
- directly concern the contain of u: the utterance v is a question or a negation which may be considered as a test against u.

In this way, a set of designs is associated with the meaning of the sentence u: such designs are the carriers of the dialogues during which a speaker S affirms and justifies the statement u against an interlocutor.

Let us look at the utterance $\mathbb{E}$ and some of its dual sentences:

($\mathbb{E}$) “All the Kpelle cultivate rice.”

(a) “But Mr. Durand does not cultivate rice.”
(b) “Even the children?”
(c) “Was it the case twenty years ago?”

We consider the formalization in Ludics of one of these starting dialogues. Suppose that a speaker S claims the utterance $\mathbb{E}$ and her addressee answers saying (a). From the point of view of S, the first intervention is a positive dialogue act which corresponds to the assertion of $\mathbb{E}$ by S. The ramification of this dialogue act is a singleton: only one locus is created on which the interaction may continue (w.r.t. to this asserted sentence), and its expression is the utterance $\mathbb{E}$. The second intervention is represented also by a unique dialogue act. The speaker receives a counter-argument from her addressee. According to the view of S, this dialogue act is a negative one whose expression is the sentence (a). Its ramification contains two elements: the dialogue may continue questioning the fact that Mr. Durand is or is not a Kpelle, or the fact that he does or does not cultivate rice. When she asserts the utterance $\mathbb{E}$, the speaker has to be ready to receive such counter-arguments that concerns individuals about whom the addressee may

[4] According to the separation theorem.

argue that they are Kpelle, but that they do not cultivate rice. The design associated with the project of dialogue of the speaker S when she asserts $\mathbb{E}$ must contain such possibilities. It is represented in figure 2.

$$\dfrac{\dfrac{\begin{matrix} \vdots & \qquad & \vdots & \\ \vdash \xi.0.1_d, \xi.0.2_d & \qquad & \vdash \xi.0.1_{d'}, \xi.0.2_{d'} & \qquad \ldots \end{matrix}}{\xi.0 \vdash}}{\vdash \xi}$$

Fig. 2. The project of dialogue of a speaker asserting $\mathbb{E}$

After the dialogue act corresponding to the assertion of $\mathbb{E}$, this design contains as many branches as individual Kpelle (denoted by d, d' . . .). Moreover, for each such Kpelle d, the speaker has to be ready to continue the dialogue by giving some argument to justify that d cultivates rice: each branch indexed by d may have ulterior actions. In fact, there is not a unique such design, but a more precise specification of these designs should be relative to contextual considerations. In order to interact with other utterances, for example with questions like (b) or (c), designs would have some longer branches in order to explicit which Kpelle are concerned by the claim (for example the adults) or in order to precise the historical context of the fact.

Nevertheless, we may observe that all such designs share the same starting actions. Then, interpreting actions as logical operations, we may associate with these designs the beginning of a logical proof, represented in figure 3.

$$\dfrac{\dfrac{\begin{matrix} \vdots & \qquad & \vdots & \\ \vdash\downarrow K(d)^{\perp}, C(d) & \qquad & \vdash\downarrow K(d')^{\perp}, C(d') & \qquad \ldots \end{matrix}}{(\forall_x(\uparrow K(x) \multimap C(x)))^{\perp} \vdash}}{\vdash\downarrow (\forall_x(\uparrow K(x) \multimap C(x)))}$$

Fig. 3. A logical reading of the assertion of $\mathbb{E}$

Let us sum up the methodology that we sketched above. As "meaning" of an utterance, we associate with the utterance a set of designs, namely the designs which are the supports of dialogues that the claim of this utterance initiates. And we remark that, since these designs share their first actions, we may organize them in a set of designs built by means of operations of Ludics. Namely, we associate as meaning of $\mathbb{E}$ the set:

$$\mathbf{E} = \downarrow (\&_d(\uparrow \mathbf{K}(d) \multimap \mathbf{C}(d)))$$

where $\mathbf{K}(d)$ and $\mathbf{C}(d)$ are also sets of designs. In particular, provided that $\mathbf{K}(d)$ and $\mathbf{C}(d)$ are behaviours associated with logical formulas, the set $\mathbf{E}$ is a behaviour associated with a logical formula. In this way, we retrieve the semantical notion of a "logical form", thus also its nice properties: vericonditional semantics, quantifiers scope . . . , in the same time we retrieve a more proof theoretical approach of semantics, hence transposing to Natural Language semantics the following motto: *the meaning of a logical formula is the set of its proofs.*

At a first glance, this does not seem very different from the way formal semantics usually proceeds. Nevertheless, let us underline some points which are slightly different and new and which could favorably extend the usual models of semantics:

- Recall that formulas are not primitive objects in Ludics: interaction is first defined on designs. It may be relevant to associate as meaning of a sentence an object which is not *a priori* closed, *i.e.* a set of designs that is not necessarily a behaviour. It may also be relevant to associate designs that are not completely defined, *i.e.* that may be *more and more refined.* For example, the formula $K(d)$ may be either an atomic one but may also be decomposable, say in $k(d) \otimes A(d)$ to give an account that the Kpelle, the claim is about, are adults.
- Another result is that the logical interpretation of the quantifier '*All the*' is not *a priori* fixed. Depending on the context and on the kind of justifications of the claim $\mathbb{E}$, the quantifier may be either interpreted by a generalized additive conjunction in case the justification is relative to each individual, what was used to define $\mathbf{E}$, or interpreted by a universal first order quantifier, provided that the justifications are the same for each individual, *i.e.* do not depend on the individuals. We recall that, in a study on first-order quantifiers in Ludics [22], Fleury and Quatrini show that the family of designs associated with a first order universal quantifier has to satisfy a *uniformity* property: roughly speaking the designs should be in some way 'the same' (and should represent the same proof).

4.2 Speech Acts

The framework brought by the formalization of dialogues in Ludics has also been used to give an account for speech acts. In [2], Fleury and Tronçon transpose the characterization of speech acts given by Searle [23] in ludical terms. A speech act may be associated with a design which is able to interact with designs associated with the preconditions of this speech act. The normal form we get after interaction gives an account of the effects of the speech act. In [3], Lecomte and Quatrini characterize some figures of dialogue according to the form of their associated designs. In particular they focus on elementary speech acts. Let us recall the cases of assertions and interrogations.

As proposed by Walton in [24], **asserting** is "willing to defend the proposition that makes up the content of the assertion, if challenged to do so". This remark was the core for defining the meaning of an utterance (see previous subsection). When a speaker asserts some utterance, she must have in mind all justifications for predictable objections, hence the design associated with an assertion starts with a positive dialogue act, followed by a set of negative dialogue acts, each of them being the basis for justifications against predictable objections. Note that an assertion is not represented as a simple (logical) proposition. Indeed a design associated with a proposition takes into account just its logical behaviour, say an atomic proposition, a decomposable formula. Although the design associated with the assertion of this proposition should contain the fact of asserting and the proposition itself. This difference is pointed in figure 4 which gives the designs for '*Mr. Smith is/is not a Kpelle*'. Designs concerning the negative cases finish with a daimon to indicate that this position is given up: this enables to express a logical negation. Note finally that our representation is extremely simplified with respect to what could be a general assertion. We consider only the logical feature of utterances, and we suppose that its logical form is univocally fixed.

Propositions		Assertions	
$\dfrac{}{\vdash K(s)}\emptyset$	$\dfrac{\dfrac{\dfrac{}{\vdash}\dagger}{K(s)\vdash}\emptyset}{\vdash\downarrow K(s)^{\perp}}$	$\dfrac{\dfrac{\dfrac{}{\vdash K(s)}\emptyset}{P^{\perp}\vdash}}{\vdash\downarrow P}$	$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash}\dagger}{K(s)\vdash}\emptyset}{\vdash\downarrow K(s)^{\perp}}}{\downarrow P\vdash}}{\vdash\downarrow\uparrow P^{\perp}}$
is	**is not**	**is**	**is not**

Fig. 4. Proof-like designs associated with the propositions/assertions '*Mr. Smith is/is not a Kpelle*'.

Unlike other speech acts, it is relevant to associate with **questions** designs with more than one locus in the base. Indeed, viewed as a speech act, an interrogation consists in asking a question and being ready to *register* the answer. Thus it is necessary to associate with an interrogation a design with base $\vdash \tau, \sigma$, where τ is the locus of the question and σ is the locus where the answer is registered: such a design ends with a $\mathcal{F}ax$ so that the answer to the question is *moved to* σ when interacting with a dual design.

Let us consider for example the question: *"Is Mr. Smith a Kpelle?"*, that we denote by q. The design associated with this question is represented in figure 5. The speaker who asks this question is ready to receive three answers:
- either an assertion, say '*Mr. Smith is/is not a Kpelle*'. In such a case, she registers the answer: the fax after the action on the locus $\tau.0.1$ is dedicated to copying this assertion (from $\tau.0.1.0$ to σ).
- or the word '*Yes*'. In such a case, the speaker rebuilds the information: it uses the structure of the design for the proposition '*Mr. Smith is a Kpelle*' (figure 4).
- or the word '*No*'. This case is similar to the previous one: it uses the structure of the design for the proposition '*Mr. Smith is not a Kpelle*' (figure 4).

If the answer is '*Yes*', *i.e.* represented by the design $\dfrac{\dfrac{\tau.0.2\vdash}{\vdash\tau.0}}{\tau\vdash}$, the normal form of the net defined by the designs associated to the question and the answer is then

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{F}ax\\ \tau.0.1.0 \vdash \sigma\end{array}}{\vdash \tau.0.1, \sigma} \qquad \dfrac{}{\vdash \tau.0.2, \sigma}\emptyset \qquad \dfrac{\dfrac{\dfrac{}{\vdash \tau.0.3}\dagger}{\sigma.0 \vdash \tau.0.3}\emptyset}{\vdash \tau.0.3, \sigma}}{\tau.0 \vdash \sigma}}{\vdash \tau, \sigma}$$

Fig. 5. The design associated with the question q.

$\dfrac{}{\vdash \sigma}\emptyset$. In other words, the result is a design localized on σ that corresponds to the proof of the proposition '*Mr Smith is a Kpelle.*'

4.3 Inferences

Designs that represent independent parts of a dialogue or independent dialogues may be used in order to do inferences: when they form a net of designs, they may normalize. We illustrate this point by considering again example 8 where we suppose that the Native answers '*Yes*'. We focus first on the overall intervention of the investigator:

– **She provides a first information '***All the Kpelle cultivate rice***'.** This utterance is given as a **true** proposition. Even more, the investigator suggests to her addressee to use it in a logical reasoning. That is a full proof is provided by her intervention. This part of the intervention is naturally associated with the design corresponding to this proof. Proof and design are both represented below. For simplification issue, we consider an axiom rule (and a fax in the design) in place of a development of a justification for the fact that a Kpelle cultivates rice, in other words being a Kpelle implies cultivating rice and also having other properties that characterize a Kpelle: we abstract from these other properties hence we get an axiom.

$$\dfrac{\dfrac{\dfrac{}{K(d) \vdash C(d)}ax}{\vdash\downarrow K(d)^{\perp}, C(d)} \quad \dfrac{\dfrac{}{K(d') \vdash C(d')}ax}{\vdash\downarrow K(d')^{\perp}, C(d')} \quad \ldots}{\dfrac{(\forall_x(\uparrow K(x) \multimap C(x)))^{\perp} \vdash}{\vdash\downarrow (\forall_x(\uparrow K(x) \multimap C(x)))}} \qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{F}ax\\ \xi.0.1_d.0 \vdash \xi.0.2_d\end{array}}{\vdash \xi.0.1_d, \xi.0.2_d} \quad \dfrac{\begin{array}{c}\mathcal{F}ax\\ \xi.0.1_{d'}.0 \vdash \xi.0.2_{d'}\end{array}}{\vdash \xi.0.1_{d'}, \xi.0.2_{d'}} \quad \ldots}{\dfrac{\xi.0 \vdash}{\vdash \xi}}$$

- **She provides a second information: '***Mr. Smith does not cultivate rice***'.** This information is a priori independent from the previous one, hence is represented as another design:

$$\dfrac{\dfrac{}{\vdash}\dagger}{\beta \vdash}\emptyset \qquad\qquad \textit{i.e.}\text{ the proof:} \qquad\qquad \dfrac{\dfrac{}{\vdash}\dagger}{C(s) \vdash}\emptyset$$

- **She asks a question: '***Is Mr. Smith a Kpelle?***'.** This corresponds to the design given in the previous subsection in figure 5.

We have then four designs at disposal:

- the design associated to a proof of the proposition '*All the Kpelle cultivate rice*', based on $\vdash \xi$, that we will denote $\mathcal{D}_\xi$;
- the design associated to a proof of the proposition '*Mr Smith does not cultivate rice.*' based on $\beta \vdash$, that we will denote $\mathcal{D}_\beta$;
- the design noted $\mathcal{D}_\alpha$ obtained by means of a shift and a delocation (from σ to $\alpha.0$) of the design that results from the interaction of the question q and the answer '*Yes*' and which is associated to a proof of the proposition '*Mr Smith is a Kpelle.*' (end of last subsection). The design $\mathcal{D}_\alpha$ is then associated with a proof of $\downarrow K^\perp(s) \vdash$.
- The following design associated with a proof which enables one to perform the interaction between the proposition '*All the Kpelle cultivate rice*' and some propositions about one Kpelle, that we denote by $\mathcal{F}_{\xi,\alpha,\beta}$:

$$\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\mathcal{F}ax \\ \alpha.0 \vdash \xi.0.1_s.0\end{array}}{\vdash \xi.0.1_s.0, \alpha}}{\xi.0.1_s \vdash \alpha} \qquad \begin{array}{c}\mathcal{F}ax \\ \xi.0.2_s \vdash \beta\end{array}}{\vdash \xi.0, \alpha, \beta}}{\xi \vdash \alpha, \beta} \\
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{K(s) \vdash K(s)}}{\vdash K(s), \downarrow K^\perp(s)}}{\downarrow K^\perp(s) \vdash \downarrow K^\perp(s)} \qquad \dfrac{}{C(s) \vdash C(s)}}{\vdash \exists_x((\uparrow K(x))^\perp \otimes C(x), \downarrow K^\perp(s), C(s)}}{\downarrow (\forall_x(\uparrow K(x) \multimap C(x))) \vdash \downarrow K^\perp(s), C(s)}
\end{array}$$

Then, the cut-net $[\![\mathcal{F}_{\xi,\alpha,\beta}, \mathcal{D}_\xi, \mathcal{D}_\alpha, \mathcal{D}_\beta]\!]$ normalizes, its normal form is the design which encodes the logical contradiction $\dfrac{\dagger}{\vdash}$.

5 Conclusion

In the previous sections, we present Ludics and its current uses in modeling Natural Language. It should be clear that this is an ongoing research that changes largely the way Natural Language may be analyzed, as Ludics changes radically the point of view we may have on logics. In Ludics, the fundamental operation concerns interaction between objects called designs. Such objects may clearly be interpreted in a certain sense as "proofs". However, two main differences exist with what is generally called a proof that should be considered. First, a design may include daimon actions, this allows for considering proofs and counter-proofs in the same language. Second, a design may be infinite (depth as well as height), hence such an object may include enough information to interact with an infinite number of counter-objects. We recall also that Ludics is a rebuilding of Linear Logic: formulas may be denoted by closed sets of designs. In that way, one recovers standard concepts of logics, say truth, proof,...

If we summarize the modeling of language by means of Ludics, the most important point is that it is the interaction in dialogue (explicitly or not) that provides content to the language, and not a relation to some external reality, hence assuming an "inferentialist" position [25]. In the previous sections, we presented a few domains where this principle has begun to be applied, namely dialogue, speech acts, semantics. Works of Terui [17] about automata and formal language show that syntax may be also studied in that perspective. However, this approach has to be carried on to improve the propositions and to tackle other questions in Natural Language modeling.

References

1. Lecomte, A., Quatrini, M.: Ludics and its applications to natural language semantics. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS, vol. 5514, pp. 242–255. Springer, Heidelberg (2009)
2. Fleury, M.-R., Tronçon, S.: Speech Acts in Ludics. In: Lecomte, A., Tronçon, S. (eds.) PRELUDE 2010. LNCS (LNAI), vol. 6505, pp. 1–24. Springer, Heidelberg (2011)
3. Lecomte, A., Quatrini, M.: Figures of Dialogue: a View from Ludics. Synthese 183, 59–85 (2011)
4. Lecomte, A., Quatrini, M.: Pour une étude du langage via l'interaction: dialogues et sémantique en Ludique. Mathématiques et Sciences Humaines 189(1), 37–67 (2010)
5. Girard, J.Y.: Locus solum: From the rules of logic to the logic of rules. Mathematical Structures in Computer Science 11(3), 301–506 (2001)
6. Girard, J.Y.: Linear logic. Theor. Comput. Sci. 50, 1–102 (1987)
7. Fouqueré, C.: Ludics and Web: Another Reading of Standard Operations. In: Lecomte, A., Tronçon, S. (eds.) PRELUDE 2010. LNCS (LNAI), vol. 6505, pp. 58–77. Springer, Heidelberg (2011)
8. Girard, J.Y.: From foundations to ludics. Bulletin of Symbolic Logic 9(2), 131–168 (2003)
9. Girard, J.Y.: Le Point Aveugle: vers l'imperfection. Visions des Sciences, vol. 2. Hermann (2007)
10. Curien, P.L.: Introduction to linear logic and ludics, part i. CoRR abs/cs/0501035 (2005)
11. Curien, P.L.: Introduction to linear logic and ludics, part ii. CoRR abs/cs/0501039 (2005)
12. Basaldella, M., Faggian, C.: Ludics with repetitions (exponentials, interactive types and completeness). In: LICS, pp. 375–384. IEEE Computer Society (2009)
13. Curien, P.-L., Faggian, C.: L-Nets, Strategies and Proof-Nets. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 167–183. Springer, Heidelberg (2005)
14. Fouqueré, C., Mogbil, V.: Rewritings for polarized multiplicative and exponential proof structures. Electr. Notes Theor. Comput. Sci. 203(1), 109–121 (2008)
15. Andreoli, J.M.: Logic programming with focusing proofs in linear logic. J. Log. Comput. 2(3), 297–347 (1992)
16. Faggian, C., Maurel, F.: Ludics nets, a game model of concurrent interaction. In: LICS, pp. 376–385. IEEE Computer Society (2005)

17. Terui, K.: Computational ludics. Theor. Comput. Sci. 412(20), 2048–2071 (2011)
18. Landragin, F.: Vers l'identification et le traitement des actes de dialogue composites. In: Traitement Automatique du Langage Naturel (TALN), pp. 460–469 (2008)
19. Chemillier, M.: Eléments pour une ethnomathématique de l'awélé. Mathématiques et Sciences Humaines 181(Varia), 5–34 (2008)
20. Schopenhauer, A.: L'art d'avoir toujours raison. Circé (1830)
21. Quatrini, M.: In: Une relecture ludique des stratagèmes de Schopenhauer. Presses de la Sorbonne (to appear)
22. Fleury, M.R., Quatrini, M.: First order in ludics. Mathematical Structures in Computer Science 14(2), 189–213 (2004)
23. Searle, J.: Speech Acts. Cambridge University Press (1969)
24. Walton, D.: The place of dialogue theory in logic, computer science and communication studies. Synthese 123, 327–346 (2000)
25. Brandom, R.: Articulating Reasons: An Introduction to Inferentialism. Harvard University Press (2000)

The Non Cooperative Basis of Implicatures[*]

Nicholas Asher

IRIT, CNRS
nicholas.asher@irit.fr

Abstract. This paper presents and addresses a problem in pragmatics concerning the inference of implicatures within a Gricean framework. I propose a model in which implicatures are reasonable even in the absence of the sort of strong cooperativity supposed by Griceans.

1 Introduction

According to Grice [1975], conversation is a biproduct of rational behavior, to be analyzed in terms of beliefs, desires, and intentions. In addition, Grice makes specialized cognitive hypotheses about conversational agents—in particular that they are highly cooperative. Grice's conversational maxims of quantity quality and relevance encode this cooperativity in a highly informal fashion, but since the work of Cohen and Perrault [1979], Allen and Litman [1987], Grosz and Sidner [1990], Lochbaum [1998] and others, researchers have formalized these principles in terms of BDI (belief, desire, intention) logics.

There are two problems with this sort of formalization. The first is that propositional attitudes like belief, desire and intention are *private* attitudes, not common knowledge or even part of the mutual beliefs of dialogue agents. The link between what agents say or dialogue content and their private beliefs, preferences and intentions is much less robust than what Griceans and Neo-Griceans have postulated for content cooperative conversation. Any model of dialogue must infer information about mental states from the observed dialogue actions and *vice versa*. So we must *interpret* those actions—in other words, we must provide a representation of dialogue content and a procedure for constructing it during dialogue processing. The current mentalist approaches to dialogue content, couched within BDI logics, all equate dialogue interpretation with updating mental states: for instance interpreting an assertion that p and updating the model of the speaker's mental state to one that includes a belief in p are treated as equivalent. But they clearly are not equivalent in even in cooperative dialogue. If I am having a bad day, then my wife may say something to make me feel better even though she does not believe it.

With dialogue content separated from the agents' mental states, we need a term for what a speaker engages in when he or she makes a conversational move: following Hamblin [1987], say that a speaker makes a *public commitment* to some content—exactly what content he commits to depends on the nature of the

[*] Thanks to LACL 2012 reviewers for helpful comments.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 45–57, 2012.

speech act he's performed. In fact, Lascarides and Asher [2009] make speakers publicly commit to the *illocutionary effects* of their speech acts and not just to the locutionary content so as to accurately predict implicit agreement and denial.

In cooperative conversation, most of the time people say or commit to what they believe. So we can offer one informal precisification of some of Grice's maxims by appealing to defeasible generalizations like the following.

– Sincerity: Normally agents who commit to ϕ believe ϕ
– Quantity: say as much as you can say to achieve conversational goals.
– Competence: Normally if B believes that A believes that ϕ, then B should believe that ϕ
– Strong Competence: Normally if B believes that A doesn't believe ϕ, then B should not believe that ϕ
– Sincerity about Intentions: Normally if A publicly commits to the intention that ϕ, then A intends that ϕ
– Strong cooperativity : Normally if A publicly commits to the intention that ϕ, then B should intend that ϕ

Defeasible rules link various sorts of speech acts to intentions, beliefs and actions of their agents; for instance, if an agent asks a question, then he normally intends to know the answer to it.

2 Implicatures and the Problem

Such rules provide the basis of an account of implicatures, and *inter alia* scalar implicatures. Implicatures are defeasible inferences that involve the following problem: under what conditions can one reasonably infer from a speakers not committing to ϕ that he commits to $\neg\phi$? Consider (1).

(1) a. A: Did all of the students pass?
b. B: Some passed.

In (1) A does not commit to the claim that all of the students passed, and most speakers would reasonably infer that A in fact commits to the claim that not all the students passed. Here is a sketch of the kind of reasoning that one can adduce as a Gricean in favor of such an inference. Suppose a set of alternative moves, that the move chosen normally conforms to all the constraints above, and those that do not deviate from one of the constraints. Suppose also as given, either by discourse structure or by the lexicon, a set of alternatives for *some*, {some, all}. We can now sketch an informal derivation of the scalar implicature that B believes that not all the students passed.

– Sincerity: implies B believes his response to A's question. Competence implies that A should believe it.
– Cooperativity: B wants A to know an answer to his question—that either all the students passed or they didn't.

- So B's response should provide A an answer. (rationality)
- He didn't say all the students passed, which would have implied an answer.
- Choosing the alternative would not have violated Cooperativity (since it clearly provides an answer), so it must violate Sincerity.
- So B doesn't believe the all the students passed. And by Strong Competence, A shouldn't believe it either.

Rather than fully formalize this reasoning in a particular nonmonotonic logic,[1] let's step back and consider what this kind of approach does and doesn't do. First it requires a strong form of cooperativity—that interlocutors adopt each other's conversational goals and that speakers tell the truth. Second, it doesn't account for why B provides an "over-answer" to the question; with the implicature, B's answer not only provides a direct answer to A's question but tells him more by picking out a subset of the worlds that constitute the direct negative answer to the question. Neither these axioms nor any Gricean account of which I am aware provides an account of why didn't B just give a direct answer to A's question. It's clearly not a matter of the Gricean maxim of Quantity, since the over answer provides more information and is longer and more complex than a simple "No" answer! Yet over answers are very common in dialogue; for instance in the Verbmobil corpus Wahlster [2000], for instance, there is a far higher proportion of over answers than direct answers to questions. Are we following a maxim of being as informative as possible? If so, then people would never shut up (of course some people don't)!

None of these reflections show that the Gricean account of the implicature is wrong, only that it is incomplete. But the real problem is that when these defeasible generalizations don't apply, no implicatures should be drawn. Let me explain. Real conversations can have many purposes, not just information exchange. People talk to bargain, to bluff, to mislead, to show off or promote themselves, to put others down, to persuade others what they want them to do regardless of the facts. They often misdirect or conceal crucial information. In other words, conversation is often, even largely, non cooperative in the Gricean sense.

Consider the cross-examination in (2) of a defendant by a prosecutor, from Solan and Tiersma [2005] (we thank Chris Potts for pointing us to this example):

(2) a. Prosecutor: Do you have any bank accounts in Swiss banks, Mr. Bronston?
b. Bronston: No, sir.
c. Prosecutor: Have you ever?
d. Bronston: The company had an account there for about six months, in Zurich.

The locutionary content of (2d) is true. But Bronston succeeds in deflecting the prosecutor's enquiry by exploiting a misleading implicature, or what one might call a *misdirection*: (2d) implicates that Bronston never had any Swiss bank account and this is false.

[1] See Asher [2012] for details.

Misdirections can happen outside the courtroom too. Dialogue (3) occurred in a context where Janet and Justin are a couple, Justin is the jealous type, and Valentino is Janet's former boyfriend (from Chris Potts and Matthew Stone (pc)).

(3) a. Justin: Have you been seeing Valentino this past week?
b. Janet: Valentino has mononucleosis.

Janet's response implicates that she hasn't seen Valentino, whereas in fact Valentino has mononucleosis but she has seen him. Clearly, neither Janet nor Bronston are abiding by Gricean maxims: they're not trying to help their interlocutors achieve the intention behind their questions—to know an answer.

Gricean maxims also don't apply when a speaker simply opts out of quite basic conversational requirements. Consider dialogue (4) (from Chris Potts (pc)):

(4) a. Reporter: On a different subject is there a reason that the Senator won't say whether or not someone else bought some suits for him?
b. Sheehan: Rachel, the Senator has reported every gift he has ever received.
c. Reporter: That wasn't my question, Cullen.
d. Sheehan: The Senator has reported every gift he has ever received.
e. We are not going to respond to unnamed sources on a blog.
f. Reporter: So Senator Coleman's friend has not bought these suits for him? Is that correct?
g. Sheehan: The Senator has reported every gift he has ever received. (Sheehan says "The Senator has reported every gift he has ever received" seven more times in two minutes. `http://www.youtube.com/watch?v=VySnpLoaUrI`)

This is different from misdirection. Sheehan's utterances cannot be interpreted as implying an answer, and so contrary to Bronston's utterance (2d) this *exposes* that Sheehan hasn't adopted the reporter's intention. Dialogue (5) is another real life example of an 'opting out' move that happened to one of the authors in New York City:

(5) a. N: Excuse me. Could you tell me the time please?
b. B: Fuck you!

These examples show us two sorts of conversational strategies that fall outside completely outside the Gricean framework: misdirection and opting out. In misdirection the response is intended to thwart asker's goals, though the response *appears* cooperative. In opting out, no cooperative response is given.

These incompletenesses or silences on the part of Griceans concerning these conversational strategies are troubling. But there is worse ahead. Misdirections like that in (2) pose severe problems for extant accounts of scalar implicature that are based on strong cooperativity. To investigate this in detail, we need some some background. For one thing we must take into account the fact that there are different responses to questions, something which discourse theories like SDRT Asher and Lascarides [2003] have investigated. SDRT has different sorts of responses to questions. One is labelled *QAP*, or *Question-Answer-Pair*. $QAP(\pi_1, \pi_2)$ entails K_{π_2} is a true direct answer to the question K_{π_1} according to the compositional semantics of questions and answers. Another is called *IQAP* or *Indirect Question Answer Pair*. $IQAP(\pi_1, \pi_2)$ entails K_{π_2} defeasibly implies, via default rules that the questioner and respondent both believe, a direct answer to the question K_{π_1}. Moreover, *IQAP* entails that the answer is true.[2] This is the relation that holds between Bronston's response and the prosecutor's question. Bronston's response implies a direct answer via a quantity implicature.

We now come to the real problem for Gricean accounts. We all attach Bronston's answer with IQAP. The derivation of IQAP in SDRT is triggered simply by sentence mood. But its soundness as explained in Asher and Lascarides [2003] and the quantity implicature the IQAP is based on rely on cooperativity principles that are not sound in this scenario. Clearly, Bronston does not share the prosecutor's goal of finding out whether Bronston had an illegal bank account in Switzerland. But then how do we conclude IQAP? Are we all irrational? Or perhaps there is another type of derivation of the implicature given by IQAP.

There are two possible strategies to rescue the situation. We could argue that implicatures rely on Gricean cooperativity but have become fossilized. We might try to account for them as an "evolutionary adaptation: over repeated interactions where cooperativity is present, implicatures become automatic and thus are calculated even when the conditions of cooperativity that validate the implicatures are not present. While this is an appealing possibility to some, it is not so easy to provide a formal framework in which this intuition is borne out. Asher et al. [2001] attempt to model strongly cooperative principles of the sort mentioned above using evolutionary game theory. They show, however, that strong Gricean cooperative principles do not form an evolutionarily stable strategy unless rather strong initial assumptions are made. The other strategy is to search for an alternative foundation for implicatures. This is what I propose to do in the next sections.

3 The Model

I propose to look at our interpretation of Bronston's response from the perspective of game theory. Conversation involves moves that are calculated via an estimation of best return given what other participants say, and this is a natural setting for game theoretic analyses. We will assume that the meanings of all moves are fixed and look at the payoffs for different conversational strategies. A crucial feature of the model, however, is that payoffs are fixed not simply by coordination on meanings or interpretations (and as such this is not a type of signaling game but something different) but by effects of politeness, broadly speaking. Importantly, there are facets of language and linguistic usage not directly related to truth conditional content. According to Brown and Levinson [1978]'s strategic theory of politeness, language does not have the role merely to convey or ask for propositional content. Language also serves a second role in

[2] In SDRT terms, *IQAP* is right veridical.

negotiating the relationships between speakers and hearers, in particular what they call their "positive" and "negative" face. Positive face involves an agent's reputation and image from the perspective of his interlocutors, while negative face involves the agent's "distance" from his interlocutors, his freedom from constraints imposed by them on his possible actions. While these terms aren't precisely defined, they define relatively intuitive dimensions of an agent's social status in a community. Face is the medium throughwhich conversational participants recognize and negotiate their partner's potential status/ needs/ autonomy.

Following Asher and Quinley [2011], I use the notion of an exchange game, which is a formal model of two or more agents sending goods to one another. Moves are dialogue speech acts, and information and face are the goods exchanged. Asher and Quinley [2011]'s model is asymmetric because the speaker places his fate in the hands of the hearer when making a request, or asking a question. Such conversational moves place one participant in the position of asking another to do something for him—this something is the *speech act related goal* or SARG of the speaker's move. Thus, the exchange game I use is a variant of a trust game. Trust games depict a scenario where Player X has an initial option to defer to Player Y for a potentially larger payoff for both. Similar to the Prisoner's Dilemma, Player Y could defect on Player X and get a reward while Y fares badly. For a one-shot game, this act of deference will not occur for a rational Player X. However, reputation and observation effects and the possibility of repeated games make deference rational (Quinley 2011).

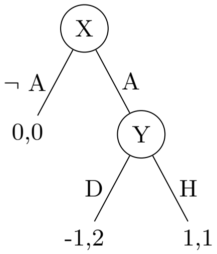


Fig. 1. Extensive Form

		Player Y	
		H	*D*
Player X	*A*	1;1	-1;2
	¬*A*	0;0	0;0

Fig. 2. Normal Form

Trust Games in Normal and Extensive Form: Player X has the option to Ask(A) Player Y for Help. Y can Help(H) or Defect(D).

The question is whether a conversation as I have conceived it is just a one shot game or can the conversational game be continued with new moves. Clearly, conversations are not just one shot games, though this is seldom recognized in formal game theoretic models. They are extended and dynamic, with an open ended sequence of conversational moves (though exactly the same move is almost never an option). There are natural endings to conversations but they have to do with a mutual agreement on facts, an exchange or that a disagreement exists with no resolution. It's not clear when this mutual agreement will take place. So reputation effects are *always* an issue in conversation. Discourse theories like SDRT model this flexibility of conversation: one can always attach to the discourse structure with new information.

While in principle any conversation may always be continued with further discourse moves, these moves have costs. They induce commitments by the speaker in the case of assertions; a speaker who asserts that p incurs the cost of potentially being challenged and having to defend his assertion. Not to do so leads to a loss of positive face. For questions and requests, the cost involves both a threat to the other's face (being too forward) and inviting a retaliatory attack on the speaker's reputation. Politeness theory following Brown and Levinson [1978] has studied the relative politeness of various types of speech acts, but these speech acts only characterize individual sentences. My proposal here is to look at the costs of *relational* speech acts, discourse moves that not only characterize the current utterance but affect the structure of the discourse context. A choice of a particular discourse move at stage m by participant i of an extensive game modeling a dialogue may make it very costly for a move of a certain type by participant j at $m+1$, effectively ending the conversation or turning it in a new direction. The reason for this has to do with already incurred costs. Suppose a speaker i makes a move that involves a particular SARG with a certain cost. Costs of turns by i that continue to develop or help realize that SARG, once such a development is started but not completed, are intuitively lower than the cost of turns that incur a new SARG, *ceteris paribus*. This will be a key feature in accounting for implicatures.

3.1 Questions and Their Responses in the Model

The next thing to specify is how to model questions and their answers. I understand questions as a dynamic operation on an information state, following the outlines of SDRT. The input information state for a question is a set of sets of possibilities, and a question's semantic effect on this set of possibilities is to introduce further structure to this set of sets by regrouping the elements of those sets into possibly overlapping subsets, where each one of the subsets corresponds to a direct answer of the question. The linguistically encoded continuations are: eliminate some of the subsets by providing a direct answer or indirect answer (which implicates a direct answer), leave the structure as it is either by doing nothing or with a statement to the effect that the addressee is not in a position to provide any information, or ask a follow up question.

Let's now look at the costs of questions and their responses, in particular the face threatening or saving nature of responses to questions. To make it concrete let us investigate the details of the conversation between Bronston (B) and the prosecutor (P). Let us assume that B does not wish to converse with P and does not, in particular, does not want to dwell on the topic of his bank accounts. If B gives an obvious non answer, he doesn't even commit to the question or address P's SARG to get an answer to his question. He affronts P's face, with potential retaliation and an unpleasant discourse move in subsequent turns, perhaps forcing him under oath to perjure himself or to admit damaging information. But this is rational if B were playing a one shot game (this is akin to the defect move in the Prisoner's Dilemma). If B responds with *QAP*, he does address P's SARG, at least as P has so far developed it. But B opens himself up to an explicit admission of guilt or explicit commitment to something purjureable. An *IQAP* answer that supplies additional information besides a direct answer, i.e. an IQAP that is an over answer, is more polite and increases the positive face of P. As such it is a lower cost move for B. More importantly, *IQAP* also increases probability of no further negotations on P's SARG, as the added information supplied in the *IQAP* anticipates follow up questions, answering them and so providing a more complete closure with respect to the questioner's SARG. This also increases the positive face of the interlocutor, making the move less costly. But, and importantly in this case, *IQAP* makes a continuation on the same topic by *P* more costly, because it forces him to introduce a new SARG or take the costly step of saying that his interlocutor hasn't answered the question (this is a direct attack on B's face and carries with it reputation effects). As it is in B's interest to avoid further questioning on this topic in particular, *IQAP* is the dominating strategy for him. If B answers *IQAP*, he avoids the potential face-threat and the politeness looks good to a judge and jury too. For P, *IQAP* is also an acceptable move by B to his question, and reputation effects make it less costly for him to accept it. Notice though that *QAP* is also an acceptable move for *P*: B gives him the information he was seeking and in a way that attends to P's reputation. B also gives additional information anticipating follow up questions, and this is information that could be of value to P. So it is in the interest of all to take a discourse move that is not a direct answer to be an indirect answer. A trust game model for conversation implies dominance of IQAP over QAP in most situations, but in this case in particular.

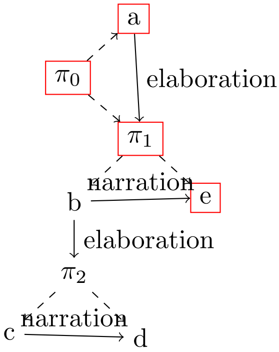
3.2 Complex Structures in Discourse and Costs of Discourse Moves

There is a close connection between SARG satisfaction and discourse structure in dialogue. Roughly, a move that satisfies a previously unsatisfied SARG forces a discourse "pop"; new material is no longer attached locally but to some higher constituent. Higher attachments incur new SARGs and in general incur higher costs, unless they are discourse closing moves or acknowledgments of a previous move or moves. By looking at discourse structure, we can examine in more detail at how *IQAP*, other discourse moves and their possible continuations have different costs.

To make these assumptions explicit, we need to clarify some findings about discourse structure in the literature. Discourse structures are graphs, where the nodes are discourse units and the arcs represent links between discourse units that are labelled with discourse relations. Discourse constituents may be elementary or complex. Elementary discourse units (EDUs), the atomic elements of a discourse structure, which correspond typically to clauses but also sub sentential constituents like appositions, non restrictive relative clauses *inter alia* [Afantenos et al., 2012], may be linked together via (one or more) discourse relations and form complex discourse units (CDUs) that are themselves arguments of discourse relations. CDUs in the ANNODIS corpus come in all sizes but the majority are relatively small (less than 10 EDUs in total [Asher et al., 2011]). In SDRT discourse graphs have a recursive structure with two sorts of edges in order to represent CDUs, one for the discourse relations and one to encode the relation between CDUs and their constituents. Consider the figure below for (6).

(6) a. Max had a great evening last night.
b. He had a great meal.
c. He ate salmon.
d. He devoured lots of cheese.
e. He then won a dancing competition.

Here is the discourse structure:



Discourse structure for texts, in particular the presence of CDUs, have interpretive effects. For example, the event described in (6e) comes after the events in (6b,c,d), as detailed in Asher and Lascarides [2003]. CDUs allow us to give a discourse relation scope over several EDUs, which is especially useful in cases where the relation cannot be "factored" or distributed over the constituents inside the CDU. This occurs for right arguments with relations like Explanation:

(7) James is sick. [He drank too much last night and he smoked too much.]

The part in brackets describes a CDU with two EDUs both of which contribute to an explanation of why James is sick. But we cannot distribute this explanation across the constituent EDUs; both EDUs contribute to cause James's sickness but neither one might be sufficient to cause the sickness on its own.

CDUs are also important for dialogue. The opening and closure of CDUs, or their boundaries, have to do with SARGs of conversational turns. Many dialogue actions, like asking a question for example, are defeasibly associated with an associated speech act goal—for instance, the asking of a question is associated with the goal of knowing the answer of the question. However, the SARG involve more information than just a direct answer to an explicit question. A SARG development provides the grounds of a single local dialogue structure, which consists of a head or superordinate element, which gives rise to the SARG together with a subordinate part, that develops the SARG. If a question is an opening move in a CDU, the closure of that CDU will occur when an answer goal of that question is either satisfied or known not to be satisfiable and related follow up questions have similarly been answered or are known not to be answerable.

An IQAP response to a question like that in (2) answers the question but also typically provides an “over answer”. An *IAQP* answer provides a higher probability termination of that discourse structure: the SARG underlying the original question is satisfied and follow up questions are anticipated. Continuations in this situation are more likely to be on the whole structure. For example, if P pursues this line of questioning in (**??**) it will be with a higher cost move—e.g. ”challenge” or a request for elaboration. (**??**e’) is a simple example of an elaboration

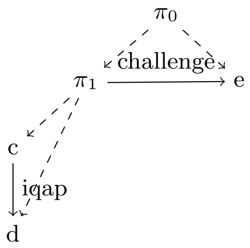
(2e’) Can you elaborate?

which sounds silly, given the extended answer. A challenge would be something like

(2e) Would you please answer the question, yes or no.

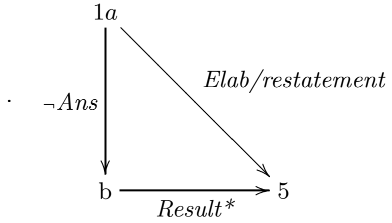
Either one of these moves is a higher cost move. IQAP thus raises the probability of a move by P to another topic or to exit the conversation.

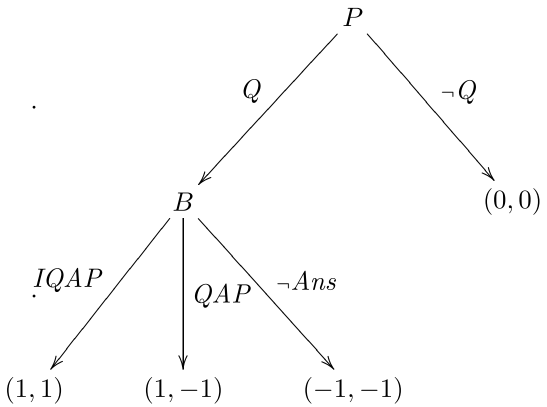
Here’s a picture of the IQAP scenario:



Let’s now look at the alternative moves, either QAP or an obvious non-answer. On the other hand, the QAP response is dispreferred for strategic reasons by *B*. A short answer QAP invites follow up questions, and so makes it easy for *P* to stay on this topic and get more information. It also leads to an explicit admission of guilt in this case or clear perjury. For B, a non answer, which I label here with ¬ *And*, is a good one shot move, but in an extended game with further moves, it invites a low cost restatement of the question (since the SARG is not satisfied). It also invites a retaliation since it does not address the SARG of the questioner and so is attack on his positive fact. The low cost move by *P* looks like this:

Here is the game tree for the different moves of B, where the costs of the different moves are motivated by the discussion above. I abstract from details and hypothesize three different discourse actions of B. The utilities provided are motivated by the preceding discussion:





The game over responses to questions is sub-tree compatible with a sequential trust game McCabe et al. [2003], as is its solution concept. While IQAP and Not Answer are equally rational for B in a one shot game, Not Answer puts P at a disadvantage. This disadvantage may lead to an unpleasant conversational turn, and reiteration of the question; the utilities on the ¬ *Ans* reflect this. In this case QAP is dispreferred by Bronston for reasons having to do with extra linguistic issues (the problem of explicit perjury); notice though that P is indifferent between QAP and IQAP since they both satisfy the SARG underlying the question. Whether he *should* be indifferent is another matter. Asher and Lascarides [forthcoming] argue that P should be indifferent between QAP and IQAP only if IQAP remains an equilibrium in a larger game arena, in which facts about B's player type that might distinguish between these IQAP and QAP are taken into account in the interpretation of B's response to the question. But I will not go into the lengthy discussion that this engenders here.

4 Back to Implicatures

So far, I've developed a game theoretic model based on politeness and on assumptions of costs of continuations of certain discourse structures. I've shown that it's reasonable to suppose that certain kinds of responses to questions are preferred in non cooperative conversations—in effect over answers to questions are preferred for strategic reasons. Thus, the account fills a gap in the Gricean account. But what about our puzzle about implicatures in non cooperative contexts?

Given the model, IQAP is strategically favored as a response. When the move doesn't entail a direct answer, we have to engage in defeasible reasoning to get a direct answer. Sometimes this reasoning depends on a set of alternatives generated lexically or by the discourse context (see Asher [2012] for a discussion of this issue and a proposal). The counterfactual reasoning goes as follows for P. B would know whether he had a bank account and so, given this presumption, would have said so; this would have been a natural and relevant issue to include in an IQAP. The IQAP move is designed to anticipate follow up questions, and a natural one in this case would be the question of whether Bronston himself had a bank account. In fact, it's the question that P asked! P can reasonably assume that since B doesn't want the questioning to go on, he says all that is relevant to P's question—he is anticipating follow-up questions. Since B didn't say that he had a bank account, he commits to not having one, given the type of discourse turn. So in this case $\neg$ Commit(ϕ) $>$ Commit$\neg\phi$. The scalar inference to the conclusion that Bronston doesn't have a bank account can be justified without appealing to any theses about cooperativity. We've substituted utility of IQAP and its semantics for cooperativity to get a similar effect.

Nevertheless, misdirection is still possible. P's assumption though reasonable is not sound in this case. The problem lies, as I intimated above in the interpretation of B's signal. Is it really an *IQAP* or is it in fact an evasion? Normal speakers do anticipate follow up questions, especially ones directly relevant to the issue. Were Bronston a normal uncooperative speaker, the inference to IQAP and the commitment to not having a bank account would have been sound. That is, B would in fact commit to not having a bank account (note that the question as to whether he commits to not having a bank account is a *very* different question from whether this information is credible). But B in fact did claim that he did not *say* that he had a bank account. He was just giving some background information about the bank and his firm. So P in fact misinterpreted him. course in this case the prosecutor P is also at fault as Asher and Lascarides [forthcoming] argue. He should have realized that Bronston's commitment here is in some sense less strong than one based just on compositional semantics; it relies on reasoning about normal speakers and about discourse moves, which are ambiguously signalled. He should have realized that Bronston might try to weasel out of his commitment, and the attendant charge of perjury. In fact, this is what happened.

5 Conclusions

I proposed a model and an argument for supporting implicatures without assumptions about Gricean cooperativity. The model also explains why over answers are so frequent. The model makes clear on a hidden reputation effect that is constant in extended conversation. The foundation of this model rests on ideas from politeness theory. And thus face and reputation emerge as important factors in the evolution of discourse structure for conversation. Which perhaps points to a new role for expressive meaning.

References

Afantenos, S., Asher, N., Benamara, F., Bras, M., Fabre, C., Ho-dac, M., Le Draoulec, A., Muller, P., Pery-Woodley, M., Prevot, L., Rebeyrolle, J., Tanguy, L., Vergez-Couret, M., Vieu, L.: An empirical resource for discovering cognitive principles of discourse organisation: the annodis corpus. In: Proceedings of LREC 2012 (2012)

Allen, J., Litman, D.: A plan recognition model for subdialogues in conversations. Cognitive Science 11(2), 163–200 (1987)

Asher, N., Lascarides, A.: Logics of Conversation. Cambridge University Press (2003)

Asher, N., Lascarides, A.: Strategic conversation. Draft copy available from the authors (forthcoming)

Asher, N., Quinley, J.: Begging questions, their answers and basic cooperativity. In: Proceedings of the 8th International Conference on Logic and Engineering of Natural Language Semantics (LENLS), Japan, (2011)

Asher, N., Sher, I., Williams, M.: Game theoretic foundations for pragmatic defaults. In: Amsterdam Formal Semantics Colloquium, Amsterdam (December 2001)

Asher, N.: Implicatures in discourse. Lingua (2012) (forthcoming)

Asher, N., Venant, A., Muller, P., Afantenos, S.D.: Complex discourse units and their semantics. In: Contstraints in Discourse (CID 2011), Agay-Roches Rouges, France (2011)

Brown, P., Levinson, S.C.: Politeness: Some Universals and Language Usage. Cambridge University Press (1978)

Cohen, P.R., Raymond Perrault, C.: Elements of a plan-based theory of speech acts. Cognitive Science 3, 177–212 (1979)

Grice, H.P.: Logic and conversation. In: Cole, P., Morgan, J.L. (eds.) Syntax and Semantics. Speech Acts, vol. 3, pp. 41–58. Academic Press (1975)

Grosz, B., Sidner, C.: Plans for discourse. In: Morgan, J., Cohen, P.R., Pollack, M. (eds.) Intentions in Communication, pp. 365–388. MIT Press (1990)

Hamblin, C.: Imperatives. Blackwells (1987)

Lascarides, A., Asher, N.: Agreement, disputes and commitment in dialogue. Journal of Semantics 26(2), 109–158 (2009)

Lochbaum, K.E.: A collaborative planning model of intentional structure. Computational Linguistics 24(4), 525–572 (1998)

McCabe, K., Rigdon, M., Smith, V.: Positive reciprocity and intentions in trust games. Journal of Economic Behavior & Organization 53 (2003)

Solan, L.M., Tiersma, P.M.: Speaking of Crime: The Language of Criminal Justice. University of Chicago Press, Chicago (2005)

Wahlster, W. (ed.): Verbmobil: Foundations of Speech-to-Speech Translation. Springer (2000)

Movement-Generalized Minimalist Grammars

Thomas Graf

Department of Linguistics,
University of California, Los Angeles
tgraf@ucla.edu
http://tgraf.bol.ucla.edu

Abstract. A general framework is presented that allows for Minimalist grammars to use arbitrary movement operations under the proviso that they are all definable by monadic second-order formulas over derivation trees. Lowering, sidewards movement, and clustering, among others, are the result of instantiating the parameters of this framework in a certain way. Even though weak generative capacity is not increased, strong generative capacity may change depending on the available movement types. Notably, TAG-style tree adjunction can be emulated by a special type of lowering movement.

Keywords: Minimalist Grammars, Movement, Monadic Second-Order Logic, Tree Languages, Transductions, Tree Adjunction Grammar.

Introduction

Ever since Joshi's conjecture that natural language is mildly context-sensitive [8], a lot of research has been devoted to characterizing this class in various ways. One of them pertains to multiple context-free languages (MCFLs; [18]) and states that they coincide with the string yield of the class of tree languages that are the image of regular tree languages under tree-to-tree transductions definable in monadic second-order logic (MSO; see [14] and the literature cited there). This result meshes well with recent approaches that decompose Minimalist grammars (MGs) — which have the same weak generative capacity as MCFLs — into an MSO-definable (= regular) tree language L and a transduction from L to the intended phrase structure trees ([12, 14] and references therein).

From a linguistic perspective, an MG's set of well-formed derivation trees provides the most natural encoding of this underlying tree language L, and [12] demonstrated that this is indeed a workable solution. In [4], however, it is shown that recognizing Minimalist derivation trees does not require the full power of MSO. In a sense, then, MGs still have some wiggle room insofar as one can increase the complexity of their derivation trees and still stay inside the confines of MSO-definability that limit the formalism to MCFLs. One way to exploit this gap is by adding MSO-definable constraints to MGs. Even though this greatly increases their linguistic usefulness, weak and strong generative capacity remain the same [3, 11]. I explore another option in this paper: allowing for derivationally more complex variants of Move, yielding Movement-Generalized MGs (MGMGs).

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 58–73, 2012.

My endeavour starts with another insight of [4], namely that the distribution of Move nodes in a derivation tree can be regulated by a few simple constraints stated in terms of proper dominance. To create new movement types, for instance sidewards movement [7, 15], one merely has to replace proper dominance by some other binary relation R. As long as R is MSO-definable, the derivation tree language will still be regular and weak generative capacity does not increase. Some parameters of Move, though, must be expressed directly in the mapping from derivation trees to derived trees. Fortunately, all of them are MSO-definable and thus pose no risk of taking us out of the class of MCFLs. The result is a general, mildly context-sensitive framework that accommodates almost all aspects of Move: directionality (raising, lowering, sideward), size of the moved constituent (head, phrase, pied-piped phrase), overt versus covert, and linearization (left or right specifier).

The paper is laid out as follows. After a few technical preliminaries, I define standard MGs in Sec. 2, focusing foremost on the constraints that ensure the well-formedness of Minimalist derivation trees. I then proceed with generalizing Move; Sec. 3.2 stays at the level of derivation trees, whereas Sec. 3.3 and 3.4 are devoted to transduction parameters. The final definition of MGMGs is given in Sec. 3.5. In the last section, I analyze the relationship between TAGs and MGs with lowering, conjecturing that their derived tree languages are identical given certain assumptions.

1 Preliminaries and Notation

Let Σ and Γ be alphabets. A *directed graph* with labeled nodes and edges over (Σ, Γ) is a triple $G(\Sigma, \Gamma) := \langle V, E, \ell \rangle$, with V a finite set of nodes, $E \subseteq V \times \Gamma \times V$ the set of labeled edges, and $\ell : V \to \Sigma$ the node labeling function. An edge $\langle u, \gamma, v \rangle$ is an edge from u to v with label γ; it is an outgoing edge of u and an incoming edge of v. In this case, u is called a *mother of* v, or equivalently, v is a *daughter of* u. A *path* from u to v is a (possibly empty) sequence of nodes $u_0 \cdots u_n$ such that $u = u_0$, $v = u_n$ and u_i is a mother of u_{i+1} for all $0 \leq i \leq n$. A path is a *cycle* iff $u_0 = u_n$ and $n \geq 1$. A graph is *cycle-free* iff it contains no cycles. A node with no incoming edges is a *root*, a node with no outgoing edges a *leaf*. A graph is *rooted* iff it has exactly one root.

Let Σ be a *ranked* alphabet, i.e. every $\sigma \in \Sigma$ has a unique non-negative *rank*; $\Sigma^{(n)}$ is the set of all n-ary symbols in Σ. A Σ*-term graph* is a cycle-free rooted graph $G(\Sigma, \Gamma)$ such that Σ is a ranked alphabet, $\Gamma := \{i \mid 1 \leq i \leq n$ and n the largest integer such that $\Sigma^{(n)} \neq \emptyset\}$ and every node with label $\sigma \in \Sigma$ of rank i has i outgoing edges with pairwise distinct labels. The integers on the outgoing edges of a node are interpreted as linear order. A Σ*-tree* is a Σ-term graph in which every node except the root has exactly one incoming edge. Let $\Pi^n := \{\Box_i \mid 0 < i \leq n\}$ be a set of distinguished nullary symbols called ports. A (Σ, n)*-context* is a $(\Sigma \cup \Pi^n)$-tree such that all ports have pairwise distinct indices. Given a (Σ, n)-context C and a sequence $s := t_1, \ldots, t_n$ of (Σ, m)-contexts, $m \in \mathbb{N}$, the *n-fold tree concatenation* of C and s replaces each $\Box_i$ in C (if it exists) by t_i.

My definition of MSO transductions follows [1] very closely. I assume that the reader is already familiar with monadic second-order logic (MSO) and write $\mathrm{MSO}(\Sigma, \Gamma)$ to denote the MSO language of (Σ, Γ)-graphs. A *finite-copying MSO graph transducer* from (Σ_1, Γ_1) to (Σ_2, Γ_2) is a triple $\mathrm{MSO}_{\mathrm{gr}} := \langle C, \Psi, \Theta \rangle$, where C is a finite set of copy names, $\Psi := \{\psi_{\sigma,c}(x) \in \mathrm{MSO}(\Sigma_1, \Gamma_1) \mid \sigma \in \Sigma_2, c \in C\}$ a set of node formulas, and $\Theta := \{\theta_{\gamma,c,c'}(x, y) \in \mathrm{MSO}(\Sigma_1, \Gamma_1) \mid \gamma \in \Gamma_2, c, c' \in C\}$ a set of edge formulas.

The *graph transduction* τ *defined by* $\mathrm{MSO}_{\mathrm{gr}}$ is as follows. For every graph $G(\Sigma_1, \Gamma_1)$, its image under τ is $G'(\Sigma_2, \Gamma_2)$ such that

- $V_{G'} := \{\langle c, u \rangle \mid c \in C, u \in V_G,$ and $G, u \models \psi_{\sigma,c}(x)$ for exactly one $\sigma \in \Sigma_2\}$,
- $E_{G'} := \{\langle \langle c, u \rangle, \gamma, \langle c', u' \rangle \rangle \mid \langle c, u \rangle, \langle c', u' \rangle \in V_{G'}, \gamma \in \Gamma_2$ and $G, u, u' \models \theta_{\gamma,c,c'}(x, y)\}$,
- $\ell_{G'} := \{\langle \langle c, u \rangle, \sigma \rangle \mid \langle c, u \rangle \in V_{G'}, \sigma \in \Sigma_2,$ and $G, u \models \psi_{\sigma,c}(x)\}$.

An *MSO term graph transducer* is a graph transducer from trees to term graphs. An *MSO tree transducer* is a graph transducer from trees to trees. Unless a transducer is explicitly designated to be finite-copying, C is assumed to be a singleton and thus superfluous.

2 Minimalist Grammars

The material covered in this paper presupposes a high level of familiarity with MGs. Unfortunately, space restrictions force me to proceed at a brisk pace, so that readers unacquainted with the formalism must be referred to [21] for a gentle introduction.

While MGs are usually defined in terms of the derived trees they generate [21] or in the chain-based format of [22], it makes more sense for our purposes to define them via Minimalist derivation tree languages (MDTLs). To this end, I adopt the approach taken by [4], which builds on the notion of *slices* (introduced in [4] and [11]). The slice of a lexical item (LI) l consists of l itself and those interior nodes which denote an operation checking a licensor or selector feature of l. Intuitively, then, the slice of l is the derivation tree equivalent of the phrase projected by l in the derived tree (cf. Fig. 1). Since every node in a well-formed derivation tree belongs to exactly one slice, MDTLs can be regarded as the result of combining a finite number of slices in all possible ways such that all conditions imposed by the feature calculus are obeyed. Consequently, every MG is fully specified by some finite set of slices. Slices can be obtained from LIs via a simple recursive procedure.

Definition 1. *Let* Base *be a non-empty, finite set of* feature names. *Furthermore,* Op := $\{merge, move\}$ *and* Polarity := $\{+, -\}$ *are the sets of* operations *and* polarities*, respectively. A* feature system *is a non-empty set* $Feat \subseteq$ Base × Op × Polarity.

Note that this is merely a different notation for the familiar system of *category features* $f := \langle f, merge, - \rangle$, *selector features* $=f := \langle f, merge, + \rangle$, *licensee features* $-f := \langle f, move, - \rangle$, and *licensor features* $+f := \langle f, move, + \rangle$. In cases

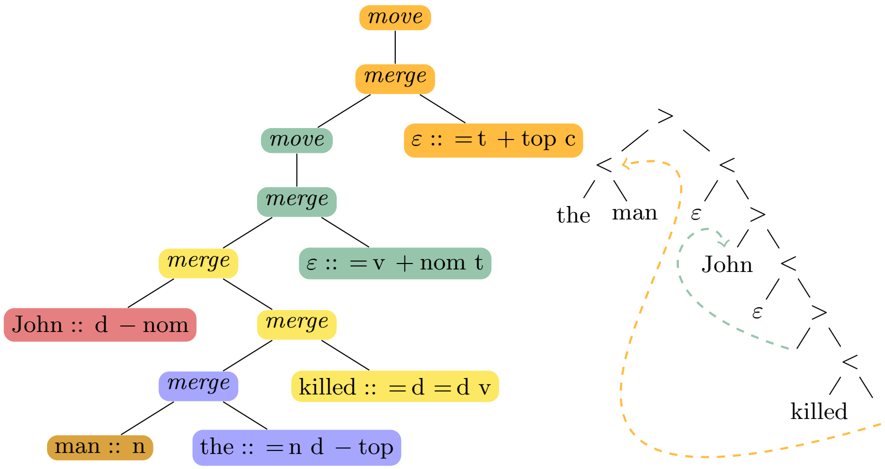


Fig. 1. Left: derivation tree of *The man, John killed*, with slices indicated by color; Right: corresponding derived tree, dashed arrows indicate movement

where only the name, operation, or polarity of f is of interest, $\nu(f)$, $\omega(f)$ and $\pi(f)$ will be used, respectively.

Definition 2. *Given a string alphabet Σ and feature system Feat, a $(\Sigma, Feat)$-* lexicon *is a finite subset of $\Sigma \times \{::\} \times Feat^*$.*

Definition 3. *Let Lex be a $(\Sigma, Feat)$-lexicon, $Lex_\star := \{\sigma :: f_1 \cdots f_n\star \mid \sigma :: f_1 \cdots f_n \in Lex\}$, and Ω the ranked alphabet $\{l^{(0)} \mid l \in Lex\} \cup \{move^{(1)}, merge^{(2)}\}$. Then the* slice lexicon of *Lex is* slice$(Lex) := \{\zeta(l) \mid l \in Lex_\star\}$, *where $\zeta : Lex_\star \to T_\Omega$ is given by*

$$\zeta(\sigma :: f_1 \cdots f_i \star f_{i+1} \cdots f_n) := \begin{cases} \sigma :: f_1 \cdots f_n & \\ \qquad\qquad \textit{if } f_1 \cdots f_i = \varepsilon & \\ \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n) & \\ \qquad\qquad \textit{if } \pi(f_i) = - & \\ move(\zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) & \\ \qquad\qquad \textit{if } \tau(f_i) = move \textit{ and } \pi(f_i) = + & \\ merge(\Box_i, \zeta(\sigma :: f_1 \cdots f_{i-1} \star f_i \cdots f_n)) & \\ \qquad\qquad \textit{if } \tau(f_i) = merge \textit{ and } \pi(f_i) = + & \end{cases}$$

I follow [4] in stipulating that slices are right branching, but this is merely a matter of convenience — linear order is irrelevant in derivation trees.

Despite its totality, ζ yields the intended result only for LIs of the form $\gamma c \delta$, where γ is a (possibly empty) string of selector and licensor features, c a category feature, and δ a (possibly empty) string of licensee features. As was shown in both [3] and [11], all LIs occurring in a well-formed derivation satisfy this condition.

In anticipation of subsequent modifications of the formalism, though, I explicitly require this feature order.

F-Order. Every LI is an element of $\Sigma \times \{::\} \times \{f \mid \pi(f) = +\}^* \times \{c \mid \omega(f) = \mathit{merge}, \pi(f) = -\} \times \{f \mid \omega(f) = \mathit{move}, \pi(f) = -\}^*$.

Given a slice lexicon slice(*Lex*), let $|\gamma|$ be the maximum of positive polarity features on a single LI. The closure of slice(*Lex*) under $|\gamma|$-fold tree concatenation is the *free slice language* FSL(slice(*Lex*)). Obviously this is not a well-formed MDTL (for instance, some trees still contain ports). Hence certain constraints must be enforced, which in turn requires additional terminology.

The *slice root* of LI $l := \sigma :: f_1 \cdots f_n$ is the unique node of $\zeta(l)$ reflexively dominating every node in $\zeta(l)$. An interior node of $\zeta(l)$ is *associated to feature* f_i *on* l iff it is the root of $\zeta(\sigma :: f_1 \cdots f_i \star f_{i+1} \cdots f_n)$. Two features f and g *match* iff they have identical names and operations but opposite feature polarities. An interior node m matches a feature g iff the feature m is associated to matches g. For every $t \in$ FSL(slice(*Lex*)) and LI l in t with string $-f_1 \cdots - f_n$ of licensee features, the *occurrences* of l in t are defined as follows:

- $\mathit{occ}_0(l)$ is the mother of the slice root of l in t (if it exists).
- $\mathit{occ}_i(l)$ is the unique node m of t labeled *move* such that m matches $-f_i$, properly dominates occ_{i-1}, and there is no node n in t that matches $-f_i$, properly dominates occ_{i-1}, and is properly dominated by m.

I also refer to occ_0(l) as the *zero occurrence of* l, while all other occurrences of l are *positive occurrences*. The i^{th} positive occurrence of l is simply the first Move node m one encounters when moving upwards through the derivation tree such that m is capable of checking the i^{th} licensee feature f_i of l after all preceding licensee features have already been checked. Given a well-formed derivation, m will always coincide with the node that actually checks f_i. In other words, the notion of occurrences provides a tree-geometric encoding of some parts of the Minimalist feature calculus pertaining to Move.

The remainder of the feature calculus can also be expressed tree-geometrically. For every $t \in$ FSL(slice(*Lex*)), node m of t, and LI l with licensee features $-f_1 \cdots - f_n$, $n \geq 0$:

Final. For $F \subseteq$ Base a distinguished set of *final categories*, if the slice root of l is the root of t, then the category feature c of l is a final category, i.e. $\nu(c) \in F$.

Merge. If m is associated to selector feature $= f$, then its left daughter is the slice root of an LI with category feature f.

Move. There exist distinct nodes $m_1, \ldots, m_n$ such that m_i (and no other node of t) is the i^{th} positive occurrence of l, $1 \leq i \leq n$.

SMC. If m is labeled *move*, there is exactly one LI for which m is a positive occurrence.

As expressed in Lemma 1 of [4], L is an MDTL iff it is the biggest subset of FSL(slice(*Lex*)) satisfying the four constraints above. Let us step back for a second and reflect on why this is the case. The first two constraints are easy to fathom. MGs require that the head of a derived tree must belong to some final category (usually c), and **Final** expresses this requirement in derivational parlance. Similarly, **Merge** ensures that an LI only selects LIs with matching category features. **Move** captures one half of MG's resource sensitivity with respect to Move: every licensee feature must be checked. For if an LI l has fewer occurrences than licensee features, some feature must remain unchecked because no Move node can check more than one of l's features. The other half of resource sensitivity is enforced by **SMC**: every Move node is responsible for checking exactly one licensee feature in the entire derivation. On the one hand, this rules out derivations with more licensor features than licensee features. On the other hand, it makes Move deterministic and rules out SMC violations. In a standard MG, the SMC guarantees for every step of a derivation that if $-f_i$ is the first licensee feature of an LI that still needs to be checked, it is not the first currently unchecked licensee feature of any other LI. If the SMC is violated, then the corresponding derivation tree must contain some node m properly dominating two LIs l and l' that both have $-f_i$ as their first unchecked feature. But then the lowest matching Move node reflexively dominating m will be an occurrence for both l and l', which is ruled out by **SMC**.

As all the constraints above can be stated in first-order logic with predicates for equality, proper dominance and right sister [4], we can view them as regular tree languages (i.e. as the sets of trees satisfying the respective constraints) and enforce them using regular control as described in [3]. The result is the desired MDTL — a regular set of derivation trees, each of which can be converted into the corresponding derived tree using the multi bottom-up transducer (mbutt) described in [12]. As is well-known, the string yield of an MG's derived tree language is an MCFL; in fact, MGs and MCFG are weakly equivalent [6, 13].

Definition 4. *A* Minimalist Grammar *is a 5-tuple* $G := \langle \Sigma, Feat, Lex, F, \mathcal{R} \rangle$ *such that*

- *Lex is a* $(\Sigma, Feat)$*-lexicon, and*
- $F \subseteq$ Base *is the set of final features, and*
- $\mathcal{R}$ *is a finite set of regular tree languages containing* ***F-Order****,* ***Final****,* ***Merge****,* ***Move****,* ***SMC****, and nothing else.*

The MDTL of G *is* $\mathrm{FSL}(\mathrm{slice}(Lex)) \cap \bigcap_{R \in \mathcal{R}} R$*. The tree language* $L(G)$ *generated by* G *is the image of its MDTL under the mbutt of [12]. Its string language is the string yield of* $L(G)$*.*

3 New Movement Types

3.1 General Strategy

By regulating Move purely via structural conditions on the distribution of occurrences, we have successfully decoupled the SMC and the resource-sensitivity of

MGs from the specifics of Move. Crucially, both **Move** and **SMC** hold independently of how occurrences are defined. In the previous section, proper dominance was used to capture *raising*, i.e. phrasal movement to a c-commanding position (not to be confused with raising constructions in the syntactic literature). However, if proper dominance was replaced by its inverse, the result would be *lowering* instead, which moves a phrase to a position it c-commands. As **Move** and **SMC** are unaffected by this change, the MDTL of an MG with lowering rather than raising would still be regular. In fact, regularity is preserved as long as the relation replacing proper dominance in the definition of occurrences is rational.

The mapping from derivation trees to derived trees, however, increases in complexity as the number of Move operations proliferates. Even if the mbutt of [12] were powerful enough to compute the mapping for any given rational relation, it would quickly become prohibitively complex. A better transduction model is provided by MSO-transductions. First, limiting ourselves to MSO-transductions guarantees that the string yield of the derived tree language is an MCFL (see [14] and references therein). Second, the MSO-transduction makes it very easy to capture other parameters of Move such as the size of the moved subtree. With respect to both expressivity and elegance, then, MSO-transductions are the ideal choice.

In order to avoid the complexities brought about by finite-copying MSO-transductions (which are necessary for the insertion of traces), I opt to decompose the transduction into two simpler steps. The derivation tree is first mapped to a term graph, also known as a multi-dominance tree in the syntactic literature. This first step handles differences in linearization and the size of the moved constituent. The term graph is then unfolded into a tree. Depending on how this unfolding is specified, one can allow for copying and covert movement.

Working with MSO-transductions obviously requires the introduction of some logical machinery. Due to space restrictions — and because it has already been accomplished in [4] — I refrain from giving a full model-theoretic formalization of MDTLs. I am also confident that the reader is sufficiently familiar with MSO to see that all constraints and definitions presented in the following sections are MSO-definable (keep in mind that a Minimalist lexicon is finite, so disjunctions, conjunctions and the recursive definitions given here are always finitely bounded). Where MSO-formulas are used, I follow the syntax of $\mathcal{L}^2_{K,P}$ [16], writing $\triangleleft$ for immediate dominance.

3.2 Step 1: Derivations and Occurrences

Generalizing the MG formalism at the level of derivation trees is quite simple. First the feature system is extended by another set M-Type of *movement types*, which represent various kinds of movement, for instance raising, lowering and sidewards movement; the movement type of a feature will usually be indicated as a superscript to avoid confusion. Each $\triangle \in$ M-Type in turn is associated with a pair $\left\langle R_0^{\triangle}, R^{\triangle} \right\rangle$ of rational relations. The first one determines the zero occurrence of an LI (e.g. the mother of its slice root for raising), while the second one assumes the role previously played by proper dominance in determining the

LI's positive occurrences. These pairs are called *movement specifications*, and I will often refer to them by the movement type they specify.

New movement types do not always behave as expected, though. With raising movement, competition between potential occurrences is eliminated by stipulating that only the closest (i.e lowest dominating) movement node counts as an occurrence. No such safeguard exists for lowering, though, and as a result many configurations are ambiguous and thus blocked by **Move** and **SMC**, which jointly ensure that Move is deterministic.

Consider the configuration in Fig. 2. From a linguistic perspective, the intended result is clear: b and d should lower into the specifiers of a and c, respectively. Given the current conception of occurrences, however, this configuration is in fact illicit. The LI d has not one but two occurrences, namely the Move nodes of c and a. Neither properly dominates the other, so neither blocks the other, either. Furthermore, the Move node of a is an occurrence for both b and d. So both **Move** and **SMC** are violated. Note, however, that the zero occurrence of b is the Merge node immediately dominating it and that this Merge node intervenes between the zero occurrence of d and the Move node of a. It seems, then, that the derivation can be salvaged by stipulating that LIs and their occurrences may act as interveners, too. To this end, movement specifications are modified to comprise another rational relation $P^{\triangle}$, and occurrences are defined in two steps.

– For all $i \geq 1$ and $\triangle \in$ M-Type, node m is a potential i-occurrence $\mathit{pocc}_i(l)$ of l iff
  - m matches $-f_i^{\triangle}$, and
  - $\langle m, \mathit{pocc}_{i-1}(l) \rangle \in R^{\triangle}$, and
  - there is no node z distinct from m such that
    - z matches $-f_i^{\triangle}$, and
    - $\langle z, \mathit{pocc}_{i-1}(l) \rangle \in R^{\triangle}$, and
    - $\langle m, z \rangle \in R^{\triangle}$.

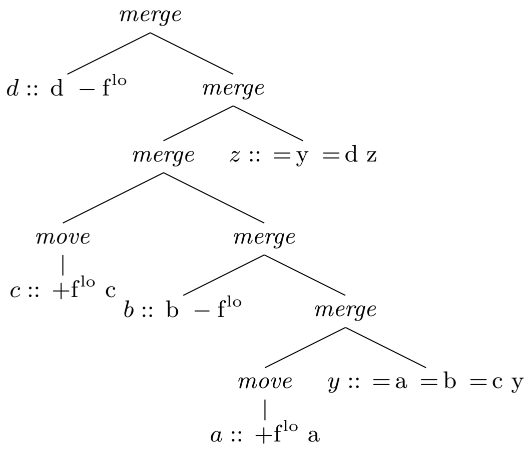

merge
d :: d $-$f$^{\text{lo}}$
merge
merge
z :: =y =d z
move
c :: +f$^{\text{lo}}$ c
merge
b :: b $-$f$^{\text{lo}}$
merge
move
y :: =a =b =c y
a :: +f$^{\text{lo}}$ a

Fig. 2. The move node of LI b is a potential occurrence for both c and e, and e has two occurrences, the move nodes of b and d

- For all $i \geq 1$ and $\triangle \in$ M-TYPE, node m is an i-occurrence $occ_i(l)$ of l iff
  - m is a potential i-occurrence of l, and
  - there is no node l' distinct from l such that m is a potential j-occurrence of l', $j \geq 1$, and $\langle occ_{j-1}(l'), occ_{i-1}(l)\rangle \in P^{\triangle}$.
- Both $occ_0(l)$ and $pocc_0(l)$ hold of node m iff $\langle m, l\rangle \in R_0^{\triangle}$.

As multiple movement types may be realized in the same grammar, I will sometimes say that m is a $\triangle$*-occurrence of* l to express that m is a positive occurrence of l and matches some $-f_i^{\triangle}$.

The reader is encouraged to verify for himself that this new definition yields the same result as the previous one on page 62 if $R^{\triangle} = P^{\triangle}$ is taken to be proper dominance (as far as I can tell, the equivalence of $R^{\triangle}$ and $P^{\triangle}$ holds for all movement operations in the syntactic literature). Restricting proper dominance to paths containing at most $k + 1$ left branches, on the other hand, gives rise to k-local movement in the sense of [4]. And as expected, lowering can be captured by using inverse proper dominance.

Sidewards movement [15, 20] is another prominent kind of movement. It does away with the c-command requirement on raising so that, for instance, complements and specifiers of specifiers are viable landing sites. For this reason, sidewards movement is heavily relied on in movement-based analyses of control and various extraction phenomena [cf. 7]. One conceivable formalization of sidewards movement is (inverse) slice containment: x slice contains y iff the slice containing a node immediately dominating x contains a node properly dominating y. If the zero occurrence of LI l is fixed to be the mother of the slice root of l, slice containment allows for locally restricted sidewards movement. The choice of slice might be freely altered to increase the locality domain, e.g. to the lowest slice of an LI of category c that properly dominates x. Minor modifications of this kind allow for sidewards movement to subsume previously implemented variants of Move such as Across-the-Board extraction [10] and clustering [2].

One peculiarity brought about by non-standard movement types is that the linguistic conception of derivation trees as a temporally ordered record of how derived trees are assembled in a step-wise fashion loses most of its intuitiveness. Going back to Fig. 2, for example, we see that b enters the derivation later than its last occurrence, the move node licensed by a (and similarly for d). Under the standard construal of derivation trees, this would entail that before b was inserted into the derived tree it had already lowered into the specifier of a. This apparent contradiction can be resolved by viewing derivation trees as a relative of proof nets (see [17, 19] and references therein): they are merely a graph-theoretic representation of the Minimalist feature calculus and its checking requirement, with movement types corresponding to specific rules of inference in this calculus.

3.3 Step 2: Mapping to Term Graphs

The next step is the mapping from derivation trees to the derived structures posited by linguists. Recent Minimalist reasoning maintains that derived trees are actually multi-dominance trees — tree-like structures in which a node may have multiple mothers. Movement of a phrase XP to YP no longer involves displacement of XP into the (newly created) specifier of YP, with a trace or copy of XP being left behind in the original position. Instead, only a new dominance branch is added between YP and XP, making the latter a specifier of the former. The derived tree in Fig. 1, for instance, may be converted into this new format by interpreting the movement arrows as dominance branches. More importantly, the multi-dominance tree can also be obtained from the derivation tree by adding branches from the slice root of an LI to all its positive occurrences (ignoring linearity and labels, for now).

The mathematical analog of converting derivation trees into multi-dominance trees is transducing trees into term graphs. For the simple mapping required for MGs, MSO term graph transductions [1] are more than sufficient. Recall that such a transduction is specified by a pair $\langle \Psi, \Theta \rangle$. The first component, Ψ, is a set of formulas determining which nodes of the input tree are present in the term graph and what their label is, while Θ consists of formulas defining the relations that hold between the nodes of the term graph. Crucially, those formulas may only use predicates from the MSO-language used to define the input tree.

In our case, all nodes of the input tree are present in the term graph, so we only need to worry about changing the label and defining precedence $\prec$ and immediate dominance $\blacktriangleleft$. The latter is readily stated in terms of occurrences and immediate dominance in the derivation tree ($occ_i(x, l)$ holds iff x is the i^{th} occurrence of l, and $|\delta|$ is the maximum of licensee features occurring on a single LI in the grammar's lexicon).

$$x \blacktriangleleft y \leftrightarrow x \triangleleft y \vee \exists l \Big[\bigvee_{1 \leq i \leq |\delta|} occ_i(x, l) \wedge \mathit{sliceroot}(y, l) \Big]$$

The predicate *sliceroot* ensures that the dominance branch is added between the occurrence and the root of the slice whose LI is undergoing movement. In other words, it enforces phrasal movement. But we can of course replace *sliceroot* by a different predicate to yield other kinds of movement. For instance, if y and l are identical, the result is head movement. For pied-piping, y is the slice root of some slice containing the slice of l. Wee see, then, that *sliceroot* can be freely exchanged for other MSO-definable predicates to alter the size of the moved subtree.

More precisely, movement specifications are extended to $\big\langle R_0^{\triangle}, R^{\triangle}, P^{\triangle}, \mathit{root}^{\triangle} \big\rangle$, where $\mathit{root}^{\triangle}$ is an MSO formula with two free variables that picks out a unique node to serve as the root of the subtree carried along by $\triangle$-movement. Then $\blacktriangleleft$ is defined as follows ($occ_i^{\triangle}(x, l)$ holds iff both $occ_i(x, l)$ and x is a $\triangle$-occurrence):

$$x \blacktriangleleft y \leftrightarrow x \triangleleft y \vee \exists l \Big[\bigvee_{\substack{1 \leq i \leq |\delta| \\ \triangle \in \text{M-T}\textsc{ype}}} \big(occ_i^{\triangle}(x, l) \wedge \mathit{root}^{\triangle}(y, l) \big) \Big]$$

Certain restrictions must be put in place, though, to ensure that the output of the transduction is indeed a term graph and in line with certain Minimalist intuitions. For all $t \in \text{FSL}(\text{slice}(\mathit{Lex}))$, nodes x, y, l of t, $\triangle, \circ \in$ M-Type, and string δ of licensee features:

Containment. If $-f^{\triangle}$ precedes $-f^{\circ}$ in δ and both $root^{\triangle}(x,l)$ and $root^{\circ}(y,l)$ hold, then x reflexively dominates y.

Dominance. If $root^{\triangle}(x,l)$ holds, then x reflexively dominates l.

Exocentricity. If m is a positive occurrence of l, m is not associated to a feature of l.

No Cycle. If $x \blacktriangleleft^{+} y$ holds, then $y \blacktriangleleft^{+} x$ does not.

The first three conditions are linguistically motivated. They prevent LIs from triggering displacement of unrelated subtrees, stop moved subtrees from seemingly recombining with material that had previously been left behind, and prohibit LIs from licensing their own movement. The last one ensures that no cycles are present in the output graph. Keep in mind that the transitive closure of $\blacktriangleleft$ is MSO-definable, so **No Cycle** is indeed an MSO formula.

Besides dominance, one must also take care of precedence and the output labels. This offers another opportunity to reign in a variant of Move, namely rightward movement (also known as extraposition). The feature system is enriched by yet another component, Headedness := $\{left, right\}$. Headedness information simplifies the task of defining predicates for left daughter $\blacktriangleleft_1$, right daughter $\blacktriangleleft_2$, and precedence $\prec$ (which isn't necessarily a strict order in term graphs). Only the headedness of positive polarity features is taken into account. This is formally expressed by restricting the predicates $left(x)$ and $right(x)$ to interior nodes associated to features of the respective headedness. Furthermore $x \sim y$ iff x and y are nodes of the same slice.

$$x \blacktriangleleft_1 y \leftrightarrow x \blacktriangleleft y \wedge \big((x \sim y \wedge left(x)) \vee (x \not\sim y \wedge right(x))\big)$$

$$x \blacktriangleleft_2 y \leftrightarrow x \blacktriangleleft y \wedge \big((x \sim y \wedge right(x)) \vee (x \not\sim y \wedge left(x))\big)$$

$$x \prec y \leftrightarrow \exists x' \exists y' \exists z \big[(x' \approx x \vee x' \blacktriangleleft^{+} x) \wedge (y' \approx y \vee y' \blacktriangleleft^{+} y) \wedge z \blacktriangleleft_1 x' \wedge z \blacktriangleleft_2 y'\big]$$

Relabeling the interior nodes based on headedness is just as simple.

$$< (x) \leftrightarrow (merge(x) \vee move(x)) \wedge left(x)$$

$$> (x) \leftrightarrow (merge(x) \vee move(x)) \wedge right(x)$$

Finally, LIs must lose all their features but keep their string exponents. In principle one would have to ensure that the LI of the highest slice keeps its category feature, but this requirement needlessly complicates the transduction and is itself merely an artefact of the original MG formalism.

$$\bigwedge_{\sigma \in \Sigma} \Big(\sigma(x) \leftrightarrow \bigvee_{l := \sigma :: f_1 \cdots f_n \in Lex} l(x)\Big)$$

3.4 Step 3: Unfolding into Derived Trees

The usual way to unfold a term graph into a tree requires unbounded copying: given a subtree t whose root has n mothers $m_1, \ldots, m_n$, create n copies t_i of t

such that m_i is the mother of t_i. While this is a feasible strategy to accommodate MGs with copying [9], it increases weak generative capacity. I therefore restrict my attention to unfoldings without unlimited copying.

Let us first consider the case of standard MGs. Suppose LI l has n occurrences, so that the nodes $m_1 := occ_1(l), \ldots, m_n = occ_n(l), m_{n+1}$ all dominate the slice root of l; $m_{n=1}$ is the unique Merge node that introduced l into the derivation. Then the unfolding just has to create $n-1$ traces and replace the dominance branches between the slice root of l and each m_i, $i < n$, by a dominance branch between a trace and m_i. As a result, only the last occurrence of l immediately dominates its slice root, which is tantamount to saying that the constituent headed by l moved into the specifier immediately dominated by m_n.

In the syntactic literature, a distinction is made between overt and covert movement, however, and only the former is visible. For the purposes of unfolding this means that the branch to l's slice root should not be preserved for the last occurrence of l, but the occurrence with the highest index that licensed overt movement. To this end, the feature system is once again modified so as to indicate overtness via the diacritics o and c. The matching relation also needs to be extended accordingly to ensure that licensor and licensee features agree on (c)overtness.

This system is still unsatisfactory, though, as MGMGs allow for the size of the moved constituent to vary with feature type. This entails that more than just one occurrence of an LI l may dominate parts of the material that was displaced by moving l. The challenge is to find the last occurrence for each of these parts. Given LI l, derivation tree t and $\triangle, \circ \in$ M-Type, $\triangle \cong \circ$ iff t contains a node x such that $root^{\triangle}(x, l) = root^{\circ}(x, l) = 1$. Now for every LI l, $\triangle, \circ \in$ M-Type, and $j > i \geq 1$, $occ_i^{\triangle}(l)$ is a *landing site* iff $occ_i^{\triangle}$ is associated to an overt feature and there is no $occ_j^{\circ}(l)$ such that $\triangle \cong \circ$. The unfolding then turns the term graph into a tree such that if $occ_i^{\triangle}(l)$ is a landing site, it immediately dominates the $\triangle$-root of l. All branches originating from a Merge node immediately dominating the root of a displaced subtree or from an occurrence of l that is not a landing site are replaced by branches immediately dominating a trace.

Clearly the number of traces per term graph cannot exceed the total number of nodes in the graph, wherefore the unfolding is of linear size increase. It follows immediately that the composition of our MSO term transduction and unfolding is an MSO-definable tree transduction (with finite copying). Let τ be this tree transduction. As MDTLs are still regular, it must be the case for every single one of them that the string yield of its image under τ is an MCFL.

3.5 Defining Movement-Generalized Minimalist Grammars

Now we are finally in a position to define MGMGs.

Definition 5. *Let* Base *and* M-Type *be disjoint, non-empty, finite sets of* feature names *and* movement types*, respectively. Furthermore,* Op $:= \{merge, move\}$, Polarity $:= \{+, -\}$, Headedness $:= \{left, right\}$*, and* Overt $:= \{o, c\}$*, are the sets of* operations*,* polarities*,* headedness parameters*, and* overtness markers*,* *respectively. A* feature system *is a non-empty set Feat* $\subseteq$ BASE $\times$ OP $\times$ POLARITY $\times$ M-TYPE $\times$ HEADEDNESS $\times$ OVERT. *Two features* match *iff they agree on their name, operation, movement type, and overtness but have opposite polarities.*

Definition 6. *Given* $\triangle \in$ M-TYPE, *the* movement specification of $\triangle$ *is given by a 4-tuple* $\langle R_0^{\triangle}, R^{\triangle}, P^{\triangle}, \mathrm{root}^{\triangle} \rangle$ *of binary rational relations.*

Definition 7. *A* Movement-Generalized Minimalist grammar *G over alphabet* Σ *and feature system Feat is a 6-tuple* $G := \langle \Sigma, \mathit{Feat}, \mathit{Lex}, F, \mathcal{R}, \mathcal{M} \rangle$, *where*

- *Lex is a* (Σ, Feat)*-lexicon, and*
- $F \subseteq$ BASE *is a set of final categories, and*
- $\mathcal{R}$ *is a finite set of regular tree languages containing at least* ***Containment, Dominance, Exocentricity, F-Order, Final, Merge, Move, No Cycle, SMC****, and*
- $\mathcal{M}$ *is an* M-TYPE*-indexed family of movement types.*

Its MDTL is $\mathrm{FSL}(\mathrm{slice}(\mathit{Lex})) \cap \bigcap_{R \in \mathcal{R}} R$. *The tree language* $L(G)$ *generated by* G *is the image of its MDTL under the MSO transduction* τ, *and its string language is the string yield of* $L(G)$.

Theorem 1. *MGs and MGMGs have the same weak generative capacity.*

4 Tree Adjunction ≡ Reset Lowering

Even though MGs properly subsume TAGs with respect to weak generative capacity [13, 18], the two formalisms are incomparable at the level of tree languages [12, 14]. This result does not hold for MGMGs. In fact, TAGs with strictly binary branching and X′-like projection are equivalent to MGs with a limited kind of lowering, as I will briefly sketch now (for a full proof see [5]).

Consider the following scenario. The tree α consists of subtrees t(op) and b(ottom), with b rooted in the node V′ of t, which is a projection of LI l_α in b. The auxiliary tree β, whose foot node is $\underline{\mathrm{V}'}$ and whose root is a projection of LI l_β, adjoins into α at V′, yielding γ. It should be easy to see that γ can be approximated via lowering. First, β is selected by l_α immediately after the subtree immediately dominated by V′ in b. After that, the foot node of β is replaced by an empty category with licensor feature $+f^{\triangle}$, and $-f^{\triangle}$ is inserted after the category feature of l_α. The $\triangle$-root of l_α is the sister of the root of β. The derived tree corresponding to this lowering step only differs from γ in the presence of two superfluous interior nodes immediately above the $\triangle$-root of b and the root of β, respectively; both can easily be detected and removed.

The procedure carries over to the general case without major problems, but it hinges on a particular definition of lowering. For example, if another auxiliary tree β' was to adjoin immediately above V′ in t, the algorithm would add the requisite nodes and features as intended (now using a new movement type $\circ$ to pick out the correct root). But since the Move nodes in β and β' are not related by inverse proper dominance, defining lowering in these terms is insufficient — only the first occurrence could ever be reached. Note, though, that each $u \in \{\beta, \beta'\}$ contains exactly one $\triangle$-occurrence of l_α, where the $\triangle$-root of l_α is the sister of the root of u. In a sense then, we do not want occurrences to be computed in sequence, but rather independently of each other using inverse proper dominance and picking the $\triangle$-root of l as the zero-occurrence for computing $\triangle$-occurrences. Emulating this behavior in the MGMG system is slightly cumbersome: for all $\triangle \neq \circ \in$ M-Type, $\langle x, y \rangle \in R^\triangle$ iff either y is a $\triangle$-root c-commanding x or there is a $\triangle$-root x' and a $\circ$-root y' such that y' properly dominates x, no $\circ$-root properly dominated by y' properly dominates x', y' c-commands y, and x' c-commands x. In conjunction with **Containment**, this always yields a well-defined relation that exhibits the desired behavior. I call this relation *reset lowering.*

Although many technical details are missing, it should nonetheless be clear that the translation described above can be carried out by a linear tree transducer. Since TAG derivation tree languages are regular, the output language L of the transducer is too. In order to convert L into an MDTL, one only needs to employ the label refinement algorithm given in [3]. The end result is an MG with reset lowering that generates the same tree language as the original TAG (under a simple homomorphism that removes the redundant interior nodes).

As for the translation in the other direction, I presuppose that all licensee features are built from the same feature name f, that is to say, there are no two distinct features $-f^\triangle$, $-g^\triangle$ for any $\triangle \in$ M-Type. The reader may verify for himself that the MGs created by the algorithm above satisfy this condition. From **Containment** and the definition of reset lowering it further follows that no LI has more than one feature of a specific movement type. Now we only have to apply the spirit of the previous translation in reverse. Suppose we are given subtrees t, b, β and a node u that is immediately dominated by a leaf v of t and immediately dominates the roots of b and β. Let α be the composition of t and b such that v immediately dominates the root of b. Then lowering of b into β corresponds to adjunction of β into α at the root of b.

Hopefully the reader can appreciate now why L is a tree adjoining language iff it is the derived tree language of some MGMG with reset lowering and only one feature name per movement type. For MGMGs with normal lowering, the translation must fail because for every $i > 1$ there is such an MGMG G with Base $= \{f\}$ that generates the language $a_1^n \cdots a_i^n$. The lexicon of G contains

- a_j :: $\mathrm{a_j}$ for all $j < i$,
- a_i :: $= \mathrm{a_1} \cdots = \mathrm{a_j}\ \mathrm{a_i}\ (-\mathrm{f}^{\mathrm{a_1}} \cdots - \mathrm{f}^{\mathrm{a_i}})$,
- a_i :: $= \mathrm{a_i}\ + \mathrm{f}^{\mathrm{a_1}} = \mathrm{a_1} \cdots + \mathrm{f}^{\mathrm{a_i}}\ \mathrm{a_i}\ (-\mathrm{f}^{\mathrm{a_1}} \cdots - \mathrm{f}^{\mathrm{a_i}})$,

where the $\mathrm{a_j}$-root of an LI is either the Merge node immediately dominating a_j or the Move node immediately above that (if it exists).

5 Conclusion

MGs can easily be generalized to MGMGs once we view them in terms of their derivation trees. The notion of occurrence, which is used to regulate the distribution of Move nodes, can be redefined to allow for variants of Move with different directionality (lowering, sidewards movement etc.). The mapping from derivation trees to derived trees, on the other hand, furnishes parameters to determine linearization, the size of the moved subtree, and the overt/covert distinction. As all these modifications are required to be MSO-definable, MGMGs have the same weak generative capacity as MGs despite their greatly increased strong generative capacity.

Acknowledgments. I would like to thank the LACL reviewers, Ed Stabler, and all the members of the UCLA MathLing Circle; without their questions and suggestions, this paper would have been even less approachable. Furthermore, several discussions with Michael Freedman during ESSLLI 2011 on an earlier version of the TAG-to-MG translation improved my understanding of the TAG formalism in various ways.

References

[1] Bloem, R., Engelfriet, J.: A comparison of tree transductions defined by monadic second-order logic and attribute grammars. Journal of Computational System Science 61, 1–50 (2000)

[2] Gärtner, H.-M., Michaelis, J.: On the treatment of multiple-wh-interrogatives in minimalist grammars. In: Hanneforth, T., Fanselow, G. (eds.) Language and Logos, pp. 339–366. Akademie Verlag, Berlin (2010)

[3] Graf, T.: Closure Properties of Minimalist Derivation Tree Languages. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 96–111. Springer, Heidelberg (2011)

[4] Graf, T.: Locality and the complexity of minimalist derivation tree languages. In: Proceedings of the 16th Conference on Formal Grammar (2011) (to appear)

[5] Graf, T.: Tree adjunction as lowering in minimalist grammars (2012), ms., UCLA

[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] Hornstein, N.: Movement and control. Linguistic Inquiry 30, 69–96 (1999)

[8] 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)

[9] Kobele, G.M.: Generating Copies: An Investigation into Structural Identity in Language and Grammar. Ph.D. thesis, UCLA (2006)

[10] Kobele, G.M.: Across-the-board extraction and minimalist grammars. In: Proceedings of the Ninth International Workshop on Tree Adjoining Grammars and Related Frameworks (2008)

[11] Kobele, G.M.: Minimalist Tree Languages Are Closed Under Intersection with Recognizable Tree Languages. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 129–144. Springer, Heidelberg (2011)

[12] 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)

[13] 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)

[14] Mönnich, U.: Grammar morphisms (2006), University of Tübingen

[15] Nunes, J.: The Copy Theory of Movement and Linearization of Chains in the Minimalist Program. Ph.D. thesis, University of Maryland, College Park (1995)

[16] Rogers, J.: A Descriptive Approach to Language-Theoretic Complexity. CSLI, Stanford (1998)

[17] Salvati, S.: Minimalist Grammars in the Light of Logic. In: Pogodalla, S., Quatrini, M., Retoré, C. (eds.) Logic and Grammar. LNCS, vol. 6700, pp. 81–117. Springer, Heidelberg (2011)

[18] Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)

[19] 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)

[20] Stabler, E.P.: Sidewards without copying. In: Penn, G., Satta, G., Wintner, S. (eds.) Proceedings of the Conference on Formal Grammar 2006, pp. 133–146. CSLI Publications, Stanford (2006)

[21] Stabler, E.P.: Computational perspectives on minimalism. In: Boeckx, C. (ed.) Oxford Handbook of Linguistic Minimalism, pp. 617–643. Oxford University Press, Oxford (2011)

[22] Stabler, E.P., Keenan, E.: Structural similarity. Theoretical Computer Science 293, 345–363 (2003)

Toward the Formulation of Presupposition by Illative Combinatory Logic

Yuri Ishishita and Daisuke Bekki

Ochanomizu University,
Graduate School of Humanities and Sciences,
Division of Advanced Sciences, Department of Computer Science
{ishishita.yuri,bekki}@is.ocha.ac.jp

1 Introduction

1.1 Proof-Theoretic Semantics of Natural Language

Model-theoretic semantics, which originates with Tarski, and proof-thoretic semantics, which originates with Gentzen, are two views in semantics of logic that are distinct from but closely related to each other. Each has advantages over the other in investigating a certain aspect of logic, and it is more or less commonly accepted that rather than being a matter of methodological choice, utilizing them gives us diversified standpoints on various issues. A good example of this is the two proofs of the consistency of LK: one proof is based on soundness and the other on cut-elimination, which makes use of different resources but also together reveals what the consistency of predicate calculus indeed depends on.

The same consideration may apply to formal semantics of natural language as well, where the model-theoretic view has been dominant for almost forty years since Montague's PTQ [10] first appeared. Recently, the proof-theoretic view of natural language semantics has been pursued [12][19][6]. During its history, formal semantics has extended its theory in model-theoretic ways in order to formulate and explain various phenomena with regard to the meaning of natural language, which otherwise cannot be formulated or explained by any simple predicate calculi. Then the natural question that arises is: "Is there a proof-theoretic counterpart for each of such extensions?" For example, is a proof-theoretic explanation of presuppositions [18] possible?

1.2 Proof-Theoretic Analyses of Presupposition

The aim of this paper is to propose a new proof-theoretic analysis of presupposition. The mainstream analyses of presupposition (filtering analyses [8][16], cancellation analyses [15][7], PIA analyses [14][13], and dynamic analyses [2], among others) have been model-theoretic, but there are a small number of proof-theoretic approaches [4][9] based on Martin-Löf type theory.

Mineshima's theory [9] adopts a version of Martin-Löf type theory extended with the ε operator (of ε-calculus) which he uses for representing presupposition triggers. The presuppositions are regarded as the well-formedness conditions

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 74–85, 2012.

(in the formation rules in terms of Martin-Löf type theory) for ε-terms. In the following set of (classical) puzzles, (2) is required as a premise in the proof of both (1a) and (1b).

(1) a. The king of France is bald.
b. The king of France is not bald.

(2) There is a king of France.

Meanwhile Martin-Löf type theory adopts a natural deduction style system for both the proof of semantic validity and the well-formedness of an inference. When applied to natural languages, it means that we use a natural deduction style system for both the proof of semantic validity and the grammaticality of a sentence. However, a logical grammar of a natural deduction style has a problem in properly describing the constraint on wh-movement/extractions [11]. This problem can be solved by adopting CCG [17] as a syntactic theory, which corresponds to a type theory of Hilbert-sytle, namely a combinatory logic [3]. Thus we are led to the use of ICL, which is a higher-order dependent type theory that corresponds to a combinatory logic, which enables us to inherit what Mineshima has achieved with Martin-Löf type theory and what CCG has achieved as a syntactic theory at the same time.

In this paper, we propose an analysis of presupposition based on *Illative Combinatory Logic (ICL)* [1][5], a logic that consists of the combinators and the lambda terms.

2 Illative Combinatory Logic

ICL is a type-free lambda calculus (Combinatory Logic) extended with the terms that behave as logical operators of first-order predicate logic. It has been known that the simple addition of the logical operators leads to a paradox. In order to avoid the paradox, a strategy to distinguish the terms representing propositions from those representing others has been adopted since Curry. This setting makes it possible to represent the meta-logical notions, such as "X is a proposition/X is a type", as object-level terms (e.g. combinators or lambda terms). This point is crucial to formulate presuppositions.

ICL corresponds to a minimal predicate logic which lacks a notion of negation. It has four variant systems, called $\mathcal{I}$P, $\mathcal{I}$F, $\mathcal{I}\Xi$, $\mathcal{I}$G, because there are two translations from propositional logic and predicate logic to ICL, respectively. Their use for the two kinds of interpretation is as follows. Let $[\ ]^1$ be the direct and $[\ ]^2$ the propositions-as-types translation. For example, $[\ ]^2$: PRED $\rightarrow \mathcal{I}$G. The latter concept is just a concept of higher-order dependent type theory. Considering the similarities to previous studies [4][9], we focus on the system $\mathcal{I}$G in this paper.

In this section, we will briefly introduce the definition of PRED and $\mathcal{I}$G given in [1], and the two translations $[\ ]^1$ and $[\ ]^2$, in order to give a basis for our proposal.

Table 1. Systems of ICL

	$[\,]^1$	$[\,]^2$
PROP	$\mathcal{I}$P	$\mathcal{I}$F
PRED	$\mathcal{I}\Xi$	$\mathcal{I}$G

2.1 PRED

The system PRED is a many-sorted predicate logic, defined as follows.

Definition 1 (Signature of PRED**).** A many-sorted *signature* $\Sigma = \langle S, F, P\rangle$ consists of:

1. An enumerable set S called *sorts*.
2. An $S^* \times S$-indexed family of sets of function symbols $(F_{a,c})_{a\in S^*, c\in S}$.
3. An S^*-indexed family of sets of predicate symbols $(P_a)_{a\in S^*}$.

where a is called an arity and c is called a co-arity.

Example 2. For example, let $s = \langle S, F, P\rangle$ be a signature specified as follows:

$$\begin{aligned} S &\equiv \{\mathbf{A}\} \\ F &\equiv (F_{()}, \mathbf{A}) \\ F_{(),\mathbf{A}} &\equiv \{\mathbf{a_1}, \mathbf{a_2}\} \qquad \mathbf{a_1}, \mathbf{a_1} \text{ are constants.} \\ P &\equiv (P_{\mathbf{A}}) \\ P_{\mathbf{A}} &\equiv \{\mathbf{P_1}, \mathbf{P_2}\} \qquad \mathbf{P_1}, \mathbf{P_2} \text{ are unary relations.} \end{aligned}$$

Definition 3 (Variables of PRED**).** Given a signature $\Sigma = \langle S, F, P\rangle$, an S-indexed family of sets of variables $(V_s)_{s\in S}$ over Σ is a family of sorted variables x^s $(s \in S)$.

Definition 4 (Terms of PRED**).** Given a signature $\Sigma = \langle S, F, P\rangle$ and a family of sorted variables V, the S-indexed family of well-sorted terms $(T_s)_{s\in S}$ over Σ and V is defined as follows.

1. if $x^s \in V_s$, then $x^s \in T_s$.
2. if $f \in F_{(),s}$, then $f \in T_s$.
3. if $t_1 \in T_{s_1}, \ldots, t_n \in T_{s_n}$ and $f \in F_{(s_1,\ldots,s_n),t}$, then $f(t_1, \ldots, t_n) \in T_t$.

Definition 5 (Formulae of PRED**).** Given a signature $\Sigma = \langle S, F, P\rangle$ and a family of sorted variables V, the set of well-sorted formulae $\mathsf{Form}_{\mathrm{PRED}}$ over Σ and V is defined as follows:

1. if $t_1 \in T_{s_1}, \ldots, t_n \in T_{s_n}$ and $p \in F_{(s_1,\ldots,s_n)}$, then $p(t_1, \ldots, t_n)$ is a well-formed formula.
2. if ϕ, ψ are well-formed formulae, then $\phi \supset \psi$ is a well-formed formula.
3. if $x^s \in V_s$ and ϕ is a well-formed formula, then $\forall x^s \phi$ is a well-formed formula.

The natural deduction system for PRED has the following inference rules:

Definition 6 (Natural deduction system for PRED)

$$\frac{\phi \in \Gamma}{\Gamma \vdash \phi} \qquad \frac{\Gamma \vdash \phi \supset \psi \qquad \Gamma \vdash \phi}{\Gamma \vdash \psi} \qquad \frac{\Gamma,\ \phi \vdash \psi}{\Gamma \vdash \phi \supset \psi}$$

$$\frac{\Gamma \vdash \forall x^{\mathbf{A}}\phi \qquad t \in T_{\mathbf{A}}}{\Gamma \vdash \phi[x^{\mathbf{A}} := t]} \qquad \frac{\Gamma \vdash \phi \qquad x^{\mathbf{A}} \notin FV(\Gamma)}{\Gamma \vdash \forall x^{\mathbf{A}}\phi}$$

2.2 $\mathcal{I}$G

The system $\mathcal{I}$G is a type-free lambda calculus extended with the combinators Ξ ("subsumes") and L ("is a type"). [1]

Definition 7 (Syntax of $\mathcal{I}$G). The set of terms $T = \Lambda(\Xi, \mathsf{L})$ in $\mathcal{I}$G is defined by the following BNF grammar, where *Var* is a set of variables, $x \in \mathit{Var}$, and $c \in \{\Xi, \mathsf{L}\}$.

$$T ::= x \mid c \mid TT \mid \lambda x.T$$

Remark 8.

- The term X has an assertive value.
- The term XZ is interpreted as a statement "Z is of type X," "$Z \in X$", or "Z satisfies the predicate X."
- The term $\lambda\zeta.X$ corresponds to the class $\{\zeta \mid X\}$.
- The term ΞXY is interpreted as "$X \subseteq Y$" or "$(\forall x \in X)Yx$."
- The term $\mathsf{L}X$ is interpreted as a statement "X is a type", or "X is a well-formed formula (wff)."

Definition 9 (Combinators). The combinators F and G are defined as follows, where $\mathsf{S} \equiv \lambda pqr.pr(qr), M \circ N \equiv \lambda x.M(Nx)$.

$$\begin{array}{rcl} \mathsf{F} & \equiv & \lambda xyz.\Xi x(y \circ z) \\ \mathsf{G} & \equiv & \lambda xyz.\Xi x(\mathsf{S}yz) \end{array}$$

The intuitive meaning of $\mathsf{F}XYZ$ and $\mathsf{G}XYZ$ are "a term Z has a type $X \supset Y$ and "a term Z has a type Y for all objects of a type X", respectively.

Definition 10 (Inference rules of $\mathcal{I}$G)

$$(\in_e)\frac{X \in \Gamma}{\Gamma \vdash X} \qquad (\beta\eta)\frac{\Gamma \vdash X \qquad X =_{\beta\eta} Y}{\Gamma \vdash Y}$$

$$(\mathsf{G}_e)\frac{\Gamma \vdash \mathsf{G}XYZ \qquad \Gamma \vdash XV}{\Gamma \vdash YV(ZV)} \qquad (\mathsf{G}_i)\frac{\Gamma, Xx \vdash Yx(Zx) \qquad \Gamma \vdash \mathsf{L}X}{\Gamma \vdash \mathsf{G}XYZ}\ (x \notin FV(\Gamma, X, Y, Z))$$

$$(\mathsf{G}_\mathsf{L})\frac{\Gamma, Xx \vdash \mathsf{L}(Yx) \qquad \Gamma \vdash \mathsf{L}X}{\Gamma \vdash \mathsf{L}(\mathsf{G}XY)}\ (x \notin FV(\Gamma, X, Y))$$

The $\beta\eta$ reduction rules are defined in the standard way. The rules with the subscript e, i, L are the elimination rule, the introduction rule, and the well-formedness rule, respectively.

Theorem 11 (The combinator F). The following rules are derivable in $\mathcal{I}$G.

$$(\mathsf{F}_e)\frac{\Gamma \vdash \mathsf{F}XYZ \qquad \Gamma \vdash XV}{\Gamma \vdash Y(ZV)} \qquad\qquad (\mathsf{F}_i)\frac{\Gamma, Xx \vdash Y(Zx) \qquad \Gamma \vdash \mathsf{L}X}{\Gamma \vdash \mathsf{F}XYZ}\ (x \notin FV(\Gamma,X,Y,Z))$$

$$(\mathsf{F}_\mathsf{L})\frac{\Gamma, Xx \vdash \mathsf{L}Y \qquad \Gamma \vdash \mathsf{L}X}{\Gamma \vdash \mathsf{L}(\mathsf{F}XY)}\ (x \notin FV(\Gamma,X,Y))$$

Proof. By replacing Y in Definition 10 with $\mathsf{K}Y$, where $\mathsf{K} \equiv \lambda pq.p$. ■

Example 12. The judgment $\vdash_{\text{PRED}} \forall x^{\mathbf{A}}(\mathbf{P}x \supset \mathbf{P}x)$ is translated into the following judgment in $\mathcal{I}$G:

$$\mathsf{L}A, \mathsf{F}A\mathsf{L}P \ \vdash \ \mathsf{L}(\mathsf{G}A(\lambda x.\mathsf{F}(Px)(Px))),$$

which is interpreted as "if A is a type and P is of type $A \to \mathsf{L}$, then $\mathsf{G}A(\lambda x.\mathsf{F}(Px)$ $(Px))$ is a type." In this case, the fact that $\mathsf{G}A(\lambda x.\mathsf{F}(Px)(Px))$ is a wff is derived from $\mathsf{L}A$ and $\mathsf{F}A\mathsf{L}P$.

On the other hand, the fact that $\forall x^{\mathbf{A}}(\mathbf{P}x \supset \mathbf{P}x)$ is tautology is interpreted in $\mathcal{I}$G as the inhabitation of the type $\mathsf{G}A(\lambda x.\mathsf{F}(Px)(Px))$ as follows.

$$\mathsf{L}A, \mathsf{F}A\mathsf{L}P \ \vdash \ \mathsf{G}A(\lambda x.\mathsf{F}(Px)(Px))(\lambda x.\lambda y.y)$$

Here, the lambda term $\lambda x.\lambda y.y$ is a proof for $\mathsf{G}A(\lambda x.\mathsf{F}(Px)(Px))$.

2.3 Translation from PRED to $\mathcal{I}$G

As mentioned in Table 1, the map $[\,]^2$ translates a judgment of PRED to a judgment of $\mathcal{I}$G. According to the signature $\Sigma = \langle S, F, P\rangle$ of PRED, we consider the system $\mathcal{I}$G extended with the set of constants which has a one-to-one correspondence to the set $S \cup (\bigcup F) \cup (\bigcup P)$. Let $\Lambda^{\Sigma}(\Xi, \mathsf{L})$ be the set of well-formed formulae defined in this extended system.

Definition 13 (Translation to $\mathcal{I}$G)

The following map $[-]^2$ and $\Gamma(-)$ are defined by Table 2 and Table 3. [1]

$$\begin{aligned}[-]^2 &: \mathsf{Form}_{\text{PRED}} \to \Lambda^{\Sigma}(\Xi, \mathsf{L})\\ \Gamma &: \mathsf{Form}_{\text{PRED}} \to \text{illative contexts}\end{aligned}$$

Moreover, the following two maps Γ^2_{-} and $\Gamma^{2,+}_{-}$ are introduced, which map a signature to illative contexts. We show it by the following examples, which are the translations of the signature in Example 2. These maps can be easily generalized for arbitrary signatures.

$$\begin{aligned}\Gamma^2_{\Sigma} &= \langle \mathsf{L}A, \mathsf{F}A\mathsf{L}P_1, \mathsf{F}A\mathsf{L}P_2, \ldots, Aa_1, Aa_2, \ldots\rangle\\ \Gamma^{2,+}_{\Sigma} &= \Gamma^2_{\Sigma}, Ax \qquad \text{where } x \in \mathit{Var}.\end{aligned}$$

Table 2. Translations of terms

t	$[t]^2$	$\Gamma(t)$
$x^{\mathbf{A}}$	x	Ax
$\mathbf{f}(t_1 \dots t_n)$	$f[t_1]^2 \dots [t_n]^2$	$\Gamma(t_1), \dots, \Gamma(t_n)$

Table 3. Translations of formulae

ϕ	$[\phi]^2$	$\Gamma(\phi)$
$\mathbf{P}(t_1 \dots t_n)$	$P[t_1]^2 \cdots [t_n]^2$	$\Gamma(t_1), \dots, \Gamma(t_n)$
$\psi \supset \chi$	$\mathsf{F}[\psi]^2[\chi]^2$	$\Gamma(\psi), \Gamma(\chi)$
$\forall x^{\mathbf{A}}\psi$	$\mathsf{G}A(\lambda x.[\psi]^2)$	$\Gamma(\psi) - \{Ax\}$

Lemma 14. Let $\phi \in \mathsf{Form}_{\mathrm{PRED}}$.[1]

(1) If $t \in T_{\mathbf{A}}$, then in $\mathcal{I}$G one has $\Gamma(t), \Gamma^2_{\Sigma} \vdash A[t]^2$.
(2) $\Gamma^2_{\Sigma}, \Gamma(\phi) \vdash_{\mathcal{I}\mathrm{G}} \mathsf{L}[\phi]^2$.

Theorem 15 (Soundness of the interpretation for PRED**).** Let $\Delta \cup \{\phi\} \subseteq \mathsf{Form}_{\mathrm{PRED}}$, then the following holds.[2]

$$\Delta \vdash_{\mathrm{PRED}} \phi \;\Rightarrow\; \Gamma^{2,+}_{\Sigma}, [\Delta]^2, \Gamma(\Delta, \phi) \vdash_{\mathcal{I}\mathrm{G}} [\phi]^2 M \quad \textit{for some } M.$$

3 Proposals

ICL is a combinatory version of higher-order minimal predicate logic which lacks a negation operator. However, the notion of negation is important when it comes to semantics of natural language. Therefore, we propose a new system $\mathcal{I}$exG, which is $\mathcal{I}$G extended with an operator of negation and the combinator ε for presuppositions.

3.1 Extension of PRED(exPRED)

We add the new symbols $\perp$ and ε to PRED.

Definition 16 (Syntax of exPRED**).** Given a signature $\Sigma = \langle S, F, (P_{()}, \dots)\rangle$ of PRED, the signature Σ' of exPRED is obtained by adding a operator $\perp$ to $P_{()}$ in P, i.e. $\Sigma' = \langle S, F, (P_{()} \cup \{\perp\}, \dots)\rangle$.

Definition 17 (Variables and Terms of exPRED**).** Given a signature Σ', a family of sorted variables are defined in the same way as Definition 3. The family of well-formed terms over Σ' and V are defined as in Definition 4.

[1] Lemma 2.13 (i)(iii) in [1].
[2] Proposition 2.14 (ii) in [1].

Definition 18 (Formulae of exPRED**).** Given a signature $\Sigma' = \langle S, F, P' \rangle$ and a family of sorted variables V, the set of well-sorted formulae over Σ' and V is defined as follows:

1. if $t_1 \in T_{s_1}, \ldots, t_n \in T_{s_n}$ and $p \in F_{(s_1,\ldots,s_n)}$, then $p(t_1, \ldots, t_n)$ is a well-formed formula.
2. if ϕ, ψ are well-formed formulae, then $\phi \supset \psi$ is a well-formed formula.
3. if $x^s \in V_s$ and ϕ is a well-formed formula, then $\forall x^s \phi$ is a well-formed formula.
4. if $x^s \in V_s$ and ϕ is a well-formed formula, then $\varepsilon x^s \phi \in T_s$.

Definition 19 (Natural deduction system for PRED**).** The natural deduction system for exPRED is obtained by adding the following inference rules to that of PRED in Definition 6.

$$\begin{array}{c} \Gamma \vdash \bot \supset \phi \\ \mathit{or} \\ \Gamma \vdash ((\phi \supset \bot) \supset \bot) \supset \phi \end{array} \quad + \quad \vdash \mathbf{P}(\varepsilon x \mathbf{P}(x))$$

As for the two axioms on the left side of Definition 19, if we add the upper one (EFQ) to PRED, exPRED becomes intuitionistic logic. If we instead add the lower one (DNE) to PRED, exPRED becomes classical logic.

Definition 20 (Logical operators). In exPRED, the logical operators ¬, ∧, ∨, ∃ are defined as follows.

$$\begin{aligned} \neg\phi &\overset{def}{\equiv} \phi \supset \bot \\ \phi \wedge \psi &\overset{def}{\equiv} (\phi \supset (\psi \supset \bot)) \supset \bot \\ \phi \vee \psi &\overset{def}{\equiv} (\phi \supset \bot) \supset \psi \\ \exists x \phi &\overset{def}{\equiv} \forall x(\phi \supset \bot) \supset \bot \end{aligned}$$

3.2 Extension of $\mathcal{I}$G($\mathcal{I}$exG)

Now, it is also necessary to extend $\mathcal{I}$G to translate PRED on it. We add new combinators and rules to the definition of $\mathcal{I}$G.

Definition 21 (Syntax of $\mathcal{I}$exG). $T = \Lambda(\Xi, \mathsf{L}, \bot, \varepsilon)$, the set of type-free lambda terms extended by the constants $\Xi, \mathsf{L}, \bot, \varepsilon$, is defined as follows. Let Var be a set of variables.

$$\begin{gathered} T ::= x \mid c \mid TT \mid \lambda x.T \\ (x \in Var, c \in \{\Xi, \mathsf{L}, \bot, \varepsilon\}) \end{gathered}$$

Definition 22 (Operators). Define the operators in $\mathcal{I}$exG as follow.

$$\begin{aligned} \neg\phi &\overset{def}{\equiv} \mathsf{F}\phi\bot \\ \phi \wedge \psi &\overset{def}{\equiv} \mathsf{F}(\mathsf{F}\phi(\mathsf{F}\psi\bot))\bot \\ \phi \vee \psi &\overset{def}{\equiv} \mathsf{F}(\mathsf{F}\phi\bot)\psi \\ \exists x \phi &\overset{def}{\equiv} \mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}\phi\bot)))\bot \end{aligned}$$

Definition 23 (Rules of ε)

$$(\varepsilon_{\mathsf{L}})\,\frac{\Gamma \vdash (\exists x X)Y}{\Gamma \vdash A(\varepsilon(\lambda x.X))} \qquad\qquad (G_{\varepsilon})\,\frac{}{\vdash P(\varepsilon P)\mathsf{E}}$$

Definition 24 (Rules of $\mathcal{I}$exG). Among the following rules, if we add (G_{EFQ}) to $\mathcal{I}$G, $\mathcal{I}$exG becomes intuitionistic logic, and if we add (G_{DNE}), $\mathcal{I}$exG becomes classical logic.

$$(G_{EFQ})\,\frac{\Gamma \vdash \mathsf{L}X}{\Gamma \vdash (\mathsf{F}\bot X)\mathsf{A}} \qquad\qquad (G_{DNE})\,\frac{\Gamma \vdash \mathsf{L}X}{\Gamma \vdash (\mathsf{F}(\neg\neg X)X)\mathsf{D}}$$

3.3 Translation from exPRED to $\mathcal{I}$exG

Translation rules from exPRED to $\mathcal{I}$exG are defined by extending the translation rules from PRED to $\mathcal{I}$G. We will show that the judgment in exPRED is sound in $\mathcal{I}$exG with respect to this translation.

Definition 25 (Translation to $\mathcal{I}$exG)
$\Lambda^{\Sigma}(\Xi, \mathsf{L}, \bot, \varepsilon)$ is the extension of $\Lambda(\Xi, \mathsf{L}, \bot, \varepsilon)$ by the signature Σ. As with Definition 13, the following maps are defined by Table 4 and Table 5.

$$\begin{aligned} [-]^2 &: \mathsf{Form}_{\mathrm{exPRED}} \to \mathsf{\Lambda}^{\mathsf{\Sigma}}(\Xi, \mathsf{L}, \bot, \varepsilon) \\ \Gamma &: \mathsf{Form}_{\mathrm{exPRED}} \to \text{illative contexts} \end{aligned}$$

Table 4. Translation of terms

t	$[t]^2$	$\Gamma(t)$
$x^{\mathbf{A}}$	x	Ax
$\mathbf{f}(t_1, \ldots, t_n)$	$f[t_1]^2 \ldots [t_n]^2$	$\oslash$
$\varepsilon x^{\mathbf{A}}\psi$	$\varepsilon(\lambda x.[\psi]^2)$	$\oslash$

Table 5. Translation of formulae

ϕ	$[\phi]^2$	$\Gamma(\phi)$
$\mathbf{P}(t_1, \ldots, t_n)$	$P[t_1]^2 \cdots [t_n]^2$	$\Gamma(t_1), \ldots, \Gamma(t_n)$
$\psi \supset \chi$	$\mathsf{F}[\psi]^2[\chi]^2$	$\Gamma(\psi), \Gamma(\chi)$
$\forall x^{\mathbf{A}}\psi$	$\mathsf{G}A_i(\lambda x.[\psi]^2)$	$\Gamma(\psi) - \{Ax\}$
$\bot$	$\bot$	$\oslash$

The following example are the translations of the signature in Example 2.

$$\begin{aligned} \Gamma^2_{\Sigma} &= \langle \mathsf{L}A, \mathsf{F}A\mathsf{L}P_1, \mathsf{F}A\mathsf{L}P_2, \ldots, Aa_1, Aa_2, \ldots, \mathsf{L}\bot \rangle \\ \Gamma^{2,+}_{\Sigma} &= \Gamma^2_{\Sigma}, Ax \qquad \text{where } x \in \mathit{Var}. \end{aligned}$$

Lemma 26. Let $\phi \in \mathsf{Form}_{\mathrm{exPRED}}$.

(1) If $t \in T_{\mathbf{A}}$, then in $\mathcal{I}$G one has $\Gamma(t), \Gamma^2_{\Sigma} \vdash A[t]^2$.
(2) $\Gamma^2_{\Sigma}, \Gamma(\phi) \vdash_{\mathcal{I}\mathrm{exG}} \mathsf{L}[\phi]^2$.

Theorem 27 (Soundness of the interpretation for exPRED**).** Let $\varDelta \cup \{\phi\} \subseteq \mathsf{Form}_{\text{exPRED}}$, then the following holds.

$$\varDelta \vdash_{\text{exPRED}} \phi \;\Rightarrow\; \varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta, \phi) \vdash_{\mathcal{I}\text{exG}} [\phi]^2 M \quad \textit{for some } M$$

Proof. The soundness of interpretation for PRED was already proved in [1]. Therefore, we show only the cases for new rules, EFQ and DNE. Before proving these, for that purpose we need the weakening rule, which is described as follows.

$$^{(w)}\dfrac{\varGamma \;\vdash\; X}{\varGamma, Y \;\vdash\; X}$$

This rule is admissible in $\mathcal{I}$exG (the proof is obvious) and we use it in the following proof of the soundness of EFQ, DNE, and the rule of ε.

<u>EFQ</u>

$$\begin{aligned}
&\varDelta \vdash_{\text{exPRED}} \bot \supset \phi\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta, (\bot \supset \phi)) \vdash_{\mathcal{I}\text{exG}} [\bot \supset \phi]^2 M \; \textit{for some } M \; (\because \textit{Theorem } 27)\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} (\mathsf{F}\bot[\phi]^2) M \; \textit{for some } M \qquad (\because \textit{Definition } 25)
\end{aligned}$$

$$^{(G_{EFQ})}\dfrac{^{(w)}\dfrac{^{(Lemma2(2))}\dfrac{}{\varGamma^{2,+}_{\varSigma}, \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} \mathsf{L}[\phi]^2}}{\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} \mathsf{L}[\phi]^2}}{\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} (\mathsf{F}\bot[\phi]^2)\mathsf{A}}$$

<u>DNE</u>

$$\begin{aligned}
&\varDelta \vdash_{\text{exPRED}} ((\phi \supset \bot) \supset \bot) \supset \phi\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta, (((\phi \supset \bot) \supset \bot) \supset \phi)) \vdash_{\mathcal{I}\text{exG}} [((\phi \supset \bot) \supset \bot) \supset \phi]^2 M \textit{for some } M\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} (\mathsf{F}(\neg\neg[\phi]^2)[\phi]^2) M \textit{for some } M
\end{aligned}$$

$$^{(G_{DNE})}\dfrac{^{(w)}\dfrac{^{(Lemma26(2))}\dfrac{}{\varGamma^{2,+}_{\varSigma}, \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} \mathsf{L}[\phi]^2}}{\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} \mathsf{L}[\phi]^2}}{\varGamma^{2,+}_{\varSigma}, [\varDelta]^2, \varGamma(\varDelta), \varGamma(\phi) \vdash_{\mathcal{I}\text{exG}} (\mathsf{F}(\neg\neg[\phi]^2)[\phi]^2)\mathsf{D}}$$

<u>Rule of ε</u>

$$\begin{aligned}
&\vdash_{\text{exPRED}} \mathbf{P}(\varepsilon x \mathbf{P}(x))\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma}, \varGamma(\mathbf{P}(\varepsilon x \mathbf{P}(x))) \vdash_{\mathcal{I}\text{exG}} [\mathbf{P}(\varepsilon x \mathbf{P}(x))]^2 M \textit{for some } M\\
\Rightarrow\; &\varGamma^{2,+}_{\varSigma} \vdash_{\mathcal{I}\text{exG}} (P(\varepsilon(\lambda x.P(x)))) M \; \textit{for some } M
\end{aligned}$$

$$^{(\beta\eta)}\dfrac{^{(w)}\dfrac{^{(G_{\varepsilon})}\dfrac{}{\vdash_{\mathcal{I}\text{exG}} (P(\varepsilon P))\mathsf{E}}}{\varGamma^{2,+}_{\varSigma} \vdash_{\mathcal{I}\text{exG}} (P(\varepsilon P))\mathsf{E}} \qquad P =_{\beta\eta} \lambda x.P(x)}{\varGamma^{2,+}_{\varSigma} \vdash_{\mathcal{I}\text{exG}} (P(\varepsilon(\lambda x.P(x))))\mathsf{E}}$$

■

4 Verification of the Proposed System

To show that proposition A is a presupposition of sentence S, we first show that A holds true in the proof that S is a wff.

Example 28. The king of France is not bald.

Firstly, we translate this sentence to the formula of exPRED, and let the semantic representation of “the king of France,” a noun phrase including a presupposition trigger, be $\varepsilon(\lambda x.koF(x))$.

$$\vdash_{\text{exPRED}} bald(\varepsilon(\lambda x.koF(x))) \supset \bot$$

Secondly, we translate this formula to the formula of $\mathcal{I}$exG by Lemma 26(2) and Definition 25.

$$\begin{aligned} &\varGamma^2_\Sigma, \varGamma(bald(\varepsilon(\lambda x.koF(x))) \supset \bot) \vdash_{\mathcal{I}\text{exG}} \mathsf{L}[bald(\varepsilon(\lambda x.koF(x))) \supset \bot]^2 \\ \Rightarrow\ &\varGamma^2_\Sigma \vdash_{\mathcal{I}\text{exG}} \mathsf{L}(\mathsf{F}(bald(\varepsilon(\lambda x.koF(x))))(\bot)) \end{aligned}$$

If we try to prove it first without adding anything to the left-side sequence, then the proof goes as follows:

$$(\mathsf{F}_\mathsf{L})\,\dfrac{(\in_e)\,\dfrac{\mathsf{L}(\bot) \in \varGamma^2_\Sigma, (bald(\varepsilon(\lambda x.koF(x))))y}{\varGamma^2_\Sigma, (bald(\varepsilon(\lambda x.koF(x))))y \vdash \mathsf{L}(\bot)} \qquad (\mathsf{F}_e)\,\dfrac{(\in_e)\,\dfrac{\mathsf{F}A\mathsf{L}(bald) \in \varGamma^2_\Sigma}{\varGamma^2_\Sigma \vdash \mathsf{F}A\mathsf{L}(bald)} \qquad (\varepsilon_\mathsf{L})\,\dfrac{\dfrac{*}{\varGamma^2_\Sigma \vdash (\exists x(koF(x)))Y}}{\varGamma^2_\Sigma \vdash A(\varepsilon(\lambda x.koF(x)))}}{\varGamma^2_\Sigma \vdash \mathsf{L}(bald(\varepsilon(\lambda x.koF(x))))}}{\varGamma^2_\Sigma \vdash \mathsf{L}(\mathsf{F}(bald(\varepsilon(\lambda x.koF(x))))(\bot))}$$

If we take a look at the part (*), where a failure occurs, it seems that the illative context $\varGamma^2_\Sigma$ has to contain some information from which $(\exists x(koF(x)))Y$ in the right-side sequence can be deduced. So, we add $(\exists x(koF(x)))Y$ to the left-side sequent and try to prove it once again.

$$(\mathsf{F}_e)\,\dfrac{(\in_e)\,\dfrac{\mathsf{F}A\mathsf{L}(bald) \in \varGamma^2_\Sigma, (\exists x(koF(x)))Y}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y \vdash \mathsf{F}A\mathsf{L}(bald)} \qquad (\varepsilon_\mathsf{L})\,\dfrac{(\in_e)\,\dfrac{(\exists x(koF(x)))Y \in \varGamma^2_\Sigma, (\exists x(koF(x)))Y}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y \vdash (\exists x(koF(x)))Y}}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y \vdash A(\varepsilon(\lambda x.koF(x)))}}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y \vdash \mathsf{L}(bald(\varepsilon(\lambda x.koF(x))))} \quad \ldots(\heartsuit)$$

$$(\mathsf{F}_\mathsf{L})\,\dfrac{(\in_e)\,\dfrac{\mathsf{L}(\bot) \in \varGamma^2_\Sigma, (\exists x(koF(x)))Y, (bald(\varepsilon(\lambda x.koF(x))))y}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y, (bald(\varepsilon(\lambda x.koF(x))))y \vdash \mathsf{L}(\bot)} \qquad \dfrac{}{(\heartsuit)}}{\varGamma^2_\Sigma, (\exists x(koF(x)))Y \vdash \mathsf{L}(\mathsf{F}(bald(\varepsilon(\lambda x.koF(x))))(\bot))}$$

This time it can be proved the proof diagram completes. A series of these operations the proof diagram completes. That the sentence does not hold true without a presupposition. In other words, it can be seen that “there is a unique king of France” is a presupposition of “the king of France is bald”. On the other hand, the following sentence is an example where “there is a king of France” is not a presupposition. This is predicted in our theory since we can prove it without including $(\exists x(koF(x)))Y$ in the left-sideof the sequent in this case, as demonstrated in Example 3.

Example 29

If there is a unique king of France, then the king of France is bald.

This sentence is translated into the following judgment of exPRED.

$$\vdash_{\text{exPRED}} (\forall x(koF(x) \supset \bot) \supset \bot) \supset (bald(\varepsilon(\lambda x.koF(x))))$$

Then we apply Lemma 26(2) and Definition 25 to this formula.

$$\begin{aligned}&\Gamma^2_{\mathsf{l},s}, \Gamma((\forall x(koF(x) \supset \bot) \supset \bot) \supset (bald(\varepsilon(\lambda x.koF(x)))))\\ &\quad\vdash_{\mathcal{I}\mathrm{exG}} \mathsf{L}[(\forall x(koF(x) \supset \bot) \supset \bot) \supset (bald(\varepsilon(\lambda x.koF(x))))]^2\\ \Rightarrow\ &\Gamma^2_{\mathsf{l},s} \vdash_{\mathcal{I}\mathrm{exG}} \mathsf{L}(\mathsf{F}(\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))(bald(\varepsilon(\lambda x.koF(x)))))\end{aligned}$$

$$(\in e)\,\dfrac{(\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y \in \Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y,\ Ax}{\Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y,\ Ax \vdash (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y}\ \ldots(\blacklozenge 1)$$

$$(\varepsilon_{\mathsf{L}})\,\dfrac{(\beta\eta)\,\dfrac{\dfrac{}{(\blacklozenge 1)} \qquad (def\exists)\,\dfrac{}{\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot) \ = \ \exists x(koF(x))}}{\Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y,\ Ax \vdash (\exists x(koF(x)))y}}{\Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y \vdash A(\varepsilon(koF(x)))}\ \ldots(\blacklozenge 2)$$

$$(\lambda x.(\mathsf{F}(koF(x))(\bot)))w =_{\beta\eta} \mathsf{F}(koF(w))(\bot)\ \ldots(\blacklozenge 3)$$

$$(\beta\eta)\,\dfrac{(\mathsf{F_L})\,\dfrac{(\in e)\,\dfrac{\mathsf{L}(\bot) \in \Gamma^2_{\Sigma}, Aw, (koF(w))v}{\Gamma^2_{\Sigma}, Aw, (koF(w))v \vdash \mathsf{L}(\bot)} \qquad (\mathsf{F}_e)\,\dfrac{Aw \in \Gamma^2_{\Sigma}, Aw}{\Gamma^2_{\Sigma}, Aw \vdash Aw}}{\Gamma^2_{\Sigma}, Aw \vdash \mathsf{L}(\mathsf{F}(koF(w))(\bot))} \qquad \dfrac{}{(\blacklozenge 3)}}{\Gamma^2_{\Sigma}, Aw \vdash \mathsf{L}((\lambda x.(\mathsf{F}(koF(x))))w)}\ \ldots(\blacklozenge 4)$$

$$(\mathsf{F_L})\,\dfrac{(\in e)\,\dfrac{\mathsf{L}(\bot) \in \Gamma^2_{\Sigma}, (\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))z}{\Gamma^2_{\Sigma}, (\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))z \vdash \mathsf{L}(\bot)} \qquad (\mathsf{G_L})\,\dfrac{\dfrac{}{(\blacklozenge 4)} \qquad (\in e)\,\dfrac{\mathsf{L}A \in \Gamma^2_{\Sigma}}{\Gamma^2_{\Sigma} \vdash \mathsf{L}A}}{\Gamma^2_{\Sigma} \vdash \mathsf{L}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))}}{\Gamma^2_{\Sigma} \vdash \mathsf{L}(\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))}\ \ldots(\blacklozenge 5)$$

$$(\mathsf{F_L})\,\dfrac{(\mathsf{F}_e)\,\dfrac{(\in e)\,\dfrac{\mathsf{F}A\mathsf{L}(bald) \in \Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y}{\Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y \vdash \mathsf{F}A\mathsf{L}(bald)} \qquad \dfrac{}{(\blacklozenge 2)}}{\Gamma^2_{\Sigma}, (\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))y \vdash \mathsf{L}(bald(\varepsilon(\lambda x.koF(x))))} \qquad \dfrac{}{(\blacklozenge 5)}}{\Gamma^2_{\Sigma} \vdash \mathsf{L}(\mathsf{F}(\mathsf{F}(\mathsf{G}A(\lambda x.(\mathsf{F}(koF(x))(\bot))))(\bot))(bald(\varepsilon(\lambda x.koF(x)))))}$$

5 Conclusion

In this paper, we defined the system $\mathcal{I}$exG, which is an extension of ICL with the symbol $\bot$ and the operator ε, in our analysis of presupposition in natural language. We also extended the language of many-order predicate calculus PRED with ε and use the resulting language exPRED to represent the meaning of sentences that includes presupposition triggers. Moreover, we extended the translation rules from PRED to ICL to the translation rules from exPRED to $\mathcal{I}$exG, and showed the soundness of the translation. Some empirical predictions were also demonstrated in which the presuppositions are properly projected or filtered.

The resulting theory is a proof-theoretic semantics of natural language fragments through the analysis of presupposition. We expect that the theory not only covers the empirical phenomena which have been explained in previous frameworks, but also has an advantage over them with respect to the computational aspect, which remains to be shown in the future work.

References

[1] Barendregt, H., Bunder, M., Dekkers, W.: Systems of illative combinatory logic complete for first-order propositional and predicate calculus. The Journal of Symbolic Logic 58(3), 769–788 (1993)

[2] Beaver, D.I.: Presupposition and Assertion in Dynamic Semantics. Studies in Logic, Language and Information. CSLI Publications & folli. (2001)

[3] Bekki, D.: Combinatory categorial grammar as a substrutural logic - preliminary remarks -. In: The Seventh International Workshop on Logic and Engeneering of Natural Language Semantics (LENLS 7), pp. 70–83 (2010)

[4] Carlström, J.: Interpreting descriptions in intensional type theory. Journal of Symbolic Logic 70(2), 488–514 (2005)

[5] Curry, H.B., Feys, R.: Combinatory logic, vol. 1. North-Holland, Amsterdam (1958)

[6] Francez, N., Dyckhoff, R.: Proof-Theoretic Semantics for a Natural Language Fragment. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10/11. LNCS (LNAI), vol. 6149, pp. 56–71. Springer, Heidelberg (2010)

[7] Gazdar, G.: Pragmatics: Implicature, Presupposition, and Logical Form. Academic Press, London (1979)

[8] Karttunen, L.: Presuppositions and liguistic contexxt. Theoretical Linguistics 1, 181–193 (1974)

[9] Mineshima, K.: A Presuppositional Analysis of Definite Descriptions in Proof Theory. In: Satoh, K., Inokuchi, A., Nagao, K., Kawamura, T. (eds.) JSAI 2007. LNCS (LNAI), vol. 4914, pp. 214–227. Springer, Heidelberg (2008)

[10] Montague, R.: The proper treatment of quantification in ordinary english. In: Hintikka, J., Moravcsic, J., Suppes, P. (eds.) Approaches to Natural Language, pp. 221–242. Reidel, Dordrecht (1973)

[11] Ozaki, H., Bekki, D.: Extractability as deduction theorem in subdirectional combinatory logic. In: The Eighth International Workshop on Logic and Engeneering of Natural Language Semantics (LENLS 8), pp. 80–93 (2011)

[12] Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)

[13] van der Sandt, R.: Presupposition projection as anaphora resolution. Journal of Semantics 9, 333–377 (1992)

[14] van der Sandt, R., Geurts, B.: Presupposition, Anaphora, and Lexical Content. In: Herzog, O., Rollinger, C.-R. (eds.) LILOG 1991. LNCS, vol. 546, pp. 259–296. Springer, Heidelberg (1991)

[15] Soames, S.: A projection problem for speaker presuppositions. Linguistic Inquiry 10, 623–666 (1979)

[16] Stalnaker, R.: Pragmatic presupposition. In: Munitz, M.K., Unger, D.K. (eds.) Semantics and Philosophy, pp. 197–213. New York University Press (1974)

[17] Steedman, M.J.: The Syntactic Process (Language, Speech, and Communication). The MIT Press, Cambridge (2000)

[18] Strawson, P.F.: On referring. Mind, New Series 59(235), 320–344 (1950)

[19] Sundholm, G.: Proof theory and meaning. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, 2nd edn., vol. 9, pp. 165–198. Kluwer Academic Publishers (2002)

Abstract Automata and a Normal Form for Categorial Dependency Grammars

Boris Karlov[⋆]

Tver State Universty, Tver, Russia, 170000

Abstract. Categorial Dependency Grammars (CDG) studied in this paper are categorial grammars expressing projective and discontinuous dependencies, stronger than cf-grammars and presumably nonequivalent to mild context-sensitive grammars. We define a normal form of CDG similar to Greibach normal form for cf-grammars and propose an effective algorithm which transforms any CDG into an equivalent CDG in the normal form. A class of push-down automata with independent counters is defined and it is proved that they accept the class of CDG-languages. We present algorithms that transform any CDG into an automaton and vice versa.

Keywords: categorial grammars, dependency grammars, projective and discontinuous dependencies, normal form of categorial dependency grammars, push-down automata with independent counters

1 Introduction

The theories of natural language syntax based on the notion of *dependency* have an old tradition. Tesnière [10] was the first who systematically described the structure of the sentence in terms of named relations between the words. When two words w_1 and w_2 are connected in the sentence with the dependency d (denoted $w_1 \xrightarrow{d} w_2$), w_1 is the *governor*, and w_2 is the *subordinate word.* Informally, the dependency d places restrictions on the grammatical and lexical properties of w_1 and w_2, on their order, context etc. In a whole this means that "w_1 governs w_2". In most usual sentences of English or Russian the dependency structure is *projective.* In particular, this means that the dependencies do not intersect. Most of the grammars generating dependency trees only deal with projective structures. On the other hand, the sentences with *nonprojective* dependency structures are not uncommon.

The classical cf-grammars ([1,8]) as well as usual categorial grammars ([2,8]) are unable to find in the sentences the discontinuous dependencies. In [7] A.Dikovsky proposed to specify long distance discontinuous dependencies by polarized dependency types (valencies). In [5] a new type of grammars was defined — categorial dependency grammars (CDG). Their peculiarity is that they

[⋆] This work was sponsored by the Russian Fundamental Studies Foundation (Grants 10-01-00532a and 12-01-00244).

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 86–102, 2012.

can find discontinuous dependencies using the valencies. A positive valency specifies the name and the direction of an outgoing discontinuous dependency. The corresponding negative valency with the same name has the opposite direction and specifies the end of this incoming dependency. In the mentioned papers results on the expressivity of CDGs were obtained and an analysis algorithm was developed. In [6] some extension of CDG was studied. In particular, it was proved that the set of languages generated by the extended CDG is an abstract family of languages.

This paper continues the research of the properties of the CDG. In Sect. 2 we give the exact definitions of CDG and CDG-languages. In Sect. 3 we define the normal form of the CDG, analogous to the Greibach normal form ([1,8,9]) for cf-grammars. It is shown how one can build for every CDG an equivalent CDG in normal form. In Sect. 4 we introduce the notion of the push-down automaton with independent counters. Informally, the push-down automaton with independent counters is a usual push-down automaton that also has several counters. These counters serve to process the polarized valencies. We prove that the push-down automata with counters accept exactly the CDG-languages.

2 Main Definitions

In order to formalize the linguistic notion of syntactic type, we use the notion of category. Let $\mathbf{C}$ be a nonempty finite set of *elementary categories* (e.g. subject, predicate, complement). The elementary categories can be iterated: for $C \in \mathbf{C}$, C^* means a corresponding *iterated category*. The set of all iterated categories will be denoted $\mathbf{C}^*$. Elementary and iterated categories are combined in *local categories* with the constructors \ and /.

Definition 1. *The set of local categories $LCat(\mathbf{C})$ is a minimal set, such that:*
1) $\mathbf{C} \cup \{\varepsilon\} \subseteq LCat(\mathbf{C})$, where ε is an empty symbol;
2) if $\alpha \in LCat(\mathbf{C}), A \in \mathbf{C} \cup \mathbf{C}^*$, then $[A\backslash\alpha], [\alpha/A] \in LCat(\mathbf{C})$.

We suppose that the constructors \ and / are associative, e.g. $[B\backslash[A/C]] = [[B\backslash A]/C] = [B\backslash A/C]$. Therefore every local category γ can be represented in the form $\gamma = [L_k\backslash \dots \backslash L_1\backslash C/R_1/\dots/R_m]$.

To describe the discontinuous dependencies between the words in the sentence the notions of polarity and polarized valency were introduced in [7]. A *polarity* v is an element of the set $V = \{\searrow, \swarrow, \nwarrow, \nearrow\}$. For every polarity v there exists a *dual* polarity $\breve{v}$:

$$\breve{\nearrow} = \searrow, \breve{\nwarrow} = \swarrow, \breve{\swarrow} = \nwarrow, \breve{\searrow} = \nearrow$$

A *polarized valency* β is an expression of the form vC, where $v \in V, C \in \mathbf{C}$. We denote the set of all polarized valencies $V(C)$. The following subsets can be considered according to the types of the polarities:

$\nwarrow\mathbf{C} = \{\nwarrow C \mid C \in \mathbf{C}\}$, $\nearrow\mathbf{C} = \{\nearrow C \mid C \in \mathbf{C}\}$,
$\searrow\mathbf{C} = \{\searrow C \mid C \in \mathbf{C}\}$,

$\nwarrow\mathbf{C} = \{\nwarrow C \mid C \in \mathbf{C}\}$,
$V^{-}(\mathbf{C}) = \searrow\mathbf{C} \ \cup \ \swarrow\mathbf{C}$, $V^{+}(\mathbf{C}) = \nwarrow\mathbf{C} \cup \nearrow\mathbf{C}$,
$V^{l}(\mathbf{C}) = \nearrow\mathbf{C} \cup \swarrow\mathbf{C}$, $V^{r}(\mathbf{C}) = \nwarrow\mathbf{C} \cup \searrow\mathbf{C}$.

A *potential* is a sequence of polarized valencies. A potential θ is *balanced*, if each of its projection on $\{v, \breve{v}\}$, where $v \in V^{l}(\mathbf{C})$, is a correct bracketed sequence. The following string is an example of a balanced potential: $\nearrow A \nearrow B \searrow A \searrow B$. The set of all potentials is denoted $Pot(\mathbf{C})$.

The categories are built from local categories and potentials.

Definition 2. *A category γ is an expression of the form α^{θ}, where $\alpha \in LCat(\mathbf{C})$, $\theta \in Pot(\mathbf{C})$. The set of all categories is denoted $Cat(\mathbf{C})$.*

The dependency calculus is defined on the set of categories. It consists of the following rules.

Definition 3. *Let Γ_1, Γ_2 be strings of categories $Cat(\mathbf{C})^{*}$, $\theta, \theta_1, \theta_2, \theta_3$ be potentials, α be a local category from $LCat(\mathbf{C})$.*

Local dependency rules:

$L^{l} : \Gamma_1 [C]^{\theta_1} [C\backslash\alpha]^{\theta_2} \Gamma_2 \vdash \Gamma_1 [\alpha]^{\theta_1\theta_2} \Gamma_2$
$L^{r} : \Gamma_1 [\alpha/C]^{\theta_1} [C]^{\theta_2} \Gamma_2 \vdash \Gamma_1 [\alpha]^{\theta_1\theta_2} \Gamma_2,$

where $C \in \mathbf{C} \cup \{\varepsilon\}$

Iterated dependency rules:

$I^{l} : \Gamma_1 [C]^{\theta_1} [C^{*}\backslash\alpha]^{\theta_2} \Gamma_2 \vdash \Gamma_1 [C^{*}\backslash\alpha]^{\theta_1\theta_2} \Gamma_2$
$I^{l}_{0} : \Gamma_1 [C^{*}\backslash\alpha]^{\theta} \Gamma_2 \vdash \Gamma_1 [\alpha]^{\theta} \Gamma_2$
$I^{r} : \Gamma_1 [\alpha/C^{*}]^{\theta_1} [C]^{\theta_2} \Gamma_2 \vdash \Gamma_1 [\alpha/C^{*}]^{\theta_1\theta_2} \Gamma_2$
$I^{r}_{0} : \Gamma_1 [\alpha/C^{*}]^{\theta} \Gamma_2 \vdash \Gamma_1 [\alpha]^{\theta} \Gamma_2,$

where $C \in \mathbf{C}$.

Discontinuous dependency rule:

$D : \Gamma_1 \alpha^{\theta_1 \beta \theta_2 \breve{\beta} \theta_3} \Gamma_2 \vdash \Gamma_1 \alpha^{\theta_1\theta_2\theta_3} \Gamma_2,$

where the valencies $(\beta, \breve{\beta})$ form a correct pair and θ_2 does not contain $\beta, \breve{\beta}$.

When one of these rules is applied, an edge is added into the dependency structure. This edge goes form the governor to the subordinate word and it is labeled with the name of the cancelled category.

This calculus induces the provability relation on the strings of dependency types. We denote this relation $\vdash^{R}$, where R is the name of one of the rules, or simply $\vdash$, if the name of the rule is not important. If Γ_2 is obtained from Γ_1 in n steps, then we write $\Gamma_1 \vdash^{n} \Gamma_2$. $\vdash^{*}$ denotes reflexive transitive closure of the relation $\vdash$ on the set $Cat(\mathbf{C})^{*}$.

Definition 4. *A categorial dependency grammar (CDG) is a system $G = \langle W, \mathbf{C}, S, \delta \rangle$, where:*

W is a finite set of symbols,
$\mathbf{C}$ is a finite set of elementary categories,
S is the selected from $\mathbf{C}$ main category,
δ is the lexicon, a function on W, that maps each symbol $w \in W$ on the finite set $\delta(w) \subseteq Cat(\mathbf{C})$ of its possible categories.

Let $s = w_1 w_2 \ldots w_n$ be a word. Let us denote $\delta(s) = \delta(w_1)\delta(w_2)\ldots\delta(w_n)$.

Definition 5. *The CDG G generates the language $L(G)$, consisting of all the words $s \in W^*$, such that there exists a string of categories $\Gamma \in \delta(s)$ such that $\Gamma \vdash^* S$.*

Example 1. Let us consider the language $L = \{\, a^n b^n c^n \mid n > 0 \,\}$. It is well known that this language is not context-free ([1,8]). But this language is generated by the following CDG ([4,5]):

$a \mapsto [A]^{\swarrow A}, [A\backslash A]^{\swarrow A}$
$b \mapsto [B/C]^{\nwarrow A}, [A\backslash S/C]^{\nwarrow A}$
$c \mapsto [C], [B\backslash C]$.

The following string of categories can be assigned to the word *aaabbbccc*: $[A]^{\swarrow A}[A\backslash A]^{\swarrow A}[A\backslash A]^{\swarrow A}[A\backslash S/C]^{\nwarrow A}$ $[B/C]^{\nwarrow A}[B/C]^{\nwarrow A}[C][B\backslash C][B\backslash C]$. It can be cancelled to S (see [5]).

The dependency structure for the word *aaabbbccc* is shown on Fig. 1.

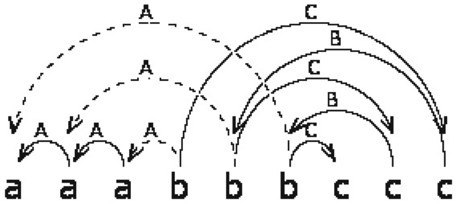


Fig. 1. Dependency structure for aaabbbccc

It is not difficult to see that local rules of cancellation $L^l, L^r, I^l, I^l_0, I^r, I^r_0$ affect only the local part of the category, and the rule of cancellation for the potential D changes only the potential. This allows to separate the analysis of words in the CDG into two independent tests: the first one in terms of the local categories and the local rules of cancellation, and the second one in terms of balanced potential. The analysis theorem for CDG was proved in [5].

Theorem 1. *The word $s = w_1 w_2 \ldots w_n$ belongs to the language $L(G)$, generated by the CDG $G = \langle W, \mathbf{C}, S, \delta\rangle$,iff there exists a string of categories $\Gamma = \alpha_1^{\theta_1} \ldots \alpha_n^{\theta_n}$, where $\alpha_i^{\theta_i} \in \delta(w_i)$, such that $\alpha_1 \ldots \alpha_n \vdash^* S$ and $\theta_1 \ldots \theta_n$ is balanced.*

3 Normal Form of CDG

It is known that for cf-grammars there are different special forms (Chomsky normal form, Greibach normal form ([1,8,9])) which simplify the analysis of the languages generated by them. In this section we define a normal form for the CDG which is analogous to Greibach normal form and we show how to build for an arbitrary grammar a weakly equivalent, i.e. generating the same language, grammar in this normal form.

Definition 6. (Greibach normal form) *A cf-grammar $G = \langle \Sigma, N, S, R \rangle$ is in Greibach normal form if each rule from R is of one of the following forms:*
1) $S \to \varepsilon$ if S is not present in the right parts of the rules,
2) $A \to x\alpha$, where $x \in \Sigma$, $A \in N$, $\alpha \in N^*$, $|\alpha| \leq 2$.

It is known that it is possible for each cf-grammar to build an equivalent cf-grammar in Greibach normal form. Let us define the normal form for the CDG in the following way.

Definition 7. *We shall say that a CDG $G = \langle W, \mathbf{C}, S, \delta \rangle$ is a grammar in normal form if all its categories have one of the forms $[X]^\theta$, $[X/Y]^\theta$, $[X/Y/Z]^\theta$, where X, Y, Z are elementary categories, θ is a potential.*

It is not difficult to see that the categorial grammar built from the cf-grammar in Greibach normal form satisfies Definition 7 in which the potentials are omitted (cf. Definition 10).

Example 2. Let us consider the language $L = \{\, a^n b^n c^n \mid n > 0 \,\}$. The CDG for this language from Sect. 2 is not in normal form because it has the category $[B\backslash C]$. The following grammar is the CDG for L in normal form.

$a \mapsto [S/A]^{\nearrow A}, [A/A]^{\nearrow A}, [A/B]^{\nearrow A}$
$b \mapsto [B/C/B]^{\searrow A}, [B/C]^{\searrow A}$
$c \mapsto [C]$

In order to transform an arbitrary CDG into a CDG in the normal form we define an auxiliary cf-grammar and we prove that the eliminations performed by the CDG can be modeled by this grammar.

Definition 8. *Let $G = \langle W, \mathbf{C}, S, \delta \rangle$ be a CDG. We denote the cf-grammar $G' = \langle \Sigma, N, S, R \rangle$ as $CF(G)$, where:*

$\Sigma = \{\, w^\theta \mid w \mapsto [\alpha]^\theta \in \delta$ for some $\alpha \,\}$;

N is the set of all local subcategories from δ ;

R is defined in the following way:

$[\alpha] \to w^\theta \in R \Leftrightarrow w \mapsto [\alpha]^\theta \in \delta$
$[\alpha] \to [A][A\backslash\alpha] \in R \Leftrightarrow [A\backslash\alpha] \in N$
$[\alpha] \to [\alpha/A][A] \in R \Leftrightarrow [\alpha/A] \in N$
$[\alpha] \to [A^*\backslash\alpha] \in R \Leftrightarrow [A^*\backslash\alpha] \in N$

$[A^*\backslash\alpha] \to [A][A^*\backslash\alpha] \in R \Leftrightarrow [A^*\backslash\alpha] \in N$
$[\alpha] \to [\alpha/A^*] \in R \Leftrightarrow [\alpha/A^*] \in N$
$[\alpha/A^*] \to [\alpha/A^*][A] \in R \Leftrightarrow [\alpha/A^*] \in N$

The words in the new alphabet Σ can be divided into two parts: the word in the original alphabet W and the potential.

Definition 9. *Let $u = w_1^{\theta_1} \dots w_n^{\theta_n}$ be a word in Σ. Then $word(u) = w_1 \dots w_n$ and $pot(u) = \theta_1 \dots \theta_n$.*

The connection of the original CDG with the cf-grammar built from it is expressed in the following lemma.

Lemma 1. *Let G be a CDG, $G' = CF(G)$, $\alpha \in N$. Then $\alpha \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta_n} \in \Sigma^*$ iff there exist categories $\gamma_1 = \alpha_1^{\theta_1} \in \delta(a_1), \dots, \gamma_n = \alpha_n^{\theta_n} \in \delta(a_n)$ such that $\gamma_1 \dots \gamma_n \vdash^*_G \alpha^{\theta_1 \dots \theta_n}$.*

Proof. The lemma is proved by induction on the length of the derivation. □

Corollary 1. *Let G be a CDG, $G' = CF(G)$. Then $w_1 \dots w_n \in L(G)$ iff $w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G')$ and the potential $\theta_1 \dots \theta_n$ is balanced.*

Proof. Let a word $u \in \Sigma^*$ be so that $pot(u)$ is balanced. By Lemma 1 $S \Rightarrow^*_{G'} u$ iff there exists a string of categories $\Gamma \in \delta(word(u))$ such that $\Gamma \vdash^*_G [S]^{pot(u)}$. The latter means by the analysis theorem that the word $word(u)$ belongs to the language generated by the grammar G. Thus the word w belongs to the language $L(G)$ iff it can be derived in G' with balanced potential. □

Now we describe the "inverse" procedure which builds a CDG from a cf-grammar.

Definition 10. *Let $G' = \langle \Sigma, N, S, R \rangle$ be a cf-grammar in Greibach normal form, where the elements of Σ are of the form w^θ. We denote by $CDG(G')$ the CDG $G = \langle W, N, S, \delta \rangle$, where $W = \{ w \mid w^\theta \in \Sigma \text{ for some } \theta \}$ and δ is defined in the following way:*

$w \mapsto [X]^\theta \in \delta \Leftrightarrow X \to w^\theta \in R,$
$w \mapsto [X/Y]^\theta \in \delta \Leftrightarrow X \to w^\theta Y \in R,$
$w \mapsto [X/Z/Y]^\theta \in \delta \Leftrightarrow X \to w^\theta Y Z \in R.$

A property analogous to Lemma 1 holds.

Lemma 2. *Let $G = CDG(G')$. Then $X \Rightarrow^*_{G'} a_1^{\theta_1} \dots a_n^{\theta_n} \in \Sigma^*$ iff there exist categories $\gamma_1 = \alpha_1^{\theta_1} \in \delta(a_1), \dots \gamma_n = \alpha_n^{\theta_n} \in \delta(a_n)$ such that $\gamma_1 \dots \gamma_n \vdash^*_G [X]^{\theta_1 \dots \theta_n}$.*

Proof. The lemma is proved by induction on the length of u. □

Corollary 2. *Let G' be a cf-grammar in Greibach normal form, $G = CDG(G')$. Then $w_1 \dots w_n \in L(G)$ iff $w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G')$ and the potential $\theta_1 \dots \theta_n$ is balanced.*

Proof. Let the word $u \in \Sigma^*$ be such that $pot(u)$ is balanced. By Lemma 2 $[S] \Rightarrow^*_{G'} u$ iff there exists a string of categories $\Gamma \in \delta(word(u))$ such that $\Gamma \vdash^*_G [S]^{pot(u)}$. The latter means by the analysis theorem that the word $word(u)$ belongs to the language generated by the grammar G. □

Now we can prove the theorem about the possibility of transformation of each CDG to normal form.

Theorem 2. *For each CDG $G = \langle W, \mathbf{C}, S, \delta\rangle$ there exists a CDG $G' = \langle W, \mathbf{C}', S, \delta'\rangle$ in normal form such that $L(G) = L(G')$ and G' is of polynomial size relatively to the size of G.*

Proof. Firstly we build from the original grammar G the cf-grammar $G_1 = CF(G)$. Then we build the cf-grammar G_2 in Greibach normal form which is equivalent to G_1. As G' we take $CDG(G_2)$. The three following assertions hold.

1) $w_1 \dots w_n \in L(G) \Leftrightarrow w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G_1)$ and the potential $\theta_1 \dots \theta_n$ is balanced (by Corollary 1)

2) $w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G_1) \Leftrightarrow w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G_2)$

3) $w_1 \dots w_n \in L(G') \Leftrightarrow w_1^{\theta_1} \dots w_n^{\theta_n} \in L(G_2)$ and the potential $\theta_1 \dots \theta_n$ is balanced (by Corollary 2)

It follows from these assertions that $w_1 \dots w_n \in L(G) \Leftrightarrow w_1 \dots w_n \in L(G')$, i.e. $L(G) = L(G')$. According to the construction the grammar G' is in normal form.

In [3] it was established that for every cf-grammar Γ there exists an equivalent cf-grammar Γ' in Greibach normal form of size $O(|\Gamma|^4)$. Therefore G_2 has a size $O(|G_1|^4)$. It is easy to see that the construction of $CF(G)$ and $CDG(G_2)$ increases the size polynomially. Therefore, $|G'|$ is of polynomial size relatively to the size of G. □

Let us notice that there are two types of equivalence: strong and weak. Two grammars are strongly equivalent if they generate the same set of dependency structures. They are weakly equivalent if they generate the same language, but possibly different sets of structures. The normalization described in Theorem 2 changes the dependency structures. Indeed, a grammar in normal form cannot generate projective dependencies of the form $v_1 \leftarrow v_2$. Thus, a grammar in normal form is only weakly equivalent to the initial grammar.

4 Push-Down Automata with Independent Counters

In this section we show that CDG-languages can be accepted with special extensions of push-down automata.

Definition 11. *A push-down automaton with independent counters is a 7-tuple $M = \langle \Sigma, Q, Z, q_0, z_0, P, n\rangle$, where:*

Σ is an input alphabet,
Q is an alphabet of states,

Z is a stack alphabet,
$q_0 \in Q$ is an initial state,
$z_0 \in Z$ is an initial symbol of the stack,
P is a set of rules,
n is a number of counters.

The rules are of the form $\langle q, a, z, \langle q', \alpha, \bar{v} \rangle \rangle$, where $q, q' \in Q$, $a \in \Sigma \cup \{ \varepsilon \}$, $z \in Z$, $\alpha \in Z^*$, $\bar{v} = (v_1, \ldots, v_n)$ is a vector of integers.

Informally speaking, this is a push-down automaton additionally augmented with a finite number of counters. It uses its stack to check the elimination of local categories, and the counters correspond to different types of valencies.

Definition 12. *A configuration of the push-down automaton with independent counters $M = \langle \Sigma, Q, Z, q_0, z_0, P, n \rangle$ is a quadruple $\langle q, w, \gamma, \bar{u} \rangle$, where $q \in Q$, $w \in \Sigma^*$, $\gamma \in Z^*$, $\bar{u} = (u_1, \ldots, u_n)$ is a vector of nonnegative integers.*

We define a one-step transition: $\langle q, w, \gamma, \bar{u} \rangle \vdash^1_M \langle q', w', \gamma', \bar{u}' \rangle$ iff there exists a rule $\langle q, a, z, \langle q', \alpha, \bar{v} \rangle \rangle \in P$ such that the following three conditions are satisfied:

1) $w = aw'$,
2) $\gamma = z\gamma''$, $\gamma' = \alpha\gamma''$,
3) $\bar{u}' = \bar{u} + \bar{v}$.

If $\gamma = \varepsilon$ or some component of $\bar{u}'$ is negative, then the step cannot be made.
The relations $\vdash^n_M$ and $\vdash^*_M$ are defined as usual.

In fact, the numbers in the counters are the numbers of unpaired left valencies. The positive numbers in the rules correspond to the left valencies, and the negative numbers correspond to the right ones. The automaton works like a push-down automaton. Additionally it changes the values of the counters on every step, but the step itself is not influenced by these values, which means that the counters are independent.

The language accepted by the automaton M can be defined by emptying the stack and zeroing the counters.

Definition 13. *The word w is accepted by the push-down automaton with independent counters M iff there exists $q \in Q$ such that $\langle q_0, w, z_0, (0, \ldots, 0) \rangle \vdash^*_M \langle q, \varepsilon, \varepsilon, (0, \ldots, 0) \rangle$.*
The language accepted by the automaton is the set of all the words accepted by the automaton.

In this article we shall consider a special subclass of automata without empty loops.

Definition 14. *The push-down automaton with independent counters M is an automaton without empty loops if there are no states $q_1, \ldots q_k$ $(k > 1)$ such that $\langle q_i, \varepsilon, z_i, \langle q_{i+1}, \gamma_i, \bar{v}_i \rangle \rangle \in P$ for $1 \leq i < k$, $\langle q_k, \varepsilon, z_k, \langle q_1, \gamma_k, \bar{v}_k \rangle \rangle \in P$.*

Without this restriction a possible can occur, when the automaton performs ε-commands in a cycle and changes the counters. Then it can increase the counters by an unbounded amount without reading new symbols. But all potentials in the CDGs have finite length. The definition of an automaton without empty loops allows one to avoid this situation.

Example 3. Let us consider the language from Sect. 2: $L = \{ a^n b^n c^n \mid n > 0 \}$. It is accepted by the following automaton $M = \langle W, Q, Z, q_a, z_0, P, k \rangle$:
$W = \{ a, b, c \}$, $Q = \{ q_a, q_b, q_c \}$, $Z = \{ z_0, a \}$, $k = 1$.

$\langle q_a, a, z_0, \langle q_a, az_0, (1) \rangle \rangle$
$\langle q_a, a, a, \langle q_a, aa, (1) \rangle \rangle$
$\langle q_a, b, a, \langle q_b, \varepsilon, (0) \rangle \rangle$
$\langle q_b, b, a, \langle q_b, \varepsilon, (0) \rangle \rangle$

$\langle q_b, c, z_0, \langle q_c, z_0, (-1) \rangle \rangle$
$\langle q_c, c, z_0, \langle q_c, z_0, (-1) \rangle \rangle$
$\langle q_c, c, z_0, \langle q_c, \varepsilon, (-1) \rangle \rangle$

When the automaton reads the block of the symbols 'a', it counts their number in the stack and in the counter. When it reads the symbols 'b', it empties the stack and thus verifies that there are as many 'b' as 'a'. After this the automaton checks that the number of symbols 'c' is the same using the counter. The sequence of configurations for the word *aaabbbccc* is the following:

$\langle q_a, aaabbbccc, z_0, (0) \rangle \vdash \langle q_a, aabbbccc, az_0, (1) \rangle \vdash \langle q_a, abbbccc, aaz_0, (2) \rangle \vdash$
$\langle q_a, bbbccc, aaaz_0, (3) \rangle \vdash \langle q_b, bbccc, aaz_0, (3) \rangle \vdash \langle q_b, bccc, az_0, (3) \rangle \vdash$
$\langle q_b, ccc, z_0, (3) \rangle \vdash \langle q_c, cc, z_0, (2) \rangle \vdash \langle q_c, c, z_0, (1) \rangle \vdash \langle q_c, \varepsilon, \varepsilon, (0) \rangle$.

Now we prove that every automaton without empty loops can be transformed in such a way that it does not change its counters on ε-steps.

Lemma 3. *For every push-down automaton with independent counters and without empty loops M there exists an automaton without empty loops N, such that $L(M) = L(N)$ and N has no commands of the form $\langle q_1, \varepsilon, z, \langle q_2, \gamma, \bar{v} \rangle \rangle$, $\bar{v} \neq \bar{0}$.*

Proof. Let us consider an arbitrary push-down automaton without empty loops $M = \langle \Sigma, Q, Z, q_0, z_0, P, n \rangle$. We may suppose that the initial state q_0 cannot be revisited by the automaton. We build an auxiliary graph $G = (V, E)$, where $V = \{ \langle q, \varepsilon, z, \langle q', \gamma, \bar{v} \rangle \rangle \mid \langle q, \varepsilon, z, \langle q', \gamma, \bar{v} \rangle \rangle \in P \}$, $E = \{ (\langle q, \varepsilon, z, \langle q', \gamma, \bar{v} \rangle \rangle, \langle q', \varepsilon, z', \langle q'', \gamma', \bar{v}' \rangle \rangle) \mid \langle q, \varepsilon, z, \langle q', \gamma, \bar{v} \rangle \rangle, \langle q', \varepsilon, z', \langle q'', \gamma', \bar{v}' \rangle \rangle \in V$ for some z, γ and $\bar{v}$.

Let $p_1, p_2, \ldots, p_k$ be all paths in G. G is an acyclic graph, so the number of paths is finite. Let $p_i = \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1 \rangle \rangle, \ldots, \langle q_{m-1}, \varepsilon, z_{m-1}, \langle q_m, \gamma_m, \bar{v}_m \rangle \rangle$ be one of these paths. Now we extend this path in the following way. We take all rules of the form $\langle q', a, z', \langle q_1, \gamma', \bar{v}' \rangle \rangle$, $\langle q_m, b, z'', \langle q'', \gamma'', \bar{v}'' \rangle \rangle \in P$, where $a, b \neq \varepsilon$, $q', z', \gamma', \bar{v}', z'', q'', \gamma'', \bar{v}''$ are arbitrary. We add in all possible ways a rule of the first form in the beginning of the path and a rule of the second form in the end of the path. In particular, it is possible not to add any rules. The original path also remains. Thus, we obtain a finite set of "extended paths" $\{ r_1, \ldots, r_k \}$. Now we can build the automaton $N = \langle \Sigma, Q_N, Z, q_0, z_0, P_N, n \rangle$. Q_n is a set of states of N. It contains all states of the automaton M and some new states that are described below. First of all we add to N all the non-ε-rules from M. Let r_i be an "extended path". Four cases are possible.

1) $r_l = \langle q', a, z', \langle q_1, \gamma', \bar{v}' \rangle \rangle, \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1 \rangle \rangle, \ldots, \langle q_m, \varepsilon, z_m, \langle q_{m+1}, \gamma_m, \bar{v}_m \rangle \rangle,$ $\langle q_{m+1}, b, z'', \langle q'', \gamma'', \bar{v}'' \rangle \rangle$. For every $j = 1, \ldots, m$ we calculate the vector $\bar{s}_j =$

$\sum_{t=1}^{j} \bar{v}_t$. $\bar{s}_j$ shows the change of the counters after the first j ε-commands were performed. $\bar{s}_m$ is the total change of the counters after performing all ε-commands. Let $\bar{s}_j = (s_j^{(1)}, \ldots, s_j^{(n)})$. Let $x_i = \min\{ s_j^{(i)} \mid s_j^{(i)} < 0 \text{ and } j = 1, \ldots, n\}$, and $x_i = 0$ if all $s_j^{(i)}$ are nonnegative. $|x_i|$ is the necessary number in the i-th counter for succefully using the ε-rules of r_l. On the other hand it is also sufficient because x_i is the biggest decrease of the i-th counter. Now we take the two vectors $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (s_m^{(1)} - x_1, \ldots, s_m^{(n)} - x_n)$. The following rules are added to N: $\langle q', a, z', \langle q_1^l, \gamma', \bar{v}' + \bar{x}\rangle\rangle$, $\langle q_{m+1}^l, b, z'', \langle q'', \gamma'', \bar{v}'' + \bar{y}\rangle\rangle$, $\langle q_j^l, \varepsilon, z_j, \langle q_{j+1}^l, \gamma_j, \bar{0}\rangle\rangle$ for $j = 1, \ldots, m$.

2) $r_l = \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1\rangle\rangle, \ldots, \langle q_m, \varepsilon, z_m, \langle q_{m+1}, \gamma_m, \bar{v}_m\rangle\rangle, \langle q_{m+1}, b, z'', \langle q'', \gamma'', \bar{v}''\rangle\rangle$. If the state q_1 is not initial, then the state q_1 cannot be reached by the automaton M immediately after reading a symbol. Then this path can only be a part of longer path and all rules are added for this long path. But if the state q_1 is the initial state q_0, then this sequence of commands can be performed in the very beginning of the work of M, before it reads any symbol. As in the previous case we calculate the vectors $\bar{x}$ and $\bar{y}$. But in the beginning all the counters contain zeros, so if $\bar{x} \neq \bar{0}$, then the automaton is unable to perform this sequence of commands. Therefore if $\bar{x} \neq \bar{0}$, we do not add new commands. And if $\bar{x} = \bar{0}$, then we add the following commands: $\langle q_{m+1}^l, b, z'', \langle q'', \gamma'', \bar{v}'' + \bar{y}\rangle\rangle$, $\langle q_j^l, \varepsilon, z_j, \langle q_{j+1}^l, \gamma_j, \bar{0}\rangle\rangle$ for $j = 1, \ldots, m$.

3) $r_l = \langle q', a, z', \langle q_1, \gamma', \bar{v}'\rangle\rangle, \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1\rangle\rangle, \ldots, \langle q_m, \varepsilon, z_m, \langle q_{m+1}, \gamma_m, \bar{v}_m\rangle\rangle$. This case is similar to the previous one. If an ε-rule can be applied after $\langle q_m, \varepsilon, z_m, \langle q_{m+1}, \gamma_m, \bar{v}_m\rangle\rangle$, then this path is a part of a longer one and all necessary constructions are performed for this longer path. If no rule can be applied in the end, then this means that the automaton reached the end of the word and now finishes its work. Again we calculate the vectors $\bar{x}$ and $\bar{y}$. But for the automaton to accept the word, all the counters must be zeros. It is impossible if $\bar{y} \neq \bar{0}$. In this case no commands are created. And if $\bar{y} = \bar{0}$, then we add the following rules: $\langle q', a, z', \langle q_1^l, \gamma', \bar{v}' + \bar{x}\rangle\rangle$, $\langle q_j^l, \varepsilon, z_j, \langle q_{j+1}^l, \gamma_j, \bar{0}\rangle\rangle$ for $j = 1, \ldots, m$.

4) $r_l = \langle q_1, \varepsilon, z_1, \langle q_2, \gamma_1, \bar{v}_1\rangle\rangle, \ldots, \langle q_m, \varepsilon, z_m, \langle q_{m+1}, \gamma_m, \bar{v}_m\rangle\rangle$. Like in the previous cases we only need to consider the situation when q_1 is the initial state and the automaton finishes its work. But then we can directly perform the commands of r_l and see if M accepts ε. If $\varepsilon \in L(M)$, then we add the new command $\langle q_0, \varepsilon, z_0, \langle q_0, \varepsilon, \bar{0}\rangle\rangle$, otherwise we do nothing.

Now we prove that $L(M) = L(N)$.

It follows from the fourth case of construction that $\varepsilon \in L(M)$ iff $\varepsilon \in L(N)$. Let w be nonempty.

1) $L(M) \subseteq L(N)$. Let $w = a_1 \ldots a_k \in L(M)$. Then there exists a sequence of configurations $\langle q_1, w_1, \gamma_1, \bar{v}_1\rangle, \ldots, \langle q_l, w_l, \gamma_l, \bar{v}_l\rangle$, where $q_1 = q_0$, $w_1 = w$, $\gamma_1 = z_0$, $w_l = \varepsilon$, $\bar{v}_l = \bar{0}$. We replace every "ε-segments" in this sequence like we did while building the automaton N. Since the new rules differ from the old ones only in the way they handle the counters, the automaton will empty the stack. But the blocks of the new rules change the counters exactly like the old rules did.

This means that after the automaton N reads w the counters will be zeros. Therefore $w \in L(N)$.

2) $L(N) \subseteq L(M)$. Let $w = a_1 \dots a_k \in L(N)$. Then there exists a sequence of configurations $\langle q_1, w_1, \gamma_1, \bar{v}_1 \rangle, \dots, \langle q_l, w_l, \gamma_l, \bar{v}_l \rangle$, where $q_1 = q_0$, $w_1 = w$, $\gamma_1 = z_0$, $w_l = \varepsilon$, $\bar{v}_l = \bar{0}$. We can replace each "ε-segment" with the sequence it was obtained from. This is always possible because different "ε-segments" have different states. Therefore the sequences of ε-rules considered while building N are all possible sequences. Then $w \in L(M)$. □

Now we prove a simple auxiliary lemma which will be useful later.

Definition 15. *A potential θ is simple if it is of the form $(\searrow A_1)^{k_1} \dots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \dots (\nearrow A_n)^{l_n}$.*

Lemma 4. *For every CDG $G = \langle W, \mathbf{C}, S, \delta \rangle$ there exists a CDG G' such that $L(G) = L(G')$ and all categories in G' have simple potentials.*

Proof. Let $\mathbf{C} = \{ C_1, \dots, C_n \}$. Firstly we replace each valency $\swarrow C_i$ by a valency $\nearrow C_i'$ and each valency $\nwarrow C_i$ by a valency $\searrow C_i'$, where C_i' is a new elementary category. This transformation will not change the language of the grammar G. The only difference is that the dependencies of the form $u \overset{C_i}{\leftarrow} v$ will be replaced by dependencies $u \overset{C_i'}{\dashrightarrow} v$ in the new dependency structures.

Let α^θ be an arbitrary category in the new grammar. Some of the valencies of its potential θ may form correct pairs. We remove all correct pairs from all categories of the grammar. This transformation does not affect the possibility of eliminating the potential, thus, the language of the grammar will remain the same. We denote this grammar G_1. No potential in G_1 can be of the form $\theta_1 \nearrow A \theta_2 \searrow A \theta_3$ because all correct pairs were removed. This means that we can put all the left valencies in the end of the potential keeping their order. And finally we can rearrange the left valencies and the right valencies so the potential will take on form $(\searrow A_1)^{k_1} \dots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \dots (\nearrow A_n)^{l_n}$. The resulting grammar G' is the desired one. □

We introduce two notations.

Let the potential θ be a prefix of a correct bracket word. We denote the vector of the numbers of unpaired left brackets as $c(\theta)$: $c(\theta) = (|\theta|_{\nearrow A_1} - |\theta|_{\searrow A_1}, \dots, |\theta|_{\nearrow A_n} - |\theta|_{\searrow A_n})$. Since θ is a prefix of a correct bracket word, all components of the vector $c(\theta)$ are nonnegative.

Let $\bar{u}$ be a vector of integers. Then $\theta(\bar{u})$ will mean the potential, "corresponding" to the vector $\bar{u}$: $\theta(\bar{u}) = \theta_1 \dots \theta_n$, where

$$\theta_i = \begin{cases} (\nearrow A_i)^{u_i} & \text{if } u_i > 0, \\ (\searrow A_i)^{-u_i} & \text{if } u_i < 0, \\ \varepsilon & \text{if } u_i = 0. \end{cases}$$

Theorem 3. *Every CDG-language is accepted by some push-down automaton with independent counters and without empty loops.*

Proof. Let $L = L(G)$, where $G = \langle W, \mathbf{C}, S, \delta\rangle$. By Lemma 4 we may suppose that G is a grammar in normal form with simple potentials. We shall build an automaton $M = \langle W, \{\, q\,\}, \mathbf{C}, q, S, P, k\rangle$, where q is a state of the automaton, $k = |\mathbf{C}|$ is the number of valency types in G. We define the rules as following.

1) $a \mapsto [X]^{\theta} \in \delta$, where $\theta = (\searrow A_1)^{k_1} \ldots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \ldots (\nearrow A_n)^{l_n}$.

a) For every i $k_i = 0$ or $l_i = 0$. Then $\langle q, a, X, \langle q, \varepsilon, \bar{u}\rangle\rangle \in P$. Here $u_i = -k_i$ if $l_i = 0$, and $u_i = l_i$ if $k_i = 0$.

b) For some i $k_i \neq 0$ and $l_i \neq 0$. Then $\langle q, \varepsilon, X, \langle q', X, \bar{u}\rangle\rangle \in P$, $\langle q', a, X, \langle q, \varepsilon, \bar{v}\rangle\rangle \in P$. Here $u_i = -k_i$, $v_i = l_i$ for $i = 1, \ldots, k$.

2) $a \mapsto [X/Y]^{\theta} \in \delta$, where $\theta = (\searrow A_1)^{k_1} \ldots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \ldots (\nearrow A_n)^{l_n}$.

a) For every i $k_i = 0$ or $l_i = 0$. Then $\langle q, a, X, \langle q, Y, \bar{u}\rangle\rangle \in P$. Here $u_i = -k_i$ if $l_i = 0$, and $u_i = l_i$ if $k_i = 0$.

b) For some i $k_i \neq 0$ and $l_i \neq 0$. Then $\langle q, \varepsilon, X, \langle q', X, \bar{u}\rangle\rangle \in P$, $\langle q', a, X, \langle q, Y, \bar{v}\rangle\rangle \in P$. Here $u_i = -k_i$, $v_i = l_i$ for $i = 1, \ldots, k$.

3) $a \mapsto [X/Y/Z]^{\theta} \in \delta$, where $\theta = (\searrow A_1)^{k_1} \ldots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \ldots (\nearrow A_n)^{l_n}$.

a) For every i $k_i = 0$ or $l_i = 0$. Then $\langle q, a, X, \langle q, ZY, \bar{u}\rangle\rangle \in P$. Here $u_i = -k_i$ if $l_i = 0$, and $u_i = l_i$ if $k_i = 0$.

b) For some i $k_i \neq 0$ and $l_i \neq 0$. Then $\langle q, \varepsilon, X, \langle q', X, \bar{u}\rangle\rangle \in P$, $\langle q', a, X, \langle q, ZY, \bar{v}\rangle\rangle \in P$. Here $u_i = -k_i$, $v_i = l_i$ for $i = 1, \ldots, k$.

The states q' are different for different categories of the grammar G. Let $\Gamma = CF(G) = \langle \Sigma, N, S, P\rangle$.

Lemma 5. *i) If $S \Rightarrow^{*}_{\Gamma} a_1^{\theta_1} \ldots a_j^{\theta_j} Z_1 \ldots Z_s$ and $\theta = \theta_1 \ldots \theta_j$ is a prefix of a correct bracket word, then $\langle q, a_1 \ldots a_j w, S, (0, \ldots, 0)\rangle \vdash^{*}_{M} \langle q, w, Z_1 \ldots Z_s, c(\theta)\rangle$.*

ii) Let $a_i^{\theta_i} \in \Sigma$ for $1 \leq i \leq j$, $\langle q, a_1 \ldots a_j w, S, (0, \ldots, 0)\rangle \vdash^{*}_{M} \langle q, w, Z_1 \ldots Z_s, c(\theta)\rangle$.

Then $S \Rightarrow^{*}_{\Gamma} a_1^{\theta_1} \ldots a_j^{\theta_j} Z_1 \ldots Z_s$ and $\theta = \theta_1 \ldots \theta_j$ is a prefix of a correct bracket word.

Proof. i) Induction on the length j of the derivation in Γ.

Base case. $j = 0$

By the definition $\langle q, w, S, (0, \ldots, 0)\rangle \vdash^{0}_{M} \langle q, w, S, (0, \ldots, 0)\rangle$.

Inductive step. Let $S \Rightarrow^{j}_{\Gamma} a_1^{\theta_1} \ldots a_j^{\theta_j} Z_1 \ldots Z_s$, then by the inductive hypothesis $\langle q, a_1 \ldots a_j a_{j+1} w, S, (0, \ldots, 0)\rangle \vdash^{*}_{M} \langle q, a_{j+1} w, Z_1 \ldots Z_s, c(\theta)\rangle$, where $\theta = \theta_1 \ldots \theta_j$. Then a rule of one of the following forms was used.

1) $Z_1 \to a_{j+1}^{\theta_{j+1}}$, $\theta_{j+1} = (\searrow A_1)^{k_1} \ldots (\searrow A_n)^{k_n} (\nearrow A_1)^{l_1} \ldots (\nearrow A_n)^{l_n}$

Then $S \Rightarrow^{j+1}_{\Gamma} a_1^{\theta_1} \ldots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} Z_2 \ldots Z_s$. If the case (a) took place in the construction of the automaton, then $\langle q, a_{j+1} w, Z_1 \ldots Z_s, c(\theta)\rangle \vdash^{1}_{M} \langle q, w, Z_2 \ldots Z_s,$

$c(\theta\theta_{j+1})\rangle$. If the case (b) took place, then let $\theta' = (\searrow A_1)^{k_1} \dots (\searrow A_n)^{k_n}$, $\theta'' = (\nearrow A_1)^{l_1} \dots (\nearrow A_n)^{l_n}$. By the construction $\langle q, a_{j+1}w, Z_1 \dots Z_s, c(\theta)\rangle \vdash_M^1 \langle q', a_{j+1}w, Z_1 \dots Z_s, c(\theta\theta')\rangle \vdash_M^1 \langle q, w, Z_2 \dots Z_s, c(\theta\theta'\theta'')\rangle$. In both cases $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, w, Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$.

2) $Z_1 \to a_{j+1}^{\theta_{j+1}} X$

Then $S \Rightarrow_\Gamma^{j+1} a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} X Z_2 \dots Z_s$ and $\langle q, a_{j+1}w, Z_1 \dots Z_s, c(\theta)\rangle \vdash_M^* \langle q, w, X Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$ (the latest property is proved exactly like before by studying two possible cases). Therefore $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, w, X Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$.

3) $Z_1 \to a_{j+1}^{\theta_{j+1}} XY$

Then $S \Rightarrow_\Gamma^{j+1} a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} XY Z_2 \dots Z_s$ and $\langle q, a_{j+1}w, Z_1 \dots Z_s, c(\theta)\rangle \vdash_M^* \langle q, w, XY Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$, therefore, $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, w, XY Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$.

ii) Induction on the number of steps of the automaton.

Base case. There are no steps. By definition $S \Rightarrow_\Gamma^0 S$.

Inductive step. Let $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, a_{j+1}w, Z_1 \dots Z_s, c(\theta)\rangle$, then by the inductive hypothesis $S \Rightarrow_\Gamma^* a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s$ and $\theta_1 \dots \theta_j$ is a prefix of a correct bracket word. Let us suppose firstly that the automaton used a non-ε-rule. There are three types of such rules.

1) $\langle q, a_{j+1}, Z_1, \langle q, \varepsilon, \bar{u}\rangle\rangle$

Then there is a rule $Z_1 \to a_{j+1}^{\theta_{j+1}}$ in Γ, where θ_{j+1} is the potential corresponding to $\bar{u}$. Then $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, w, Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$, and $S \Rightarrow_\Gamma^* a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s \Rightarrow_\Gamma^1 a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} Z_2 \dots Z_s$. $\theta_1 \dots \theta_{j+1}$ is a prefix of a correct bracket word, because θ_{j+1} does not have both left and right valencies of the same type.

2) $\langle q, a_{j+1}, Z_1, \langle q, X, \bar{u}\rangle\rangle$

Then there is a rule $Z_1 \to a_{j+1}^{\theta_{j+1}} X$ in Γ. Then $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^*$

$\langle q, w, X Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$, and $S \Rightarrow_\Gamma^* a_1^{\theta_1} \dots a_j^{\theta_j}\ Z_1 \dots Z_s \Rightarrow_\Gamma^1 a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} X Z_2 \dots Z_s$.

3) $\langle q, a_{j+1}, Z_1, \langle q, XY, \bar{u}\rangle\rangle$

Then there is a rule $Z_1 \to a_{j+1}^{\theta_{j+1}} XY$ in Γ. Then $\langle q, a_1 \dots a_j a_{j+1} w, S, (0, \dots, 0)\rangle \vdash_M^* \langle q, w, XY Z_2 \dots Z_s, c(\theta\theta_{j+1})\rangle$, and $S \Rightarrow_\Gamma^* a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s \Rightarrow_\Gamma^1 a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}}\ XY Z_2 \dots Z_s$.

Now let us consider the case, when the automaton used the ε-rule $\langle q, \varepsilon, Z_1, \langle q', Z_1, \bar{u}_1\rangle\rangle$. After this rule the automaton used a non-ε-rule. This rule is unambiguously defined by the state q', because according to the construction all such states are unique.

Let the rule be of the form $\langle q', a_{j+1}, Z_1, \langle q, \varepsilon, \bar{u}_2\rangle\rangle$. There is a rule $Z_1 \to a_{j+1}^{\theta'\theta''}$ in Γ, where θ' corresponds to $\bar{u}_1$, and θ'' corresponds to $\bar{u}_2$. Then $S \Rightarrow_{\Gamma}^{*} a_1^{\theta_1} \dots a_j^{\theta_j} Z_1 \dots Z_s \Rightarrow_{\Gamma}^{1} a_1^{\theta_1} \dots a_j^{\theta_j} a_{j+1}^{\theta_{j+1}} Z_2 \dots Z_s$. θ' contains only the right valencies and θ'' contains only the left valencies, therefore the potentials $\theta\theta'$ and $\theta\theta'\theta''$ are prefixed of correct bracket word. The cases of the rules $\langle q', a_{j+1}, Z_1, \langle q, X, \bar{u}_2\rangle\rangle$ and $\langle q', a_{j+1}, Z_1, \langle q, XY, \bar{u}_2\rangle\rangle$ are analogous. □

It follows from this lemma that $S \Rightarrow_{\Gamma}^{*} a_1^{\theta_1} \dots a_n^{\theta_n}$ iff $\langle q, a_1 \dots a_n, S, (0, \dots, 0)\rangle \vdash_M^{*} \langle q, \varepsilon, \varepsilon, c(\theta)\rangle$.

$a_1 \dots a_n \in L(G) \Leftrightarrow a_1^{\theta_1} \dots a_n^{\theta_n} \in L(\Gamma)$ and the potential $\theta_1 \dots \theta_n$ is balanced $\Leftrightarrow$

$\langle q, a_1 \dots a_n, S, (0, \dots, 0)\rangle \vdash_M^{*} \langle q, \varepsilon, \varepsilon, (0, \dots, 0)\rangle \Leftrightarrow a_1 \dots a_n \in L(M)$

This means that $L(G) = L(M)$. □

Remark 1. If the grammar G has no potentials with both $\nearrow A$ and $\searrow A$ for all $A \in \mathbf{C}$, then the case (b) in the construction is impossible. Therefore the equivalent automaton will have no ε-rules.

The push-down automata with independent counters possess properties analogous to the properties of the push-down automata.

Lemma 6. *Let* $\langle q, w, \alpha_1 \dots \alpha_k, \bar{u}\rangle \vdash_M^{n} \langle q', \varepsilon, \varepsilon, \bar{u}'\rangle$. *Then* $w = w_1 \dots w_k$, *and* $\langle q, w_1,\ \alpha_1, \bar{u}\rangle \vdash_M^{n_1} \langle q_1, \varepsilon, \varepsilon, \bar{u}_1\rangle$, $\dots$, $\langle q_{k-1}, w_k, \alpha_k, \bar{u}_{k-1}\rangle \vdash_M^{n_k} \langle q', \varepsilon, \varepsilon, \bar{u}'\rangle$, *where* $n = n_1 + \dots + n_k$.

Lemma 7. *Let* $\langle q_0, w_1, \alpha_1, \bar{u}_0\rangle \vdash_M^{n_1} \langle q_1, \varepsilon, \varepsilon, \bar{u}_1\rangle, \dots, \langle q_{m-1}, w_m, \alpha_m, \bar{u}_{m-1}\rangle \vdash_M^{n_m} \langle q_m, \varepsilon, \varepsilon, \bar{u}_m\rangle$. *Then* $\langle q_0, w_1 \dots w_m, \alpha_1 \dots \alpha_m, \bar{u}_0\rangle \vdash_M^{n_1 + \dots + n_m} \langle q_m, \varepsilon, \varepsilon, \bar{u}_m\rangle$.

Now we prove the theorem inverse to Theorem 3.

Theorem 4. *If a language L is accepted by a push-down automaton with independent counters and without empty loops, then $L - \{\, \varepsilon \,\}$ is a CDG-language.*

Proof. Let the language L be accepted by the automaton $M = \langle \Sigma, Q, Z, q_0, z_0, P, k\rangle$. By Lemma 3 we may suppose that ε-rules of M do not change the counters. We build a cf-grammar $\Gamma = \langle \Sigma, N, S, R\rangle$ from M:

$\Sigma = \{\, a^{\theta(\bar{u})} \mid \langle q, a, z, \langle q', \alpha, \bar{u}\rangle\rangle \in P \,\}$ $N = \{\, [qzq'] \mid q, q' \in Q, z \in Z \,\} \cup \{\, S \,\}$

R is defined in the following way:

$S \to [q_0 z_0 q]$ for all $q \in Q$
$[qzq'] \to a^{\theta(\bar{u})} \in R$ iff $\langle q, a, z, \langle q', \varepsilon, \bar{u}\rangle\rangle \in P$, $a \neq \varepsilon$
$[qzq_s] \to a^{\theta(\bar{u})}[q'z_1q_1][q_1z_2q_2] \dots [q_{s-1}z_sq_s] \in R$ for all $q_1, \dots, q_s \in Q$ iff $\langle q, a, z, \langle q', z_1 \dots z_s, \bar{u}\rangle\rangle \in P$, $a \neq \varepsilon$
$[qzq'] \to \varepsilon \in R$ iff $\langle q, \varepsilon, z, \langle q', \varepsilon, \bar{0}\rangle\rangle \in P$
$[qzq_s] \to [q'z_1q_1][q_1z_2q_2] \dots [q_{s-1}z_sq_s] \in R$ for all $q_1, \dots, q_s \in Q$ iff $\langle q, \varepsilon, z, \langle q', z_1 \dots z_s, \bar{0}\rangle\rangle \in P$, $a \neq \varepsilon$

The following property holds.

Lemma 8. *i) Let $\theta(\bar{u})\theta_1 \dots \theta_j$ be a prefix of a correct bracket word. If $[qzq'] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$, then $\langle q, a_1 \dots a_j, z, \bar{u}\rangle \vdash^*_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1 \dots \theta_j)\rangle$.*
ii) Let $a_i^{\theta_i} \in \Sigma$ for $1 \leq i \leq j$. If $\langle q, a_1 \dots a_j, z, \bar{u}\rangle \vdash^*_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1 \dots \theta_j)\rangle$, then $[qzq'] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$.

Proof. i) Induction on the length of the derivation in Γ.

Base case. Let the derivation be $[qzq'] \Rightarrow^1_\Gamma a_1^{\theta_1}$. Then there is a rule $\langle q, a_1, z, \langle q', \varepsilon, c(\theta_1)\rangle\rangle$ in the automaton. Therefore $\langle q, a_1, z, \bar{u}\rangle \vdash^1_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1)\rangle$. If the derivation is $[qzq'] \Rightarrow^1_\Gamma \varepsilon$, then there is a rule $\langle q, \varepsilon, z, \langle q', \varepsilon, \bar{0}\rangle\rangle$ in the automaton. Therefore $\langle q, \varepsilon, z, \bar{u}\rangle \vdash^1_M \langle q', \varepsilon, \varepsilon, \bar{u}\rangle$.

Inductive step. Let $[qzq'] \Rightarrow^k_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$ for some $k\rangle 1$.

1) The derivation is of the form $[qzq'] \Rightarrow^1_\Gamma a_1^{\theta_1}[q_1z_1q_2][q_2z_2q_3]\dots[q_sz_sq_{s+1}] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$, where $q_{s+1} = q'$. Therefore for all $i = 1, \dots, s$ $[q_iz_iq_{i+1}] \Rightarrow^{k_i}_\Gamma v_i \in \Sigma^*$, where $v_1v_2 \dots v_s = a_2^{\theta_2} \dots a_j^{\theta_j}$, and $k_1 + \dots + k_s = k$, $0\langle k_i \langle k$. By the inductive hypothesis $\langle q_i, word(v_i), z_i, \bar{u} + c(\theta_1 pot(v_1) \dots pot(v_{i-1}))\rangle \vdash^*_M \langle q_{i+1}, \varepsilon, \varepsilon, \bar{u} + c(\theta_1 pot(v_1) \dots pot(v_i))\rangle$. Instead of just $\bar{u}$ the first configuration contains $\bar{u} = c(\theta_1 pot(v_1) \dots pot(v_{i-1}))$. This is correct, because this vector is also a prefix of a correct bracket word. Then by Lemma 7 $\langle q, a_1 word(v_1) \dots word(v_s), z, \bar{u}\rangle \vdash^1_M \langle q_1, word(v_1) \dots word(v_s), z_1 \dots z_s,\ \bar{u} + c(\theta_1)\rangle \vdash^k_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1 pot(v_1) \dots pot(v_s))\rangle$, i.e. $\langle q, a_1 \dots a_j, z, \bar{u}\rangle \vdash^*_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1 \dots \theta_j)\rangle$.

2) The derivation is of the form $[qzq'] \Rightarrow^1_\Gamma [q_1z_1q_2][q_2z_2q_3]\dots[q_sz_sq_{s+1}] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$, where $q_{s+1} = q'$. The proof is analogous to the previous case. The only difference is that the first step of the automaton is $\langle q, a_1 \dots a_j, z, \bar{u}\rangle \vdash^1_M \langle q', a_1 \dots a_j, z_1 \dots z_s, \bar{u}\rangle$.

ii) Induction on the length of the derivation for the automaton.

Base case. If the derivation is of the form $\langle q, a_1, z, \bar{u}\rangle \vdash^1_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1)\rangle$, then the command $\langle q, a_1, z, \langle q', \varepsilon, c(\theta_1)\rangle\rangle$ was used. Then there is a rule $[qzq'] \to a_1^{\theta_1}$ in Γ, and $[qzq'] \Rightarrow^1_\Gamma a_1^{\theta_1}$. If the derivation is of the form $\langle q, \varepsilon, z, \bar{u}\rangle \vdash^1_M \langle q', \varepsilon, \varepsilon, \bar{u}\rangle$, then the command $\langle q, \varepsilon, z, \langle q', \varepsilon, \bar{0}\rangle\rangle$ was used. Then there is a rule $[qzq'] \to \varepsilon$ in Γ, and $[qzq'] \Rightarrow^1_\Gamma \varepsilon$.

Inductive step. Let us consider a sequence of configurations of length k.

1) Firstly let us suppose that on the first step a non-ε-rule $\langle q, a_1, z, \langle q_1, z_1 \dots z_s, \bar{v}\rangle\rangle$ was used. Then the sequence of configurations is of the form $\langle q, a_1 \dots a_j, z, \bar{u}\rangle$, $\langle q_1, a_2 \dots a_j, z_1 \dots z_s, \bar{u} + c(\theta_1)\rangle, \dots,$ $\langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta_1 \dots \theta_j)\rangle$, and there is a rule $[qzq'] \to a_1^{\theta_1}[q_1z_1q_2]\dots[q_sz_sq']$ in Γ (here $c(\theta_1) = \bar{v}$). By Lemma 6 the word $a_2 \dots a_j$ can be represented in the form $w_1 \dots w_s$, so that $\langle q_1, w_1, z_1, \bar{u} + c(\theta_1)\rangle \vdash^{k_1}_M \langle q_2, \varepsilon, \varepsilon, \bar{u} + c(\theta_1\theta'_1)\rangle, \dots, \langle q_s, w_s, z_s, \bar{u} + c(\theta_1\theta'_1 \dots \theta'_{s-1})\rangle \vdash^{k_s}_M \langle q', \varepsilon, \varepsilon, \bar{u} + c(\theta\theta'_1 \dots \theta'_s)\rangle$, where $\theta_1\theta'_1 \dots \theta'_s = \theta_1 \dots \theta_j$, $k_1 + \dots + k_s = k$, $0 < k_i < k$. By the inductive hypothesis $[q_iz_iq_{i+1}] \Rightarrow^{j_i}_\Gamma v_i$ $(q_{s+1} = q')$, and

$word(v_i) = w_i$, $pot(v_i) = \theta'_i$. Therefore there exists a derivation $[qzq'] \Rightarrow^1_\Gamma a_1^{\theta_1}[q_1 z_1 q_2][q_2 z_2 q_3] \dots [q_s z_s q'] \Rightarrow^*_\Gamma a_1^{\theta_1} v_1 \dots v_s = a_1^{\theta_1} \dots a_j^{\theta_j}$.

2) Now let us suppose that the rule $\langle q, \varepsilon, z, \langle q_1, z_1 \dots z_s, \bar{0}\rangle\rangle$ was used on the first step. Then there is a rule $[qzq'] \to [q_1 z_1 q_2] \dots [q_s z_s q']$ in Γ. This case differs from the previous one in the first rule used in the derivation for Γ. The derivation is of the form $[qzq'] \Rightarrow^1_\Gamma [q_1 z_1 q_2] \dots [q_s z_s q'] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_j^{\theta_j}$. □

Now we build the grammar in Greibach normal form, equivalent to Γ. If it has the rule $S \to \varepsilon$, we omit it. Let Γ' be the resulting grammar. Let $G = CDG(\Gamma')$.

$a_1 \dots a_n \in L(M) \Leftrightarrow \langle q_0, a_1 \dots a_n, z_0, (0, \dots, 0)\rangle \vdash^*_M \langle q, \varepsilon, \varepsilon, (0, \dots, 0)\rangle \Leftrightarrow$
$[q_0 z_0 q] \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_n^{\theta_n}$ and the potential $\theta_1 \dots \theta_n$ is balanced $\Leftrightarrow$
$S \Rightarrow^*_\Gamma a_1^{\theta_1} \dots a_n^{\theta_n}$ and the potential $\theta_1 \dots \theta_n$ is balanced $\Leftrightarrow$
$S \Rightarrow^*_{\Gamma'} a_1^{\theta_1} \dots a_n^{\theta_n}$ and the potential $\theta_1 \dots \theta_n$ is balanced $\Leftrightarrow a_1 \dots a_n \in L(G)$

This means that $L(M) = L(G)$. □

Remark 2. In the built grammar G no potential contains both left and right valencies of the same type.

Corollary 3. *For every push-down automaton with independent counters and without empty loops there exists an equivalent automaton without ε-commands.*

Proof. Let M be a push-down automaton without empty loops. We can build a grammar G, such that $L(G) = L(M)$ and no potential contains both left and right valencies of the same type (Remark 2). Then we build an automaton M' from the grammar G. M' is equivalent to M and it does not contain ε-commands (Remark 1). □

5 Conclusion

In this paper we studied some properties of the CDG-languages. In Sect. 3 we defined a normal form of CDG analogous to the Greibach normal form of cf-grammars. It was proved that for every CDG there exists a weakly equivalent CDG in this normal form. In Sect. 4 we introduced push-down automata with counters. They are usual push-down automata which also have several independent counters. We proved that the class of CDG-languages is equal to the class of languages accepted by these automata without the empty loops. There are some interesting open problems concerning CDGs.

1. Can the normal form be further simplified? For example, can all categories be represented in the form of $[A/B]^\theta$?
2. Is the restriction placed on the ε-commands of the automata essential? Can the automata without restrictions accept non-CDG-languages?
3. Does the number of counters influence the expressivity of the automata? We conjecture that there is a hierarchy on the number of counters, i.e. for every k there exists a language which is accepted by some automaton with k counters, but cannot be accepted by automata with a fewer number of counters.

4. The automata studied in this paper are nondeterministic. The deterministic automata with counters and the deterministic CDG-languages can be defined in a usual way. A study of deterministic languages is needed, in particular, their comparison with general CDG-languages.

Acknowledgements. The author is grateful to M. Dekhtyar and A. Dikovsky for useful discussions and the anonymous referees whose detailed comments helped greatly to improve the presentation of this paper.

References

1. Aho, A.V., Ullman, J.D.: The theory of parsing, translation and compiling. Parsing, vol. 1. Prentice-Hall, Inc., Englewood Cliffs (1972)
2. Bar-Hillel, Y., Gaifman, H., Shamir, E.: On categorial and phrase structure grammars. Bull. Res. Council Israel 9F, 1–16 (1960)
3. Blum, N., Koch, R.: Greibach Normal Form Transformation, Revisited. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 47–54. Springer, Heidelberg (1997)
4. Dekhtyar, M., Dikovsky, A.: Categorial Dependency Grammars. In: Moortgat, M., Prince, V. (eds.) Proc. of Int. Conf. on Categorial Grammars, Montpellier, pp. 76–91 (2004)
5. Dekhtyar, M., Dikovsky, A.: Generalized Categorial Dependency Grammars. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Trakhtenbrot/Festschrift. 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.: Grammars for Local and Long Dependencies. In: Proc. of the Intern. Conf. ACL 2001, Toulouse, France, pp. 156–163. ACL & Morgan Kaufman (2001)
8. Gladkij, A.V.: Formal Grammars and Languages, Moscow, "Nauka" (1973) (in Russian)
9. Greibach, S.A.: A new normal-form theorem for context-free phrase structure grammars. J. Assoc. Computing Machinery 12(1), 42–52 (1965)
10. Tesnière, L.: Éléments de syntaxe structurale. Librairie C. Klincksieck, Paris (1959)

Importing Montagovian Dynamics into Minimalism

Gregory M. Kobele

University of Chicago
kobele@uchicago.edu

Abstract. Minimalist analyses typically treat quantifier scope interactions as being due to movement, thereby bringing constraints thereupon into the purview of the grammar. Here we adapt De Groote's continuation-based presentation of dynamic semantics to minimalist grammars. This allows for a simple and simply typed compositional interpretation scheme for minimalism.

1 Introduction

Minimalist grammars [33] provide a mildly context sensitive perspective on mainstream chomskyian linguistic theory. Although semantic interpretation in chomskyian linguistics is traditionally viewed as operating on derived structures [17], this was faithfully reformulated in terms of a compositional semantics over derivation trees in [23]. There, in keeping with the treatment of pronouns as denoting variables, the standard semantic domains (of individuals E and of propositions T) were paramaterized with the set G of assignment functions, and a function $\leftthreetimes : \left[E^G \rightarrow T^G \rightarrow E^G \rightarrow T^G\right]$ which behaves in a manner similar to lambda abstraction was defined. Around the same time, a continuation-based reinterpretation of dynamic semantics using the simply typed lambda calculus was presented [8]. Instead of being variables, pronouns are treated there as (lifted) choice functions over *contexts*, which parameterize the type o of propositions.[1]

In this paper, we adopt the choice function treatment of pronouns [8], and reformulate the non-canonical semantics of minimalist grammars [23] in these terms. This allows for a simply typed and variable free (§3.3) presentation of minimalist semantics, within which constraints on quantifier scoping are most naturally formulated syntactically. This constrasts with a previous semantic interpretation scheme for (a logical reconstruction of) minimalist grammars [2,27], which, using the lambda-mu calculus [29] to represent the meanings of sentences, treated scope taking as a consequence of different reduction orders. (And which, as a consequence, was not able to account for the various seemingly syntactic constraints on scope.)

[1] A greater similarity with [23] emerges if we do not lift the pronouns of [8], but instead treat them as denoting functions from contexts to individuals. Then both propositions *and* individuals must be parameterized by contexts.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 103–118, 2012.

The main ‘data’ to be covered include the following sentences.

1. Every boy believed that every man believed that he smiled.
2. Some man believed that every woman smiled.
3. Some man believed every woman to have smiled.

The interest in these sentences is as follows. Sentence 1 has a reading in which the pronoun *he* is bound by the matrix subject *every boy*. This can be ‘straightforwardly’ dealt with if *he* denotes a variable, as long as this variable has a different name than the one bound by the embedded subject. If we identify variable names with movement features (as we will here), this cannot be done. Sentences 2 and 3 differ in the scope taking behavior of the quantified noun phrase *every woman*. In 2, this QNP must scope under the matrix subject, while in 3 either scope order is possible.

The remainder of this paper is structured as follows. Section §2 introduces the minimalist grammar formalism. Section 3 presents a semantics for this formalism in terms of the simply typed lambda calculus. In §4, a grammar fragment is presented which allows for quantifiers to scope out of arbitrarily many non-finite clauses, but not past a tense clause boundary, as is the received wisdom in the linguistic literature [21]. Section 5 is the conclusion.

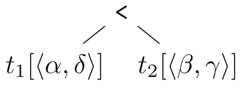
2 Minimalist Grammars

Minimalist grammars make use of two syntactic structure building operations; binary **merge** and unary **move**. **Merge** acts on its two arguments by combining them together into a single tree. The operation **move** rearranges the pieces of its single and syntactically complex argument. The generating functions **merge** and **move** are not defined on all objects in their domain. Whether a generating function is defined on a particular object in its domain (a pair of expressions in the case of **merge**, or a single expression in the case of **move**) is determined solely by the syntactic categories of these objects. In minimalist grammars, syntactic categories take the form of ‘feature bundles’, which are simply finite sequences of features. The currently accessible feature is the feature at the beginning (leftmost) position of the list, which allows for some features being available for checking only after others have been checked. In order for **merge** to apply to arguments Γ and Δ, the heads of both expressions must have matching first features in their respective feature bundles. These features are eliminated in the derived structure which results from their merger. In the case of **move**, the head of its argument Γ must have a feature matching a feature of the head of one of its subconstituents’ Δ. In the result, both features are eliminated. Each feature type has an attractor and an attractee variant (i.e. each feature is either positive or negative), and for two features to match, one must be positive and the other negative. The kinds of features relevant for the **merge** and **move** operations are standardly taken for convenience to be different. For **merge**, the attractee feature is a simple categorial feature, written `x`. There are two kinds of attractor features, `=x` and `x=`, depending on whether the selected expression is to be merged on the right (=x) or on the left (x=). For the **move** operation, there is a single attractor feature, written +y, and two attractee features, -y and ⊖y, depending on whether the movement is overt (-y) or covert (⊖y).

A lexical item is an atomic pairing of form and meaning, along with the syntactic information necessary to specify the distribution of these elements in more complex expressions. We write lexical items using the notation $\langle\sigma,\delta\rangle$, where σ is a lexeme, and δ is a feature bundle.

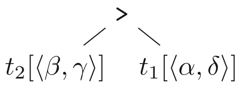
Complex expressions are written using the notation of [33] for the 'bare phrase structure' trees of [5]. These trees are essentially X-bar trees without phrase and category information represented at internal nodes. Instead, internal nodes are labeled with 'arrows' > and <, which point to the head of their phrase. A tree of the form $[_< \alpha\ \beta]$ indicates that the head is to be found in the subtree α, and we say that α projects over β, while one of the form $[_> \alpha\ \beta]$ that its head is in β, and we say that β projects over α. Leaves are labeled with lexeme/feature pairs (and so a lexical item $\langle\alpha,\delta\rangle$ is a special case of a tree with only a single node). The head of a tree t is the leaf one arrives at from the root by following the arrows at the internal nodes. If t is a bare phrase structure tree with head H, then I will write t[H] to indicate this. (This means we can write lexical items $\langle\alpha,\delta\rangle$ as $\langle\alpha,\delta\rangle[\langle\alpha,\delta\rangle]$.) The **merge** operation is defined on a pair of trees t_1, t_2 if and only if the head of t_1 has a feature bundle which begins with either =x or x=, and the head of t_2 has a feature bundle beginning with the matching x feature. The bare phrase structure tree which results from the merger of t_1 and t_2 has t_1 projecting over t_2, which is attached either to the right of t_1 (if the first feature of the head was =x) or to the left of t_1 (if the first feature of the head was x=). In either case, both selection features are checked in the result.

$$\mathbf{merge}(t_1[\langle\alpha,\texttt{=x}\delta\rangle],t_2[\langle\beta,\texttt{x}\gamma\rangle]) =$$



<
$t_1[\langle\alpha,\delta\rangle]$
$t_2[\langle\beta,\gamma\rangle]$

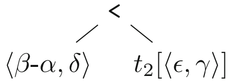
$$\mathbf{merge}(t_1[\langle\alpha,\texttt{x=}\delta\rangle],t_2[\langle\beta,\texttt{x}\gamma\rangle]) =$$



>
$t_2[\langle\beta,\gamma\rangle]$
$t_1[\langle\alpha,\delta\rangle]$

If the selecting tree is both a lexical item and an affix (which I notate by means of a hyphen preceding/following the lexeme in the case of a suffix/prefix), then head movement is triggered from the head of the selected tree to the head of the selecting tree.

$$\mathbf{merge}(\langle\text{-}\alpha,\texttt{=x}\delta\rangle,t_2[\langle\beta,\texttt{x}\gamma\rangle]) =$$



<
$\langle\beta\text{-}\alpha,\delta\rangle$
$t_2[\langle\epsilon,\gamma\rangle]$

The operation **move** applies to a single tree $t[\langle\alpha,\texttt{+y}\delta\rangle]$ only if there is *exactly one* leaf ℓ in t with matching first feature -y or ⊖y.[2] This is a radical version of

[2] Other constraints have been explored in [12].

the shortest move constraint [5], and will be called the SMC – it requires that an expression move to the first possible landing site. If there is competition for that landing site, the derivation crashes (because the losing expression will have to make a longer movement than absolutely necessary). If it applies, **move** moves the maximal projection of ℓ to a newly created specifier position in t (overtly, in the case of $\texttt{-y}$, and covertly, in the case of $\ominus\mathrm{y}$), and deletes both licensing features. To make this precise, let $t\{t_1 \mapsto t_2\}$ denote the result of replacing all subtrees t_1 in t with t_2, for any tree t, and let ℓ_t^M denote the maximal projection of ℓ in t, for any leaf ℓ.

$$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta\rangle]) = \begin{array}{c} > \\ \diagup \qquad \diagdown \\ t'[\langle\beta,\gamma\rangle] \quad t[\langle\alpha,\delta\rangle]\{t' \mapsto \langle\epsilon,\epsilon\rangle\} \end{array}$$
$$\text{(where } t' = \langle\beta, \texttt{-y}\gamma\rangle_t^M\text{)}$$

$$\mathbf{move}(t[\langle\alpha, \texttt{+y}\delta\rangle]) = \begin{array}{c} > \\ \diagup \qquad \diagdown \\ \langle\epsilon,\gamma\rangle \quad t[\langle\alpha,\delta\rangle]\{t' \mapsto t'[\langle\beta,\epsilon\rangle]\} \end{array}$$
$$\text{(where } t' = \langle\beta, \ominus\mathrm{y}\gamma\rangle_t^M\text{)}$$

2.1 The Shortest Move Constraint

Minimalist grammars with the shortest move constraint were proven [28] to be weakly equivalent to multiple context free grammars [31]. The proof that minimalist languages are contained in the MCFLs proceeds by constructing an equivalent MCFG whose derivation trees are identical to those of the minimalist grammar (modulo a projection). Each component of a derived tuple of strings in the target MCFG corresponds to either a moving subexpression in the MG or to the fixed head and non-moving material around it. The shortest move constraint ensures that there is a finite upper bound on the number of possible moving subexpressions, and thus on the dimension of the target MCFG.

The derived trees of Minimalist Grammars, while not corresponding to so natural a class due to non-logical restrictions (such as the distribution of traces), are contained [26] in the tree languages derivable by multiple regular tree grammars, which are MCFGs where the derived tuples contain trees instead of strings [10].

These observations motivate the idea that, at least in the context of the SMC, the natural data structure for the objects derived by minimalist grammars are tuples, where all positions but the first are indexed by feature types. An alternative presentation of this data structure is as a *store*, in the sense of [6], which is a pair of an object and a finite map from feature types to objects. In the case of syntax, the objects are trees. In the case of semantics, they will turn out to be simply typed lambda terms, as explained next.

2.2 Derivations

A derivation tree is an element of the term language over the ranked alphabet $A_0 \cup A_1 \cup A_2$, where $A_0 = \mathit{Lex}$ is the set of nullary symbols, $A_1 = \{\mathbf{move}\}$

is the set of unary symbols, and $A_2 = \{\textbf{merge}\}$ the set of binary symbols. As a consequence of the translation of minimalist grammars into multiple context free grammars [28,16], and as described in [26], the set of derivation trees in a minimalist grammar of an expression with unchecked feature string γ at the root and no features anywhere else is regular.

For reasons of space (and because the derivation tree is more informative than any single derived tree), we will present only derivation trees for the expressions in this paper.

3 Minimalist Semantics

Here we present a rule-by-rule semantic interpretation scheme for minimalist grammars. The denotation of a syntactic object t is a pair of a simply typed lambda term and a quantifier store. A quantifier store is a partial function from feature types to simply typed terms. The idea is that a syntactic object t with a moving subexpression $t' = \langle\beta, \text{-y}\gamma\rangle_t^M$ has a quantifier store $\mathcal{Q}$ such that $\mathcal{Q}(y)$ is the stored meaning of t'.

We use lower case greek letters $(\alpha, \beta, \ldots)$ to stand for denotations of syntactic objects (pairs of simply typed lambda terms and quantifier stores), and the individual components of these denotations will be referred to with the corresponding roman letters in lowercase for the lambda term component, and in uppercase for the store component.[3] Thus, $\alpha = \langle a, A\rangle$.

We treat lexical items as being paired with the empty store.

Quantifier Stores. With $\emptyset$ we denote the empty quantifier store, such that for all feature types $\textsc{f}$, $\emptyset(f)$ is undefined. If two quantifier stores $\mathcal{Q}_1, \mathcal{Q}_2$ have disjoint domains,[4] then $\mathcal{Q}_1 \vee \mathcal{Q}_2$ denotes the store such that:

$$\mathcal{Q}_1 \vee \mathcal{Q}_2(f) := \begin{cases} \mathcal{Q}_1(f) & \text{if defined} \\ \mathcal{Q}_2(f) & \text{otherwise} \end{cases}$$

Let $\mathcal{Q}$ be a quantifier store, and $\textsc{f}, \textsc{g}$ feature types. Then $\mathcal{Q}/f$ is the quantifier store just like $\mathcal{Q}$ except that it is undefined on $\textsc{f}$, $\mathcal{Q}[f := \alpha]$ is the store just like $\mathcal{Q}$ except that it maps $\textsc{f}$ to α, and $\mathcal{Q}_{f \leftarrow g}$ is the store just like $\mathcal{Q}/g$ except that it maps $\textsc{f}$ to whatever $\mathcal{Q}$ mapped $\textsc{g}$ to.

Variable Naming. Our approach to variable naming is from [32], and is based on the observation that, in the context of the SMC, the type of the next feature of a moving expression uniquely identifies it. We will thus need no free individual variables (those of type e) other than these, and thus we subscript them with feature types.

[3] Lower case greek letters also have been used to stand for feature sequences, lexemes, etc. This overloading is hoped to be clear from context.

[4] If this is not the case, then the syntactic expressions they correspond to cannot be syntactically merged, as this would result in a violation of the SMC.

3.1 Merge

There are two possible semantic reflexes of syntactic merger. The first case is function application from one argument to the other (in a type driven manner). In the second case one of the arguments is a moving expression, and the other denotes a function from individuals. Here we insert it into the store indexed by the next licensee feature type it will move to check. The other argument is applied to a variable of the same name as the index under which the moving argument was stored. We denote these two semantic operations **mergeApp** and **mergeStore**.[5]

$$\mathbf{mergeApp}(\alpha, \beta) = \begin{cases} \langle a(b),\ \ A \cup B \rangle \\ \qquad\quad \text{or} \\ \langle b(a),\ \ B \cup A \rangle \end{cases}$$

$$\mathbf{mergeStore}(\alpha, \beta) = \langle a(x_f),\ A \cup B[f := b] \rangle$$
$$\text{(where } \beta\text{'s next feature is } \texttt{-f}\text{)}$$

3.2 Move

There are multiple possible semantic reflexes of syntactic movement as well.[6] The first, **moveEmpty**, is used when the moving expression has previously taken scope. The second, **moveLater**, will be used when the moving expression is going to take scope in a later position. The third, **moveNow**, is used when the moving expression is going to take scope in this position. Here, the appropriate variable is abstracted over, and the resulting predicate is given as argument to the stored expression. In the below, we assume `-f` to be the feature checked by this instance of **move**.

$$\mathbf{moveEmpty}(\alpha) = \langle a, A \rangle$$
$$\text{where } A \text{ is undefined at } f$$

$$\mathbf{moveLater}(\alpha) = \langle (\lambda x_f.a)(x_g), A_{g \leftarrow f} \rangle$$
$$\text{for } \texttt{-g} \text{ the next feature to check}$$

$$\mathbf{moveNow}(\alpha) = \langle A(f)(\lambda x_f.a),\ A/f \rangle$$

[5] Note that the condition on **mergeStore** refers to the features of the moving expression. This information is present in the categories used in the MCFG translation of a minimalist grammar, and is a finite state bottom up relabeling of the standard minimalist derivation tree. Thus while not a homomorphic interpretation of minimalist derivations, it is a homomorphic interpretation of a finite state relabeling of minimalist derivations, or a transductive interpretation of minimalist derivations.

[6] [24] argues for the inclusion of function composition, in order to account for inverse linking constructions, which are beyond the scope of this paper.

3.3 Going Variable Free

As observed in [23], remnant movement wreaks havoc with the interpretation of movement dependencies presented here. The problem is that we can end up with a term in which variables remain free, with the lambda which was supposed to have bound them to their right. The reason this problem exists is that the semantics presented here treats each moving expression independently of all others, while the existence of remnant movement forces us to 'coordinate' the interpretations of two moving expressions where one contains the base position of the other. There are various strategies for resolving this difficulty [23], however, the one which suggests itself in the present simply typed context is nice because it simply eliminates free variables (named after features or not).

The basic idea is straightforward: an expression $\alpha = \langle a, A\rangle$ 'abbreviates' an expression $\alpha' = \langle \lambda x_{f_1}, \ldots, x_{f_n}.a, A\rangle$, where the domain of A is exactly the set of features $f_1, \ldots, f_n$. In other words, we take the 'main denotation' of an expression to have the variables expressions in the store may have introduced in it already and always bound. For any store A, define $\mathsf{var}(A)$ to be a fixed enumeration of the domain of A (viewed as variables), and $\Lambda(A)$. ϕ to be a prefix of lambda abstractions over the domain of A viewed as variables. Then the **mergeApp** rule can be given as follows:

$$\mathbf{mergeApp}(\alpha, \beta) = \begin{cases} \Lambda(A \cup B).\ a(\mathsf{var}(A))(b(\mathsf{var}(B))),\ \ A \cup B\rangle \\ \qquad\quad \text{or} \\ \langle \Lambda(B \cup A).b(\mathsf{var}(B))(a(\mathsf{var}(A))),\ \ B \cup A\rangle \end{cases}$$

The rules **moveEmpty** and **moveLater** are also simply recast in these terms.

$$\begin{aligned} \mathbf{moveEmpty}(\alpha) &= \langle \Lambda(A).a(\mathsf{var}(A)), A\rangle \\ \mathbf{moveLater}(\alpha) &= \langle \Lambda(A_{j\leftarrow i}).(\lambda x_i.a(\mathsf{var}(A)))(x_j), A_{j\leftarrow i}\rangle \end{aligned}$$

Not everything is so straightforward, unfortunately. What are we to do with the **mergeStore** rule, when the expression whose denotation is to be stored itself contains moving pieces? An example is verb phrase topicalization (in a sentence like *"Use the force, Luke will"*). In a typical minimalist analysis of this construction, the lexical item *will* merges with the verb phrase t_k *use the force*, which itself contains a moving expression, the subject *Luke*$_k$. Thus, under our 'variable free' view, we have a VP with main denotation of type *et* (because its store contains the denotation of *Luke*), but *will* is of type *tt*. There is a natural resolution to this (self-inflicted) problem, which implements the transformational observation that 'remnant movement obligatorilly reconstructs' [4]. We split the **mergeStore** rule into **mergeStoreMB**,[7] which simply stores expressions with empty stores, and **mergeStoreHO**,[8] which stores the denotation of a merging expression with moving pieces, and inserts a higher order variable.

[7] For '**M**onadic **B**ranching' [22].

[8] For '**H**igher **O**rder'.

$$\begin{aligned}\textbf{mergeStoreMB}(\alpha, \langle b, \emptyset\rangle) &= \langle \Lambda(X).\ a(\mathsf{var}(A))(x_i),\ X\rangle \\ &\qquad (\text{for } X = A[i := b], \text{ and } \mathtt{ty}(b) = (\mathtt{ty}(x_i) \to \gamma) \to \delta.) \\ \textbf{mergeStoreHO}(\alpha, \beta) &= \langle \Lambda(X).\ a(\mathsf{var}(A))(x_i(\mathsf{var}(B))),\ X\rangle \\ &\qquad (\text{for } X = A \cup B[i := b], \text{ and } \mathtt{ty}(x_i) = \mathtt{ty}(b).)\end{aligned}$$

The quantifier store holds both generalized quantifiers, as before, as well as relations like $\lambda x.x$ *uses the force*. This latter type of expression seems to be used as an argument of some operator (such as a focus operator). Accordingly, we adjust the **moveNow** rule to allow a type driven retrieval scheme:

$$\begin{aligned}\textbf{moveNowFunc}(\alpha) &= \langle \Lambda(X).\ A(i)(\lambda x_i.a(\mathsf{var}(A)),\ X\rangle && (\text{for } X = A/i) \\ \textbf{moveNowArg}(\alpha) &= \langle \Lambda(X).\ \lambda x_i.a(\mathsf{var}(A))(A(i)),\ X\rangle && (\text{for } X = A/i)\end{aligned}$$

This is not a particularly interesting semantic treatment of remnant movement, as it simply implements obligatory 'semantic reconstruction' [7]. To properly implement linguistic analyses, it seems that some lexical items need access to the stores of their arguments. This would allow us to assign the following interpretation to a topicalization lexical item, which checks the topic feature of a moving expression, and returns a bipartite structure of the form $\mathtt{top}(\phi)(\psi)$, which asserts that ψ is the topic of ϕ: $[\![\langle\epsilon, \mathtt{=t\ +top\ c}\rangle]\!](\alpha) = \Lambda(A).\mathtt{top}(\lambda x_{top}.a(v(A)))(x_{top})$. This and alternatives need to be worked out further.

3.4 Determining Quantifier Scope

It is natural to view the minimalist grammar operations as pairs of syntactic and semantic functions. Thus, for example, we have both ⟨`merge`, **mergeApp**⟩ and `merge`, **mergeStore**⟩. The semantic interpretation rules implement a reconstruction theory of quantifier scope [19], according to which the positions in which a quantified noun phrase can take scope are exactly those through which it has moved.

A shortcoming of this proposal as it now stands is that the yield language of the well formed derivation trees is ambiguous (in contrast to the uninterpreted minimalist grammar system [15]). Because minimalist grammar derivation tree languages are closed under intersection with regular sets [14,25], any regular strategy for determining which of the possible positions a quantified noun phrase should be interpreted in can be used to give control over scope taking back to the lexicon. A simple such strategy is to assign to a licensee feature which may be used at **moveNow** a particular diacritic. A moving expression is then required to take scope at the highest (derivational) position permitted in which it checks its features, or in its merged position, should no other possibility obtain. We will adopt this proposal here, writing a licensee feature of type F which requires scope taking with a hat ($\mathtt{-\hat{f}}$ or $\ominus\mathtt{\hat{f}}$), and one which does not without ($\mathtt{-f}$ or $\ominus\mathtt{f}$). This move restores the functionality of the relation between sequences of lexical items and well-formed derivations, at the cost of increasing the size of the lexicon (by an additive factor).

3.5 Lexical Interpretations

A model is given by a set of atomic individuals E, a set of propositional interpretations T, and an interpretation function $\mathcal{I}$ assigning to each lexical item a model theoretic object in a set built over E and T. We assume throughout that $\mathcal{I}$ assigns to lexical items denotations of the 'standard' type; common nouns denote functions from individuals to propositions, n-ary verbs functions from n individuals to propositions, determiners relations between common noun denotations, etc.

We call the type of atomic individuals e, and that of propositions as o. Following [8], let the type of a (discourse) context be γ, and a sentence (to be revised shortly) to evaluate to a proposition only in a context (i.e. to be a function from contexts to propositions). Contexts will serve here to provide the input to pronoun resolution algorithms. For simplicity, we will treat them as lists of individuals (with [8], but see [3] for a more sophisticated treatment). The operation of updating a context $c \in \gamma$ with an individual a is written $a :: c$. Pronoun resolution algorithms *select* individuals from contexts, and are generically written $\texttt{sel}$.[9] We will assume that the individual selected from the context is actually present in the context ($\texttt{sel}(\gamma) \in \gamma$), leaving aside the question of empty contexts, and more precise conditions on the identity of the selected individual.

To deal with dynamic phenomena in his system, [8] lifts the type of a sentence once again to be a function from contexts (of type γ) and discourse continuations (functions of type γo) to propositions; an expression of type $\gamma(\gamma o)o$, which we will abbreviate as t. The idea is that a sentence is interpreted as a function from both its left context c and its right context ϕ to propositions.

With the exception of verb denotations, and expressions (such as relative pronouns) which are analysed there as taking verb denotations as arguments, we are able to simply take over the denotations assigned to lexical items from [8].

The style of analysis popular in the minimalist syntactic literature (and followed here), makes less straightforward a homomorphic relation between syntactic and semantic types. For example, the subject of a sentence is typically selected by a 'functional head', i.e. a lexical item other than the verb.

Relations. We first look at the denotations of n-ary relation denoting lexical items, such as nouns and verbs. While nouns are treated here just as in [8], we treat verbs as on par with nouns, not, as does [8], as functions which take generalized quantifier denotations as arguments. This is because, in the minimalist grammar system, scope is dealt with by movement, not by modifying the verbal denotation (as is standard in categorial approaches to scope [18]).

A common noun, such as *monkey*, which is interpreted in the model as a function **monkey** of type eo mapping entities to true just in case they are monkeys, denotes a function of type et:

$$[\![monkey]\!](x)(c)(\phi) := \mathbf{monkey}(x) \wedge \phi(c)$$

[9] [9] gives pronoun resolution algorithms an additional property argument, and proposes that they select an individual with that property from the context.

Similarly, a transitive verb, such as *eat*, interpreted as a function **eat** : eeo mapping pairs of entities to true just in case the second ate the first, denotes a function of type eet:

$$[\![eat]\!](x)(y)(c)(\phi) := \mathbf{eat}(x)(y) \wedge \phi(c)$$

In general, given a function $f : \underbrace{e \cdots e}_{n \text{ times}} o$, we lift it to $\textsc{lift}(f) : \underbrace{e \cdots e}_{n \text{ times}} t$:

$$\textsc{lift}(f)(x_1) \cdots (x_n)(c)(\phi) := f(x_1) \cdots (x_n) \wedge \phi(c)$$

The conjunction and conjunct $\phi(c)$ common to all such predicates cashes out the empirical observation that propositions in a discourse combine conjunctively [30].

We adopt the following convention regarding arguments of type o (propositions): they are lifted to arguments of type t, and are passed as arguments the left and right context parameters of the lifted lexical item. As an example, take **believe** to be interpreted as a function of type oeo; a relation between a proposition (the belief) and an individual (the believer).

$$\textsc{lift}(\mathbf{believe})(S)(x)(c)(\phi) := \mathbf{believe}(S(c)(\phi))(x) \wedge \phi(c)$$

This illustrates a difficulty with the dynamic aspects of the system; it is not obvious how to allow a context to pick up referents contained in a different branch (here the propositional argument to **believe**) than which the continuation is (here in a position 'c-commanding' this argument). Should the propositional argument to **believe** contain a proper name, for example, this should be able to be found in the context of the remainder of the sentence. We do not dwell further on this difficulty here (though see footnote 11).

Noun Phrases. Traditional noun phrases (which are here, in line with [1], called 'determiner phrases', or DPs) such as *Mary*, *he*, or *every monkey*, denote functions $g : (et)t$. Proper names are interpreted as generalized quantifiers of type $(eo)o$, which are then lifted to the higher type via the operation GQ:

$$\textsc{gq}(G)(P)(c)(\phi) = G(\lambda x_e.\ P(x)(c)(\lambda d.\phi(x::d)))$$

Note that the individual 'referred to' by the generalized quantifier is incorporated into the context $(x::c)$ of the continuation of the sentence (ϕ). This permits future pronouns to pick up this individual as a possible referent. As an example, $\mathbf{Mary} = \lambda P_{eo}.P(m)$. And so $\textsc{gq}(\mathbf{Mary}) = \lambda c, \phi.P(m)(c)(\lambda d.\phi(m::d))$.

As mentioned above, we interpret pronouns, not as variables, but as noun phrase denotations involving pronoun resolution algorithms: $\texttt{sel} : \gamma e$.

$$[\![pro]\!](P)(c)(\phi) := P(\texttt{sel}(c))(c)(\phi(c))$$

Determiners. The system presented in [8] is limited to quantifiers of type $\langle 1 \rangle$.[10],[11] Accordingly, we restrict ourselves to the standard universal and existential quantifiers $\forall, \exists : (eo)o$. Determiners *every* and *some* are of type $(et)(et)t$, and denote the following functions.

$$[\![every]\!](P)(Q)(c)(\phi) := \forall(\lambda x.\neg P(x)(c)(\lambda d.\neg Q(x)(x :: d)(\lambda d.\top)) \wedge \phi(c))$$
$$[\![some]\!](P)(Q)(c)(\phi) := \exists(\lambda x.P(x)(c)(\lambda d.Q(x)(x :: d)(\phi)))$$

As explained by De Groote, the negations inside of the lambda term representing the denotation of *every* make the conjunctive meanings of the properties P and Q equivalent to the desired implication. As an example, take $P = [\![man]\!] = \lambda x, c, \phi.\mathbf{man}(x) \wedge \phi(c)$ and take $Q = [\![smile]\!] = \lambda x, c, \phi.\mathbf{smile}(x) \wedge \phi(c)$. Then $[\![every]\!]([\![man]\!])([\![smile]\!])(c)(\phi)$ β-reduces to the following lambda term, taking $A \rightarrow B$ as an abbreviation for $\neg(A \wedge \neg B)$.

$$\forall(\lambda x.\mathbf{man}(x) \rightarrow \mathbf{smile}(x) \wedge \top) \wedge \phi(c)$$

Finally, $\top$ stands for the always true proposition. As it always occurs as part of a conjunction, we simply and systematically replace $A \wedge \top$ everywhere with the equivalent A.

Everything Else. All other lexical items are interpreted as the identity function of the appropriate type. While some (such as auxiliaries) should be assigned a more sophistiated denotation in a more sophisticated fragment, others (such as most of the 'functional' lexical items) play no obvious semantic role, and are there purely to express syntactic generalizations.

4 A Fragment

There are two main constructions addressed in this section. First (in §4.1), we show the necessity of de Groote's discourse contexts (or something like them) for the variable naming scheme adopted here. Then (in §4.2), as advertised in the abstract, we illustrate the 'tensed-clause boundedness' of quantifier raising.

We draw lexical items mostly unchanged from [23] (see figure 1). The fragment derives the following sentences, with the indicated scope relations.

4. Some man believed that every woman smiled. $(\exists > \forall)$
5. Some man believed every woman to have smiled. $(\exists > \forall,\ \forall > \exists)$

Although the number of lexical items (especially in the verbal domain) may look at first blush imposing, there are two things to bear in mind. First, most of them are (intended to be) 'closed class' items, meaning that they needn't ever

[10] A quantifier of type $\langle n_1, \ldots, n_k \rangle$ takes k predicates of arities $n_1, \ldots, n_k$ respectively, and returns a truth value.

[11] The extension to quantifiers of type $\langle n \rangle$ for any n is straightforward, but how it should be extended to handle binary quantifiers (of type $\langle 1, 1 \rangle$) is non-obvious.

NAME	FEATURES	PRONUNCIATION	MEANING
that	=s t	“that”	id
-ed	=p +k +q s	“-ed”	id
to	=p t	“to”	id
have	=en p	“have”	id
-en	=v en	“-en”	id
Asp	=v p	“ϵ”	id
Qv	=v +q v	“ϵ”	id
v	=V =d v	“ϵ”	id
AgrO	=V +k V	“ϵ”	id
smile	=d v	“smile”	LIFT(**smile**)
praise	=d V	“praise”	LIFT(**praise**)
believe	=t V	“believe”	LIFT(**believe**)

(a) verbal elements

NAME	FEATURES	PRONUNCIATION	MEANING
dD	=D d -k ⊖q	“ϵ”	id
dQ	=D d -k ⊖q̂	“ϵ”	id
every	=n D	“every”	⟦*every*⟧
some	=n D	“some”	⟦*some*⟧
man	n	“man”	LIFT(**man**)
woman	n	“woman”	LIFT(**woman**)
pro	D	“he”	⟦*pro*⟧
j	D	“John”	GQ(**John**)
b	D	“Bill”	GQ(**Bill**)

(b) nominal elements

Fig. 1. Lexical items

$$\begin{aligned}
R(a)(b) &= \mathbf{merge}(a,b) & \mathtt{iv}(s)(v) &= R(v)(s)\\
V(a) &= \mathbf{move}(a) & \mathtt{to}(v) &= R(\mathsf{to})(R(\mathsf{have})(R(\mathsf{\text{-}en})(v)))\\
ObjK(v) &= V(R(\mathsf{AgrO})(v)) & \mathtt{pst}(v) &= V(V(R(\mathsf{\text{-}ed})(R(\mathsf{Asp})(v))))\\
ObjQ(v) &= V(R(\mathsf{Qv})(v)) & \mathtt{c} &= R(\mathsf{that})\\
Sub(s)(v) &= R(R(\mathsf{v})(v))(s) & \mathtt{d} &= R(\mathsf{dD})\\
\mathtt{tv}(s)(v)(o) &= ObjQ(Sub(s)(ObjK(R(v)(o)))) & \mathtt{q} &= R(\mathsf{dQ})
\end{aligned}$$

Fig. 2. Abbreviations

be added to as more novel words are encountered. Second, these closed class items in fact participate in a very regular way in derivations. We introduce some abbreviations (figure 2), so as to be able to concisely describe derivations of sentences.[12]

The sentence in 6 has the two (semantically equivalent) derivations represented in 7, and interpretation as in 8.

6. Some man smiled.
7. $\mathtt{pst}(\mathtt{iv}(f(\mathsf{some})(\mathsf{man}))(\mathsf{smile}))$, for $f \in \{D, Q\}$
8. $\lambda c, \phi.\ \exists(\lambda x.\ \mathbf{man}(x) \wedge \mathbf{smile}(x) \wedge \phi(c))$

The transitive sentence in 9 has the derivation in 10, which corresponds to the object wide scope reading in 11, and the derivation in 12 which corresponds to the subject wide scope reading in 13.

9. Some woman praised every man.
10. $\mathtt{pst}(\mathtt{tv}(\mathtt{d}(\mathbf{some})(\mathbf{woman}))(\mathbf{praise})(\mathtt{q}(\mathbf{every})(\mathbf{man})))$
11. $\lambda c, \phi.\forall(\lambda x.\mathbf{man}(x) \rightarrow \exists(\lambda y.\mathbf{woman}(y) \wedge \mathbf{praise}(x)(y))) \wedge \phi(c)$
12. $\mathtt{pst}(\mathtt{tv}(\mathtt{q}(\mathbf{some})(\mathbf{woman}))(\mathbf{praise})(\mathtt{q}(\mathbf{every})(\mathbf{man})))$
13. $\lambda c, \phi.\exists(\lambda y.\mathbf{woman}(y) \wedge \forall(\lambda x.\mathbf{man}(x) \rightarrow \mathbf{praise}(x)(y)) \wedge \phi(y :: c))$

[12] The ‘abbreviations’ in figure 2 are λ-terms over derivation trees. The naming of bound variables is purely mnemonic, as they are all of atomic type.

4.1 Pronouns Are Not Variables

The idea that pronouns are to be rendered as variables in some logical language is widespread in the semantic literature (though see [11,20] for some alternatives). However in the present system, where possible binders (DPs) bind variables according to the reason for their movement (of which there are only finitely many), sentences like the below prove an insurmountable challenge to this naïve idea.

14. Every boy believed that every man believed that he smiled.

In sentence 14, the pronoun can be bound either by *every boy* or by *every man*. However, if it 'denotes' a variable, it must be one of x_k or x_q, and both of these are bound by the closer DP, *every man*, leaving no possibility for the reading in which every boy smiles.[13] Here we see that treating pronouns as (involving) pronoun resolution algorithms provides a simple way to approach a resolution to this problem.[14]

Sentence 14 has its equivalent derivations in 15, with meaning representation in 16.

15. $\mathtt{pst}(\mathtt{iv}\ (f(\mathsf{every})(\mathsf{boy}))$
$(R(\mathsf{believe})(\mathsf{c}(\mathtt{pst}(\mathtt{iv}\ (g(\mathsf{every})(\mathsf{man}))$
$(R(\mathsf{believe})(\mathtt{pst}(\mathtt{iv}(h(\mathsf{pro})(\mathsf{smile}))))))))))$
for $f, g, h \in \{\mathtt{d}, \mathtt{q}\}$
16. $\lambda c, \phi.\forall(\lambda x.\ \mathbf{boy}(x) \rightarrow$
$\mathbf{believe}(\forall(\lambda y.\ \mathbf{man}(y) \rightarrow$
$\mathbf{believe}(\mathbf{smile}(\mathtt{sel}(y :: x :: c)))(y))(x)) \wedge \phi(c)$

Crucially, in every context, the input to the pronoun resolution algorithm includes the (bound) variables x and y, the choice of which would result in the bound reading of the pronoun for *every boy* and *every man* respectively.

4.2 The Tensed-Clause Boundedness of QR

Now we present derivations for sentences 4 and 5 (repeated below as 24 and 17). We begin with 17, which has two surface scope readings which correspond to *de re* (19) and *de dicto* (21) beliefs, and an inverse scope reading (in 23).

17. Some man believed every woman to have smiled.
18. $\mathtt{pst}(\mathtt{tv}(f(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathtt{d}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile})))),$
where $f \in \{\mathtt{d}, \mathtt{q}\}$
19. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \mathbf{believe}(\forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{smile}(y)))(x) \wedge \phi(x :: c))$
20. $\mathtt{pst}(\mathtt{tv}(\mathtt{q}(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathtt{q}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))$

[13] In response to this problem, [23] abandons the idea of [32] that variables are named after movement dependencies, and with it the simply typed lambda calculus.

[14] It is not in itself an answer to this problem, as it introduces an unanalyzed 'unknown' in the form of a pronoun resolution algorithm.

21. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{believe}(\mathbf{smile}(y))(x)) \wedge \phi(x :: c))$
22. $\mathtt{pst}(\mathtt{tv}(\mathsf{d}(\mathsf{some})(\mathsf{man}))(\mathsf{believe})(\mathtt{to}(\mathtt{iv}(\mathsf{q}(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))$
23. $\lambda c, \phi.\forall(\lambda y.\mathbf{woman} \rightarrow \exists(\lambda x.\mathbf{man}(x) \wedge \mathbf{believe}(\mathbf{smile}(y))(x))) \wedge \phi(c)$

Now we turn to 24. Here the inverse scope reading is not available for the simple reason that a tensed clause (one with the lexical item -ed) forces a Q feature to be checked. Thus the embedded DP *every woman* does not enter into a movement relationship with anything after the matrix subject is present.

24. Some man believed that every woman smiled.
25. $\mathtt{pst}(\mathtt{iv}(f(\mathsf{some})(\mathsf{man}))(R(\mathsf{believe})(\mathsf{c}(\mathtt{pst}(\mathtt{iv}(g(\mathsf{every})(\mathsf{woman}))(\mathsf{smile}))))))$,
    for all $f, g \in \{\mathsf{q}, \mathsf{d}\}$
26. $\lambda c, \phi.\exists(\lambda x.\mathbf{man} \wedge \mathbf{believe}(\forall(\lambda y.\mathbf{woman}(y) \rightarrow \mathbf{smile}(y)))(x) \wedge \phi(x :: c))$

The tensed-clause boundedness of quantifier scope is a fragile and analysis dependent property; a simple 'splitting' of the -ed lexical item into one of the form $\langle -\mathsf{ed}, \mathtt{=p\ +k\ s}\rangle$ and another of the form $\langle \epsilon, \mathtt{=s\ +q\ s}\rangle$ suddenly makes the inverse scope reading in sentence 24 derivable. There are two things to say at this point. First, the 'actual' generalization about scope taking inherent in (this version of) the minimalist grammar framework is that an expression may take scope over only those others dominated by a node in the derivation tree with which it enters into a feature checking relationship. This happens to coincide in our fragment with tensed clauses. Second, the intuitive generalizations this fragment is making are (1) that scope can be checked at the VP and at the S level, and (2) that scope must be checked as soon as possible. If this latter condition is formulated as a transderivational constraint [13], and applied to derivations in the alternative fragment (in which the -ed lexeme is 'split' as per the above remarks), the present fragment can be viewed as the result of 'compiling out' the effect of the constraint on the alternative fragment.

5 Conclusion

We have shown how De Groote's simply typed account of dynamic phenomena can be used in a minimalist grammar. This allows us to maintain Stabler's ideas about variables in movement dependencies, as well as to use nothing but the simply typed lambda calculus to deliver the same meanings as the more standard 'LF' interpretative accounts of semantics in the chomskyian tradition (modulo pronouns). What is doing the work here is the rejection of the pronouns-as-variables view, in favor of a pronouns-as-functions-from-contexts view. This latter seems to be a novel perspective on the the pronouns-as-definite-description view [11] (especially in the light of [9]).

In order to preserve the functional relation between derivations and meanings, we have incorporated information into the feature system (whether or not a licensee feature has a hat diacritic). However, this strategy seems non-ideal, as it results in cases of spurious ambiguity (the worst offender in this paper was example 1, with six equivalent derivations).

Finally, we have noted that there are open questions regarding the best way of incorporating sentential complement embedding verbs and type $\langle 1, 1\rangle$ quantifiers into the dynamism-via-continuation framework used here [8].

References

1. Abney, S.P.: The English Noun Phrase in its Sentential Aspect. Ph.D. thesis, Massachusetts Institute of Technology (1987)
2. Amblard, M.: Calculs de représentations sémantiques et syntaxe générative: les grammaires minimalistes catégorielles. Ph.D. thesis, Université Bordeaux I (2007)
3. Asher, N., Pogodalla, S.: A montagovian treatment of modal subordination. In: Li, N., Lutz, D. (eds.) Semantics and Linguistic Theory (SALT), vol. 20, pp. 387–405. eLanguage (2011)
4. Barss, A.: Chains and Anaphoric Dependence: On Reconstruction and its Implications. Ph.D. thesis, Massachusetts Institute of Technology (1986)
5. Chomsky, N.: The Minimalist Program. MIT Press, Cambridge (1995)
6. Cooper, R.: Quantification and Syntactic Theory. D. Reidel, Dordrecht (1983)
7. Cresti, D.: Extraction and reconstruction. Natural Language Semantics 3, 79–122 (1995)
8. de Groote, P.: Towards a montagovian account of dynamics. In: Gibson, M., Howell, J. (eds.) Proceedings of SALT 16, pp. 1–16 (2006)
9. De Groote, P., Lebedeva, E.: Presupposition accommodation as exception handling. In: Fernandez, R., Katagiri, Y., Komatani, K., Lemon, O., Nakano, M. (eds.) The 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue - SIGDIAL 2010, pp. 71–74. Association for Computational Linguistics, Tokyo (2010)
10. Engelfriet, J.: Context-free graph grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages. Beyond Words, vol. 3, ch.3, pp. 125–213. Springer (1997)
11. Evans, G.: Pronouns. Linguistic Inquiry 11(2), 337–362 (1980)
12. Gärtner, H.M., Michaelis, J.: Some remarks on locality conditions and minimalist grammars. In: Sauerland, U., Gärtner, H.M. (eds.) Interfaces + Recursion = Language?, Studies in Generative Grammar, vol. 89, pp. 161–195. Mouton de Gruyter, Berlin (2007)
13. Graf, T.: A tree transducer model of reference-set computation. UCLA Working Papers in Linguistics 15, Article 4 (2010)
14. Graf, T.: Closure Properties of Minimalist Derivation Tree Languages. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 96–111. Springer, Heidelberg (2011)
15. Hale, J.T., Stabler, E.P.: Strict Deterministic Aspects of Minimalist Grammars. In: Blache, P., Stabler, E.P., Busquets, J., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 162–176. Springer, Heidelberg (2005)
16. Harkema, H.: Parsing Minimalist Languages. Ph.D. thesis, University of California, Los Angeles (2001)
17. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell Publishers (1998)
18. Hendriks, H.: Studied Flexibility: Categories and types in syntax and semantics. Ph.D. thesis, Universitaet van Amsterdam (1993)
19. Hornstein, N.: Movement and chains. Syntax 1(2), 99–127 (1998)

20. Jacobson, P.: Towards a variable-free semantics. Linguistics and Philosophy 22(2), 117–184 (1999)
21. Johnson, K.: How far will quantifiers go? In: Martin, R., Michaels, D., Uriagereka, J. (eds.) Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik, ch.5, pp. 187–210. MIT Press, Cambridge (2000)
22. Kanazawa, M., Michaelis, J., Salvati, S., Yoshinaka, R.: Well-Nestedness Properly Subsumes Strict Derivational Minimalism. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 112–128. Springer, Heidelberg (2011)
23. Kobele, G.M.: Generating Copies: An investigation into structural identity in language and grammar. Ph.D. thesis, University of California, Los Angeles (2006)
24. Kobele, G.M.: Inverse linking via function composition. Natural Language Semantics 18(2), 183–196 (2010)
25. Kobele, G.M.: Minimalist Tree Languages Are Closed Under Intersection with Recognizable Tree Languages. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 129–144. Springer, Heidelberg (2011)
26. 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)
27. LeComte, A.: Semantics in minimalist-categorial grammars. In: de Groote, P. (ed.) FG 2008, pp. 41–59. CSLI Press (2008)
28. Michaelis, J.: On Formal Properties of Minimalist Grammars. Ph.D. thesis, Universität Potsdam (2001)
29. Parigot, M.: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 190–201. Springer, Heidelberg (1992)
30. Pietrowski, P.M.: Events and Semantic Architecture. Oxford University Press (2005)
31. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science 88, 191–229 (1991)
32. Stabler, E.P.: Computing quantifier scope. In: Szabolcsi, A. (ed.) Ways of Scope Taking, ch. 5, pp. 155–182. Kluwer, Boston (1997)
33. Stabler, E.P.: Derivational Minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)

CoTAGs and ACGs

Gregory M. Kobele[1] and Jens Michaelis[2]

[1] University of Chicago, Chicago, Illinois, USA
[2] Bielefeld University, Bielefeld, Germany

Abstract. Our main concern is to provide a complete picture of how *coTAGs*, as a particular variant within the general framework of *tree adjoining grammars (TAGs)*, can be captured under the notion of *abstract categorial grammars (ACGs)*. *coTAGs* have been introduced by Barker [1] as an "alternative conceptualization" in order to cope with the tension between the TAG-mantra of the "locality of syntactic dependencies" and the seeming non-locality of quantifier scope. We show how our formalization of Barker's proposal leads to a class of higher order ACGs. By taking this particular perspective, Barker's proposal turns out as a straightforward extension of the proposal of Pogodalla [11], where the former in addition to "simple" inverse scope phenomena also captures *inverse linking* and *non-inverse linking* phenomena.

1 Introduction

In [1], Barker sketches an alternative approach to semantics in the *tree adjoining grammar (TAG)*-framework,[1] which he refers to as *coTAG*-approach, after the extension of the modified *substitution* operation, *cosubstitution*. The presentation in [1] is simple and intuitive, tying the scope of a quantificational noun phrase together with its position in the derivation. The approach is presented via examples, however, and is not formalized. As far as the syntactic component is concerned, it does, interestingly, preserve the weak and strong generative capacity of "classical" TAGs, though, clearly, not the derivation sets, as we will make precise here.

Our contribution in this paper is to formalize coTAGs. We use the formalism of *abstract categorial grammars (ACGs)* [3], which has already been used to formalize TAGs in, e.g., [4,10,11]. We present a mapping from coTAG-lexica to ACGs, and demonstrate that it coincides with the informal examples from Barker's paper. The mapping has as a special case the one of de Groote and Pogodalla for TAGs. Of particular interest is the fact that our mapping reveals Barker's treatment of scope taking in coTAGs to be largely identical to the TAG-inspired ACG-analysis of Pogodalla [11].

The paper is structured as follows. We begin with an (informal) introduction to coTAGs in Sec. 2. Then, in Sec. 3, we present abstract categorial grammars, which we use to formalize coTAGs in Sec. 4. Section 5 is the conclusion.

[1] We assume the reader to be familiar with the basics of the TAG-framework. See e.g. [6] or [7] for an introduction to TAGs.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 119–134, 2012.

2 CoTAGs

For A a set of *atomic types*, $\mathcal{T}(A)$ is the set of *types over* A, the smallest superset of A closed under pair formation, i.e., $A \subseteq \mathcal{T}(A)$, and if $\alpha, \beta \in \mathcal{T}(A)$ then $(\alpha\,\beta) \in \mathcal{T}(A)$. We sometimes omit outer parentheses when writing types. We sometimes write $\alpha_1\,\alpha_2\,\alpha_3$ instead of $\alpha_1(\alpha_2\,\alpha_3)$, and $\alpha_1 \cdots \alpha_{k+1}$ instead of $\alpha_1(\alpha_2 \cdots \alpha_{k+1})$ for $k \geq 2$ and types $\alpha_1, \ldots, \alpha_{k+1} \in \mathcal{T}(A)$, where $\alpha_1(\alpha_2 \cdots \alpha_3) = \alpha_1(\alpha_2\,\alpha_3)$. We write $\alpha^{k+1}\,\beta$ instead of $\alpha\,\alpha^k\beta$ for $k \geq 1$ and types $\alpha, \beta \in \mathcal{T}(A)$, where $\alpha^1 = \alpha$.

For Cat a set of *categories*, i.e., a set of *non-terminals* in the sense of a classical TAG, let $\tau_{\mathit{Cat}} : \mathit{Cat} \to \mathcal{T}(A_{\mathit{Cat}})$ be a type assigning function with A_{Cat} a set of atomic types. τ_{Cat} uniquely determines the function $\widehat{\tau}_{\mathit{Cat}}$ given next, which assigns a type from $\mathcal{T}(A_{\mathit{Cat}})$ to each $\delta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*\{\downarrow\}^?$, where $\uparrow$ and $\downarrow$ are two distinct new symbols not appearing in Cat.[2] For each $\delta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*\{\downarrow\}^?$, we use its boldface variant, $\boldsymbol{\delta}$, to denote $\widehat{\tau}_{\mathit{Cat}}(\delta)$: if $\delta \in \mathit{Cat}$, we have $\boldsymbol{\delta} = \tau_{\mathit{Cat}}(\delta)$. If $\delta = \zeta{\downarrow}$ for some $\zeta \in \mathit{Cat}(\{\uparrow\}\mathit{Cat}\,)^*$, we have $\boldsymbol{\delta} = \boldsymbol{\zeta}$. If $\delta = \zeta{\uparrow}\gamma$ for some $\zeta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*$ and $\gamma \in \mathit{Cat}$, we have $\boldsymbol{\delta} = ((\boldsymbol{\zeta}\,\boldsymbol{\gamma})\,\boldsymbol{\gamma})$, and we often write $\boldsymbol{\zeta{\uparrow}\gamma}$ instead of $((\boldsymbol{\zeta}\,\boldsymbol{\gamma})\,\boldsymbol{\gamma})$ in this case.

A *coTAG* can be defined as an octuple $\langle V_T, \mathit{Cat}\,, A_{\mathit{Cat}}\,, \widehat{\tau}_{\mathit{Cat}}\,, \mathcal{I}\times\mathcal{I}', \mathcal{A}\times\mathcal{A}', \frown, \mathrm{S}\rangle$, where V_T, Cat and A_{Cat} are a set of *terminals*, a set of *categories* and a set of *atomic types*, respectively. $\widehat{\tau}_{\mathit{Cat}}$ is the extension to the domain $\mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*\{\downarrow\}^?$ of a type assigning function $\tau_{\mathit{Cat}} : \mathit{Cat} \to \mathcal{T}(A_{\mathit{Cat}})$ in the above sense. S is a distinguished category from Cat, the *start symbol*. The set $(\mathcal{I} \times \mathcal{I}') \cup (\mathcal{A} \times \mathcal{A}')$ is a finite set of pairs of finite labeled trees which supplies the *lexical entries* of G.

The first component of a pair $\langle \alpha_{\mathrm{syn}}, \alpha_{\mathrm{sem}} \rangle \in (\mathcal{I} \times \mathcal{I}') \cup (\mathcal{A} \times \mathcal{A}')$ provides the syntactic, the second component the semantic representation of the corresponding lexical entry. Nodes in α_{syn} are linked to nodes in α_{sem} by the relation $\frown$. An operation applying to some node in α_{syn} must be accompanied by a parallel operation applying to the linked node(s) in α_{sem}. Regarding the projection to the first component of this general connection between syntactic and semantic representations, coTAGs are nearly identical to classical TAGs: labels of inner nodes are always from Cat. The labels of the root and the foot node of a syntactic coTAG-auxiliary tree are, as in the regular TAG-case, from Cat and $\mathit{Cat}\,\{^\star\}$, respectively, and up to the foot node indicating *-suffix both labels coincide. But whereas the root nodes of TAG-initial trees and the substitution nodes of arbitrary TAG-trees are labeled with elements from Cat and $\mathit{Cat}\,\{\downarrow\}$, respectively, the root nodes of syntactic coTAG-initial trees and the substitution nodes of arbitrary syntactic coTAG-trees have labels from the sets $\mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*$ and $\mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*\{\downarrow\}$, respectively.[3] That is to say, except for the differences with respect to the possible labeling of roots and substitution nodes, the 5-tuple $\langle V_T, \mathit{Cat}\,, \mathcal{I}, \mathcal{A}, \mathrm{S}\rangle$ constitutes a classical TAG, in particular providing a set of *initial elementary* trees, $\mathcal{I}$, and a set of *auxiliary elementary* trees, $\mathcal{A}$.

[2] For any set A, A^* is the set of all finite strings over A, including ϵ, the empty string. We identify A with the set of strings of length 1 over A, and take $A^?$ to denote an optional occurrence of an element from A (in a string), i.e., the set of strings $A \cup \{\epsilon\}$.

[3] For $\delta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*$ and $\gamma \in \mathit{Cat}$, we write $\delta{\uparrow}$ instead of $\delta{\uparrow}\gamma$, and $\boldsymbol{\delta{\uparrow}}$ instead of $\boldsymbol{\delta{\uparrow}\gamma}$ in case $\gamma = \mathrm{S}$.

In a somewhat relaxed sense, the projection to the second component provides a TAG as well. Given a lexical entry $\langle \alpha_{\text{syn}}, \alpha_{\text{sem}} \rangle \in (\mathcal{I} \times \mathcal{I}') \cup (\mathcal{A} \times \mathcal{A}')$, the relation $\frown$ links the root node of α_{syn} to the root node of α_{sem}, terminal nodes to terminal nodes, substitution nodes to substitution nodes and, in case α_{syn} is an auxiliary tree, the foot node to the foot node. For these nodes it holds that, if $\delta \in \mathit{Cat}\,(\{\downarrow\}\mathit{Cat}\,)^{*}\{\uparrow\}^{?}$ is the syntactic label then $\boldsymbol{\delta} = \widehat{\tau}_{\mathit{Cat}}\,(\delta)$ is the semantic label of the linked node, if $w \in V_T$ is the syntactic label then $w \in V_T$ is the semantic label too. More concretely, up to a certain extent, we may think of the 5-tuple $\langle V_T', \mathcal{T}(A_{\mathit{Cat}}\,), \mathcal{I}', \mathcal{A}', \mathbf{S} \rangle$ as a TAG, where $V_T' = V_T \cup \mathcal{X} \cup \mathit{Con} \cup \{\lambda\}$ with $\mathcal{X}$ being a denumerable set of variables, and *Con* being a set which consists at least of the "usual" logical connectives and quantifiers of FOL.

For each $\alpha_{\text{sem}} \in \mathcal{I}' \cup \mathcal{A}'$ the node-labeling is constrained via $\widehat{\tau}_{\mathit{Cat}}$ by the relation linking nodes of α_{sem} to those of the syntactic tree paired to α_{sem} in the lexicon. The "plain" yield of a α_{sem} is a string over $V_T' \cup \mathcal{T}(A_{\mathit{Cat}}\,)$. By definition we additionally demand the hierarchical tree structure to uniquely determine a closed well-typed lambda-term associated with the yield: if ν is an interior node in α_{sem} then the label of ν, $\mathit{label}(\nu)$ is a type from $\mathcal{T}(A_{\mathit{Cat}}\,)$, and ν has either one, two or three children. We distinguish these cases as i.1), i.2) and i.3), respectively.

i.1) Let μ be the single child of ν. μ's label, $\mathit{label}(\mu)$, is always in V_T'. $\mathit{label}(\nu)$ is virtually determining the type of $\mathit{label}(\mu)$, and for each node ν' of some $\alpha' \in \mathcal{I}' \cup \mathcal{A}'$ with a child of ν' labeled $\mathit{label}(\mu)$, $\mathit{label}(\nu) = \mathit{label}(\nu')$ holds.

i.2) Both children have a label from $\mathcal{T}(A_{\mathit{Cat}}\,)$. Say, the first child, μ_1, has label $\mathit{label}(\mu_1)$, and the second, μ_2, label $\mathit{label}(\mu_2)$. The labels are compatible with functional application in the sense that $\mathit{label}(\mu_1) = \mathit{label}(\mu_2)\mathit{label}(\nu)$ holds.

i.3) The first child, μ_0, is labeled λ. The second child, μ_1, and the third child, μ_2, are labeled by $\mathit{label}(\mu_1)$ and $\mathit{label}(\mu_2)$, respectively. As in i.2), $\mathit{label}(\mu_1)$ and $\mathit{label}(\mu_2)$ are from $\mathcal{T}(A_{\mathit{Cat}}\,)$, but now they virtually cope for lambda abstraction, i.e., $\mathit{label}(\nu) = (\mathit{label}(\mu_1)\mathit{label}(\mu_2))$ holds. In addition, μ_1, is necessarily dominating a single node labeled by a variable $x \in \mathcal{X}$. The logical scope of the thus virtually instantiated lambda abstraction over x is given by the subtree of ν.

If ν is a terminal node in $\alpha_{\text{sem}} \in \mathcal{I}' \cup \mathcal{A}'$ and ν's label, $\mathit{label}(\nu)$, is from V_T', ν is always the leftmost child of its parent node. In case $\mathit{label}(\nu) = \lambda$, ν has exactly two sister nodes. In case $\mathit{label}(\nu) \in V_T' - \{\lambda\}$, ν is the unique child of its parent. If ν is a terminal node in α_{sem} and its label is not from V_T', then its label is from $\mathcal{T}(A_{\mathit{Cat}}\,)$.

In order to arrive at a closed well-typed lambda-term associated with the yield of some $\alpha_{\text{sem}} \in \mathcal{I}' \cup \mathcal{A}'$, we have to replace each leaf-label from $\mathcal{T}(A_{\mathit{Cat}}\,)$ (i.e., each label of some substitution node and, if existing, the foot node) by a fresh variable of the corresponding type, and to lambda-abstract over this variable again.

As a "consequence" of the parallels between a TAG- and a coTAG-lexicon, the metaphor of constructing expressions in a coTAG is very similar to how it is in regular TAGs with one important difference. More concretely, we have the operations of *substitution* and *adjunction* in the "classical" sense, cf. Fig. 1 and

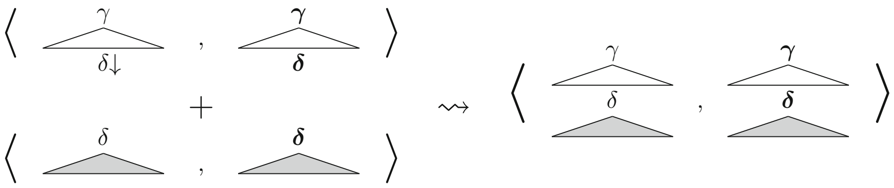


Fig. 1. *Substitution* schematically: syntactically, tree with root-label δ is substituted at leaf labeled $\delta{\downarrow}$ in tree with root-label γ; while semantically, the corresponding tree with root-label $\boldsymbol{\delta}$ is substituted at leaf labeled $\boldsymbol{\delta}$ in the corresponding tree with root-label $\boldsymbol{\gamma}$.

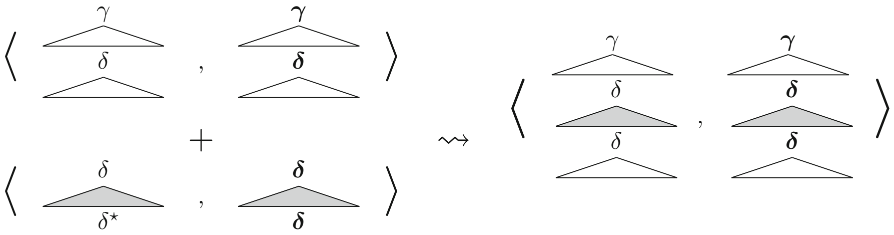


Fig. 2. *Adjoining* schematically: syntactically, tree with root-label δ and foot node-label $\delta^{\star}$ is adjoined at interior node labeled δ in tree with root-label γ; while semantically, the corresponding tree with root-label $\boldsymbol{\delta}$ and foot node-label $\boldsymbol{\delta}$ is adjoined at interior node labeled $\boldsymbol{\delta}$ in the corresponding tree with root-label $\boldsymbol{\gamma}$.

2. But in addition a derived structure with syntactic root-label $\delta{\uparrow}\gamma$ for some $\delta \in \mathtt{Cat}(\{\uparrow\}\mathtt{Cat})^{*}$ and $\gamma \in \mathtt{Cat}$, can be *cosubstituted* into a derived structure with syntactic root-label γ at a leaf labeled $\delta{\downarrow}$ *at any point in the derivation.* Within the resulting semantic representation, new terminal leaves are introduced labeled by λ and a variable x. The operation is set up such that x is chosen to be "fresh," cf. Fig. 3.

The semantic result of applying two cosubstitution steps depends on the order in which those steps are applied, no matter whether the two steps do not derivationally depend on each other. Accordingly, we require a novel notion of derivation to represent this "timing" information. An ACG-based representation

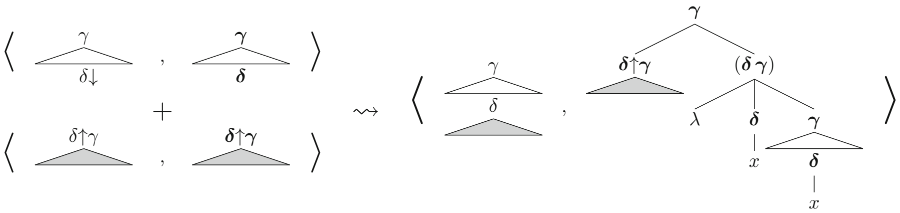


Fig. 3. *Cosubstitution* schematically: syntactically, tree with root-label $\delta{\uparrow}\gamma$ is cosubstituted at leaf labeled $\delta{\downarrow}$ in tree with root-label $\gamma \in \mathtt{Cat}$; while semantically, the corresponding tree with root-label $\boldsymbol{\delta{\uparrow}\gamma}$ is "quantified in" at the root of the corresponding tree with root-label $\boldsymbol{\gamma}$.

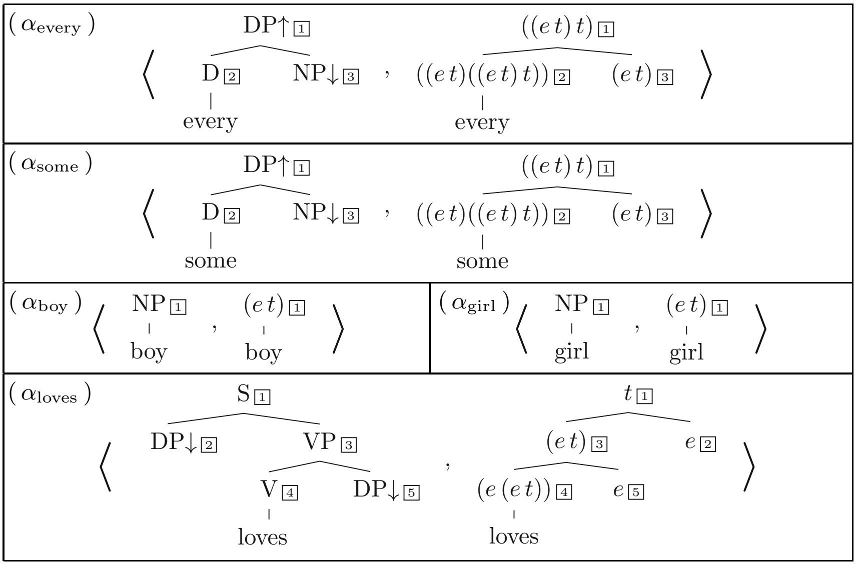


Fig. 4. The example coTAG G_{scope} (part 1)

of this notion we will be concerned with in Sec. 4. We defer a somewhat more detailed semantic analysis to that section, but emphasize that—in the same way as outlined above for elementary semantic trees—for any given derived semantic tree, closed well-typed lambda-terms may be read off "left to right" from the yield taking into account the hierarchical tree structure.

As a concrete example, consider the grammar G_{scope} presented in Fig. 4 and 5, where S is supposed to be the start symbol, and e and t the atomic types.[4] Part 1 of G_{scope}, as given in Fig. 4, allows deriving the sentence *every boy loves some girl*. One can start deriving this sentence either by substituting *boy* into *every*, and then cosubstituting *every boy* into the subject position of *loves* as shown in Fig. 6, or by substituting *girl* into *some*, and then cosubstituting *some girl* into the object position of *loves* as shown in Fig. 7. Both complete derived pairs of structures for the sentence are given in Fig. 8. As desired, the derived syntactic surface trees are identical.

Part 2 of G_{scope}, as given in Fig. 5, would in principle allow us to derive an NP with an embedded PP like, e.g., *senator from every city* in two different ways providing us with the two different scope readings in a sentence like *a senator from every city sleeps*: the *inverse linking*-reading and the *linear scope*-reading. Note that, though not strictly linear, the lambda-terms (implicitly) associated

[4] Links will usually be marked with diacritics of the form $\boxed{n}$ for some $n \geq 1$. We may occasionally avoid explicitly mentioning the links between nodes of the syntactic and the semantic representation, when we think the canonical linking is obvious. Also recall our convention to write $\delta\!\uparrow$ instead of $\delta\!\uparrow\!\mathrm{S}$, and $\boldsymbol{\delta\!\uparrow}$ instead of $\boldsymbol{\delta\!\uparrow\!\mathrm{S}}$ for $\delta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^*$ and the start symbol S.

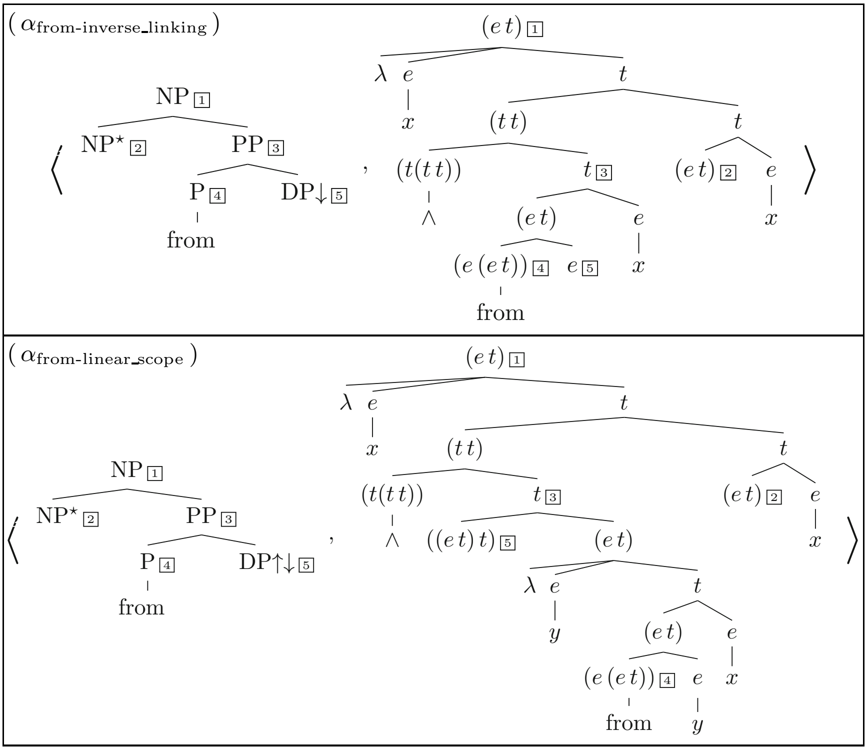


Fig. 5. The example coTAG G_{scope} (part 2)

with the semantic component are still *almost linear*, since the variables appearing more than once are dominated by an atomic type.[5]

3 Abstract Categorial Grammars

An *abstract categorial grammar (ACG)* [3] is a quadruple $\langle \Sigma_1, \Sigma_2, \mathcal{L}, s\rangle$ consisting of an "alphabet" Σ_1 from which underlying structures are determined, an "alphabet" Σ_2 from which possible surface structures are determined, a pair of mappings $\mathcal{L}$ realizing underlying structures as surface structures, and a distinguished "start" symbol s provided by Σ_1. Particular to ACGs is that underlying and surface structures are given as (almost) linear terms of the simply typed lambda calculus.[6] Accordingly, the "alphabets" are higher-order signatures of the form $\Sigma = \langle A, C, \tau\rangle$, where A is a finite set of *atomic types*, C is a finite

[5] More concretely, the notion of *almost linear* employed here is the same as used by Kanazawa [8]: a lambda-term is *almost linear* if it is a lambda I-term such that any variable occurring free more than once in any subterm has an atomic type.

[6] As to the notion of an *almost linear lambda-term* cf. fn. 5.

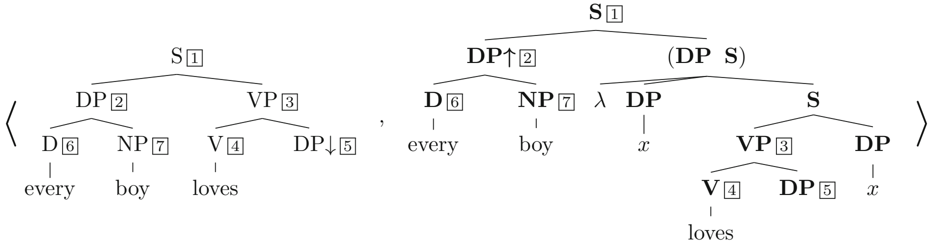


Fig. 6. Cosubstituting *every boy* into the subject position before filling the object position of *loves*: derived syntactic and semantic tree

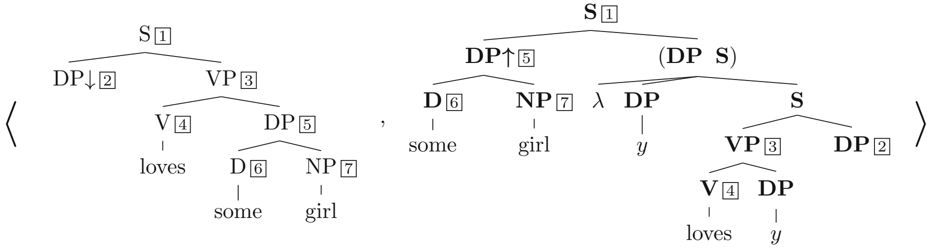


Fig. 7. Cosubstituting *some girl* into the object position before filling the subject position of *loves*: derived syntactic and semantic tree

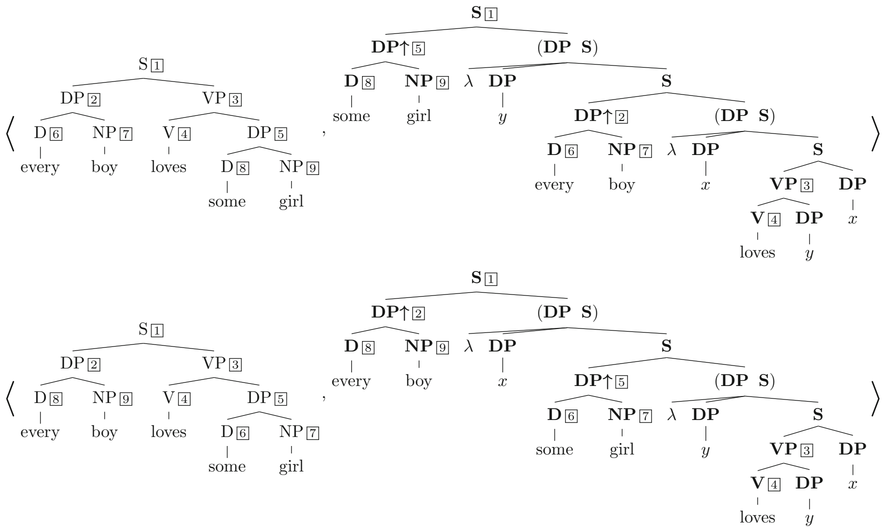


Fig. 8. The two complete derived pairs of structures of *every boy loves some girl*

set of *constants*, and $\tau : C \to \mathcal{T}(A)$ assigns a type to each constant.[7] Given a denumerably infinite set $\mathcal{X}$ of variables, we define a *(type) environment* to be a partial, finite mapping $\Gamma : \mathcal{X} \to \mathcal{T}(A)$, which we typically write as a list $x_1 : \alpha_1, \ldots, x_n : \alpha_n$. Given environments Γ and Δ, the composite type environment Γ, Δ is defined iff their domains are (almost) disjoint, in which case Γ, Δ is the union of Γ and Δ.[8] For $\Lambda(\Sigma)$, the set of *(untyped) lambda-terms* build on Σ,[9] we say that a term $M \in \Lambda(\Sigma)$ has type $\alpha \in \mathcal{T}(A)$ in environment Γ (written $\Gamma \vdash_\Sigma M : \alpha$) just in case it is derivable using the inference rules in Figure 9.

$$\vdash_\Sigma c : \tau(c) \qquad \text{for } c \in C \qquad\qquad x : \alpha \vdash_\Sigma x : \alpha \qquad \text{for } x \in \mathcal{X} \text{ and } \alpha \in \mathcal{T}(A)$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma M : \beta}{\Gamma \vdash_\Sigma (\lambda x.M) : (\alpha\beta)} \qquad\qquad \frac{\Gamma \vdash_\Sigma M : (\alpha\beta) \quad \Delta \vdash_\Sigma N : \alpha}{\Gamma, \Delta \vdash_\Sigma (MN) : \beta}$$

Fig. 9. ACG-inference rules

Given an ACG $G = \langle \Sigma_1, \Sigma_2, \mathcal{L}, s \rangle$ with $\Sigma_i = \langle A_i, C_i, \tau_i \rangle$ and $s \in A_1$, the *abstract language* of G, $A(G)$, is the set $\{M \mid \vdash_{\Sigma_1} M : s\}$. The *object language* of G, $O(G)$, is the set $\{\hat{\theta}(M) \mid M \in A(G)\}$. More concretely, $O(G)$ is the image of $A(G)$ under the pair of mappings $\mathcal{L} = \langle \sigma, \theta \rangle$, referred to as the *lexicon* from Σ_1 to Σ_2, which meets the conditions:

1. $\sigma : A_1 \to \mathcal{T}(A_2)$ is the kernel of the type substitution $\hat{\sigma} : \mathcal{T}(A_1) \to \mathcal{T}(A_2)$ obeying $\hat{\sigma} \restriction A_1 = \sigma$, and $\hat{\sigma}((\alpha\beta)) = (\hat{\sigma}(\alpha)\hat{\sigma}(\beta))$
2. $\theta : C_1 \to \Lambda(\Sigma_2)$ specifies $\hat{\theta} : \Lambda(\Sigma_1) \to \Lambda(\Sigma_2)$ by setting $\hat{\theta}(c) = \theta(c)$, $\hat{\theta}(x) = x$, $\hat{\theta}(MN) = \hat{\theta}(M)\hat{\theta}(N)$, and $\hat{\theta}(\lambda x.M) = \lambda x.\hat{\theta}(M)$
3. $\vdash_{\Sigma_2} \theta(c) : \hat{\sigma}(\tau_1(c))$ for all $c \in C_1$

By abuse of notation, $\mathcal{L}$ will usually be used to denote both σ and θ, and also their respective extensions $\hat{\sigma}$ and $\hat{\theta}$.

An ACG G belongs to the class $\mathcal{ACG}(m, n)$ iff the maximal order of the types of any of its abstract constants is less than or equal to m, and the maximal order of the type assigned to any constant by the lexicon is less than or equal to n. The order of a type α, $\mathrm{ord}(\alpha)$, is given recursively: For each atomic type a, $\mathrm{ord}(a) = 1$. For each two types α and β, $\mathrm{ord}((\alpha\beta))$ is the greater of $\mathrm{ord}(\beta)$ and

[7] Recall that $\mathcal{T}(A)$ is the smallest superset of A closed under pair formation, i.e., $A \subseteq \mathcal{T}(A)$, and if $\alpha, \beta \in \mathcal{T}(A)$ then $(\alpha\,\beta) \in \mathcal{T}(A)$.

[8] *Almost disjoint* is meant to denote the restriction that, if $x : \alpha$ belongs to the $\Gamma \cap \Delta$ then the type assigned to x is atomic, i.e., $\alpha \in A$.

[9] That is, $\Lambda(\Sigma)$ is the smallest set such that $C \cup \mathcal{X} \subseteq \Lambda(\Sigma)$, and such that $(MN) \in \Lambda(\Sigma)$, and $(\lambda x.M) \in \Lambda(\Sigma)$ for $M, N \in \Lambda(\Sigma)$ and $x \in \mathcal{X}$. We omit outer parentheses when writing lambda-terms, and we write $M_1\,M_2\,M_3$ instead of $(M_1\,M_2)\,M_3$, and $M_1 \cdots M_{n+1}$ instead of $(M_1 \cdots M_n)M_{n+1}$ for $n \geq 2$ and $M_1, \ldots, M_{n+1} \in \Lambda(\Sigma)$, where $(M_1 \cdots M_2) = M_1 M_2$. We write $\lambda x.M\,N$ instead of $\lambda x.(M\,N)$, $\lambda x_1 x_2.M$ instead of $\lambda x_1.\lambda x_2.M$, and $\lambda x_1 \ldots x_{n+1}.M$ instead of $\lambda x_1.\lambda x_2 \ldots \lambda x_n.M$ for $n \geq 2$, $x, x_1, \ldots, x_n \in \mathcal{X}$ and $M, N \in \Lambda(\Sigma)$, where $\lambda x_2 \ldots \lambda x_2.M = \lambda x_2.M$.

$\mathrm{ord}(\alpha)+1$. For each m, $\mathcal{ACG}(m)$ denotes the class of all m^{th}-order ACGs, i.e., $\mathcal{ACG}(m)=\bigcup_{n\geq 1}\mathcal{ACG}(m,n)$.

As shown by Salvati [12], if only ACGs of the form $\langle \Sigma_1, \Sigma_2, \mathcal{L}, s\rangle$ are considered where Σ_2 is a string signature,[10] $\mathcal{ACG}(2)$ derives exactly the string languages generated by set-local multicomponent TAGs, or likewise, a series of other weakly equivalent grammar formalisms. Kanazawa [9] proves that, similarly, if Σ_2 is a tree signature,[11] $\mathcal{ACG}(2)$ derives exactly the tree languages generated by, e.g., *context-free graph grammars* [2], or likewise, *hyperedge replacement grammars* [5].

One of the advantages of ACGs is that they provide a logical setting in which the abstract language can be used as a specification of the derivation set of a grammar instantiation of some grammar formalism, and that by applying two different lexicons to the abstract language, we can "simultaneously" obtain a syntactic object language and a semantic object language.

4 CoTAGs as ACGs

In order to translate a coTAG into an ACG, we build on the methods previously developed by de Groote and Pogodalla. De Groote [4] has shown how a regular TAG can be translated into a second-order ACG on trees. Pogodalla [10,11] has shown how those techniques can be extended in order to define a third-order ACG which, "in parallel" to the syntactic derivation, by means of a different semantic object language provides the two different scope readings in a sentence like *every boy likes some girl*.

Let $G_{\text{coTAG}} = \langle V_T, \mathit{Cat}, A_{\mathit{Cat}}, \widehat{\tau}_{\mathit{Cat}}, \mathcal{I}\times\mathcal{I}', \mathcal{A}\times\mathcal{A}', ^\frown, \mathrm{S}\rangle$ be a coTAG.

For expository reasons we assume that no node-label of any tree from $\mathcal{I}\cup\mathcal{A}$ is of the form $\mathrm{S}{\uparrow}\gamma$ or $\mathrm{S}{\uparrow}\gamma{\downarrow}$ for some $\gamma\in\mathit{Cat}(\{\uparrow\}\mathit{Cat})^*$. In other words, the simple category S "is never lifted."

First let $\Sigma_{\text{abs}} = \langle A_{\text{abs}}, C_{\text{abs}}, \tau_{\text{abs}}\rangle$ be the higher-order signature, where

$$\begin{aligned} A_{\text{abs}} &= \{\boldsymbol{\delta}^{\bullet}, \boldsymbol{\delta}^{\bullet}_{\mathrm{A}} \mid \delta\in\mathit{Cat}\} \\ &\quad\cup \left\{\boldsymbol{\zeta{\uparrow}\delta}^{\bullet}, \boldsymbol{\zeta}^{\bullet}, \boldsymbol{\delta}^{\bullet} \,\middle|\, \begin{array}{l} \zeta\in\mathit{Cat}(\{\uparrow\}\mathit{Cat})^* \text{ and } \delta\in\mathit{Cat} \text{ with} \\ \zeta{\uparrow}\delta \text{ or } \zeta{\uparrow}\delta{\downarrow} \text{ labeling a node of some } \alpha\in\mathcal{I}\cup\mathcal{A} \end{array}\right\}, \text{[12]} \\ C_{\text{abs}} &= \{id_X \mid X\in\mathit{Cat}\}\cup\{\overline{\alpha}\mid\alpha\in\mathcal{I}\cup\mathcal{A}\} \\ &\quad\cup\{\overline{\overline{\alpha}}\mid \alpha\in\mathcal{I} \text{ and } \alpha\text{'s root-label belongs to } \mathit{Cat}\{\uparrow\}\mathit{Cat}(\{\uparrow\}\mathit{Cat})^*\} \end{aligned}$$

In order to make the typing function τ_{abs} more precise, consider $\alpha\in\mathcal{I}\cup\mathcal{A}$ with root-label $\gamma\in\mathit{Cat}(\{\uparrow\}\mathit{Cat})^*$. Let $\nu_1,\ldots,\nu_m$ be the interior nodes of α except for the root, and let $\nu_{m+1},\ldots,\nu_{m+n}$ be the substitution sites of α.[13] Furthermore,

[10] That is, Σ_2 consists of the single atomic type o and assigns to each alphabet constant a the type $(o\,o)$.

[11] That is, Σ_2 consists of the single atomic type o and assigns to each constant a of "rank" n the type o^{n+1}, where $o^1=o$ and $o^{n+1}=(o\,o^n)$.

[12] Following the notational convention introduced above, we write $\boldsymbol{\delta{\uparrow}}^{\bullet}$ instead of $\boldsymbol{\delta{\uparrow}\mathrm{S}}^{\bullet}$ for $\delta\in\mathit{Cat}(\{\uparrow\}\mathit{Cat})^*$ and the start symbol S.

[13] Regarding the tree structure, we assume the interior nodes $\nu_1,\ldots,\nu_m$ to be ordered "top-down, left-to-right," and the substitution nodes $\nu_{m+1},\ldots,\nu_{m+n}$ "left-to-right.".

let $\gamma_i \in \mathtt{Cat}$ be the label of ν_i for $1 \leq i \leq m$, and let $\delta_i{\downarrow}$ be the label of ν_{m+i} for $1 \leq i \leq n$, where $\delta_i \in \mathtt{Cat}(\{\uparrow\}\mathtt{Cat})^*$. The type of the constant $\overline{\alpha}$ is defined as follows: for $\alpha \in \mathcal{I}$, let $\tau_{\mathrm{abs}}(\overline{\alpha}) = \boldsymbol{\gamma}^{\bullet}_{\mathrm{A}}\boldsymbol{\gamma}^{\bullet}_{1\mathrm{A}} \cdots \boldsymbol{\gamma}^{\bullet}_{m\mathrm{A}}\boldsymbol{\delta}^{\bullet}_{1} \cdots \boldsymbol{\delta}^{\bullet}_{n}\boldsymbol{\gamma}^{\bullet}$. For $\alpha \in \mathcal{A}$, let $\tau_{\mathrm{abs}}(\overline{\alpha}) = \boldsymbol{\gamma}^{\bullet}_{\mathrm{A}}\boldsymbol{\gamma}^{\bullet}_{1\mathrm{A}} \cdots \boldsymbol{\gamma}^{\bullet}_{m\mathrm{A}}\boldsymbol{\delta}^{\bullet}_{1} \cdots \boldsymbol{\delta}^{\bullet}_{n}\boldsymbol{\gamma}^{\bullet}_{\mathrm{A}}\boldsymbol{\gamma}^{\bullet}_{\mathrm{A}}$. If $\alpha \in \mathcal{I}$, and if $\gamma = \zeta{\uparrow}\delta$ for some $\zeta \in \mathtt{Cat}(\{\uparrow\}\mathtt{Cat})^*$ and $\delta \in \mathtt{Cat}$, the type of the additionally existing constant $\overline{\overline{\alpha}}$ is defined in the following way:[14] $\tau_{\mathrm{abs}}(\overline{\overline{\alpha}}) = \boldsymbol{\gamma}^{\bullet}_{1\mathrm{A}} \cdots \boldsymbol{\gamma}^{\bullet}_{m\mathrm{A}}\boldsymbol{\delta}^{\bullet}_{1} \cdots \boldsymbol{\delta}^{\bullet}_{n}\boldsymbol{\gamma}^{\circ}$.[15] For each $X \in \mathtt{Cat}$, the type of the constant id_X is defined by $\tau_{\mathrm{abs}}(id_X) = \boldsymbol{X}^{\bullet}_{\mathrm{A}}$.

In the way the abstract constants resulting from elementary trees are typed, adjunction at the adjunction sites of an elementary tree becomes obligatory. The abstract constants id_X for $X \in \mathtt{Cat}$ have been introduced to provide the possibility of "vacuous" adjunction.

As to our example coTAG G_{scope}, the abstract typing function τ_{abs} assigns the following types to the abstract constants resulting from the lexical entries:[16]

$$\begin{aligned}
\tau_{\mathrm{abs}}(\overline{\overline{\alpha}}_{\mathrm{every}}) &= \mathbf{D}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\ (\mathbf{DP}^{\bullet}\ \mathbf{S}^{\bullet})\ \mathbf{S}^{\bullet} &\qquad \tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{every}}) &= \mathbf{D}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\ \mathbf{DP{\uparrow}}^{\bullet}\\
\tau_{\mathrm{abs}}(\overline{\overline{\alpha}}_{\mathrm{some}}) &= \mathbf{D}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\ (\mathbf{DP}^{\bullet}\ \mathbf{S}^{\bullet})\ \mathbf{S}^{\bullet} &\qquad \tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{some}}) &= \mathbf{D}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\ \mathbf{DP{\uparrow}}^{\bullet}
\end{aligned}$$

$$\begin{aligned}
\tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{boy}}) &= \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\\
\tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{girl}}) &= \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}\\
\tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{loves}}) &= \mathbf{S}^{\bullet}_{\mathrm{A}}\ \mathbf{VP}^{\bullet}_{\mathrm{A}}\ \mathbf{V}^{\bullet}_{\mathrm{A}}\ \mathbf{DP}^{\bullet}\ \mathbf{DP}^{\bullet}\ \mathbf{S}^{\bullet}\\
\tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{from\_inverse\_linking}}) &= \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{PP}^{\bullet}_{\mathrm{A}}\ \mathbf{P}^{\bullet}_{\mathrm{A}}\ \mathbf{DP}^{\bullet}\ \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}_{\mathrm{A}}\\
\tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{from\_linear\_scope}}) &= \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{PP}^{\bullet}_{\mathrm{A}}\ \mathbf{P}^{\bullet}_{\mathrm{A}}\ \mathbf{DP{\uparrow}}^{\bullet}\ \mathbf{NP}^{\bullet}_{\mathrm{A}}\ \mathbf{NP}^{\bullet}_{\mathrm{A}}
\end{aligned}$$

Representing Derivations. We next give the ACG $G_{\mathrm{der}} = \langle \Sigma_{\mathrm{abs}}, \Sigma_{\mathrm{der}}, \mathcal{L}_{\mathrm{der}}, \mathbf{S}^{\bullet} \rangle$. The higher-order signature Σ_{abs} provides us with an abstract language representing the coTAG-derivations including information on the derivational order. This is achieved qua lambda-terms whose order are greater than 2. The higher-order signature Σ_{der} provides us, via the lexicon $\mathcal{L}_{\mathrm{der}}$, with the object language. In case we stick to coTAGs in a normalized form (see below), the object language represents the "plain" TAG-derivation trees of the coTAG, i.e., the object language provides representations of the derivations which (without distinguishing between the operations of *substitution* and *cosubstitution*) keep track only of which operation was applied at which node by using which trees, and which do not keep track of the relative order in which the operations took place.

More concretely, we have $\Sigma_{\mathrm{der}} = \langle A_{\mathrm{abs}}, \{\beta' \mid \beta \in C_{\mathrm{abs}}\}, \tau_{\mathrm{der}} \rangle$, and the lexicon $\mathcal{L}_{\mathrm{der}}$ of the ACG G_{der} and the typing function τ_{der} of the higher-order signature

[14] For each $\gamma \in \mathtt{Cat}(\{\uparrow\}\mathtt{Cat})^*$ such that $\gamma = \zeta{\uparrow}\delta$ for some $\zeta \in \mathtt{Cat}(\{\uparrow\}\mathtt{Cat})^*$ and $\delta \in \mathtt{Cat}$, we take the "circled" boldface version $\boldsymbol{\gamma}^{\circ}$ to denote the type $(\boldsymbol{\zeta}^{\bullet}\boldsymbol{\delta}^{\bullet})\,\boldsymbol{\delta}^{\bullet}$.

[15] That is, we allow adjunction at the root of an initial tree only if the root-label is a "simple" category.

[16] To avoid notational overload, we use the name of a lexical entry of G_{scope}, when referring to the abstract constant typed with regard to the first, i.e. the syntactic, component of the entry.

Σ_{der} are given in the following way: as a mapping from types to types, $\mathcal{L}_{\mathrm{der}}$ functions as identity mapping. For each $X \in \boldsymbol{\mathit{Cat}}$, the abstract constant id_X is mapped to $\lambda x.\, x$ under $\mathcal{L}_{\mathrm{der}}$. For $\alpha \in \mathcal{I} \cup \mathcal{A}$, we have $\tau_{\mathrm{der}}(\overline{\alpha}') = \tau_{\mathrm{abs}}(\overline{\alpha})$ and $\mathcal{L}_{\mathrm{der}}(\overline{\alpha}) = \overline{\alpha}'$. For $\alpha \in \mathcal{I}$ with root-label γ such that $\gamma = \zeta{\uparrow}\delta$ for some $\zeta \in \boldsymbol{\mathit{Cat}}(\{\uparrow\}\boldsymbol{\mathit{Cat}})^*$ and $\delta \in \boldsymbol{\mathit{Cat}}$, and with $\tau_{\mathrm{abs}}(\overline{\overline{\alpha}}) = \boldsymbol{\gamma}^{\bullet}_{1\mathrm{A}} \cdots \boldsymbol{\gamma}^{\bullet}_{m\mathrm{A}} \boldsymbol{\delta}^{\bullet}_{1} \cdots \boldsymbol{\delta}^{\bullet}_{n} \boldsymbol{\gamma}^{\circ}$, we also have $\tau_{\mathrm{der}}(\overline{\overline{\alpha}}') = \boldsymbol{\gamma}^{\bullet}_{1\mathrm{A}} \cdots \boldsymbol{\gamma}^{\bullet}_{m\mathrm{A}} \boldsymbol{\delta}^{\bullet}_{1} \cdots \boldsymbol{\delta}^{\bullet}_{n} \boldsymbol{\zeta}^{\bullet}$ and $\mathcal{L}_{\mathrm{der}}(\overline{\alpha}) = \lambda y_1 \ldots y_{m+n} f.\, f(\overline{\overline{\alpha}}' y_1 \cdots y_{m+n})$, where the variable f is of type $(\boldsymbol{\zeta}^{\bullet}\, \boldsymbol{\delta}^{\bullet})$. That is, in case the root-label of an elementary tree consists of a (possibly multiple) lifted type, we "undo" the (last) type lifting.

Put differently, the lexicon $\mathcal{L}_{\mathrm{der}}$ is quite simple, and is based on viewing higher-order constants as literally type lifted versions of themselves:[17] accordingly, an abstract constant as, e.g., $\overline{\overline{\alpha}}_{\mathrm{everyone}}$ of type $((\mathbf{DP}^{\bullet}\ \mathbf{S}^{\bullet})\ \mathbf{S}^{\bullet})$ would be mapped to $\lambda P.P\, \overline{\overline{\alpha}}'_{\mathrm{everyone}}$, where the variable P is of type $(\mathbf{DP}^{\bullet}\ \mathbf{S}^{\bullet})$, and where $\overline{\overline{\alpha}}'_{\mathrm{everyone}}$ is the object constant of type $\mathbf{DP}^{\bullet}$ associated with the elementary tree for *everyone* in the TAG-grammar obtained from the syntactic component of the corresponding lexical entry $\alpha_{\mathrm{everyone}}$ of the coTAG-grammar by removing all arrow annotations. Abstract constants without arrow annotations are simply mapped to "themselves" as is, e.g., demonstrated by $\mathcal{L}_{\mathrm{der}}(\overline{\alpha}_{\mathrm{loves}}) = \overline{\alpha}'_{\mathrm{loves}}$ and $\tau_{\mathrm{der}}(\overline{\alpha}'_{\mathrm{loves}}) = \tau_{\mathrm{abs}}(\overline{\alpha}_{\mathrm{loves}})$. Applying this lexicon to a simple example is illustrative:[18]

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathcal{L}_{\mathrm{der}}(\overline{\overline{\alpha}}_{\mathrm{everyone}}(\lambda x.\, \overline{\alpha}_{\mathrm{loves}}\, x\, \overline{\alpha}_{\mathrm{mary}}))} \\
= & \mathcal{L}_{\mathrm{der}}(\overline{\overline{\alpha}}_{\mathrm{everyone}})\, \mathcal{L}_{\mathrm{der}}(\lambda x.\, \overline{\alpha}_{\mathrm{loves}}\, x\, \overline{\alpha}_{\mathrm{mary}}) \\
= & \mathcal{L}_{\mathrm{der}}(\overline{\overline{\alpha}}_{\mathrm{everyone}})\, (\lambda x.\, \mathcal{L}_{\mathrm{der}}(\overline{\alpha}_{\mathrm{loves}})\, \mathcal{L}_{\mathrm{der}}(x)\, \mathcal{L}_{\mathrm{der}}(\overline{\alpha}_{\mathrm{mary}})) \\
= & (\lambda P.\, P\, \overline{\overline{\alpha}}'_{\mathrm{everyone}})(\lambda x.\, \overline{\alpha}'_{\mathrm{loves}}\, x\, \overline{\alpha}'_{\mathrm{mary}}) \\
\twoheadrightarrow_{\beta} & (\lambda x.\, \overline{\alpha}'_{\mathrm{loves}}\, x\, \overline{\alpha}'_{\mathrm{mary}})\, \overline{\overline{\alpha}}'_{\mathrm{everyone}} \\
\twoheadrightarrow_{\beta} & \overline{\alpha}'_{\mathrm{loves}}\, \overline{\overline{\alpha}}'_{\mathrm{everyone}}\, \overline{\alpha}'_{\mathrm{mary}}
\end{array}$$

Semantic Representations. In Sec. 2 we have explained, how the derived semantic trees of a coTAG determine closed well-typed lambda-terms associated with the yields of the trees, cf. page 2. The ACG $G_{\mathrm{sem}} = \langle \Sigma_{\mathrm{abs}}, \Sigma_{\mathrm{sem}}, \mathcal{L}_{\mathrm{sem}}, \mathbf{S}^{\bullet} \rangle$ will do nothing but making the lambda-terms of the finally derived semantic trees concrete as its object language. Therefore, the "flat" lambda-term representation of intermediately derived semantic trees—which takes into account only the "open slots" of the yield provided by coTAG-nonterminals—has to be enriched by information about those interior nodes which allow adjunction. We have to lambda-abstract about those sites by variables of appropriate type as well. As far as the general case is concerned, we will skip further technical

[17] In the same way as, e.g., the generalized quantifier $\lambda P.P(\mathbf{john})$ with variable P of type $(e\,t)$ is the type lifted version of the individual constant **john**.

[18] If we were more precise, we would actually be concerned with a lambda-term like $\overline{\overline{\alpha}}_{\mathrm{everyone}}(\lambda x.\, \overline{\alpha}_{\mathrm{loves}}\, id_{\mathrm{S}}\, id_{\mathrm{VP}}\, id_{\mathrm{V}}\, x\, \overline{\alpha}_{\mathrm{mary}})$. For better readability we leave out any possible instantiations of "vacuous" adjunction indicated by a functional application to a constant id_X.

details how to arrive at those lambda-terms, and look at our coTAG G_{scope} as an example case below. Let us assume here, that for each semantic elementary tree $\alpha_{\mathrm{sem}} \in \mathcal{I}' \cup \mathcal{A}'$, $(\alpha_{\mathrm{sem}})_\lambda$ is the corresponding lambda-term.

We set $\Sigma_{\mathrm{sem}} = \langle A_{\mathit{Cat}}, \{\mathbf{a} \,|\, a \in V_T \cup \mathit{Con}\}, \tau_{\mathrm{sem}} \rangle$. In order to define τ_{sem} we assume, without loss of generality, that each $a \in V_T \cup \mathit{Con}$ appears as the leaf-label of some elementary semantic tree of our coTAG-lexicon. For $a \in V_T \cup \mathit{Con}$, we choose a node ν of some elementary semantic tree labeled a and let $\mathit{label}(\mu)$ be the label of the parent node of ν, μ. We set $\tau_{\mathrm{sem}}(\mathbf{a}) = \mathit{label}(\mu)$.[19]

Defining the lexicon $\mathcal{L}_{\mathrm{sem}}$, for each abstract atomic type $\boldsymbol{\delta}^\bullet$ arising from some $\delta \in \mathit{Cat}(\{\uparrow\}\mathit{Cat})^*$, we let $\mathcal{L}_{\mathrm{sem}}(\boldsymbol{\delta}^\bullet) = \widehat{\tau}_{\mathit{Cat}}(\delta)$, and if $\delta \in \mathit{Cat}$, we also let $\mathcal{L}_{\mathrm{sem}}(\boldsymbol{\delta}^\bullet_{\mathrm{A}}) = \widehat{\tau}_{\mathit{Cat}}(\delta)\ \widehat{\tau}_{\mathit{Cat}}(\delta)$. For each abstract constant $\overline{\alpha}_{\mathrm{syn}}$, respectively, $\overline{\overline{\alpha}}_{\mathrm{syn}}$, arising from some lexical coTAG-entry $\langle \alpha_{\mathrm{syn}}, \alpha_{\mathrm{sem}} \rangle \in (\mathcal{I} \times \mathcal{I}') \cup (\mathcal{A} \cup \mathcal{A}')$ with $\alpha_{\mathrm{syn}} \in \mathcal{I} \cup \mathcal{A}$ and $\alpha_{\mathrm{sem}} \in \mathcal{I}' \cup \mathcal{A}'$, we let $\mathcal{L}_{\mathrm{sem}}(\overline{\alpha}_{\mathrm{syn}}) = (\alpha_{\mathrm{sem}})_\lambda$, respectively, $\mathcal{L}_{\mathrm{sem}}(\overline{\overline{\alpha}}_{\mathrm{syn}}) = (\alpha_{\mathrm{sem}})_\lambda$.

As an example consider the coTAG G_{scope} as given in Fig. 4 and 5. The function τ_{sem} assigning types to the (semantic) object constants is given by:

$$\begin{array}{lll}
\tau_{sem}(\mathbf{every}) = (e\ t)\ (e\ t)\ t & \tau_{sem}(\mathbf{boy}) = e\ t & \tau_{sem}(\mathbf{loves}) = e\ e\ t \\
\tau_{sem}(\mathbf{some})\ = (e\ t)\ (e\ t)\ t & \tau_{sem}(\mathbf{girl}) = e\ t & \tau_{sem}(\mathbf{from}) = e\ e\ t \\
 & \tau_{sem}(\wedge) \quad = t\ t\ t &
\end{array}$$

The lexicon, $\mathcal{L}_{\mathrm{sem}}$, connecting abstract atomic types with (semantic) object types and abstract constants with (semantic) object lambda-terms is given by

$$\begin{array}{ll}
\mathcal{L}_{\mathrm{sem}}(\mathbf{D}^\bullet) \quad = (e\ t)\ (e\ t)\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{D}^\bullet_{\mathrm{A}}) \quad = \mathcal{L}_{\mathrm{sem}}(\mathbf{D}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{D}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{DP}^\bullet) \quad = e & \mathcal{L}_{\mathrm{sem}}(\mathbf{DP}^\bullet_{\mathrm{A}}) = \mathcal{L}_{\mathrm{sem}}(\mathbf{DP}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{DP}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{DP}{\uparrow}^\bullet) = (e\ t)\ t & \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{P}^\bullet) \quad = e\ e\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{P}^\bullet_{\mathrm{A}}) \quad = \mathcal{L}_{\mathrm{sem}}(\mathbf{P}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{P}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{PP}^\bullet) \quad = e\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{PP}^\bullet_{\mathrm{A}}) = \mathcal{L}_{\mathrm{sem}}(\mathbf{PP}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{PP}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{V}^\bullet) \quad = e\ e\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{V}^\bullet_{\mathrm{A}}) \quad = \mathcal{L}_{\mathrm{sem}}(\mathbf{V}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{V}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{VP}^\bullet) \quad = e\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{VP}^\bullet_{\mathrm{A}}) = \mathcal{L}_{\mathrm{sem}}(\mathbf{VP}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{VP}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{NP}^\bullet) \quad = e\ t & \mathcal{L}_{\mathrm{sem}}(\mathbf{NP}^\bullet_{\mathrm{A}}) = \mathcal{L}_{\mathrm{sem}}(\mathbf{NP}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{NP}^\bullet) \\
\mathcal{L}_{\mathrm{sem}}(\mathbf{S}^\bullet) \quad = t & \mathcal{L}_{\mathrm{sem}}(\mathbf{S}^\bullet_{\mathrm{A}}) \quad = \mathcal{L}_{\mathrm{sem}}(\mathbf{S}^\bullet)\ \mathcal{L}_{\mathrm{sem}}(\mathbf{S}^\bullet)
\end{array}$$

and furthermore, for

$$\begin{array}{lll}
d \quad := \mathcal{L}_{\mathrm{sem}}(\mathbf{D}^\bullet) & p \ := \mathcal{L}_{\mathrm{sem}}(\mathbf{P}^\bullet) & vp := \mathcal{L}_{\mathrm{sem}}(\mathbf{VP}^\bullet) \\
dp \quad := \mathcal{L}_{\mathrm{sem}}(\mathbf{DP}^\bullet) & pp := \mathcal{L}_{\mathrm{sem}}(\mathbf{PP}^\bullet) & np := \mathcal{L}_{\mathrm{sem}}(\mathbf{NP}^\bullet) \\
dp{\uparrow} := \mathcal{L}_{\mathrm{sem}}(\mathbf{DP}{\uparrow}^\bullet) & v \ := \mathcal{L}_{\mathrm{sem}}(\mathbf{V}^\bullet) & s \quad := \mathcal{L}_{\mathrm{sem}}(\mathbf{S}^\bullet)
\end{array}$$

[19] By i.1) on page 2, *label*(μ) is uniquely determined independently of the choice of ν.

by[20]

$$
\begin{aligned}
\mathcal{L}_{\text{sem}}(\overline{\overline{\alpha}}_{\text{every}}) &= \mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{every}}) = \lambda F^{(d\ d)} x^{np}.\, (F\ \textbf{every})\, x \\
\mathcal{L}_{\text{sem}}(\overline{\overline{\alpha}}_{\text{some}}) &= \mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{some}}) = \lambda F^{(d\ d)} x^{np}.\, (F\ \textbf{some})\, x
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{boy}}) &= \lambda F^{(np\ np)}.\, F\ \textbf{boy} \\
\mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{girl}}) &= \lambda F^{(np\ np)}.\, F\ \textbf{girl}
\end{aligned}
$$

$$
\mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{loves}}) = \lambda F^{(s\ s)} G^{(vp\ vp)} H^{(v\ v)} y^{dp} x^{dp}.\, F(\,(G\,(\,(H\ \textbf{loves}\,)\,x\,)\,)\,y\,)
$$

$$
\begin{aligned}
\mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{from\_inverse\_linking}}) = \lambda F^{(np\ np)} G^{(pp\ pp)} H^{(p\ p)} P^{np} y^{dp} x^{dp}. \\
F\,(\wedge\,(G\,(\,(H\ \textbf{from}\,)\,y\,x)\,)\,(\,P\,x\,)\,)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}_{\text{sem}}(\overline{\alpha}_{\text{from\_linear\_scope}}) = \lambda F^{(np\ np)} G^{(pp\ pp)} H^{(p\ p)} Q^{dp\uparrow} P^{np} x^{dp}. \\
F(\,\wedge\,(\,G\,(\,Q\,(\lambda y^{dp}.\,(\,H\ \textbf{from}\,)\,y\,x)\,)\,)\,(\,P\,x)\,)
\end{aligned}
$$

while, finally, we have

$$
\mathcal{L}_{\text{sem}}(id_X) \ = \ \lambda x^{\mathcal{L}_{\text{sem}}(\mathbf{X}^{\bullet})}.\, x \qquad \text{for } X \in \mathit{Cat}
$$

On (co)Substitution Nodes. For $\langle V_T, \mathit{Cat}, A_{\mathit{Cat}}, \widehat{\tau}_{\mathit{Cat}}, \mathcal{I}\times\mathcal{I}', \mathcal{A}\times\mathcal{A}', ^\frown, \mathrm{S}\rangle$, the coTAG G_{coTAG} we started with, the nodes of the derivable syntactic trees are labeled with strings from $\mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^{*}\{\downarrow\}^{?}$. These labels come with seemingly implicit compositional structure by means of the occurrences of ↑ (and ↓): given some set of atomic types A_{Types}, and assigning a type $\boldsymbol{\gamma}^{\blacklozenge}$ from $\mathcal{T}(A_{\mathit{Types}})$ to each $\gamma \in \mathit{Cat}$, it is straightforward to interpret ↑ as a type lifting operation (and ↓ as the identity function on types), thereby recursively assigning the type $\boldsymbol{\zeta{\downarrow}}^{\blacklozenge} := \boldsymbol{\zeta}^{\blacklozenge}$ to $\zeta{\downarrow}$, and $\boldsymbol{\zeta{\uparrow}\delta}^{\blacklozenge} := ((\boldsymbol{\zeta}^{\blacklozenge}\,\boldsymbol{\delta}^{\blacklozenge})\,\boldsymbol{\delta}^{\blacklozenge})$ to $\zeta{\uparrow}\delta$ for $\zeta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat}\,)^{*}$ and $\delta \in \mathit{Cat}$.[21]

As far as the operation of substitution is concerned, we are not exploiting this structural potential in the translation to ACGs. Instead, given a substitution site with syntactic node label $\gamma{\downarrow}$ we are treating γ as an atomic category (label) with regard to the coTAG, irrespective of whether γ contains any ↑ characters. We are doing so in terms of the abstract atomic type $\gamma^{\bullet}$ belonging to A_{abs}. Looking at the example above, instances of such a $\gamma^{\bullet}$ occur in terms of $\mathbf{DP}{\uparrow}^{\bullet}$ within the types assigned to $\overline{\alpha}_{\text{every}}$, $\overline{\alpha}_{\text{some}}$ and $\overline{\alpha}_{\text{from\_linear\_scope}}$ via τ_{abs}.

As far as the operation of cosubstitution is concerned, the compositional structure potential of a syntactic node label is exploited in the translation to ACGs only in its "simplest" non-recursive form: if there are any, only the last instance of ↑ within a syntactic root label is interpreted as a type lifting operation. More

[20] As usual a superscript connected with a variable denotes the type of that variable. In the example, the variables F, G, H indicate potential adjunction sites. Again we use the name of a lexical entry of G_{scope}, when referring to the abstract constant typed with regard to the first component of the entry, cf. fn. 16.

[21] $\widehat{\tau}_{\mathit{Cat}}$ as defined on page 120 is a particular instance of such a recursively defined function $\cdot^{\blacklozenge}$, assigning the types $\boldsymbol{\zeta}$ and $(\boldsymbol{\zeta}\,\boldsymbol{\delta})\,\boldsymbol{\delta}$ from $\mathcal{T}(A_{\mathit{Cat}})$ to the same ζ and $\zeta{\uparrow}\delta$, respectively.

concretely, for $\zeta \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat})^*$ and $\delta \in \mathit{Cat}$ such that $\zeta{\uparrow}\delta$ labels the root of a syntactic tree, the abstract type associated with that node is $(\boldsymbol{\zeta}^\bullet\,\boldsymbol{\delta}^\bullet)\,\boldsymbol{\delta}^\bullet$, where $\boldsymbol{\zeta}^\bullet$ and $\boldsymbol{\delta}^\bullet$ are atomic abstract types from A_{abs}. Looking at the example above, instances of such a $(\boldsymbol{\zeta}^\bullet\,\boldsymbol{\delta}^\bullet)\,\boldsymbol{\delta}^\bullet$ occur in terms of $(\mathbf{DP}^\bullet\,\mathbf{S}^\bullet)\,\mathbf{S}^\bullet$ within the types assigned to $\overline{\overline{\alpha}}_{\mathrm{every}}$ and $\overline{\overline{\alpha}}_{\mathrm{some}}$ via τ_{abs}.

Technically of course, it would be straightforwardly possible to exploit "all the way down" the compositional structure implicit in the syntactic node labels of the coTAG in an ACG-translation: starting from our Σ_{abs} this would essentially be achieved by

- assuming for each $\delta \in \mathit{Cat}$, $\boldsymbol{\delta}^\blacklozenge$ and $\boldsymbol{\delta}^\bullet_{\mathrm{A}}$ to be atomic types,
- replacing A_{abs} by $\{\boldsymbol{\delta}^\blacklozenge, \boldsymbol{\delta}^\bullet_{\mathrm{A}} \,|\, \delta \in \mathit{Cat}\}$, an d
- replacing $\boldsymbol{\gamma}^\bullet$ by $\boldsymbol{\gamma}^\blacklozenge$ for each $\gamma \in \mathit{Cat}\,(\{\uparrow\}\mathit{Cat})^*$, where $\boldsymbol{\gamma}^\blacklozenge \in \mathcal{T}(\{\boldsymbol{\delta}^\blacklozenge \,|\, \delta \in \mathit{Cat}\})$ is the type assigned to γ, recursively defined as above.

This potential alternative views a coTAG as an ACG of arbitrarily high, but lexically fixed, order. Although rather canonical from a formal perspective, this alternative encoding of coTAGs into ACGs is incorrect—cosubstitution as defined requires exactly third-order terms. The coTAG-grammar fragment above, e.g., is such that its alternative translation into an ACG does not preserve the form-meaning relation. This is due to the fact that higher-order types can be "lowered" by hypothetical reasoning, which cannot be simulated in the original coTAG. More concretely, in the alternative ACG-encoding, a substitution site with syntactic label DP$\uparrow\downarrow$ selects an argument of higher-order abstract type $(\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge)\ \mathbf{S}^\blacklozenge$, and not atomic type $\mathbf{DP}{\uparrow}^\bullet$. Therefore, in ACG-terms, the substitution site can be given as argument a term $\lambda P^{(\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge)}.\,P\,y$, where y is an unbound variable of type $\mathbf{DP}^\blacklozenge$. The variable y can be abstracted over at a later point in time, giving rise to a term of type $\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge$, which can then be the argument to the cosubstitutor of type $(\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge)\ \mathbf{S}^\blacklozenge$. Thus, the substitution for a cosubstitutor was only "tricked" by the combinator $\lambda P.\,P\,y$ into "thinking" that it had already been given the correct argument.

The lexical entry $\alpha_{\mathrm{from\_linear\_scope}}$ from G_{scope} provides a case in point. This entry would be translated into the abstract constant $\overline{\alpha}_{\mathrm{from\_linear\_scope}}$ of fourth-order type $\mathbf{NP}^\bullet_{\mathrm{A}}\,\mathbf{PP}^\bullet_{\mathrm{A}}\,\mathbf{P}^\bullet_{\mathrm{A}}\,((\,\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge\,)\,\mathbf{S}^\blacklozenge\,)\,\mathbf{NP}^\bullet_{\mathrm{A}}\,\mathbf{NP}^\bullet_{\mathrm{A}}$, and as a consequence, the following term would be well-typed:[22]

$$\overline{\alpha}_{\mathrm{every}}\,\overline{\alpha}_{\mathrm{city}}\,(\,\lambda y.\,\overline{\alpha}_{\mathrm{some}}\,(\,\overline{\alpha}_{\mathrm{boy}}\,(\,\overline{\alpha}_{\mathrm{from\_linear\_scope}}\,(\,\lambda P.\,P\,y\,)\,)\,)\,(\,\lambda x.\,\overline{\alpha}_{\mathrm{left}}\,x\,)\,)\ ,$$

where the variables x and y are of type $\mathbf{DP}^\blacklozenge$, and the variable P is of type $\mathbf{DP}^\blacklozenge\,\mathbf{S}^\blacklozenge$. This term, however, evaluates semantically to the inverse scope reading of the noun phrase *some boy from every city*, despite the fact that the elementary tree $\alpha_{\mathrm{from\_linear\_scope}}$ was used.

[22] We owe this example to the anonymous reviewer mentioned in the acknowledgments. Again, for better readability we ignore "vacuous" adjunction indicated by a functional application to a constant id_X.

Thus, the alternative translation into higher-order ACGs is not faithful to Barker's original presentation. On the other hand, the higher-order ACG allows for a single lexical item, $\alpha_{\text{from\_linear\_scope}}$, to derive both inverse and linear scope readings, which might be seen as more elegant than the original coTAG-analysis. This behavior seems to be obtainable in the coTAG-formalism if we alter the definition of cosubstitution so as to also allow cosubstitution of trees with root-label $\zeta{\uparrow}\delta$ into nodes labeled $\zeta{\uparrow}\delta{\downarrow}$.

5 Conclusion

We have shown how Barker's ideas about coTAGs have a natural home in the ACG-formalism. Our formalization makes explicit the graph structure of coTAG-derivations, in particular the dependence in cosubstitution on both the substitution node (in a particular elementary tree) and the derivation over which it takes scope.

The ACG-perspective allows us to better understand the surprising fact that coTAGs have the same strong generative capacity as regular TAGs: it is due to the fact that the "lifted" underlying derivation is first lowered back into a regular TAG-derivation on the syntactic side, and then interpreted to obtain a derived tree. The crucial piece in this puzzle is our normal form theorem, which states that every coTAG (qua ACG) has an equivalent third-order variant.

This also highlights the fact that "strong generative capacity" in the TAG-sense, i.e. the sets of structures derivable, is not the most insightful measure of the complexity a grammar formalism; rather it is the relation between derived objects and derivations (which stand proxy for meanings in a compositional system) which provides the most insight into the grammar formalism. According to this measure, coTAGs are much more complex than regular TAGs, whose derivation sets are regular tree languages, and therefore, second-order ACGs.

Acknowledgments. We are grateful to an anonymous reviewer for pushing us to a much more rigorous presentation of Section 4. Any remaining lack of clarity and errors are due to us.

References

1. Barker, C.: Cosubstitution, derivational locality, and quantifier scope. In: Proceedings of the Tenth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+10), New Haven, CT, pp. 135–142 (2010)
2. Bauderon, M., Courcelle, B.: Graph expressions and graph rewriting. Mathematical Systems Theory 20, 83–127 (1987)
3. de Groote, P.: Towards abstract categorial grammars. In: 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001), Toulouse, pp. 252–259. ACL (2001)
4. de Groote, P.: Tree-adjoining grammars as abstract categorial grammars. In: Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+6), Venezia, pp. 145–150 (2002)

5. Habel, A., Kreowski, H.J.: Some Structural Aspects of Hypergraph Languages Generated by Hyperedge Replacement. In: Brandenburg, F.J., Vidal-Naquet, G., Wirsing, M. (eds.) STACS 1987. LNCS, vol. 247, pp. 207–219. Springer, Heidelberg (1987)
6. Joshi, A.K.: An introduction to tree adjoining grammars. In: Manaster-Ramer, A. (ed.) Mathematics of Language, pp. 87–114. John Benjamins, Amsterdam (1987)
7. Joshi, A.K., Schabes, Y.: Tree adjoining grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 69–124. Springer, Heidelberg (1997)
8. Kanazawa, M.: Parsing and generation as datalog queries. In: 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), Prague, pp. 176–183. ACL (2007)
9. Kanazawa, M.: Second-order abstract categorial grammars as hyperedge replacement grammars. Journal of Logic, Language and Information 19, 137–161 (2010)
10. Pogodalla, S.: Computing semantic representation. Towards ACG abstract terms as derivation trees. In: Proceedings of the Seventh International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+7), Vancouver, BC, pp. 64–71 (2004)
11. Pogodalla, S.: Ambiguïté de portée et approche fonctionnelle des grammaires d'arbres adjoints. In: Traitement Automatique des Langues Naturelles (TALN 2007), Toulouse, 10 pages (2007)
12. Salvati, S.: Encoding second order string ACG with deterministic tree walking transducers. In: Wintner, S. (ed.) Proceedings of FG 2006: The 11th Conference on Formal Grammar, pp. 143–156. CSLI Publications, Stanford (2007)

Gapping as Like-Category Coordination

Yusuke Kubota[1] and Robert Levine[2]

[1] University of Tokyo
yk@phiz.c.u-tokyo.ac.jp
[2] Ohio State University
levine@ling.ohio-state.edu

Abstract. We propose a version of Type-Logical Categorial Grammar (TLCG) which combines the insights of standard TLCG (Morrill 1994, Moortgat 1997) in which directionality is handled in terms of forward and backward slashes, and more recent approaches in the CG literature which separate directionality-related reasoning from syntactic combinatorics by means of λ-binding in the phonological component (Oehrle 1994, de Groote 2001, Muskens 2003). The proposed calculus recognizes both the directionality-sensitive modes of implication ($/$ and $\backslash$) of the former and the directionality-insensitive mode of implication tied to phonological λ-binding in the latter (which we notate here as $|$).

Empirical support for the proposed system comes from the fact that it enables a straightforward treatment of Gapping, a phenomenon that has turned out to be extremely problematic in the syntactic literature including CG-based approaches.

Keywords: Gapping, coordination, Type-Logical Categorial Grammar, Lambda Grammar, scope, phenogrammar, tectogrammar.

1 (Apparent) Anomalies of Gapping

The examples in (1) are instances of *Gapping*:

(1) a. Leslie bought a CD, and Robin, a book.
b. I gave Leslie a book, and she a CD.
c. Terry can go there with me, and Pat with you.

Gapping is a type of non-canonical coordination, but what distinguishes it from other kinds of non-canonical coordinations is that the strings which appear to be coordinated do not look very much like each other. For example, in cases of nonconstituent coordination such as *I told the same joke to Robin on Friday and (to) Leslie on Sunday*, or in examples of Right-Node Raising (RNR), it is possible to identify two coordinated substrings which are parallel up to the point where they combine with the rest of the sentence in which they appear; the problem is only that expressions such as *to Robin on Friday and (to) Leslie on Sunday* are not phrase structure constituents, nor are the partial clauses in RNR. But in the case of Gapping, we seem to be coordinating a whole clause with a sequence of

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 135–150, 2012.

words which would be a clause if a copy of the verb in the first conjunct were introduced into the second conjunct. As they stand, however, *Leslie bought a CD* has a completely different status from *Robin a book*.

For this reason, Gapping has continued to pose a difficult challenge in the tradition of both phrase structure grammar and categorial grammar. In categorial grammar (CG), there have been proposals both in the tradition of CCG (Steedman 1990) and TLCG (Morrill and Solias 1993, Morrill 1994, Hendriks 1995, Morrill and Merenciano 1996, Morrill et al. 2011). These proposals share an important key analytic intuition (which we take to be basically correct) which views Gapping as a case of discontinuous constituency: in Gapping, if the shared verb is stripped off from the left conjunct, then it has the same combinatorial property and semantic type as the right conjunct, which therefore supports coordination under the law of coordination of likes. The challenge essentially lies in characterizing precisely the status of the two coordinated conjuncts manifesting (in this view) discontinuity. The most recent proposal by Morrill et al. (2011) improves on previous related approaches in this respect, but it still suffers from empirical shortcomings in not straightforwardly extending to cases in which Gapping interacts with other phenomena which themselves manifest discontinuous constituency, as we will discuss in section 3.

A further challenge for any analysis of Gapping comes from the scopal properties of modal and negative auxiliaries, in examples like the following (Oehrle 1987):

(2) a. Kim didn't play bingo or Sandy sit at home all evening.
b. Mrs. J can't live in Boston and Mr. J in LA.

The only available interpretation of (2a) is $\neg\varphi \wedge \neg\psi$ ($\equiv \neg(\varphi \vee \psi)$), where φ is the proposition expressed by *Kim played bingo* and ψ the proposition expressed by *Sandy sat at home all evening*. (2b) is—at least for some speakers including one of the authors—ambiguous between the reading in which the modal *can't* scopes over the conjunction and one in which it scopes below it; the latter reading can be made prominent by having an intonational break between the first and second conjuncts. The only explicit analysis of data like (2) in CG to date is Oehrle (1987). Oehrle's very insightful analysis, which alone among prior treatments of Gapping provides a starting point for an explanation for the apparent scope anomaly of such examples, unfortunately falls short of a general treatment of Gapping given the several non-standard assumptions about syntax that he crucially exploits in formulating his semantic analysis (see section 3).

In short, there is as yet no analysis of Gapping in CG that captures both the range of syntactic patterns and semantic interpretations associated with this construction. In the next section, we propose a version of Type-Logical Categorial Grammar utilizing a typed λ-calculus for notating the phonologies of linguistic signs. The novelty of the proposed system consists in recognizing both the directionality-sensitive modes of implication (/ and \) of the standard TLCG and the directionality-insensitive mode of implication tied to phonological λ-binding in more recent versions of CG (Oehrle 1994, de Groote 2001, Muskens 2003). In the framework we propose below, λ-binding in phonology provides a simple and explicit mechanism for representing constituents with missing objects in the medial position. This plays a key role in enabling an analysis of Gapping that overcomes the inadequacies of the previous approaches. It will be shown that once a proper analysis of the apparent asymmetry between the two conjuncts is formulated, the apparent anomalies related to scopal interactions with auxiliaries in examples like (2) above immediately disappear.

2 λTLCG and Gapping

We assume a version of Type-Logical Categorial Grammar (TLCG) in the labelled deduction format utilizing a typed λ-calculus for notating the phonologies of linguistic expressions, called λTLCG. We write linguistic expressions as tuples of phonological representation, semantic interpretation and syntactic category (written in that order). The full set of inference rules are given in (3).

(3) Connective Introduction Elimination

$/$

$$\cfrac{\cfrac{\begin{array}{ccc} \vdots\,\vdots & \underline{[\pi; x; A]^n} & \vdots\,\vdots \\ \vdots\,\vdots & \vdots\,\vdots & \vdots\,\vdots \end{array}}{b \circ \pi\,; \varphi; B}}{b\,; \lambda x.\varphi; B/A}\,/\mathrm{I}^n \qquad \cfrac{a\,; \varphi; A/B \quad b\,; \psi; B}{a \circ b\,; \varphi(\psi); A}\,/\mathrm{E}$$

$\backslash$

$$\cfrac{\cfrac{\begin{array}{ccc} \vdots\,\vdots & \underline{[\pi; x; A]^n} & \vdots\,\vdots \\ \vdots\,\vdots & \vdots\,\vdots & \vdots\,\vdots \end{array}}{\pi \circ b\,; \varphi; B}}{b; \lambda x.\varphi; A\backslash B}\,\backslash\mathrm{I}^n \qquad \cfrac{b\,; \psi; B \quad a\,; \varphi; B\backslash A}{b \circ a\,; \varphi(\psi); A}\,\backslash\mathrm{E}$$

$|$

$$\cfrac{\cfrac{\begin{array}{ccc} \vdots\,\vdots & \underline{[\pi; x; A]^n} & \vdots\,\vdots \\ \vdots\,\vdots & \vdots\,\vdots & \vdots\,\vdots \end{array}}{b;\ \varphi;\ B}}{\lambda\pi.b;\ \lambda x.\varphi;\ B|A}\,|\mathrm{I}^n \qquad \cfrac{a\,; \psi; A \quad b\,; \varphi; B|A}{b(a)\,; \varphi(\psi); B}\,|\mathrm{E}$$

The key difference between /,\ and | is that while the Introduction and Elimination rules for /,\ refer to the phonological forms of the input and output strings (so that, for example, the applicability of the /I rule is conditioned on the presence of the phonology of the hypothesis p on the right periphery of the phonology of the input $b \circ p$),[1] the rules for | are not constrained that way. For reasoning involving |, the phonological terms themselves fully specify the ways in which the output phonology is constructed from the input phonologies. Specifically, for

[1] In this respect, the present calculus follows most closely Morrill and Solias (1993) and Morrill (1994); see Moortgat (1997) and Bernardi (2002) for an alternative formulation where sensitivity to directionality is mediated through a presumed correspondence between surface string and the form of structured antecedents in the sequent-style notation of natural deduction.

|, the phonological operations associated with the Introduction and Elimination rules mirror exactly the semantic operations for these rules: function application and λ-abstraction, respectively. We assume that the binary connective $\circ$ in the phonological term calculus represents the string concatenation operation and that $\circ$ is associative in both directions. For notational convenience, we implicitly assume the axiom $(\pi_1 \circ \pi_2) \circ \pi_3 \equiv \pi_1 \circ (\pi_2 \circ \pi_3)$ and leave out all the brackets indicating the internal constituency of complex phonological terms.[2] Thus, the present system without the rules for | is equivalent to the Lambek calculus (Lambek 1958), while the system with only the rules for | is essentially equivalent to the term-labelled calculus of Oehrle (1994), λ-Grammar (Muskens 2003) and Abstract Categorial Grammar (de Groote 2001), with some implementational details aside, which are irrelevant for the following discussion.

λ-binding in the phonological component provides a simple and explicit way of modelling expressions with medial gaps. Oehrle (1994) originally showed this point by formalizing an explicit and straightforward implementation of Montague's (1973) quantifying-in; Muskens (2003) discusses how the same technique can be employed to solve the problem of medial extraction—a perennial problem in TLCG, where the directionality-sensitive modes of implication / and \ are inherently not suited for that purpose. The analysis of Gapping we propose below builds crucially on this analytic technique. Specifically, we treat Gapping to be a case of coordination of like-category constituents (with standard generalized conjunction for its semantics), where the coordinated constituents have medial gaps of the verbal category created via phonological (and semantic) variable binding. After the coordinate structure is built, the verb is 'lowered' into the medial position (just like the treatment of quantifiers by Oehrle (1994)), but the extra phonological property of the Gapping construction (which we encode in the Gapping-specific lexical entry for the conjunction) dictates that it be realized only once, in the initial conjunct of the whole coordinate structure.

In the present system, such constituents with medial verbal category gaps can be obtained by simply hypothesizing a variable for the main verb of the sentence and binding it by | after the whole sentence is built up:

(4)

$$\dfrac{\dfrac{\lambda\sigma_1.\sigma_1(\mathsf{a} \circ \mathsf{book});\ \exists_{\mathbf{book}};\ \mathrm{S}|(\mathrm{S}|\mathrm{NP}) \qquad \dfrac{\dfrac{\mathsf{robin};\ \mathbf{r};\ \mathrm{NP} \qquad \dfrac{[\pi_1; P; \mathrm{VP}/\mathrm{NP}]^1 \ [\pi_2; x; \mathrm{NP}]^2}{\pi_1 \circ \pi_2;\ P(x);\ \mathrm{VP}}}{\mathsf{robin} \circ \pi_1 \circ \pi_2;\ P(x)(\mathbf{r});\ \mathrm{S}}}{\lambda\pi_2.\mathsf{robin} \circ \pi_1 \circ \pi_2;\ \lambda x.P(x)(\mathbf{r});\ \mathrm{S}|\mathrm{NP}}}{\mathsf{robin} \circ \pi_1 \circ \mathsf{a} \circ \mathsf{book};\ \exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ \mathrm{S}}}{\lambda\pi_1.\mathsf{robin} \circ \pi_1 \circ \mathsf{a} \circ \mathsf{book};\ \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\ \mathrm{S}|(\mathrm{VP}/\mathrm{NP})}$$

[2] For a more fine-grained control of surface morpho-phonological constituency, see Kubota and Pollard (2010) (and also Muskens (2007) for a related approach), which formalizes the notion of multi-modality from the earlier TLCG literature (Moortgat and Oehrle 1994, Morrill 1994) by modelling the mapping from syntax to phonology by means of an interpretation of (phonological) λ-terms into preorders.

For coordinating such $st \to st$ functions (phonologically), we introduce the following Gapping-specific lexical entry for the conjunction:

(5) $\lambda\sigma_2\lambda\sigma_1\lambda\pi_0[\sigma_1(\pi_0) \circ \mathsf{and} \circ \sigma_2(\varepsilon)]; \lambda\mathscr{W}\lambda\mathscr{V}.\,\mathscr{V} \sqcap \mathscr{W};$
$((\mathrm{S}|(\mathrm{VP/NP}))|(\mathrm{S}|(\mathrm{VP/NP})))|(\mathrm{S}|(\mathrm{VP/NP}))$

where ε is the empty string and $\mathscr{V}$ and $\mathscr{W}$ are variables over terms of type $\langle\langle e, \langle e, t\rangle\rangle, t\rangle$. Note here that syntactically the conjunction takes two arguments of the same category S|(VP/NP), and returns an expression of the same category, and the semantics is nothing other than the standard generalized conjunction. This is fully consistent with the general treatment of coordination in CG in terms of like category coordination. The only slight complication is in the phonology. The output phonological term is of the same phonological type $st \to st$ as the input phonologies, but instead of binding the variables in each conjunct by the same λ-operator, the gap in the second conjunct is filled by an empty string ε, since the verb is pronounced only once in the first conjunct in Gapping. This is an idiosyncrasy of the construction that needs to be stipulated in any account, and in the present approach it is achieved by a lexical specification of the phonological interpretation of the conjunction, without invoking any extra rule, empty operator or null lexical item.

With this entry for the conjunction, the simple Gapping sentence (6) can be derived as in (7) (with TV an abbreviation for VP/NP).

(6) Leslie bought a CD, and Robin, a book.

(7)

$$\dfrac{\begin{array}{l}\mathsf{bought};\\ \mathbf{buy};\\ \mathrm{TV}\end{array} \quad \dfrac{\begin{array}{l}\lambda\pi_1.\mathsf{leslie} \circ \pi_1 \circ \mathsf{a} \circ \mathsf{CD};\\ \lambda Q.\exists_{\mathbf{CD}}(\lambda y.Q(y)(\mathbf{l}));\\ \mathrm{S|TV}\end{array} \quad \dfrac{\begin{array}{l}\lambda\sigma_2\lambda\sigma_1\lambda\pi_0.\sigma_1(\pi_0) \circ \mathsf{and} \circ \sigma_2(\varepsilon);\\ \lambda\mathscr{W}\lambda\mathscr{V}.\mathscr{V} \sqcap \mathscr{W};\\ \mathrm{(S|TV)|(S|TV)|(S|TV)}\end{array} \quad \begin{array}{l}\lambda\pi_1.\mathsf{robin} \circ \pi_1 \circ \mathsf{a} \circ \mathsf{book};\\ \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\\ \mathrm{S|TV}\end{array}}{\begin{array}{l}\lambda\sigma_1\lambda\pi_0.\sigma_1(\pi_0) \circ \mathsf{and} \circ \mathsf{robin} \circ \varepsilon \circ \mathsf{a} \circ \mathsf{book};\\ \lambda\mathscr{V}.\mathscr{V} \sqcap \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\\ \mathrm{(S|TV)|(S|TV)}\end{array}}}{\begin{array}{l}\lambda\pi_0[\mathsf{leslie} \circ \pi_0 \circ \mathsf{a} \circ \mathsf{CD} \circ \mathsf{and} \circ \mathsf{robin} \circ \varepsilon \circ \mathsf{a} \circ \mathsf{book}];\\ \lambda Q.\exists_{\mathbf{CD}}(\lambda y.Q(y)(\mathbf{l})) \sqcap \lambda P.\exists_{\mathbf{book}}(\lambda x.P(x)(\mathbf{r}));\\ \mathrm{S|TV}\end{array}}}{\begin{array}{l}\mathsf{leslie} \circ \mathsf{bought} \circ \mathsf{a} \circ \mathsf{CD} \circ \mathsf{and} \circ \mathsf{robin} \circ \varepsilon \circ \mathsf{a} \circ \mathsf{book};\\ \exists_{\mathbf{CD}}(\lambda y.\mathbf{buy}(y)(\mathbf{l})) \wedge \exists_{\mathbf{book}}(\lambda x.\mathbf{buy}(x)(\mathbf{r})); \mathrm{S}\end{array}}$$

What is crucial in the above analysis is that two conjoined gapped sentences form a tectogrammatical constituent, to which the verb lowers into. This enables a treatment of Gapping without any surface deletion operation or phonologically inaudible verbal pro-form of any sort. We will see below that this is also what enables a straightforward analysis of the scopal interactions between negative and modal auxiliaries and Gapping.

For the analysis of cases involving modal and negative auxiliaries, we assume an analysis of auxiliaries that treats them as quantifier-like scope-taking expressions. Morpho-phonologically, auxiliaries have a distributional property of a VP modifier of category VP/VP (which differs from VP adverbs VP\VP only in the direction in which the argument is sought). But semantically, modals and negation are sentential operators μ which take some proposition φ as an argument and return another proposition $\mu(\varphi)$. In the present approach, this syntax-semantics mismatch can be straightforwardly captured by assigning lexical entries of the following form to auxiliaries:

(8) $\lambda\sigma.\sigma(\mathsf{must});\ \lambda\mathscr{F}.\Box\mathscr{F}(\vartheta);\ \mathrm{S}|(\mathrm{S}|\mathrm{VP}/\mathrm{VP})$ (where $\vartheta =_{def} \lambda f_{\langle e,t\rangle}.f$)

That is, the auxiliary verb binds a VP/VP (i.e. forward-looking VP modifier) gap in a sentence to return a fully saturated S. The VP modifier gap is vacuously bound by supplying an identify function in its place, and the real semantic contribution of the auxiliary corresponds to the modal operator that takes as its scope the entire proposition obtained by binding that VP modifier gap.[3]

As an illustration, *Robin must discover a solution* is derived as:

(9)

$$\dfrac{\begin{array}{l}\lambda\sigma_0.\sigma_0(\mathsf{must});\\ \lambda\mathscr{F}.\Box\mathscr{F}(\vartheta);\\ \mathrm{S}|(\mathrm{S}|(\mathrm{VP}/\mathrm{VP}))\end{array}\quad \dfrac{\dfrac{\begin{array}{l}\lambda\sigma.\sigma(\mathsf{a}\circ\mathsf{solution});\\ \exists_{\mathbf{solution}};\mathrm{S}|(\mathrm{S}|\mathrm{NP})\end{array}\quad \dfrac{\dfrac{\begin{array}{l}\mathsf{robin};\\ \mathbf{r};\mathrm{NP}\end{array}\quad \dfrac{\begin{array}{l}\pi_2;\\ f;\\ \mathrm{VP}/\mathrm{VP}\end{array}\quad \dfrac{\begin{array}{l}\mathsf{discover};\\ \mathbf{discover};\\ \mathrm{VP}/\mathrm{NP}\end{array}\quad \begin{array}{l}\pi_1;\\ x;\\ \mathrm{NP}\end{array}}{\begin{array}{l}\mathsf{discover}\circ\pi_1;\\ \mathbf{discover}(x);\\ \mathrm{VP}\end{array}}}{\begin{array}{c}\pi_2\circ\mathsf{discover}\circ\pi_1;\\ f(\mathbf{discover}(x));\mathrm{VP}\end{array}}}{\begin{array}{c}\mathsf{robin}\circ\pi_2\circ\mathsf{discover}\circ\pi_1;\\ f(\mathbf{discover}(x))(\mathbf{r});\mathrm{S}\end{array}}}{\begin{array}{c}\lambda\pi_1.\mathsf{robin}\circ\pi_2\circ\mathsf{discover}\circ\pi_1;\\ \lambda x.f(\mathbf{discover}(x))(\mathbf{r});\mathrm{S}|\mathrm{NP}\end{array}}}{\begin{array}{c}\mathsf{robin}\circ\pi_2\circ\mathsf{discover}\circ\mathsf{a}\circ\mathsf{solution};\\ \exists_{\mathbf{solution}}(\lambda x.f(\mathbf{discover}(x))(\mathbf{r}));\mathrm{S}\end{array}}}{\begin{array}{c}\lambda\pi_2.\mathsf{robin}\circ\pi_2\circ\mathsf{discover}\circ\mathsf{a}\circ\mathsf{solution};\\ \lambda f.\exists_{\mathbf{solution}}(\lambda x.f(\mathbf{discover}(x))(\mathbf{r}));\mathrm{S}|(\mathrm{VP}/\mathrm{VP})\end{array}}}{\begin{array}{l}\mathsf{robin}\circ\mathsf{must}\circ\mathsf{discover}\circ\mathsf{a}\circ\mathsf{solution};\\ \Box\exists_{\mathbf{solution}}(\lambda x.\mathbf{discover}(x)(\mathbf{r}));\mathrm{S}\end{array}}$$

We are now ready to illustrate how the auxiliary wide-scope readings are obtained for Gapping sentences. We start with a variant in (10a) in which only the auxiliary is gapped (for which the derivation is a bit simpler), and then move on to the case of (10b) where the whole auxiliary + verb combination is gapped.

(10) a. John can't eat steak and Mary eat pizza.

b. John can't eat steak and Mary pizza.

[3] As it is, the analysis of auxiliaries here overgenerates. To capture the clause-boundedness of the scope of auxiliaries, we can employ the technique proposed by Pogodalla and Pompigne (2011) which enables formulating constraints on scope islands at the tectogrammatical level with the notion of dependent types (dependent types are used in certain logic-based grammars for the purpose of implementing syntactic features; see de Groote and Maarek (2007)).

As in the basic-case analysis given in (7) above, the overall strategy is straightforward: we coordinate two categories which are in effect clauses missing VP/VP functors in each conjunct, forming a larger sign of the same category:

(11)

$$\dfrac{\dfrac{\dfrac{\begin{array}{l}\mathsf{john};\\ \mathbf{j};\\ \mathrm{NP}\end{array}\quad \dfrac{\begin{array}{l}\pi_1;\\ f;\\ \mathrm{VP/VP}\end{array}\quad\begin{array}{l}\mathsf{eat}\circ\mathsf{steak};\\ \mathbf{eat}(\mathbf{s});\\ \mathrm{VP}\end{array}}{\begin{array}{l}\pi_1\circ\mathsf{eat}\circ\mathsf{steak};\\ f(\mathbf{eat}(\mathbf{s}));\\ \mathrm{VP}\end{array}}}{\begin{array}{l}\mathsf{john}\circ\pi_1\circ\mathsf{eat}\circ\mathsf{steak};\\ f(\mathbf{eat}(\mathbf{s}))(\mathbf{j});\mathrm{S}\end{array}}}{\begin{array}{l}\lambda\pi_1.\mathsf{john}\circ\pi_1\circ\mathsf{eat}\circ\mathsf{steak};\\ \lambda f.f(\mathbf{eat}(\mathbf{s}))(\mathbf{j});\mathrm{S}|(\mathrm{VP/VP})\end{array}}\quad\dfrac{\begin{array}{l}\lambda\sigma_2\lambda\sigma_1\lambda\pi_0.\sigma_1(\pi_0)\circ\mathsf{and}\circ\sigma_2(\varepsilon);\\ \lambda\mathscr{F}_2\lambda\mathscr{F}_1.\mathscr{F}_1\sqcap\mathscr{F}_2;\\ (\mathrm{S}|\mathrm{X})|(\mathrm{S}|\mathrm{X})|(\mathrm{S}|\mathrm{X})\end{array}\quad\begin{array}{l}\lambda\pi_2.\mathsf{mary}\circ\pi_2\circ\mathsf{eat}\circ\mathsf{pizza};\\ \lambda g.g(\mathbf{eat}(\mathbf{p}))(\mathbf{m});\\ \mathrm{S}|(\mathrm{VP/VP})\end{array}}{\begin{array}{l}\lambda\sigma_1\lambda\pi_0.\sigma_1(\pi_0)\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{eat}\circ\mathsf{pizza};\\ \lambda\mathscr{F}_1.\mathscr{F}_1\sqcap\lambda g.g(\mathbf{eat}(\mathbf{p}))(\mathbf{m});(\mathrm{S}|(\mathrm{VP/VP}))|(\mathrm{S}|(\mathrm{VP/VP}))\end{array}}}{\begin{array}{l}\lambda\pi_0.\mathsf{john}\circ\pi_0\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{eat}\circ\mathsf{pizza};\\ \lambda f.f(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\sqcap\lambda g.g(\mathbf{eat}(\mathbf{p}))(\mathbf{m});\mathrm{S}|(\mathrm{VP/VP})\end{array}}$$

This coordinated 'gapped' constituent is then given as an argument to the auxiliary to complete the derivation, just as in the simpler example in (9) above.

(12)

$$\dfrac{\begin{array}{l}\lambda\sigma_0.\sigma_0(\mathsf{can't});\\ \lambda\mathscr{F}.\neg\Diamond\mathscr{F}(\vartheta);\\ \mathrm{S}|(\mathrm{S}|(\mathrm{VP/VP}))\end{array}\quad\begin{array}{l}\lambda\pi_0.\mathsf{john}\circ\pi_0\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{eat}\circ\mathsf{pizza};\\ \lambda f.f(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\sqcap\lambda g.g(\mathbf{eat}(\mathbf{p}))(\mathbf{m});\\ \mathrm{S}|(\mathrm{VP/VP})\end{array}}{\begin{array}{l}\mathsf{john}\circ\mathsf{can't}\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{eat}\circ\mathsf{pizza};\\ \neg\Diamond[\mathbf{eat}(\mathbf{s})(\mathbf{j})\wedge\mathbf{eat}(\mathbf{p})(\mathbf{m})];\mathrm{S}\end{array}}$$

Here, crucially, due to generalized conjunction, the proposition that the modal scopes over is the conjunction of the propositions expressed by the first conjunct (without the modal) and the second conjunct. Thus, we get an interpretation in which the modal scopes over the conjunction, as desired. For the phonology, just as in the simpler Gapping example, due to the lexical definition of the Gapping-type conjunction, the modal auxiliary is pronounced only in the first conjunct, resulting in the surface string corresponding to (10a).[4]

We now show how this same approach yields a wide scope reading for the auxiliary where both the auxiliary and the main verb are missing in the second conjunct, as in (10b). The derivation goes as in (13). The extra complexity involved in this case is that we need to fill in both the verb and the auxiliary in the first conjunct to obtain the surface form of the sentence. This is done in a stepwise manner. First, the verb and a hypothesized forward-looking VP modifier (to be bound by the auxiliary) form an expression of the VP/NP category via

[4] See Siegel (1987) for a closely related approach in terms of wrapping in the framework of Montague Grammar. For the auxiliary-gapping example like (10a), our analysis can be thought of as a formally precise rendition of the basic analytic idea prefigured in Siegel's analysis. However, the presence vs. absence of an explicit prosodic calculus that interacts with the combinatoric component of syntax becomes crucial in the more complex case in (10b), where both the verb and the auxiliary are gapped. It is not at all clear how the right pairing of meaning and surface string can be derived for such examples in Siegel's setup, which assumes a rather primitive and unformalized infixation operation within Montague Grammar for dealing with discontinuous constituency.

hypothetical reasoning. This is then given as an argument to a coordinated gapped sentence of type S|(VP/NP). Finally, by binding the VP modifier of type VP/VP, the sentence has the right syntactic (and phonological and semantic, as well) type to be given as an argument to the auxiliary *can't*.

(13)

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{l}\pi_0;\\ f;\mathrm{VP/VP}\end{array}\quad \dfrac{\begin{array}{l}\mathsf{eat};\\ \mathbf{eat};\mathrm{VP/NP}\end{array}\quad\begin{array}{l}\pi_1;\\ x;\mathrm{NP}\end{array}}{\mathsf{eat}\circ\pi_1;\ \mathbf{eat}(x);\ \mathrm{VP}}}{\pi_0\circ\mathsf{eat}\circ\pi_1;\ f(\mathbf{eat}(x));\ \mathrm{VP}}}{\pi_0\circ\mathsf{eat};\ \lambda x.f(\mathbf{eat}(x));\ \mathrm{VP/NP}}\quad \begin{array}{l}\lambda\pi_2.\mathsf{john}\circ\pi_2\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{pizza};\\ \lambda Q.[Q(\mathbf{s})(\mathbf{j})]\sqcap\lambda P.[P(\mathbf{p})(\mathbf{m})];\\ \mathrm{S|(VP/NP)}\end{array}}{\mathsf{john}\circ\pi_0\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{pizza};\ f(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\wedge f(\mathbf{eat}(\mathbf{p}))(\mathbf{m});\ \mathrm{S}}}{\begin{array}{c}\lambda\pi_0.\mathsf{john}\circ\pi_0\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{pizza};\\ \lambda f.[f(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\wedge f(\mathbf{eat}(\mathbf{p}))(\mathbf{m})];\mathrm{S|(VP/VP)}\end{array}}$$

$$\dfrac{\begin{array}{l}\lambda\sigma_0.\sigma_0(\mathsf{can't});\\ \lambda\mathscr{F}.\neg\Diamond\mathscr{F}(\vartheta);\\ \mathrm{S|(S|(VP/VP))}\end{array}\quad\begin{array}{l}\lambda\pi_0.\mathsf{john}\circ\pi_0\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{pizza};\\ \lambda f.[f(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\wedge f(\mathbf{eat}(\mathbf{p}))(\mathbf{m})];\\ \mathrm{S|(VP/VP)}\end{array}}{\begin{array}{l}\mathsf{john}\circ\mathsf{can't}\circ\mathsf{eat}\circ\mathsf{steak}\circ\mathsf{and}\circ\mathsf{mary}\circ\varepsilon\circ\mathsf{pizza};\\ \neg\Diamond[\mathbf{eat}(\mathbf{s})(\mathbf{j})\wedge\mathbf{eat}(\mathbf{p})(\mathbf{m})];\mathrm{S}\end{array}}$$

Again, since the auxiliary takes the coordinated sentence (after the verb is fed to it) as its argument in the derivation, we obtain the auxiliary wide-scope interpretation. In the present account, the wide-scope option for the auxiliary in examples like (10a,b) transparently reflects the (tectogrammatical) syntax of Gapping where sentences with missing elements are *directly* coordinated and the missing element is supplied at a later point in the derivation. Thus, the availability of such a reading is not a surprise, but a naturally expected consequence.

The present analysis predicts the availability of conjunction wide-scope readings for sentences like those in (10) as well. The key component of the analysis involves deriving a VP/VP entry for an auxiliary from the more basic type assigned in the lexicon above in the category S|(S|(VP/VP)), which reflects their semantic property more transparently. The derivation proceeds through a couple of steps of hypothetical reasoning:

(14)

$$\dfrac{\dfrac{\dfrac{\begin{array}{l}\lambda\sigma.\sigma(\mathsf{can't});\\ \lambda\mathscr{F}.\neg\Diamond\mathscr{F}(\vartheta);\\ \mathrm{S|(S|VP/VP)}\end{array}\quad\dfrac{\dfrac{\pi_1;\ x;\ \mathrm{NP}\quad\dfrac{\pi_2;\ g;\ \mathrm{VP/VP}\quad \pi_3;\ f;\ \mathrm{VP}}{\pi_2\circ\pi_3;\ g(f);\ \mathrm{VP}}}{\pi_1\circ\pi_2\circ\pi_3;\ g(f)(x);\ \mathrm{S}}}{\begin{array}{l}\lambda\pi_2.\pi_1\circ\pi_2\circ\pi_3;\\ \lambda g.g(f)(x);\\ \mathrm{S|(VP/VP)}\end{array}}}{\pi_1\circ\mathsf{can't}\circ\pi_3;\ \neg\Diamond f(x);\ \mathrm{S}}}{\mathsf{can't}\circ\pi_3;\ \lambda x.\neg\Diamond f(x);\ \mathrm{VP}}}{\mathsf{can't};\ \lambda f\lambda x.\neg\Diamond f(x);\ \mathrm{VP/VP}}$$

The derived entry in the VP/VP category is the familiar entry for auxiliaries in non-transformational approaches like G/HPSG and categorial grammar. The above result depends crucially on the property of the present system where reasoning involving the directional mode of implication can be carried out based on the results of reasoning involving |, which allows for operations that are more complex than string concatenation.[5]

With the above, derived, type assignment for the auxiliary, the conjunction wide-scope reading for (10a) is straightforward. The derivation is identical to the one for the auxiliary wide-scope reading up to the point that the coordinated gapped sentence is formed, and differs only at the final step. Instead of having the scope-taking S|(S|(VP/VP)) entry of the auxiliary take this coordinated gapped S as an argument, we simply give the lowered VP/VP entry for the auxiliary as an argument to the gapped sentence, as follows:

(15)
$$\dfrac{\begin{array}{ll}\textsf{can}'\textsf{t}; & \lambda\pi.[\textsf{john}\circ\pi\ \ \circ\textsf{eat}\circ\textsf{steak}\circ\textsf{and}\circ\textsf{mary}\circ\varepsilon\ \ \circ\textsf{eat}\circ\textsf{pizza}];\\ \lambda f\lambda x.\neg\Diamond f(x); & \lambda h.[h(\mathbf{eat}(\mathbf{s}))(\mathbf{j})\wedge h(\mathbf{eat}(\mathbf{p}))(\mathbf{m})];\\ \mathrm{VP/VP} & \mathrm{S}|(\mathrm{VP/VP})\end{array}}{\begin{array}{l}\textsf{john}\circ\textsf{can}'\textsf{t}\circ\textsf{eat}\circ\textsf{steak}\circ\textsf{and}\circ\textsf{mary}\circ\varepsilon\ \ \circ\textsf{eat}\circ\textsf{pizza};\\ \neg\Diamond\mathbf{eat}(\mathbf{s})(\mathbf{j})\wedge\neg\Diamond\mathbf{eat}(\mathbf{p})(\mathbf{m});\mathrm{S}\end{array}}$$

The resulting string is identical as above since the phonology of the auxiliary is embedded in the gap site in the initial conjunct only, but the semantic interpretation that is paired with it is different from the above analysis. Here crucially, the VP-modifier meaning of the auxiliary is distributed to the two conjuncts via the definition of generalized conjunction, which results in an interpretation where the auxiliary takes scope separately within each conjunct which is then conjoined by the conjunction, resulting in the conjunction wide-scope reading.

The derivation for the conjunction wide-scope reading for (10b), a sentence in which both the auxiliary and the main verb are missing, is also straightforward. In fact, like the previous case, the analysis just involves replacing the auxiliary entry at the final step of the derivation for the auxiliary wide-scope reading of the same sentence (given above in (13)) with the derived entry in (14). This yields the conjunction wide-scope reading for the sentence for exactly the same reason as in the previous example, as the reader can easily verify by themselves.

3 Comparison with Related Approaches

3.1 Steedman (1990)

Steedman (1990) proposes an insightful analysis of Gapping in CCG which can be thought of as a precursor of the present proposal. The key analytic idea of Steedman's approach is the assumption that Gapping involves coordination of like-category constituents. To reconcile the strictly surface-oriented syntax of CCG with this assumption about the 'underling' syntax of Gapping, Steedman invokes a syntactic rule called the Left Conjunct Revealing Rule (LCRR):[6]

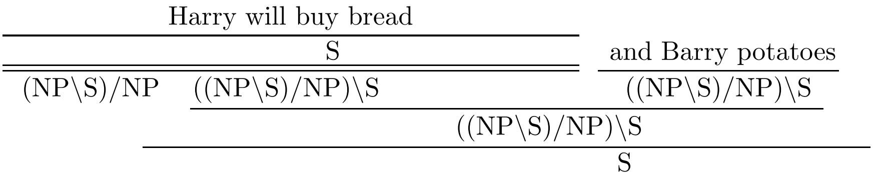
[5] So far as we are aware, interactions between the two kinds of syntactic reasoning (or modes of composition) of this sort is a completely novel property of the present system that is not shared by any other formal theory of syntax, whether categorial, constraint-based, or transformational. This certainly opens up many questions conceptually, technically and empirically, which we will not pursue further here.

[6] We use the Lambek-style notation for slashes for consistency.

(16) **The Left Conjunct Revealing Rule**
$\mathrm{S} \Rightarrow Y\ Y\backslash\mathrm{S}$

This rule 'decomposes' the left conjunct into two syntactic categories, one corresponding to the right conjunct and the other corresponding to the shared verbal element. Once this decomposition is in place, the rest of the derivation is straightforward with standard generalized conjunction to form a coordinate structure which recombines with the shared verbal category as in:

(17)



Simple and elegant though it might appear, this analysis is problematic for both conceptual/technical and empirical reasons. Conceptually and technically, note that (16) constitutes a clearcut violation of the principle of compositionality. The key problem is that there is no way, on Steedman's account, to guarantee that either the Y or the $Y\backslash\mathrm{S}$ can be independently assembled from the component of S on the basis of its subcomponents that they are supposed to correspond to, and, concomitantly, no way to ensure the existence of an actual semantic interpretation for the 'revealed' $Y\backslash\mathrm{S}$ pseudocategory or its complement Y.[7] Steedman's account thus seems to ride roughshod over the fundamental motivation of the categorial approach, viz., the conception of syntactic derivations as logical proofs (whose structure is not an object that the grammar can manipulate).[8]

There is also an empirical problem. In the strictly surface-oriented syntax of CCG, there does not seem to be any straightforward account of the auxiliary wide-scope readings of sentences with modal and negation. Like in other non-transformational approaches to syntax, CCG assumes the VP/VP-type entry for auxiliaries. However, as shown in the previous section, such an entry produces only the conjunction wide-scope readings for examples like those in (2).

[7] So far as we can see, the only way to ensure such a decomposition in the grammar is by 'appealing to the parser, or to some reification of the derivation' (Steedman 1990, 247), a possibility which, curiously enough, Steedman rejects flatly. He instead resorts to some vague (and what seems to us to be an ill-conceived) pragmatic strategy of recovering the *syntactic category* (as opposed to just the interpretation) of the gapped verb through the presupposition of the gapped sentence.

[8] Note in this connection that the formal status of the LCRR is quite unclear. In a labelled deduction presentation of derivations of the kind we have adopted above, there is no way to formulate such a rule, since in (16) the pieces of linguistic expression that the decomposed categories are supposed to correspond to are entirely unspecified. This alone makes Steedman's whole approach to Gapping quite dubious.

3.2 Morrill et al. (2011)

In the TLCG literature, a series of related approaches to the analysis of Gapping have been proposed (see the references cited in section 1) that provide an explicit solution for the problem of identifying the gap constituent that is left open in Steedman's analysis. These approaches all treat Gapping as a case of discontinuous constituency, employing the various extensions to the Lambek calculus for handling discontinuity that they respectively propose. We review Morrill et al.'s proposal here since it is the most recent among these related approaches and it improves both technically and empirically on the earlier accounts.

The key analytic idea of Morrill et al.'s approach, which is due to Hendriks (1995), and which is a formalization of the underlying idea of Steedman's approach, is that Gapping can be thought of as a case of like-category coordination by allowing the conjunction to coordinate two discontinuous constituents with medial gaps of the verbal type and then infixing the missing verb in the gap position of the initial conjunct after the whole coordinate structure is built. Specifically, Morrill et al. assign the syntactic type $((\mathrm{S}{\uparrow}\mathrm{TV})\backslash(\mathrm{S}{\uparrow}\mathrm{TV}))/\hat{\ }(\mathrm{S}{\uparrow}\mathrm{TV})$ to the conjunction. $\uparrow$ is roughly equivalent to our $|$. Thus, $\mathrm{S}{\uparrow}\mathrm{TV}$ is the category for a sentence missing a TV somewhere inside it. $\hat{\ }$ corresponds to an operation that erases the 'insertion point' keeping track of the gap position of a discontinuous constituent. Thus, the category of the right conjunct $\hat{\ }(\mathrm{S}{\uparrow}\mathrm{TV})$ indicates that it is a sentence with a TV gap inside it like $\mathrm{S}{\uparrow}\mathrm{TV}$, except that the gap is already 'closed off'. The whole coordinate structure inherits the gap position from the left conjunct alone, to which the verb is infixed after the whole coordinate structure is built.

If we limit ourselves to cases in which the gapped material is just a string, our analysis and Morrill et al.'s can be thought of as notational variants of each other.[9] However, a difference between the two emerges when we examine more complex examples where the missing material in the gapped clause is itself a discontinuous constituent, such as the following:

(18) a. John *gave* Mary *a cold shoulder*, and Bill, Sue.
b. John *called* Mary *up*, and Bill, Sue.

Idiomatic expressions like *give ... the cold shoulder* and verb-particle constructions like *call ... up* are analyzed as discontinuous constituents in CG, including Morrill et al.'s own approach. However, their analysis of Gapping does not interact properly with their analysis of these constructions to license examples like those in (18). The difficulty essentially lies in the fact that Morrill et al.'s system is set up in such a way that it only recognizes discontinuous constituents with string-type gaps (whose positions are kept track of by designated symbols

[9] But note that it remains to be established that the analysis of the scope ambiguity of auxiliaries in examples like (10) in our account can be replicated in their setup—so far as we can see, the analysis of the auxiliary-wide scope reading seems to carry over to their setup straightforwardly, whereas the case of the conjunction-wide scope reading is less clear.

called separators for marking insertion points in the prosodic representations of linguistic expressions) and the only operations that one can perform on such expressions (each tied to different syntactic rules in the calculus) are to close off the gap with some string (including an empty string) or to pass it up to a larger expression. Thus, there is no direct way of representing the combinatorial property of the gapped clause *Bill, Sue* in (18a), which needs to be treated as a sentence missing a VP↑NP in order to induce a like-category analysis of Gapping along the above lines. It should be noted that this problem is by no means specific to Morrill et al.'s account but is common to all previous approaches in the TLCG literature that employ some form of wrapping operation for the treatment of discontinuity. Such approaches fall short of extending to cases like (18) essentially because the prosodic component is not fully independent of the syntactic calculus and the extension to the basic concatenative system is directly regulated by the set of additional syntactic connectives introduced in the system. (To see this, note that in these approaches, each syntactic connective for discontinuity is tied to some specific operation (such as infixation) on the propsodic form(s) of the input expressions).

Our proposal differs from these earlier approaches precisely in this respect. Indeed, with the flexible syntax-prosody interface enabled by having a full-blown λ-calculus for the prosodic component—a feature that the present system inherits from λ-Grammar/ACG—the analysis of examples like (18) turns out to be relatively straightforward. Assuming that the discontinuous constituency of idioms and verb-particle constructions is treated by assigning to the relevant expressions lexical entries of the following form (of phonological type $st \to st$):

(19) $\lambda\pi_1.\mathsf{gave} \circ \pi_1 \circ \mathsf{the} \circ \mathsf{cold} \circ \mathsf{shoulder}$; **shun**; VP|NP

it only suffices to generalize the lexical entry of the Gapping-type conjunction to a higher-order (phonological) type which takes arguments of type $(st \to st) \to st$ (phonologically) as left and right conjuncts:

(20) $\lambda\rho_1\lambda\rho_2\lambda\sigma.\rho_2(\sigma) \circ \mathsf{and} \circ \rho_1(\lambda\pi.\pi)$; $\lambda\mathscr{W}\lambda\mathscr{V}.\mathscr{V} \sqcap \mathscr{W}$; (S|(VP|X))|(S|(VP|X))|(S|(VP|X))

Then, via hypothetical reasoning with a variable of (phonological) type $st \to st$, the left and right conjuncts can be treated as discontinuous constituents of type $(st \to st) \to st$ of the following form, where the gap itself is a discontinuous constituent of type $st \to st$:

(21) $\lambda\sigma_1.\mathsf{john} \circ \sigma_1(\mathsf{mary})$; $\lambda P.P(\mathbf{m})(\mathbf{j})$; S|(VP|NP)

It is straightforward to see that by giving such expressions to the higher-order Gapping conjunction entry (20), the right surface string in (18) is obtained.

Thus, while the proposed analysis owes much to previous approaches to Gapping in terms of discontinuous constituency in the TLCG literature in the formulation of the basic analysis, it goes beyond all previous proposals in straightforwardly generalizing to more complex cases like (18) where Gapping interacts with other phenomena exemplifying discontinuity. So far as we are aware, such a systematic interaction of complex empirical phenomena is unprecedented in any previous work.

3.3 Oehrle (1987)

Oehrle's analysis assumes a free Boolean algebra over the set of generators comprising the cartesian product NP × NP, with meet $\wedge$ and join $\vee$, which Oehrle notates $\mathbf{L}$[NP×NP]. The set NP × NP is the domain of functors corresponding to verb signs, which are taken to comprise both phonological and semantic functors. An embedding from NP × NP to an algebra with meet and join operations yields $\mathbf{L}$[NP× NP], which is the closure of its atoms in NP × NP under $\vee$ and $\wedge$.

For each verbal sign $\boldsymbol{v}$, which is a function NP×NP $\mapsto$ 2, Oehrle defines an extension of that function $\boldsymbol{v}^*$ $\mathbf{L}$[NP×NP] $\mapsto$ 2. For example, for *bakes*, we have $\boldsymbol{bake}$ NP×NP $\mapsto$ 2, comprising the phonological function $\boldsymbol{bake}_\pi$ and the semantic function $\boldsymbol{bake}_\sigma$ such that (here again, π_1 and π_2 are projection functions):

(22) a. $\boldsymbol{bake}_\pi = \lambda P.\ \pi_1(P) \circ \mathsf{bakes} \circ \pi_2(P)$
 b. $\boldsymbol{bake}_\sigma = \lambda X.\mathbf{bake}(\pi_1(X), \pi_2(X))$

$\boldsymbol{bake}^*$ is then defined as consisting of the phonological function $\boldsymbol{bake}^*_\pi$ and the semantic function $\boldsymbol{bake}^*_\sigma$ such that:

(23) i. For all P that are atomic, $\boldsymbol{bake}^*_\pi(P) = \boldsymbol{bake}_\pi(P)$
 ii. If $P = P_1 \wedge P_2$, then $\boldsymbol{bake}^*_\pi(P) = \boldsymbol{bake}_\pi(P_1) \circ \mathsf{and} \circ \pi_1(P_2) \circ \pi_2(P_2)$
 iii. If $P = P_1 \vee P_2$, then $\boldsymbol{bake}^*_\pi(P) = \boldsymbol{bake}_\pi(P_1) \circ \mathsf{or} \circ \pi_1(P_2) \circ \pi_2(P_2)$

(24) i. For all X that are atomic, $\boldsymbol{bake}^*_\sigma(X) = \boldsymbol{bake}_\sigma(X)$
 ii. If $X = X_1 \wedge X_2$, then $\boldsymbol{bake}^*_\sigma(X) = \boldsymbol{bake}_\sigma(X_1) \wedge \boldsymbol{bake}_\sigma(X_2)$
 iii. If $X = X_1 \vee X_2$, then $\boldsymbol{bake}^*_\sigma(X) = \boldsymbol{bake}_\sigma(X_1) \vee \boldsymbol{bake}_\sigma(X_2)$

The grammar thus admits (25), comprising the phonology and semantics in (26).

(25) $\boldsymbol{bake}^*(\langle \boldsymbol{john}, \boldsymbol{bread}\rangle \wedge \langle \boldsymbol{mary}, \boldsymbol{cake}\rangle)$
 $= \langle \boldsymbol{bake}^*_\pi(\langle \mathsf{john}, \mathsf{bread}\rangle \wedge \langle \mathsf{mary}, \mathsf{cake}\rangle), \boldsymbol{bake}^*_\sigma(\langle \mathbf{j}, \mathbf{b}\rangle \wedge \langle \mathbf{m}, \mathbf{c}\rangle)\rangle$

(26) a. $\boldsymbol{bake}^*_\pi(\langle \mathsf{john}, \mathsf{bread}\rangle \wedge \langle \mathsf{mary}, \mathsf{cake}\rangle)$
 $= \boldsymbol{bake}_\pi(\langle \mathsf{john}, \mathsf{bread}\rangle) \circ \mathsf{and} \circ \mathsf{mary} \circ \mathsf{cake}$
 $= \mathsf{john} \circ \mathsf{bakes} \circ \mathsf{bread} \circ \mathsf{and} \circ \mathsf{mary} \circ \mathsf{cake}$
 b. $\boldsymbol{bake}^*_\sigma(\langle \mathbf{j}, \mathbf{b}\rangle \wedge \langle \mathbf{m}, \mathbf{c}\rangle)$
 $= \boldsymbol{bake}_\sigma(\langle \mathbf{j}, \mathbf{b}\rangle) \wedge \boldsymbol{bake}_\sigma(\langle \mathbf{m}, \mathbf{c}\rangle) = \mathbf{bake}(\mathbf{j},\mathbf{b}) \wedge \mathbf{bake}(\mathbf{m},\mathbf{c})$

The key insight that enables Oehrle to account for the scope ambiguity of negative and modal auxiliaries is that when propositional operators like negation interact with verb meaning, there are two maps from $\mathbf{L}$[NP×NP] to $\mathbf{2}$, with two different semantic results. The first option is to compose the negation operator **neg** with the lexical verb $\boldsymbol{v}$ and then extend it with the * operator to obtain $(\mathbf{neg} \circ \boldsymbol{v})^*$, which produces a function that takes arguments in the domain of conjoined pairs of verbal arguments. This yields the conjunction wide-scope interpretation since * extends verb meanings that are already negated and which take unconjoined pairs of arguments to the domain of conjoined pairs of arguments. The other option is to compose the negation operator **neg** with the result of the application of *, which gives us $\mathbf{neg} \circ \boldsymbol{v}^*$. Since $\boldsymbol{v}^*$ is the closure of $\boldsymbol{v}$ under meet and join which semantically correspond to conjunction and disjunction, this yields a function that takes conjoined pairs of arguments as input, and then first form unnegated conjunction or disjunction of two propositions obtained by applying the verb meaning to the each of the conjoined pair of arguments, which is then passed on to the negation operator as an argument to produce a negated proposition, which corresponds to the negation wide-scope interpretation.

Thus, in Oehrle's analysis, the assumption that argument pairs of verbs can be treated as conjoinable constituents and that the * operator that maps verb meanings from NP×NP to $\mathbf{L}$[NP×NP] which contains such conjoined argument pairs plays a crucial role in deriving the scopal interactions between conjunction and operators such as negation and modals. While the elegance and systematicity by which the auxiliary wide-scope readings are derived is remarkable, Oehrle's analysis relies on several nonstandard assumptions about both the basic clause structure of English and the syntax of Gapping. In particular, since the analysis crucially hinges on the assumption that the remnants that appear in the right conjunct are pairs of *arguments* of a verb, it is not clear how the analysis might be extended to cases involving adjuncts in the remnant, such as (1c) from section 1. Since adverbs are adjuncts which are functions that take verbs as arguments rather than themselves being arguments of the verb, it is not clear how examples like (1c) can be licensed in Oehrle's setup. Note furthermore that such argument-adjunct pairs in Gapping can also induce the same kind of scopal interaction with auxiliaries as the argument-pair examples examined above:

(27) Terry can't go there with me and Pat with you—one and the same person has to accompany them both.

This suggests that generating surface strings like (1c) isn't enough and that the mechanism for licensing the two scoping possibilities for argument pairs has to be extended to cases involving adjuncts too. However, given the nonstandard assumptions about syntax that Oehrle's analysis builds on, it is not clear whether such an extension can be worked out straightforwardly.

4 Conclusion

We have proposed a system of TLCG that models phonologies of linguistic signs by λ-terms, allowing for higher-order abstraction over string-type entities. The flexible treatment of linguistic expressions manifesting discontinuous constituency that the present system allows for enables a straightforward treatment of Gapping which subsumes this construction—despite its appearance—under the law of coordination of likes. Furthermore, this analysis provides an immediate solution for a seemingly separate puzzle of apparently anomalous wide-scoping auxiliaries in Gapping, for which no explicit analysis exists except for Oehrle (1987) (which itself suffers from a different kind of problem).

The proposed calculus is unique among contemporary alternatives of CG-based syntactic frameworks in that it recognizes *both* directionality-sensitive modes of implication traditionally assumed in TLCG and the directionality-insensitive mode of implication from the more recent variants of CG such as λ-Grammar and ACG that deal with word order by enriching operations available in the (morpho-)phonological component (in particular, by having a full-fledged λ-calculus for it). This novel architecture of the present theory raises two related larger questions. First, one might wonder whether the relatively elaborate theoretical setup of the present system is justified. Second, the formal properties of the proposed system is as yet unexplored.

For the first, more empirical question, note that what enables subsuming Gapping under the case of like-categorial coordination in the present approach is its ability to analyze any *substring* containing a verb within a sentence as a constituent that can be abstracted over. This requires an interaction of the directional and non-directional slashes precisely of the kind that the present approach provides. In a system with only one mode of implication in the syntactic component (corresponding to our $|$), significant complications will arise, since in such an approach, verb phonologies in the lexicon are not simply strings but rather are n-place functions over strings (e.g., for transitive verbs, of the form $\lambda\pi_1\lambda\pi_2.\pi_2 \circ \mathsf{bought} \circ \pi_1$, of type $st \to st \to st$) that specify the relative positions of their arguments purely in the phonological representation. Abstracting over such a sign creates a higher-order phonological entity. To simulate the results of our analysis of Gapping in such a framework, one would then need to define a polymorphic entry for *and* which would yield the correct surface string for the right conjunct from such higher-order functions for each case in which a different type of functional phonology is abstracted over. But defining the appropriate entry for the conjunction word that would extend to cases involving auxiliaries—which have still more complex phonological types—is a non-trivial task, to put it mildly. It thus seems reasonable to conclude that, however one implements it, the kind of interaction between (tectogrammatical) syntax and surface linearization that the present system enables (via the interactions between $/,\backslash$ and $|$) needs to be part of the formal calculus for dealing with natural language syntax.

And this brings up the second question: if such a mixed system is empirically motivated, what are its exact formal underpinnings? Although previous proposals exist that propose calculi that recognize both directional and non-directional modes of implication within a single system (cf. de Groote (1996), Polakow and Pfenning (1999)), our system differs from these formal systems in that it allows for the two kinds of reasoning to freely feed into one another. In fact, this is precisely the source of the flexibility exploited in our analysis of Gapping, and, so far as we are aware, such a system is unprecedented and its mathematical properties are unknown. Given the linguistic motivation that we have demonstrated in this paper, the mathematical properties of the proposed system should be studied closely. We acknowledge this as an important issue to be investigated in future work.

References

[Bernardi (2002)]Bernardi, R.: Reasoning with Polarity in Categorial Type Logic. Ph.D. thesis, University of Utrecht (2002)

[de Groote (1996)]de Groote, P.: Partially commutative linear logic: sequent calculus and phase semantics. In: Abrusci, M., Casadio, C. (eds.) Proofs and Linguistic Categories, Proceedings 1996 Roma Workshop. Cooperativa Libraria Universitaria Editrice Bologna (1996)

[de Groote (2001)]de Groote, P.: Towards abstract categorial grammars. In: Proceedings of ACL 2001, pp. 148–155 (2001)

[de Groote and Maarek (2007)]de Groote, P., Maarek, S.: Type-theoretic extensions of abstract categorial grammars. In: Muskens, R. (ed.) Proceedings of Workshop on New Directions in Type-theoretic Grammars, pp. 19–30 (2007)

[Hendriks (1995)]Hendriks, P.: Ellipsis and multimodal categorial type logic. In: Morrill, G.V., Oehrle, R.T. (eds.) Formal Grammar: Proceedings of the Conference of the European Summer School in Logic, Language and Information, Barcelona (1995)

[Kubota and Pollard (2010)]Kubota, Y., Pollard, C.: Phonological Interpretation into Preordered Algebras. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) MOL 10/11. LNCS (LNAI), vol. 6149, pp. 200–209. Springer, Heidelberg (2010)

[Montague (1973)]Montague, R.: The proper treatment of quantification in ordinary English. In: Hintikka, J., Moravcsik, J.M., Suppes, P. (eds.) Approaches to Natural Language, pp. 221–242. D. Reidel, Dordrecht (1973)

[Moortgat (1997)]Moortgat, M.: Categorial Type Logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, pp. 93–177. Elsevier, Amsterdam (1997)

[Morrill and Merenciano (1996)]Morrill, G., Merenciano, J.-M.: Generalizing discontinuity. Traitement Automatique des Langues 27(2), 119–143 (1996)

[Morrill and Solias (1993)]Morrill, G., Solias, T.: Tuples, discontinuity, and gapping in categorial grammar. In: Proceedings of EACL 6, pp. 287–297. Association for Computational Linguistics, Morristown (1993)

[Morrill et al. (2011)]Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20, 1–48 (2011)

[Morrill (1994)]Morrill, G.V.: Type Logical Grammar: Categorial Logic of Signs. Kluwer Academic Publishers, Dordrecht (1994)

[Muskens (2003)]Muskens, R.: Language, lambdas, and logic. In: Kruijff, G.-J., Oehrle, R. (eds.) Resource Sensitivity in Binding and Anaphora, pp. 23–54. Kluwer (2003)

[Muskens (2007)]Muskens, R.: Separating syntax and combinatorics in categorial grammar. Research on Language and Computation 5(3), 267–285 (2007)

[Oehrle (1987)]Oehrle, R.T.: Boolean properties in the analysis of gapping. In: Huck, G.J., Ojeda, A.E. (eds.) Syntax and Semantics 20: Discontinuous Constituency, pp. 203–240. Academic Press (1987)

[Oehrle (1994)]Oehrle, R.T.: Term-labeled categorial type systems. Linguistics and Philosophy 17(6), 633–678 (1994)

[Pogodalla and Pompigne (2011)]Pogodalla, S., Pompigne, F.: Controlling Extraction in Abstract Categorial Grammars. In: FG 2010, Copenhagen, Denmark (2011)

[Polakow and Pfenning (1999)]Polakow, J., Pfenning, F.: Natural Deduction for Intuitionistic Non-commutative Linear Logic. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 295–309. Springer, Heidelberg (1999)

[Siegel (1987)]Siegel, M.A.: Compositionality, case, and the scope of auxiliaries. Linguistics and Philosophy 10(1), 53–75 (1987)

[Steedman (1990)]Steedman, M.: Gapping as constituent coordination. Linguistics and Philosophy 13(2), 207–263 (1990)

L-Completeness of the Lambek Calculus with the Reversal Operation

Stepan Kuznetsov

Moscow State University
skuzn@inbox.ru

Abstract. We extend the Lambek calculus with rules for a unary operation corresponding to language reversal and prove that this calculus is complete with respect to the class of models on subsets of free semigroups (L-models). We also prove that categorial grammars based on this calculus generate precisely all context-free languages without the empty word.

1 The Lambek Calculus and L-Models

We consider the calculus L, introduced in [2]. The set $\mathrm{Pr} = \{p_1, p_2, p_3, \dots\}$ is called the set of *primitive types*. *Types* of L are built from primitive types using three binary connectives: $\backslash$ *(left division)*, $/$ *(right division)*, and $\cdot$ *(multiplication)*; we shall denote the set of all types by Tp. Capital letters $(A, B, \dots)$ range over types. Capital Greek letters (except Σ) range over finite (possibly empty) sequences of types; Λ stands for the empty sequence. Expressions of the form $\Gamma \to C$, where $\Gamma \neq \Lambda$, are called *sequents* of L.

Axioms: $A \to A$.

Rules:

$$\frac{A\Pi \to B}{\Pi \to A \setminus B}\ (\to \setminus),\ \Pi \neq \Lambda \qquad \frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma \Pi (A \setminus B) \Delta \to C}\ (\setminus \to)$$

$$\frac{\Pi A \to B}{\Pi \to B \,/\, A}\ (\to /),\ \Pi \neq \Lambda \qquad \frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma (B \,/\, A) \Pi \Delta \to C}\ (/ \to)$$

$$\frac{\Pi \to A \quad \Delta \to B}{\Pi \Delta \to A \cdot B}\ (\to \cdot) \qquad \frac{\Gamma A B \Delta \to C}{\Gamma (A \cdot B) \Delta \to C}\ (\cdot \to)$$

$$\frac{\Pi \to A \quad \Gamma A \Delta \to C}{\Gamma \Pi \Delta \to C}\ (\mathrm{cut})$$

Now let Σ be an alphabet (an arbitrary nonempty set, finite or countable). By Σ^+ we denote the set of all nonempty words over Σ; the set of all words over Σ, including the empty word, is denoted by Σ^*. The set Σ^+ with the operation of word concatenation is the *free semigroup* generated by Σ. Subsets of Σ^+ are called *languages* over Σ. The three connectives of L correspond to three natural operations on languages ($M, N \subseteq \Sigma^+$): $M \cdot N \leftrightharpoons \{uv \mid u \in M, v \in N\}$, $M \setminus N \leftrightharpoons$

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 151–160, 2012.

$\{u \in \Sigma^+ \mid (\forall v \in M)\, vu \in N\}$, and $N\,/\,M \leftrightharpoons \{u \in \Sigma^+ \mid (\forall v \in M)\, uv \in N\}$ ("$\leftrightharpoons$" here and further means "equals by definition").

An *L-model* is a pair $\mathcal{M} = \langle \Sigma, w \rangle$, where Σ is an alphabet and w is a function that maps Lambek calculus types to languages over Σ, such that $w(A \cdot B) = w(A) \cdot w(B)$, $w(A \setminus B) = w(A) \setminus w(B)$, and $w(B\,/\,A) = w(B)\,/\,w(A)$ for all $A, B \in \mathrm{Tp}$. Obviously, w can be defined on primitive types in an arbitrary way, and then it is uniquely propagated to all types.

A sequent of the form $F \to G$ is considered *true* in a model $\mathcal{M}$ ($\mathcal{M} \vDash F \to G$) if $w(F) \subseteq w(G)$. L-models give sound and complete semantics for L, due to the following theorem:

Theorem 1. *A sequent $F \to G$ is provable in* L *if and only if it is true in all L-models.*

This theorem is proved in [6]; its special case for the product-free fragment (where we keep only types without multiplication) is much easier and appears in [1]. (The notion of truth in an L-model and this theorem can be easily generalized to sequents with more than one type on the left, since $\mathrm{L} \vdash F_1 F_2 \dots F_n \to G$ if and only if $\mathrm{L} \vdash F_1 \cdot F_2 \cdot \ldots \cdot F_n \to G$.)

2 The Lambek Calculus with the Reversal Operation ($\mathbf{L}^{\mathbf{R}}$)

Now let us consider an extra operation on languages, the *reversal.* For $u = a_1 a_2 \dots a_n$ ($a_1, \dots, a_n \in \Sigma$, $n \geq 1$) let $u^{\mathrm{R}} \leftrightharpoons a_n \dots a_2 a_1$, and for $M \subseteq \Sigma^+$ let $M^{\mathrm{R}} \leftrightharpoons \{u^{\mathrm{R}} \mid u \in M\}$. Let us enrich the language of the Lambek calculus with a new unary connective $^{\mathrm{R}}$ (written in the postfix form, A^{R}). We shall denote the extended set of types by Tp^{R}. If $\Gamma = A_1 A_2 \dots A_n$, then $\Gamma^{\mathrm{R}} \leftrightharpoons A_n^{\mathrm{R}} \dots A_2^{\mathrm{R}} A_1^{\mathrm{R}}$.

The notion of L-model is also easily adapted to the new language by adding an additional constraint on w: $w(A^{\mathrm{R}}) = w(A)^{\mathrm{R}}$.

The calculus L^{R} is obtained from L by adding three new rules for $^{\mathrm{R}}$:

$$\frac{\Gamma \to C}{\Gamma^{\mathrm{R}} \to C^{\mathrm{R}}}\ ({}^{\mathrm{R}} \to {}^{\mathrm{R}}) \qquad \frac{\Gamma A^{\mathrm{RR}} \Delta \to C}{\Gamma A \Delta \to C}\ ({}^{\mathrm{RR}} \to)_{\mathrm{E}} \qquad \frac{\Gamma \to C^{\mathrm{RR}}}{\Gamma \to C}\ (\to {}^{\mathrm{RR}})_{\mathrm{E}}$$

It is easy to see that L^{R} is sound with respect to L-models.

Lemma 1. *The calculus* L^{R} *is a conservative extension of* L *(if $F, G \in \mathrm{Tp}$, then $\mathrm{L}^{\mathrm{R}} \vdash F \to G$ if and only if $\mathrm{L} \vdash F \to G$).*

Proof. The "if" part is obvious. The "only if" part follows from L-completeness of L and L-soundness of L^{R}: if $F \to G$ is provable in L^{R}, then it is true in all L-models, and, therefore, is provable in L.

L-completeness for the product-free fragment of L^{R} is proved in [4] by a modification of Buszkowski's argument [1] (in [4] the reversal connective is called *involution* and denoted by $\breve{}$ instead of $^{\mathrm{R}}$; the calculus is formulated in a different, but equivalent way). In [4] one can also find a proof of L-completeness of the division-free fragment (where only $\cdot$ and $^{\mathrm{R}}$ connectives are kept). We shall prove L-completeness of the whole calculus.

Theorem 2. *A sequent $F \to G$ ($F, G \in \mathrm{Tp}^{\mathrm{R}}$) is provable in L^{R} if and only if it is true in all L-models.*

A variant of this calculus that allows empty antecedents (an extension with the $^{\mathrm{R}}$ connective of L^*, the variant of L without the restriction $\Pi \neq \Lambda$ on the $(\to \backslash)$ and $(\to /)$ rules) is presented in [3]. The calculus L^* itself is complete with respect to L-models allowing empty words in the languages (free monoid models) [7], but L-completeness of its extension with the $^{\mathrm{R}}$ connective is still an open problem.

3 Equivalences in L^{R} and Normal Form for Types

Types A and B are called *equivalent* in L^{R} (denotation: $A \leftrightarrow B$), if $\mathrm{L}^{\mathrm{R}} \vdash A \to B$ and $\mathrm{L}^{\mathrm{R}} \vdash B \to A$. The relation $\leftrightarrow$ is reflexive, symmetric, and transitive (due to the rule (cut)). Using (cut) one can prove that if $\mathrm{L}^{\mathrm{R}} \vdash F_1 \to G_1$, $F_1 \leftrightarrow F_2$, and $G_1 \leftrightarrow G_2$, then $\mathrm{L}^{\mathrm{R}} \vdash F_2 \to G_2$. Also, $\leftrightarrow$ is a *congruence relation,* in the sense of the following lemma (checked explicitly):

Lemma 2. *If $A_1 \leftrightarrow A_2$ and $B_1 \leftrightarrow B_2$, then $A_1 \cdot B_1 \leftrightarrow A_2 \cdot B_2$, $A_1 \backslash B_1 \leftrightarrow A_2 \backslash B_2$, $B_1 / A_1 \leftrightarrow B_2 / A_2$, $A_1^{\mathrm{R}} \leftrightarrow A_2^{\mathrm{R}}$.*

The following lemma is checked explicitly by presenting the corresponding derivations in L^{R}:

Lemma 3. *The following equivalences hold in L^{R}:*

1. $(A \cdot B)^{\mathrm{R}} \leftrightarrow B^{\mathrm{R}} \cdot A^{\mathrm{R}}$;
2. $(A \backslash B)^{\mathrm{R}} \leftrightarrow B^{\mathrm{R}} / A^{\mathrm{R}}$;
3. $(B / A)^{\mathrm{R}} \leftrightarrow A^{\mathrm{R}} \backslash B^{\mathrm{R}}$;
4. $A^{\mathrm{RR}} \leftrightarrow A$.

For $A \in \mathrm{Tp}^{\mathrm{R}}$ we define $tr(A)$ by induction on the number of connectives in A:

1. $tr(p_i) \leftrightharpoons p_i$;
2. $tr(p_i^{\mathrm{R}}) \leftrightharpoons p_i^{\mathrm{R}}$;
3. $tr(A \cdot B) \leftrightharpoons tr(A) \cdot tr(B)$;
4. $tr(A \backslash B) \leftrightharpoons tr(A) \backslash tr(B)$;
5. $tr(B / A) \leftrightharpoons tr(B) / tr(A)$;
6. $tr((A \cdot B)^{\mathrm{R}}) \leftrightharpoons tr(B^{\mathrm{R}}) \cdot tr(A^{\mathrm{R}})$;
7. $tr((A \backslash B)^{\mathrm{R}}) \leftrightharpoons tr(B^{\mathrm{R}}) / tr(A^{\mathrm{R}})$;
8. $tr((B / A)^{\mathrm{R}}) \leftrightharpoons tr(A^{\mathrm{R}}) \backslash tr(B^{\mathrm{R}})$;
9. $tr(A^{\mathrm{RR}}) \leftrightharpoons tr(A)$.

The following lemma is proved by induction using Lemma 2 and Lemma 3:

Lemma 4. *Any $A \in \mathrm{Tp}^{\mathrm{R}}$ is equivalent to $tr(A)$.*

We call $tr(A)$ the *normal form* of A. In the normal form, the $^{\mathrm{R}}$ connective can appear only on occurrences of primitive types.

4 L-Completeness of L^{R} (Proof)

Now we are going to prove Theorem 2 (the "if" part) by contraposition. Let $\mathrm{L}^{\mathrm{R}} \nvdash F_0 \to G_0$. We need to construct a countermodel for $F_0 \to G_0$, i.e., a model in which this sequent is not true.

Let $\mathrm{Pr}' \leftrightharpoons \mathrm{Pr} \cup \{p^{\mathrm{R}} \mid p \in \mathrm{Pr}\}$, and let L' be the Lambek calculus with Pr' taken as the set of primitive types instead of Pr. Here $^{\mathrm{R}}$ is not a connective, and p^{R} is considered just a new primitive type, independent from p. Obviously, if $\mathrm{L}' \vdash F \to G$, then $\mathrm{L}^{\mathrm{R}} \vdash F \to G$.

Let $F \leftrightharpoons tr(F_0)$, $G \leftrightharpoons tr(G_0)$. Then $\mathrm{L}^{\mathrm{R}} \nvdash F \to G$, whence $\mathrm{L}' \nvdash F \to G$. The calculus L' is essentially the same as L, therefore Theorem 1 gives us a structure $\mathcal{M} = \langle \varSigma, w \rangle$ such that $w(F) \not\subseteq w(G)$. The structure $\mathcal{M}$ indeed falsifies $F \to G$, but it is not a model in the sense of our new language: some of the conditions $w(p_i^{\mathrm{R}}) = w(p_i)^{\mathrm{R}}$ might be not satisfied.

Let $\varPhi$ be the set of all subtypes of $F \to G$ (including F and G themselves; the notion of subtype is understood in the sense of L^{R}). The construction of $\mathcal{M}$ (see [6]) guarantees that $w(A) \neq \varnothing$ for all $A \in \varPhi$. This is the only specific property of $\mathcal{M}$ we shall need.

We introduce an inductively defined counter $f(A)$, $A \in \varPhi$: $f(p_i) \leftrightharpoons 1$, $f(p_i^{\mathrm{R}}) \leftrightharpoons 1$, $f(A \cdot B) \leftrightharpoons f(A) + f(B) + 10$, $f(A \setminus B) \leftrightharpoons f(B)$, $f(B \,/\, A) \leftrightharpoons f(B)$. Let $K \leftrightharpoons \max\{f(A) \mid A \in \varPhi\}$, $N \leftrightharpoons 2K + 25$ (N should be odd, greater then K, and big enough itself).

Let $\varSigma_1 \leftrightharpoons \varSigma \times \{1, \dots, N\}$. We shall denote the pair $\langle a, j \rangle \in \varSigma_1$ by $a^{(j)}$. Elements of $\varSigma$ and $\varSigma_1$ will be called *letters* and *symbols* respectively. A symbol can be *even* or *odd* depending on the parity of the superscript. Consider a homomorphism $h\colon \varSigma^+ \to \varSigma_1^+$, defined as follows: $h(a) \leftrightharpoons a^{(1)} a^{(2)} \dots a^{(N)}$ ($a \in \varSigma$), $h(a_1 \dots a_n) \leftrightharpoons h(a_1) \dots h(a_n)$. Let $P \leftrightharpoons h(\varSigma^+) = \{a_1^{(1)} \dots a_1^{(N)} \dots a_n^{(1)} \dots a_n^{(N)} \mid n \geq 1,\ a_i \in \varSigma\}$. Note that h is a bijection between $\varSigma^+$ and P.

Lemma 5. *For all $M, N \subseteq \varSigma^+$ we have*

1. *$h(M \cdot N) = h(M) \cdot h(N)$;*
2. *if $M \neq \varnothing$, then $h(M \setminus N) = h(M) \setminus h(N)$ and $h(N \,/\, M) = h(N) \,/\, h(M)$.*

Proof

1. By the definition of a homomorphism.
2. $\boxed{\subseteq}$ Let $u \in h(M \setminus N)$. Then $u = h(u')$ for some $u' \in M \setminus N$. For all $v' \in M$ we have $v'u' \in N$. Take an arbitrary $v \in h(M)$, $v = h(v')$ for some $v' \in M$. Since $u' \in M \setminus N$, $v'u' \in N$, whence $vu = h(v')h(u') = h(v'u') \in h(N)$. Therefore $u \in h(M) \setminus h(N)$.
   $\boxed{\supseteq}$ Let $u \in h(M) \setminus h(N)$. First we claim that $u \in P$. Suppose the contrary: $u \notin P$. Take $v' \in M$ (M is nonempty by assumption). Since $v = h(v') \in P$, $vu \notin P$. On the other hand, $vu \in h(N) \subseteq P$. Contradiction. Now, since $u \in P$, $u = h(u')$ for some $u' \in \varSigma^+$. For an arbitrary $v' \in M$ and $v \leftrightharpoons h(v')$ we have $h(v'u') = vu \in h(N)$, whence $v'u' \in N$, whence $u' \in M \setminus N$. Therefore, $u = h(u') \in h(M \setminus N)$.
   The / case is handled symmetrically.

We construct a new model $\mathcal{M}_1 = \langle \Sigma_1, w_1 \rangle$, where $w_1(z) \leftrightharpoons h(w(z))$ ($z \in \mathrm{Pr}'$). Due to Lemma 5, $w_1(A) = h(w_1(A))$ for all $A \in \Phi$, whence $w_1(F) = h(w(F)) \not\subseteq h(w(G)) = w_1(G)$ ($\mathcal{M}_1$ is also a countermodel in the language without $^{\mathrm{R}}$).

Now we introduce several auxiliary subsets of Σ_1^+ (by $\mathrm{Subw}(M)$ we denote the set of all nonempty subwords of words from M, i.e. $\mathrm{Subw}(M) \leftrightharpoons \{u \in \Sigma_1^+ \mid (\exists v_1, v_2 \in \Sigma_1^*)\, v_1 u v_2 \in M\}$):
$T_1 \leftrightharpoons \{u \in \Sigma_1^+ \mid u \notin \mathrm{Subw}(P \cup P^{\mathrm{R}})\}$;
$T_2 \leftrightharpoons \{u \in \mathrm{Subw}(P \cup P^{\mathrm{R}}) \mid$ the first or the last symbol of u is even$\}$;
$E \leftrightharpoons \{u \in \mathrm{Subw}(P \cup P^{\mathrm{R}}) - (P \cup P^{\mathrm{R}}) \mid$ both the first symbol and the last symbol of u are odd$\}$.

The sets P, P^{R}, T_1, T_2, and E form a partition of Σ_1^+ into nonintersecting parts. For example, $a^{(1)}b^{(10)}a^{(2)} \in T_1$, $a^{(N)}b^{(1)} \dots b^{(N-1)} \in T_2$, $a^{(7)}a^{(6)}a^{(5)} \in E$ ($a, b \in \Sigma$).

Let $T \leftrightharpoons T_1 \cup T_2$, $T_i(k) \leftrightharpoons \{u \in T_i \mid |u| \geq k\}$ ($i = 1, 2$, $|u|$ is the length of u), $T(k) \leftrightharpoons T_1(k) \cup T_2(k) = \{u \in T \mid |u| \geq k\}$.

Note that if the first or the last symbol of u is even, then it belongs to T, no matter whether it belongs to $\mathrm{Subw}(P \cup P^{\mathrm{R}})$.

The index k (possibly with subscripts) here and further ranges from 1 to K. For all k we have $T(k) \supseteq T(K)$.

Lemma 6

1. *$P \cdot P \subseteq P$, $P^{\mathrm{R}} \cdot P^{\mathrm{R}} \subseteq P^{\mathrm{R}}$;*
2. *$T^{\mathrm{R}} = T$, $T(k)^{\mathrm{R}} = T(k)$;*
3. *$P \cdot P^{\mathrm{R}} \subseteq T(K)$, $P^{\mathrm{R}} \cdot P \subseteq T(K)$;*
4. *$P \cdot T \subseteq T(K)$, $T \cdot P \subseteq T(K)$;*
5. *$P^{\mathrm{R}} \cdot T \subseteq T(K)$, $T \cdot P^{\mathrm{R}} \subseteq T(K)$;*
6. *$T \cdot T \subseteq T$;*

Proof

1. Obvious.
2. Directly follows from our definitions.
3. Any element of $P \cdot P^{\mathrm{R}}$ or $P^{\mathrm{R}} \cdot P$ does not belong to $\mathrm{Subw}(P \cup P^{\mathrm{R}})$ and its length is at least $2N > K$. Therefore it belongs to $T_1(K) \subseteq T(K)$.
4. Let $u \in P$ and $v \in T$. If $v \in T_1$, then uv is also in T_1. Let $v \in T_2$. If the last symbol of v is even, then $uv \in T$. If the last symbol of v is odd, then $uv \notin \mathrm{Subw}(P \cup P^{\mathrm{R}})$, whence $uv \in T_1 \subseteq T$. Since $|uv| > |u| \geq N > K$, $uv \in T(K)$.
   The claim $T \cdot P \subseteq T$ is handled symmetrically.
5. $P^{\mathrm{R}} \cdot T = P^{\mathrm{R}} \cdot T^{\mathrm{R}} = (T \cdot P)^{\mathrm{R}} \subseteq T(K)^{\mathrm{R}} = T(K)$. $T \cdot P^{\mathrm{R}} = T^{\mathrm{R}} \cdot P^{\mathrm{R}} = (P \cdot T)^{\mathrm{R}} \subseteq T(K)^{\mathrm{R}} = T(K)$.
6. Let $u, v \in T$. If at least one of these two words belongs to T_1, then $uv \in T_1$. Let $u, v \in T_2$. If the first symbol of u or the last symbol of v is even, then $uv \in T$. In the other case u ends with an even symbol, and v starts with an even symbol. But then we have two consecutive even symbols in uv, therefore $uv \in T_1$.

Let us call words of the form $a^{(i)}a^{(i+1)}a^{(i+2)}$, $a^{(N-1)}a^{(N)}b^{(1)}$, and $a^{(N)}b^{(1)}b^{(2)}$ ($a, b \in \Sigma$, $1 \le i \le N-2$) *valid triples of type I* and their reversals (namely, $a^{(i+2)}a^{(i+1)}a^{(i)}$, $b^{(1)}a^{(N)}a^{(N-1)}$, and $b^{(2)}b^{(1)}a^{(N)}$) *valid triples of type II.* Note that valid triples of type I (resp., of type II) are the only possible three-symbol subwords of words from P (resp., P^{R}).

Lemma 7. *A word u of length at least three is a subword of a word from $P \cup P^{\mathrm{R}}$ if and only if any three-symbol subword of u is a valid triple of type I or II.*

Proof. The nontrivial part is "if". We proceed by induction on $|u|$. Induction base ($|u| = 3$) is trivial. Let u be a word of length $m+1$ satisfying the condition and let $u = u'x$ ($x \in \Sigma_1$). By induction hypothesis ($|u'| = m$), $u' \in \mathrm{Subw}(P \cup P^{\mathrm{R}})$. Let $u' \in \mathrm{Subw}(P)$ (the other case is handled symmetrically); u' is a subword of some word $v \in P$. Consider the last three symbols of u. Since the first two of them also belong to u', this three-symbol word is a valid triple of type I, not type II. If it is of the form $a^{(i)}a^{(i+1)}a^{(i+2)}$ or $a^{(N)}b^{(1)}b^{(2)}$, then x coincides with the symbol next to the occurrence of u' in v, and therefore $u = u'x$ is also a subword of v. If it is of the form $a^{(N-1)}a^{(N)}b^{(1)}$, then, provided $v = v_1u'v_2$, v_1u' is also an element of P, and so is the word $v_1u'b^{(1)}b^{(2)}\dots b^{(N)}$, which contains $u = u'b^{(1)}$ as a subword. Thus, in all cases $u \in \mathrm{Subw}(P)$.

Now we construct one more model $\mathcal{M}_2 = \langle \Sigma_1, w_2 \rangle$, where $w_2(p_i) \leftrightharpoons w_1(p_i) \cup w_1(p_i^{\mathrm{R}})^{\mathrm{R}} \cup T$, $w_2(p_i^{\mathrm{R}}) \leftrightharpoons w_1(p_i)^{\mathrm{R}} \cup w_1(p_i^{\mathrm{R}}) \cup T$. This model is a model even in the sense of the enriched language. To finish the proof, we need to check that $\mathcal{M}_2 \not\models F \to G$.

Lemma 8. *For any $A \in \Phi$ the following holds:*

1. $w_2(A) \subseteq P \cup P^{\mathrm{R}} \cup T$;
2. $w_2(A) \supseteq T(f(A))$;
3. $w_2(A) \cap P = w_1(A)$ *(in particular, $w_2(A) \cap P \neq \varnothing$);*
4. $w_2(A) \cap P^{\mathrm{R}} = w_1(tr(A^{\mathrm{R}}))^{\mathrm{R}}$ *(in particular, $w_2(A) \cap P^{\mathrm{R}} \neq \varnothing$).*

Proof. We prove all the statements simultaneously by induction on type A. The induction base is trivial. Further we shall refer to the i-th statement of the induction hypothesis ($i = 1, 2, 3, 4$) as "IH-i".

1. Consider three possible cases.

a) $A = B \cdot C$. Then $w_2(A) = w_2(B) \cdot w_2(C) \subseteq (P \cup P^{\mathrm{R}} \cup T) \cdot (P \cup P^{\mathrm{R}} \cup T) \subseteq P \cup P^{\mathrm{R}} \cup T$ (Lemma 6).

b) $A = B \setminus C$. Suppose the contrary: in $w_2(A)$ there exists an element $u \in E$. Then $vu \in w_2(C)$ for any $v \in w_2(B)$. We consider several subcases and show that each of those leads to a contradiction.

i) $u \in \mathrm{Subw}(P)$, and the superscript of the first symbol of u is not 1. Let the first symbol of u be $a^{(i)}$. Note that i is odd and $i > 2$. Take $v = a^{(3)} \dots a^{(N)}a^{(1)} \dots a^{(i-1)}$. The word v has length at least $N \ge K$ and ends with an even symbol, therefore $v \in T(K) \subseteq T(f(B)) \subseteq w_2(B)$ (IH-2). On the other hand, $vu \in \mathrm{Subw}(P)$ and the first symbol and the last symbol of vu are odd. Therefore, $vu \in E$ and $vu \in w_2(C)$, but $w_2(C) \cap E = \varnothing$ (IH-1). Contradiction.

ii) $u \in \mathrm{Subw}(P)$, and the first symbol of u is $a^{(1)}$ (then the superscript of the last symbol of u is not N, because otherwise $u \in P$). Take $v \in w_2(B) \cap P$ (this set is nonempty due to IH-3). The first and the last symbol of vu is odd, and $vu \in \mathrm{Subw}(P) - P$, therefore $vu \in E$. Contradiction.

iii) $u \in \mathrm{Subw}(P^{\mathrm{R}})$, and the superscript of the first symbol of u is not N (the first symbol of u is $a^{(i)}$, i is odd). Take $v = a^{(N-2)} \dots a^{(1)} a^{(N)} \dots a^{(i+1)} \in T(K) \subseteq w_2(B)$. Again, $vu \in E$.

iv) $u \in \mathrm{Subw}(P^{\mathrm{R}})$, and the first symbol of u is $a^{(N)}$. Take $v \in w_2(B) \cap P^{\mathrm{R}}$ (nonempty due to IH-4). $vu \in E$.

c) $A = C \,/\, B$. Proceed symmetrically.

2. Consider three possible cases.

a) $A = B \cdot C$. Let $k_1 \leftrightharpoons f(B)$, $k_2 \leftrightharpoons f(C)$, $k \leftrightharpoons k_1 + k_2 + 10 = f(A)$. Due to IH-2, $w_2(B) \supseteq T(k_1)$ and $w_2(C) \supseteq T(k_2)$. Take $u \in T(k)$. We have to prove that $u \in w_2(A)$. Consider several subcases.

i) $u \in T_1(k)$. By Lemma 7 ($|u| \ge k > 3$ and $u \notin \mathrm{Subw}(P \cup P^{\mathrm{R}})$) in u there is a three-symbol subword xyz that is not a valid triple of type I or II. Divide the word u into two parts, $u = u_1 u_2$, such that $|u_1| \ge k_1 + 5$, $|u_2| \ge k_2 + 5$. If needed, shift the border between parts by one symbol to the left or to the right, so that the subword xyz lies entirely in one part. Let this part be u_2 (the other case is handled symmetrically). Then $u_2 \in T_1(k_2)$. If u_1 is also in T_1, then the proof is finished. Consider the other case. Note that in any word from $\mathrm{Subw}(P \cup P^{\mathrm{R}})$ among any three consecutive symbols at least one is even. Shift the border to the left by at most 2 symbols to make the last symbol of u_1 even. Then $u_1 \in T(k_1)$, and u_2 remains in $T_1(k_2)$. Thus $u = u_1 u_2 \in T(k_1) \cdot T(k_2) \subseteq w_2(B) \cdot w_2(C) = w_2(A)$.

ii) $u \in T_2(k)$. Let u end with an even symbol (the other case is symmetric). Divide the word u into two parts, $u = u_1 u_2$, $|u_1| \ge k_1 + 5$, $u_2 \ge k_2 + 5$, and shift the border (if needed), so that the last symbol of u_1 is even. Then both u_1 and u_2 end with an even symbol, and therefore $u_1 \in T(k_1)$ and $u_2 \in T(k_2)$.

b) $A = B \setminus C$. Let $k \leftrightharpoons f(C) = f(A)$. By IH-2, $w_2(C) \supseteq T(k)$. Take $u \in T(k)$ and an arbitrary $v \in w_2(B) \subseteq P \cup P^{\mathrm{R}} \cup T$. By Lemma 6, statements 4–6, $vu \in (P \cup P^{\mathrm{R}} \cup T) \cdot T \subseteq T$, and since $|vu| > |u| \ge k$, $vu \in T(k) \subseteq w_2(C)$. Thus $u \in w_2(A)$.

c) $A = C \,/\, B$. Symmetrically.

3. Consider three possible cases.

a) $A = B \cdot C$.

$\boxed{\supseteq}$ $u \in w_1(A) = w_1(B) \cdot w_1(C) \subseteq w_2(B) \cdot w_2(C) = w_2(A)$ (IH-3); $u \in P$.

$\boxed{\subseteq}$ Suppose $u \in P$ and $u \in w_2(A) = w_2(B) \cdot w_2(C)$. Then $u = u_1 u_2$, where $u_1 \in w_2(B)$ and $u_2 \in w_2(C)$. First we claim that $u_1 \in P$. Suppose the contrary, $u_1 \notin P$. By IH-1, $u_1 \in P^{\mathrm{R}} \cup T$, $u_2 \in P \cup P^{\mathrm{R}} \cup T$, and therefore $u = u_1 u_2 \in (P^{\mathrm{R}} \cup T) \cdot (P \cup P^{\mathrm{R}} \cup T) \subseteq P^{\mathrm{R}} \cup T$ (Lemma 6, statements 1, 3–6). Hence $u \notin P$. Contradiction. Thus, $u_1 \in P$. Similarly, $u_2 \in P$, and by IH-3 we obtain $u_1 \in w_1(B)$ and $u_2 \in w_1(C)$, whence $u = u_1 u_2 \in w_1(A)$.

b) $A = B \setminus C$.

$\boxed{\supseteq}$ Take $u \in w_1(B \setminus C)$. For any $v \in w_1(B)$ we have $vu \in w_1(C)$. We claim that $u \in w_2(B \setminus C)$. Take $v \in w_2(B) \subseteq P \cup P^{\mathrm{R}} \cup T$ (IH-1). If $v \in P$, then $v \in w_1(B)$ (IH-3), and $vu \in w_1(C) \subseteq w_2(C)$ (IH-3). If $v \in P^{\mathrm{R}} \cup T$, then $vu \in (P^{\mathrm{R}} \cup T) \cdot P \subseteq T(K) \subseteq w_2(C)$ (Lemma 6, statements 3 and 4, and IH-2). Therefore, $u \in w_2(B) \setminus w_2(C) = w_2(B \setminus C)$; also we have $u \in P$, since $w_1(B \setminus C) \subseteq P$.

$\boxed{\subseteq}$ If $u \in w_2(B \setminus C)$ and $u \in P$, then for any $v \in w_1(B) \subseteq w_2(B)$ we have $vu \in w_2(C)$. Since $v, u \in P$, $vu \in P$. By IH-3, $vu \in w_1(C)$. Thus $u \in w_1(B \setminus C)$.

c) $A = C \,/\, B$. Symmetrically.

4. Consider three cases.

a) $A = B \cdot C$. Then $tr(A^{\mathrm{R}}) = tr(C^{\mathrm{R}}) \cdot tr(B^{\mathrm{R}})$.

$\boxed{\supseteq}$ $u \in w_1(tr(A^{\mathrm{R}}))^{\mathrm{R}} = w_1(tr(C^{\mathrm{R}}) \cdot tr(B^{\mathrm{R}}))^{\mathrm{R}} = \big(w_1(tr(C^{R})) \cdot w_1(tr(B^{\mathrm{R}}))\big)^{\mathrm{R}} = w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}} \cdot w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}} \subseteq w_2(B) \cdot w_2(C) = w_2(A)$ (IH-4); $u \in P^{\mathrm{R}}$.

$\boxed{\subseteq}$ Let $u \in P^{\mathrm{R}}$ and $u \in w_2(A) = w_2(B) \cdot w_2(C)$. Then $u = u_1 u_2$, where $u_1 \in w_2(B)$, $u_2 \in w_2(C)$. We claim that $u_1, u_2 \in P^{\mathrm{R}}$. Suppose the contrary: $u_1 \notin P^{\mathrm{R}}$. Then $u_1 \in P \cup T$ (IH-1), $u_2 \in P \cup P^{\mathrm{R}} \cup T$, whence $u = u_1 u_2 \in (P \cup T) \cdot (P \cup P^{\mathrm{R}} \cup T) \subseteq P \cup T$. Contradiction ($u \in P^{\mathrm{R}}$). Thus, $u_1 \in P^{\mathrm{R}}$, and therefore $u_2 \in P^{R}$, and, using IH-4, we obtain $u_1 \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}}$, $u_2 \in w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}}$. Hence $u = u_1 u_2 \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}} \cdot w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}} = \big(w_1(tr(C^{\mathrm{R}})) \cdot w_1(tr(B^{\mathrm{R}}))\big)^{\mathrm{R}} = w_1(tr(C^{\mathrm{R}}) \cdot tr(B^{\mathrm{R}}))^{\mathrm{R}} = w_1(tr(A^{\mathrm{R}}))^{\mathrm{R}}$.

b) $A = B \setminus C$. Then $tr(A^{\mathrm{R}}) = tr(C^{\mathrm{R}}) \,/\, tr(B^{\mathrm{R}})$.

$\boxed{\supseteq}$ Let $u \in w_1(tr(C^{\mathrm{R}}) \,/\, tr(B^{\mathrm{R}}))^{\mathrm{R}} = w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}} \setminus w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}}$, so for every $v \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}}$ we have $vu \in w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}}$. We claim that $u \in w_2(B \setminus C)$. Take an arbitrary $v \in w_2(B) \subseteq P \cup P^{\mathrm{R}} \cup T$ (IH-1). If $v \in P^{\mathrm{R}}$, then $v \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}}$ (IH-4), whence $vu \in w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}} \subseteq w_2(C)$.
If $v \in P \cup T$, then (since $u \in P^{\mathrm{R}}$) we have $vu \in (P \cup T) \cdot P^{\mathrm{R}} \subseteq T(K) \subseteq w_2(C)$ (Lemma 6 and IH-2).

$\boxed{\subseteq}$ If $u \in w_2(B \setminus C)$ and $u \in P^{\mathrm{R}}$, then for any $v \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}} \subseteq w_2(B)$ we have $vu \in w_2(C)$. Since $v, u \in P^{R}$, $vu \in P^{R}$, therefore $vu \in w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}}$ (IH-4). Thus $u \in w_1(tr(B^{\mathrm{R}}))^{\mathrm{R}} \setminus w_1(tr(C^{\mathrm{R}}))^{\mathrm{R}} = w_1(A^{\mathrm{R}})^{\mathrm{R}}$.

c) $A = C \,/\, B$. Symmetrically.

This completes the proof of Lemma 8.

Since $w_1(F) \not\subseteq w_1(G)$, there exists an element u_0 such that $u_0 \in w_1(F)$ and $u_0 \notin w_1(G)$. Since $u_0 \in P$, $u_0 \in w_2(F)$ and $u_0 \notin w_2(G)$. Therefore, $w_2(F) \not\subseteq w_2(G)$. Since $F_0 \leftrightarrow F$, $G_0 \leftrightarrow G$, and L^{R} is L-sound, we see that $w_2(F_0) = w_2(F)$, $w_2(G_0) = w_2(G)$, and $\mathcal{M}_2$ is a countermodel for $F_0 \to G_0$. This completes the proof of Theorem 2.

Note that we have constructed a countermodel (in the sense of the extended language) for any sequent $F \to G$ that is not provable in L' (this could be potentially weaker than $\mathrm{L}^{\mathrm{R}} \nvdash F \to G$). Thus we get the following statement:

Lemma 9. $\mathrm{L}^{\mathrm{R}} \vdash A_1 \dots A_n \to B$ *if and only if* $\mathrm{L}' \vdash tr(A_1) \dots tr(A_n) \to tr(B)$.

5 $\mathbf{L}^{\mathbf{R}}$-Grammars

The Lambek calculus and its variants are used for describing formal languages via Lambek categorial grammars. An L-*grammar* is a triple $\mathcal{G} = \langle \Sigma, H, \rhd \rangle$, where Σ is a finite alphabet, $H \in \mathrm{Tp}$, and $\rhd$ is a finite correspondence between Tp and Σ ($\rhd \subset \mathrm{Tp} \times \Sigma$). The *language generated by* $\mathcal{G}$ is the set of all nonempty words $a_1 \dots a_n$ over Σ for which there exist types $B_1, \dots, B_n$ such that $\mathrm{L} \vdash B_1 \dots B_n \to H$ and $B_i \rhd a_i$ for all $i \le n$. We denote this language by $\mathfrak{L}(\mathcal{G})$. This class of grammars is *weakly equivalent* to the class of context-free grammars (without ϵ-rules) in the following sense:

Theorem 3. *A formal language without the empty word is context-free if and only if it is generated by some* L*-grammar.* [5]

By modifying our definition in a natural way one can introduce the notion of L^{R}-grammar. These grammars also generate precisely all context-free languages without the empty word:

Theorem 4. *A formal language without the empty word is context-free if and only if it is generated by some* L^{R}*-grammar.*

Proof. The "only if" part follows directly from Theorem 3 due to conservativity of L^{R} over L (Lemma 1).

Let us prove the "if" part. Let $M = \mathfrak{L}(\mathcal{G})$ for some L^{R}-grammar $\mathcal{G}$. Let $\mathcal{G}'$ be the grammar obtained from $\mathcal{G}$ by replacing all the types with their normal forms and considering it as an L'-grammar. By Lemma 9, $\mathfrak{L}(\mathcal{G}') = \mathfrak{L}(\mathcal{G}) = M$, and this language is context-free due to Theorem 3 (because L' and L are essentially the same calculus).

Acknowledgments. I am grateful to Prof. Mati Pentus for fruitful discussions and constant attention to my work. I am also grateful to Prof. Sergei Adian for inspiring techniques of working with words over an alphabet given in his lectures and papers.

This research was supported by the Russian Foundation for Basic Research (grant 11-01-00281-a), by the Presidential Council for Support of Leading Research Schools (grant NSh-65648.2010.1) and by the Scientific and Technological Cooperation Programme Switzerland–Russia (STCP-CH-RU, project "Computational Proof Theory").

References

1. Buszkowski, W.: Compatibility of categorial grammar with an associated category system. Zeitschr. für Math. Logik und Grundl. der Math. 28, 229–238 (1982)
2. Lambek, J.: The mathematics of sentence structure. American Math. Monthly 65(3), 154–170 (1958)
3. Lambek, J.: From categorial grammar to bilinear logic. In: Došen, K., Schroeder-Heister, P. (eds.) Substructural Logics. Studies in Logic and Computation, vol. 2, pp. 128–139. Clarendon Press, Oxford (1993)

4. Minina, V. A.: Completeness of the Lambek syntactic calculus with the involution operation (in Russian). Diploma paper, Dept. of Math. Logic and Theory of Algorithms, Moscow State University (2001)
5. Pentus, M.: Lambek grammars are context free. In: 8th Annual IEEE Symposium on Logic in Computer Science, pp. 429–433. IEEE Computer Society Press, Los Alamitos (1993)
6. Pentus, M.: Models for the Lambek calculus. Annals of Pure and Applied Logic 75(1–2), 179–213 (1995)
7. Pentus, M.: Free monoid completeness of the Lambek calculus allowing empty premises. In: Larrazabal, J.M., Lascar, D., Mints, G. (eds.) Logic Colloquium 1996. LNL, vol. 12, pp. 171–209. Springer, Berlin (1998)

Distributive Full Nonassociative Lambek Calculus with S4-Modalities Is Context-Free

Zhe Lin[1,2,⋆]

[1] Institute of Logic and Cognition, Sun Yat-sen University, Guangzhou, China
[2] Faculty of Mathematics and Computer Science Adam Mickiewicz University,
Poznań, Poland
pennyshaq@gmail.com

Abstract. We study Nonassociative Lambek Calculus with additives, satisfying the distributive law and $S4$-modalities. We prove that the categorial grammars based on it, also enriched with assumptions, generate context-free languages. This extends earlier results of Buszkowski [4] for NL (Nonassociative Lambek Calculus), Buszkowski and Farulewski [6] for DNFL (Distributive Full Nonassociative Lambek Calculus) and Plummer [19], [20] for $\mathrm{NL_{S4}}$ (Nonassociative Lambek Calculus with $S4$-modalities) without assumptions.

1 Introduction

Nonassociative Lambek Calculus NL was introduced by Lambek [13] as a nonassociative version of Syntatic Calculus of Lambek [12]. NL is a complete logic of residuated groupoids. A residuated groupoid $(G, \cdot, \backslash, /, \leq)$ is an ordered algebra such that $(G, \leq)$ is a poset and $\cdot, \backslash, /$ are binary operations on G satisfying the residuation law:

$$a \cdot b \leq c \quad \text{iff} \quad b \leq a\backslash c \quad \text{iff} \quad a \leq c/b$$

for all $a, b, c \in G$.

Categorial grammars based on NL generate precisely the ε-free context-free languages, see [3], [4], [11]. Pentus [18] proves the same for Associative Lambek Calculus. However, using FL (Full Lambek Calculus) [10], one can generate languages which are not context-free. Buszkowski and Farulewski [6] prove that the categorial grammars based on DFNL with finitely many assumptions generate context-free languages. and Similar results for DFNL, supplied with arbitrary unary operations and its residuals, a boolean or Heyting negation are obatianed in the same paper.

NL can be supplied with modal operations $\Diamond$, $\Box^{\downarrow}$, satisfying the adjoint law: $\Diamond A \Rightarrow B$ iff $A \Rightarrow \Box^{\downarrow} B$. This system, denoted by NL$\Diamond$, enriched with modal postulates enables one to use certain structural postulates in a controlled way. Such systems were introduced by Moortgat [15] in connection with Type-Logical

[⋆] The work of author is supported by the grant GD10CZX01 (Guangdong Social Sciences Fund, China)

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 161–172, 2012.

Grammar. The problems of the generative capacity of the categorial grammars based on NL$\Diamond$ and L$\Diamond$ were settled by Jäger [9], [8]. Both are context-free. Plummer [19], [20] studies NL$_{\text{S4}}$, the enrichment of NL$\Diamond$ by S4-axioms (4: $\Diamond\Diamond A \Rightarrow \Diamond A$ and T: $A \Rightarrow \Diamond A$). His results concern the context-freeness of this system.

In [14], we extend Plummer's results for NL$_{\text{S4}}$ with assumptions and prove the polynomial time decidability of the consequence relation of NL$_{\text{S4}}$ and the context-freeness of the corresponding grammars. (We also admit an additional axiom K: $\Diamond(a \cdot b) \leq \Diamond a \cdot \Diamond b$ or additional axioms K$_1$: $\Diamond(a \cdot b) \leq \Diamond a \cdot b$, K$_2$: $\Diamond(a \cdot b) \leq a \cdot \Diamond b$).

In the present paper, we combine these two lines of research and extend the results to NL$_{\text{S4}}$ with distributive lattice and finitely many non-logical assumptions (but we omit the extra axioms e.g. K, K$_1$, K$_2$). Our proof also shows that the same results hold for NL$\Diamond$ with distributive lattice without any additional axiom or NL$\Diamond$ with distributive lattice, admitting 4 or T only.

The key arguments of the proofs in [14] and the present paper are the extended subformula property for Gentzen style axiomatization and the interpolation lemma (interpolation of formula-trees by formulae, following some ideas from [6], [5]). We present a sequent system, which is complete with respect to the class of distributive lattice ordered residuated groupoids with $S4$ unary residuated operators. This system is an extension of Full Nonassociative Lambek Calculus (FNL). Since the distributive law $A \wedge (B \vee C) \Rightarrow (A \wedge B) \vee (A \wedge C)$ is added as a new axiom, this system does not admit cut-elimination. We prove the strong finite model property of this class of lattice ordered residuated groupoids and the extended subformula property for the corresponding Gentzen style axiomatization. Here we adopt the model-theoretic method from [6], [5] to NL$_{\text{S4}}$ enriched with distributive lattice and finitely many non-logical assumptions to prove the extended subformula property. However our proof for the interpolation lemma is new and finer than in earlier papers (e.g. [6], [5], [14], [20]), which yields a simpler description of the set of possible interpolants.

Our interest in modal extensions, additives and non-logical assumptions can be partially motivated by linguistics application. Non-logical assumptions are useful in linguistics, especially when we need some sequents that cannot be derived in the logic. For instance, in the Lambek Calculus we cannot transform $s\backslash(s/s)$ (the type of sentence conjunction) to $vp\backslash(vp/vp)$ (the type of verb phrase conjunction). However, we can add the sequent $s\backslash(s/s) \Rightarrow vp\backslash(vp/vp)$ as an assumption. A typical linguistic usage of additives is connected with lexical ambiguity. For assignments $a \to A_i$, $i = 1, \ldots, n$, by using $\wedge$, one can combine these assignments into a rigid type assignment $a \to A_1 \wedge \ldots \wedge A_n$, in the type lexicon of categorial grammar. Another example was given by Kanazawa [10]. He makes use of additives to encode feature decomposition. For instance, consider the sentence *John walks* and *John becomes rich*. According to Kanazawa, *John* is assigned $sg \wedge np$, where sg denote singular, *walks* type $(np \wedge sg)\backslash s$, *becomes* $((np \wedge sg)\backslash s)/(np \vee ap)$, and *rich* ap. However, such usage of additives may sometimes cause problems (see [16]). Now let us consider the following example discussed in [7]. One can ambiguously assign an accusative personal $np \wedge acc$ or a possessive $np\backslash n$ to a pronoun *her*. Combing both into a single type leads to the assignment to *her* of the type $(np \wedge acc) \wedge (np/n)$. Then, the problem arises of how to prevent the *acc* marking to reassociate with the possessive type np/n.

A possible solution, due to the present author, looks as follows. It can be solved by using additives and reflexive unary modalities $\Box^{\downarrow}$ (satisfying $\Box^{\downarrow}A \Rightarrow A$). Simply, one can replace $np \wedge acc$ by $\Box^{\downarrow}_{acc}np$. Since $\Box^{\downarrow}_{acc}np \Rightarrow np$, $\Box^{\downarrow}_{acc}np$ has the same desired property as $np \wedge acc$. By doing so, we obtain the type $\Box^{\downarrow}_{acc}np \wedge (np/n)$ such that the *acc* marking can never reassociate with np/n. It is more natural to treat the feature as a mark of its mother type than a individual atomic type. Hence, using modalities to characterize the feature of type and additives for ambiguity seems to be a better choice, because one can get rid of some undesired property like $sg \wedge np \Rightarrow sg$. Modalities with reflexive behaviour are used for analyzing different linguistic phenomenons by many authors as well (see Morrill[17], Versmissen [21], Heylen [7] and Bernardi [2]). This behaviour can also be characterized by a pair of residuated modalities $\Diamond$ and $\Box^{\downarrow}$, since $\Diamond\Box^{\downarrow}A \Rightarrow A \Rightarrow \Box^{\downarrow}\Diamond A$ stands in NL$\Diamond$.

The logic with $S4$-modalities also has certain interesting features from the theoretical point of view. Firstly, with axioms 4 and T, one can easily prove $\Diamond\Box A \Leftrightarrow \Box A$ and $\Box\Diamond A \Leftrightarrow \Diamond A$, for any formula A. Hence, such extension might be the simplest modal extension, since there are only two distinguished modal operations $\Diamond$ and $\Box^{\downarrow}$. Secondly, to the best of my knowledge, there is no evidence of using complex modalities like $\Box^{\downarrow}\Box^{\downarrow}\Diamond\Box^{\downarrow}$ in the literature. Due to the results in [19], [20], [14] and the present paper, the generative capacity of the categorial grammars based on logic with $S4$-modalities does not surpass context-freeness. Therefore, it might be safe to consider such extension in linguistic application. Finally, it may be helpful for identifying sufficient conditions when modal postulates preserve context-freeness. For instance, the categorial grammars based on $\mathrm{NL_{S4}}$ enriched $\mathrm{K_1}$ and $\mathrm{K_2}$ generate context-free languages (see [14]), while NL$\Diamond$ with $\mathrm{K_1}$ and $\mathrm{K_2}$ is conjectured to be non-context-free ([19]).

2 Interpolation and Context-Freeness

Let us recall the sequent system of NL$\Diamond$. Formulae (types) are formed out of atomic types $p, q, r \ldots$ by means of three binary operation symbols $\bullet$, $\backslash$, $/$ and two unary operation symbols $\Diamond$, $\Box^{\downarrow}$. Formula trees (formula-structures) are recursively defined as follow: (i) every formula is a formula-tree, (ii) if Γ, Δ are formula-trees, then $(\Gamma \circ \Delta)$ is a formula-tree, (iii) if Γ is a formula-tree, then $\langle\Gamma\rangle$ is a formula-tree. A context is a structure $\Gamma[\circ]$ containing a single occurrence of special substructure $\circ$ (a place for subsitution): $\Gamma[\Delta]$ denotes the result of substitution of Δ for $\circ$ in $\Gamma[\circ]$. Sequents are of the form $\Gamma \Rightarrow A$ such that Γ is a formula tree and A is a formula. NL$\Diamond$ admits the axioms

$$(\mathrm{Id}) \quad A \Rightarrow A$$

and the inference rules

$$(\backslash\mathrm{L}) \quad \frac{\Delta \Rightarrow A; \quad \Gamma[B] \Rightarrow C}{\Gamma[\Delta \circ (A\backslash B)] \Rightarrow C} \qquad (\backslash\mathrm{R}) \quad \frac{A \circ \Gamma \Rightarrow B}{\Gamma \Rightarrow A\backslash B}$$

$$(/L)\ \frac{\mathit{\Gamma}[A] \Rightarrow C; \quad \mathit{\Delta} \Rightarrow B}{\mathit{\Gamma}[(A/B) \circ \mathit{\Delta}] \Rightarrow C} \quad (/R)\ \frac{\mathit{\Gamma} \circ B \Rightarrow A}{\mathit{\Gamma} \Rightarrow A/B}$$

$$(\cdot L)\ \frac{\mathit{\Gamma}[A \circ B] \Rightarrow C}{\mathit{\Gamma}[A \cdot B] \Rightarrow C} \quad (\cdot R)\ \frac{\mathit{\Gamma} \Rightarrow A; \quad \mathit{\Delta} \Rightarrow B}{\mathit{\Gamma} \circ \mathit{\Delta} \Rightarrow A \cdot B}$$

$$(\mathrm{CUT})\ \frac{\mathit{\Delta} \Rightarrow A; \quad \mathit{\Gamma}[A] \Rightarrow B}{\mathit{\Gamma}[\mathit{\Delta}] \Rightarrow B}$$

$$(\Diamond L)\ \frac{\mathit{\Gamma}[\langle A \rangle] \Rightarrow B}{\mathit{\Gamma}[\Diamond A] \Rightarrow B} \quad (\Diamond R)\ \frac{\mathit{\Gamma} \Rightarrow A}{\langle \mathit{\Gamma} \rangle \Rightarrow \Diamond A}$$

$$(\Box^{\downarrow} L)\ \frac{\mathit{\Gamma}[A] \Rightarrow B}{\mathit{\Gamma}[\langle \Box^{\downarrow} A \rangle] \Rightarrow B} \quad (\Box^{\downarrow} R)\ \frac{\langle \mathit{\Gamma} \rangle \Rightarrow A}{\mathit{\Gamma} \Rightarrow \Box^{\downarrow} A}$$

($\Diamond$L), ($\Diamond$R), ($\Box^{\downarrow}$L) and ($\Box^{\downarrow}$R) are rules for unary modalities. Distributive Full Nonassociative Lambek Calculus enriched with unary modalities DFNL$\Diamond$ employs operations $\cdot$, $\backslash$, $/$, $\Diamond$, $\Box^{\downarrow}$, $\wedge$ and $\vee$. One admits additional rules

$$(\wedge L)\ \frac{\mathit{\Gamma}[A_i] \Rightarrow B}{\mathit{\Gamma}[A_1 \wedge A_2] \Rightarrow B} \quad (\wedge R)\ \frac{\mathit{\Gamma} \Rightarrow A \quad \mathit{\Gamma} \Rightarrow B}{\mathit{\Gamma} \Rightarrow A \wedge B}$$

$$(\vee L)\ \frac{\mathit{\Gamma}[A_1] \Rightarrow B \quad \mathit{\Gamma}[A_2] \Rightarrow B}{\mathit{\Gamma}[A_1 \vee A_2] \Rightarrow B} \quad (\vee R)\ \frac{\mathit{\Gamma} \Rightarrow A_i}{\mathit{\Gamma} \Rightarrow A_1 \vee A_2}$$

and the distributive axiom

$$(\mathrm{D}) \quad A \wedge (B \vee C) \Rightarrow (A \wedge B) \vee (A \wedge C).$$

In ($\wedge$L) and ($\vee$R), the subscript equals 1 or 2. Notice that the converse sequent of (D) is provable in DFNL$\Diamond$.

By DFNL$_{\mathrm{S4}}$, we denote the system DFNL$\Diamond$ enriched with the following structural rules (4) and (T) (corresponding to axioms 4, T)

$$(\mathrm{T})\ \frac{\mathit{\Gamma}[\langle \mathit{\Delta} \rangle] \Rightarrow A}{\mathit{\Gamma}[\mathit{\Delta}] \Rightarrow A} \quad (4)\ \frac{\mathit{\Gamma}[\langle \mathit{\Delta} \rangle] \Rightarrow A}{\mathit{\Gamma}[\langle\langle \mathit{\Delta} \rangle\rangle] \Rightarrow A}$$

(CUT) can be eliminated from NL$_{\mathrm{S4}}$ (see [15]), but not for DFNL$\Diamond$ and DFNL$_{\mathrm{S4}}$. Notice that $\Diamond$ can be treated as usual modal operation $\Diamond$ in classical modal logic. $\Box^{\downarrow}$ is the residuated operations of $\Diamond$, which is different from $\Box$ in classical modal logic. Hence, in the present paper, we call this system DNFL$_{\mathrm{S4}}$ following A. R. Plummer.

By $F(\mathit{\Gamma})$, we denote the formula arising from $\mathit{\Gamma}$ by dropping all binary and unary operators $\circ$ and $\langle\rangle$, respectively, and the corresponding parentheses (). $\mathit{\Gamma} \Rightarrow A$ and $F(\mathit{\Gamma}) \Rightarrow A$ are mutually deducible in DFNL$_{\mathrm{S4}}$. Let $\mathit{\Phi}$ be a set of sequents of the form $A \Rightarrow B$. Hereafter, we assume that $\mathit{\Phi}$ is finite. By DNFL$_{\mathrm{S4}}(\Phi)$, one denotes DNFL$_{\mathrm{S4}}$ enriched with finitely many assumptions $\mathit{\Phi}$. We write $\mathit{\Phi} \vdash_{\mathrm{DNFL_{S4}}} \mathit{\Gamma} \Rightarrow A$ if $\mathit{\Gamma} \Rightarrow A$ is derivable from $\mathit{\Phi}$ in DNFL$_{\mathrm{S4}}$ (provable in DNFL$_{\mathrm{S4}}(\Phi)$).

A $\mathrm{DNFL_{S4}}(\Phi)$-grammar over an alphabet Σ is a pair $\langle \mathcal{L}, \mathcal{D}\rangle$, where $\mathcal{L}$, the lexicon, is a finite relation between strings from Σ^+ and formulae of $\mathrm{DNFL_{S4}}(\Phi)$, and $\mathcal{D}$ is a finite set of designated formulae (types). A language $L(G)$ generated by a $\mathrm{DNFL_{S4}}(\Phi)$-grammar $G = \langle \mathcal{L}, \mathcal{D}\rangle$ is defined as a set of strings $a_1 \cdots a_n$, where $a_i \in \Sigma^+$, $1 \leq i \leq n$, and $n \geq 1$, satisfying the following condition: there exist formulae $A_1, \ldots, A_n$, S, and a formula structure Γ such that for all $1 \leq i \leq n$ $\langle a_i, A_i\rangle \in \mathcal{L}$, $S \in \mathcal{D}$, and $\Phi \vdash_{\mathrm{DNFL_{S4}}} \Gamma \Rightarrow S$, where $F(\Gamma) = A_1 \cdots A_n$.

Here below, we prove that every language recognized by a $\mathrm{DFNL_{S4}}$ (Φ)-grammar can also be generated by a context-free grammar, by showing that every derivable sequent in $\mathrm{DNFL_{S4}}(\Phi)$ can be derived (by means of (CUT) only) from some short derivable sequents containing at most three formulae, where all of the formulae belong to a finite set of formulae. This follows from the fact that there exists a finite set of formulae such that each structure Δ in a derivable sequent $\Gamma[\Delta] \Rightarrow A$ has an interpolant (formula) that belongs to this set (cf. Lemma 9).

By a T-sequent we mean a sequent such that all formulae occurring in it belong to T. We write $\Phi \vdash_S \Gamma \Rightarrow_T A$ if $\Gamma \Rightarrow A$ has a deduction from Φ (in the given calculus S), which consists of T-sequents only (called a T-deduction). Two formulae A and B are said to be T-equivalent in a calculus S, if and only if $\vdash_S A \Rightarrow_T B$ and $\vdash_S B \Rightarrow_T A$. T-equivalence is a equivalence relation, by (Id) and (CUT).

Lemma 1. *Let T be a set of formulae closed under $\vee$ and $\wedge$. If $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle\Delta\rangle] \Rightarrow_T A$, then there exists a $D \in T$ such that $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T D$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D\rangle \Rightarrow_T D$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D] \Rightarrow_T A$.*

Proof: D is called an interpolant of $\langle\Delta\rangle$ in $\Gamma[\langle\Delta\rangle] \Rightarrow A$. The proof proceeds by induction on T-deduction of $\Gamma[\langle\Delta\rangle] \Rightarrow A$. The cases of axiom and assumptions are trivial, since there is no structure of the from $\langle\Delta\rangle$ in an axiom or assumption. Let $\Gamma[\langle\Delta\rangle] \Rightarrow A$ be the conclusion of the rule R. (CUT) is easy. Let us consider other rules.

Firstly, we assume that $\langle\Delta\rangle$ does not contain the formula or structure operation, introduced by R (active formula and active structure operation). Let R = ($\wedge$R). Assume the premise are $\Gamma[\langle\Delta\rangle] \Rightarrow A_1$, $\Gamma[\langle\Delta\rangle] \Rightarrow A_2$ and the conclusion is $\Gamma[\langle\Delta\rangle] \Rightarrow A_1 \wedge A_2$. By induction hypothesis, there are interpolants D_1, D_2 such that $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T D_1$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_1\rangle \Rightarrow_T D_1$ $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D_1] \Rightarrow_T A_1$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T D_2$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_2\rangle \Rightarrow_T D_2$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D_2] \Rightarrow_T A_2$. Then, one gets $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T D_1 \wedge D_2$ by ($\wedge$R). By ($\wedge$L) and ($\wedge$R), we obtain $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_1 \wedge D_2\rangle \Rightarrow_T D_1 \wedge D_2$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D_1 \wedge D_2] \Rightarrow_T A_1 \wedge A_2$. The case of ($\vee$L) can be treated similarly. For other cases, $\langle\Delta\rangle$ must come exactly from one premise of R. One takes an interpolant from this premise, then the thesis follows directly from the induction hypothesis.

Secondly, we assume that $\langle\Delta\rangle$ contains the active formula or the active structure operation (So the rule must be an L-rule or $(\Diamond R)$). For $(\Diamond R)$ with the premise $\Delta \Rightarrow A$ and the conclusion $\langle\Delta\rangle \Rightarrow \Diamond A$. Let the interpolant be $\Diamond A$. By (Id), $(\Diamond R)$, (4) and $(\Diamond L)$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Diamond A\rangle \Rightarrow_T \Diamond A$. This yields that $\Phi \vdash_{\mathrm{DFNL_{S4}}}$

$\langle\Delta\rangle \Rightarrow_T D$ ($\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T \Diamond A$), $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D\rangle \Rightarrow_T D$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D\rangle \Rightarrow_T \Diamond A$ ($\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Diamond A\rangle \Rightarrow_T \Diamond A$). For ($\Box^{\downarrow}L$) with premise $\Gamma[C] \Rightarrow A$ and conclusion $\Gamma[\langle\Box^{\downarrow}C\rangle] \Rightarrow A$, where $\langle\Delta\rangle = \langle\Box^{\downarrow}C\rangle$. It's similar to cases ($\Diamond R$). One takes $\Box^{\downarrow}C$ as the interpolant. Then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Box^{\downarrow}C\rangle \Rightarrow_T \Box^{\downarrow}C$, by (Id), ($\Box^{\downarrow}L$), (4) and ($\Box^{\downarrow}R$). For R=(4), assume $\langle\Delta\rangle$ arises from structure Δ^* by rule R. It follows that the interpolant of Δ^* can also be the interpolant of $\langle\Delta\rangle$, by induction hypothesis and R. Similarly, if R is rule ($\cdot L$), ($\wedge L$), ($\backslash L$) or ($/L$) and $\langle\Delta^*\rangle$ be the source of $\langle\Delta\rangle$, which means that $\langle\Delta\rangle$ arises from $\langle\Delta^*\rangle$ by rule R, then an interpolant of $\langle\Delta^*\rangle$ is also an interpolant of $\langle\Delta\rangle$. Let us consider the final case $R = (\vee L)$. Assume the premise are $\Gamma[\langle\Delta^*[B_1]\rangle] \Rightarrow A$, $\Gamma[\langle\Delta^*[B_2]\rangle] \Rightarrow A$ and the conclusion is $\Gamma[\langle\Delta^*[B_1 \vee B_2]\rangle] \Rightarrow A$, where $\langle\Delta\rangle = \langle\Delta^*[B_1 \vee B_2]\rangle$. Let D_1 be an interpolant of $\langle\Delta^*[B_1]\rangle$ in the first premise and D_2 be an interpolant of $\langle\Delta^*[B_2]\rangle$ in the second premise. By induction hypothesis, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_1\rangle \Rightarrow_T D_1$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_2\rangle \Rightarrow_T D_2$. By ($\vee R$) and ($\vee L$) $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle D_1 \vee D_2\rangle \Rightarrow_T D_1 \vee D_2$. Hence, $D_1 \vee D_2$ is an interpolant of $\langle\Delta\rangle$ in the conclusion, by induction hypothesis, ($\vee R$) and ($\vee L$). □

Lemma 2. *Let T be a set of formulae closed under $\vee$, $\wedge$. If $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Delta] \Rightarrow_T A$ then there exists a $D \in T$ such that $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_T D$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D] \Rightarrow_T A$.*

Proof: If Δ is a single formula then we take $D = \Delta$. Assume Δ is not a single formula. If Δ is of the form $\langle\Delta^*\rangle$ for some structure Δ^* then the thesis follows from lemma 1. Otherwise, the proof is similar to the proof of lemma 1 without considering the additional conditions. □

The following lemma is an analogue of lemma 2 in Buszkowski [6]

Lemma 3. *If T is a set of formulae generated from a finite set and closed under $\wedge$, $\vee$, then T is finite up to the relation of T-equivalence in* $\mathrm{DFNL_{S4}}$.

Proof: Since $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ is provable in $\mathrm{DNFL_{S4}}$, $A \wedge (B \vee C)$ and $(A \wedge B) \vee (A \wedge C)$ are T-equivalent, for any $A, B, C \in T$. Consequently, every formula from T is T-equivalent to finite disjunction of finite conjunctions of formulae from T. There are only finitely many formulae of latter form, if one omits repetitions. □

One can obtain an analogous interpolation lemma for the system with an additional axiom K by proving a variant of lemma 1 (replacing $\langle\Delta\rangle$ by $\langle\Delta_1\rangle \circ \ldots \circ \langle\Delta_n\rangle$, where $n \geq 1$). Similarly, let us consider the system enriched with additional axioms K_1, K_2. Using the above strategy, one can prove another variant of lemma 1 (replacing $\langle\Delta\rangle$ by $\Lambda \circ \langle\Delta\rangle$ and $\langle\Delta\rangle \circ \Lambda$, where Λ can be empty), which yields the interpolation lemma for this system. If we consider some systems enriched with distributive lattice and modalities satisfying some additional axioms (e.g. 4, T here) then we cannot prove lemma 2 by the strategies in [6], [20] and [14]. Notice that if one assumes that T is also closed under $\Diamond$ in Lemma 2, like in [20] and [14], then T is not finite up to the relation of T-equivalence in $\mathrm{DFNL_{S4}}$, which yields the failure of lemma 3. Hence lemma 1 is essential in the proof.

A distributive lattice-ordered residuated groupoid with S4-operators (S4-DLRG) is a structure $(G, \wedge, \vee, \cdot, \backslash, /, \Diamond, \Box^{\downarrow})$ such that $(G, \wedge, \vee)$ is a distributive lattice and $(G, \cdot, \backslash, /, \Diamond, \Box^{\downarrow})$ is a structure, where $\cdot, \backslash, /$ and $\Diamond$, $\Box^{\downarrow}$ are binary and unary operations on G, respectively, satisfying the following conditions:

$$a \cdot b \leq c \quad \text{iff} \quad b \leq a\backslash c \quad \text{iff} \quad a \leq c/b \tag{1}$$

$$\Diamond a \leq b \quad \text{iff} \quad a \leq \Box^{\downarrow} b \tag{2}$$

$$4: \quad \Diamond\Diamond a \leq \Diamond a \qquad \mathrm{T}: \quad a \leq \Diamond a \tag{3}$$

for all $a, b, c \in G$, where $\leq$ is the lattice ordering. It is easy to prove that $\mathrm{DFNL_{S4}}$ is strongly complete with respect to S4-DLRGs. Besides by S4-LRG, we denote a lattice-ordered residuated groupoid with S4-operators. Let $\mathcal{G}$ be a S4-DLRG. We recall some basic notions. A valuation μ in $\mathcal{G}$ is a homomorphism from the formula algebra into $\mathcal{G}$. A sequent $\Gamma \Rightarrow A$ is true in the model $(\mathcal{G}, \mu)$, if $\mu(\Gamma) \leq \mu(A)$. The strong completeness means the following: $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow A$ if and only if, for any model $(\mathcal{G}, \mu)$. if all sequents from Φ are true, then $\Gamma \Rightarrow A$ is true.

By $\mathcal{G}$ and G, we denote a groupoid and its universe, respectively. Let $\mathcal{G}$ =(G, $\cdot$, $\diamond$) be a groupoid, where $\diamond$ is a unary operation on G. On the powerset $P(G)$, one defines operations: $U \odot V = \{a \cdot b \in G : a \in U, b \in V\}$, $\Diamond U = \{\diamond a \in G : a \in U\}$, $U\backslash V = \{a \in G : U \odot \{a\} \subseteq V\}$, $V/U = \{a \in G; \{a\} \odot U \subseteq V\}$, $\Box^{\downarrow} U\{a \in G : \diamond a \in U\}$, $U \vee V = U \cup V$, $U \wedge V = U \cap V$. $P(G)$ with operations $\odot$, $\Diamond$, $\backslash$, $/$, $\Box^{\downarrow}$, $\vee$ and $\wedge$ is a distributive lattice-ordered residuated groupoid satisfying (1) and (2) (it is a complete lattice). The order is $\subseteq$. An operator C : $P(G) \rightarrow P(G)$ is called a S4-closure operator (or: a S4-nucleus) on $\mathcal{G}$, if it satisfies the following conditions: (C1) $U \subseteq C(U)$, (C2) if $U \subseteq V$ then $C(U) \subseteq C(V)$, (C3) $C(C(U)) \subseteq C(U)$, (C4) $C(U) \odot C(V) \subseteq C(U \odot V)$, (C5) $\Diamond C(U) \subseteq C(\Diamond U)$, (C6) $C(\Diamond C(\Diamond C(U))) \subseteq C(\Diamond U)$, (C7) $C(U) \subseteq C(\Diamond U)$. For $U \subseteq P(G)$, U is called $C - \mathit{closed}$ if $U = C(U)$. By $C(G)$, we denote the family of $C - \mathit{closed}$ subsets of G. Let $U \otimes V = C(U \odot V)$, $\blacklozenge U = C(\Diamond U)$, $U \vee_C V = C(U \vee V)$, and $\backslash$, $/$, $\Box^{\downarrow}$, $\wedge$, be defined as above. By (C1)-(C5), $C(\mathcal{G}) = (C(G), \wedge, \vee_C, \otimes, \backslash, /, \blacklozenge, \Box^{\downarrow})$ is a complete lattice-ordered residuated groupoid [5], it need not be distributive. The order is $\subseteq$. Using (C6)-(C7), it is easy to prove $\blacklozenge\blacklozenge U \subseteq \blacklozenge U$, $U \subseteq \blacklozenge U$. It follows that $C(\mathcal{G})$ is a S4-LRG.

Let T be a nonempty set of formulae. By T^*, we denote the set of all formula structures formed out of formulae in T. Similarly, $T^*[\circ]$ denotes the set of all contexts in which all formulae belong to T. $\mathcal{G}(\mathcal{T}^*) = (T^*, \circ, \langle\rangle)$ is a groupoid such that $\langle\rangle$ is a unary operation on T^*. Let $\Gamma[\circ] \in T^*[\circ]$ and $A \in T$, we define:

$$[\Gamma[\circ], A] = \{\Delta | \Delta \in T^* \quad \text{and} \quad \Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Delta] \Rightarrow_T A\}$$

We define $B(T)$ as the family of all sets $[\Gamma[\circ], A]$, defined above. One defines C_T as follows:

$$C_T(U) = \bigcap \{[\Gamma[\circ], A] \in B(T) : U \subseteq [\Gamma[\circ], A]\}$$

We prove the following proposition.

Proposition 1. *C_T is a* S4 *closure operator.*

Proof: It is easy to see that C_T satisfies (C1),(C2),(C3). For (C4), (C5) one may see [6] and [5]. We prove that C_T satisfies (C6)-(C7). Firstly, for (C6), let $U \subseteq T^*$ and $\Delta \in C_T(U)$. We show $\langle\langle\Delta\rangle\rangle \in C_T(\Diamond U)$. Let $[\Gamma[\circ], A] \in B(T)$ be such that $\Diamond U \subseteq [\Gamma[\circ], A]$. For any $\langle\Pi\rangle \in \Diamond U$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle\Pi\rangle] \Rightarrow_T A$, whence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle\langle\Pi\rangle\rangle] \Rightarrow_T A$, by (4). Consequently, $U \subseteq [\Gamma[\langle\langle\circ\rangle\rangle], A]$. It follows that $C_T(U) \subseteq [\Gamma[\langle\langle\circ\rangle\rangle], A]$, by the definition of C_T. Consequently, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle\langle\Delta\rangle\rangle] \Rightarrow_T A$, whence $\langle\langle\Delta\rangle\rangle \in [\Gamma[\circ], A]$. This yields that $\langle\langle\Delta\rangle\rangle \in C_T(\Diamond U)$. Then, $\Diamond\Diamond C_T(U) \subseteq C_T(\Diamond U)$. By (C2) and (C3), $C_T(\Diamond\Diamond C_T(U)) \subseteq C_T(\Diamond U)$. By (C5), $\Diamond C_T(\Diamond C_T(U)) \subseteq C_T(\Diamond\Diamond C_T(U))$, whence $\Diamond C_T(\Diamond C_T(U)) \subseteq C_T(\Diamond U)$. It follows that (C6) stands, by (C2) and (C3).

Secondly, let us consider the case (C7). Assume $U \subseteq T^*$ and $\Delta \in C_T(U)$. For any $[\Gamma[\circ], A] \in B(T)$ satisfies $\Diamond U \subseteq [\Gamma[\circ], A]$. Let $\langle\Pi\rangle \in \Diamond U$, then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle\Pi\rangle] \Rightarrow_T A$, whence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Pi] \Rightarrow_T A$, by (T). So, $C_T(U) \subseteq [\Gamma[\circ], A]$. Hence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Delta] \Rightarrow_T A$. Consequently, $\Delta \in C_T(\Diamond U)$, which follows (C7) stands. □

Accordingly, $C_T(\mathcal{G}(\mathcal{T}^*))$ is a S4-LRG. We define:

$$[A] = [\circ, A] = \{\Gamma | \Gamma \in T^* \quad \text{and} \quad \Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow_T A\}$$

for $A \in T$ and $[A] \in B(T)$. The following equations are true in $C_T(\mathcal{G}(\mathcal{T}^*))$ provided that all formulae appearing in them belong to T.

$$[A] \otimes [B] = [A \cdot B], \quad [A]\backslash[B] = [A\backslash B], \quad [A]/[B] = [A/B] \tag{4}$$

$$\blacklozenge[A] = [\Diamond A], \quad \Box^{\downarrow}[A] = [\Box^{\downarrow} A] \tag{5}$$

$$[A] \cap [B] = [A \wedge B], \quad [A] \vee_C [B] = [A \vee B] \tag{6}$$

Such equations are also discussed in [6], [5], [1]. For the completeness of the proof, we show parts of the proof here. We prove the equation (5) and (6). We show the first equation (5). Let $\Gamma \in \Diamond[A]$, whence $\Gamma = \langle\Delta\rangle$ for some $\Delta \in [A]$. Hence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_T A$. By $(\Diamond R)$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \langle\Delta\rangle \Rightarrow_T \Diamond A$. Consequently, $\Diamond[A] \subseteq [\Diamond A]$. By (C2) and (C3), $\blacklozenge[A] = C(\Diamond[A]) \subseteq C([\Diamond A])$. We prove the converse inclusion. Let $\Diamond[A] \subseteq [\Gamma[\circ], C]$, then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\langle A\rangle] \Rightarrow_T C$. Hence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Diamond A] \Rightarrow_T C$. Consequently, $[\Diamond A] \subseteq [\Gamma[\circ], C]$. By the definition of C_T, $[\Diamond A] \subseteq C_T(\Diamond[A])$. The proof of the second equation (5) is similar.

We prove the second equation (6). Assume $\Delta \in [A]$. Then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_T A$, whence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_T A \vee B$, by$(\vee R)$. Consequently, $[A] \subseteq [A \vee B]$. Similarly, $[B] \subseteq [A \vee B]$, whence $[A] \cup [B] \subseteq [A \vee B]$. By (C2) and (C3), $[A] \vee_C [B] = C_T([A] \cup [B]) \subseteq [A \vee B]$. For the converse inclusion, let $[A] \cup [B] \subseteq [\Gamma[\circ], C]$, then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[A] \Rightarrow_T C$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[B] \Rightarrow_T C$. By $(\vee L)$, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[A \vee B] \Rightarrow_T C$, which yields $[A \vee B] \subseteq [A] \vee_C [B]$. The proof of the first equation (6) is similar.

Let T be a finite nonempty set of formulae. By $\overline{T}$, one denotes the smallest set of formulae containing all formulae from T and closed under subformulae and $\wedge$, $\vee$. By lemma 3, $\overline{T}$ is finite up to $\overline{T}$-equivalence. Let R be a selector of the family of equivalence classes of $\overline{T}$-equivalence. R chooses one formula from each equivalence class. If there are some formulae of T contained in an equivalence class then R chooses one formula from these formulae. Otherwise, R chooses an arbitrary formula from an equivalence class. Clearly R is finite. Hereafter, by $r(T)$, we denote the set R generated from set T. Notice that $r(r(T)) = r(T)$, due to lemma 3. Lemma 4 to lemma 9 is analogous to corresponding lemmas from [6], [5] for DFNL and DGL. Here we only provide the proof of lemma 4 and lemma 6 in details, which are the key arguments of the proof of the extended subformula property of $\mathrm{DFNL_{S4}}$. We skip the proofs for others. Lemma 4 is essential for proving that $C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$ is finite and satisfies distributive axiom in lemma 5. Corollary 7 directly follows from lemma 6 and lemma 3.

Lemma 4. *For any nontrivial set $U \in C_T(G(\overline{T}^*))$, there exists a formula $A \in r(\overline{T})$ such that $U = [A]$.*

Proof: For any $U \in C_T(G(\overline{T}^*))$, assume $U \subseteq [\Gamma[\circ], A]$. Let $\Delta \in U$, then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[\Delta] \Rightarrow_{\overline{T}} A$. By lemma 2, there exists $D \in \overline{T}$ such that $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma[D] \Rightarrow_{\overline{T}} A$ and $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_{\overline{T}} D$. By lemma 3, D can be replaced by a $\overline{T}$-equivalent formula in $r(\overline{T})$. Assume $D \in r(\overline{T})$. For each $\Delta \in U$, one obtains a $D_i \in r(\overline{T})$ fulfilling above. Let $u(D)$ be the $\overline{T}$-equivalent formula of the disjunction of all D_i $(1 \leq i \leq n)$. Clearly, $U \subseteq [u(D)]$ and $[u(D)] \subseteq [\Gamma[\circ], A]$, by $(\vee R)$ and $(\vee L)$. For each $[\Gamma[\circ], A]$ satisfying $U \subseteq [\Gamma[\circ], A]$, one takes a $u(D_i) \in r(\overline{T})$ fulfilling above for $1 \leq i \leq m$. Let $j(D)$ be the $\overline{T}$-equivalent formula of the conjunction of all $u(D_i)$. By (6), $U \subseteq [u(D_1) \cap \ldots \cap u(D_m)] = [j(D)]$. Since for each $[\Gamma[\circ], A]$ satisfying $U \subseteq [\Gamma[\circ], A]$, there exists a $u(D_i) \subseteq [\Gamma[\circ], A]$, for $1 \leq i \leq m$. Therefore, by the definition of C_T, $j(D) \subseteq U$, which yields $U = [j(D)]$ and $j(D) \in r(\overline{T})$. □

Lemma 5. *$C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$ is a finite* $\mathrm{S4} - \mathrm{DLRG}$.

Lemma 6. *T denotes a finite set of formulae, containing all formulae in Φ. Let μ be a valuation in $C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$ such that $\mu(p) = [p]$. For any $\overline{T}$-sequent $\Gamma \Rightarrow A$, this sequent is true in $(C_T(\mathcal{G}(\overline{\mathcal{T}}^*)), \mu)$ if and only if $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow_{\overline{T}} A$.*

Proof: Assume a $\overline{T}$-sequent $\Gamma \Rightarrow A$ be true in $(C_T(\mathcal{G}(\overline{\mathcal{T}}^*)), \mu)$. Then $\mu(\Gamma) \subseteq \mu(A)$. Since $\Gamma \in \mu(\Gamma)$, we get $\Gamma \in \mu(A) = [A]$. Hence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow_{\overline{T}} A$. Assume $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow_{\overline{T}} A$. We prove that $\Gamma \Rightarrow A$ is true in $(C_T(\mathcal{G}(\overline{\mathcal{T}}^*)), \mu)$, by induction on $\overline{T}$-deductions. The axioms (Id), (D) and the assumptions from Φ, restricted to $\overline{T}$-sequents, are of the form $E \Rightarrow F$. By (4), (5) and (6) , we get $\mu(E) = [E]$ and $\mu(F) = [F]$. Assume $\Delta \in [E]$, we get $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_{\overline{T}} E$. Hence $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Delta \Rightarrow_{\overline{T}} F$, by (CUT), which yields $[E] \subseteq [F]$. By lemma 5, $C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$ is a finite S4-DLRG, all rules of $\mathrm{DFNL_{S4}}$ preserve the truth in $(C_T(\mathcal{G}(\overline{\mathcal{T}}^*)), \mu)$, whence $\Gamma \Rightarrow A$ is true in $C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$. □

Corollary 7. *Let T be a finite set of formulas, containing all formulae appearing in $\Gamma \Rightarrow A$ and Φ. If $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow A$ then $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow_{r(\overline{T})} A$.*

Now, we can prove the strong finite model property of $\mathrm{DFNL_{S4}}$

Theorem 8. *Assume that $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma \Rightarrow A$ does not hold. Then there exist a finite distributive lattice ordered residuated groupoid with* S4*-operators $\mathcal{G}$ and a valuation μ such that all sequents from Φ are true but $\varGamma \Rightarrow A$ is not true in $(\mathcal{G}, \mu)$.*

Proof: Let T be the set of all formulas appearing in $\varPhi$ and $\varGamma \Rightarrow A$. Hence $C_T(\mathcal{G}(\overline{\mathcal{T}}^*))$ is a finite $\mathrm{S4} - \mathrm{DLRG}$, by Lemma 5. Assume $\varPhi \nvdash_{\mathrm{DFNL_{S4}}} \varGamma \Rightarrow A$, which yields $\varPhi \nvdash_{\mathrm{DFNL_{S4}}} \varGamma \Rightarrow_{\overline{T}} A$. Let $\mu(p) = [p]$. By lemma 6, all sequents from $\varPhi$ are true in $(C_T(\mathcal{G}(\overline{\mathcal{T}}^*)), \mu)$ but $\varGamma \Rightarrow A$ is not true. □

Lemma 9. *Let T be a finite set of formulas, containing all formulae appearing in $\varGamma[\varDelta] \Rightarrow A$ and $\varPhi$. If $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma[\varDelta] \Rightarrow A$, then there exists a $D \in r(\overline{T})$ such that $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varDelta \Rightarrow_{r(\overline{T})} D$ and $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma[D] \Rightarrow_{r(\overline{T})} A$.*

Proof: Assume $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma[\varDelta] \Rightarrow A$. By corollary 7, $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma[\varDelta] \Rightarrow_{r(\overline{T})} A$. Then, we apply lemma 2. □

Let T be a finite nonempty set of formulae. Let $S^T(\varPhi)$ be the following system. The set of axioms of $S^T(\varPhi)$ consists of all the sequents $A \circ B \Rightarrow C$, $\langle A \rangle \Rightarrow B$ and $A \Rightarrow B$ derivable from $\varPhi$ in $\mathrm{DFNL_{S4}}$, where $A, B, C \in T$. The only rule of $S^T(\varPhi)$ is (CUT).

Theorem 10. *Let T be a finite set of formulae, containing all formulae appearing in $\varGamma \Rightarrow A$ and $\varPhi$. $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma \Rightarrow A$ iff $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma \Rightarrow A$.*

Proof: The *if* part is easy. The proof of the *only if* part proceeds by induction on the lengths of $\varGamma$, denoted by $l(\varGamma)$ (the number of structure operations $\circ$ appearing in $\varGamma$). If $l(\varGamma)$ equals 0 or 1, then $\varGamma$ contains only one or two formulae. Hence $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma \Rightarrow A$, by the consturction of $S^{r(\overline{T})}(\varPhi)$. Similarly, if $\varGamma = \langle B \rangle$, then by the consturction of $S^{r(\overline{T})}(\varPhi)$, $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma \Rightarrow A$. Assume $\varGamma = \varXi[\varGamma_1 \circ \varGamma_2]$. By Lemma 9, there exist formulae $D_1, D_2 \in r(\overline{T})$ such that $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma_1 \Rightarrow_{r(\overline{T})} D_1$, $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varGamma_2 \Rightarrow_{r(\overline{T})} D_2$ and $\varPhi \vdash_{\mathrm{DFNL_{S4}}} \varXi[D_1 \circ D_2] \Rightarrow_{r(\overline{T})} A$. Since $r(r(\overline{T})) = r(\overline{T})$, by induction hypothesis, one gets $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma_1 \Rightarrow D_1$, $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma_2 \Rightarrow D_2$, and $\vdash_{S^{r(\overline{T})}(\varPhi)} \varXi[D_1 \circ D_2] \Rightarrow A$. It follows that $\vdash_{S^{r(\overline{T})}(\varPhi)} \varGamma \Rightarrow A$, by (CUT). □

Theorem 11. *Every language generated by a* $\mathrm{DFNL_{S4}}(\Phi)$*-grammar is context-free.*

Proof: Let $\varPhi$ be a finite set of sequents of the form $A \Rightarrow B$, $G_1 = \langle \mathcal{L}, \mathcal{D} \rangle$ be a $\mathrm{DFNL_{S4}}(\Phi)$-grammar, and T be the set of all subformulae of formulae appearing in $\mathcal{D}$, $\mathcal{L}$ and $\varPhi$. We construct $r(\overline{T})$ and $S^{r(\overline{T})}(\varPhi)$ as above. Now we construct an equivalent *CFG* (context-free grammar) G_2, in the following way. The terminal elements of G_2 are lexical items of G_1. The non-terminals are all formulae from $r(\overline{T})$ and a fresh non-terminal S. Productions are $\{A \to B \mid\ \vdash_{S^{r(\overline{T})}(\varPhi)} B \Rightarrow A\} \cup \{A \to B \mid\ \vdash_{S^{r(\overline{T})}(\varPhi)} \langle B \rangle \Rightarrow A\} \cup \{A \to B,\, C \mid\ \vdash_{S^{r(\overline{T})}(\varPhi)} B \circ C \Rightarrow A\} \cup \{A \to v \mid \langle v,\, A \rangle \in L\} \cup \{S \to A \mid A \in D\}$.

If $v_1 \dots v_m$ is generated by G_1, then there is a sequent $\varGamma \Rightarrow B$ derivable from $\varPhi$ in $\mathrm{DFNL_{S4}}$, where B is a designated type, $F(\varGamma) = A_1 \cdots A_m$, and $\langle v_i,\, A_i \rangle \in \mathcal{L}$

for $1 \leq i \leq m$. We get $S \rightarrow^*_{G_2} B$ by the construction of G_2. Due to theorem 10 and the construction of G_2, we obtain $S \rightarrow^*_{G_2} A_1 \cdots A_m$ which leads to $S \rightarrow^*_{G_2} v_1 \ldots v_m$. Hence $v_1 \cdots v_m$ is generated by G_2.

Now suppose $v_1 \cdots v_m$ is generated by G_2, which means $S \rightarrow^*_{G_2} v_1 \cdots v_n$. Then, there exists a $B \in \mathcal{D}$ such that $B \rightarrow^*_{G_2} A_1 \cdots A_n$, where $\langle v_i, A_i \rangle \in \mathcal{L}$, $1 \leq i \leq m$. Hence, by the construction of G_2, there exists a formula structure Γ such that $F(\Gamma) = A_1 \cdots A_n$ and $\vdash_{S^{r(\overline{T})}(\Phi)} \Gamma \Rightarrow B$. By theorem 10, $\Phi \vdash_{\mathrm{DFNL_{S4}}} \Gamma \Rightarrow B$. Therefore, $v_1 \cdots v_m$ is generated by G_1. □

Obviously, we can easily obtain the same results for systems DFNL$\Diamond$, DFNL$_4$, and DFNL$_\mathrm{T}$. The inclusion of the class of ε-free context free languages in the class of DFNL$_{\mathrm{S4}}(\Phi)$-recognizable languages can be easily established. Every context-free language is generated by some NL-grammars (see [11]). Since neither the *lexicon* nor *designated formulae* contain modal operators and additives, these NL-grammars can be conceived of as DFNL$_{\mathrm{S4}}(\Phi)$-grammars, where Φ is empty. Hence DFNL$_{\mathrm{S4}}(\Phi)$-grammars generate exactly the ε-free context-free languages.

Acknowledgements. I would like to thank Prof. Wojciech Buszkowski for the patient supervision of my research and valuable suggestions.

References

1. Belardinelli, F., Jipsen, P., Ono, H.: Algebraic aspects of cut elimination. Studia Logica 77, 209–240 (2004)
2. Bernardi, R.: Reasoning with Polarity in Categorial Type Logic. PhD thesis, Utrecht (2002)
3. Buszkowski, W.: Generative Capacity of Nonassociative Lambek Calculus. Bulletin of Polish Academy of Sciences: Math 34, 507–516 (1986)
4. Buszkowski, W.: Lambek Calculus with Nonlogical Axioms. In: Casadio, C., Scott, P., Seely, R. (eds.) Languages and Grammars Studies in Mathematical Linguistics and Natural Language. CSLI Lectures Notes, vol. 168, pp. 77–93 (2005)
5. Buszkowski, W.: Interpolation and FEP for logic of residuated algebras. Logic Journal of the IGPL 19(3), 437–454 (2011)
6. Buszkowski, W., Farulewski, M.: Nonassociative Lambek Calculus with Additives and Context-Free Languages. In: Grumberg, O., Kaminski, M., Katz, S., Wintner, S. (eds.) Francez Festschrift. LNCS, vol. 5533, pp. 45–58. Springer, Heidelberg (2009)
7. Heylen, D.: Types and Sorts Resource Logic for Feature Cheking. PhD thesis, Utrecht (1999)
8. Jäger, G.: On the generative capacity of multi-modal categorial grammars. Research on Language and Computation 1, 105–125 (2003)
9. Jäger, G.: Residuation, Structural Rules and Context Freeness. Journal of Logic, Language and Informationn 13, 47–59 (2004)
10. Kanazawa, M.: The Lambek Calculus enriched with Additional Connectives. Journal of Logic, Language and Information 1(2), 141–171 (1992)
11. Kandulski, M.: The equivalence of Nonassociative Lambek Categorial Grammars and Context-free Grammars. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 52, 34–41 (1988)

12. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–170 (1958)
13. Lambek, J.: On the calculus of syntactic types. In: Structure of Language and Its Mathematical Aspects, pp. 168–178. American Mathematical Society (1961)
14. Lin, Z.: Modal Nonassociative Lambek Calculus with Assumptions: Complexity and Context-Freeness. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 414–425. Springer, Heidelberg (2010)
15. Moortgat, M.: Multimodal linguistic inference. Journal of Logic, Language and Information 5, 349–385 (1996)
16. Moortgat, M.: Categorial types logic. In: van Benthem, J., ter Meulen, A. (eds.) Hand Book of Logic and Language, pp. 93–177. Elsevier (1997)
17. Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press (2011)
18. Pentus, M.: Lambek grammars are context free. In: Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science, pp. 429–433 (1993)
19. Plummer, A.: S4 enriched multimodal categorial grammars are context-free. Theoretical Computer Science 388, 173–180 (2007)
20. Plummer, A.: S4 enriched multimodal categorial grammars are context-free: corrigendum. Theoretical Computer Science 403, 406–408 (2008)
21. Versmissen, J.: Grammatical Composition: Modes, Models, Modalities. PhD thesis, Universiteit Utrecht (1996)

Common Nouns as Types

Zhaohui Luo[⋆]

zhaohui.luo@hotmail.co.uk

Abstract. When modern type theories are employed for formal semantics, common nouns (CNs) are interpreted as types, not as predicates. Although this brings about some technical advantages, it is worthwhile to ask: *what is special about CNs that merits them to be interpreted as types?* We discuss the observation made by Geach that, unlike other lexical categories, CNs have criteria of identity, a component of meaning that makes it legitimate to compare, count and quantify. This is closely related to the notion of set (type) in constructive mathematics, where a set (type) is not given solely by specifying its objects, but together with an equality between its objects, and explains and justifies to some extent why types are used to interpret CNs in modern type theories. It is shown that, in order to faithfully interpret modified CNs as Σ-types so that the associated criteria of identity can be captured correctly, it is important to assume proof irrelevance in type theory. We shall also briefly discuss a proposal to interpret mass noun phrases as types in a uniform approach to the semantics of CNs.

1 Introduction

It has been proposed that common nouns be interpreted as types, when modern type theories (MTTs) are used to give formal semantics [25]. This is different from the Montague semantics [21], where common nouns are interpreted as predicates. For instance, consider the CN 'book': it is interpreted in the Montague semantics as a predicate of type $e \rightarrow t$, while in a modern type theory, it is interpreted as a type. It has been argued that, because CNs are interpreted as types rather than predicates, many linguistic phenomena (eg, copredication), whose formal semantic treatments involve subtyping and have been found difficult in the Montagovian setting, can be dealt with satisfactorily in a straightforward way in MTTs [16]. This has provided some justifications, among others, for MTTs to be employed for formal semantics.

However, one may ask: why can CNs be interpreted as types, but not the other lexical terms such as verbs or adjectives? In the Montagovian setting, CNs, verbs and adjectives are all interpreted as predicates, but in a formal semantics based on MTTs, CNs are interpreted as types and verbs and adjectives are still interpreted as predicates, not as types. The above question may be put in another

[⋆] This work is partially supported by the research grant F/07-537/AJ of the Leverhulme Trust in U.K.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 173–185, 2012.

way: *what is special about CNs that merits them to be interpreted as types?* This paper attempts to answer this question and discusses some of the related issues.

We revisit the observation made by Geach and others [11,12,2] that CNs have a special feature that they have their own *criteria of identity*. It is based on this identity criterion that one can decide whether two objects of a CN are the same and it is also based on this that counting, measuring and quantification become possible and meaningful. We argue that it is this special feature that makes it adequate for CNs to be interpreted as types. In constructive mathematics, a set (or a type) is not given solely by specifying its objects, but together with an equality between its objects. In fact, every type in MTTs is associated with such an equality. As illustrated in the paper, when a type is used to interpret a CN, the associated equality corresponds closely to, and formally captures, the criterion of identity of the CN.

In order to correctly capture the criteria of identity for modified CNs in a type-theoretical semantics, where modified CNs are formally represented as Σ-types, one should adopt the principle of *proof irrelevance* (see, for instance, [28] for a recent study). This reflects the basic intuitive idea that, for instance, two handsome men are the same if and only if they are the same man (and it does not matter how one demonstrates that they are handsome). Proof irrelevance identifies the proofs of the same logical proposition and, as a consequence of adopting such a principle, the intended criteria of identity for modified CNs, when formalised as Σ-types, are captured correctly.

Count nouns and mass nouns are both CNs. In formal semantics, one has mostly considered count nouns as primary examples. For mass nouns, although there have been many proposals for their semantic interpretations either as mereological sums or as sets or predicates, no consensus has been reached. In this paper, we take the view that mass noun phrases be interpreted as types and consider a proposal that, when a mass noun is used with a classifier (or measure word), it may be interpreted as a type whose associated equality represents the information given by the classifier. When a mass noun is used without any explicit classifier associated, one may regard it as underspecified in that the criteria of identity can only be determined when more contextual information is available. In MTTs, such underspecification can be represented by means of overloading, as supported by coercive subtyping [17]. As for count nouns, they can be seen as special cases where the measures are obvious and known.[1] This is only a tentative proposal and, when further developed, it may lead to a uniform approach to interpreting CNs, either count or mass, as types.

Introducing the background, §2 contains a brief account of the type-theoretical semantics based on MTTs. In §3, we first introduce the notion of criterion of identity for CNs and then discuss how this is reflected in the formal semantics based on MTTs and how it is linked to the constructive notion of set (or type). Proof irrelevance is discussed in §4, where we show how it can be used in the

[1] This is consistent with the fact that, in the languages with classifiers such as Chinese, a count noun is also used together with a classifier.

	Example	Montague semantics	Semantics in MTTs
CN	man, human	$\llbracket man \rrbracket,\ \llbracket human \rrbracket : e \rightarrow t$	$\llbracket man \rrbracket,\ \llbracket human \rrbracket : Type$
IV	talk	$\llbracket talk \rrbracket : e \rightarrow t$	$\llbracket talk \rrbracket : \llbracket human \rrbracket \rightarrow Prop$
ADJ	handsome	$\llbracket handsome \rrbracket : (e \rightarrow t) \rightarrow (e \rightarrow t)$	$\llbracket handsome \rrbracket : \llbracket man \rrbracket \rightarrow Prop$
MCN	handsome man	$\llbracket handsome \rrbracket(\llbracket man \rrbracket)$	$\Sigma m : \llbracket man \rrbracket .\ \llbracket handsome \rrbracket(m)$
S	A man talks	$\exists m : e.\ \llbracket man \rrbracket(m) \& \llbracket talk \rrbracket(m)$	$\exists m : \llbracket man \rrbracket .\ \llbracket talk \rrbracket(m)$

Fig. 1. Examples in formal semantics

formal semantics based on MTTs to obtain adequate descriptions of modified CNs. Type-theoretical semantics of mass nouns phrases is discussed in §5.

2 CNs as Types in Formal Semantics

In this section, we give a brief introduction to the type-theoretical semantics based on modern type theories (MTTs) [25,16].[2] It is the formal semantics in the style of the Montague semantics [21], but in type theories with dependent types and inductive types, among others, rather than in Church's simple type theory [7] as employed in the Montague semantics. Examples of MTTs include Martin-Löf's predicative type theory [19,20] and the impredicative type theory ECC/UTT [14]. In an *impredicative* type theory like UTT, there is a type $Prop$ of all logical propositions, as to be used in this paper. (This is similar to the simple type theory where there is a type t of truth values.)

In Figure 1, we give some basic examples to illustrate how linguistic categories are interpreted in MTTs and compare them to those in the Montague semantics. A key difference between these two is the interpretation of CNs. In the Montague semantics whose underlying logic can be regarded as 'single-sorted'[3], CNs are interpreted as predicates of type $e \rightarrow t$ (and so are verbs and adjectives). In contrast, MTTs can be regarded as 'many-sorted' logical systems where there are many types (eg, inductive types such as the finite types as introduced in Appendix B) that may be used to stand for the domains to be represented; in particular, CNs are interpreted as types [25]. Similarly, modified CNs (MCNs) are interpreted as predicates in the Montague semantics, while they are interpreted by means of Σ-types in MTTs. For instance, in MTTs, 'handsome man' can be interpreted as the type $\Sigma m : \llbracket man \rrbracket .\ \llbracket handsome \rrbracket(m)$. (For Σ-types and their notations, see Appendix A.) Because CNs are interpreted as types, verbs and adjectives are interpreted as predicates over the types (eg, $\llbracket human \rrbracket$) that interpret the domains in which they are meaningful: examples are given in Figure 1.

Note that subtyping is crucial for the formal semantics in MTTs. For instance, consider the MTT semantics of the sentence 'A man talks' in Figure 1: for m of type $\llbracket man \rrbracket$ and $\llbracket talk \rrbracket$ of type $\llbracket human \rrbracket \rightarrow Prop$, $\llbracket talk \rrbracket(m)$ is only well-typed because

[2] One may also consult the lecture notes [18], where some informal explanations of MTTs with subtyping are given in the context of formal semantics.

[3] By 'single-sorted' here, we mean that there is only one type e of all entities. Strictly speaking, there is another 'sort'/type t of truth values in Church's simple type theory.

we have that $[\![man]\!]$ is a subtype of $[\![human]\!]$. For MTTs, coercive subtyping as studied in [15] is an adequate framework to be employed for formal semantics [16].

To employ modern type theories, instead of the simple type theory, for formal semantics has many implications, some of which are philosophical, some methodological, and some technical. Technically, for example, the powerful type structures in MTTs give us new useful mechanisms for formal semantics of various linguistic features, examples of which include the use of the dependent Σ-types to interpret modified CNs [25] and the introduction of a type universe $\textsc{cn}$ of the interpretations of common nouns in many linguistic interpretations, including that of adverbs [17].

One of the most notable methodological implications comes from the fact that CNs are interpreted as types, not as predicates. For example, in modelling some linguistic phenomena semantically, one may introduce various subtyping relations by postulating a collection of subtypes (physical objects, informational objects, eventualities, etc.) of the type of entities [1]. It has become clear that, once such subtyping relations are introduced, it is very difficult to deal with some linguistic phenomena such as copredication satisfactorily if CNs are interpreted as predicates as in the traditional Montagovian setting. Instead, if CNs are interpreted as types, as in the type-theoretical semantics based on MTTs, copredication can be given a straightforward and satisfactory treatment [16].

The above methodological advantage may go some way to justify that CNs be interpreted as types (and the employment of MTTs for formal semantics). However, why should CNs be interpreted as types in the first place? Why are they different from the other lexical categories such as verbs and adjectives? After all, in the Montague semantics, CNs, verbs, and adjectives are all interpreted as predicates. Put in another way:

What is special about CNs that merits them to be interpreted as types?

The rest of this paper investigates the related issues and may be regarded as a first step to answer the above question.

3 Criteria of Identity

3.1 An Informal Account

CNs are special, as observed by Geach [11] and others, in that they have the associated *criteria of identity*. Intuitively, a CN determines a concept that does not only have a criterion of application, to be employed to determine whether the concept applies to an object, but a criterion of identity, to be employed to determine whether two objects of the concept are the same. It has been argued that CNs are distinctive in this as other lexical terms like verbs and adjectives do not have such criteria of identity (cf, the arguments in [2]).

The origin of the notion of criteria of identity can be traced back to Frege [10] when he considered abstract mathematical objects such as numbers or lines. For instance, in geometry, the criterion of identity for directions is the parallelism of lines: the direction of line A is equal to that of line B if and only if A and B are parallel. Geach has noticed that such criteria of identity exist for every common noun and is the basis for counting. Gupta [12] has studied this systematically with very interesting examples. For instance, consider the following two sentences:

(1) EasyJet has transported 1 million passengers in 2010.

(2) EasyJet has transported 1 million persons in 2010.

It is easy to see that the first sentence (1) does not imply the second (2), because some people may have traveled more than once by EasyJet in 2010. It has been argued that this is because that the CNs ‘passenger’ and ‘person’ have different criteria of identity, which are the basis for counting and have led to such phenomena [11,12,2].[4]

It may be worth noting that the notion of the criterion of identity is context sensitive. In other words, what a CN means depends on the context in which it is used. For instance, consider the word ‘student’. In the following sentences, the associated criteria of identity can be different, because in (3) John may have taught several classes and it may be reasonable to count a student in two different classes twice, while this is not the case in (4) where one would say that in that case ‘student’ seems to have the same criterion of identity with ‘person’.

(3) John taught 500 students last year.

(4) 1000 students have applied for campus cards last year.

A close link between the notion of criterion of identity and the constructive notion of set (type) can be established. In constructive mathematics, a set is a ‘preset’, which gives its application criterion, together with an equality, which gives its criterion of identity that determines whether two objects of the set are the same [5,4]. Modern type theories such as Martin-Löf’s type theory [19,20] were originally developed for formalisation of constructive mathematics, where each type is associated with such an equality or criterion of identity. In the following, we shall first consider the formalisations in MTTs of the above example of passengers to demonstrate how the criteria of identity are reflected in such formal representations, and then discuss the link to constructive mathematics in more details.

3.2 Formalisation of an Example

The above example about ‘passengers’ can be formalised in the MTTs. Let T be the type of the journeys in concern (eg, the journeys that one may make via EasyJet in 2010). We shall consider two different formal presentations of $Passenger[T]$, the type of passengers in journeys of type T, one using finite types and the other considering proof irrelevance.

[4] There have been arguments against the idea of criteria of identity. For instance, Gupta [12] mentioned that one might consider some ontological arguments and Barker [3] has argued against it on the grounds that the linguistic phenomena could better be explained by means of pragmatics. The author believes that the notion of criteria of identity still offers the best explanations.

Representation using finite types. The type $Passenger[T]$ can be defined as the following Σ-type:[5]

$$Passenger[T] = \Sigma p : Person.\ Journey[T](p),$$

where $Person$ is the type of persons and $Journey[T](p)$ the finite type of journeys in T that the person p has made. (See Appendix B for the formal definition of finite types.) In other words, a passenger is a person together with a journey he/she made and, formally, this is represented as a pair (p, t) of type $Passenger[T]$, where t is a journey that the person p has made. There are two points to note about this definition:

1. If $Journey[T](p)$ is empty (the finite type with no object), p has not made any journey in T and is hence not a passenger by definition (there is no passenger (p, t) of type $Passenger[T]$ because there is no such a t of type $Journey[T](p)$.)
2. Formally, (p, t) and (p', t') are equal passengers if, and only if, $p = p'$ and $t = t'$.[6] As passengers, 'John at journey t' and 'John at journey t'' are only equal if t and t' are the same journey.

This last note about equality between passengers is important. It is different from that between persons: the same person making different journeys is regarded as different passengers.

Representation assuming proof irrelevance. Another representation assumes proof irrelevance:

$$\frac{\Gamma \vdash P : Prop \quad \Gamma \vdash p : P \quad \Gamma \vdash q : P}{\Gamma \vdash p = q : P}$$

which intuitively says that every two proofs of the same logical proposition are equal. (See §4 for further discussions on proof irrelevance.) With proof irrelevance, the following Σ-type can be used to represent the type of passengers:

$$Passenger[T] = \Sigma p : Person. \Sigma t : T.\ J(p, t),$$

where $J : Person \to T \to Prop$ is the predicate such that the predicate $J(p)$ of type $T \to Prop$ represents the set of journeys that p has made. It is straightforward to show that two passengers are the same if and only if they are the same person on the same journey. In symbols, $(p, t, v) = (p', t', v')$ if, and only if, $p = p'$ and $t = t'$, because by proof irrelevance, we always have $v = v'$ when the other two components are the same.

[5] An alternative notation for the Σ-type $Passenger[T]$ is $\Sigma(Person, Journey[T])$. See Appendix A for a brief introduction of Σ-types.

[6] Note that, for any $p : Person$, $Journey[T](p)$ is a type, not a logical proposition. Therefore, this is the case even when we have proof irrelevance, as to be discussed in §4.

Remark 1. Both of the above representations give the correct criterion of identity. Intuitively, they give rise to the same criterion of identity between passengers as intended. Other formalisations are also possible. However, it is easy to arrive at some unintended formulations. For instance, it might be tempting to say that 'a passenger is a person who has made one or more journeys'. This would lead to the formalisation of the type of passengers as a Σ-type

$$\Sigma p : Person.\ Travelled[T](p),$$

where $Travelled[T](p)$ is a logical proposition expressing that p has made some journeys in T. Such a formulation does not capture the intended criterion of identity between passengers, no matter whether we assume proof irrelevance or not. For instance, if we do, we have that two objects of the above type ('passengers') are the same if and only if they are the same person who has travelled, because the proofs that the person has travelled are identified. Therefore, such a formal representation would not have captured the intended criterion of identity correctly. □

3.3 Constructive Notion of Set or Type: Further Remarks

In constructive mathematics, the notion of set is associated with an equality (an equivalence relation) [5]. As Beeson [4] puts it, it is a 'preset' together with an equality. A preset X is given by its criterion of application that determines whether an object is in X, while the associated equality determines whether two objects of the set are the same. Unlike classical mathematics, there is no universal equality that can be applied to all objects; instead, different sets are associated with different equalities. When CNs are interpreted as sets, this is directly linked to the notion of criteria of identity in that different CNs may have different criteria of identity.

In modern type theories, a type is a constructive set.[7] For instance, in Martin-Löf's type theory, a type is specified by making clear the following simultaneously:

1. What are the canonical objects of the type?
2. Under what conditions are two canonical objects equal?

Please note that, for completely presented types, these two are enough to determine both criterion of application and criterion of identity.

In MTTs in general, every type is associated with its own equality. For instance, the equality for $\Sigma x{:}A.\ B(x)$ is: for any of its two objects (a, b) and (a', b'), they are equal only when $a = a'$ in type A and $b = b'$ in type $B(a)$ (please note that B is dependent on the objects of A; in this case, it is a). According to

[7] We are rather imprecise here. Types in MTTs are also *restricted* sets with further properties. For example, a type can be inductively defined and, if so, it has many special properties that are not shared by all types. Also, in Martin-Löf's type theory, every type is *completely presented* in the sense that, informally, its criterion of application can be evidenced by computation. The author thinks that such a requirement for objects to be completely presented should not be imposed on linguistic interpretations when MTTs are used for formal semantics.

this definition of equality for Σ-types, both of the above representations for $Passenger[T]$ correctly capture the criterion of identity between passengers as intended. Please note that, in the second representation above, we have to assume proof irrelevance for, otherwise, the representation would not capture the criterion of identity between passengers correctly: if we do not have proof irrelevance, the 'passengers' (p, t, v) and (p, t, v') can be different because the proofs v and v' that p made the journey t are different, although they should be the same passenger.

Remark 2. It seems that the close link between CNs with criteria of identity and types with associated equalities is one of the instances where principles in constructive mathematics can be successfully applied to linguistic semantics in an interesting way. This is reflected above when constructive types are used to represent the semantics of CNs. Further studies of the use of MTTs for formal semantics may shed more light in this respect. □

4 Proof Irrelevance and Identity for Modified CNs

For modified CNs, it is often the case that the criteria of identity are inherited from those before modification. For example, two 'handsome men' are the same if, and only if, they are the same man. In such situations, the adjective used to modify the CN has no effect on the resulting criterion of identity for the modified CN. This should be captured faithfully in a semantic framework.

In MTTs, it has been proposed that modified CNs be interpreted as Σ-types [25].[8] For instance, for $[\![man]\!] : Type$ and $[\![handsome]\!] : [\![man]\!] \rightarrow Prop$, the interpretation of 'handsome man' is the following Σ-type:

$$[\![handsome\ man]\!] = \Sigma m : [\![man]\!] \, . \; [\![handsome]\!](m).$$

However, in type theories (as those implemented in the proof assistants such as Agda, Coq and Lego), the notion of equality between objects of such Σ-types does not capture the intended criteria of identity. In the above example, for the representations (m, h) and (m', h') of two handsome men to be equal, we require not only that the two men m and m' be equal, but that the proofs h and h' be equal as well. But this is usually not the case (there can be more than one way to demonstrate that a man is handsome)!

In order for such Σ-type representations to be faithful in capturing the intended criteria of identity, we should employ *proof irrelevance*, which intuitively says that every two proofs of the same logical proposition are equal. In an impredicative type theory such as UTT, the formal rule for proof irrelevance is (as repeated from §3.2):

$$\frac{\Gamma \vdash P : Prop \quad \Gamma \vdash p : P \quad \Gamma \vdash q : P}{\Gamma \vdash p = q : P}$$

[8] Gupta [12] has suggested a special form of formulae $(K, x)A$, called *restrictions*, for modified CNs. Linking formulae to types, we can easily see the close correspondence between $(K, x)A$ and $\Sigma x : K. \ A$.

Proof irrelevance has been studied for impredicative type theories. For instance, a set-theoretic proof irrelevant model was given in [9] and one can find a recent study on this in [28], where the author has employed the untyped notion of conversion instead of a judgemental equality.

In the above example about handsome men, if $m = m'$, the two proofs h and h' prove the same logical proposition $[\![handsome]\!](m)$ and are hence equal. As a consequence, under this representation, two handsome men are the same if, and only if, they are the same man. In other words, under the assumption of proof irrelevance, the proposed representations of modified CNs by Σ-types do capture the intended criteria of identity.

It is also worth noting that, although proof irrelevance can be considered for impredicative type theories directly as above, it is unclear how this can be done for predicative type theories. For instance, in Martin-Löf's type theory [19,20], propositions are identified with types. Because of such an identification, one cannot use the above rule to identify proofs, for it would identify the objects of a type as well. Put in another way, proof irrelevance is incompatible with the identification of propositions and types. In order to introduce proof irrelevance, one has to distinguish logical propositions and types (see, for example, [14]).

Remark 3. Proof irrelevance is very interesting and has several important applications. Besides the above application in formal semantics, it is also employed in several other fields, including dependently-typed programming (see, for example, [28] for some relevant discussions). □

5 Semantics for Mass Nouns with Classifiers

Common nouns include both count and mass nouns. It seems clear how to consider the criteria of identity for count nouns, but less clear for mass nouns. As for the semantics of mass nouns, scholars have rather different opinions and there seems to be no consensus on this matter.[9] The proposed semantic theories on mass terms fall into two camps: those based on the idea that mass nouns denote *mereological sums*, as advocated by Quine [24] and others, and those based on ideas that they denote *sets*, as considered by people like Laycock [13] (see [6,29] for some rather comprehensive analysis of early work on this).

Here, based on the idea that CNs denote sets/types with associated criteria of identity, I offer a tentative proposal to suggest how some, if not all, of the uses of mass noun phrases may be interpreted. Mass nouns are often used together with measure words or classifiers, as in the following sentences about the mass noun 'water':

(5) John drinks a glass of water.

(6) He has fetched two buckets of water from the river.

[9] One may quote several references, among many, including Quine's remarks on 'divided reference' [24], Strawson's on 'individuals' [26], Baker's on measures [2], Bunt's work on ensemble theory [6] and Nicolas' work on plural logic [22].

The measure words such as ‘glass’ and ‘bucket’ provide information for the criteria of identity in these mass noun phrases. It has been proposed [26,8] that mass terms should be understood as elliptical for some phrases with classifiers. For instance, ‘water’ would be elliptical for ‘drop of water’, ‘glass of water’, ‘bucket of water’, ’body of water’, etc. These latter phrases with classifiers are semantically easier to be interpreted as sets and, in particular, it is clearer what the criteria of identity should be. In a type-theoretical semantics based on MTTs, they can be interpreted as types. For example, the mass noun phrase ‘glass of water’ can be interpreted as a type and the above sentence (5) can be given the following semantics:

(7) $\exists w : [\![glass\ of\ water]\!] .\ [\![drink]\!](j, w)$

In such cases, the criteria of identity are provided by the measure words (see [2] for relevant remarks).

In languages like English, count nouns are usually[10] used without measure words (we say ‘a table’ rather than ‘a CLASSIFIER table’). A plausible explanation for this is that the criterion of identity for a count noun, when it is used without a measure word, is ‘obvious’ or already built in the noun itself. This suggests a uniform way to approach the semantic interpretations of count and mass nouns: they are all interpreted as types and, in the case of mass noun phrases, the measure words in the phrase provide us the information for the criteria of identity while, in the case of count nouns, the counting principle is usually obviously given by the noun itself. Put in another way, this uniform approach is based on the idea that the ‘obvious’ measures for count nouns are just special cases of the numerals-measures-CN pattern for mass noun phrases such as ‘a glass of water’ in (5) and ‘two buckets of water’ in (6).

Such an approach is also informed by considering the languages with classifiers such as Chinese, where every noun is usually used together with a classifier or measure word, even for the count nouns in an English-like language. In Chinese, instead of saying ‘a book’, one says ‘a CLASSIFIER book’, where CLASSIFIER is a suitable measure word for ‘book’. Some people take the view that the Chinese language does not have count nouns – every CN in Chinese is a mass noun whose uses are often coupled with classifiers. This is consistent with the uniform approach that both count and mass noun phrases are interpreted as types.

When a mass noun is not explicitly used together with a measure word, it becomes a context-dependent matter to determine which measure should be used. There have been some criticisms on the proposal that mass nouns be interpreted as sets (see, for example, [23]). One of the main criticisms is that, when a mass noun is used without a measure word explicitly given, such context dependency seems to amount to a difficulty in considering a ‘general translation procedure’ to give the semantics for any mass noun [23]. To deal with such potential difficulties, we consider one proposal: in many situations where a mass noun is used without a measure word, it can be seen as *underspecified* and, according

[10] Occasionally, measure words are also used for count nouns in, for example, ‘a deck of cards’.

to further enrichments of the contexts, its meaning can then be determined. In MTTs, such underspecified phrases may be given meanings by means of *overloading*, supported by coercive subtyping [17].

The proposal to interpret both count and mass CNs as types requires further studies, both for its justification in general and for the possibility of using the idea to implement reasoning systems that involve mass nouns on the computer.

6 Conclusion

This paper has investigated the idea that CNs have the meaning component of criteria of identity and can be semantically represented by means of types in MTTs. We have pointed out that, in order to interpret CNs adequately, especially to interpret modified CNs by means of Σ-types, the principle of proof irrelevance should be adopted and, furthermore, this principle can only be properly formulated in a type theory where there is a clear distinction between logical propositions and data types (eg, as in most of the impredicative type theories or logic-enriched type theories), but not in those type theories such as Martin-Löf's type theory where there is no such a distinction.

One of the interesting issues is to study generalised quantifiers based on MTTs (see [27] for an early investgation of this in Martin-Löf's type theory.) In this setting, the generalised quantifiers such as 'every' and 'most' have the following type:

$$\Pi A : \textsc{cn}.\ (A \rightarrow Prop) \rightarrow Prop,$$

where CN is the type universe of the interpretations of common nouns. In other words, a generalised quantifier takes as its arguments a type that interprets a CN and a predicate over the type. This is rather different from the 'symmetric' notion of GQs in the Montagovian setting where GQs are concerned with two predicates. This is obviously an interesting topic to study about the formal semantics based on MTTs and requires future investigations.

Acknowledgements. Thanks to Robin Adams, Stergios Chatzikyriakidis and members of the 'Type Theory and Applications' research group at RHUL for helpful discussions.

References

1. Asher, N.: Lexical Meaning in Context: A Web of Words. Cambridge University Press (2011)
2. Baker, M.: Lexical Categories: Verbs, Nouns and Adjectives. Cambridge University Press (2003)
3. Barker, C.: Nominals don't provide criteria od identity. In: Alexiadou, A., Rathert, M. (eds.) Nominalizations across Languages and Frameworks. Intrface Explorations (2008)
4. Beeson, M.: Foundations of Constructive Mathematics. Springer (1985)
5. Bishop, E.: Foundations of Constructive Analysis. McGraw-Hill (1967)

6. Bunt, H.: Mass Terms and Model-Theoretic Semantics. Cambridge University Press (1985)
7. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic 5(1) (1940)
8. Clarke, D.: Mass terms as subjects. Philosophical Studies 21(1/2) (1970)
9. Coquand, T.: Metamathematical investigations of a calculus of constructions. In: Oddifredi, P. (ed.) Logic and Computer Science (1990)
10. Frege, G.: Grundlagen der Arithmetik. Basil Blackwell (1884), (Translation by J. Austin in 1950: The Foundations of Arithmetic)
11. Geach, P.: Reference and Generality. Cornell University Press (1962)
12. Gupta, A.: The Logic of Common Nouns. Yale University Press (1980)
13. Laycock, H.: Some questions of ontology. Philosophical Review 81(1) (1972)
14. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ. Press (1994)
15. Luo, Z.: Coercive subtyping. J. of Logic and Computation 9(1), 105–130 (1999)
16. Luo, Z.: Type-theoretical semantics with coercive subtyping. Semantics and Linguistic Theory 20 (SALT20), Vancouver (2010)
17. Luo, Z.: Contextual Analysis of Word Meanings in Type-Theoretical Semantics. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 159–174. Springer, Heidelberg (2011)
18. Luo, Z.: Type-theoretical semantics with coercive subtyping. Lecture notes at ESSLLI 2011 (for Lexical Semantics, a course taught together with N. Asher), Ljubljana (2011), http://www.cs.rhul.ac.uk/home/zhaohui/ESSLLI11notes.pdf?
19. Martin-Löf, P.: An intuitionistic theory of types: predicative part. In: Rose, H., Shepherdson, J.C. (eds.) Logic Colloquium'73 (1975)
20. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
21. Montague, R.: Formal Philosophy. Yale University Press (1974)
22. Nicolas, D.: Mass nouns and plural logic. Linguistics and Philosophy 31(2) (2008)
23. Pelletier, F.: On some proposals for the semantics of mass nouns. J. of Philosophical Logic 3 (1974)
24. Quine, W.: Word & Object. MIT Press (1960)
25. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)
26. Strawson, P.: Individuals: An Essay in Descriptive Metaphyscis. Anchor Books (1963)
27. Sundholm, G.: Constructive generalized quantifiers. Synthese 79(1) (1989)
28. Werner, B.: On the strength of proof-irrelevant type theories. Logical Methods in Computer Science 4(3) (2008)
29. Zimmerman, D.: Theories of masses and problems of constitution. Philosophical Review 104(1) (1995)

A Σ-Types

A Σ-type is an inductive type of dependent pairs. Here are the informal descriptions of the basic laws governing Σ-types (see, for example, [20] for the formal rules).

- 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.
- $\Sigma(A, B)$ consists of pairs (a, b) such that a is of type A and b is of type $B(a)$.
- Σ-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 $\Sigma(A, B)$.

When $B(x)$ is a constant type (i.e., always the same type no matter what x is), the Σ-type degenerates into the product type $A \times B$ of non-dependent pairs.

B Finite Types of Journeys

Finite types are inductive types which contain finitely many objects. They were considered in [19,20] for natural numbers: intuitively, N_n contains the natural numbers $0, 1, ..., n-1$. Here, we introduce them as containing finitely many journeys.

Let T be the type of the journeys in concern. Then, for $n \in \omega$ and $t_i : T$ $(i = 1, ..., n)$, $Fin[t_1, ..., t_n]$ is the finite type with the journeys t_i as its objects. For instance, $Fin[t_1, t_2, t_3]$ contains only three journeys t_1, t_2 and t_3, while $Fin[]$ is an empty type that does not contain any object. Formally, these finite types are specified by means of the rules in Figure 2.

Now, for any $p : Person$, $Journey[T](p)$ is the finite type of journeys in T that p has made. For instance, intuitively, if $Journey[T](p) = Fin[t_1, t_2, t_3]$, p has made journeys t_i $(i = 1, 2, 3)$, while if $Journey[T](p) = Fin[]$, p has made no journeys.[11]

Formation Rule

$$\frac{t_i : T \quad (i = 1, ..., n)}{Fin[t_1, ..., t_n] : Type} \quad (n \in \omega)$$

Introduction Rule

$$\frac{Fin[t_1, ..., t_n] : Type}{t_i : Fin[t_1, ..., t_n]} \quad (i = 1, ..., n)$$

Elimination Rule

$$\frac{c : Fin[t_1, ..., t_n] \quad C : (Fin[t_1, ..., t_n])Type \quad c_j : C(t_j)\ (j = 1, ..., n)}{\mathcal{E}_n(C, c_1, ..., c_n, c) : C(c)}$$

Computation Rule

$$\frac{C : (Fin[t_1, ..., t_n])Type \quad c_j : C(t_j)\ (j = 1, ..., n)}{\mathcal{E}_n(C, c_1, ..., c_n, t_i) = c_i : C(t_i)} \quad (i = 1, ..., n)$$

Fig. 2. Rules for finite types of journeys

[11] The details of the formal definition of $Journey[T]$ is not given here: to do it properly, we need to consider the type structure of $Person$ (eg, its inductive structure) and employs a type universe U that contains the finite types of journeys as objects and to define $Journey[T]$ to be of type $Person \to U$.

Extractability as the Deduction Theorem in Subdirectional Combinatory Logic*

Hiroko Ozaki and Daisuke Bekki**

Ochanomizu University,
Faculty of Science, Department of Information Science,
2-1-1 Ohtsuka, Bunkyo-ku, Tokyo 112-8610, Japan

1 Introduction: Subdirectional Combinatory Logic and the Curry-Howard Isomorphism between Grammars, Term Calculi and Logics

The formulation of Combinatory Categorial Grammar (CCG) [7], especially the choice of combinatory rules and their nominatum, strongly imply connection with a typed-version of Combinatory Logic (CL). Since typed CL is a term calculus for an implication fragment of a Hilbert-style proof system, in the sense of the Curry-Howard isomorphism, it seems plausible to regard CCG as a grammar that corresponds to a Hilbert-style proof system, in that the associative Lambek calculus [3] corresponds to a Gentzen-style proof system.

This correspondence was not, however, as strictly established as expected. The main difference between CCG and CL is that the different linear orders of words are distinguished in CCG (as a grammar), that is, the exchange rule is not fully available in the contexts (as a proof system), which is reflected in the existence of two functional application rules. This is also true in Lambek lambda calculus, and gives rise to the emergence of two different lambda operators and function application constructions, in order to maintain the parallelism between the terms and their types. In CCG, however, this parallelism is not pursued so rigorously: CCG adopts simply-typed lambda calculus for its semantic representations and says nothing about the directionality of the lambda operators.

Subdirectional Combinatory Logic (SDCL) [1] is a term calculus that exactly corresponds to CCG. It is a kind of CL with directionality-sensitive combinators. More precisely speaking, SDCL is a class of logics containing various instances of CCG for each individual language, and we conjectured that the language variation can be described in terms of availability/absence of each directional

* We are grateful to Nicholas Asher, Frank Veltman, Alastair Butler, Shunsuke Yatabe and other audience of LENLS8 for their valuable comments and suggestions for the earlier version of this paper. We are also grateful to Kei Yura in Joint Research Symposium 2011.

** 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.

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 186–200, 2012.

combinator. The definition of SDCL that we use throughout this paper is given in the Appendix A.

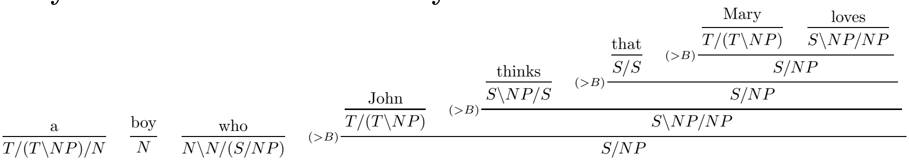
The Curry-Howard isomorphism between a grammar and a term calculus or logic affords many advantages in the study of formal grammar. In Lambek calculus, for example, many results and methods of proof theory, such as cut-elimination, subformula properties and decidability, have been utilized and have brought about fruitful insights. As an illustration of such advantages, this paper discusses the following issues:

1. The relation between the status of extractability in CCG and the Deduction Theorem (DT) in Hilbert-style proof systems, with respect to the correspondence obtained through SDCL.
2. The relation between DT and the structural rules in CCG.
3. The relation between the type-raising rule and the complex NP constraints in CCG.

2 Extraction in CCG

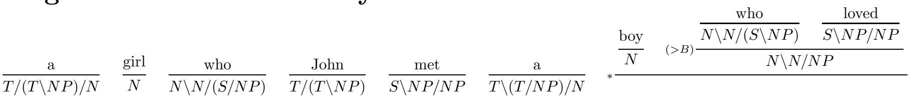
In lexical grammars, the term *extraction* signifies *wh*-movements in generative grammar, that is, the dislocation of a *wh*-NP to a non-argument position. The following sentence is an instance of extraction of **who** from the object position of **loves**:

a boy who John thinks that Mary loves



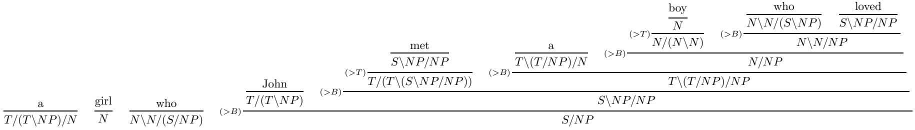
The sentence is acceptable and is also derivable in CCG. However, it is well known that extraction from a complex NP is not allowed in English (*the complex NP constraint* [6]), as illustrated below:

*** a girl who John met a boy who loved**



This sentence is unacceptable and is not derivable in CCG. However, if we allow a type-raising rule for CCG, the sentence becomes derivable in the following way:

*** a girl who John met a boy who loved**



Note that this unacceptable instance is also derivable in associative Lambek caliculus L [4] as follows[1]:

$$
{\scriptstyle (/R)}\dfrac{
{\scriptstyle (<)}\dfrac{
\text{John} \Rightarrow NP \quad
{\scriptstyle (<)}\dfrac{
\text{met} \Rightarrow S\backslash NP/NP \quad
{\scriptstyle (>)}\dfrac{
\text{a} \Rightarrow T\backslash(T/NP)/N \quad
{\scriptstyle (<)}\dfrac{
\text{boy} \Rightarrow N \quad
{\scriptstyle (>)}\dfrac{
\text{who} \Rightarrow N\backslash N/(S\backslash NP) \quad
{\scriptstyle (>)}\dfrac{
\text{loved} \Rightarrow S\backslash NP/NP \quad
{\scriptstyle (id)}\dfrac{}{NP \Rightarrow NP}
}{\text{loved } NP \Rightarrow S\backslash NP}
}{\text{who loved } NP \Rightarrow N\backslash N}
}{\text{boy who loved } NP \Rightarrow N}
}{\text{a boy who loved } NP \Rightarrow T\backslash(T/NP)}
}{\text{met a boy who loved } NP \Rightarrow S\backslash NP}
}{\text{Jhon met a boy who loved } NP \Rightarrow S}
}{\text{John met a boy who loved } \Rightarrow S/NP}
$$

The $(/R)$ rule of L corresponds to DT. This means that both CCG with the type-raising rule and associative Lambek calculus do not account for the complex NP constraint.

In general, the extraction in CCG (without type-raising) is available only in the case where the syntactic categories of the relevant phrases match one of the following forms:

$$
{\scriptstyle (>B)\times n}\dfrac{X_1/X_2 \quad X_2/X_3 \quad \cdots \quad X_{n+1}/X_{n+2}}{X_1/X_{n+2}}
$$

$$
{\scriptstyle (<B)\times n}\dfrac{X_{n+1}\backslash X_{n+2} \quad \cdots \quad X_2\backslash X_3 \quad X_1\backslash X_2}{X_1\backslash X_{n+2}}
$$

In other words, we can use this rule under the condition that all the relevant *slashes* have the same direction. We will show that this condition is deeply connected with the proof of DT in SDCL, in which the *directionality* of the available combinators plays an important role.

3 Abstraction in SDCL

The $(/R)$ rule corresponds to the implication-introduction rule in natural deduction. By Curry-Howard correspondence between natural deduction and typed-Lambda calculus, a proof that ends with the implication-introduction rule is a lambda abstraction term. Hilbert-style proof systems do not have the implication-introduction rule; however, DT is admissible. As a term calculus for Hilbert-style proof systems, CL does not have lambda abstraction terms, but these terms can be encoded solely by the combinators in CL (see [2] among others).

Definition 1 (Abstraction in CL [2]). *For every term M and every variable x, terms of the form $\lambda x.M$ are defined recursively as follows:*

$$
\begin{array}{llll}
(1) & \lambda x.x & \overset{def}{\equiv} \mathbf{I} & \\
(2) & \lambda x.MN & \overset{def}{\equiv} \mathbf{S}(\lambda x.M)(\lambda x.N) & (x \in fv(MN)) \\
(3) & \lambda x.M & \overset{def}{\equiv} \mathbf{K}M & (x \notin fv(M))
\end{array}
$$

Following this strategy, lambda abstraction terms in SDCL can be encoded by means of directional combinators. Following [5], let us consider an instance of SDCL with the combinators $\mathbf{S}$ and $\mathbf{K}$.

[1] $(>)$ is derivable in L.

Definition 2 (Abstraction in SDCL [1] (slightly modified)). *For every term M and every variable x, terms of the form $\overset{\triangleright}{\lambda}x.M$ and $\overset{\triangleleft}{\lambda}x.M$ are defined recursively as follows:*

$$
\begin{array}{llcll}
(1/) & \overset{\triangleright}{\lambda}x.x & \overset{def}{\equiv} & \mathbf{I}_/ & \\
(1\backslash) & \overset{\triangleleft}{\lambda}x.x & \overset{def}{\equiv} & \mathbf{I}_\backslash & \\
(2/) & \overset{\triangleright}{\lambda}x.M\,{}^{\triangleright}N & \overset{def}{\equiv} & \mathbf{S}_/{}^{\triangleleft\!\triangleright}(\mathbf{K}{}^{\triangleleft\!\triangleright}M){}^{\triangleright}(\overset{\triangleright}{\lambda}x.N) & (x \notin fv(M), x \in fv(N)) \\
(2\backslash) & \overset{\triangleleft}{\lambda}x.M\,{}^{\triangleleft}N & \overset{def}{\equiv} & \mathbf{S}_\backslash{}^{\triangleleft\!\triangleright}(\mathbf{K}{}^{\triangleleft\!\triangleright}M){}^{\triangleleft}(\overset{\triangleleft}{\lambda}x.N) & (x \notin fv(M), x \in fv(N))
\end{array}
$$

Note that in SDCL, lambda abstraction cannot be applied to all kinds of SDCL terms. The following are examples of *non-SDCL terms*:

$$\overset{\triangleright}{\lambda}x.y \qquad \overset{\triangleright}{\lambda}p.p\,{}^{\triangleright}x \qquad \overset{\triangleright}{\lambda}x.f\,{}^{\triangleright}y \qquad \overset{\triangleright}{\lambda}x.p\,{}^{\triangleleft}x$$

η-reduction holds for the encoded lambda abstraction terms in SDCL.

Theorem 3 (η-reduction in SDCL). *For every term M and variable x:*

$$
\begin{array}{c}
(\overset{\triangleright}{\lambda}x.M){}^{\triangleright}x \underset{CL}{\rightarrow} M \\
(\overset{\triangleleft}{\lambda}x.M){}^{\triangleleft}x \underset{CL}{\rightarrow} M
\end{array}
$$

Proof. We prove $(\overset{\triangleright}{\lambda}x.M){}^{\triangleright}x \underset{CL}{\rightarrow} M$ by induction on the structure of M.

<u>Case: $M \equiv x$</u> $(\overset{\triangleright}{\lambda}x.M){}^{\triangleright}x \;\equiv\; \mathbf{I}{}^{\triangleright}x \;\underset{CL}{\rightarrow}\; x$

<u>Case: $M \equiv U{}^{\triangleleft}V$</u>

$$
\begin{array}{rll}
(\overset{\triangleright}{\lambda}x.M){}^{\triangleright}x & \equiv \mathbf{S}_/{}^{\triangleleft\!\triangleright}(\mathbf{K}{}^{\triangleleft\!\triangleright}U){}^{\triangleright}(\overset{\triangleright}{\lambda}x.V){}^{\triangleright}x & \\
& \underset{CL}{\rightarrow} ((\mathbf{K}{}^{\triangleleft\!\triangleright}U){}^{\triangleleft}x){}^{\triangleright}((\overset{\triangleright}{\lambda}x.V){}^{\triangleright}x) & \\
& \underset{CL}{\rightarrow} U{}^{\triangleright}V & \text{(by the induction hypothesis)}
\end{array}
$$

$(\overset{\triangleleft}{\lambda}x.M){}^{\triangleleft}x \underset{CL}{\rightarrow} M$ can be proved in the same way.

4 DT in SDCL

The encoding of lambda abstraction terms corresponds to the proof of DT in Hilbert-style proof systems. Conversely, DT corresponds to the typing rule for the encoded lambda abstraction terms. The fully general version of DT in SDCL is considered to have the following forms:

$$
(DT_l)\,\frac{x : A, \Gamma \vdash M : B}{\Gamma \vdash \overset{\triangleleft}{\lambda}x.M : B\backslash A}
\qquad
(DT_r)\,\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \overset{\triangleright}{\lambda}x.M : B/A}
$$

However, each instance of SDCL adopts only a subset of directional combinators, so each DT has a restriction on its use resulting from the combinators it adopts. In the instance that corresponds to CCG without type raising, which is presented in the Appendix A, a restricted version of DT holds as follows:

Theorem 4 (DT in SDCL). *In SDCL, the following rules are admissible (Γ is an arbitrary context, x is an SDCL variable, M, N are SDCL terms and A, B are SDCL types).*

$$(DT_l)\frac{x : A, \Gamma \vdash M : B}{\overline{\Gamma \vdash \overset{\lhd}{\lambda} x.M : B\backslash A}} \qquad (DT_r)\frac{\Gamma, x : A \vdash M : B}{\overline{\Gamma \vdash \overset{\rhd}{\lambda} x.M : B/A}}$$

where $(DT_l)'s$ *and* $(DT_r)'s$ B *is obtained by* ${\scriptstyle(<)}\dfrac{x : A, \Gamma' \vdash N : C \quad \Gamma'' \vdash M : B\backslash C}{x : A, \Gamma', \Gamma'' \vdash M^{\lhd}N : B}$ $(\Gamma \equiv \Gamma', \Gamma'')$ *and* ${\scriptstyle(>)}\dfrac{\Gamma' \vdash M : B/C \quad \Gamma'', x : A \vdash N : C}{\Gamma', \Gamma'', x : A \vdash M^{\rhd}N : B}$ $(\Gamma \equiv \Gamma', \Gamma'')$*, respectively.*

Proof. ($\Uparrow$): If $\Gamma \vdash \overset{\lhd}{\lambda} x.M : B\backslash A$ is provable, then $x : A, \Gamma \vdash (\overset{\lhd}{\lambda} x.M)^{\lhd}x : B$ is provable as follows.

$$(<)\frac{(VAR)\dfrac{}{x : A \vdash x : A} \qquad \Gamma \vdash \overset{\lhd}{\lambda} x.M : B\backslash A}{x : A, \Gamma \vdash (\overset{\lhd}{\lambda} x.M)^{\lhd}x : B}$$

Then, by the subject reduction theorem of SDCL [5] and η-reduction, $x : A, \Gamma \vdash M : B$ is also provable. DT_r can be proved via the following deduction:

$$(>)\frac{\Gamma \vdash \overset{\rhd}{\lambda} x.M : B/A \qquad (VAR)\dfrac{}{x : A \vdash x : A}}{\Gamma, x : A \vdash (\overset{\rhd}{\lambda} x.M)^{\rhd}x : B}$$

($\Downarrow$): By induction on the depth of the proof of $x : A, \Gamma \vdash M : B$ (or $\Gamma, x : A \vdash M : B$). Suppose that the depth of the proof of $x : A, \Gamma \vdash M : B$ (or $\Gamma, x : A \vdash M : B$) is k. We will now show that $\Gamma \vdash \overset{\lhd}{\lambda} x.M : B\backslash A$ (or $\Gamma \vdash \overset{\rhd}{\lambda} x.M : B/A$) is provable.

(case 1 (DT_l)) $k = 1$: The rule used to prove $x : A, \Gamma \vdash M : B$ is either (VAR) or (CON), but the possibility of (CON) is excluded since the left side of $x : A, \Gamma \vdash M : B$ has one or more formulas. Thus $x : A, \Gamma \vdash M : B$ must be proved by (VAR) (i.e., $x : A \vdash x : A$). In this case, the lowest sequent of (DT_l) should be $\vdash \overset{\lhd}{\lambda} x.x : A\backslash A$. So, it is sufficient if the combinator $I_{\backslash}$ is proved.

$$(<)\frac{(\mathbf{K})\dfrac{}{\vdash \mathbf{K} : (A \times A)\backslash A} \qquad (<>)\dfrac{(\mathbf{K}_{\backslash})\dfrac{}{\vdash \mathbf{K}_{\backslash} : (A\backslash(A \times A))/A} \quad (\mathbf{S}_{\backslash})\dfrac{}{\vdash \mathbf{S}_{\backslash} : (A\backslash A)\backslash((A \times A)\backslash A) \times ((A \times A))/A)}}{\vdash \mathbf{S}^{\Diamond}_{\backslash}\mathbf{K}_{\backslash} : (A\backslash A)\backslash((A \times A)\backslash A)}}{\vdash \mathbf{S}^{\Diamond}_{\backslash}\mathbf{K}^{\lhd}_{\backslash}\mathbf{K} : A\backslash A}$$

(case 2 (DT_r)) $k = 1$: Similar to case 1, it is sufficient if the combinator $I_{/}$ is proved (i.e. $\vdash \overset{\rhd}{\lambda} x.x : A/A$).

$$(>)\frac{(<>)\dfrac{(\mathbf{K}_{/})\dfrac{}{\vdash \mathbf{K}_{/} : (A/(A \times A))\backslash A} \quad (\mathbf{S}_{/})\dfrac{}{\vdash \mathbf{S}_{/} : (A/A)/((A \times A)/A) \times ((A/(A \times A))\backslash A)}}{\vdash \mathbf{S}^{\Diamond}_{/}\mathbf{K}_{/} : (A/A)/((A \times A)/A)} \qquad (\mathbf{K})\dfrac{}{\vdash \mathbf{K} : (A \times A)/A}}{\vdash \mathbf{S}^{\Diamond}_{/}\mathbf{K}^{\rhd}_{/}\mathbf{K} : A/A}$$

(case 3 (DT_l)) $k > 1$: Suppose that if there exists a proof of $x : A, \Gamma \vdash M : B$ whose depth is less than $k-1$, $\Gamma \vdash \overset{\triangleleft}{\lambda} x.M : B\backslash A$ is provable (IH). The proof of $x : A, \Gamma \vdash M : B$ whose depth is k must take the following form.

$$(<)\dfrac{x : A, \Gamma^{'} \vdash N : C \quad \Gamma^{''} \vdash M : B\backslash C}{x : A, \Gamma^{'}, \Gamma^{''} \vdash M^{\triangleleft}N : B} \ (\Gamma \equiv \Gamma^{'}, \Gamma^{''})$$

Then, $\Gamma^{'}, \Gamma^{''} \vdash \overset{\triangleleft}{\lambda} x.(M \overset{\triangleleft}{} N) : B\backslash A$ is provable as follows:

$$(<)\dfrac{(IH)\dfrac{x : A, \Gamma^{'} \vdash N : C}{\Gamma^{'} \vdash \lambda^{\triangleleft}x.N : C\backslash A} \quad (<>)\dfrac{(<>)\dfrac{\Gamma^{''} \vdash M : B\backslash C \quad (\mathbf{K}_{/})\dfrac{}{\vdash \mathbf{K}_{/} : (B\backslash C/A)\mathbin{)\!(}(B\backslash C)}}{\Gamma^{''} \vdash \mathbf{K}_{/}^{\diamond} M : B\backslash C/A} \quad (\mathbf{S}_{\backslash})\dfrac{}{\vdash \mathbf{S}_{\backslash} : (B\backslash A)\backslash(C\backslash A)\mathbin{)\!(}(B\backslash C/A)}}{\Gamma^{''} \vdash \mathbf{S}_{\backslash}^{\diamond}(\mathbf{K}_{/}^{\diamond} M) : (B\backslash A)\backslash(C\backslash A)}}{\Gamma^{'}, \Gamma^{''} \vdash \mathbf{S}_{\backslash}^{\diamond}(\mathbf{K}_{/}^{\diamond} M)^{\triangleleft}(\lambda^{\triangleleft}x.N) : B\backslash A}$$

Note that, due to the condition that $x \in fv(N)$, the proofs of $x : A, \Gamma \vdash M : B$ cannot take the following form:

$$(<)\dfrac{\vdash N : C \quad x : A, \Gamma \vdash M : B\backslash C}{x : A, \Gamma \vdash M \overset{\triangleleft}{} N : B}$$

(case 4 (DT_r)) $k > 1$: Suppose that if there exists a proof of $\Gamma, x : A \vdash M : B$ whose depth is less than $k-1$, $\Gamma \vdash \overset{\triangleright}{\lambda} x.M : B/A$ is provable (IH). The proof of $\Gamma, x : A \vdash M : B$ whose depth is k must take the following form:

$$(>)\dfrac{\Gamma^{'} \vdash M : B/C \quad \Gamma^{''}, x : A \vdash N : C}{\Gamma^{'}, \Gamma^{''}, x : A \vdash M \overset{\triangleright}{} N : B} \ (\Gamma \equiv \Gamma^{'}, \Gamma^{''})$$

Then $\Gamma^{'}, \Gamma^{''} \vdash \overset{\triangleright}{\lambda} x.M \overset{\triangleright}{} N$ is provable as follows:

$$(>)\dfrac{(<>)\dfrac{(<>)\dfrac{\Gamma' \vdash M : B/C \quad (\mathbf{K}_{\backslash})\dfrac{}{\vdash \mathbf{K}_{\backslash} : (B/C\backslash A)\mathbin{)\!(}(B/C)}}{\Gamma^{'} \vdash \mathbf{K}_{\backslash}^{\diamond} M : B/C\backslash A} \quad (\mathbf{S}_{/})\dfrac{}{\vdash \mathbf{S}_{/} : (B/A)/(C/A)\mathbin{)\!(}(B/C\backslash A)}}{\Gamma^{'} \vdash \mathbf{S}_{/}^{\diamond}(\mathbf{K}_{\backslash}^{\diamond} M) : (B/A)/(C/A)} \quad (IH)\dfrac{\Gamma^{''}, x : A \vdash N : C}{\Gamma^{''} \vdash \lambda^{\triangleright}x.N : C/A}}{\Gamma^{'}, \Gamma^{''} \vdash \mathbf{S}_{/}^{\diamond}(\mathbf{K}_{\backslash}^{\diamond} M)^{\triangleright}(\lambda^{\triangleright}x.N) : B/A}$$

Note that, due to the condition that $x \in fv(N)$, the proofs of $\Gamma, x : A \vdash M : B$ cannot take the following form:

$$(>)\dfrac{\Gamma, x : A \vdash M : B/C \quad \vdash N : C}{\Gamma, x : A \vdash M \overset{\triangleright}{} N : B}$$

Theorem 5 (Complex NP constraint in SDCL). *If the judgment $\Gamma_k, \ldots, \Gamma_1 \vdash \overset{\triangleright}{\lambda} x.(M_k \overset{\triangleright}{} \cdots \overset{\triangleright}{} (M_1 \overset{\triangleright}{} x)) : X_k/X_1$ is proved by (DT_r), then it is deduced from the following sequence of premises:*

$$\Gamma_k \vdash M_k : X_{k+1}/X_k, \cdots, \Gamma_1 \vdash M_1 : X_2/X_1$$

Proof. By induction on k.

Case $k = 1$:

$$
{\scriptstyle (>)}\dfrac{{\scriptstyle (<>)}\dfrac{{\scriptstyle (<>)}\dfrac{\Gamma_1 \vdash M_1 : X_2/X_1 \qquad {\scriptstyle (\mathbf{K}_\backslash)}\dfrac{}{\vdash \mathbf{K}_\backslash : (X_2/X_1\backslash X_1)\between(X_2/X_1)}}{\Gamma_1 \vdash \mathbf{K}^{\Diamond}_\backslash M_1 : X_2/X_1\backslash X_1} \qquad {\scriptstyle (\mathbf{S}_/)}\dfrac{}{\vdash \mathbf{S}_/ : (X_2/X_1)/(X_1/X_1)\between(X_2/X_1\backslash X_1)}}{\Gamma_1 \vdash \mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M) : (X_2/X_1)/(X_1/X_1)} \qquad {\scriptstyle (\mathbf{I}_/)}\dfrac{}{\vdash \overset{\triangleright}{\lambda} x.x : X_1/X_1}}{\Gamma_1 \vdash \mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M)^{\triangleright}(\lambda^{\triangleright} x.N) : X_2/X_1 \quad (\equiv \overset{\triangleright}{\lambda} x.M_1 \overset{\triangleright}{x})}
$$

In $\mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M)^{\triangleright}(\lambda^{\triangleright} x.N)$, $\mathbf{K}^{\Diamond}_\backslash M$ must have the type $X_2/X_1\backslash X_1$, which means that M must be of the type X_2/X_1.

Case $k > 1$:

$$
{\scriptstyle (>)}\dfrac{{\scriptstyle (<>)}\dfrac{{\scriptstyle (<>)}\dfrac{\Gamma_k \vdash M_k : X_{k+1}/X_k \qquad {\scriptstyle (\mathbf{K}_\backslash)}\dfrac{}{\vdash \mathbf{K}_\backslash : (X_{k+1}/X_k\backslash X_1)\between(X_{k+1}/X_k)}}{\Gamma_k \vdash \mathbf{K}^{\Diamond}_\backslash M_k : X_{k+1}/X_k\backslash X_1} \qquad {\scriptstyle (\mathbf{S}_/)}\dfrac{}{\vdash \mathbf{S}_/ : (X_{k+1}/X_1)/(X_k/X_1)\between(X_{k+1}/X_k\backslash X_1)}}{\Gamma_k \vdash \mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M) : (X_{k+1}/X_1)/(X_k/X_1)} \qquad {\scriptstyle (IH)}\dfrac{\Gamma_{k-1} \vdash M_{k-1} : X_k/X_{k-1} \quad \cdots \quad \Gamma_1 \vdash M_1 : X_2/X_1}{\Gamma_{k-1}, \ldots, \Gamma_1 \vdash \lambda^{\triangleright} x.(M_{k-1} \overset{\triangleright}{\cdots} \overset{\triangleright}{(} M_1 \overset{\triangleright}{x})) : X_k/X_1}}{\Gamma_k, \ldots, \Gamma_1 \vdash \mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M_k)^{\triangleright}(\lambda^{\triangleright} x.(M_{k-1} \overset{\triangleright}{\cdots} \overset{\triangleright}{(} M_1 \overset{\triangleright}{x}))) : X_{k+1}/X_1 \quad (\equiv \overset{\triangleright}{\lambda} x.(M_k \overset{\triangleright}{\cdots} \overset{\triangleright}{(} M_1 \overset{\triangleright}{x})))}
$$

Similar to the case of $k = 1$, in $\mathbf{S}^{\Diamond}_/ (\mathbf{K}^{\Diamond}_\backslash M_k)^{\triangleright}(\lambda^{\triangleright} x.(M_{k-1} \overset{\triangleright}{\cdots} \overset{\triangleright}{(} M_1 \overset{\triangleright}{x})))$, $\mathbf{K}^{\Diamond}_\backslash M_k$ must have the type $X_{k+1}/X_k\backslash X_1$, which means that M_k must be of the type X_{k+1}/X_k.

Therefore, the premises must be of the form $\Gamma_{k-1} \vdash M_{k-1} : X_k/X_{k-1} \cdots \Gamma_1 \vdash M_1 : X_2/X_1$: in other words, their "slashes" must have the same direction.

We claim that this is the source of the complex NP constraint. In other words, the assumption that Universal Grammar is equipped with only a subset of directional combinators [1] derives the consequence that the operation of extraction (i.e., the application of lambda abstraction) is available only when the premises (i.e., the phrases) are of functional types whose "slashes" have the same direction.

5 Structural Rules

As is implied in the proof of DT, the structural rules relying on DT in Hilbert-style proof systems inherit its constraints.

Definition 6 (Structural rules in Hilbert-style proof systems). *The following rules are derivable in Hilbert-style proof systems, where Γ, Δ are arbitrary sequences and A, B are arbitrary logical formulas.*

$$
(w_l)\dfrac{\Gamma \vdash M : B}{x : A, \Gamma \vdash M : B} \qquad (w_r)\dfrac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}
$$

$$
(c_l)\dfrac{x : A, x : A, \Gamma \vdash M : B}{x : A, \Gamma \vdash M : B} \qquad (c_r)\dfrac{\Gamma, x : A, x : A \vdash M : B}{\Gamma, x : A \vdash M : B}
$$

$$
(e)\dfrac{\Delta, y : B, x : A, \Gamma \vdash M : C}{\Delta, x : A, y : B, \Gamma \vdash M : C}
$$

On the other hand, in SDCL, only the following rules are derivable.

Definition 7 (Structural rules in SDCL). *The following rules are derivable in SDCL, where Γ, Δ are arbitrary sequences and A, B are arbitrary logical formulas.*

$$
{\scriptstyle (w_l)}\dfrac{\Gamma \vdash M : B}{x : A, \Gamma \vdash M : B}
\qquad
{\scriptstyle (w_r)}\dfrac{\Gamma \vdash M : B}{\Gamma, x : A \vdash M : B}
$$

$$
{\scriptstyle (e_h)}\dfrac{\Delta, y : B, x : A, \Gamma \vdash M : C}{y : B, \Delta, x : A, \Gamma \vdash M : C}
\qquad
{\scriptstyle (e_e)}\dfrac{\Delta, y : B, x : A, \Gamma \vdash M : C}{\Delta, y : B, \Gamma, x : A \vdash M : C}
$$

Proof. (Weakening)

$$
{\scriptstyle (<)}\dfrac{{\scriptstyle (VAR)}\dfrac{}{x : A \vdash x : A} \qquad {\scriptstyle (<>)}\dfrac{\Gamma \vdash M : B \qquad {\scriptstyle (\mathbf{K}_{\backslash})}\dfrac{}{\vdash \mathbf{K}_{\backslash}^{\Diamond} M : B\backslash A \times B}}{\Gamma \vdash (\mathbf{K}_{\backslash}^{\Diamond} M)^{\triangleleft} x : B\backslash A}}{x : A, \Gamma \vdash M : B}
$$

$$
{\scriptstyle (<)}\dfrac{{\scriptstyle (<>)}\dfrac{{\scriptstyle (\mathbf{K}_{/})}\dfrac{}{\vdash \mathbf{K}_{/} : B/A \times B} \qquad \Gamma \vdash M : B}{\Gamma \vdash \mathbf{K}_{/}^{\Diamond} M : B/A} \qquad {\scriptstyle (VAR)}\dfrac{}{x : A \vdash x : A}}{\Gamma, x : A \vdash (\mathbf{K}_{/}^{\Diamond} M)^{\triangleright} x : B}
$$

(Contraction)
Not provable because of the constraint on DT.

(Exchange)
We use the new combinator **C** to prove the Exchange rule. The following **C** is derivable from **SKB** that we set currently.($C \overset{def}{\equiv} \mathbf{S}(\mathbf{BBS})(\mathbf{KK})$ in combinatory logic)

$$
\begin{aligned}
\mathbf{C}_{/}^{\triangleright} x^{\triangleright} y^{\triangleright} z &\overset{def}{\equiv} \mathbf{S}_{/}^{\Diamond}(\mathbf{B}_{\backslash}^{\Diamond}\mathbf{B}_{/}^{\triangleleft}\mathbf{S}_{/}^{\triangleleft})^{\triangleright}(\mathbf{K}_{/}^{\Diamond}\mathbf{K}_{/})^{\triangleright} x^{\triangleright} y^{\triangleright} z \\
&\underset{CL}{\rightarrow} \mathbf{B}_{\backslash}^{\Diamond}\mathbf{B}_{/}^{\triangleleft}\mathbf{S}_{/}^{\triangleleft} x^{\triangleright}(\mathbf{K}_{/}^{\Diamond}\mathbf{K}_{/}^{\triangleright} x)^{\triangleright} y^{\triangleright} z \\
&\underset{CL}{\rightarrow} \mathbf{B}_{/}^{\triangleleft}(\mathbf{S}_{/}^{\triangleleft} x)^{\triangleright}(\mathbf{K}_{/}^{\Diamond}\mathbf{K}_{/}^{\triangleright} x)^{\triangleright} y^{\triangleright} z \\
&\underset{CL}{\rightarrow} \mathbf{S}_{/}^{\triangleleft} x^{\triangleright}(\mathbf{K}_{/}^{\Diamond}\mathbf{K}_{/}^{\triangleright} x^{\triangleright} y)^{\triangleright} z \\
&\underset{CL}{\rightarrow} x^{\triangleleft} z^{\triangleright}(\mathbf{K}_{/}^{\Diamond}\mathbf{K}_{/}^{\triangleright} x^{\triangleright} y^{\triangleright} z) \\
&\underset{CL}{\rightarrow} x^{\triangleleft} z^{\triangleright}(\mathbf{K}_{/}^{\triangleright} y^{\triangleright} z) \\
&\underset{CL}{\rightarrow} x^{\triangleleft} z^{\triangleright} y
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{C}_{\backslash}^{\triangleleft} x^{\triangleleft} y^{\triangleleft} z &\overset{def}{\equiv} \mathbf{S}_{\backslash}^{\Diamond}(\mathbf{B}_{/}^{\Diamond}\mathbf{B}_{\backslash}^{\triangleright}\mathbf{S}_{\backslash}^{\triangleright})^{\triangleleft}(\mathbf{K}_{\backslash}^{\Diamond}\mathbf{K}_{\backslash})^{\triangleleft} x^{\triangleleft} y^{\triangleleft} z \\
&\underset{CL}{\rightarrow} \mathbf{B}_{/}^{\Diamond}\mathbf{B}_{\backslash}^{\triangleright}\mathbf{S}_{\backslash}^{\triangleright} x^{\triangleleft}(\mathbf{K}_{\backslash}^{\Diamond}\mathbf{K}_{\backslash}^{\triangleleft} x)^{\triangleleft} y^{\triangleleft} z \\
&\underset{CL}{\rightarrow} \mathbf{B}_{\backslash}^{\triangleright}(\mathbf{S}_{\backslash}^{\triangleright} x)^{\triangleleft}(\mathbf{K}_{\backslash}^{\Diamond}\mathbf{K}_{\backslash}^{\triangleleft} x)^{\triangleleft} y^{\triangleleft} z \\
&\underset{CL}{\rightarrow} \mathbf{S}_{\backslash}^{\triangleright} x^{\triangleleft}(\mathbf{K}_{\backslash}^{\Diamond}\mathbf{K}_{\backslash}^{\triangleleft} x^{\triangleleft} y)^{\triangleleft} z \\
&\underset{CL}{\rightarrow} x^{\triangleright} z^{\triangleleft}(\mathbf{K}_{\backslash}^{\Diamond}\mathbf{K}_{\backslash}^{\triangleleft} x^{\triangleleft} y^{\triangleleft} z) \\
&\underset{CL}{\rightarrow} x^{\triangleright} z^{\triangleleft}(\mathbf{K}_{\backslash}^{\triangleleft} y^{\triangleleft} z) \\
&\underset{CL}{\rightarrow} x^{\triangleright} z^{\triangleleft} y
\end{aligned}
$$

The above proof indicates that the combinator **C** in SDCL with the **S** and **K** operators in the current setting must be one of the following forms:

$$\begin{array}{l}\mathbf{C}_{/} : (X\backslash Z/Y)/(X/Y/Z)\\ \mathbf{C}_{\backslash} : (X/Z\backslash Y)\backslash(X\backslash Y\backslash Z)\end{array}$$

Then, we start the proof of the Exchange rule (the terms are omitted):

<u>case 1</u> $\Delta \overset{def}{\equiv} B_1, \cdots, B_n$

$$(def\Delta)\dfrac{(DT_l*)\dfrac{(<)\dfrac{(DT_l*)\dfrac{(def\Delta)\dfrac{\Delta, B, A, \Gamma \vdash C}{B_1, \cdots, B_n, B, A, \Gamma \vdash C}}{\Gamma \vdash C\backslash B_1\backslash \cdots \backslash B_n\backslash B\backslash A} \qquad (C_{\backslash})\dfrac{}{\vdash (C\backslash B_1\backslash \cdots \backslash B_n/A\backslash B)\backslash(C\backslash B_1\backslash \cdots \backslash B_n\backslash B\backslash A)}}{\Gamma \vdash C\backslash B_1\backslash \cdots \backslash B_n/A\backslash B}}{B_1, \cdots, B_n, B, \Gamma, A \vdash C}}{\Delta, B, \Gamma, A \vdash C}$$

<u>case 2</u> $\Gamma \overset{def}{\equiv} A_1, \cdots, A_n$

$$(def\Gamma)\dfrac{(DT_r*)\dfrac{(>)\dfrac{(C_{/})\dfrac{}{\vdash (C/A_n/\cdots/A_1\backslash B/A)/(C/A_n/\cdots/A_1/A/B)} \qquad (DT_r*)\dfrac{(def\Gamma)\dfrac{\Delta, B, A, \Gamma \vdash C}{\Delta, B, A, A_1, \cdots, A_n \vdash C}}{\Delta \vdash C/A_n/\cdots/A_1/A/B}}{\Delta \vdash C/A_n/\cdots/A_1\backslash B/A}}{B, \Delta, A, A_1, \cdots, A_n \vdash C}}{B, \Delta, A, \Gamma \vdash C}$$

The result indicates that we can use the Exchange rule only in the case where an argument is moved to the head or end of the row of formulas. Thus, we cannot exchange arguments freely.

6 Power of Type-Raising in DT

In Section 2 we compared the grammars with and without the type-raising rule with respect to the restriction on extractability. The type-raising rules in CCG are defined as follows.

$$(>T)\dfrac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{C}*_{\backslash}^{\lhd\!\rhd} M : B/(B\backslash A)} \qquad (<T)\dfrac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{C}*_{/}^{\lhd\!\rhd} M : B\backslash(B/A)}$$

These rules can be regarded as the application of the combinators $\mathbf{C}*_{/}(\equiv \mathbf{C}_{/}^{\lhd\!\rhd}\mathbf{I}_{/})$ and $\mathbf{C}*_{\backslash}(\equiv \mathbf{C}_{\backslash}^{\lhd\!\rhd}\mathbf{I}_{\backslash})$, which are derived in SDCL as follows:

$$(<>)\dfrac{\mathbf{C}_{/} : X\backslash(X/Y)/Y/(X/Y/(X/Y)) \quad \mathbf{I}_{/} : (X/Y)/(X/Y)}{\mathbf{C}_{/}^{\lhd\!\rhd}\mathbf{I}_{/} : X\backslash(X/Y)/Y}$$

$$(<>)\dfrac{\mathbf{C}_{\backslash} : (X/(X\backslash Y)\backslash Y)\backslash(X\backslash Y\backslash(X\backslash Y)) \quad \mathbf{I}_{\backslash} : (X\backslash Y)\backslash(X\backslash Y)}{\mathbf{C}_{\backslash}^{\lhd\!\rhd}\mathbf{I}_{\backslash} : X/(X\backslash Y)\backslash Y}$$

Now we can answer the question why the addition of the type-raising rule allows the grammar to violate the complex NP constraint. Recall that the extraction is blocked if the sequence of phrases contains a phrase whose "slash" has a different direction:

$$\frac{\text{John}}{T/(T\backslash NP)} \quad \frac{\text{met}}{S\backslash NP/NP} \quad \frac{\text{a}}{NP/N} \quad {}_{*}\frac{\dfrac{\text{boy}}{N} \quad {\scriptstyle(>B)}\dfrac{\dfrac{\text{who}}{N\backslash N/(S\backslash NP)} \quad \dfrac{\text{loved}}{S\backslash NP/NP}}{N\backslash N/NP}}{}$$

However, this derivation is remedied when the combinators $\mathbf{C}*_{/}$ and $\mathbf{C}*_{\backslash}$ are available as follows:

$${\scriptstyle(>B)}\frac{\dfrac{\text{John}}{T/(T\backslash NP)} \quad {\scriptstyle(>B)}\dfrac{\dfrac{\text{met}}{S\backslash NP/NP} \quad {\scriptstyle(>B)}\dfrac{\dfrac{\text{a}}{NP/N} \quad {\scriptstyle(>B)}\dfrac{{\scriptstyle(>T)}\dfrac{\dfrac{\text{boy}}{N}}{N/(N\backslash N)} \quad {\scriptstyle(>B)}\dfrac{\dfrac{\text{who}}{N\backslash N/(S\backslash NP)} \quad \dfrac{\text{loved}}{S\backslash NP/NP}}{N\backslash N/NP}}{N/NP}}{NP/NP}}{S\backslash NP/NP}}{S/NP}$$

Generally, the addition of $\mathbf{C}*_{/}$ and $\mathbf{C}*_{\backslash}$ allows us to consider a third case in the proof of DT in SDCL, when $k > 1$:

<u>(case 5-1 (DT_l))</u>

$$(>)\frac{x : A, \Gamma^{'} \vdash M : B/C \quad \Gamma^{''} \vdash N : C}{x : A, \Gamma^{'}, \Gamma^{''} \vdash M^{\triangleright}N : B}$$

$$\Longrightarrow (<)\frac{x : A, \Gamma' \vdash M : B/C \quad {\scriptstyle(<T)}\dfrac{\Gamma^{''} \vdash N : C}{\Gamma^{''} \vdash \mathbf{C}*_{/}^{\lhd\!\rhd}N : B\backslash(B/C)}}{x : A, \Gamma^{'}, \Gamma^{''} \vdash (\mathbf{C}*_{/}^{\lhd\!\rhd}N)^{\triangleleft}M : B}$$

<u>(case 5-2 (DT_r))</u>

$$(<)\frac{\Gamma^{'} \vdash N : C \quad \Gamma^{''}, x : A \vdash M : B\backslash C}{\Gamma^{'}, \Gamma^{''}, x : A \vdash M^{\triangleleft}N : B}$$

$$\Longrightarrow (>)\frac{{\scriptstyle(>T)}\dfrac{\Gamma^{'} \vdash N : C}{\Gamma^{'} \vdash \mathbf{C}*_{\backslash}^{\lhd\!\rhd}N : B/(B\backslash C)} \quad \Gamma^{''}, x : A \vdash M : B\backslash C}{\Gamma^{'}, \Gamma^{''}, x : A \vdash (\mathbf{C}*_{\backslash}^{\lhd\!\rhd}N)^{\triangleright}M : B}$$

Therefore, DT that is admissible in SDCL has a restriction due to 1) the last rule being used to prove the upper sequent and 2) the constraint on the slash-directions. For example, when one or more formulas have only backslashes (\) in the whole proof, the rules we can use are (<) or (DT_l). Also thereafter, we can use only (<) or (DT_l) because the proof has only backslashes (the other slash (/) is similar). In addition, we cannot use crossed composition ([2]) because of the constraints on the slash directions.

This additional generality in the proof of DT corresponds to the additional constraction in the encoding of lambda abstraction as follows:

Definition 8 (Abstraction in SDCL with $\mathbf{C}{*}_/$ and $\mathbf{C}{*}_\backslash$). *For every term M and every variable x, terms of the form $\overset{\triangleright}{\lambda}x.M$ and $\overset{\triangleleft}{\lambda}x.M$ are defined recursively as follows:*

$$
\begin{array}{llcll}
(1/) & \overset{\triangleright}{\lambda}x.x & \overset{def}{\equiv} & \mathbf{I}_/ & \\
(1\backslash) & \overset{\triangleleft}{\lambda}x.x & \overset{def}{\equiv} & \mathbf{I}_\backslash & \\
(2/) & \overset{\triangleright}{\lambda}x.M\,{}^{\triangleright}N & \overset{def}{\equiv} & \mathbf{S}_/\,{}^{\triangleleft\triangleright}(\mathbf{K}_\backslash\,{}^{\triangleleft\triangleright}M)\,{}^{\triangleright}(\overset{\triangleright}{\lambda}x.N) & (x \notin fv(M), x \in fv(N)) \\
(2\backslash) & \overset{\triangleleft}{\lambda}x.M\,{}^{\triangleleft}N & \overset{def}{\equiv} & \mathbf{S}_\backslash\,{}^{\triangleleft\triangleright}(\mathbf{K}_/\,{}^{\triangleleft\triangleright}M)\,{}^{\triangleleft}(\overset{\triangleleft}{\lambda}x.N) & (x \notin fv(M), x \in fv(N)) \\
(3/) & \overset{\triangleright}{\lambda}x.M\,{}^{\triangleleft}N & \overset{def}{\equiv} & \mathbf{S}_/\,{}^{\triangleleft\triangleright}(\mathbf{K}_\backslash\,{}^{\triangleleft\triangleright}(\mathbf{C}{*}_/\,{}^{\triangleleft\triangleright}N))\,{}^{\triangleright}(\overset{\triangleright}{\lambda}x.M) & (x \in fv(M), x \notin fv(N)) \\
(3\backslash) & \overset{\triangleleft}{\lambda}x.M\,{}^{\triangleright}N & \overset{def}{\equiv} & \mathbf{S}_\backslash\,{}^{\triangleleft\triangleright}(\mathbf{K}_/\,{}^{\triangleleft\triangleright}(\mathbf{C}{*}_\backslash\,{}^{\triangleleft\triangleright}N))\,{}^{\triangleleft}(\overset{\triangleleft}{\lambda}x.M) & (x \in fv(M), x \notin fv(N))
\end{array}
$$

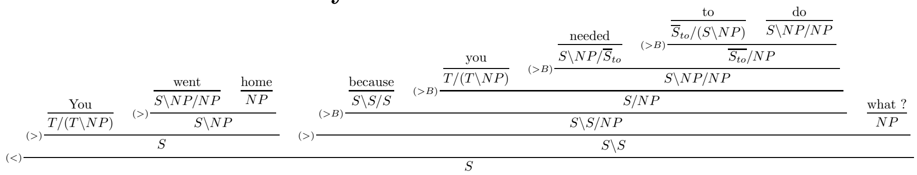
Therefore, the SDCL system with $\mathbf{C}{*}_/$ and $\mathbf{C}{*}_\backslash$ is a different system from that without them: in the former case, the restriction of DT, and hence extraction, is relaxed to the point where the premises (i.e., the phrases) do not have to have "slashes" of the same direction, nor have to be a functional type in the first place. The only restriction is that in applying (DT_l), the rightmost element must be of the type X/Y, and in applying (DT_r), the leftmost element must be of the type $X\backslash Y$. In other words, only an element on the edge of phrases can be extracted.

7 On Adjunct and Subject Islands

We can give explanations about other kinds of wh-movement. As we mentioned in Theorem 5, "slashes" of each words must have the same direction. In this way, we can explain 'Adjunct islands' and 'Subject islands'.

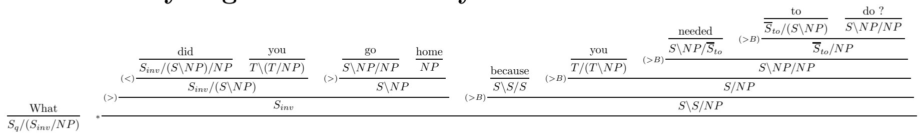
An adjunct islands is a kind of island formed from an adjunct clause. The following sentence is an instance of the sentence including adjunct phrase:

You went home because you needed to do what?



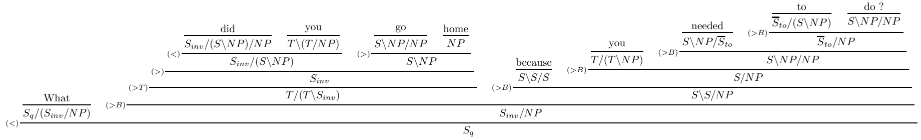
This sencence is acceptable and is also derivable in CCG. However, it is well known that extraction from an adjunct clause is not allowed in English (*Adjunct islands*), as illustrated below:

*** What did you go home because you needed to do?**



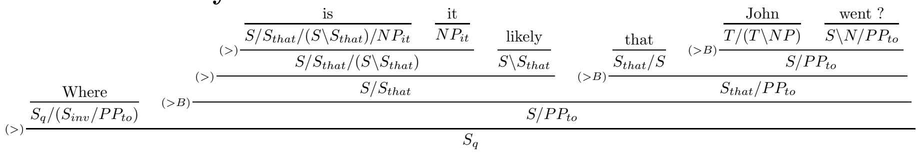
This sentence is unacceptable and is not derivable in CCG. However, if we allow a type-raising rule for CCG, the sentence becomes derivable in the following way:

*** What did you go home because you needed to do?**



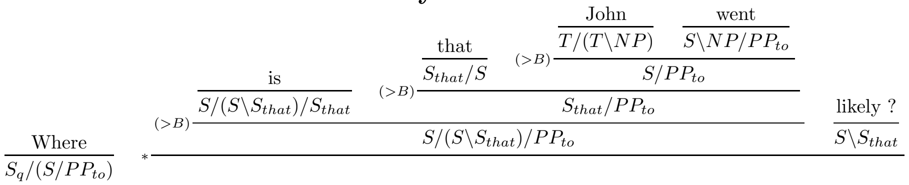
'*Subject islands*' are another example of where wh-movement is excluded. A subject island is a kind of island formed from a subject position. The following example illustrates a wh-phrase in subject position:

Where is it likely that John went?



This sentence is acceptable and is also derivable in CCG. However, it is well known that extraction from a subject clause is not allowed in English (Subject islands), as illustrated below:

*** Where is that John went likely?**



As we illustrated above, we can explain 'Adjunct islands' and 'Subject islands' if we have the constraint of Theorem 5. In general, extraction in CCG (without type-raising) is available only when the syntactic category of the relevant phrases match one of the following forms:

$$(>B)\times n\,\frac{X_1/X_2 \quad X_2/X_3 \quad \cdots \quad X_{n+1}/X_{n+2}}{X_1/X_{n+2}}$$

$$(<B)\times n\,\frac{X_{n+1}\backslash X_{n+2} \quad \cdots \quad X_2\backslash X_3 \quad X_1\backslash X_2}{X_1\backslash X_{n+2}}$$

It has turned out that our discussion above of the complex-NP constraint applies as an account for other restrictions on wh-movement.

8 Conclusion

A certain instance of CCG can be re-formulated as a certain instance of SDCL that has a subset of directional combinators (i.e., *subdirectionality*). This reformulation of CCG by means of SDCL allows us to regard extraction as DT in the subdirectional version of a Hilbert-style proof system, and by Curry-Howard correspondence, as abstraction in the subdirectional version of CL. Then, the subdirectionality of CCG poses a restriction on the applicability of DT/abstraction, which gives us a fundamental explanation as to why the complex NP constraint exists, as a result of subdirectionality of the chosen combinator (especially **S** in this setting). This is a deeper explanation than that which merely states that NP is somehow a boundary node for a wh-movement, because this explanation connects the notion of extractability to the primitive combinators that also serves as the basis of combinatory rules (i.e., *merge* operations).

Moreover, we have revealed the relationship between extractability and the presence/absence of $\mathbf{C}*_{/}$ and $\mathbf{C}*_{\backslash}$, namely, the type-raising operation. In this operation, the presence of $\mathbf{C}*_{/}$ and $\mathbf{C}*_{\backslash}$ drastically relax the restriction on extractability to the extent that the complex NP constraint does not hold, and this derives from the fact that the addition of $\mathbf{C}*_{/}$ and $\mathbf{C}*_{\backslash}$ to the grammar gives rise to the additional generality to DT/abstraction.

On the whole, we demonstrated that the re-formulation of CCG in terms of SDCL allows us to reduce the existence of some linguistic constraints to some formal properties of the subdirectional logic. In other words, there are linguistic constraints that can be captured well as a formal consequence of the subdirectionality of a logic we regard our grammar to correspond to. We believe that this argument highlights another advantage of the categorial/logical view of natural language syntax.

Appendix

A Subdirectional Combinatory Logic

In Subdirectional Combinatory Logic (SDCL), the two directions of functional applications are distinguished. For example, the combinator $\mathbf{K} : A \rightarrow (B \rightarrow A)$ in Combinatory Logic (CL) corresponds to the following 4 *directional* combinators:

$$\mathbf{K}_{//} : (A/B)/A \qquad \mathbf{K}_{\backslash/} : (A\backslash B)/A \qquad \mathbf{K}_{/\backslash} : (A/B)\backslash A \qquad \mathbf{K}_{\backslash\backslash} : (A\backslash B)\backslash A$$

(other combinators are defined in a similar way)

Thus, the logic which is expanded from CL and can describe order between natural language by giving directions to combinators is called SDCL.

A.1 Syntax, Axioms, Type Assignment Rules and Underspecified Notations

In SDCL, syntax, axioms, type assignment rules and underspecified notations are defined as follows:

Syntax:

$$\begin{array}{lll} \text{type } \tau & ::= \gamma \mid \tau/\tau \mid \tau\backslash\tau & (\gamma \text{ is a base type}) \\ \text{term } \Lambda & ::= x \mid c \mid \Lambda^{\triangleright}\Lambda \mid \Lambda^{\triangleleft}\Lambda & (c \text{ is a directional combinator}) \end{array}$$

Axioms:

$$(VAR)\frac{}{x : A \vdash x : A} \qquad (CON)\frac{}{\vdash c : A}$$

Typing rules:

$$(>)\frac{\Gamma \vdash M : A/B \quad \Delta \vdash N : B}{\Gamma, \Delta \vdash M^{\triangleright}N : A} \qquad (<)\frac{\Delta \vdash N : B \quad \Gamma \vdash M : A\backslash B}{\Delta, \Gamma \vdash M^{\triangleleft}N : A}$$

Underspecified notations:

$$\begin{array}{rcl} \tau \between \sigma & \overset{def}{\equiv} & \tau/\sigma \text{ or } \tau\backslash\sigma \\ M^{\triangleleft\!\triangleright}N & \overset{def}{\equiv} & M^{\triangleright}N \text{ or } M^{\triangleleft}N \\ (<>) & \overset{def}{\equiv} & (<) \text{ or } (>) \end{array}$$

A.2 Dependency between Combinators

In [1], the combinators **S** and **B** are chosen as below, following [7].

$$\mathbf{S}_/ : (A/C)\backslash(B/C)\between(A\backslash B/C) \quad \mathbf{S}_\backslash : (A\backslash C)/(B\backslash C)\between(A/B\backslash C)$$
$$\mathbf{B}_/ : (A/C)/(B/C)\between(A/B) \quad \mathbf{B}_\backslash : (A\backslash C)\backslash(B\backslash C)\between(A\backslash B)$$

[1] claims that Universal Grammar is not equipped with the combinator **K**, but for purely theoretical reasons, we use the combinator **K** of the following form in order to show the dependencies between combinators:

$$\mathbf{K}_/ : (A/B)\between A \quad \mathbf{K}_\backslash : (A\backslash B)\between A$$

In typed-CL, it is known that the combinator **B** is derivable from the combinators **S** and **K**. However, the combinator **S** in [1], chosen for linguistic reasons, does not derive **B** above (Figure 1). Instead, again for purely theoretical reasons, we use the combinator **S** of the following form:

$$\mathbf{S}_/ : (A/C)/(B/C)\between(A/B\backslash C) \quad \mathbf{S}_\backslash : (A\backslash C)\backslash(B\backslash C)\between(A\backslash B/C)$$

Then, we obtain $\mathbf{B}_/$ from $\mathbf{S}_/$and $\mathbf{K}_\backslash$ (Figure 2).

$$\scriptstyle (<)\dfrac{\mathbf{K}_/ = ((A/B)/C)/(A/B) \qquad (<>)\dfrac{(<>)\dfrac{(\mathbf{A}/\mathbf{C})/(\mathbf{B}/\mathbf{C})\backslash((\mathbf{A}/\mathbf{B})/\mathbf{C}) \quad \mathbf{K}_/ = ((A/C)/(B/C)\backslash(A/B)/C)/(A/B)\between(A/C)/(B/C)\backslash((A/B)/C)}{((A/C)/(B/C)\backslash(A/B)/C)/(A/B)} \quad \mathbf{S}_/ = ((A/C)/(B/C)/(A/B))\backslash(((A/B)/C)/(A/B))\backslash(((A/B)/C)/(A/B))\between((A/C)/(B/C)\backslash(A/B)/C)/(A/B)}{((A/C)/(B/C)/(A/B))\backslash(((A/B)/C)/(A/B))\backslash(((A/B)/C)/(A/B))}}{B_/ = (A/C)/(B/C)/(A/B)}$$

Fig. 1. Derivation of $B_/$ (failed)

$$\dfrac{\dfrac{\dfrac{\mathbf{S}_/ = (A/C)/(B/C)/((A/B)\backslash C) \quad \mathbf{K}_\backslash = ((A/C)/(B/C)/(A/B)\backslash C)\backslash(A/B){/\!\!\!\backslash}(A/C)/(B/C)/((A/B)\backslash C)}{((A/C)/(B/C)/(A/B)\backslash C)\backslash(A/B)}(<>) \quad \mathbf{S}_/ = ((A/C)/(B/C)/(A/B))/(((A/B)\backslash C)/(A/B)){/\!\!\!\backslash}((A/C)/(B/C)/(A/B)\backslash C)\backslash(A/B)}{((A/C)/(B/C)/(A/B))/(((A/B)\backslash C)/(A/B))}(<>) \quad \mathbf{K}_\backslash = ((A/B)\backslash C)/(A/B)}{B_/ = (A/C)/(B/C)/(A/B)}(>)$$

Fig. 2. Derivation of $B_/$

A.3 Reduction Rules

The reduction rules in SDCL are defined as follows:

$$\begin{array}{llll}
\mathbf{B}_/^{\triangleleft\!\triangleright} f^{\triangleright} g^{\triangleright} x \underset{CL}{\rightarrow} f^{\triangleright}(g^{\triangleright}x) & & \mathbf{B}_\backslash^{\triangleleft\!\triangleright} f^{\triangleleft} g^{\triangleleft} x \underset{CL}{\rightarrow} f^{\triangleleft}(g^{\triangleleft}x) \\
\mathbf{S}_/^{\triangleleft\!\triangleright} f^{\triangleright} g^{\triangleright} x \underset{CL}{\rightarrow} (f^{\triangleleft}x)^{\triangleright}(g^{\triangleright}x) & & \mathbf{S}_\backslash^{\triangleleft\!\triangleright} f^{\triangleleft} g^{\triangleleft} x \underset{CL}{\rightarrow} (f^{\triangleright}x)^{\triangleleft}(g^{\triangleleft}x) \\
\mathbf{K}_/^{\triangleleft\!\triangleright} x^{\triangleright} y \underset{CL}{\rightarrow} x & & \mathbf{K}_\backslash^{\triangleleft\!\triangleright} x^{\triangleleft} y \underset{CL}{\rightarrow} x
\end{array}$$

It is proved that the reduction rules of SDCL enjoy formal properties such as subject reduction, the Church-Rosser property and strong normalization. See [5] for the detailed proofs.

References

1. Bekki, D.: Combinatory categorial grammar as a substructural logic — preliminary remarks —. In: The Seventh International Workshop on Logic and Engineering of Natural Language Semantics (LENLS 7), JSAI International Symposia on AI 2010. pp. 70–83. Campus Innovation Center, Tokyo (2010)
2. Hindley, J.R., Seldin, J.P.: Lambda-Calculus and Combinators: an Introduction. Cambridge University Press, Cambridge (2008)
3. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65, 154–169 (1958)
4. Morrill, G.V.: Type Logical Grammar. Kluwer Academic Publishers, Dordrecht (1994)
5. Ozaki, H., Bekki, D.: Computational properties of subdirectional combinatory logic. Tech. rep., Ochanomizu University, OCHA-IS 10-2, February 7 (2011)
6. Ross, J.R.: Constraints on Variables in Syntax. Unpublished ph.d. dissertation. MIT (1967)
7. Steedman, M.J.: The Syntactic Process (Language, Speech, and Communication). The MIT Press, Cambridge (2000)

Agnostic Possible Worlds Semantics

Andrew Plummer and Carl Pollard

The Ohio State University, Columbus, OH 43210, USA
{plummer,pollard}@ling.ohio-state.edu

Abstract. Working within standard classical higher-order logic, we propose a possible worlds semantics (PWS) which combines the simplicity of the familiar Montague semantics (MS), in which propositions are sets of worlds, with the fine-grainedness of the older but less well-known **tractarian** semantics (TS) of Wittgenstein and C.I. Lewis, wherein worlds are maximal consistent sets of propositions. The proposed **agnostic** PWS makes neither montagovian nor tractarian ontological commitments, but is consistent with (and easily extensible to) either alternative (among many others). It is technically straightforward and, we believe, capable of everything linguists need PWS to do, such as interfacing with a logical grammar and serving as a basis for dynamic semantics.

Keywords: propositions, possible worlds, maximal consistent sets, Montague semantics, tractarian semantics.

1 Introduction

1.1 Montague Semantics

When Montague [1974] pioneered the systematic application of mathematical logic to the analysis of natural language meaning, he borrowed one key idea from Kripke [1963], and another from Carnap [1947]. From Kripke came the assumption of "an arbitrary set K of 'possible worlds' and a function $\Phi(P, H)$ assigning to each proposition [= atomic formula] P a truth-value in the world H", in contradistinction to the earlier notion of a possible world as a maximal consistent set of propositions (Wittgenstein [1921], C.I. Lewis [1923]) or formulas (Carnap [1947], Kripke [1959]). And from Carnap came the idea of a linguistic meaning as a Carnapian *intension*, a function whose domain is the set of possible worlds. In the case of a declarative sentence, the meaning—the proposition expressed by the sentence—is nothing more or less than (the characteristic function) of a set of these possible worlds, because Carnap followed Frege [1892] in assuming that the reference of a sentence is the truth value of the proposition that it expresses.

Together, these two ideas have as consequences that the set of propositions is a powerset algebra, the meanings of the English 'logic words' are just the familiar boolean operations on sets of worlds, and the centrally important notion of entailment is just the relation between them of subset inclusion. Because of its sheer simplicity and familiarity (skillfully illuminated in the influential

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 201–212, 2012.

textbook by Dowty et al. [1981]), and its early adoption by certain philosophers well-known to linguists, most notably Stalnaker [1976,1984], this style of PWS became, and remains, *the* mainstream framework for theorizing about natural language meaning in the linguistic semantics community. At the same time, it has long been recognized—at least as early as C.I. Lewis [1943] and Carnap [1947]—that treating sentence meanings as sets of worlds also has the troubling consequence that entailment is *antisymmetric*, i.e. that distinct sentences with the same truth conditions *mean* the same thing, the best-known aspect of what is more generally known as the 'granularity problem' (that distinct linguistic expressions with necessarily identical denotations have the same meaning).

1.2 Structured Meanings

The (independent) responses of Lewis and Carnap to this challenge were, in essence, that the meanings of sentences are not the intensions associated with them, but rather 'structured' objects with lexical meanings as parts. Since then, one version or other of this general approach has been embraced by a number of philosophers (e.g. D. Lewis [1970], Cresswell [1985], Soames [1987], King [1996,2007]). The various versions differ both with respect to what kinds of lexical meanings are embedded within the structures (e.g. whether they are intensional or extensional) and what kinds of structure they have (e.g. tuples, phrase markers, some version of Chomskian LF). And although structured-meaning approaches are broadly familiar to the linguistic semantics community, they seem not to have gained much of a foothold there, perhaps for lack of a canonical and accessible exposition with a degree of formal explicitness comparable to that of Dowty et al. [1981].

1.3 Propositions in Themselves

However, long before Carnap, Kripke, and Montague, there were well worked out conceptions of propositions as things in their own right (independent of sentences that might express them or conditions under which they might be true); and of worlds, not as unanalyzed primitives, but rather as certain sets of propositions (the *maximal consistent* ones). As early as 1837, Bolzano's notion of 'proposition in itself' (*Satz an sich*) embodied most of the key properties that present-day semanticists attribute to propositions:

a. They are expressed by declarative sentences.
b. They are the primary bearers of truth and falsity; a sentence is only secondarily, or derivatively, true or false, depending on what proposition it expresses.
c. They are the objects of the attitudes, i.e. they are the things that are known, believed, doubted, etc.
d. They are not linguistic.
e. They are not mental.

f. They are not located in space or time.

g. Sentences in different languages, or different sentences in the same language, can express the same proposition.

h. Two distinct propositions can entail each other.

1.4 Tractarian Semantics

Wittgenstein [1921] seems to have been the first to explicitly identify worlds with maximal consistent sets of facts, a characterization which we henceforth call **tractarian**. More precisely, the worlds of the *Tractatus* are maximal consistent assemblages of positive and negative facts. A fact (*Tatsache*), the closest counterpart in the *Tractatus* to our (or Bolzano's) propositions, consists of the (non)existence of a state of affairs (*Sachverhalt*); the term 'proposition' (*Satz*) is reserved for the linguistic entities that express (potential) facts, or equivalently, describe states of affairs.

More technically precise, but much less-known, is C.I. Lewis [1923], wherein a world is defined to be a 'system of facts' which is maximal in the sense of containing each fact or its 'contradictory'. In Lewis' terminology, 'facts' correspond to our propositions and the 'actual facts' of a system are those facts which belong to it ; while a 'system' is what would later come to be called a *proper filter* (a proper subset of the propositions, closed under conjunction and entailment). A significant advantage of Lewis' theory over Wittgenstein's is that it is not *atomistic*. That is, there is no requirement that there be a collection of 'basic facts', i.e. the ones expressed in the *Tractatus* system by elementary propositions (*Elementarsätze*).

More recent tractarian characterizations of possible worlds by philosophers include those of Adams [1974], Plantinga [1974], and Lycan [1979]. Within linguistics, there have been several logical theories of natural language meaning which take propositions to be primitives rather than sets of worlds, e.g. Thomason [1980], Muskens [2005], and Pollard [2008,2011], the last of which is explicitly tractarian.

As is the case with structured meanings, none of the propositions-as-primitives approaches, tractarian or not, has attracted much of a following among semanticists, perhaps because the philosophers in question have not been as closely associated with the linguistic community, or perhaps for want of a sufficiently accessible exposition of the underlying technicalia. Although we strongly believe that this robust tradition, alternative to both montagovian PWS and structured-meaning approaches, deserves a fuller hearing in the linguistic community, it is not our purpose here to provide such an exposition. Instead, we will propose a *weakening* of MS, a PWS which we believe possesses all the positive attributes of MS but without the fatal identification of propositions with sets of worlds and the concomitant antisymmetry of entailment. This **agnostic** PWS is a weakening in the sense that, if the type p of propositions is identified with the type w → t of sets of worlds, then the addition of a single axiom produces a theory equivalent to Montague semantics (or, more precisely, to Gallin's ([1975])

reformulation of MS within the simple theory of types). On the other hand, by adding different axioms instead, we can obtain TS, or a wide range of alternative semantic theories. In practical terms, working semanticists can just use the weak theory 'off the shelf', without taking on further ontological commitments that don't have any empirical consequences.

2 The Theory

2.1 Preliminaries

We work within the simple theory of types of Church [1940] as modified by Henkin [1950] with the addition of the axiom of truth-value extensionality, identifying truth value equality with biimplication.

We adopt the following notational conventions. Application terms are written $(f\ a)$, not $f(a)$. Application associates to the left, so $(f\ a\ b)$ abbreviates $((f\ a)\ b)$. Outermost parentheses are often dropped. We abbreviate multiple abstractions, e.g. $\lambda_{xy}.a$ for $\lambda_x.\lambda_y.a$. Functional types are written $A \to B$; and implication associates to the right, so $A \to B \to C$ abbreviates $A \to (B \to C)$. Some binary function symbols are written infix without comment, e.g. p and q, p entails q. The turnstyle '⊢' is used both for typing judgments of terms and for assertions of higher-order provability.

2.2 Types

Besides Church's types o and ι, here (following Montague) called e and t, we employ two additional basic types w (worlds) and p (propositions). We adopt the following type abbreviations:

a. $\mathrm{p}_{\mathit{0}} =_{\mathrm{def}} \mathrm{p}$
b. $\mathrm{p}_{n+\mathit{1}} =_{\mathrm{def}} \mathrm{e} \to \mathrm{p}_n$

2.3 Constants and Axioms

In MS, being true at is the relation between propositions (qua sets of worlds) and worlds such that p is true at w just in case $w \in p$; whereas in TS, it is the relation between propositions and worlds (*qua* maximal consistent sets of propositions) such that p is true at w just in case $p \in w$. In agnostic semantics, hereafter AS, by contrast, we only assume that being true at, denoted by the constant @, is *some relation or other* between propositions and worlds, without committing to whether it is membership, inverse membership, or something else altogether. Hence we have

$$\vdash @ : \mathrm{p} \to \mathrm{w} \to \mathrm{t} \text{ ('is true at')}$$

Axioms and definitions to be given below will then ensure that this relation has the two crucial properties that, for any world w and any propositions p and q:

a. the set of facts of w (i.e. the propositions which are true at w) form a maximal consistent set; and
b. p entails q iff q is true at every world where p is true.

We begin with the following abbreviations:

$$\mathsf{facts} =_{\text{def}} \lambda_{wp}.p@w$$
$$\mathsf{entails} =_{\text{def}} \lambda_{pq}.\forall_w.p@w \rightarrow q@w$$
$$\equiv\ =_{\text{def}} \lambda_{pq}.(p \ \mathsf{entails}\ q) \wedge (q \ \mathsf{entails}\ p) \text{ (‘equivalent to’)}$$

Thus facts w is the set of propositions true at w; p entails q just in case q is true at every world where p is; and p is (truth-conditionally) equivalent to q just in case p and q are mutually entailing.

The preceding three definitions could just as well have been written as axioms:

$$\vdash \forall_w.(\mathsf{facts}\ w) = \lambda_p.p@w$$
$$\vdash \forall_{pq}.(p \ \mathsf{entails}\ q) \leftrightarrow \forall_w.p@w \rightarrow q@w$$
$$\vdash \forall_{pq}.(p \equiv q) \leftrightarrow (p \ \mathsf{entails}\ q) \wedge (q \ \mathsf{entails}\ p)$$

Note that the last of these could be expressed equivalently as

$$\vdash \forall_{pq}.(p \equiv q) \leftrightarrow \forall_w.(p@w) = (q@w)$$

which will become relevant later when we generalize the notion of equivalence from propositions to meanings of other types.

Next, we introduce constants for the usual proposition-level connectives and quantifiers, as follows:

$\vdash \mathsf{truth} : \mathrm{p}$ (a necessary truth)

$\vdash \mathsf{falsity} : \mathrm{p}$ (a necessary falsehood)

$\vdash \mathsf{not} : \mathrm{p} \rightarrow \mathrm{p}$ (propositional negation)

$\vdash \ \mathsf{and}\ : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$ (propositional conjunction)

$\vdash \ \mathsf{or}\ : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$ (propositional disjunction)

$\vdash \ \mathsf{implies}\ : \mathrm{p} \rightarrow \mathrm{p} \rightarrow \mathrm{p}$ (propositional implication)

$\vdash \mathsf{forall}\ : (\mathrm{e} \rightarrow \mathrm{p}) \rightarrow \mathrm{p}$ (propositional universal)

$\vdash \mathsf{exists}\ : (\mathrm{e} \rightarrow \mathrm{p}) \rightarrow \mathrm{p}$ (propositional existential)

Of these, the connectives not, and , or , and implies would be employed in a (static) semantically interpreted grammar as translations, respectively, for the sentential negation *it is not the case that*, sentential conjunction *and*, sentential disjunction *or*, and sentence subordinator *if*; and the quantifiers would be employed as expected in the translations of the quantification determiners *every* and *some*:

$$\mathsf{every} =_{\text{def}} \lambda_{PQ}.\mathsf{forall}\ \lambda_x.(P\ x)\ \mathsf{implies}\ (Q\ x)$$
$$\mathsf{some} =_{\text{def}} \lambda_{PQ}.\mathsf{exists}\ \lambda_x.(P\ x)\ \mathsf{and}\ (Q\ x)$$

The propositional connectives and quantifiers are subject to the following axioms:

$\vdash \forall_w.\mathsf{truth}@w$

$\vdash \forall_w.\neg(\mathsf{falsity}@w)$

$\vdash \forall_{pw}.(\mathsf{not}\ p)@w \leftrightarrow \neg(p@w)$

$\vdash \forall_{pqw}.(p\ \mathsf{and}\ q)@w \leftrightarrow (p@w \wedge q@w)$

$\vdash \forall_{pqw}.(p\ \mathsf{or}\ q)@w \leftrightarrow (p@w \vee q@w)$

$\vdash \forall_{pqw}.(p\ \mathsf{implies}\ q)@w \leftrightarrow (p@w \rightarrow q@w)$

$\vdash \forall_{Pw}.(\mathsf{forall}\ P)@w \leftrightarrow \forall_x.(P\ x)@w$

$\vdash \forall_{Pw}.(\mathsf{exists}\ P)@w \leftrightarrow \exists_x.(P\ x)@w$

The first six of these axioms say that, in an interpretation, the propositions form a *pre-boolean algebra*, i.e. an algebra that satisfies all the boolean axioms up to the equivalence relation induced by the underlying preorder; in the present case, that preorder is entailment and the induced equivalence is truth-conditional equivalence. Crucially, nothing licenses the inference that entailment is antisymmetric, and so the granularity problem does not arise. Of course there is no shortage in the literature of solutions to the granularity problem, but we believe that this one is by far the simplest.

The Facts of a World Are a Maximal Consistent Set

In a (pre-)boolean algebra, a **maximal consistent set**, also called an **ultrafilter**, is a subset S which 'settles all issues' in the sense that for any algebra element p, either p or its complement is in S, but not both; and which is upper-closed relative to the underlying (pre-)order and closed under the meet operation of the algebra.

In Lewis' [1923] formulation of TS, worlds are easily seen to be maximal consistent by definition, once we align his terminology with more a contemporary one. Specifically: Lewis' 'systems' correspond to 'proper filters'; 'facts' to 'propositions'; 'actual facts' to 'facts' *simplicter*; 'requires' to 'entails', the 'joint fact' of two facts to their 'propositional conjunction'; the 'contradictory of a fact' to its 'propositional negation'; and 'p is inconsistent with q' to 'p entails $\mathsf{not}\ q$'.

In MS, the facts of a world are maximal consistent because the boolean preorder of propositions is the powerset of the set of worlds with entailment as subset inclusion and propositional conjunction as intersection, so that the set of facts of a world w is just the (principal) ultrafilter consisting of all the propositions which have w as a member.

In AS, that the set of facts of any world is maximal consistent is an (easy) theorem of the axioms and definitions given in the previous subsection. To prove it, we just define maximal consistency within our semantic theory in the obvious way (here the variable s is of type $\mathrm{p} \rightarrow \mathrm{t}$):

$\mathsf{upc} =_{\mathrm{def}} \lambda_s.\forall_{pq}.((s\ p) \wedge (p\ \mathsf{entails}\ q)) \rightarrow (s\ q)$ ('upper-closed')

$\mathsf{cjc} =_{\text{def}} \lambda_s.\forall_{pq}.((s\ p) \wedge (s\ q)) \rightarrow (s\ (p\ \mathsf{and}\ q))$ (‘conjunction-closed’)
$\mathsf{sai} =_{\text{def}} \lambda_s.\forall_p.((s\ p) \vee (s\ (\mathsf{not}\ p)) \wedge \neg((s\ p) \wedge (s\ (\mathsf{not}\ p)))$ (‘settles all issues’)
$\mathsf{mxc} =_{\text{def}} \lambda_s.(\mathsf{upc}\ s) \wedge (\mathsf{cjc}\ s) \wedge (\mathsf{sai}\ s)$ (‘maximal consistent’)

Then the theorem in question takes the form:

$$\vdash \forall_w.\mathsf{mxc}\ (\mathsf{facts}\ w)$$

3 From Agnostic Semantics to Montagovian Semantics

Among fine-grained approaches to natural language semantics, AS is perhaps unique in being *consistent with MS.* In fact, AS *becomes* MS (or more precisely, becomes Gallin’s [1975] reformulation of MS within the simple theory of types) with the addition of the following **montagovian axiom:**

$\vdash \forall_{pw}.p@w = (p\ w)$

or, equivalently: $@ =_{\text{def}} \lambda_p.p$. In order for this to be well-typed, we must concomitantly drop the assumption that p is a basic type and instead treat it as an abbreviation for w → t, so that *every set of worlds is a proposition.* It is easily verified that, with this addition, each proposition is identical with its own set of facts:

$\vdash \forall_p.p = \lambda_w.p@w$

that entailment reduces to subset inclusion, and that the propositional connectives become identified with the usual set-theoretic operations on the powerset of the set of worlds. Other consequences include the following:

a. Entailment is antisymmnmetric, i.e. equivalent propositions are identical.
b. For every set of propositions, there is a proposition which, necessarily, is true iff every member of the set is true (namely, the intersection of the set.)
c. For every world w, there is a proposition true only at that world (namely the singleton set whose member is w).
d. Every world is uniquely determined by its set of facts.
e. Not every maximal consistent set of propositions is the set of facts for some world.

The first two of these consequences are explicitly defended by Stalknaker [1976,1984]. But it appears to us that, since they are inescapable consequences of his assumption of the montagovian axiom, he is just making a virtue of necessity. Consequence (c), though evidently lacking empirical consequences, is perhaps an ontological commitment that not every semanticist would wish to take on. Consequence (d) is, as Kripke [1963] notes, a property of the modal semantics in Kripke [1959] which he now wished to avoid, though again it may not have empirical consequences for natural language semantics. Consequence (e) arises because, for any world w, ($\mathsf{facts}\ w$) is the *principal* filter over the powerset of the set of worlds (ordered by subset inclusion) generated by the singleton of w. But (assuming Choice), every infinite boolean algebra has a nonprincipal ultrafilter. This leaves MS in the rather uncomfortable position of being charged with providing an explanation for the fact that there are maximal consistent sets of propositions which don't correspond to any possible world.

To summarize: there is nothing in AS for an advocate of MS to take issue with, since AS is a weaker theory than MS. Any semanticist or philosopher who is comfortable with the additional consequences and commitments of MS is free to add the montagovian axiom. From our perspective though, this is a rather perverse thing to do. To put it in terms of a metaphor: AS is a house that we invite semanticists to inhabit. Some might hesitate to accept, saying: but we are used to living in a house with a leaky roof, and this roof doesn't leak! To them, we say: fine, you can punch a hole in the roof. The montagovian axioms is the tool provided expressly for that purpose.

4 From Agnostic Semantics to Tractarian Semantics

The central tractarian tenet, that worlds are maximal consistent sets of propositions, is hard to formulate in standard higher-order logic. Intuitively, we would like to identify each world with its set of facts:

$$\vdash \forall_w . w = (\mathsf{facts}\ w)$$

Alas, this is ill-typed unless $\mathrm{w} = \mathrm{p} \rightarrow \mathrm{t}$. And we can't just identify w with $\mathrm{p} \rightarrow \mathrm{t}$ because not every set of propositions is a set of worlds, only the maximal consistent ones.[1] What we can do, however, is to assert that there is a *bijection* between worlds and maximal consistent sets of propositions. This we do with two axioms, for injectivity and surjectivity respectively. The first of these can be thought of as a **weak tractarian axiom**:

$$\vdash \forall_{vw} . ((\mathsf{facts}\ v) = (\mathsf{facts}\ w)) \rightarrow v = w$$

i.e. a world is *uniquely determined* by its set of facts (which, recall, has been shown to be maximal consistent).[2]

The other axiom, for surjectivity, says that *every* maximal consistent set of propositions is the set of facts for some world:

$$\vdash \forall_s . (\mathsf{mxc}\ s) \rightarrow \exists_w . s = \mathsf{facts}\ w$$

[1] This technical obstacle can be overcome by working in a version of higher-order logic with separation-style subtyping, such as the categorical logic of Lambek and Scott [1986]: then we identify w with the *subtype* of $\mathrm{p} \rightarrow \mathrm{t}$ whose characteristic function is denoted by mxc.

[2] Again, this is the ontological commitment that Kripke explicitly rejected in his [1963] semantics for normal modal logic (but not in his [1959] semantics for S5).

Together, these two axioms give a (strongly) TS essentially the same as Pollard's ([2008,2011]) hyperintensional semantics (but expressed in standard HOL rather than categorical logic).

Unlike the montagovian extension of agnostic semantics, the (weak or strong) tractarian extensions are free of the (in our view pernicious) consequence that entailment is antisymmetric.[3] Of course, this is already true of agnostic semantics. At this point, it remains unclear to us whether adopting either of the tractarian ontological commitments (injectivity and surjectivity of the facts function) confers any empirical or theoretical advantages on the working semanticist. In the mean time, it seems that we can just go about our usual semantic business in the agnostic setting.

5 Business as Usual in Agnostic Semantics

Here we provide a few illustrations of how to conduct the normal daily business of Montague semantics in the absence of either montagovian or tractarian assumptions.

5.1 Word Meanings

As usual, we introduce lots of constants for word meanings, e.g.

$\vdash \mathsf{p} : \mathrm{e}$ (Pedro)
$\vdash \mathsf{c} : \mathrm{e}$ (Chiquita)
$\vdash \mathsf{m} : \mathrm{e}$ (Maria)
$\vdash \mathsf{donkey} : \mathrm{p}_1$
$\vdash \mathsf{farmer} : \mathrm{p}_1$
$\vdash \mathsf{yell} : \mathrm{p}_1$
$\vdash \mathsf{kick} : \mathrm{p}_2$
$\vdash \mathsf{give} : \mathrm{p}_3$
$\vdash \mathsf{believe} : \mathrm{e} \to \mathrm{p} \to \mathrm{p}$
$\vdash \mathsf{persuade} : \mathrm{e} \to \mathrm{e} \to \mathrm{p} \to \mathrm{p}$
$\vdash \mathsf{that} : \mathrm{p}_1 \to \mathrm{p}_1 \to \mathrm{p}_1$ (property conjunction)
$\vdash \mathsf{every} : \mathrm{p}_1 \to \mathrm{p}_1 \to \mathrm{p}$ (universal determiner)
$\vdash \mathsf{some} : \mathrm{p}_1 \to \mathrm{p}_1 \to \mathrm{p}$ (existential determiner)

These can be made subject to nonlogical axioms (cf. Montague's 'meaning postulates' , or equivalently, be treated as abbreviations, e.g.:

$\mathsf{that} =_{\mathrm{def}} \lambda_{PQx}.(P\ x)\ \mathsf{and}\ (Q\ x)$
$\mathsf{some} =_{\mathrm{def}} \lambda_{PQ}.\mathsf{exists}(\lambda_x.(P\ x)\ \mathsf{and}\ (Q\ x)) = \lambda_{PQ}.\mathsf{exists}(P\ \mathsf{that}\ Q)$
$\mathsf{every} =_{\mathrm{def}} \lambda_{PQ}.\mathsf{forall}(\lambda_x.(P\ x)\ \mathsf{implies}\ (Q\ x))$

[3] The framework of Jónsson and Tarski [1951] is a tractarian system with antisymmetry of entailment, as it is an elaboration (with the addition of boolean operators) of Stone's [1936] duality theory of boolean (not pre-boolean) algebras.

5.2 Extensions of Meanings

We recursively define the set of **meaning types** as follows:

a. e is a **basic** meaning type.
b. p is a **basic** meaning type.
c. If A and B are meaning types, then $A \to B$ is a **functional** meaning type.
c. Nothing else is a meaning type.

For each meaning type A, there is a corresponding type $\mathrm{Ext}(A)$ for the extensions of meanings of type A.

a. $\mathrm{Ext}(\mathrm{e}) = \mathrm{e}$
b. $\mathrm{Ext}(\mathrm{p}) = \mathrm{t}$
c. $\mathrm{Ext}(A \to B) = A \to \mathrm{Ext}(B)$ (*not* $\mathrm{Ext}(A) \to \mathrm{Ext}(B)$)

To handle the notion of the **extension** of a meaning at a world, we extend the @ function to all meaning types by introducing a family of constants

$$\vdash @_A \ : A \to \mathrm{w} \to \mathrm{Ext}(A)$$

where A ranges over meaning types. (Usually the type subscript on '@' is omitted.) Here $a@w$ is read 'the extension of a at w'. The @ functions are subject to the axioms:

$\vdash \forall_{xw}.x@_{\mathrm{e}}w = x$
$\vdash \forall_{pw}.p@_{p}w = p@w$
$\vdash \forall_{fw}.f@w = \lambda_x.(f\ x)@w$ (A a functional type)

The first of these embodies a version of the direct referential theory of names: that the meaning of a name coincides with its reference at whatever world. The second captures the Fregean identification of the reference of a sentence with the truth value of the proposition it expresses. The third, the interesting one, defines the extension of a function meaning in terms of the extensions of the values of the function for all possible arguments. To take a very simple example: to say that Pedro is a farmer in the actual world $\mathbf{w}_0$ is to say that $\mathsf{farmer}@\mathbf{w}_0\ \mathsf{p}$, which in turn amounts to $(\mathsf{farmer}\ \mathsf{p})@\mathbf{w}_0$. That is: to be a $\mathbf{w}_0$-farmer is no more and no less than being an entity such that the proposition that that entity is a farmer is one of the facts of $\mathbf{w}_0$.

5.3 Equivalence of Meanings, Generalized

Recall that two propositions are **equivalent** iff they are true at the same worlds, i.e. $p \equiv q$ iff for every world w, $p@w = q@w$. More generally, we can now say that two meanings a and b of the same type are **equivalent** iff, for every world w, a and b have the same extension at w. That is, for *every* meaning type A, we define **meaning equivalence** by:

$$\equiv_A =_{\text{def}} \lambda_{xy}.\forall_w.x@w = y@w$$

As with mutually entailing propositions, nothing forces equivalent functional meanings to be equal. In the presence of the montagavian axiom, however, equivalence of functional meanings reduces to identity.

6 Conclusion

We sketched the outlines of a weak PWS, of which the familiar Montague semantics and the older, largely forgotten, PWS of Wittgenstein and C.I. Lewis can be viewed as straightforward extensions. We suggested the possibility that the core 'agnostic' theory might replace Montague semantics as a practical framework for the analysis of linguistic meaning. Among the work that remains to be done is a more detailed formalization of the various schemes of PWS proposed by philosophers and linguists; a consideration of the status of structured-meaning theories; and development of a more robust agnostic fragment (covering, for example, interrogatives, conditionals, and various categories of projective meaning). Some of these tasks are taken up in Plummer and Pollard [in prep.].

References

1974. Adams, R.: Theories of actuality. Noûs 8, 211–231 (1974)

1837. Bolzano, B.: Theory of Science. Translation of Wissenschaftslehre, 1837, edited and translated by R. George. University of California Press, Berkeley (1972)

1837. Bolzano, B.: Theory of Science. Translation of Wissenschaftslehre, 1837, edited by J. Berg and translated by B. Terrell. D. Reidel Publishing Company, Berkeley and Los Angeles, Dordrecht (1973)

1947. Carnap, R.: Meaning and Necessity. University of Chicago Press, Chicago (1947)

1940. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)

1985. Cresswell, M.: Structured Meanings. MIT Press (1985)

1981. Dowty, D., Wall, R., Peters, S.: Introduction to Montague Semantics. D. Reidel Publishing Company, Dordreht (1981)

1892. Frege, G.: On sense and reference. Translation of Über Sinn und Bedeutung, 1892. In: Geach, P., Black, M. (eds.) Translations from the Philosophical Writings of Gottlob Frege, 3rd edn. Blackwell, Oxford (1980)

1975. Gallin, D.: Intensional and Higher Order Modal Logic. North-Holland, Amsterdam (1975)

1950. Henkin, L.: Completeness in the theory of types. Journal of Symbolic Logic 15, 81–91 (1950)

1951. Jónsson, B., Tarski, A.: Boolean algebras with operators, part 1. American Journal of Mathematics 73(4), 891–939 (1951)

1996. King, J.: Structured propositions and sentence structure. Journal of Philosophical Logic 25, 495–521 (1996)

2007. King, J.: The Nature and Structure of Content. Oxford University Press, Oxford (2007)

1959. Kripke, S.: A completeness theorem in modal logic. Journal of Symbolic Logic 24, 1–14 (1959)

1963. Kripke, S.: Semantic analysis of modal logic I: normal modal propositional calculi. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 9, 67–96 (1963)

1986. Lambek, J., Scott, P.: Introduction to Higher-Order Categorical Logic. Cambridge University Press, Cambridge (1986)

1923. Lewis, C.I.: Facts, systems, and the unity of the world. Journal of Philosophy 20, 141–151 (1923)

1943. Lewis, C.I.: The modes of meaning. Philosophy and Phenomenological Reseach 4(2), 236–250 (1943)

1970. Lewis, D.: General semantics. Synthese 22, 18–67 (1970)

1979. Lycan, W.: The trouble with possible worlds. In: Loux, M. (ed.) The Possible and the Actual, pp. 274–316. Cornell University Press, Ithaca (1979)

1974. Montague, R.: The proper treatment of quantification in ordinary English. In: Thomason, R. (ed.) Formal Philosophy: Selected Papers of Richard Montague, pp. 247–270. Yale University Press, New Haven (1974)

2005. Muskens, R.: Sense and the computation of reference. Linguistics and Philosophy 28(4), 473–504 (2005)

1974. Plantinga, A.: The Nature of Necesiity. Clarendon, Oxford (1974)

in prep.. Plummer, A., Pollard, C.: A flexible higher order framework for possible-worlds semantics (in preparation)

2008. Pollard, C.: Hyperintensions. Journal of Logic and Computation 18(2), 257–282 (2008)

2011. Pollard, C.: Are (Linguists') Propositions (Topos) Propositions? In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 205–218. Springer, Heidelberg (2011)

1987. Soames, S.: Direct reference, propositional attitudes, and semantic content. Philosophical Topics 15, 47–87 (1987)

1976. Stalnaker, R.: Propositions. In: MacKay, A.F., Merril, D.D. (eds.) Issues in the Philosophy of Language, pp. 79–91. Yale University Press, New Haven (1976)

1984. Stalnaker, R.: Inquiry. Bradford Books/MIT Press, Cambridge (1984)

1936. Stone, M.: The theory of representation for boolean algebras. Transactions of the American Mathematical Society 40, 37–111 (1936)

1980. Thomason, R.: A model theory for propositional attitudes. Linguistics and Philosophy 4, 47–70 (1980)

1921. Wittgenstein, L.: Tractatus Logico-Philosophicus. Translation by D.F. Pears and B.F. McGuinness of Logisch-Philosophische Abhandlung in Annalen der Naturphilosophie, 1921. Routledge & Kegan Paul, London and Henley (1961)

Abstract Machines for Argumentation

Kurt Ranalter

k.ranalter@gmail.com

Abstract. One of the most striking features of ludics is that it provides us with convenient tools for the modelling of interaction. As a consequence, ludics can be employed as a potential framework for the study of dialogues. In this paper we address some of the issues that arise when one tries to model certain types of dialogues that occur in the field of argumentation. We shall exploit that ludics' designs can be regarded as abstract Böhm trees and explain how the pointer interaction of the associated geometric abstract machine relates to a notion of backtracking.

Keywords: ludics, abstract Böhm tree, dialogue, argumentation theory.

1 Introduction

One of the aims of this paper is to show how tools and techniques stemming from investigations on the foundations of programming languages can be applied to the field of argumentation. That interactive approaches to logic are well suited for the modelling of dialogue and thus provide an optimal tool for their formal study is perhaps the main lesson learnt from [3], an in-depth exploration of the potential use of ludics in natural language semantics.

A typical area where the modelling of dialogues plays a central role is that of argumentation. In general, the various approaches considered in the vast literature are highly formalised and hence provide a good benchmark for testing new ideas: any serious proposal for a dialogical framework should be able to capture the kind of dialogues considered there. We shall provide such a proof of concept for the framework given in [4], aimed at modelling persuasion dialogues incorporating disputes about the burden of proof.

An aspect of the dialogues in [4] that seems worth mentioning is the way in which they are build. Roughly speaking, one starts from a given set of dialogue moves that incorporate various options for the continuation of the exchange. For example, a claim C made by the proponent can be either challenged or conceded by the opponent. Even if the opponent first challenges C, it may well turn out at a later point of the exchange that the arguments provided by the proponent are sound, thus forcing the opponent to concede C.

Such instances of backtracking, i.e. the reconsideration or the reevaluation of an earlier point in the exchange, seem to occur quite naturally in all kinds of dialogues and it would thus seem appropriate to have a framework that comes equipped with a primitive for such a pattern of behaviour. We claim that this

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 213–228, 2012.

can naturally be achieved by the so called pointer interaction of abstract Böhm trees [2], a formal framework that subsumes ludics.

It should be noted that the main focus of this paper is on the formalisation of dialogues by means of ludics and abstract Böhm trees. Although insights from the formalisation of dialogues within the field of argumentation have been more than crucial for the development of some of the underlying ideas, we shall leave a detailed comparison of these two strands of work for future investigations. The paper is organised as follows: section 2 uses a simple example to point out the cornerstones of the formalisation; section 3 deals with the use of the so called pointer interaction for the modelling of backtracking.

2 A Case Study

We shall now consider a simple dialogue for information seeking and show how it can be formalised via ludics. The basic example is inspired from a recurrent activity related to teaching: to find out whether a student has grasped a certain concept, i.e. is able to solve a given task or problem.

2.1 Setting

The required task is to find a natural deduction proof of $\varphi_1 \wedge \varphi_2 \vdash \varphi_2 \wedge \varphi_1$. Let us assume that student s has just been given the rules $(\wedge\, i)$ for $\wedge$ introduction and $(\wedge\, e)_1$, $(\wedge\, e)_2$ for $\wedge$ elimination, and that teacher t wants to see whether s is able to apply them correctly. In such a situation it would seem most appropriate for t to exhibit a partial proof in which certain information is missing and then let s complete the proof. Assuming that, whenever possible, introduction rules have to be applied before elimination rules, one yields the following sequences of partial proofs in (Fitch-style) natural deduction.

$$\textbf{Step 1}\left\{\begin{array}{lll}
1.\ \varphi_1 \wedge \varphi_2 & \text{assume} & \\
2.\ \varphi_2 & \dots,1 & \\
3.\ \varphi_1 & \dots,1 & \\
4.\ \varphi_2 \wedge \varphi_1 & \dots,2,3 & \Longleftarrow
\end{array}\right.$$

$$\textbf{Step 2}\left\{\begin{array}{lll}
1.\ \varphi_1 \wedge \varphi_2 & \text{assume} & \\
2.\ \varphi_2 & \dots,1 & \Longleftarrow \\
3.\ \varphi_1 & \dots,1 & \Longleftarrow \\
4.\ \varphi_2 \wedge \varphi_1 & (\wedge\, i),2,3 &
\end{array}\right.$$

$$\textbf{Step 3}\left\{\begin{array}{ll|lll}
1.\ \varphi_1 \wedge \varphi_2 & \text{assume} & 1.\ \varphi_1 \wedge \varphi_2 & \text{assume} & \\
2.\ \varphi_2 & \dots,1 & 2.\ \varphi_2 & (\wedge\, e)_2,1 & \Longleftarrow \\
3.\ \varphi_1 & (\wedge\, e)_1,1 & 3.\ \varphi_1 & \dots,1 & \Longleftarrow \\
4.\ \varphi_2 \wedge \varphi_1 & (\wedge\, i),2,3 & 4.\ \varphi_2 \wedge \varphi_1 & (\wedge\, i),2,3 &
\end{array}\right.$$

$$\textbf{Step 4}\left\{\begin{array}{ll|ll}
1.\ \varphi_1 \wedge \varphi_2 & \text{assume} & 1.\ \varphi_1 \wedge \varphi_2 & \text{assume} \\
2.\ \varphi_2 & (\wedge\, e)_2,1 & 2.\ \varphi_2 & (\wedge\, e)_2,1 \\
3.\ \varphi_1 & (\wedge\, e)_1,1 & 3.\ \varphi_1 & (\wedge\, e)_1,1 \\
4.\ \varphi_2 \wedge \varphi_1 & (\wedge\, i),2,3 & 4.\ \varphi_2 \wedge \varphi_1 & (\wedge\, i),2,3
\end{array}\right.$$

Notice that the constraint on the order of application of the rules serves mainly the purpose to reduce the number of possible cases: the arrows indicate which options are available at a given step. To get an idea of how the formalisation in ludics works it is worth being more specific about the interactive nature of the process that is left implicit in the above scheme.

In a sense, what matters is that the transition from one step to the next models s's contribution to the dialogue. The role of t is to register the choice made by s and to license the partial proof given at the next step. The insight of [3] is that such interactions can be split into two strands, one in which t is either active or passive and one in which s is either active or passive, thus providing two points of view for one and the same dialogue.

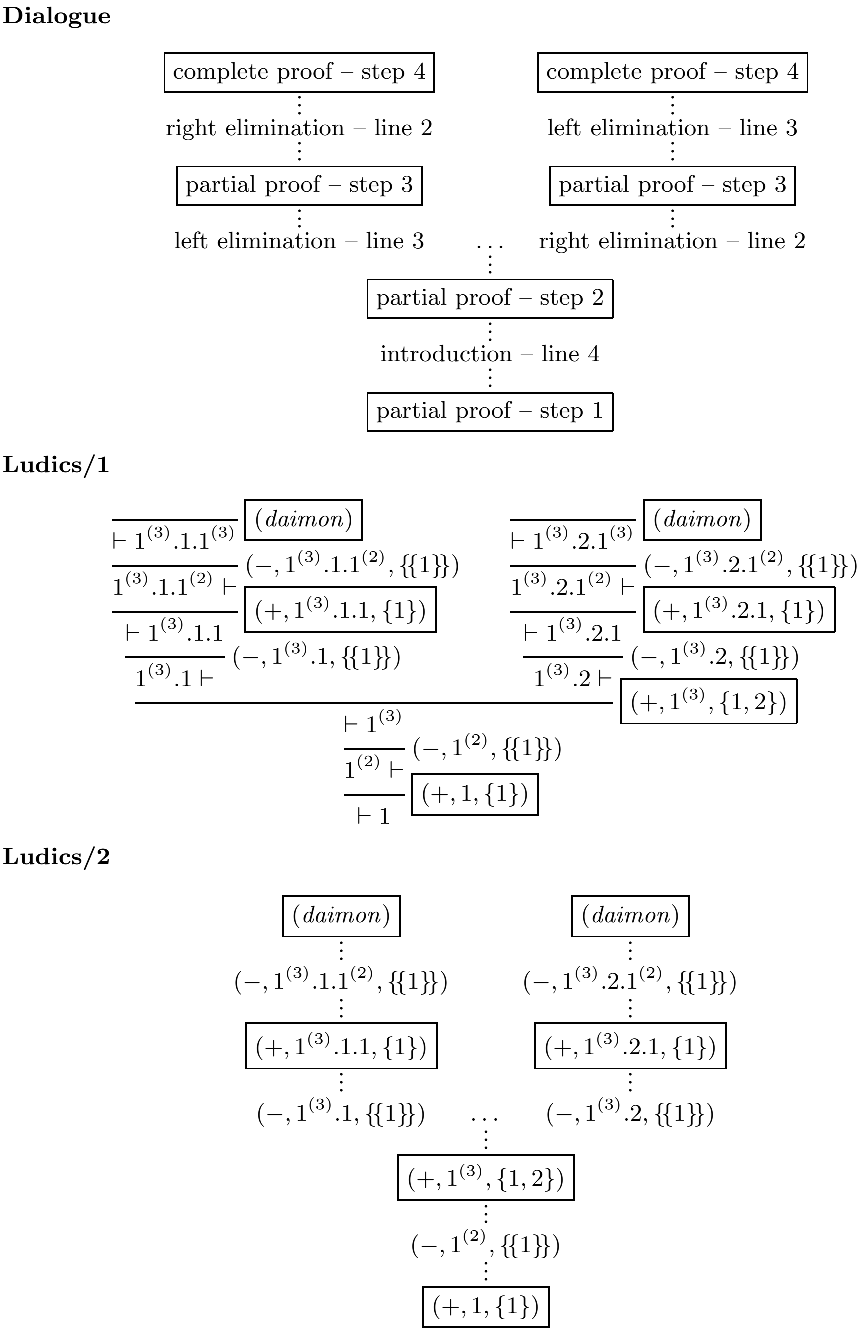
2.2 Teacher

Let us start with t's point of view. Obviously, t is active when it licenses partial proofs and passive when s is about to choose rules. Indeed, at step 1, t licenses the initial partial proof and then waits for s to choose a rule which may then lead to t's licensing of the partial proof shown at step 2, and so on. An interesting feature of the above scheme is that, at step 3, t must be prepared to license two different partial proofs, depending on the choice made by s at step 2. A schematic representation of t's point of view of the exchange is provided by the uppermost tree of figure 1: it is read from bottom to top and the frames indicate when t has an active role during the interaction.

The switch from active to passive and viceversa can be neatly modelled in ludics. Without going into the genesis of ludics, what matters for our purpose is that ludics is a proof-theoretic framework that provides means to distinguish between so called negative sequents $\xi \vdash$ and positive sequents $\vdash \xi$. Because of the special nature of ξ, ludics only needs three rules. For t's point of view the following stripped down variant of the rules is sufficient.

1. Daimon rule $\dfrac{}{\vdash \xi}\,(\mathit{daimon})$
2. Positive rule $\dfrac{\xi.1 \vdash \quad \cdots \quad \xi.n \vdash}{\vdash \xi}\,(+, \xi, \{1, \ldots, n\})$
3. Negative rule $\dfrac{\vdash \xi.1}{\xi \vdash}\,(-, \xi, \{\{1\}\})$

When read from bottom to top, these rules get the following intended meaning. When the conclusion consists of a positive sequent $\vdash \xi$ then we can apply either the (daimon) rule or the $(+, \xi, \{1, \ldots, n\})$ rule. In both of these cases t has an active role: whereas the daimon rule allows t to terminate the exchange, the positive rule allows t to go on with the interaction. For a better understanding of the positive rule it is indispensable to come clear about the role of ξ. Whereas in standard proof-theoretic frameworks ξ stands for some logical formula, in ludics it stands for a sequence of numbers indicating an address. In the simplified version of the rules given above, the address ξ is in one-to-one correspondence



Dialogue

Ludics/1

Ludics/2

Fig. 1. *t*'s point of view

with its actual position in the underlying proof tree. The premises of a positive rule are negative sequents $\xi \vdash$ and, thus, can only be obtained by means of the $(-, \xi, \{\{1\}\})$ rule. In this case t has a passive role: the negative rule allows t to register what s has to say. It is worth pointing out that the negative rule is the one where we have simplified the most. In its original formulation it depends on a possibly infinite collection of sets, thus justifying the use of double curly brackets. However, in the remainder we shall only see cases where this collection contains exactly one set (which often will be a singleton).

We can now turn our attention to the two lowermost trees of figure 1 where $1^{(n)}$ stands for a sequence of n consecutive 1's. These are the formal analogues of the schematic representation of t's point of view given in the uppermost tree. One aspect is particularly worth mentioning: the rule names of ludics are so informative that we can actually forget about the sequents. Such a way of looking at proofs in ludics is expressed by the lowermost tree of figure 1: notice the perfect match between the uppermost and lowermost trees.

2.3 Student

The case of s's point of view can be treated in a similar fashion, provided that two simple observations are taken into account. The first thing to notice is that s can choose the order in which the elimination rules are applied and so can actually choose between two strategies. Their schematic representation is shown in the uppermost forest (collection of trees) of figure 2.

The next thing to consider is that the simplified version of the rules given above does not suffice for the formalisation. What is needed is a version in which there can be more than one address on the right hand side of $\vdash$, the main reason being that we need a variant of the negative rule that does not just deal with a collection containing a singleton. Fortunately, the extension is straightforward and the new rules get only slightly more complicated.

1. Daimon rule ($\Xi = \xi_1, \ldots, \xi_n$) $\dfrac{}{\vdash \Xi}\,(\mathit{daimon})$
2. Positive rule ($\biguplus \Xi_i \subseteq \Xi$) $\dfrac{\xi.1 \vdash \Xi_1 \quad \cdots \quad \xi.n \vdash \Xi_n}{\vdash \xi, \Xi}\,(+, \xi, \{1, \ldots, n\})$
3. Negative rule ($\Xi' \subseteq \Xi$) $\dfrac{\vdash \xi.1, \ldots, \xi.n, \Xi'}{\xi \vdash \Xi}\,(-, \xi, \{\{1, \ldots, n\}\})$

Notice that the formalisation is actually more accurate than the schematic representation. For instance, the bottom-up application of the negative rule $(-, 1^{(3)}, \{\{1, 2\}\})$ in figure 2 leads to a situation where the positive rule which is applied next (either $(+, 1^{(3)}.1, \{1\})$ or $(+, 1^{(3)}.2, \{1\})$) can choose between $1^{(3)}.1$ and $1^{(3)}.2$ (the choice made is emphasised by the frame around the corresponding address). This can be regarded as s being aware of the options that are available to it at a given point of the exchange or interaction.

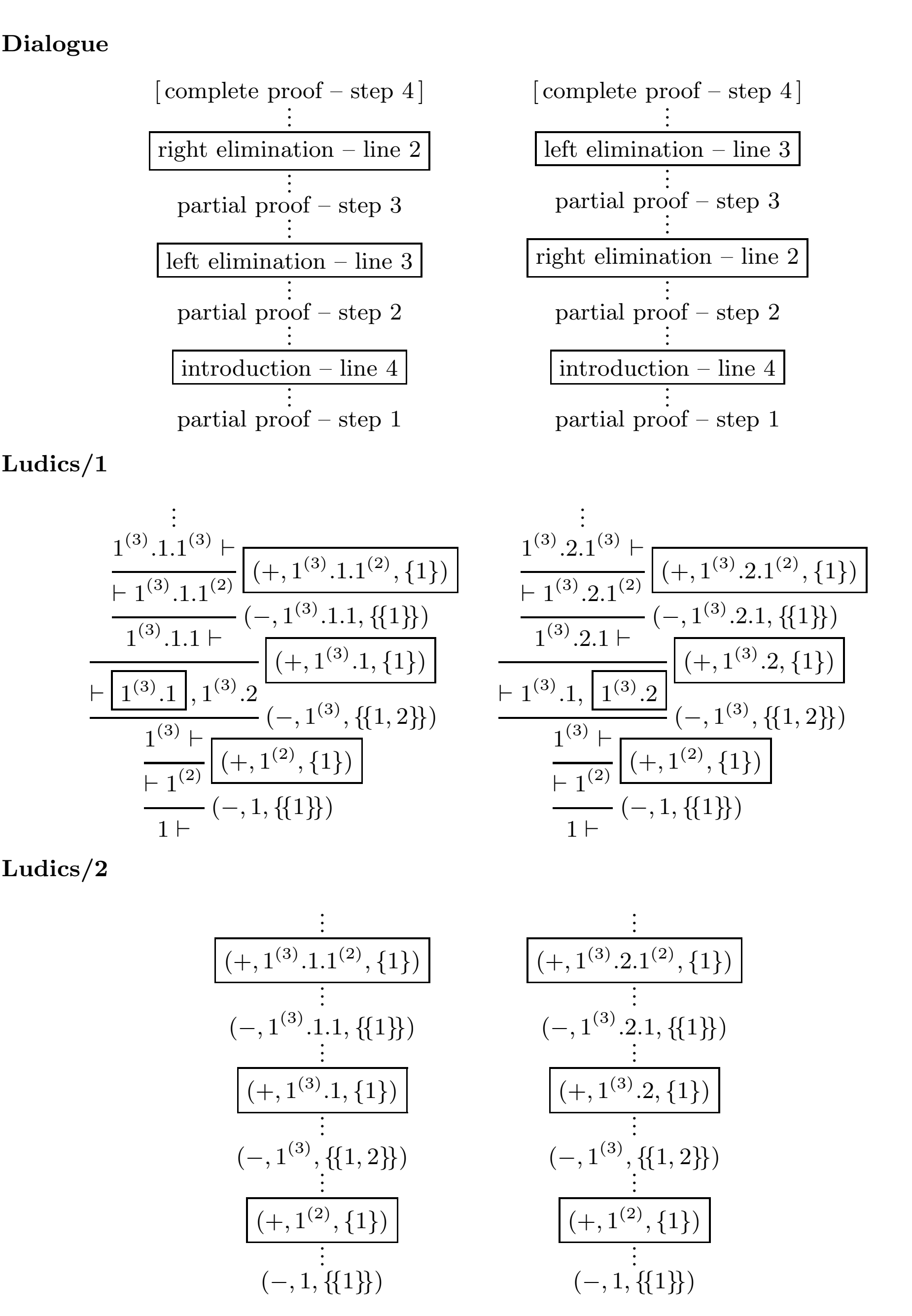


Fig. 2. *s*'s point of view

2.4 Interaction

Let us briefly summarise what we have achieved up to now: we have considered both t's and s's point of view of the exchange and formalised these as proofs in ludics. As a consequence we can now take advantage of ludics and simulate their interaction via cut elimination, a fundamental result in proof theory which was used as a design principle in ludics. If the cut elimination procedure ends in a proof of the empty sequent $\vdash$ obtained by a single application of the (*daimon*) rule, then the interaction is successful. In the light of our example one might say that s has replied to t in a satisfactory manner and this leads t to terminate the exchange. The cut elimination procedure with respect to the rightmost of the two strategies available to s just proceeds as follows.

1. (a) $(\vdash 1)$ vs $(1 \vdash)$
   (b) $(+, 1, \{1\})$ vs $(-, 1, \{\{1\}\})$

$$\dfrac{\dfrac{\overset{\vdots}{1^{(2)} \vdash}}{\vdash 1}\ \boxed{(+,1,\{1\})} \qquad \dfrac{\overset{\vdots}{\vdash 1^{(2)}}}{1 \vdash}\ (-,1,\{\{1\}\})}{\vdash}\ (\mathit{cut})$$

2. (a) $(1^{(2)} \vdash)$ vs $(\vdash 1^{(2)})$
   (b) $(-, 1^{(2)}, \{\{1\}\})$ vs $(+, 1^{(2)}, \{1\})$

$$\dfrac{\dfrac{\overset{\vdots}{\vdash 1^{(3)}}}{1^{(2)} \vdash}\ (-,1^{(2)},\{\{1\}\}) \qquad \dfrac{\overset{\vdots}{1^{(3)} \vdash}}{\vdash 1^{(2)}}\ \boxed{(+,1^{(2)},\{1\})}}{\vdash}\ (\mathit{cut})$$

3. (a) $(\vdash 1^{(3)})$ vs $(1^{(3)} \vdash)$
   (b) $(+, 1^{(3)}, \{1, 2\})$ vs $(-, 1^{(3)}, \{\{1, 2\}\})$

$$\dfrac{\dfrac{\overset{\vdots}{1^{(3)}.1 \vdash} \quad \overset{\vdots}{1^{(3)}.2 \vdash}}{\vdash 1^{(3)}}\ \boxed{(+,1^{(3)},\{1,2\})} \qquad \dfrac{\overset{\vdots}{\vdash 1^{(3)}.1, \boxed{1^{(3)}.2}}}{1^{(3)} \vdash}\ (-,1^{(3)},\{\{1,2\}\})}{\vdash}\ (\mathit{cut})$$

4. (a) $(1^{(3)}.2 \vdash)$ vs $(\vdash 1^{(3)}.2)$
   (b) $(-, 1^{(3)}.2, \{\{1\}\})$ vs $(+, 1^{(3)}.2, \{1\})$

$$\dfrac{\dfrac{\overset{\vdots}{\vdash 1^{(3)}.2.1}}{1^{(3)}.2 \vdash}\ (-,1^{(3)}.2,\{\{1\}\}) \qquad \dfrac{\overset{\vdots}{1^{(3)}.2.1 \vdash}}{\vdash 1^{(3)}.1, \boxed{1^{(3)}.2}}\ \boxed{(+,1^{(3)}.2,\{1\})}}{\vdash}\ (\mathit{cut})$$

5. (a) $(\vdash 1^{(3)}.2.1)$ vs $(1^{(3)}.2.1 \vdash)$
   (b) $(+, 1^{(3)}.2.1, \{1\})$ vs $(-, 1^{(3)}.2.1, \{\{1\}\})$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ 1^{(3)}.2.1^{(2)}\vdash\end{array}}{\vdash 1^{(3)}.2.1}\boxed{(+,1^{(3)}.2.1,\{1\})} \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash 1^{(3)}.2.1^{(2)}\end{array}}{1^{(3)}.2.1\vdash}(-,1^{(3)}.2.1,\{\{1\}\})}{\vdash}(\mathit{cut})$$

6. (a) $(1^{(3)}.2.1^{(2)}\vdash)$ vs $(\vdash 1^{(3)}.2.1^{(2)})$
   (b) $(-,1^{(3)}.2.1^{(2)},\{\{1\}\})$ vs $(+,1^{(3)}.2.1^{(2)},\{1\})$

$$\dfrac{\dfrac{\dfrac{}{\vdash 1^{(3)}.2.1^{(3)}}(\mathit{daimon})}{1^{(3)}.2.1^{(2)}\vdash}(-,1^{(3)}.2.1^{(2)},\{\{1\}\}) \qquad \dfrac{\begin{array}{c}\vdots\\ 1^{(3)}.2.1^{(3)}\vdash\end{array}}{\vdash 1^{(3)}.2.1^{(2)}}\boxed{(+,1^{(3)}.2.1^{(2)},\{1\})}}{\vdash}(\mathit{cut})$$

7. (a) $(\vdash 1^{(3)}.2.1^{(3)})$ vs $(1^{(3)}.2.1^{(3)}\vdash)$
   (b) (*daimon*) vs unspecified rule

$$\dfrac{\dfrac{}{\vdash 1^{(3)}.2.1^{(3)}}(\mathit{daimon}) \qquad \begin{array}{c}\vdots\\ 1^{(3)}.2.1^{(3)}\vdash\end{array}}{\vdash}(\mathit{cut})$$

which then leads to a successful termination of the interaction.

It should be clear that the analysis of both t's and s's point of view has made heavy use of the simplifying assumption that each makes the right move at the right moment. Whereas this sort of behaviour seems to be justified for t (at least in the context of the setting outlined above), one can easily imagine that it cannot be the case for s because there are various ways in which s could go wrong. Indeed, the most immediate one is that s could try to apply an elimination rule instead of the required introduction rule and viceversa. However, there are more subtle situations that can be formalised in ludics. For instance, the following variant of s's rightmost strategy, i.e. proof, considered above illustrates that s might want to change its mind during the exchange.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\vdots\\ 1^{(3)}.1.1^{(3)}\vdash 1^{(3)}.2.1^{(2)}\end{array}}{\vdash 1^{(3)}.2.1^{(2)},\boxed{1^{(3)}.1.1^{(2)}}}\boxed{(+,1^{(3)}.1.1^{(2)},\{1\})}}{1^{(3)}.1.1\vdash 1^{(3)}.2.1^{(2)}}(-,1^{(3)}.1.1,\{\{1\}\})}{\vdash 1^{(3)}.2.1^{(2)},\boxed{1^{(3)}.1}}\boxed{(+,1^{(3)}.1,\{1\})}}{1^{(3)}.2.1\vdash 1^{(3)}.1}(-,1^{(3)}.2.1,\{\{1\}\})}{\vdash 1^{(3)}.1,\boxed{1^{(3)}.2}}\boxed{(+,1^{(3)}.2,\{1\})}}{1^{(3)}\vdash}(-,1^{(3)},\{\{1,2\}\})}{\vdash 1^{(2)}}\boxed{(+,1^{(2)},\{1\})}}{1\vdash}(-,1,\{\{1\}\}) \tag{1}$$

Since the rules allow one to keep formulae on the right hand side of $\vdash$, we have that, although s has opted for address $1^{(3)}.2$ previously (meaning that s wants to finish the partial proof given at step 2 by going from top to bottom), it picks out the address $1^{(3)}.1$ at a later point (meaning that s backtracks from its previous decision and now wants to finish the partial proof given at step 2 by going from bottom to top). Notice that such a modification suffices to make the interaction unsuccessful: the cut elimination procedure gets stuck when it tries to cut $(-, 1^{(3)}.2.1^{(2)}, \{\{1\}\})$ with $(+, 1^{(3)}.1, \{1\})$.

2.5 Discussion

This sudden breakdown of the interaction is more than undesirable and a natural question that poses itself in such a situation is whether we can do better than that. The solution we propose is based on the idea that proofs in ludics can be regarded as so called abstract Böhm trees (see for instance [2]).[1] In the remainder of this section we shall provide a rough introduction to some basic concepts, the actual solution will be presented in the next section.

Roughly speaking, abstract Böhm trees are trees that come equipped with two types of nodes, so called queries q and replies $[r, p]$. As usual in such definitions, there exist a bunch of properties that need to hold. First, on every branch there is a strict alternation between replies and queries, i.e. there are no consecutive occurrences of queries or replies. Second, each query has at most one reply as child, i.e. branchings can only occur below replies. The intuition behind this asymmetry or restriction is that there should always be just one reply that relates to a given query. Notice that replies $[r, p]$ come equipped with an extra piece of information, the pointer p: it points to a query that occurs on the branch from the root of the tree to the reply itself and serves the purpose to overcome the limitation that replies can only relate to queries that immediately precede them, i.e. to queries occurring at the parent node.

Our claim is that pointers provide us with the right tool for the modelling of a change of mind such as the one mentioned above. Besides that, there is one more reason for encoding dialogues as abstract Böhm trees. They come equipped with a collection of abstract machines that can be exploited to implement the cut elimination procedure. Indeed, every example interaction given in this paper has been verified by running it on a concrete implementation of such a machine, the so called geometric abstract machine GAM.

2.6 Encoding

The transformation of ludics proofs (seen as trees of rules) into abstract Böhm trees is done as follows: negative rules $(-, \xi.x, \{\{1, \ldots, n\}\})$ are mapped to queries $(x, \{1, \ldots, n\})$, positive rules $(+, \xi.x, \{1, \ldots, n\})$ to replies $[(x, \{1, \ldots, n\}), p]$. That is, one forgets everything about the address $\xi.x$ except for the last number x.

[1] The same idea was exploited in [1] to provide an extension of ludics that incorporates exponentials and thus goes way beyond the standard variant of ludics presented here.

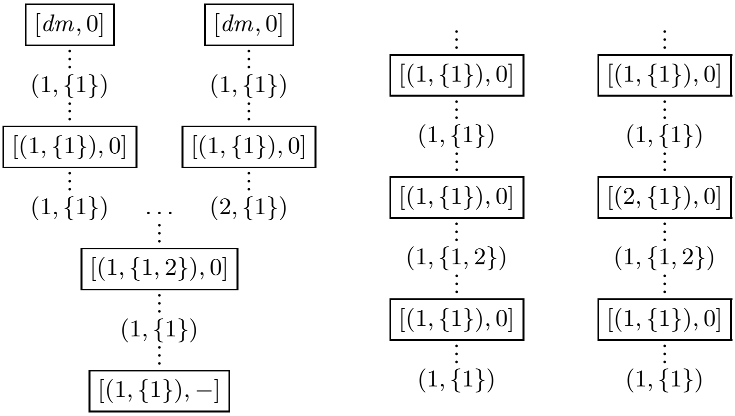


Fig. 3. t's and s's encoding

Since in all our examples the collection of sets in negative rules consists of exactly one set we apply the notational convention to replace double curly brackets with simple curly brackets. Figure 3 illustrates the effect of the transformation in the case of t's and s's point of view, respectively.

The assignment of pointers is a nontrivial issue. Since pointers may be exploited to relate replies to queries that have occurred earlier on in the exchange, they need to be chosen in quite a careful manner. Fortunately, the encoding of both t's and s's point of view as abstract Böhm trees does not make any serious use of the pointer structure, i.e. p takes value 0 almost all the time. By this we mean that the reply relates to the query that immediately precedes it. The case in which p takes no value, i.e. in which $p = -$, is also worth considering: p takes no value only at the beginning of an exchange, that is when the dialogue starts with an initial reply, i.e. whitout any query that precedes it. Examples in which p takes values other than 0 are given in the next section.

2.7 Execution

We shall now outline how the underlying dialogue can be obtained as a product of letting t's and s's point of view interact by means of the geometric abstract machine. To get an intuition for what is actually going let us return briefly to the cut elimination procedure discussed in subsection 2.4.

1. (a) $(+, 1, \{1\})$ vs $(-, 1, \{\{1\}\})$
   (b) $[(1, \{1\}), 0] \longrightarrow (1, \{1\})$
2. (a) $(-, 1^{(2)}, \{\{1\}\})$ vs $(+, 1^{(2)}, \{1\})$
   (b) $(1, \{1\}) \longleftarrow [(1, \{1\}), 0]$
3. (a) $(+, 1^{(3)}, \{1, 2\})$ vs $(-, 1^{(3)}, \{\{1, 2\}\})$
   (b) $[(1, \{1, 2\}), 0] \longrightarrow (1, \{1, 2\})$

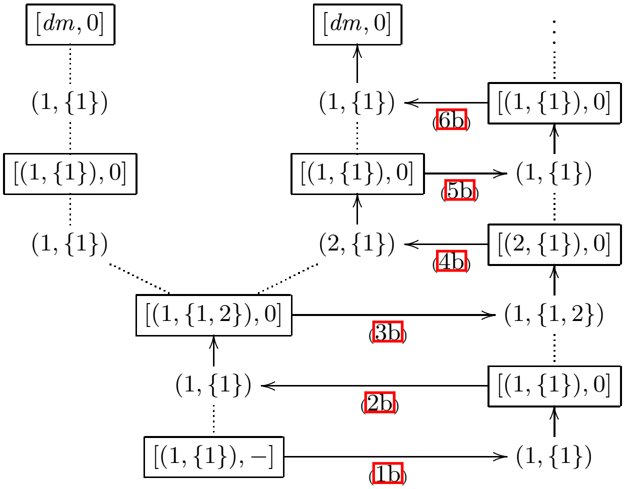


Fig. 4. t's and s's interaction

4. (a) $(-, 1^{(3)}.2, \{\{1\}\})$ vs $(+, 1^{(3)}.2, \{1\})$
   (b) $(2, \{1\}) \longleftarrow [(2, \{1\}), 0]$
5. (a) $(+, 1^{(3)}.2.1, \{1\})$ vs $(-, 1^{(3)}.2.1, \{\{1\}\})$
   (b) $[(1, \{1\}), 0] \longrightarrow (1, \{1\})$
6. (a) $(-, 1^{(3)}.2.1^{(2)}, \{\{1\}\})$ vs $(+, 1^{(3)}.2.1^{(2)}, \{1\})$
   (b) $(1, \{1\}) \longleftarrow [(1, \{1\}), 0]$

Here we have that (a)-items refer to the corresponding (b)-items provided in subsection 2.4. In a sense, (b)-items can simply be seen as translations of the corresponding (a)-items where we have used the convention to replace occurrences of "vs" with an arrow that goes from replies to queries. However, the full meaning is best illustrated by the picture shown in figure 4 where both t's and s's point of view are put next to each other.

The most fundamental feature of the GAM is that it interleaves t-moves with s-moves. That is, after t's initial reply $[(1, \{1\}), -]$, s replies with $[(1, \{1\}), 0]$ to t's initial reply seen as the query $(1, \{1\})$, and so on. Therefore, each single step of the cut elimination procedure can be put in correspondence with one of the turn takings that occur during the dialogue. As a consequence, we get partial visits of both t's and s's point of view.

3 Argumentation

The purpose of our case study was to provide a yet simple but still convincing argument in support of the claim that ludics can be employed as a framework for formalising dialogue. We have argued that one can get for free a fully computational approach by means of the transition to abstract Böhm trees and the associated abstract machines. The purpose of this section is to highlight yet another feature of the approach by making explicit how pointers can be exploited to deal with the issue of backtracking.

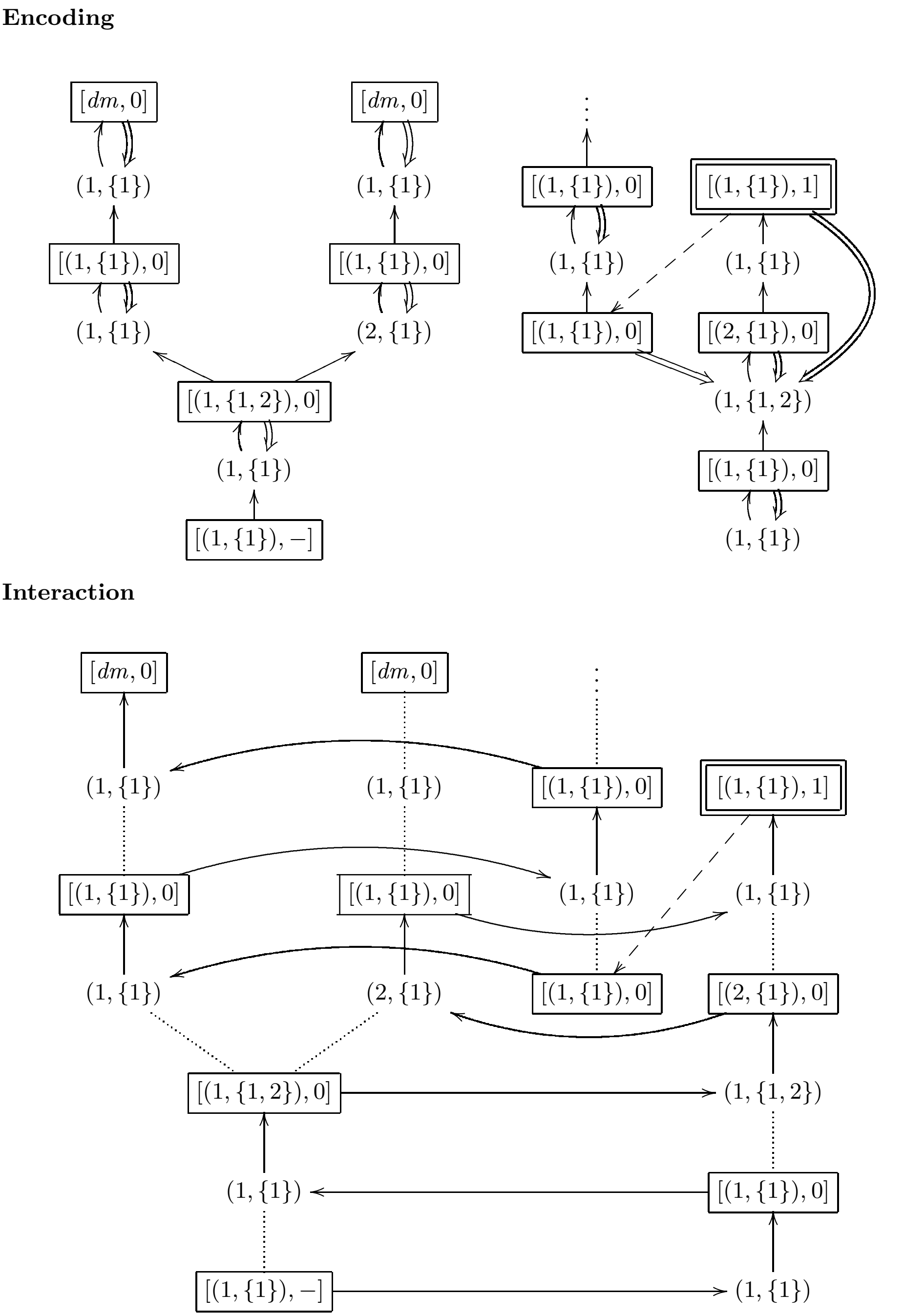
3.1 Backtracking

Consider again strategy (1) given on page 220 above: it shows that in ludics one can formalise that a participant may change its mind and return to an earlier point of the exchange. Though such a change of mind might well be accepted as a legal move in the dialogue, we have seen that the interaction between t's and s's point of view actually becomes unsuccessful, i.e. does not lead to a satisfactory termination of the cut elimination procedure. We shall now see how pointers can help to overcome such a crucial limitation.

The fundamental issue is that we can translate rule $(+, 1^{(3)}.1, \{1\})$ of strategy (1) into various replies, depending on the value we choose for the pointer p. Indeed, if we transform it into the reply $[(1, \{1\}), 1]$ then the encoding of the entire proof as abstract Böhm tree is given by the graph in the top right corner of figure 5 (the graph on its left corresponds to the usual encoding for t). Notice that we have introduced some new notational conventions. The dots from figure 3 have been replaced by solid arrows. Whenever the pointer p of a reply takes a value, i.e. whenever $p \neq -$, a double arrow points to the query to which it relates. So, the reply $[(1, \{1\}), 1]$ (emphasised by a double frame) does not relate to the query $(1, \{1\})$ that immediately precedes it but to the query $(1, \{1, 2\})$. In other words, it counts as a reply $[(1, \{1\}), 0]$ to the query $(1, \{1, 2\})$ and this reading is symbolised by the dashed arrow. We might say that s takes back $[(2, \{1\}), 0]$ and then opts for the alternative $[(1, \{1\}), 0]$.

Although the reading of the dashed arrow as some sort of counts-as definition is rather appealing from an intuitive point of view the introduction of the reply $[(1, \{1\}), 0]$ pointed to by $[(1, \{1\}), 1]$ needs some explanation. The reason is that the definition of abstract Böhm trees makes it impossible to have two consecutive replies, i.e. the sequence of moves $[(1, \{1\}), 1]\,; [(1, \{1\}), 0]$ is not permitted. So, the introduction of the spurious reply $[(1, \{1\}), 0]$ following $[(1, \{1\}), 1]$ is best seen as a device for improving readability.

Such an encoding of the ludics strategy (1) as abstract Böhm tree suffices to make the interaction successful again. The trace of the execution is provided in the second part of figure 5. As an immediate consequence of s's change of mind (symbolised by the dashed arrow), the GAM stops visiting the right hand branch of t's point of view and starts visiting its left hand branch instead. An interesting aspect of the formalisation is that, as a side product, we obtain a neat and convenient graphical language for representing dialogues. Although the move names are still a bit tedious to read, we can now represent t's and s's point of view and, most importantly, their interaction in quite an intuitive way. Notice that the use of double arrows to indicate how replies relate to queries makes the pointer p redundant in the syntax of moves.



Encoding

Interaction

Fig. 5. Instance of backtracking

p_1 : $\textsc{claim}\ C$
o_2 : $\textsc{why}\ C$
p_3 : $C\ \textsc{since}\ \mathit{says}(e, C) \wedge \mathit{expert}(e, C)$
o_4 : $\textsc{why}\ \neg \mathit{biased}(e)$
p_5 : $\textsc{why}\ \mathit{biased}(e)$
o_6 : $\mathrm{BoP}\ (\neg \mathit{biased}(e), p)\ \textsc{since}\ \neg \mathit{biased}(e) \rightarrow \mathit{trusted}(e)$
p_7 : $\textsc{why}\ \neg \mathit{biased}(e) \rightarrow \mathit{trusted}(e)$
o_8 : $\textsc{why}\ \neg(\neg \mathit{biased}(e) \rightarrow \mathit{trusted}(e))$
p_9 : $\neg(\neg \mathit{biased}(e) \rightarrow \mathit{trusted}(e))\ \textsc{since}\ \mathit{presumed}(\neg \mathit{biased}(e))$
o_{10} : $\textsc{retract}\ \neg \mathit{biased}(e) \rightarrow \mathit{trusted}(e)$
o_{11} : $\mathit{biased}(e)\ \textsc{since}\ \mathit{paid}(e, c) \wedge \mathit{testifies}(e, c)$
p_{12} : $\textsc{concede}\ \mathit{biased}(e)$
p_{13} : $\textsc{retract}\ C$

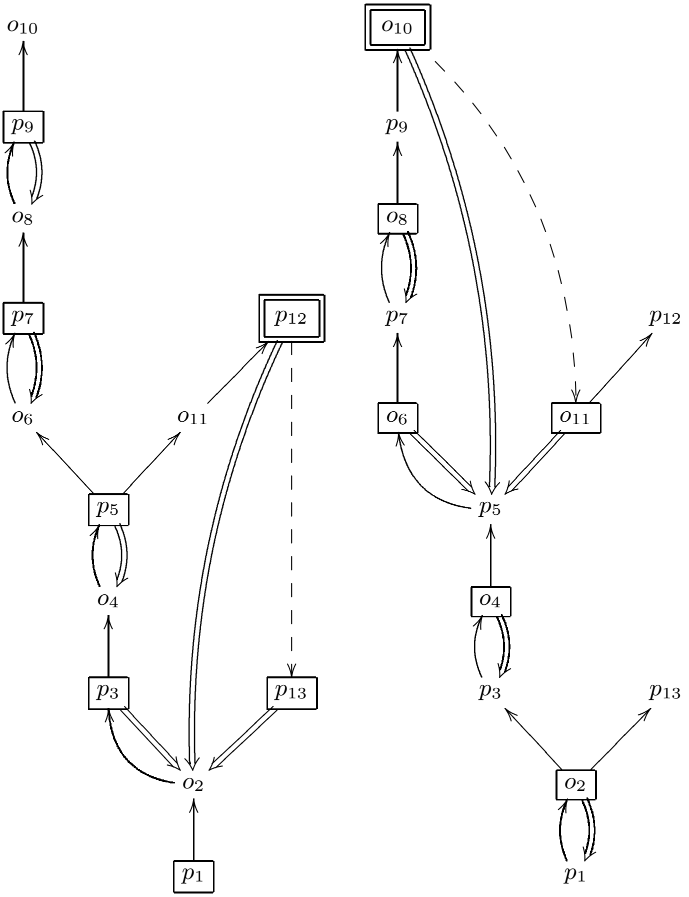


Fig. 6. Persuasion dialogue

3.2 Persuasion

With all that in place, we can turn our attention to the formalisation of an example taken from the field of argumentation theory. The dialogue we shall consider is taken from section 5.1 of [4] and a synthesised version of it is provided in the first part of figure 6. In this exchange p tries to persuade o that C is the case. Roughly, the most interesting passages are the following ones. In move p_3, p employs an argument from expert opinion (e stands for a given individual, the expert). In move o_4, o replies to p's argument by making appeal to a so called critical question which provides means for o to defeat p. If we forget for a moment the moves from o_6 to o_{10}, then this is exactly what happens: in move o_{11}, o provides a reason for why it is the case that e is biased (e is paid by company c and e testifies for c) and this then leads p to retract C in move p_{13}. The moves from o_6 to o_{10} represent a dialogue about the burden of proof. In move o_6, o points out that p has the burden of proof (BoP) with respect to the claim that e is unbiased. The moves from p_7 to o_{10} just serve the purpose to evaluate whether the reason offered by o in move o_6 is justified: in the end, o retracts it and so terminates the BoP dialogue.

With respect to the formalisation of this dialogue in the proposed framework, one aspect is particularly relevant. As we have just observed, move o_{10} indicates the end of the BoP dialogue and so the next move (o_{11}) can readily be seen as a change of mind and so we have an instance of backtracking. The case is similar with respect to move p_{12} where p concedes that e is biased. Here again, the next move (p_{13}) can be regarded as an instance of backtracking. It should now be relatively easy to see that p's and o's point of views are accurately represented by the two graphs in the second part of figure 6.

It is worth mentioning that, in contrast to the examples we have considered previously, move names now correspond to labels used in the transcript of the persuasion dialogue. Together with the convention to make the pointer structure explicit, this leads to a considerable improvement of the readability of the graphical language. Notice also the special role of the moves o_{10} and p_{12} (emphasised by a double frame in figure 6). When regarded as components of an abstract Böhm tree, they do not count as complete replies. Instead, they serve the purpose to indicate that we have a pointer taking a value different from 0. Only in combination with the moves o_{11} and p_{13} that immediately follow them do we obtain a complete reply. That is, consecutive dialogue moves of either p or o actually correspond to just one single reply. This relates nicely to the observation made in [4, p. 120] that "a *turn* of a player always consists of zero or more surrenders followed by a single attack" where surrender/attack is a classification for the various continuations of a dialogue move.

4 Conclusion

This paper combines ideas from [2] and [3] and shows how these can be applied to dialogues considered in the theory of argumentation [4]. Although the technical development is very similar to the one presented in [1], it seems worth to point out the different aims of these two strands of work: whereas [1] is an in-depth investigation of foundational issues in ludics, this paper is intended as a pointer towards possible applications for such a framework.

We have tried to organise the paper in such a way that new concepts are introduced step by step, trying to make the basic issues accessible to anyone not already familiar with the general framework. In the light of this objective, it is clear that certain aspects could be addressed only in a somehow superficial manner. Others, such as a comparison of the analysis provided in subsection 3.2 with the framework presented in [4], have simply been omitted. What seems clear is that an essential part of the protocol for persuasion dialogues presented in [4] can be expressed by means of abstract Böhm trees.

It would thus seem that a framework such as the one presented in [4] could take full advantage of the computational approach to dialogues advocated in this paper. However, the transfer of knowledge does not just go in one direction. As mentioned in the introduction, the dialogues in [4] are build from a finite collection of dialogue moves; the dialogue protocol just mentioned defines how these can be combined to obtain a well-formed dialogue. This aspect is particularly relevant if one is interested in a bottom-up or incremental approach to dialogue: the top-down or holistic approach seen in this paper is rather well suited for an a posteriori analysis of dialogues, but it is quite difficult to see how it could help for the generation of dialogues. A more promising approach seems to be to start from dialogue moves such as the ones considered in [4].

References

1. Basaldella, M., Faggian, C.: Ludics with repetitions (exponentials, interactive types and completeness). Logical Methods in Computer Science 7(2:13), 1–85 (2011)
2. Curien, P.-L., Herbelin, H.: Abstract machines for dialogue games. In: Interactive Models of Computation and Program Behavior. Panoramas et Sythèses, vol. 27, pp. 231–275. SMF (2009)
3. Lecomte, A., Quatrini, M.: Ludics and Its Applications to Natural Language Semantics. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS (LNAI), vol. 5514, pp. 242–255. Springer, Heidelberg (2009)
4. Prakken, H., Reed, C., Walton, D.: Dialogues about the burden of proof. In: Proceedings of the 10th International Conference on Artificial Intelligence and Law, ICAIL 2005, pp. 115–124. ACM (2005)

On the Completeness of Lambek Calculus with Respect to Cofinite Language Models*

Alexey Sorokin

Moscow State University, Faculty of Mechanics and Mathematics

Abstract. We give an alternative proof of the fact that the product-free Lambek calculus is complete with respect to cofinite language models. It was first proved by Buszkowski in 1982 by the method of barriers. We use another method, which is also based on the technique of canonical models, to obtain a new proof of this result.

Lambek calculus was introduced by Joachim Lambek in [7]. In the last decades it has found different algebraic applications. In [1] it was proved, in particular, that the product-free Lambek calculus is complete with respect to language models. The paper [2] strengthens this result, showing that it is also complete with respect to cofinite language models. The articles [3] and [4] prove the finite model property for a class of logics, including the product-free Lambek calculus and the product-free Lambek calculus with empty antecedents, both enriched with the intersection operation. In fact, this result implies the completeness of these calculi with respect to cofinite language models. We give another proof of this fact for the product-free Lambek calculus. Our method involves a canonical model and the cut-elimination theorem for some axiomatic extension of this calculus (a similar strategy was used in [6] for proving finite model property of MALL and in [8] for proving finite model property of related intuitionistic logics).

Let us fix an enumerable set of primitive types $p_1, \dots, p_n, \dots$ and denote it by Pr. A set of product-free types Tp is defined as the smallest set satisfying the following conditions: 1)Pr $\subset$ Tp, 2) for any two types A and B from Tp the types (A/B) and $(B\backslash A)$ are also in Tp. The sequents of the product-free Lambek calculus have the form $\Gamma \to A$, where A is in Tp and Γ is a non-empty sequence of types. We shall use Λ to denote the empty sequence.

The product-free Lambek calculus has the only axiom $A \to A$, $A \in$ Tp. The inference rules are the following:

$$\frac{\Gamma B \to A}{\Gamma \to A/B}(\to /) \quad \Gamma \neq \Lambda \qquad \frac{\Pi \to A \qquad \Gamma B \Delta \to C}{\Gamma (B/A) \Pi \Delta \to C}(/ \to)$$

$$\frac{B \Gamma \to A}{\Gamma \to B\backslash A}(\to \backslash) \quad \Gamma \neq \Lambda \qquad \frac{\Pi \to A \quad \Gamma B \Delta \to C}{\Gamma \Pi (A\backslash B) \Delta \to C}(\backslash \to)$$

$$\frac{\Pi \to B \qquad \Gamma B \Delta \to C}{\Gamma \Pi \Delta \to C}(cut)$$

* This research was partially supported by the Russian Foundation for Basic Research (grant 11-01-00958).

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 229–233, 2012.

It can be proved that the cut rule can be eliminated from Lambek calculus (and also in its product-free fragment). We denote the product-free Lambek calculus by $\mathrm{L}(\backslash, /)$ and the derivability of a sequent $\Gamma \to A$ in this calculus by $\mathrm{L}(\backslash, /) \vdash \Gamma \to A$. Also we define the notion of derivability from a set $\mathcal{D}$ of additional axioms, which means that we can also use the sequents from $\mathcal{D}$ in the derivation. Note that the cut rule cannot be eliminated in this variant of the calculus.

We call a language an arbitrary set of words over a given alphabet. Below we recall some basic definitions concerning languages. A finite alphabet Σ given, we denote by Σ^+ the set of all nonempty finite sequences containing only letters from Σ.

Definition 1. *For two languages L_1 and L_2 their product (or concatenation) is defined as $L_1 \cdot L_2 = \{u_1 \cdot u_2 \mid u_1 \in L_1,\, u_2 \in L_2\}$, where $u_1 \cdot u_2$ is the concatenation of words u_1 and u_2.*

Definition 2. *Let L_1 and L_2 be languages. Then $L_1/L_2 = \{u \in \Sigma^+ \mid (\forall u_2 \in L_2)\ u \cdot u_2 \in L_1\}$ and $L_2 \backslash L_1 = \{u \in \Sigma^+ \mid (\forall u_2 \in L_2)\ u_2 \cdot u \in L_1\}$.*

Definition 3. *A pair $\mathbf{W} = \langle \Sigma^+, w \rangle$, where Σ is a finite alphabet and w is a function from Tp to $\mathcal{P}(\Sigma^+)$, is called a language model if for every A, B in Tp we have $w(A/B) = w(A)/w(B)$ and $w(B\backslash A) = w(B)\backslash w(A)$. A language model is called cofinite if for every primitive type $p \in \mathrm{Pr}$ the set $\Sigma^+ - w(p)$ is finite.*

We can extend the mapping w to finite sequences of types setting $w(A_1 \dots A_n) = w(A_1) \cdot \ldots \cdot w(A_n)$. Note that a language model is cofinite if and only if the images of all types are cofinite, which is easily proved by induction on the type structure.

Definition 4. *A sequent $\Gamma \to A$ is said to be valid in a language model $\langle \Sigma^+, w \rangle$ if $w(\Gamma) \subset w(A)$.*

It is easy to see that each sequent derivable in Lambek calculus is valid in every language model (which is called the soundness of Lambek calculus with respect to language models). In [1] it is proved that the converse also holds: every sequent that is valid in each language model is derivable in the product-free Lambek calculus (which is called completeness). Here we want to strengthen this result by showing that even each sequent that is valid in every cofinite language model is also derivable in the product-free Lambek calculus, which was already proved in [2].

To achieve this goal we need to introduce some auxiliary notions.

Definition 5. *A set of types $\mathcal{U}$ is called downward closed if for every type of the form A/B or $B\backslash A$ which belongs to $\mathcal{U}$ the types A and B also belong to $\mathcal{U}$.*

Our main purpose is to show that every underivable sequent is invalid in some cofinite language model. In what follows $\mathcal{T}$ will be some downward closed set of types and K will be a natural number. Their values will be specified in the proof of Theorem 1. We denote by $|\Delta|$ the number of types in the sequence Δ (so,

$|(p_1/p_2)p_3(p_4\backslash p_2)|$ equals 3) and by $\|\Gamma\|$ the total number of primitive types and connectives in the sequence Γ (for example, $\|(((p_1/p_2)\backslash p_3)/p_2)(p_3/p_1)\| = 10$). We also define for every sequence $\Gamma \to A$ its length $\|\Gamma \to A\|$, which is equal to $\|\Gamma\| + \|A\|$.

Now we want to define a new calculus L_K. Its axioms contain the standard axioms of Lambek calculus and all sequents of the form $\Gamma \to A$ where $|\Gamma| \geq K$ (we call such axioms "long"). The inference rules are the rules of Lambek calculus.

Lemma 1. *The cut rule can be eliminated from* L_K*.*

Proof. We want to show that if the last rule in the derivation is $\frac{\Pi\to B \quad \Gamma B \Delta \to C}{\Gamma \Pi \Delta \to C}$ and both premises can be derived without using the cut rule, then the conclusion of the rule can be derived without using this rule. We use induction by $\|\Pi\| + \|\Gamma\| + \|\Delta\| + \|B\| + \|C\|$. There are several cases: 1) when both premises are obtained in the last step with some inference rule of the product-free Lambek calculus or are axioms of this calculus, 2) $\Pi \to B$ is a long axiom, 3) $\Gamma B \Delta \to C$ is a long axiom.

In the first case the standard proof of cut elimination in Lambek calculus is appropriate (see, e.g. [7]). In the second case $|\Pi| \geq K$, whence $|\Gamma \Pi \Delta| \geq K$ and $\Gamma \Pi \Delta \to C$ is an axiom of L_K, so it is derivable. In the third case $|\Gamma B \Delta| \geq K$ and because Π is non-empty $|\Gamma \Pi \Delta| \geq K$ and the conclusion is again an axiom. The lemma is proved.

Lemma 2. *If* $\mathrm{L}_K \vdash \Gamma \to A/B$*, then* $\mathrm{L}_K \vdash \Gamma B \to A$*. If* $\mathrm{L}_K \vdash \Gamma \to B\backslash A$*, then* $\mathrm{L}_K \vdash B\Gamma \to A$*.*

Proof. $\frac{\Gamma \to A/B \quad (A/B)B \to A}{\Gamma B \to A}$ (cut). The case of the other division is symmetric.

Lemma 3. *For all sequents* ϕ *such that* $\|\phi\| \leq K$ *the conditions* $\mathrm{L}_K \vdash \phi$ *and* $L(\backslash, /) \vdash \phi$ *are equivalent.*

Proof. Assume that $\|\phi\| \leq K$ and $\mathrm{L}_K \vdash \phi$. For every "long" axiom its length is greater than K. It is easy to prove by induction on the length of a cut-free derivation that every sequent for which this axiom is used in a derivation has length greater than K. So ϕ cannot have "long" axioms in its derivation, which means that it is derivable in $\mathrm{L}_{\backslash,/}$.

Let us define a language model $\mathbf{W} = \langle \mathcal{T}^+, w \rangle$ in the following way: for a primitive type p we set $w(p) = \{\Gamma \in \mathcal{T}^+ | \mathrm{L}_K \vdash \Gamma \to p\}$ and for other types their images are obtained by induction. Note that $\mathbf{W}$ is cofinite provided that $\mathcal{T}$ is finite.

Lemma 4. *For every type* A *from* $\mathcal{T}$ $w(A) = \{\Gamma \in \mathcal{T}^+ | \mathrm{L}_K \vdash \Gamma \to A\}$.

Proof. We proceed by induction on type A, for primitive types the statement holds by definition of function w. We consider only the case $A = B/C$, the case of other division is analogous. Assume a sequence $\Gamma \to B/C$, where Γ contains only types from $\mathcal{T}$, is derivable in L_K, then by Lemma 2 the sequent $\Gamma C \to B$

is also derivable. Also by the cut rule if $\mathrm{L}_K \vdash \Delta \to C$ then $\mathrm{L}_K \vdash \Gamma\Delta \to B$ and by the induction assumption this implies that for every $\Delta \in w(C)$ we have $\Gamma\Delta \in w(B)$. This means that $\{\Gamma \in \mathcal{T}^+ | \mathrm{L}_K \vdash \Gamma \to B/C\}$ is a subset of $w(B/C)$.

For the other inclusion we mention that if a sequent Γ belongs to $w(B/C) = w(B)/w(C)$ then ΓC belongs to $w(B)$ because $C \to C$ is an axiom of L_K. Hence $\mathrm{L}_K \vdash \Gamma C \to B$. Then by the rule $(\to /)$ we obtain that $\mathrm{L}_K \vdash \Gamma \to B/C$, which was required. The lemma is proved.

Theorem 1. *For every sequent $\Gamma \to A$ that is not derivable in the product-free Lambek calculus, there is a cofinite language model where ϕ is wrong.*

Proof. We choose some downward closed set of types containing all types from Γ and A as $\mathcal{T}$ and assign $K = \|\Gamma\| + \|A\|$. Consider the cofinite language model $\mathbf{W} = \langle \mathcal{T}^+, w\rangle$ defined before Lemma 4. Using this lemma it is easy to see that $\Gamma \in w(\Gamma)$, because for every type C we have $C \in w(C)$. On the other hand, $\Gamma \notin w(A)$ because in view of Lemma 3 the sequent $\Gamma \to A$ is not derivable in L_K. Hence we have built a model where $\Gamma \to A$ is not valid.

If we recall the theorem from [1], this result can be reformulated in other terms.

Corollary 1. *The set of formulae of the form $A \subseteq B$ where A and B are types made from primitive types and the signs of left and right divisions that are valid in every language model, coincides with the set of formulae that are valid in all cofinite language models.*

It is interesting if the full Lambek calculus L has the same completeness properties (its completeness with respect to arbitrary language models is proved in [9]), also the analogous question can be asked for the full variant of calculus L^* (* means that sequences with empty antecedents are allowed and the restrictions $\Pi \neq \Lambda$ are omitted in the rules $(\to /)$ and $(\to \backslash)$). Farulewski proved the finite model property for these calculi in [5], but his result cannot be translated in terms of language models, at least directly. Similar problems can be stated if we enrich the calculus with new connectives (such as $\cap$ and $\cup$, interpreted as the intersection and union of languages) and structural rules for them. For example, for the calculus $\mathrm{L}(\backslash, /, \cap)$ the construction used in this article is also suitable (the completeness of this calculus with respect to cofinite language models was already proved in [3]), but for the calculi including union the question is still open.

I am grateful to Mati Pentus for helpful consultations during the work. Also I want to thank the anonymous referees for very useful remarks that helped to improve the paper.

References

1. Buszkowski, W.: Compatibility of a Categorial Grammar with an Associated Category System. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 28, 229–237 (1982)

2. Buszkowski, W.: Some Decision Problems in the Theory of Syntactic Categories. Zeitschrift für mathematische Logik und Grundlagen der Mathematik 28, 539–548 (1982)
3. Buszkowski, W.: The finite model property for BCI and related systems. Studia Logica 57, 303–323 (1996)
4. Buszkowski, W.: Finite Models of Some Substructural Logics. Math. Log. Q. 48(1), 63–72 (2002)
5. Farulewski, M.: On the Finite Models of the Lambek Calculus. Studia Logica 80, 63–74 (2005)
6. Lafont, Y.: The Finite Model Property for Various Fragments of Linear Logic. Journal of Symbolic Logic 62, 1202–1208 (1997)
7. Lambek, J.: The mathematics of sentence structure. American Mathematical Journal 65(3), 154–170 (1958)
8. Okada, M., Terui, K.: The Finite Model Property for Various Fragments of Intuinistic Linear Logic. The Journal of Symbolic Logic 64(2), 790–802 (1999)
9. Pentus, M.: Models for the Lambek calculus. The Annals of Pure and Applied Logic 75(1–2), 179–213 (1995)

Dot-types and Their Implementation

Tao Xue and Zhaohui Luo*

Department of Computer Science
Royal Holloway, University of London
taoxue@cs.rhul.ac.uk,
zhaohui.luo@hotmail.co.uk

Abstract. Dot-types, as proposed by Pustejovsky and studied by many others, are special data types useful in formal semantics to describe interesting linguistic phenomena such as copredication. In this paper, we present an implementation of dot-types in the proof assistant Plastic base on their formalization in modern type theories.

Keywords: Dot-types, Coercive Subtyping, Type-Theoretical Semantics, Type Theory, Proof Assistant.

1 Introduction

Dot-types, or sometimes called dot objects or complex types, were introduced by Pustejovsky in the Generative Lexicon Theory [19] and studied by many others, including [3]. Intuitively, a dot-type is formed from two constituent types that present distinct aspects of those objects in the dot-type. For example, a book may be considered to have two aspects: one informational (eg, when it is read) and the other physical (eg, when it is picked up). One may therefore consider a dot-type PHY • INFO whose objects, including books, have both physical and informational aspects. In particular, such objects can be involved in the linguistic phenomenon of copredication and dot-types play a promising role in its analysis and formalisation.

Although the meaning of dot-types is intuitively clear, its proper formal account seems surprisingly difficult and tricky (see [2] for a discussion). Researchers have made several proposals to model dot-types formally including, for example, [3,4] and [6,7]. Besides discussions on whether the proposed solutions do capture and therefore give successful formal accounts of dot-types, most of these proposals are considered in the Montagovian setting. In [13], the second author has proposed a formal treatment of dot-types in modern type theories[1] (MTTs) with the help of coercive subtyping and it is argued that, because in the formal semantics based on MTTs common nouns are interpreted as types (rather

* Partially supported by the research grant F/07-537/AJ of the Leverhulme Trust in U.K.

[1] Modern type theories may be classified into the predicative type theories such as Martin-Löf's type theory [16,18] and the impredicative type theories such as the Calculus of Constructions (CC) [9] and the Unifying Theory of dependent Types (UTT) [11].

D. Béchet and A. Dikovsky (Eds.): LACL 2012, LNCS 7351, pp. 234–249, 2012.

than predicates as in the Montague semantics), the linguistic phenomena such as copredication can be given satisfactory treatments by means of dot-types.

In this paper, we present an implementation of dot-types in the proof assistant Plastic [5], based on the formalization of dot-types in MTTs. As far as we know, this is the first attempt to implement the dot-types.[2] It allows us to use dot-types in the development of formal semantics in proof assistants and, at the same time, gives us a better understanding of dot-types and their relationship with other data types in type theory.

Dot-types are not ordinary inductive types, as found in the MTT-based proof assistants such as Agda [1], Coq and Plastic. In particular, for $A \bullet B$ to be a dot-type, the constituent types A and B should not share components (see the main text for the formal definition). In an implementation of dot-types, this special condition of type formation must be checked and adhered to. In order to make sure of this, we have to implement the dot-types as special data types, different from ordinary inductive types. We shall show how this is done in our implementation in Plastic.

Dot-types are introduced informally in section 2, where we focus on the idea that the constituent types of a dot-type do not share common components. In section 3, we discuss the formal formulation of dot-types in MTTs. The implementation of dot-types is presented in section 4, where we will first briefly introduce the proof assistant Plastic and then explain how to implement dot-types in Plastic with several examples to illustrate the use.

2 Dot-types in Formal Semantics: An Introduction

In the Generative Lexicon Theory [19], Pustejovsky has introduced the idea of employing dot-types to model various linguistic data that involve objects with distinct aspects. Typical examples are concerned about copredication, where different aspects of a word are selected when predication comes into force. For example, in the following sentences [3], the words ‘lunch’ and ‘book’ both have two distinct aspects to be selected: in (1) a ‘lunch’ was delicious as food and took forever as an event, and in (2) a ‘book’ was picked up as a physical object and mastered as an informational object.

(1) The lunch yesterday was delicious but took forever.

(2) John picked up and mastered the mathematical book.

There have been studies of dot-types in various formal systems or semi-formal systems including, for example, [3,4,19,21]. Most of these proposals are given in the Montagovian setting where, in particular, common nouns are interpreted as

[2] In [14] the second author has presented some examples in Coq [8] and, since Coq does not support dot-types, he had to use Σ-types to mimic dot-types, although fully aware of the fact that this is in general impossible and leads to incoherence, since there is no guarantee that the constituent types of a dot-type do not share components.

predicates. It has been argued in [13] that the way that CNs are interpreted in the Montagovian setting is incompatible with the subtyping postulates that the type of entities has subtypes Event (of events), Phy (of physical objects), Info (of informational objects), etc. This leads to unnecessary difficulties and formal complications when formalizing dot-types. On the other hand, if we interpret CNs as types, as in the formal semantics based on MTTs, the treatment becomes straightforward and satisfactory [13]. (See section 3.1 for further details.)

Usually the two aspects involved in a dot-type are incompatible: in the above examples, Food and Event are incompatible and so are the physical and informational objects. This incompatibility of the two aspects that form a dot-type was expressed by Pustejovsky as follows:

> *Dot objects have a property that I will refer to as* inherent polysemy*. This is the ability to appear in selectional contexts that are contradictory in type specification.* [20]

In other words, an important feature is that, to form a dot-type $A \bullet B$, its constituent types A and B should not share common parts. For instance,

- Phy $\bullet$ Phy should not be a dot-type because its constituent types are the same type Phy.
- Phy $\bullet$ (Phy $\bullet$ Info) should not be a dot-type because its constituent types Phy and Phy $\bullet$ Info share the component Phy.

Put in another way, a dot-type $A \bullet B$ can only be formed if the types A and B do not share any components: it is a dot-type only when the constituent types A and B represent different and incompatible aspects of the objects.

This incompatibility is one of the two key features based on which dot-types are introduced in MTTs [13]: it is stipulated that the constituent types of a dot-type do not share components. The other feature is that the relationships between the dot-type and its constituent types are captured by means of coercive subtyping so that the dot-type is the subtype of both of its constituent types. We now turn to the type-theoretic formulation of dot-types.

3 Dot-types in Modern Type Theories

In this section, we show how dot-types can be introduced in modern type theories with the help of coercive subtyping [13]. We will first explain informally, in the formal semantics based on MTTs, called *type-theoretical semantics* henceforth, how to use dot-types to interpret copredication in natural language. Then we will lay down the formal rules of the dot-types in modern type theories.

3.1 Dot-types and Coercive Subtyping

Type-Theoretical Semantics. In [22] Ranta has studied various semantic issues of natural languages in Martin-Löf's type theory, introducing the basic ideas of type-theoretical semantics based on MTTs. Unlike Montague grammar in which common nouns like Man and $Human$ are interpreted as functional subsets (or predicates) of entities, in the type-theoretical semantics based on modern type theories, common nouns are interpreted as types. For instance, in Montague grammar, Man and $Human$ are interpreted as objects of type $e \to t$, where e is the type of entities and t the type of propositions. In type-theoretical semantics, the interpretations of Man, $Human$ and $Book$ are types:

$$[[man]], [[human]], [[book]] : Type$$

This is natural in a modern type theory, which is many-sorted in the sense that there are many types like $[[man]]$ and $[[book]]$ consisting of objects standing for different sorts of entities, while the simple type theory may be thought of as single sorted in the sense that there is the type e of all entities. In a type-theoretical semantics, verbs and adjectives are interpreted as predicates. For example, we can have

$$[[nice]] : [[book]] \to Prop$$
$$[[read]] : [[human]] \to [[book]] \to Prop$$

where $Prop$ is the type of propositions.

Let's we consider a kind of dependent type called Σ-types. It basically means that 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 $B(a)$.

Modified common noun phrases could be interpreted by means of Σ-types: for instance,

$$[[nice\ book]] = \Sigma([[book]], [[nice]])$$

Coercive Subtyping. Coercive subtyping was introduced in [12]. The basic idea of coercive subtyping is to consider subtyping as an abbreviation mechanism: A is a subtype of B ($A <_c B$), if there is a unique implicit coercion c from type A to type B. If so, an object a of type A can be used in any context that expects an object of type B, and it is equal to $c(a)$. The formal coercive definition rule is defined as:

$$\frac{\Gamma \vdash f : B \to C \quad \Gamma \vdash a : A \quad \Gamma \vdash A <_c B : Type}{\Gamma \vdash f(a) = f(c(a)) : C}$$

For instance, one may consider the type of men to be a subtype of the type of human beings by declaring a coercion between them: $[[man]] <_m [[human]]$. If we assume that $walk$ be interpreted as $[[walk]] : [[human]] \to Prop$ and $Jack$ as $[[Jack]] : [[man]]$, we could interpret the sentence (3) as (4).

(3) Jack walks.

(4) $[[walk]]([[Jack]])$

The reason that (4) is well-typed is that $[[man]]$ is now a subtype of $[[human]]$, an appropriate coercion can be inserted to fill up the gap in the term in (4).

Notation. We shall adopt the following notational abbreviations, writing

- $A < B$ for $A <_c B : \mathbf{Type}$ for some c,
- $A \leq B$ for $A = B : \mathbf{Type}$ or $A < B$.

Dot-type and Coercive Subtyping. Intuitively, a dot-type should be a subtype of its constituent types. For instance, it is natural to think that the type consisting of the objects with both aspects of food and event be a subtype of Food as well as a subtype of Event. Similarly, the type consisting of objects with both physical and informational aspects should be a subtype of the type Phy of physical objects and a subtype of the type of informational aspect:

$$\textsc{Phy} \bullet \textsc{Info} < \textsc{Phy}$$

$$\textsc{Phy} \bullet \textsc{Info} < \textsc{Info}$$

Consider sentence (2) again. In a type-theoretical semantics, we may assume that

$$\begin{aligned} [[book]] &< \textsc{Phy} \bullet \textsc{Info} \\ [[pick\ up]] &: [[human]] \rightarrow \textsc{Phy} \rightarrow Prop \\ [[master]] &: [[human]] \rightarrow \textsc{Info} \rightarrow Prop \end{aligned}$$

Because of the above subtyping relationship (and contravariance of subtyping for the function types), we have

$$\begin{aligned} [[pick\ up]] &: [[human]] \rightarrow \textsc{Phy} \rightarrow Prop \\ &< [[human]] \rightarrow \textsc{Phy} \bullet \textsc{Info} \rightarrow Prop \\ &< [[human]] \rightarrow [[book]] \rightarrow Prop \end{aligned}$$

$$\begin{aligned} [[master]] &: [[human]] \rightarrow \textsc{Info} \rightarrow Prop \\ &< [[human]] \rightarrow \textsc{Phy} \bullet \textsc{Info} \rightarrow Prop \\ &< [[human]] \rightarrow [[book]] \rightarrow Prop \end{aligned}$$

Therefore, $[[pick\ up]]$ and $[[master]]$ can both be used in a context where terms of type $[[human]] \rightarrow [[book]] \rightarrow Prop$ are required and the interpretation of the sentence (2) can proceed as intended.

However, as we mentioned above, there are some difficulties if we do the same thing in the Montagovian settings, we can show it with a simple case. Take the example of "heavy book", in Montague semantics, we should have

$$\begin{aligned} [[heavy]] &: (\textsc{Phy} \rightarrow t) \rightarrow (\textsc{Phy} \rightarrow t) \\ [[book]] &: \textsc{Phy} \bullet \textsc{Info} \rightarrow t \end{aligned}$$

In order to interpret "heavy book" as [[heavy]]([[book]]), we need

$$\textsc{Phy} \bullet \textsc{Info} \to t < \textsc{Phy} \to t$$

By contravariance, we need

$$\textsc{Phy} < \textsc{Phy} \bullet \textsc{Info}$$

But this is not the case, the subtype relation is actually in another way around.

3.2 Dot-types in Type Theory: A Formal Formulation

In the following, we present a type-theoretic treatment of dot-types with the help of coercive subtyping. There are two important ingredients in this type-theoretic definition:

i. The constituent types of a dot-type should not share common components.
ii. A dot-type, if well-formed, should be a subtype of both of its constituent types.

Because of (i), the first and the most important thing is to define the notion of component and, when doing this, because of (ii), the set of components of a type should contain those of all of its constituents and super-types. This is formally given by means of the following definition.

Definition 1 (component). *Let* T:**Type** *be a type in the empty context. Then,* $\mathscr{C}(T)$*, the set of components of* T*, is defined according to the normal form*[3] *of* T *as :*

$$\mathscr{C}(T) =_{def} \begin{cases} SUP(T) & \textit{if the normal form of } T \textit{ is not of the form } X \bullet Y \\ \mathscr{C}(T_1) \cup \mathscr{C}(T_2) & \textit{if the normal form of } T \textit{ is } T_1 \bullet T_2 \end{cases}$$

where $SUP(T) = \{T' | T \leq T'\}$*.*

Now, we give the formal rules for the dot-types in Figure 1[4]. Note that, in the formation rule, we require that the constituent types do not share common components:

$$\mathscr{C}(A) \cap \mathscr{C}(B) = \emptyset$$

According to the rules in Figure 1, $A \bullet B$ is a subtype of A and a subtype of B. In other words, an object of the dot-type $A \bullet B$ can be regarded as an object of type A, in a context requiring an object of A, and can also be regarded as an object of type B in a context requiring an object of B.

Finally, the subtyping relations are propagated through the dot-types, by means of the coercions d_1, d_2 and d as specified in the last three rules in Figure 1.

[3] Intuitively, in a modern type theory with strong normalization and Church-Rosser properties, every process of computation starting from a well-typed term terminates and it computes to a (unique) normal form.

[4] In the formation rule of Figure 1, $\Gamma \vdash$ **valid** means that Γ is a valid context and <> stands for an empty context.

Formation Rule

$$\frac{\Gamma \vdash \mathbf{valid} \quad <> \vdash A : \mathbf{Type} \quad <> \vdash B : \mathbf{Type} \quad \mathscr{C}(A) \cap \mathscr{C}(B) = \emptyset}{\Gamma \vdash A \bullet B : \mathbf{Type}}$$

Introduction Rule

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash A \bullet B : \mathbf{Type}}{\Gamma \vdash \langle a, b \rangle : A \bullet B}$$

Elimination Rules

$$\frac{\Gamma \vdash c : A \bullet B}{\Gamma \vdash p_1(c) : A} \qquad \frac{\Gamma \vdash c : A \bullet B}{\Gamma \vdash p_2(c) : B}$$

Computation Rules

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash A \bullet B : \mathbf{Type}}{\Gamma \vdash p_1(\langle a, b \rangle) = a : A} \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B \quad \Gamma \vdash A \bullet B : \mathbf{Type}}{\Gamma \vdash p_2(\langle a, b \rangle) = b : B}$$

Projections as Coercions

$$\frac{\Gamma \vdash A \bullet B : \mathbf{Type}}{\Gamma \vdash A \bullet B <_{p_1} A : \mathbf{Type}} \qquad \frac{\Gamma \vdash A \bullet B : \mathbf{Type}}{\Gamma \vdash A \bullet B <_{p_2} B : \mathbf{Type}}$$

Coercion Propagation

$$\frac{\Gamma \vdash A \bullet B : \mathbf{Type} \quad \Gamma \vdash A' \bullet B' : \mathbf{Type} \quad \Gamma \vdash A <_{c_1} A' : \mathbf{Type} \quad \Gamma \vdash B = B' : \mathbf{Type}}{\Gamma \vdash A \bullet B <_{d_1[c_1]} A' \bullet B' : \mathbf{Type}}$$

where $d_1[c_1](\langle a, b \rangle) = \langle c_1(a), b \rangle$.

$$\frac{\Gamma \vdash A \bullet B : \mathbf{Type} \quad \Gamma \vdash A' \bullet B' : \mathbf{Type} \quad \Gamma \vdash A = A' : \mathbf{Type} \quad \Gamma \vdash B <_{c_2} B' : \mathbf{Type}}{\Gamma \vdash A \bullet B <_{d_2[c_2]} A' \bullet B' : \mathbf{Type}}$$

where $d_2[c_2](\langle a, b \rangle) = \langle a, c_2(b) \rangle$.

$$\frac{\Gamma \vdash A \bullet B : \mathbf{Type} \quad \Gamma \vdash A' \bullet B' : \mathbf{Type} \quad \Gamma \vdash A <_{c_1} A' : \mathbf{Type} \quad \Gamma \vdash B <_{c_2} B' : \mathbf{Type}}{\Gamma \vdash A \bullet B <_{d[c_1, c_2]} A' \bullet B' : \mathbf{Type}}$$

where $d[c_1, c_2](\langle a, b \rangle) = \langle c_1(a), c_2(b) \rangle$.

Fig. 1. The rules of Dot-type

Remark 1. It is worth pointing out that, under the definition of component and rules of dot-types, there is no "universal supertype" of all types.

Propagations. To explain the propagation rules, we can think of interpreting the phrase

pick up and read the book

Instead of simply considering book having physical and informational aspect, we might think book contains *readable* information, compared to *radio program* which does not have a *readable* informational aspect. So we could interpret

$$\begin{aligned} [[readable]] &: \textsc{Info} \to Prop \\ [[readable\ info]] &= \Sigma(\textsc{Info}, [[readable]]) \\ [[book]] &< \textsc{Phy} \bullet [[readable\ info]] \end{aligned}$$

With the coercion relation we have for Σ-types [15],

$$\Sigma(\textsc{Info}, [[readable]]) < \textsc{Info}$$

we have $[[readable\ info]] < \textsc{Info}$ and trivially we have $\textsc{Phy} = \textsc{Phy}$. So we could get the following through propagation rule

$$[[book]] < \textsc{Phy} \bullet [[readable\ info]] < \textsc{Phy} \bullet \textsc{Info}$$

This conforms with the example we've explained above.

Coherence. When we consider coercive subtyping, coherence [12] is the most important property that is required to hold. Informally, coherence means that the coercion between any two types is unique. Put in another way, if there are two coercions c and c' from type A to type B, c and c' are required to be equal. One may intuitively understand the importance of this property like this: it guarantees that there's no ambiguity when we use a coercion.

Remark 2. If there are only finitely many coercions and we identify equal coercions, the coercions form a forest.

Since the constituent types of a well-formed dot-type do not share components, it is straightforward to prove the following coherence property.

Proposition 2. *(coherence) The coercions p_1, p_2, d_1, d_2 and d are coherent together.*

Note that coherence is important as it guarantees the correctness of employing the projections p_1 and p_2 and the propagation operator d as coercions, and hence the subtyping relationships $A \bullet B <_{p_1} A$ and $A \bullet B <_{p_2} B$.

If the constituent types of a dot-type shared a common component, coherence would fail, like in product type. For instance, A and $A \bullet B$ share the component A. If $A \bullet (A \bullet B)$ were a dot-type, with the transitivity rule[5] there would be the following two coercions p_1 and $p_2 \circ p_1$:

$$A \bullet (A \bullet B) <_{p_1} A$$

$$A \bullet (A \bullet B) <_{p_2 \circ p_1} A$$

[5] Transitivity for the coercion means that, if we have two coercions $A <_{c_1} B$ and $B <_{c_2} C$, then there's coercion from A to $C (A <_{c_2 \circ c_1} C)$ where $c_2 \circ c_1 \equiv [x{:}A](c_2(c_1(x)))$ is the composition of c_1 and c_2 .

which are between the same types but not equal, coherence would then fail.

One may find that, when a dot-type $A \bullet B$ is well-formed, its behavior is similar to that of a product type $A \times B$: intuitively, its objects are pairs and the projections p_1 and p_2 correspond to the projection operations π_1 and π_2 in the product type, respectively. However, there are two important differences between dot-types and product types:

1. The constituent types of a dot-type do not share components, while in a product type the constituent parts can possibly share component. For instance, $A \times A$ is a well-formed product type, but $A \bullet A$ is not a well-formed dot-type.
2. It is fine for both of the projections p_1 and p_2 for dot-types to be coercions (Proposition 2), but for product types, only one of them can be coercion, otherwise, coherence would fail [10].

4 Implementation

We have shown how to formalize the dot-types in type theory in the last section. As we have proof assistants which have implemented various data types, we would also like to put dot-types into a proof assistant. However unlike inductive types such as the product types or Σ-types which could be defined with inductive schemata, dot-type cannot be simply be defined as such in a library of the proof assistant. The main reason is that we need to check whether the constituents of the dot-type share components. Especially in our definition of component, we need to check all the coercion relations of the term and its constituents in the context. This is not covered by existing approaches to define inductive types in the libraries of proof assistants. So we have to proceed in a hard way: defining dot-types directly in a proof assistant.

In this section, we present how we define dot-types in the proof assistant Plastic, and show how to use it.

4.1 Proof Assistants and Plastic

A proof assistant is a piece of software to assist with the formal proofs in a man-machine interactive way, based on constructive mathematical proofs. One can define mathematical problems in the provided formal language, choose the right strategy or algorithms in the library to achieve the proof. Modern type theories have been implemented in the proof assistants such as Agda [1], Coq [8], Matita [17] and Plastic [5] used in applications to formalization of mathematics and verification of programs.

Plastic [5] is an implementation of the type theory UTT as presented in chapter 9 of [11] with inductive families and universes. In the library of the proof assistant, we have various predefined inductive types and an second order logic. We also have implemented coercive subtyping in Plastic.

With the help of proof assistants like Plastic, we can also implement the formal semantics, which would help us to study the type-theoretical semantics for linguistic issue.

Here, we do not explain the technical details of using Plastic. Instead, we present a very simple example to show how we use Plastic to develop formal semantics.

Example 3. *Consider the example* (3) *"Jack walks" in section* 2*. Its semantics* (4) *can be presented by the last line of the following code in Plastic:*

```
> [Man, Human :Type];
> [c : Man -> Human];
> Coercion = c;
> [Jack : Man];
> [walk : Human -> Prop];
> [Jack_walks = walk(Jack)];
```

In the code, "Coercion = c" makes the function c as a coercion from Man to $Human$. With the help of coercion, the term $Jack\_walks$ is well typed: $Jack\_walks = walk(Jack) = walk(c(Jack)) : Prop$.

4.2 Dot-types in Plastic

As explained above, the dot-types have to be directly implemented in Plastic and, at the same time, the associated subtyping relations have to be specified.

- In the syntax of Plastic, we use $A * B$ to present the dot-type $A \bullet B$ and $dot < a, b >$ to present dot term $< a, b >$.
- When we declare a new dot-type $A \bullet B$, or a dot term $< a, b >$ where $a{:}A$ and $b{:}B$, we will first check whether it is a proper dot-type. If $\mathscr{C}(A) \cap \mathscr{C}(B) = \emptyset$, $A \bullet B$ will be a legal dot-type or $< a, b >$ will be a legal dot term; otherwise, they will be rejected and an error message '*dot-type should not share component*' will be shown.
- Once a dot-type $A \bullet B$ or dot term $< a, b >$ is considered to be well-formed (legal), we will consider the coercions generated from the dot-type $A \bullet B$. We will add $[x{:}A \bullet B]p_1(x)$ and $[x{:}A \bullet B]p_2(x)$ as coercions from $A \bullet B$ to A and B[6].
- Furthermore, we will check the existing coercions of other dot-types to see whether there are cases for coercion propagation, if so, Plastic will add the new coercion generated by the coercion propagation into context.

In the implementation, we define some reductions for the computation rules for the projections p_1 and p_2, and the propagation operators d, d_1, and d_2. Assume $A, B, C, D{:}Type$, $a{:}A$, $b{:}B$, $A <_{c_1} C$, $B <_{c_2} D$, we have:

[6] The system will automatically assign new metavariable names $cx1$, $cx2$, ... to the new introduced coercions by the dot-type rules.

$$
\begin{aligned}
p_1(< a, b >) &\rhd a\\
p_2(< a, b >) &\rhd b\\
d[c_1, c_2](< a, b >) &\rhd < c_1(a), c_2(b) >\\
d_1[c_1](< a, b >) &\rhd < c_1(a), b >\\
d_2[c_2](< a, b >) &\rhd < a, c_2(b) >
\end{aligned}
$$

In the following, we present three main algorithms in our implementation. First we need to give an algorithm to calculate the components of a type.

Algorithm 4. *(checking components) Given a type A, we will calculate the component of A, $\mathscr{C}(A)$, in the following way.*

1. *Check the form of A to see whether it is a dot-type or not.*
2. *If A is not a dot-type, check all the coercion relations in the context to find out every type T which satisfies $A <_c T$ with some coercion c. $\mathscr{C}(A)$ is the set of all these super type T.*
3. *If A is a dot-type of the form $T_1 \bullet T_2$, $\mathscr{C}(A) = \mathscr{C}(T_1) \cup \mathscr{C}(T_2)$. (The algorithm is called recursively.)*

The second algorithm deals with the introduction of dot-types.

Algorithm 5. *When defining a type to be a dot-type $A \bullet B$:*

1. *Check the context to see whether A and B are already defined types. If so, go to next step; otherwise alert the type is not defined and end the algorithm.*
2. *Calculate $\mathscr{C}(A)$ and $\mathscr{C}(B)$ to see whether the intersection of these two is empty. If so go to the next step; if not, alert that dot-type do not share component and end the algorithm.*
3. *Check the existing coercions, to see whether $A \bullet B$ has already been considered. If so, simply finish the algorithm; otherwise go to the next step.*
4. *Add coercion from $A \bullet B$ to A and from $A \bullet B$ to B into the context, add the coercions generated by transitivity as well.*
5. *Check the existing coercion of dot-type in the context to add coercions introduced by propagation rules. For every existing dot-type $C \bullet D$,*
   - *if there's a coercion c_1 from A to C, and a coercion c_2 from B to D, add a new coercion $[x{:}A \bullet B]d[c_1, c_2](x)$ from $A \bullet B$ to $C \bullet D$.*
   - *if there's a coercion c_1 from A to C, and B equals to D, add a new coercion $[x{:}A \bullet B]d_1[c_1](x)$ from $A \bullet B$ to $C \bullet D$.*
   - *if there's a coercion c_2 from B to D, and A equals to C, add a new coercion $[x{:}A \bullet B]d_2[c_2](x)$*
   - *otherwise, do nothing.*
6. *Check the transitivity possibilities of the new generated coercion.*

The third algorithm deals with the introduction of dot-terms (similar to that for dot-type introduction).

Algorithm 6. *When defining a term to be a dot term $< a, b >$:*

1. *Check the context to see whether a and b are defined terms. If so take the types of a and b, let say A and B, and go to the next step; otherwise alert the term is not defined and end the algorithm.*
2. *Calculate $\mathscr{C}(A)$ and $\mathscr{C}(B)$ to see whether the intersection of these two is empty. If so go to the next step; if not, alert that dot-type do not share component and end the algorithm.*
3. *Check the existing coercions, to see whether $A{\bullet}B$ has already been considered. If so, simply finish the algorithm; otherwise go to the next step.*
4. *Add coercion from $A \bullet B$ to A and from $A \bullet B$ to B into the context, add the coercions generated by transitivity as well.*
5. *Check the existing coercion of dot-type in the context. For every existing dot-type $C \bullet D$,*
   - *if there's a coercion c_1 from A to C, and a coercion c_2 from B to D, add a new coercion $[x{:}A \bullet B]d[c_1, c_2](x)$ from $A \bullet B$ to $C \bullet D$.*
   - *if there's a coercion c_1 from A to C, and B equals to D, add a new coercion $[x{:}A \bullet B]d_1[c_1](x)$ from $A \bullet B$ to $C \bullet D$.*
   - *if there's a coercion c_2 from B to D, and A equals to C, add a new coercion $[x{:}A \bullet B]d_2[c_2](x)$*
   - *otherwise, do nothing.*
6. *Check the transitivity possibilities of the new generated coercion.*

Another part we should take care of, is that, since we need to consider the propagation rules of dot-types, when we introduce a new coercion, it links two existing dot-types and generates a new coercion through the propagation rules. So when we introduce a new coercion, we should also check all the dot-types in the context to see whether there're types satisfy the conditions of propagation rule, and add corresponding coercions for the propagation rules.

4.3 Examples of Dot-types in Plastic

In this subsection, we will first give some abstract examples to show how to declare a dot-type in Plastic, what we will get from the declaration, and some examples of illegal declaration of dot-types. Then we will give a concrete case to interpret sentences in natural language into code in Plastic.

Example 7. *We can define a dot-type or a dot-term simply in the following way:*

1. *If we have two types A, B which do not share components, we could simply define a type M of type $A * B$ like this:*

   ```
   > [M = A*B];
   ```

   The system will generate two coercions $cx1$ from $A * B$ to A and $cx2$ from $A * B$ to B.

2. *We can also define a dot term . If we have two terms a, b, a:A and b:B, we can define a dot term $m =< a, b >$ like this:*

   ```
   > [m = dot<a,b>];
   ```

   Now m is defined to be a dot term dot $< a, b >$ and it is of type $A * B$. The system will generate two coercions $cx1$ from $A * B$ to A and $cx2$ from $A * B$ to B.

Example 8. *In the following examples, the types share components in different ways and, therefore, none of them could be defined as a dot-type or dot term, they fail and warnings will be shown in all the following cases.*

1. *The two constituents are the same*

   ```
   > [M = A*A];
   ```

2. *$A * C$ and $A * B$ have the same component A*

   ```
   > [M = (A*C)*(A*B)];
   ```

3. *A is a subtype of B, by definition of component $\mathscr{C}(A) \cap \mathscr{C}(B) = \{A\}$*

   ```
   > [c:A->B];
   > Coercion = c;
   > [M = A*B];
   ```

4. *a and b are both of type A, but $A * A$ is not a legal dot-type, so dot $< a, b >$ is not a legal dot term.*

   ```
   > [a,b:A];
   > [ab = dot<a,b>];
   ```

5. *a is of type A and b is of type B, while A is a subtype of B. As shown above, $A * B$ is not a legal dot-type, hence dot $< a, b >$ is not a legal dot term.*

   ```
   > [a:A];
   > [b:B];
   > [c:A->B];
   > Coercion = c;
   > [ab = dot<a,b>];
   ```

Example 9. *When we have dot-type $A * B$, $A <_{c1} C$ and $B <_{c2} D$, if we claim $C * D$ to be another dot-type, coercions from the propagation rule will also be added.*

```
> [c1:A->C];
> [c2:B->D];
> Coercion = c1;
> Coercion = c2;
> [M1 = A*B];
> [M2 = C*D];
```

In this example several coercions will be added according to the dot-type rule and transitivity. $cx1$ *from* $A * B$ *to* A, $cx2$ *from* $A * B$ *to* B, $< c1, cx1 >= [x{:}(A * B)]cx3(cx1\ x)$ *by transitivity from* $A{*}B$ *to* C, $< c2, cx1 >= [x{:}(A{*}B)]cx4(cx1\ x)$ *by transitivity from* $A * B$ *to* D, $cx3$ *from* $C * D$ *to* C *and* $cx4$ *from* $C * D$ *to* D. *However, we will get one more coercion from the propagation rule, there will be a coercion* $cx5$ *from* $A{*}B$ *to* $C{*}D$ *and* $cx5 = [x{:}A{*}B]d[c1, c2](x)$, *where for any dot term* $dot < a, b >$ *of dot-type* $A * B$, $d[c1, c2]dot < a, b >= dot < c1(a), c2(b) >$.

Now, let's use a concrete example to show how we could interpret natural language in Plastic:

Example 10. *Let's consider the sentence*

John picked up and mastered a book

We should contain the following data:

```
> [PhyInfo = Phy*Info];
> [cb : Book -> PhyInfo];
> Coercion = cb;
> [John:Human];
> [b:Book];
```

Note that 'b' is an arbitrary object of type Book. The verbs 'picked up' and 'mastered' are of the following types, where "==>" is the Plastic notation for the functional arrow:

```
> [pickup : Human ==> (Phy ==> Prop)];
> [master : Human ==> (Info ==> Prop)];
```

With the above, we could interpret the sentences "John picked up a book" and "John mastered a book" separately and then use the predefined connective $and : Prop \to Prop \to Prop$ *to connect them. However, this would not be correctly the original sentence since the book picked up and that mastered must be the same book.*

We want 'and' to connect 'picked up' with 'mastered', so we consider 'and' as a generic semantic kind: for any type A, [[AND]](A) is of kind $A \to A \to A$. *For A being* $Human ==> (Book ==> Prop)$[7],

$$And = [[AND]](Human ==> (Book ==> Prop)).$$

In particular, the term "And pickup master" is well-typed, thanks to the coercive subtyping relations and the contravariance in subtyping function types as explained in section 2. Now, interpreting the indefinite article by means of the existential quantifier, the above sentence is interpreted as (in a readable notation)

$$\exists b : Book.\ And(pickup, master)\ John\ b.$$

The full code in Plastic is given in Appendix A.

[7] An alternative possibility is letting A be $Human ==> (PhyInfo ==> Prop)$

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. Cambridge University Press (2011)
4. Asher, N., Pustejovsky, J.: Word meaning and commonsense metaphysics (2005)
5. Callaghan, P., Luo, Z.: An implementation of LF with coercive subtyping and universes. Journal of Automated Reasoning 27(1), 3–27 (2001)
6. Cooper, R.: Copredication, Quantification and Frames. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 64–79. Springer, Heidelberg (2011)
7. Cooper, R.: Copredication, dynamic generalized quantification and lexical innovation by coercion. In: Proceedings of GL 2007, the Fourth International Workshop on Generative Approaches to the Lexicon (2007)
8. The Coq Development Team: The Coq Proof Assistant Reference Manual (Version 8.1), INRIA (2007)
9. Coquand, T., Huet, G.: The calculus of constructions. Infor. and Computation 76(2/3) (1988)
10. Luo, Y.: Coherence and Transitivity in Coercive Subtyping. Ph.D. thesis, University of Durham (2005)
11. Luo, Z.: Computation and Reasoning: A Type Theory for Computer Science. Oxford Univ. Press (1994)
12. Luo, Z.: Coercive subtyping. J. of Logic and Computation 9(1), 105–130 (1999)
13. Luo, Z.: Type-theoretical semantics with coercive subtyping. Semantics and Linguistic Theory 20 (SALT20), Vancouver (2010)
14. Luo, Z.: Contextual Analysis of Word Meanings in Type-Theoretical Semantics. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 159–174. Springer, Heidelberg (2011)
15. Luo, Z., Luo, Y.: Transitivity in coercive subtyping. Infor. and Computation 197(1-2) (2005)
16. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis (1984)
17. The Matita proof assistant (2008), http://matita.cs.unibo.it/
18. Nordström, B., Petersson, K., Smith, J.: Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press (1990)
19. Pustejovsky, J.: The Generative Lexicon. MIT (1995)
20. Pustejovsky, J.: A survey of dot objects (2005) (manuscript)
21. Pustejovsky, J.: Mechanisms of coercion in a general theory of selection (2011)
22. Ranta, A.: Type-Theoretical Grammar. Oxford University Press (1994)

A Plastic Code for Example 10

```
> import Sol_All;
> import FnCoercion;
> import SigmaCoercion;
> [Human, Phy, Info, Book :Type];
> [PhyInfo = Phy*Info];
> [cb:Book -> PhyInfo];
> Coercion = cb;
> [pickup:Human ==> (Phy ==> Prop)];
> [master:Human ==> (Info ==> Prop)];
> [John:Human];
> [b:Book];

("John picked up b and John mastered b")



> [pickup_b = ap_ Book Prop (ap_ Human (Book==>Prop)  pickup John) b];
> [master_b = ap_ Book Prop (ap_ Human (Book==>Prop)  master John) b];
> [sentence1 = and pickup_b master_b];

("John picked up and mastered b")

> [AND: (A:Type) (A->A->A)]
> [And = AND (Human ==> (Book ==> Prop))];
> [sentence2 = ap_ Book Prop (ap_ Human (Book==>Prop)
                        (And pickup master) John) b];

("John picked up and mastered a book")

> [sentence3 = Ex Book [b:Book](ap_ Book Prop (ap_ Human (Book==>Prop)
                        (And pickup master) John) b)];
```

Author Index