The Constraint Language for Lambda Structures

Markus Egg (egg@coli.uni-sb.de) and
Alexander Koller (koller@coli.uni-sb.de)
Dept. of Computational Linguistics, University of the Saarland

Joachim Niehren (niehren@ps.uni-sb.de)
Programming Systems Lab, University of the Saarland

Abstract. This paper presents the Constraint Language for Lambda Structures (CLLS), a first-order language for semantic underspecification that conservatively extends dominance constraints. It is interpreted over lambda structures, tree-like structures that encode λ-terms. Based on CLLS, we present an underspecified, uniform analysis of scope, ellipsis, anaphora, and their interactions. CLLS solves a variable capturing problem that is omnipresent in scope underspecification and can be processed efficiently.

Keywords: Underspecification, scope ambiguity, ellipsis, anaphora, tree descriptions.

1. Introduction

Underspecification (van Deemter and Peters, 1996; Pinkal, 1996) is a recent approach to controlling the combinatorial explosion caused by ambiguity. Its key idea is to derive a single, compact *description* of all readings instead of the (exponential number of) readings themselves. These descriptions are then used in later processing steps as much as possible; individual readings are only enumerated by need.

One type of ambiguity for which underspecification has been investigated particularly well is scope ambiguity. Common to many recent scope underspecification formalisms (Alshawi and Crouch, 1992; Reyle, 1993; Muskens, 1995; Bos, 1996; Niehren et al., 1997b; Egg et al., 1998) is that they provide the 'semantic material' of which the description of the readings is built, plus structural relations that constrain possible combinations of this material. *Dominance constraints*, a tree logic which can express dominance of nodes in a tree, are a very natural way for doing this. Muskens (1995) was the first to do this explicitly, but Reyle (1993) and Bos (1996) can be subsumed under this kind of approach as well. Dominance constraints also have many other applications in computational linguistics (Vijay-Shanker, 1992; Rambow et al., 1995; Gardent and Webber, 1998).

In this article, we show how to integrate a treatment of other linguistic phenomena – in particular, anaphora and VP ellipsis – with

© 2001 *Kluwer Academic Publishers. Printed in the Netherlands.*

a scope underspecification formalism based on dominance constraints. The interaction of these three phenomena has been considered before (Gawron and Peters, 1990; Crouch, 1995; Shieber et al., 1996). In our approach, it is modelled by a conservative extension of the language of dominance constraints by additional atomic formulas, viz., *anaphoric binding constraints* and *parallelism constraints*. While the former are based on an idea by Kehler (1993), the latter are formally equivalent to *context unification* as used in Niehren et al. (1997b), which was shown by Niehren and Koller (2001). In addition, λ-*binding constraints* are introduced in order to solve a capturing problem that appears in all dominance-based underspecification formalisms.

The combination of all of these constraints, called CLLS (Constraint Language for Lambda Structures), correctly describes many challenging cases from the literature that exhibit interaction of scope, anaphora, and ellipsis. In addition, its being an extension of dominance constraints makes for some appealing differences to existing formalisms with comparable coverage: Unlike Shieber et al. (1996), the CLLS analysis is completely order-independent; and in contrast to the QLF-based analysis in Crouch (1995), it achieves a clean separation between underspecified descriptions and the described readings.

The paper is structured as follows. In Section 2, we introduce the phenomena under consideration and present the intuitions behind our linguistic analysis without going into detail. Section 3 presents the formal definitions of lambda structures and CLLS and fills in the formal details that were omitted in Section 2. In Section 4, we extend the analysis to more involved cases from the literature. Section 5 defines a syntax/semantics interface that derives CLLS constraints on the basis of a simple phrase structure syntax. In Section 6, we give a brief overview over results on formal and computational aspects of CLLS. Finally, Section 7 concludes and outlines directions for future work.

2. Elements of CLLS

This section introduces the three main phenomena considered in the paper – scope, ellipsis, and anaphora – and the major concepts used in their CLLS analyses. After discussing lambda structures and their correspondence to λ-terms, we go through the phenomena and present the intuitions underlying our analysis. The section does not even attempt to be formally precise; the formal details are put off until section 3.

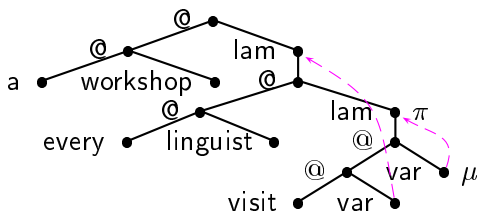
2.1. Lambda structures and lambda terms

The idea behind *lambda structures* is to represent a λ-term by a tree-like graph; a node in a lambda structure corresponds to an occurrence of a subterm in the λ-term it represents. Consider e.g. the λ-term in (1), which happens to represent one of the readings of (3) below:

(1)
```
(a workshop)(λx
    (every linguist)(λy
        (attend x) y))
```

We draw the corresponding lambda structure as the graph in (2). An occurrence of a λ-abstraction corresponds to a node labelled by lam, an occurrence of an application, to a node labelled by @, and an occurrence of a bound variable, to a node labelled by var. Variable binding is represented in a lambda structure by dashed arrows from the variables to their binders. For instance, the unique occurrence of λx in (1) corresponds to the node π in (2); it binds the variable represented by the node μ.

(2)



Note that variable names in a λ-term (such as x and y in (1)) are not reflected by the node labels of the corresponding λ-structure. Consequently, a lambda structure represents a closed λ-term uniquely modulo consistent renaming of bound variables (α-equality).

Via the detour through λ-structures, we can use tree descriptions in semantic underspecification: We simply talk about λ-structures in the same way one usually talks about trees. The explicit binding relation is important in that context because variable names may not be sufficient to ensure proper binding in an underspecified framework; we will come back to this in a minute.

2.2. Scope

The first class of phenomena we are concerned with is *scope ambiguity*, as in the following sentence.

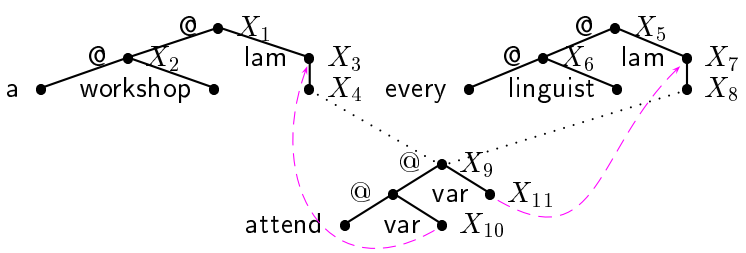
(3) Every linguist attends a workshop.

This sentence has two readings, which differ only in a permutation of quantifiers: Either the sentence means that there is one specific workshop which every linguist attends, or it means that every linguist attends a workshop, but not necessarily the same as the others. The first reading is represented by the λ-term in (1), and the second by (4):

(4)
```
(every linguist)(λy
      (a workshop)(λx
            (attend x) y))
```

Closer inspection reveals that (1) and (4), and hence their corresponding lambda structures, are closely related. The lambda structures are composed from the same tree-like subgraphs, corresponding to the term fragments (every linguist)(λx(.)), (a workshop)(λy(.)), and ((attend x) y). But these fragments are composed in different order. We can describe both lambda structures at once by specifying the fragments and their relationships. Such a description is given in (5).

(5)



Intuitively, description (5) is satisfied by all λ-structures into which the graph in (5) can be embedded in such a way that no labelled nodes of the graph overlap. Dotted edges in (5) signify *dominance*: Of the two nodes they connect, the upper one must be above the lower one in the lambda structure. The dashed arrows, for λ-binding, act like elastic bands, which can be stretched indefinitely without being broken by intervening lambdas.

The description leaves the relative ordering between the two quantifier fragments unspecified. But since both fragments dominate the nuclear scope and, just like trees, λ-structures cannot branch in the bottom-up direction, one of the two quantifier fragments must dominate the other. This situation is very common in scope underspecification. Note that variable binders must always dominate their bound variables; so the λ-binding relations, $\lambda(X_{10}) = X_3$ and $\lambda(X_{11}) = X_7$, logically entail the dominance relations, $X_4 \triangleleft^* X_9$ and $X_8 \triangleleft^* X_9$. Inserting these dominance edges in (5) only enhances readability.

In Section 3 we will give pictures like (5) a formal meaning as *constraints over λ-structures*, i.e. logical formulas that are interpreted

over the class of λ-structures. Then nodes of the graph correspond to variables denoting nodes of a λ-structure, whereas labels, edges, and arrows correspond to various types of atomic constraints.

We have said that a description like (5) is satisfied by all lambda structures into which they can be embedded. In particular, the lambda structure may contain additional fragments that are not mentioned in the description. This is a desired feature because we want to compute with limited information that can in principle be augmented at any later point; the fact that the information is complete must be expressed independently. As long as one is only concerned with scope, one can impose a general *constructiveness* restriction: each node in a lambda structure that we describe must be denoted by a variable in the description. But constructiveness would be too strict for our treatment of ellipses below and also for an underspecified account of reinterpretation (Egg, 2001; Koller et al., 2000b).

2.3. Ellipsis

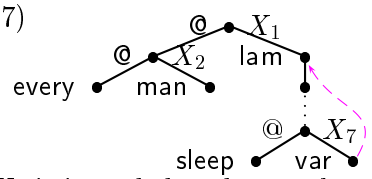
The second phenomenon we treat is *VP ellipsis*. VP ellipsis interacts with scope ambiguities, thus the treatment of both phenomena must be integrated. In the framework presented in this article, this integration simply amounts to adding further constraints on lambda structures.

For ellipses, we introduce *parallelism constraints*, which permit to express structural equality between fragments of λ-structures. As an example, consider the elliptical sentence (6):

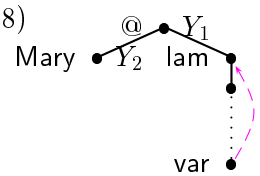
(6) Every man sleeps, and so does Mary.

The meaning of the target sentence *so does Mary* is exactly like the meaning of the source sentence *every man sleeps*, except that the contrasting element in the source sentence *(every man)* is replaced by the one in the target sentence *(Mary)*. This can be modelled in two steps. First, we describe the lambda structure for the source sentence, as in (7), and the lambda structure for the target parallel element, as in (8).

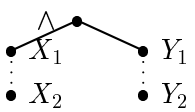
(7)



(8)



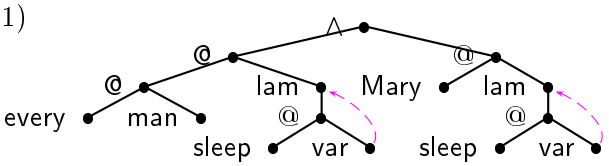
X_1 is intended to denote the root of the semantic representation for the source sentence, and Y_1, that of the target sentence representation. The λ-structure for the entire sentence (6) will be the conjunction of source and target sentence, like this:

(9) 

The second step is to impose the *parallelism constraint* (10), which says that the lambda structures for the source and target sentence (i.e. below X_0 and Y_0) are equal, with the exception of the substructure representing the contrasting elements (below X_2 and Y_2):

(10) $X_1/X_2 \sim Y_1/Y_2$

Indeed, the intended semantic representation of (6), shown in (11), simultaneously satisfies all constraints in (7), (8), (9), and (10).

(11) 

This shows that ellipsis resolution can be reduced to solving constraints over lambda structures.

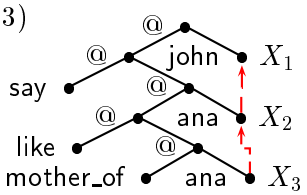
Since proper names and quantifier NPs can be contrasting elements in an ellipsis (as in (6)), our analysis requires type-raising of proper names; the constant Mary in (8) corresponds to a λ-term of type $\langle\langle e, t\rangle, t\rangle$. In the absence of ellipsis, we will freely analyze proper names as constants of type e anyway, as this simplifies the presentation. We will use all-lowercase labels mary in the latter case and capitalize the first letter in the former. Note that these details will be completely ignored by the constraint language, which will only speak about the label and not its type.

2.4. Anaphora

Anaphora are the final class of phenomena we consider. In order to represent anaphora, we extend our notion of lambda structures. Anaphora are represented as nodes labelled by ana which are linked to their antecedents by *anaphoric links*. We draw anaphoric links as dashed arrows using straight lines. For instance, a representation of (12) is shown in (13). For simplicity, we use a constant mother_of here; we will analyze the possessive differently later.

(12) John^i said he$_i^j$ liked his$_j$ mother.

(13)



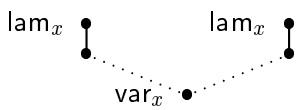
In the example, the two anaphoric links form a *link chain*. Link chains go back to Kehler (1993, 1995); they have been worked out in this setting by Xu (1998). This representation of anaphoric reference can cope with a series of problematic cases from the literature, such as strict/sloppy ambiguities, the many-pronouns puzzle (of which (12) is the source clause), and cascaded ellipses.

2.5. The Capturing Problem

Variable binding in λ-terms is usually indicated by using variable names: λx binds all occurrences of x in its scope. Even ordinary (i.e. non-underspecified) λ-calculus has to exclude the *capturing* of free variables by unintended binders through freeness conditions. In the context of underspecification, some additional problems occur. We solve these problems by using explicit binding functions in lambda structures.

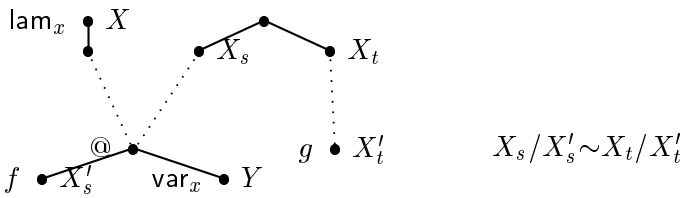
One problem occurs with constraints as they are used for scope ambiguities. Let us encode variable names into the node labels, i.e. we use new labels lam_x and var_x for each object-level variable x. Now consider the following constraint:

(14)



The constraint describes *two* binders lam_x, and there is a scope ambiguity between the two occurrences. In this case, it is completely unclear which the intended binder of a node with label var_x is; in each solution, the lower binder wins. The lesson is that variable names are not sufficient to indicate the binding relations when the structure of the term is not fully known. But in principle, this problem could be circumvented by naming variables apart when generating a description, at least so long as we only consider constructive solutions.

Similar problems become apparent in the presence of parallelism constraints, e.g. in the following constraint:

(15) 

When we solve this constraint, we must introduce a copy of Y. What label should this copy have? If X takes scope over X_t, it should probably be var_x; if X ends up below X_s, it should probably be var_u for a fresh variable name u, and X should get a copy lam_u. It is probably possible to define the semantics of parallelism constraints on this basis, although it will become more complicated. But the necessary variable name management will definitely make constraint solving much more difficult. At best, we can work around the problems created by the variable names; we do not gain anything from having them.

On the other hand, explicit binding functions provide a simple, clean, and general solution to the problem which are very helpful in processing (Erk et al., 2001). Indeed, they are so powerful that one can even lift β-reduction to the level of underspecified descriptions without ever worrying about freeness conditions or variable capturing, even in cases where this would be necessary in the ordinary λ-calculus (Bodirsky et al., 2001a).

3. Syntax and Semantics of CLLS

Section 2 presented the general intuitions behind lambda structures. We showed how to describe λ-structures and how to apply their descriptions in an underspecified analysis of scope, ellipsis, and anaphora. Now we give a rigorous definition of the constraint language for lambda structure (CLLS), which we will use throughout the paper. We will start with the semantic level by first presenting tree structures, then move over to lambda structures as an extension of tree structures. Once the semantics is fixed, a definition of a syntax to talk about it is simple.

3.1. Tree Structures

A tree corresponds to a unique tree structure in the same way that a lambda structure relates to a unique λ-term modulo α-equivalence. Lambda structures are extended tree structures because λ-terms are extended trees (trees plus variable binding).

Let Σ be a set of *function symbols*, f, g, a, b. Each function f symbol comes with a fixed *arity*. We write f^k for a function symbol with arity

$k \geq 0$. A (finite constructor) tree is simply a ground term (without variables) built from a set of function symbols. For instance, $f(g(a,a))$ is a tree built from function symbols f^1, g^2, a^0. Taking $\mathbb{N}$ to be the set of natural numbers starting at 1, we can define a *path* as a word over $\mathbb{N}$. We identify each node of a tree with the path from the root to this node; the empty word, ϵ, identifies the root. Paths are named with lowercase Greek letters, such as $\pi, \pi_1, \pi_2, \mu, \mu_1$. Concatenation of words π and μ is written as $\pi\mu$, and a word π is a *prefix* of μ iff there is a word π_1 such that $\mu = \pi\pi_1$.

A tree structure is a first-order model structure which characterizes a tree. The domain of a tree structure contains all nodes of the corresponding tree, and its relations specify labelling and childhood. E.g., the domain of the tree structure for $f(g(a,a))$ is $\{\epsilon, 1, 11, 12\}$. The fact that the root ϵ is labelled with f and has a single child 1 is expressed by a relation we write as $\epsilon{:}f(1)$; the children and labels of the remaining nodes are specified by the relationships $1{:}g(11,12)$, $11{:}a$, and $12{:}a$.

More generally, a *tree domain* Δ is a finite nonempty set of nodes which is prefixed-closed (that is, $\pi\mu \in \Delta$ implies $\pi \in \Delta$) and closed under left siblings (i.e. if $\pi i \in \Delta$ then $\pi j \in \Delta$ for all $1 \leq j < i$), and a tree structure is defined as follows:

DEFINITION 3.1. *A* tree structure *over* Σ *consists of a tree domain* $\Delta \subseteq \mathbb{N}^*$ *and a* labelling relation $:f \subseteq \Delta^{n+1}$ *for each* n*-ary function symbol* $f \in \Sigma$*, such that for all nodes* $\pi \in \Delta$ *and the maximal* $n \in \mathbb{N}$ *with* $\pi n \in \Delta$*, there is a unique symbol* $f^n \in \Sigma$ *such that* $\pi{:}f(\pi 1, \ldots, \pi n)$.

Given nodes $\pi_0, \ldots, \pi_n$, we write $\pi_0{:}f(\pi_1, \ldots, \pi_n)$ for $(\pi_0, \pi_1, \ldots, \pi_n) \in :f$. This relation expresses that π_0 is a node of the tree structure with the label f and the children $\pi_1, \ldots, \pi_n$, in this order. Note that two nodes may be labelled by the same constant without being equal. The condition in the definition means that the number of children of a node is specified by the arity of its label.

It is maybe important to mention here that using tree domains is only one way to specify trees; it is just as possible to simply define trees as ground terms, or as specific graphs, and this has been done in other papers on CLLS. The distinction is immaterial because we are only interested in the tree structure that is induced, and one can use whatever underlying object is most convenient.

3.2. Lambda Structures

A lambda structure extends a tree structure with two additional relations for λ-binding and anaphoric links. For lambda structures, we assume:

$$\{\mathsf{var}^0, \mathsf{ana}^0, \mathsf{lam}^1, @^2\} \subseteq \Sigma$$

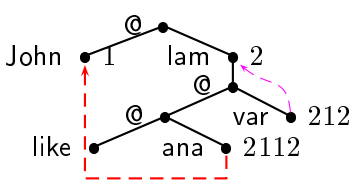
DEFINITION 3.2. *A* lambda structure *over* Σ *consists of a tree structure over* Σ *with tree domain* Δ *and two additional binary relations* λ *and* ante *on* Δ *which satisfy the following properties.*

1. $\lambda \subseteq \Delta \times \Delta$ *is a partial function that maps every node* $\pi \in \Delta$ *with label* var *to a prefix of* π *which is labelled with* lam*.*
2. $\mathsf{ante} \subseteq \Delta \times \Delta$ *is a partial function that maps every node with label* ana *to another node.*

The lambda structures we are interested in here have additional properties, which we have not included in the definition because they are not essential for what follows: For instance, chains of anaphoric links will not have cycles.

We draw lambda structures as tree-like graphs. E.g., the binding and anaphoric linking functions of the lambda structure (16) are specified by $\lambda(212) = 2$ and $\mathsf{ante}(2112) = 1$.

(16)



Lambda structures and λ-terms can almost be mapped to each other one-to-one. The two exceptions are that lambda structures do not mention variable names and that lambda structures contain anaphoric links. We do not elaborate on the further interpretation of anaphoric links here; roughly, they express that the interpretation of the last element of the link chain determines the interpretation of all other elements of the chain.

3.3. Dominance and Parallelism

Finally, we conservatively extend lambda structures by two additional relations between nodes: *dominance* and *parallelism*. These relations

were not included in Def. 3.2, as they can be defined from the relations specified there.

Let Λ be a lambda structure and π, μ two of its nodes. We say that π *dominates* μ ($\pi \lhd^* \mu$) if it lies above it, i.e. if π is a prefix of μ. The dominance relation is a partial order on the domain of Λ, i.e. reflexive, transitive, and antisymmetric.

For the definition of parallelism, we first call any pair π/μ of nodes π, μ in Λ with $\pi \lhd^* \mu$ a *segment* of Λ. π is called the *root* and μ the *hole* of the segment. We define

$$\begin{aligned} \mathsf{b}^-(\pi/\mu) &:= \{\pi' \mid \pi \lhd^* \pi', \text{ but not } \mu \lhd^* \pi'\} \\ \mathsf{b}(\pi/\mu) &:= \mathsf{b}^-(\pi/\mu) \cup \{\mu\} \end{aligned}$$

Intuitively, these are sets of nodes *between* π and μ; the difference is that the hole belongs to $\mathsf{b}(\pi/\mu)$, but not to $\mathsf{b}^-(\pi/\mu)$. Taking the labelling relations of Λ into account, $\mathsf{b}^-(\pi/\mu)$ looks almost like a tree domain starting at π, except that one node (the mother of μ) has one too few children.

Now we can define *correspondence functions* between two segments, which bijectively map the nodes of one segment to those of the other while respecting the underlying tree structure.

DEFINITION 3.3. *Let Λ be a lambda structure over Σ and $\pi_1, \mu_1, \pi_2, \mu_2$ nodes of its domain Δ. A bijective function $c : \mathsf{b}(\pi_1/\mu_1) \to \mathsf{b}(\pi_2/\mu_2)$ is called a* correspondence function *between π_1/μ_1 and π_2/μ_2 iff for all $f^n \in \Sigma$ and $\pi \in \mathsf{b}^-(\pi_1/\mu_1)$,*

$$\pi{:}f(\pi 1, \ldots, \pi n) \Leftrightarrow c(\pi){:}f(c(\pi)1, \ldots, c(\pi)n).$$

In particular, correspondence functions map roots to roots and holes to holes. For all $\pi \in I\!N^*$, $\pi_1\pi \in \mathsf{b}(\pi_1/\mu_1)$ iff $c(\pi_1\pi) = \pi_2\pi$; that is, corresponding nodes can be reached over the same path from the roots of their respective segments.

The parallelism relation extends correspondence with conditions on parallel binding.

DEFINITION 3.4. Parallelism $\pi_1/\mu_1 \sim \pi_2/\mu_2$ *holds in Λ iff there is a correspondence function c between π_1/μ_1 and π_2/μ_2 such that* **P1**–**P4** *are satisfied.*

P1 *Binding within the segments is parallel:*

$$\forall \pi, \mu \in \mathsf{b}^-(\pi_1/\mu_1) : \ \lambda(\mu){=}\pi \Leftrightarrow \lambda(c(\mu)){=}c(\pi)$$

P2 *For $i \in \{1, 2\}$, no node below μ_i is bound by a node inside the respective segment:*

$$\forall \pi : \ \mu_i \triangleleft^* \pi \Rightarrow \lambda(\pi) \notin \mathsf{b}^-(\pi_i/\mu_i)$$

P3 *If Λ binds a variable node in $\mathsf{b}^-(\pi_1/\mu_1)$ properly above π_1 or the corresponding variable node in $\mathsf{b}^-(\pi_2/\mu_2)$ properly above π_2, then the two binder nodes are* binding equivalent *('$\triangleleft^+$' denotes proper dominance).*

$$\forall \pi \in \mathsf{b}^-(\pi_1/\mu_1) : \ (\lambda(\pi) \triangleleft^+ \pi_1 \vee \lambda(c(\pi)) \triangleleft^+ \pi_2) \Rightarrow \lambda(\pi) \approx \lambda(c(\pi))$$

For the time being, assume that binding equivalence $\approx$ is simply node identity. We will modify this definition in Section 4.5.

P4 *If π is an anaphoric node in $\mathsf{b}^-(\pi_1/\mu_1)$, then the corresponding anaphoric node in $\mathsf{b}^-(\pi_2/\mu_2)$ must be linked either to π (strict) or, if it exists, to the correspondence of $\mathsf{ante}(\pi)$ (sloppy).*

$$\forall \pi \in \mathsf{b}^-(\pi_1/\mu_1) : \ \mathsf{ante}(c(\pi)) = \pi \ \vee \ \mathsf{ante}(c(\pi)) = c(\mathsf{ante}(\pi)).$$

The parallelism relation is almost symmetric; the only exception is that the strict variant of the rule **P4** only allows to link the anaphoric node in $\mathsf{b}^-(\pi_2/\mu_2)$ to the corresponding node in $\mathsf{b}^-(\pi_1/\mu_1)$, but not vice versa.

3.4. The Constraint Language CLLS

Now that lambda structures and their relations are defined, it is straightforward to fix a constraint language to go with them. We assume an infinite set of *node variables*, ranged over by X, X_i, Y, etc., and pick relation symbols for all relations defined so far. To keep the presentation simple, we use the same symbols as for the relations themselves. The constraint language over lambda structures (CLLS) is given by the following abstract syntax:

$$\begin{array}{rcll} \varphi & ::= & X{:}f(X_1, \ldots, X_n) \mid X \triangleleft^* Y \mid X \neq Y & (f^n \in \Sigma) \\ & \mid & \lambda(X){=}Y \mid \mathsf{ante}(X){=}Y \mid X/X' {\sim} Y/Y' & \\ & \mid & \varphi \wedge \varphi' & \end{array}$$

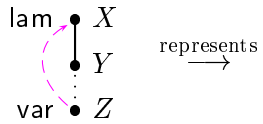
The semantics of CLLS is defined by interpretation of constraints over the class of lambda structures in the usual Tarski style: A pair of a lambda structure Λ and a variable assignment α into the domain of Λ satisfies a constraint φ iff it satisfies each atomic conjunct in the obvious way. We call (Λ, α) a *solution* of φ in this case.

Abbreviations we will occasionally use below are $X{=}Y$ for $X \triangleleft^* Y \wedge Y \triangleleft^* X$ (node equality) and $X \triangleleft^+ Y$ for $X \triangleleft^* Y \wedge X \neq Y$ (proper dominance).

3.5. Constraint Graphs

CLLS constraints as just defined can become unreadable rather quickly. Throughout the paper, we will use *constraint graphs* instead. Constraint graphs are an alternative, graphical syntax for CLLS; a constraint graph corresponding to a constraint φ contains one node for each variable in φ and labels and edges for the atomic constraints that appear in φ. A very simple example is given in (17).

(17)



represents $\longrightarrow$ $X{:}\mathsf{lam}(Y) \wedge Y \lhd^* Z \wedge$ $Z{:}\mathsf{var} \wedge \lambda(Z){=}X \wedge$ $X{\neq}Z$

For every dominance constraint $X \lhd^* Y$, we draw a dotted line from X to Y. For every labelling constraint $X{:}f(X_1, \ldots, X_n)$, we label the graph node for X with f and draw solid lines to the nodes for $X_1, \ldots, X_n$. For every λ-binding constraint $\lambda(X){=}Y$, we draw a curved dashed arrow from X to Y; and for every anaphoric binding constraint $\mathsf{ante}(X){=}Y$, we draw a dashed arrow from X to Y that only uses straight lines. Parallelism constraints cannot be represented easily in constraint graphs; we will simply write them next to the graph.

Finally, a constraint graph implicitly represents inequality constraints $X_1{\neq}X_2$ for each two labelled nodes in the graph. This assumption nicely prevents 'overlaps' between the 'fragments' of the constraint graph.

Now we can interpret the intuitive diagrams from Section 2 as CLLS constraint graphs. This gives them a formal meaning as CLLS constraints, which can be satisfied by lambda structures. These lambda structures, in turn, represent λ-terms which stand for the actual meaning(s) of the sentence.

4. Interaction of Quantifiers, Anaphora, and Ellipsis

Now that we know how to deal with simple examples, we turn to some well-known, more complex challenges in the area of quantifiers, anaphora, and ellipses, involving in particular their interactions. We look into quantifier parallelism, strict/sloppy ambiguities, nested ellipses, a sentence involving interaction of all three, and antecedent-contained deletion (ACD). There are other interesting things that the analysis can do; e.g., to deal with gapping, parallelism could be extended to allow segments with *two* holes rather than one. This is realized in newer variants of parallelism constraints for different reasons (Bodirsky et al., 2001a).

Throughout, we assume a fixed signature:

$$\Sigma = \{@^2, \mathsf{lam}^1, \mathsf{var}^0, \mathsf{ana}^0, \mathsf{before}^2, \mathsf{mary}^0, \mathsf{read}^0, \ldots\}$$

We follow the convention that proper nouns are always analyzed as constants of type e, except as contrasting elements in ellipses where the other contrasting element is a quantifier. We indicate the first analysis by labelling the corresponding nodes in lowercase letters (mary), the second one, by using labels with a capitalized first letter (Mary). This only makes the graphs more readable; the analyses would of course go through with higher-order names as well. For the same reason we draw (logically redundant) dominance edges wherever dominance is entailed by λ-binding constraints.

4.1. Quantifier Parallelism

The first phenomenon we consider is *quantifier parallelism*, an interaction of scope and ellipsis. By way of example, we continue the sentence (3), repeated here as (18), with the elliptical sentence (19).

(18) Every linguist attends a workshop.

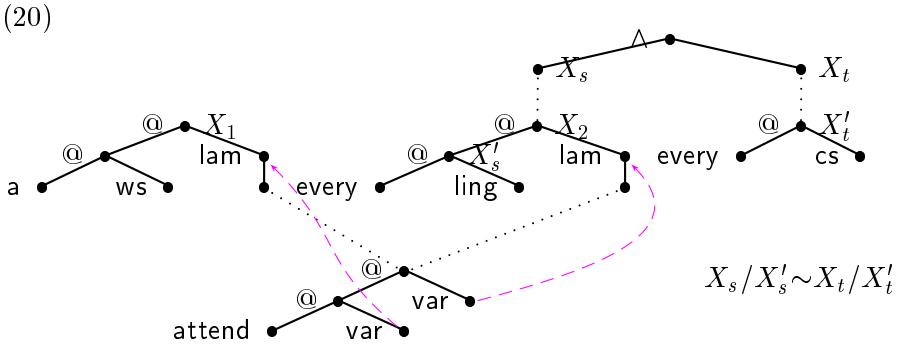
(19) Every computer scientist does, too.

This pair of sentences, modelled after an example by Hirschbühler (1982), has three readings: Either there is a single workshop which both all linguists and all computer scientist attend; or there is one workshop for all linguists and one for all computer scientists (but not necessarily the same one); or every linguist and computer scientist can attend a different workshop.

If one simply copies the surface form of the source sentence, the target sentence reads "Every computer scientist attends a workshop". Now both sentences contain a scope ambiguity, so we would wrongly predict at least four readings. The problem is that the scope ambiguities cannot be resolved independently; copies of the same quantifier must have analogous scope.

In CLLS, this is automatically enforced by the parallelism constraint. We represent the meaning of the sentences by the following constraint (abbreviating workshop to ws, linguist to ling, and computer_scientist to cs):

(20)



X_s is the root of the semantics of the source sentence; X_t is the root of the target sentence. The contrasting elements, *every linguist* and *every computer scientist*, are below the variables X'_s and X'_t, respectively. There is a dominance constraint between X_s and X_2, but not between X_s and X_1, since, like most quantifiers, universal quantifiers must obey a 'scope island constraint' that prevents them from escaping from a sentence. Such a constraint is not imposed on the indefinite *a workshop*, so it can take scope over the entire conjunction.

There are three places where the fragment below X_1 can go in a lambda structure satisfying the constraint: below the fragment starting at X_2, between X_s and X_2, and above the conjunction. In each case, the parallelism constraint enforces that whatever is below X_s is also below X_t, replacing what is below X'_s with what is below X'_t. This produces exactly the three correct readings of the sentence.

A particular consequence of the parallelism constraint enforcing structural parallelism of the lambda structures below X_s and X_t is that the relative scopes of the quantifiers must be identical in both the source and the target sentence (if the indefinite is below X_s, of course).

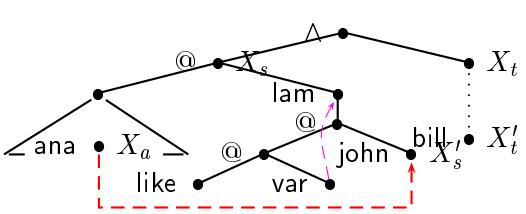
4.2. Strict/sloppy Ambiguities

Strict/sloppy ambiguities are ambiguities of anaphoric reference that occur in the context of VP ellipses. They are an important benchmark for analyses of ellipsis. The prototypical example looks as follows:

(21) John likes his mother, and Bill does too.

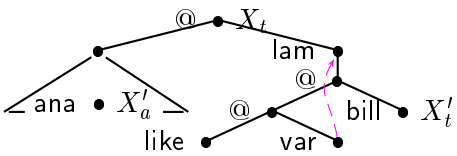
This sentence has two readings: Either Bill likes John's mother (this is the *strict* reading), or Bill likes his own mother (the *sloppy* reading).

We describe the meaning of (21) as in (22), using parallelism and anaphoric linking constraints. We have hidden the subconstraint for *his mother* and only drawn an empty triangle, as the only node we are interested in is the anaphor.

(22) 

$X_s/X'_s \sim X_t/X'_t$

Except for the fact that the source clause contains an anaphor, this constraint is totally analogous to the simple ellipses we considered above. The parallelism constraint first enforces that the tree part of the lambda structure below X_t is the same as the tree part of the one below X_s, except for the contrasting elements, as follows:

(23) 

(23) is not a complete lambda structure because the anaphor at X'_a does not have an antecedent. According to condition P4 of Definition 3.4, there are two legal antecedents that this anaphor can be linked to. It can either be linked to the anaphor in the source context, i.e. to the node denoted by X_a. In this case, the link chain starting at X'_a ends at X_s, which is labelled with john; so this is the strict reading. Or it can be linked to the node corresponding to the antecedent of X_a. As X_a is linked to the hole of the source context, X'_a must be linked to the hole of the target context; this is X'_t, which produces the sloppy reading.

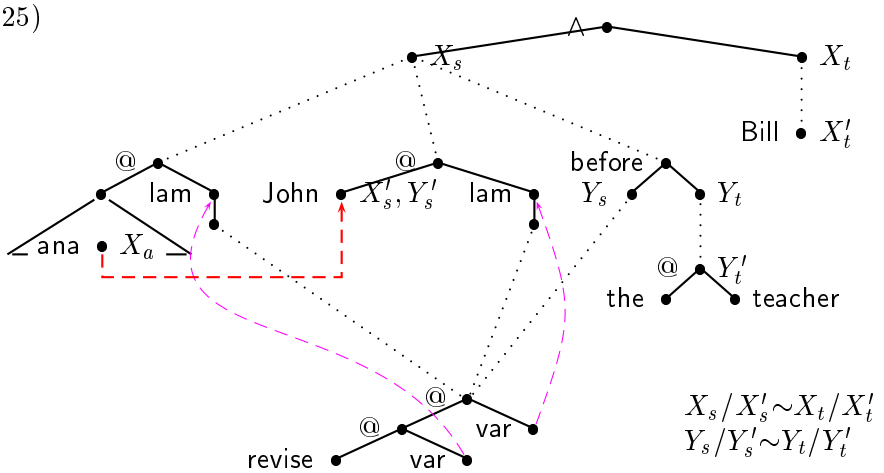
4.3. Nested Ellipses

This analysis of strict/sloppy ambiguity carries over to more sophisticated cases like the Gawron and Peters (1990) example

(24) John revised his paper before the teacher did, and so did Bill.

This sentence comprises nested ellipsis: the source clause *John revised his paper before the teacher did* of the ellipsis is elliptical itself. It is further complicated by the anaphor, which induces a complex strict/sloppy ambiguity. We follow Dalrymple et al. (1991) in assuming five readings (listed in (26)) for (24). These five readings are described by the constraint in (25).

(25)



For readability, we have again compressed the subconstraint describing *his paper* into an empty triangle; the only visible (and interesting) node is X_a, the anaphor. The constraint contains *two* parallelism constraints, one for each ellipsis. Note that the source sentence seems to contain a scope ambiguity between the three fragments corresponding to *his paper*, *John*, and *before*. However, the constraint entails that *before* takes scope over *John* because $Y_s/Y'_s \sim Y_t/Y'_t$ entails $Y_s \lhd^* Y'_s$. The remaining scope ambiguity between *his paper* and the other two fragments is spurious; we chose to leave it in the constraint to be closer to what the syntax/semantics interface (Section 5) would produce.

To understand how the five (constructive) solutions come about, let us try to 'solve' the constraint by mimicking first resolution of the inner and then of the outer ellipsis. An actual solution algorithm might behave differently; in particular, it could interleave resolution steps for the inner and outer parallelism constraint and other constraints such as the dominance constraints for a scope ambiguity, as the formalism is totally declarative. We call the correspondence function on the parallel contexts that is induced by the inner parallelism constraint c_i, and that of the outer one, c_o.

Resolution of the parallelism constraint for the inner ellipsis may proceed as follows. Let us say that the node denoted by the variable for the original anaphor is π_a; then its copy via this parallelism constraint is $c_i(\pi_a)$. There are two options where $c_i(\pi_a)$ can be linked: either to π_a (strict – this produces the first three readings in (26)) or to μ'_t, the denotation of Y'_t (sloppy – this produces the other two readings).

If we now resolve the parallelism constraint for the outer ellipsis, we produce copies of π_a and of $c_i(\pi_a)$: $c_o(\pi_a)$ and $c_o(c_i(\pi_a))$. The node $c_o(\pi_a)$ can be linked either to π_a (strict) or to π'_t, the denotation of X'_t

(sloppy). Likewise, $c_o(c_i(\pi_a))$ can be linked either to $c_i(\pi_a)$ (strict), or it can be linked to the respective value of $c_o(\mathsf{ante}(c_i(\pi_a)))$ (sloppy).

So there are five possible combinations of linking relations:

(26)
1. $\mathsf{ante}(c_i(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_i(\pi_a)$ and $\mathsf{ante}(c_i(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_o(\pi_a)$:
   John revised *John's* paper before the teacher revised *John's* paper,
   Bill revised *John's* paper before the teacher revised *John's* paper
2. $\mathsf{ante}(c_i(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(\pi_a))=\pi'_t$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_i(\pi_a)$:
   John revised *John's* paper before the teacher revised *John's* paper,
   Bill revised *Bill's* paper before the teacher revised *John's* paper
3. $\mathsf{ante}(c_i(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(\pi_a))=\pi'_t$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_o(\pi_a)$:
   John revised *John's* paper before the teacher revised *John's* paper,
   Bill revised *Bill's* paper before the teacher revised *Bill's* paper
4. $\mathsf{ante}(c_i(\pi_a))=\mu'_t$, $\mathsf{ante}(c_o(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_i(\pi_a)$ and $\mathsf{ante}(c_i(\pi_a))=\mu'_t$, $\mathsf{ante}(c_o(\pi_a))=\pi_a$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_o(\mu'_t)$:
   John revised *John's* paper before the teacher revised *the teacher's* paper,
   Bill revised *John's* paper before the teacher revised *the teacher's* paper
5. $\mathsf{ante}(c_i(\pi_a))=\mu'_t$, $\mathsf{ante}(c_o(\pi_a))=\pi'_t$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_i(\pi_a)$ and $\mathsf{ante}(c_i(\pi_a))=\mu'_t$, $\mathsf{ante}(c_o(\pi_a))=\pi'_t$, $\mathsf{ante}(c_o(c_i(\pi_a)))=c_o(\mu'_t)$:
   John revised *John's* paper before the teacher revised *the teacher's* paper,
   Bill revised *Bill's* paper before the teacher revised *the teacher's* paper

A reading where π_a, $c_i(\pi_a)$, and $c_o(\pi_a)$ are all linked to John and $c_o(c_i(\pi_a))$ is linked to Bill, which is particularly difficult to exclude in other approaches, is impossible here, as $c_o(c_i(\pi_a))$ must be linked either to $c_i(\pi_a)$ or to $c_o(\pi_a)$.

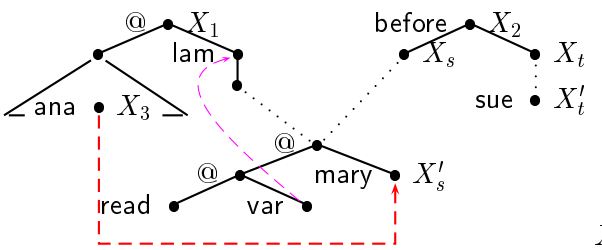
4.4. A Complex Interaction

Now we consider an example involving quantification, anaphora, and ellipsis where all three phenomena interact.

(27) Mary read a book she liked before Sue did.

This sentence, modelled after a Gawron and Peters (1990) example, has three readings. In the first, the indefinite NP *a book she liked* takes wide scope over both clauses (a particular book liked by Mary is read by both Mary and Sue). The second and third arise from the strict/sloppy ambiguity that occurs if the operator *before* outscopes the indefinite: either both read a book that Mary liked, or each read a book she liked herself. A constraint describing these three readings is shown in (28).

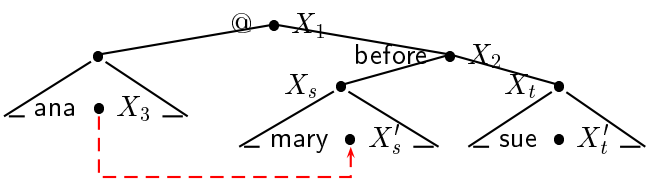
(28)



For the sake of readability, we once again simplify the semantics of *a book she liked* to an empty triangle with the anaphoric node X_3.

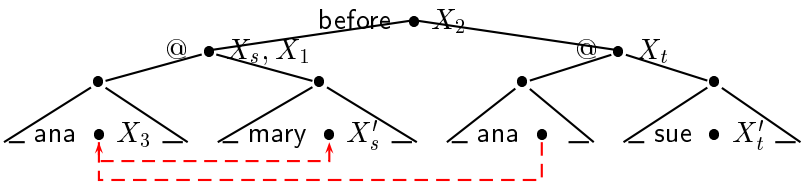
The first reading, with the NP taking wide scope, results when the relative scope of X_1 and X_2 is resolved such that X_1 dominates X_2. In this case, the anaphoric node is not part of the parallel subdomain, which means that there is no strict/sloppy ambiguity. A schematic representation of this solution (in which we have compressed even more sub-lambda structures into empty triangles) is shown in (29).

(29)

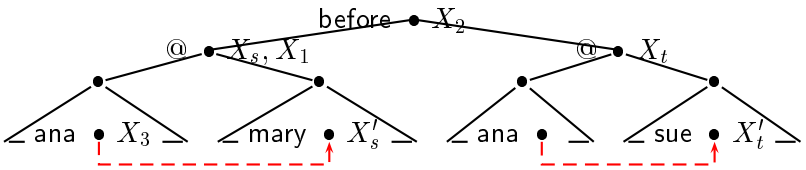


The other possible scoping is for X_2 to dominate X_1. The two solutions this gives rise to are displayed in (30) and (31). Now the interpretation of the indefinite, below X_1, is dominated by X_s and is copied by resolving the parallelism constraint. In particular, the anaphoric node, denoted by X_3, is copied; the strict/sloppy ambiguity of its copy is dealt with in the now familiar way.

(30)



(31)



4.5. Antecedent-Contained Ellipsis

We conclude this section with a notoriously difficult case of ellipsis, *antecedent-contained deletion* (ACD), as in (32).

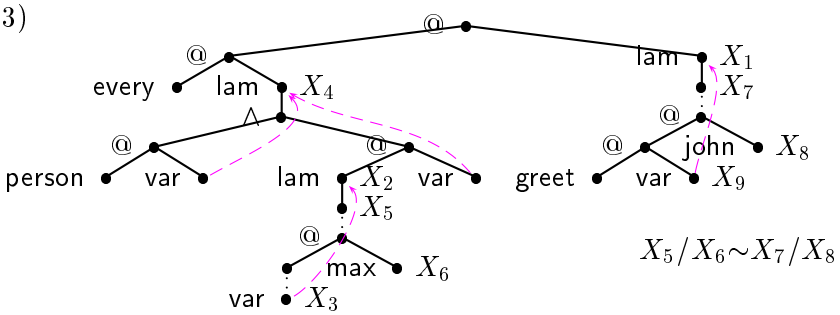
(32) John greeted every person that Max did.

The problem here is is that the ellipsis is contained in the VP that it refers to (its 'antecedent'). A naive approach to ellipsis that simply copies phonetic material from the source to the target sentence runs into an infinite loop as there is always an elliptic sub-sentence left.

Generative syntax solves this problem by raising the universal quantifier out of the sentence in LF (Sag, 1980; Fiengo and May, 1994); this has the effect that the ellipsis is no longer contained in the copied material. Dalrymple et al. (1991) and Crouch (1995) avoid a similar kind of recursion by an 'occurs check' of higher-order unification.

In CLLS, the meaning of (32) can be described as follows.

(33)



Like in the other analyses, the universal quantifier is raised out of the source segment in the CLLS analysis. The equivalent of an occurs check failure in our approach is the inclusion of the target segment in the source segment. This would happen here if we chose the root of the source segment to be not X_5 but the root of the entire sentence, and it would make the constraint unsatisfiable because we interpret over finite lambda structures.

There is one subtle problem, however. Condition P3 in Definition 3.4 requires that if π is labeled with var, the binders of π and $c(\pi)$ must be *binding equivalent*, which we have taken to mean 'equal' so far. However, equality is too strong a restriction for ACD: The var-node X_3 is bound by X_2, but X_2 cannot bind the copy of X_3 because it does not dominate it. Intuitively, it should be permitted for the copy (X_9) to be bound by X_1 because if it is, a sequence of β-reductions will map X_3 and its copy to the same variable. The separation of the two binders is merely an artifact of the type-raised analysis of *every*.

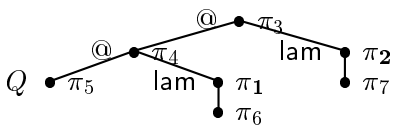
Incorporating a notion of $\beta\eta$-equivalence into the semantics of the parallelism constraint would, however, be very inconvenient. It would be conceptually questionable, as the constraint would have to speak at the same time about a lambda structure and about an equivalence class of lambda structures, and it would probably also be unpleasant from a computational point of view. So we approximate the intended binding behaviour by the following revised definition of binding equivalence, which works for the examples.

DEFINITION 4.1. *Let Λ be a lambda structure.* Binding equivalence in Λ *is the smallest equivalence relation $\approx$ on nodes in the domain of Λ such that for all $\pi_1, \ldots, \pi_7$,*

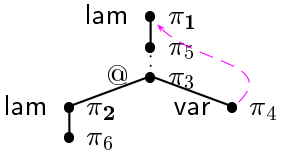
1. *if* $\pi_1{:}\mathsf{lam}(\pi_6)$, $\pi_2{:}\mathsf{lam}(\pi_7)$, $\pi_3{:}@(\pi_4, \pi_2)$, $\pi_4{:}@(\pi_5, \pi_1)$, *and* $\pi_5{:}Q$, *where Q is a quantifier label, i.e. a member of* $\{\mathsf{every}, \mathsf{a}, \ldots\}$, *then* $\pi_1 \approx \pi_2$;

2. *if* $\pi_3{:}@(\pi_2, \pi_4)$, $\pi_1{:}\mathsf{lam}(\pi_5)$, $\pi_2{:}\mathsf{lam}(\pi_6)$, $\pi_4{:}\mathsf{var}$, *and* $\lambda(\pi_4){=}\pi_1$, *then* $\pi_1 \approx \pi_2$.

(34) (a)



(b)



The first branch of the definition models β-equivalence; the precondition is shown in (34a). The second branch models η-equivalence; its precondition is shown in (34b). Incorporating the meanings of the quantifier node labels in this way is admittedly somewhat clumsy, but under the modified definition, the constraint in (33) receives exactly the correct solution. Finding a more general approach is an open problem.

Note, by way of conclusion, that our analysis can also account for the difference between (35) and (36) (Sag, 1980), the first of which lacks one of the two readings of the second one (where the matrix verb *want* outscopes the universal quantifier).

(35) John wants Bill to read everything that Max does.

(36) John wants Bill to read everything Max wants him to read.

This blocking is achieved in the CLLS analysis because, as discussed above, *John* must be in the scope of the universal quantifier (it is the contrasting element). As every NP argument outscopes the verb of which it is a syntactic argument, *John* must outscope *want*; so

the relative scope of the universal quantifier and *want* is fixed. If the elliptic material of (35) is spelt out like in (36), there is no parallelism constraint to enforce wide scope of the universal quantifier over *John*.

5. The Syntax-Semantics Interface

This section defines a grammar with a syntax/semantics interface which allows the derivation of the CLLS constraints for the example sentences discussed in this paper. We will first define the grammar and then the rules for semantic construction and then go through an example to show how the interface works.

We make two simplifying assumptions throughout. First, we disregard all aspects of syntactic analysis that are inconvenient for a pure phrase structure grammar. Any serious NLP system would employ some type of unification grammar, in which these aspects could be taken care of easily. As a matter of fact, we have implemented an HPSG grammar that does this and produces essentially the same constraints as the one presented here.

Second, we assume that the syntax provides coindexation that relates anaphoric pronouns to their antecedents and relative pronouns to their corresponding traces, and that the parallelism constraints that model VP ellipsis are generated by an independent source. Determining elliptic elements is, to our knowledge, an open problem and therefore beyond the scope of this paper.

5.1. Grammar

The grammar is defined by the phrase structure rules in Fig. 1.

(a1) $\texttt{S} \rightarrow \texttt{NP VP}$

(a2) $\texttt{VP} \rightarrow \texttt{IV}$

(a3) $\texttt{VP} \rightarrow \texttt{TV NP}$

(a4) $\texttt{VP} \rightarrow \texttt{AV NP VP}$

(a5) $\texttt{VP} \rightarrow \texttt{SV S}$

(a6) $\texttt{VP} \rightarrow \texttt{VP Conj S}$

(a7) $\texttt{NP} \rightarrow \texttt{PN}$

(a8) $\texttt{NP} \rightarrow \texttt{Det}\ \overline{\texttt{N}}$

(a9) $\overline{\texttt{N}} \rightarrow \texttt{N}$

(a10) $\overline{\texttt{N}} \rightarrow \texttt{N RCl}$

(a11) $\texttt{RCl} \rightarrow \texttt{RP S}$

(a12) $\texttt{S} \rightarrow \texttt{S Coord S}$

(a13) $\alpha \rightarrow W$ if $(W, \alpha) \in Lex$

Figure 1. The grammar

Most labels for syntactic categories are self-explanatory, perhaps except for the following: `PN` is a category that comprises proper nouns and anaphora, `Conj` refers to the category of conjunctions like *before*, `Coord`, to the one of sentence coordinators like *and*, and `RP` and `RCl`, to the categories of relative pronoun and relative clause, respectively. Beyond transitive and intransitive verbs, we also allow 'AcI verbs' `AV` (such as *expect*) and sentence complement verbs `SV` (such as *say*). `Conj` and `Coord` are distinguished because they percolate scope domains in a different fashion.

The lexicon is defined by a relation Lex, which relates words W and lexical categories $\alpha \in \{\texttt{Det}, \texttt{N}, \texttt{IV}, \texttt{TV}, \texttt{SV}, \texttt{RP}, \ldots\}$. Terminal productions **(a13)** expand lexical categories to words of this category.

5.2. Semantic construction

The overall strategy of our syntax-semantics interface is to factor out as much of the constraint construction as possible into the interface rules. This allows us to keep the entries in the semantic lexicon extremely simple: most of them introduce just one labelling constraint. We generate a constraint for each node in the syntax tree; the constraint for the entire tree will then be simply the conjunction of these subconstraints.

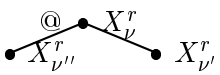
Each node $\nu \in \mathbb{N}^*$ in the syntax tree is associated with two variables, X^s_ν (the *local scope domain* of ν) and X^r_ν (the *root* of the subconstraint for ν). Intuitively, X^r_ν denotes the root of the 'semantic contribution' of the syntactic constituent below ν.

X^s_ν is referred to in scope island constraints: It is used to force quantifiers in embedded sentences not to be raised outside these embedded sentences. We make the simplifying assumption that all quantifiers except for indefinites respect such scope domains; hence we add a constraint $X^s_\nu \lhd^* X^r_\nu$ for each determiner ν which is not an indefinite. In addition, we add such a constraint whenever ν is a verb node.

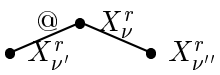
Furthermore, we associate with each NP index i that is used for coindexation in the syntax tree a variable X_i. We will use these variables to model anaphoric binding and relative clauses.

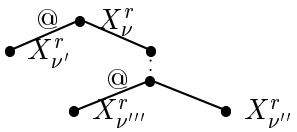
The (semantic) variables associated with syntactic nodes are related by the rules in Fig. 2, one for each syntactic rule from Fig. 1. Each construction rule is applied to syntactic nodes that have been expanded by a certain syntax rule. These rules are specified on the left hand sides of the arrows; $[_{\nu:P}\ Q\ R]$ means that node ν in the syntax tree is labelled with P, and its two daughter nodes ν' and ν'' are labelled with Q and R, respectively.

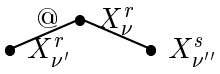
The complete constraint which the interface generates is the conjunction of all of these local constraints, plus the dominances $X^s_\nu \lhd^* X^r_\nu$

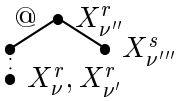
$[_{\nu:\texttt{S}}\ \texttt{NP VP}]$ (b1) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

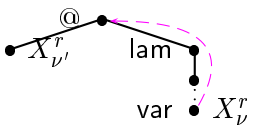
$[_{\nu:\texttt{VP}}\ \texttt{IV}]$ (b2) $\Rightarrow$ $\bullet\ X^r_\nu, X^r_{\nu'}$ $X^s_\nu = X^s_{\nu'}$

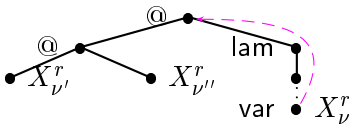
$[_{\nu:\texttt{VP}}\ \texttt{TV NP}]$ (b3) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

$[_{\nu:\texttt{VP}}\ \texttt{AV NP VP}]$ (b4) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

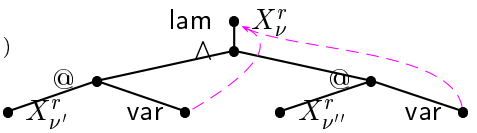
$[_{\nu:\texttt{VP}}\ \texttt{SV S}]$ (b5) $\Rightarrow$  $X^s_\nu = X^s_{\nu'}$

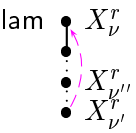
$[_{\nu:\texttt{VP}}\ \texttt{VP Conj S}]$ (b6) $\Rightarrow$  $X^s_\nu = X^s_{\nu'}$

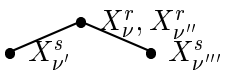
$[_{\nu:\texttt{NP}}\ \texttt{PN}]$ (b7) $\Rightarrow$  $X^s_\nu = X^s_{\nu'}$

$[_{\nu:\texttt{NP}}\ \texttt{Det}\ \overline{\texttt{N}}]$ (b8) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

$[_{\nu:\overline{\texttt{N}}}\ \texttt{N}]$ (b9) $\Rightarrow$ $\bullet\ X^r_\nu, X^r_{\nu'}$ $X^s_\nu = X^s_{\nu'}$

$[_{\nu:\overline{\texttt{N}}}\ \texttt{N RCl}]$ (b10) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

$[_{\nu:\texttt{RCl}}\ \texttt{RP S}]$ (b11) $\Rightarrow$  $X^s_\nu = X^s_{\nu'} = X^s_{\nu''}$

$[_{\nu:\texttt{S}}\ \texttt{S Coord S}]$ (b12) $\Rightarrow$ 

$[_{\nu:\alpha}\ W]$ (b13) $\Rightarrow$ $\bullet\ X^r_\nu$

where $(W, \alpha) \in Lex$

Figure 2. The syntax/semantics interface

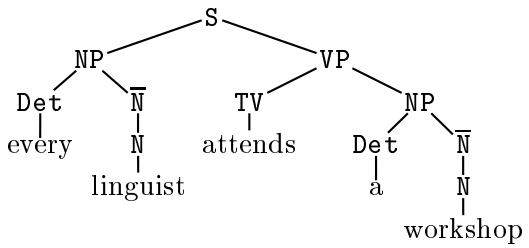
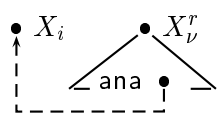


Figure 3. Syntax tree of (3).

for scope islands, plus constraints for the lexical entries. Most of the latter constraints are simple labelling constraints; for instance, the constraint we add for the intransitive verb *sleep* would be X^r_ν:sleep.

There are some exceptions to this general rule. First, the elliptic *does (too)* does not add a labelling constraint; its semantics is determined via a parallelism constraint. Furthermore, whenever coindexation signifies a relation between an anaphor ν' and its antecedent ν, we add the constraint $X^r_\nu{=}X_i$ when we process ν and the constraint $X^r_{\nu'}$:ana $\wedge$ ante$(X^r_{\nu'}){=}X_i$ when we process ν'. Similarly, the constraint for a relative pronoun with index i at ν is $X^r_\nu{=}X_i \wedge X_i$:var, and the constraint for the corresponding trace (say, at ν') is $X^r_{\nu'}{=}X_i$. This, together with rule **(b11)**, enforces correct binding of the trace.

Finally, the constraints for possessive pronouns, such as *his*, are as in (37). Intuitively, we want to analyze the NP *his$_i$ book* just like the NP *the book of him$_i$*; the diagram in (37) hides most details of this construction for readability, but notice that there is an anaphoric linking constraint from the anaphoric node to the antecedent X_i, as specified by the syntactic coindexation.

(37) 

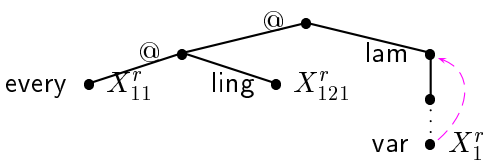
The interface as presented so far will always produce type-raised representations of all NPs. A version of the interface which produces first-order representations of proper names (i.e. as constants of type e) can be obtained by making exactly two changes to the interface. For one, the rule **(b7)** must be replaced by a rule that looks like **(b9)**; second, the lexicon must be changed to label the nodes with the constants of type e (i.e., mary instead of Mary).

5.3. An Example

To see how semantic construction works, we go through the derivation of the constraint (5) for (3). The parse tree of this sentence is shown in Fig. 5.3. We will first construct the NP descriptions, then combine them with the verb, and finally add the scope island constraints.

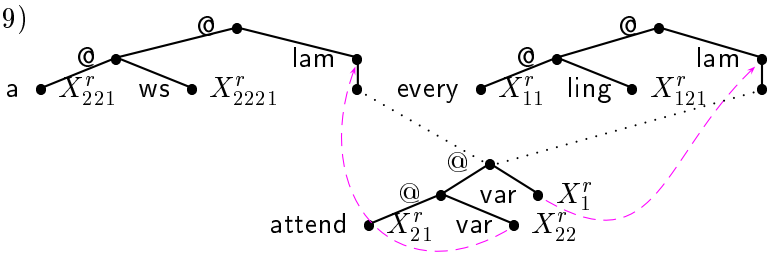
First of all, the nouns and determiners introduce several labelling constraints, e.g. X^r_{11}:every, X^r_{21}:attend, and so forth. The constraints for the NPs are built up from these by the rules **(b8)** and **(b9)**. This produces e.g. the following constraint for *every linguist*:

(38)



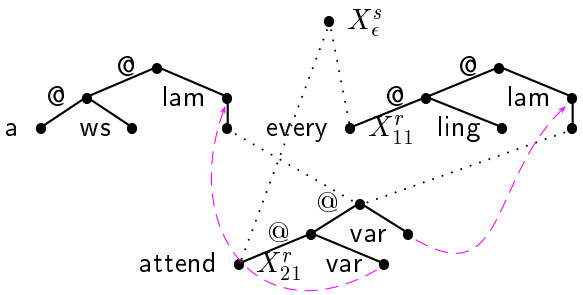
Note the equality constraint for X^r_{12} and X^r_{121} introduced by **(b9)**. An analogous construction is performed for *a workshop*. Now the rule **(b3)** combines the transitive verb and its object, basically by adding an application between X^r_{21} (the node with the verb semantics) and X^r_{22} (the root of the object NP representation). X^r_{22} is a variable node, which has already been bound by the quantifier. The rule **(b1)** analogously combines the subject and the VP, and the result looks as follows:

(39)



This is exactly the same constraint as in (5). The one thing left to do is to add the scope island constraints, which is trivial in this example because there are no scope islands. The complete sentence is associated with the variable X^s_ϵ, and all other X^s variables in the constraint are forced to be equal to X^s_ϵ by the various constraints on scope islands we have collected during the construction process. Now the node 11 in the syntax tree is a determiner which is not an indefinite, and the node 21 is a verb; so we add the constraint $X^s_{11} \lhd^* X^r_{11} \wedge X^s_{21} \lhd^* X^r_{21}$.

(40)



This constraint is of course equivalent to (39). But if the universal quantifier appeared inside an embedded sentence ("John says that every linguist attends a workshop"), the island constraint would force it not to be raised outside the embedded sentence. Because we do not impose a similar constraint on the indefinite, the specific reading of the example is available.

6. Computational Aspects

While the very motivation of underspecification is to put off the enumeration of readings for as long as possible, it should still be possible to do this efficiently when necessary. We briefly sketch the main results on processing of CLLS here and refer the reader to (Erk et al., 2001) for a more in-depth overview.

The most optimistic expectation in solving a constraint would be that each reading can be enumerated in time polynomial in the size of the constraint. Such a behaviour, however, cannot be expected of a framework that allows the analysis of ellipsis and anaphora in addition to scope. For instance, Dalrymple et al. (1991) employ higher-order unification in their analysis of ellipsis, which is known to be undecidable.

Disambiguation of arbitrary CLLS descriptions is very complex as well. Niehren and Koller (2001) showed that even CLLS without binding is equivalent to context unification (Schmidt-Schauß and Schulz, 1998; Niehren et al., 1997a), whose decidability is a prominent open problem in theoretical computer science (RTA, 1998). There are, however, semi-decision procedures which will eventually enumerate all solved forms of a constraint (Erk and Niehren, 2000).

For the sublanguage of dominance constraints (which are sufficient for scope underspecification), the situation is better: It was shown in (Koller et al., 2001) that the satisfiability problem of dominance constraints is decidable, but NP-complete. This means that the most

efficient solution procedure may take exponential time for enumerating all readings, even if the number of readings is polynomial.

An implementation of a solver for dominance constraints can be obtained by employing constraint programming with finite sets (Koller and Niehren, 2001; Duchier and Niehren, 2000). Here dominance constraints are encoded as constraints over finite sets of integers. These constraints can be solved by always performing deterministic *propagation* steps to eliminate hopeless choices before making case distinctions. This implementation performs surprisingly well; uncharacteristically for an NP-complete problem, the search never makes a wrong choice.

Indeed, Koller et al. (2000a) could show that all dominance constraints that are needed for the linguistic application belong to a fragment called *normal* dominance constraints. Satisfiability of a normal constraint can be checked with a graph algorithm of polynomial runtime; each reading can be enumerated in polynomial time as well. Accordingly, this implementation is about an order of magnitude faster than the one based on constraint programming. It is important to note, however, that while the normal fragment is sufficient for the application, the graph algorithm is not a complete solver for *all* dominance constraints.

Thus an important question for future research is to identify a fragment of parallelism constraints which contains all constraints necessary for the application, is decidable, and can be processed efficiently. An efficient solver could be obtained by extending the constraint programming solver for dominance constraints in two directions. First, the encoding into set constraints would have to be extended to parallelism constraints. Second, special-purpose algorithms for subproblems could be incorporated into the CP system. The graph algorithm for normal dominance constraints can be seen as an extreme case of this approach in which the special-purpose algorithm is so strong that it can completely replace the CP part; but less powerful algorithms could easily be combined within the CP framework.

All mentioned algorithms have been implemented and incorporated into a demo system which derives constraints from a sentence and solves them. The demo is available on the WWW (CHORUS Project, 1999).

7. Conclusion and Outlook

In this paper, we have presented the Constraint Language for Lambda Structures (CLLS), a first-order language for semantic underspecification. CLLS allows the representation of scope ambiguities, anaphora, and ellipses in simple underspecified structures that are transparent

and suitable for processing. It is a language of tree descriptions that conservatively extends the language of dominance constraints.

We have shown that CLLS correctly represents many notorious problems from the literature involving scope, anaphora, ellipses, and their interactions. An advantage over earlier approaches to this area is that CLLS analyses are completely declarative, and the formal foundations are very transparent. CLLS also provides a clean solution to a problem with variable binding in the context of scope underspecification that becomes most obvious in the presence of ellipses.

Furthermore, CLLS can be used to model reinterpretation (meaning shift) of aspect and NPs in an underspecified way (Egg, 2001; Koller et al., 2000b). In addition, one can build dynamic accessibility relations into the definition of the binding links and use this to make the effects of anaphora on quantifier scope explicit by reasoning on underspecified structures (Koller and Niehren, 2000). Nevertheless, the linguistic coverage of CLLS still needs to be extended; most obviously, a more complete theory of ellipsis is necessary.

Finally, various more formal aspects can be pursued in the future. One is to find decidable fragments and tractable algorithms for processing parallelism constraints. Another problem, which is essential for any underspecification formalism, is to lift operations on the object level to the level of underspecified descriptions. While research on lifting first-order deduction is still in its infancy (Jaspars and Koller, 1999), there are some encouraging results on underspecified beta reduction (Bodirsky et al., 2001a; Bodirsky et al., 2001b) which have to be explored further.

Acknowledgements

This work was supported by the SFB 378 (project CHORUS) at Saarland University. The article is an updated and revised version of Egg et al. (1998); we are indebted to Peter Ruhrberg and Feiyu Xu for their contributions to these papers. The authors wish to thank Manfred Pinkal, Gert Smolka, all members of the SFB 378 projects CHORUS, LISA, NEGRA, and NEP, and the commentators and participants at the Bad Teinach workshop on underspecification.

References

Alshawi, H. and R. Crouch: 1992, 'Monotonic semantic interpretation'. In: *Proceedings of the 30th ACL*. Kyoto, pp. 32–39.

Bodirsky, M., K. Erk, A. Koller, and J. Niehren: 2001a, 'Beta Reduction Constraints'. In: *Proceedings of the 12th Intl. Conference on Rewriting Techniques and Applications*. Utrecht.

Bodirsky, M., K. Erk, A. Koller, and J. Niehren: 2001b, 'Underspecified Beta Reduction'. In: *Proceedings of the 39th ACL*. Toulouse. To appear.

Bos, J.: 1996, 'Predicate Logic Unplugged'. In: *Proceedings of the 10th Amsterdam Colloquium*. pp. 133–143.

CHORUS Project: 1999, 'The CHORUS Demo System'. `http://www.coli.uni-sb.de/cl/projects/chorus/demo.html`.

Crouch, R.: 1995, 'Ellipsis and Quantification: A Substitutional Approach'. In: *Proceedings of the 7th EACL*. Dublin, pp. 229–236.

Dalrymple, M., S. Shieber, and F. Pereira: 1991, 'Ellipsis and Higher-Order Unification'. *Linguistics & Philosophy* **14**, 399–452.

Duchier, D. and J. Niehren: 2000, 'Dominance Constraints with Set Operators'. In: *Proceedings of the First International Conference on Computational Logic (CL2000)*.

Egg, M.: 2001, 'Reinterpretation from a synchronic and diachronic point of view'. *Linguistics*. To appear.

Egg, M., J. Niehren, P. Ruhrberg, and F. Xu: 1998, 'Constraints over Lambda-Structures in Semantic Underspecification'. In: *Proceedings COLING/ACL'98*. Montreal.

Erk, K., A. Koller, and J. Niehren: 2001, 'Processing Underspecified Semantic Representations in the Constraint Language for Lambda Structures'. *Journal of Language and Computation*. To appear.

Erk, K. and J. Niehren: 2000, 'Parallelism Constraints'. In: *Proceedings of the 11th Intl. Conference on Rewriting Techniques and Applications*. Norwich, U.K.

Fiengo, R. and R. May: 1994, *Indices and Identity*. Cambridge: MIT press.

Gardent, C. and B. Webber: 1998, 'Describing discourse semantics'. In: *Proceedings of the 4th TAG+ Workshop*. Philadelphia.

Gawron, M. and S. Peters: 1990, *Anaphora and Quantification in Situation Semantics*, No. 19 in CSLI Lecture Notes. CSLI/University of Chicago Press.

Hirschbühler, P.: 1982, 'VP deletion and across the board quantifier scope'. In: J. Pustejovsky and P. Sells (eds.): *NELS 12*. Univ. of Massachusetts.

Jaspars, J. and A. Koller: 1999, 'A Calculus for Direct Deduction with Dominance Constraints'. In: *Proceedings of the Twelfth Amsterdam Colloquium*. Amsterdam.

Kehler, A.: 1993, 'A Discourse Copying Algorithm for Ellipsis and Anaphora Resolution'. In: *Proceedings of EACL*.

Kehler, A.: 1995, 'Interpreting Cohesive Forms in the Context of Discourse Inference'. Ph.D. thesis, Harvard University.

Koller, A., K. Mehlhorn, and J. Niehren: 2000a, 'A Polynomial-Time Fragment of Dominance Constraints'. In: *Proceedings of the 38th ACL*. Hong Kong.

Koller, A. and J. Niehren: 2000, 'On Underspecified Processing of Dynamic Semantics'. In: *Proceedings of the 18th COLING*. Saarbrücken.

Koller, A. and J. Niehren: 2001, 'Constraint Programming in Computational Linguistics'. In: D. Barker-Plummer, D. Beaver, J. van Benthem, and P. S. di Luzio (eds.): *Logic Unleashed: Language, Diagrams, and Computation. Proceedings of the 8th CSLI Workshop on Logic, Language, and Computation*. CSLI Press. To appear.

Koller, A., J. Niehren, and K. Striegnitz: 2000b, 'Relaxing Underspecified Semantic Representations for Reinterpretation'. *Grammars* **3**(2–3). Special issue on MOL 6.

Koller, A., J. Niehren, and R. Treinen: 2001, 'Dominance Constraints: Algorithms and Complexity'. In: M. Moortgat (ed.): *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Vol. 2014 of *Lecture Note in Artificial Intelligence*. Heidelberg: Springer-Verlag.

Muskens, R.: 1995, 'Order-Independence and Underspecification'. In: J. Groenendijk (ed.): *Ellipsis, Underspecification, Events and More in Dynamic Semantics*. DYANA Deliverable R.2.2.C.

Niehren, J. and A. Koller: 2001, 'Dominance Constraints in Context Unification'. In: M. Moortgat (ed.): *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics (LACL '98)*, Vol. 2014 of *Lecture Note in Artificial Intelligence*. Heidelberg: Springer-Verlag.

Niehren, J., M. Pinkal, and P. Ruhrberg: 1997a, 'On Equality Up-to Constraints over Finite Trees, Context Unification, and One-Step Rewriting'. In: *Proceedings 14th CADE*. Townsville: Springer-Verlag.

Niehren, J., M. Pinkal, and P. Ruhrberg: 1997b, 'A Uniform Approach to Underspecification and Parallelism'. In: *Proceedings ACL'97*. Madrid, pp. 410–417.

Pinkal, M.: 1996, 'Radical Underspecification'. In: *Proceedings of the 10th Amsterdam Colloquium*. pp. 587–606.

Rambow, O., K. Vijay-Shanker, and D. Weir: 1995, 'D-Tree Grammars'. In: *Proceedings of ACL'95*.

Reyle, U.: 1993, 'Dealing with ambiguities by underspecification: construction, representation, and deduction'. *Journal of Semantics* **10**, 123–179.

RTA: 1998, 'Decidability of context unification'. The RTA list of open problems, number 90, `http://www.lri.fr/~rtaloop/`.

Sag, I. A.: 1980, *Deletion and Logical Form*. New York and London: Garland Publishing.

Schmidt-Schauß M. and K. Schulz: 1998, 'On the Exponent of Periodicity of Minimal Solutions of Context Equations'. In: *Proceedings of the 9th Intl. Conference on Rewriting Techniques and Applications*.

Shieber, S., F. Pereira, and M. Dalrymple: 1996, 'Interaction of Scope and Ellipsis'. *Linguistics & Philosophy* **19**, 527–552.

van Deemter, K. and S. Peters: 1996, *Semantic Ambiguity and Underspecification*. Stanford: CSLI.

Vijay-Shanker, K.: 1992, 'Using Descriptions of Trees in a Tree Adjoining Grammar'. *Computational Linguistics* **18**, 481–518.

Xu, F.: 1998, 'Underspecified Treatment of Verb Phrase Ellipsis'. In: I. Kruijff-Korbayová (ed.): *Proceedings of the Third ESSLLI Student Session*. Saarbrücken, Germany, pp. 269–280.