Processing Underspecified Semantic Representations in the Constraint Language for Lambda Structures

KATRIN ERK

Programming Systems Lab, Universitt des Saarlandes, Saarbrcken, Germany. E-Mail: erk@ps.uni-sb.de

ALEXANDER KOLLER

Dept. of Computational Linguistics, Universitt des Saarlandes, Saarbrcken, Germany. E-Mail: koller@coli.uni-sb.de

JOACHIM NIEHREN

Programming Systems Lab, Universitt des Saarlandes, Saarbrcken, Germany. E-Mail: niehren@ps.uni-sb.de

ABSTRACT: The constraint language for lambda structures (CLLS) is an expressive language of tree descriptions which combines dominance constraints with powerful parallelism and binding constraints. CLLS was introduced as a uniform framework for defining underspecified semantics representations of natural language sentences, covering scope, ellipsis, and anaphora. This article presents saturation-based algorithms for processing the complete language of CLLS. It also gives an overview of previous results on questions of processing and complexity.

KEYWORDS: *natural language processing, constraints, lambda-calculus, tree descriptions*

1 Introduction

The constraint language for lambda structures (CLLS) is an expressive language of tree descriptions that was proposed recently as a uniform framework for semantic underspecification [18, 17]. The models of CLLS are so-called λ-structures. They are first-order tree structures that uniquely model a λ-term modulo renaming of bound variables. CLLS lets us conjoin dominance constraints [34, 8, 51, 2] with powerful parallelism [19] and binding constraints.

The idea of semantic underspecification [50, 45] is to postpone the enumeration of meanings of a semantically ambiguous sentence. Instead, one represents the set of all meanings in a compact manner and provides an algorithm that can enumerate the individual meanings by need. CLLS combines the descriptive idea of underspecification with the classical λ-calculus approach to formal semantics [37]: a formula of CLLS serves as an underspecified semantic representation that compactly describes a set of λ-terms, each of which models an individual meaning in the traditional sense. It provides dominance and λ-binding constraints (to represent scope), parallelism constraints (to represent VP ellipsis), and anaphoric binding constraint (to represent intrasentential anaphora). These analyses integrate smoothly with an underspecified treatment of reinterpretation [28, 15].

This paper is a comprehensive investigation of the computational aspects of CLLS. Its main contribution is a presentation of saturation-based procedures for processing underspecified representations in CLLS. These procedures test the satisfiability of an arbitrary CLLS formula and can be used to enumerate the set of its *solved forms*. Our procedures always terminate for dominance and binding constraints, but not necessarily for parallelism constraints. Termination for unrestricted parallelism constraints cannot be expected, as this would solve the prominent open problem of whether context unification is decidable [41, 6, 48]. Furthermore, we review earlier results on the complexity of CLLS and of dominance constraints, its most important sublanguage.

The article complements [17], which presented the application of CLLS to natural language semantics. It draws heavily on the algorithms for dominance and parallelism constraints proposed in [13, 19] and on complexity results in [29, 41, 26]; many technical results that we can only mention here are proved in these papers. However, the algorithmic treatment of binding constraints (and hence, of CLLS as a whole) is new, as is the discussion of how VP-ellipses can be resolved by saturating parallelism constraints.

There is a large number of other formalisms that serve similar purposes as CLLS. For instance, scope underspecification is also handled in [1, 47, 38, 5, 7]; but the processing of these formalisms is typically not discussed in the literature. Dominance constraints have many applications besides scope ambiguities [34, 51, 46, 21], and computational aspects of some of these applications have been analyzed before [52, 12]. Furthermore, there are several related approaches to modeling VP-ellipses based on higher-order unification [10, 9, 20] or the more restricted context unification [40], which is second-order linear unification [44, 31, 32]. The analysis of strict-sloppy ambiguities in CLLS uses anaphoric link chains as proposed by [23].

Plan of the article

The paper is divided into two large parts. The first part (Sections 2 through 5) deals with the fragment of CLLS without parallelism, the language of dominance and binding constraints. This fragment is sufficient to represent scope ambiguities. The models of this fragments are λ-structures without parallelism (Section 2). The language itself is introduced in Section 3. Sections 4 and 5 are concerned with processing. Section 4 investigates the complexity of the fragment (satisfiability is NP-complete) and shows how to deal with a large sublanguage (*normal* dominance and binding constraints) with polynomial-time satisfiability. Section 5 uses a different approach to process the whole fragment, an approach that is re-used and extended in the second part of the paper.

The second part (Sections 6 through 10) deals with CLLS in general, i.e with parallelism. Section 6 introduces the idea of parallelism and gives a definition. Section 7 presents the core of a procedure for CLLS, but yet without lambda and anaphoric binding. Section 8 gives an axiomatic semantics of parallelism constraints which captures the interaction of parallelism with lambda and anaphoric binding; these axioms are turned into a saturation procedure for CLLS in a second step. This procedure reuses the saturation rules for dominance and binding constraints specified in the first part. The expressiveness of parallelism constraints is discussed in Section 10 by relating it to context unification.

We give a brief overview of implementations in Section 11. Finally, we conclude and collect all saturation rules for CLLS in the appendix.

2 Lambda structures

The models of formulas in the constraint language for λ-structures (CLLS) are λ-structures [14]. These are first-order tree structures representing λ-terms uniquely modulo renaming of bound variables. In this section, we recall the definition of λ-structures. We leave out parallelism, which will be added in Section 6. We start with trees and tree structures and then turn to λ-terms and λ-structures.

In defining λ-structures, we have a choice of how to represent trees: we can see them as labeled graphs, or as ground terms, or as sets of node addresses plus labeling functions. In the literature, the third alternative has often been adopted [14, 29]; here we take the graph view. However, this distinction is purely cosmetic, as all three concepts are only tools to induce the logical *tree structures* we are really interested in (Def. 2.2), so there is no danger in choosing the most convenient perspective.

2.1 Trees and tree structures

We let f, g range over the function symbols of a signature Σ, each function symbol f being equipped with an arity $\mathsf{ar}(f) \geq 0$. We write a, b for constants.

We first define (finite) constructor trees built from function symbols in Σ. Constructor trees are ground terms over Σ, such as $f(g(a,b))$. Equivalently, we can view them as labeled directed graphs in the usual way; for instance, the graph corresponding to $f(g(a,b))$ is shown in Fig. 1.

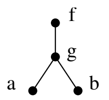


FIG. 1. $f(g(a,b))$

We define an (unlabeled) *tree* to be a finite directed graph (V, E). The set V is a finite set of *nodes* ranged over by u, v, w, and the set $E \subseteq V \times V$ is a finite set of *edges*. The in-degree of each node is at most 1; each tree has exactly one *root*, i.e. a node with in-degree 0. We call the nodes with out-degree 0 the *leaves* of the tree.

DEFINITION 2.1
A (finite) *constructor tree* over Σ is a triple (V, E, L) where (V, E) is a tree, $L : V \to \Sigma$ a *node labeling* and $L : E \to \mathbb{N}$ an *edge labeling*, such that for each node $u \in V$ and each $1 \leq k \leq \mathsf{ar}(L(u))$, there is exactly one edge $(u, v) \in E$ with $L(u, v) = k$.

The symbol L is overloaded to serve both as a node and an edge labeling; there is no danger of confusion. We draw constructor trees as in Fig. 1, by annotating nodes with their labels and ordering the edges along their labels from left to right.

DEFINITION 2.2
The *tree structure* of a constructor tree (V, E, L) is a first-order structure with domain V. It provides the *dominance relation* $\lhd^* \subseteq V \times V$ and a *labeling relation* for each function symbol $f \in \Sigma$, which is defined as follows (for all $u, v, v_1, \ldots v_n \in V$):

$u \lhd^* v$	iff	there is a path from u to v in (V, E);
$u{:}f(v_1, \ldots, v_n)$	iff	$L(u) = f, \mathsf{ar}(f) = n,$ and $L(u, v_i) = i$ for all $1 \leq i \leq n$.

Below, we freely identify a constructor tree with its tree structure. In passing, note that the dominance relation of a tree structure is fully determined by its labeling relations: if we consider an *immediate dominance* relation $\lhd$ such that $u \lhd v$ holds in a tree structure iff some labeling $u{:}f(\ldots, v, \ldots)$ is valid in it, then dominance $\lhd^*$ becomes the reflexive and transitive closure of immediate dominance $\lhd$.

2.2 Lambda terms and lambda structures

We want to represent not only ground terms but also λ-terms by tree structures. In order to do so, we assume from now on that our signature Σ contains constants (i.e. nullary function symbols) for the words of natural language, e.g. mary, john, run, drive, car. In addition, we use the symbol lam of arity 1 for *lambda abstraction*, the symbol @ arity 2 for *application*, the constant var for representing occurrences of bound variables in a λ-term, and the constant ana for representing anaphoric reference. Note that we only consider intrasentential anaphora here; for an integration of CLLS with dynamic semantics, see [27].

A λ-term can be represented by a λ-structure much in the same way as a ground term can be represented by a tree structure. For instance, consider the λ-term in (2.2), which incidentally represents the meaning of sentence (2.1).

(2.1) Mary$_1$ drives her$_1$ car

(2.2) $(\mathsf{car\text{-}of}\ \mathsf{ana}_1)\ (\lambda x\ ((\mathsf{drive}\ x)\ \mathsf{mary}_1))$

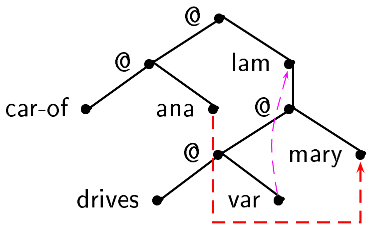


FIG. 2. Mary$_1$ drives her$_1$ car.

The λ-structure for (2.2) is shown in Fig. 2. Its internal nodes represent applications and abstractions. Variable binding is represented by an explicit *λ-binding function*, drawn using curved arrows, which maps bound-variable nodes to binder nodes. Similarly, we represent anaphora by mapping anaphoric nodes to antecedent nodes with an *anaphoric linking function*, drawn as angled arrows.

DEFINITION 2.3
A (total) *λ-structure* is a quintuple $(V, E, L, \lambda, \mathsf{ante})$ consisting of a tree structure (V, E, L) over Σ, a total function $\lambda : L^{-1}(\mathsf{var}) \to L^{-1}(\mathsf{lam})$ mapping variable nodes to their lambda binders, and a total function $\mathsf{ante} : L^{-1}(\mathsf{ana}) \to V$ relating anaphoric nodes to their antecedents. Additionally, each var-node must be situated below its λ-binder, i.e. the function λ must have the property that $\lambda(v) \lhd^* v$ is satisfied for all nodes v in the domain of λ.

It will sometimes be useful (e.g. in the proofs of Theorem 4.3 and Proposition 5.1) to also have *partial lambda structures*, where the functions λ and ante are partial instead of total. But this particular choice is irrelevant with respect to the question of constraint satisfiability that we will investigate.

3 Dominance and binding constraints in CLLS

We now introduce *dominance and binding constraints*, or DB constraints for short. This is the fragment of CLLS which leaves out parallelism constraints. DB constraints are tree descriptions whose models are λ-structures. DB constraints can be drawn perspicuously as *constraint graphs*. We also briefly discuss the applications of this fragment to scope underspecification.

3.1 The constraint language

Let X, Y, Z range over an infinite set of variables Vars for nodes in a tree structure. A *dominance and binding constraint* φ (*DB constraint* for short) is a conjunction of *atomic constraints* that we also call *literals*. There are literals for dominance, labeling, inequality, lambda and anaphoric binding:

$$\varphi ::= X \triangleleft^* Y \mid X{:}f(X_1, \ldots, X_n) \mid X{\neq}Y \mid \lambda(X){=}Y \mid \mathsf{ante}(X){=}Y \mid \varphi \wedge \varphi'$$

Dominance constraints, as studied in [41, 29, 13, 26], are the sublanguage of DB constraints that does not contain binding literals.

Constraints φ of CLLS are interpreted in the class of λ-structures in the classical Tarskian way. Note that the relation symbols in our constraints are the same as the matching λ-structure relations. There should be no danger of confusion, as relation symbols are always applied to node variables, whereas relations can only be applied to the nodes of a λ-structure.

The constraints of CLLS do not support other first-order connectives beside conjunction. Richer descriptions would require more powerful algorithms for dealing with the additional computational complexity (see Section 4.1). Even simple propositional connectives – in particular negation – would impose an additional computational burden. This might be less critical without parallelism [11] but becomes awkward otherwise.

We will nevertheless make use of first-order formulas Φ over CLLS constraints later on. Their function will be to facilitate reasoning about algorithms dealing with underspecified semantic representations in CLLS; they will not serve as underspecified representations themselves.

3.2 Solutions of a constraint

The set of (free) node variables of a first-order formula Φ over CLLS-constraints is denoted by $\mathsf{Vars}(\Phi)$. A *variable assignment* into a λ-structure $(V, E, L, \lambda, \mathsf{ante})$ is a partial function $\alpha : \mathsf{Vars} \rightsquigarrow V$. We write $\mathrm{Dom}(\alpha)$ for the domain of α. A *solution* of a formula Φ consists of a λ-structure τ and a variable assignment α into τ with $\mathsf{Vars}(\Phi) \subseteq \mathrm{Dom}(\alpha)$ under which Φ evaluates to true. In this case, we call τ a *model* of Φ. We also say that (τ, α) *satisfies* Φ and write $\tau, \alpha \models \Phi$ if (τ, α) is a solution of Φ. We write $\Phi \models \Phi'$ and say that Φ *entails* Φ' if every solution of Φ interpreting all variables in $\mathsf{Vars}(\Phi')$ is a solution of Φ'.

The constraint on the left of Fig. 3 has an obvious solution: the λ-structure depicted next to it with the assignment α_1. However, this solution is not the only one; there are satisfying λ-structures that are much larger. For example, the rightmost λ-structure

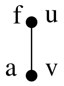
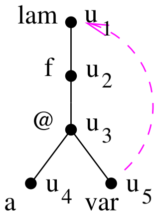
$X{:}f(Y)\wedge$ $Y\triangleleft^* Z\wedge$ $Z{:}a$	 $\alpha_1(X)=u$ $\alpha_1(Y)=v$ $\alpha_1(Z)=v$	 $\alpha_2(X)=u_2$ $\alpha_2(Y)=u_3$ $\alpha_2(Z)=u_4$

FIG. 3. A constraint and two of its solutions

in Fig. 3 together with the assignment α_2 is another solution of the constraint. This λ-structure contains nodes that are not referred to by any variable of φ. Note that CLLS differs from most other underspecification formalisms [1, 47, 5] in this respect. The existence of these larger models is essential for an underspecified approach to reinterpretation in CLLS [28, 16, 15]. Also, our treatment of parallelism relies on models where some nodes are not referred to in the constraint.

3.3 Constraint graphs

We often draw CLLS constraints as graphs (they are much easier to read that way). The nodes of these graphs stand for the variables of a constraint, and the edges represent literals. By way of example, an (unsatisfiable) constraint and its constraint graph are shown in Fig. 4. We represent the labeling constraint $X{:}f(X_1, X_2)$ by drawing the

$X{:}f(X_1, X_2)\wedge$
$X_1\triangleleft^* Y \wedge X_2\triangleleft^* Y$

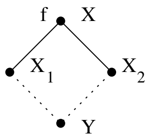


FIG. 4. An unsatisfiable constraint and its graph

node label f next to the variable X and by connecting X to its children X_1, X_2, which are ordered from left to right, by solid lines. The dominance constraints $X_1\triangleleft^* Y$ and $X_2\triangleleft^* Y$ are drawn as dotted lines, with X_1, X_2 situated above Y. The complete constraint is unsatisfiable because trees do not branch upwards.

Binding constraints can also be represented naturally in constraint graphs: we draw literals for λ-binding as curved and for anaphoric binding as angled arrows. An example is given in Fig. 5.

Constraint graphs look very similar to our pictures of λ-structures. While this suggestive similarity is intended, it is important to keep the two apart. The nodes of a constraint graph are variables; constraint graphs are simply an alternative syntax for DB constraints. These variables *denote* the nodes in a λ-structure; λ-structures are

$$\begin{array}{l} X_0{:}@(X_1,X_4)\wedge \\ X_1{:}@(X_2,X_3)\wedge X_2{:}\mathsf{car\_of}\wedge \\ X_3{:}\mathsf{ana}\wedge \mathsf{ante}(X_3){=}Y_4\wedge \\ X_4{:}\mathsf{lam}(X_5)\wedge X_5{\lhd^*}Y_0\wedge \\ Y_0{:}@(Y_1,Y_4)\wedge Y_4{:}\mathsf{mary} \\ Y_1{:}@(Y_2,Y_3)\wedge Y_2{:}\mathsf{drives} \\ Y_3{:}\mathsf{var}\wedge \lambda(Y_3){=}X_4 \end{array}$$

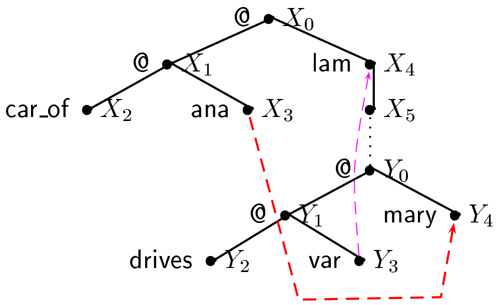


FIG. 5. Drawing binding constraints

semantic entities for CLLS.

One important concept related to constraint graphs is that of a *(tree) fragment* – a tree-shaped subgraph whose nodes are connected by solid lines. Fragments have roots and leaves; unlabeled leaves are called *holes*. For instance, the constraint graph in Fig. 5 consists of two fragments, the upper fragment with variables $X_0, \ldots, X_6$ and the lower fragment with variables $Y_0, \ldots, Y_4$.

The one type of literal that is not easily representable in a constraint graph is inequality $X{\neq}Y$. One way of representing them would be to annotate graphs with explicit inequality literals. Here we choose another alternative: we leave inequalities implicit if they just prevent *overlap* of fragments; that is, labeled variables in two different fragments should never be mapped to the same node. This is made precise in the following definition.

DEFINITION 3.1
We call a constraint φ *overlap-free* if for each pair $X{:}f(\ldots), Y{:}g(\ldots)$ of distinct labeling literals in φ (where f, g need not be different), $X{\neq}Y$ belongs to φ.

Drawing convention. For easier readability, we keep with some simplifying conventions in drawing constraint graphs. First of all, we omit the name of the variable represented by a graph node wherever convenient. Second, we leave out inequalities as mentioned above. In order to fix the set of implicit inequalities, we assume from now on that all constraint graphs represent overlap-free constraints. Finally, we omit some literals that are entailed by the represented constraints. For instance, we never draw edges for literals $X{\lhd^*}X$; also, if a constraint graph contains dominance edges for $X{\lhd^*}Y$ and $Y{\lhd^*}Z$ then we may freely suppress the dominance edge for the (entailed) literal $X{\lhd^*}Z$.

3.4 Scope underspecification

Now we take a look at some examples from scope underspecification that illustrate how the formalism relates to natural language semantics. First, consider the sentence (3.1). It contains a prototypical scope ambiguity with two readings.

(3.1) Every plan has a catch.

It can either mean that there is one specific drawback that all plans suffer from; we get this meaning if we continue (3.1) by . . . *namely the big watchdog in the prison yard*. Or it can mean that each plan is flawed in a different way, e.g. plan A fails because we do not possess the key to the prison door, and plan B will not work because we are too lazy to dig our way out. The two readings only differ in the order of the two quantifiers: the first reading is represented by the λ-term in (3.2) and the second by the λ-term in (3.3).

(3.2) $(\mathsf{a\ catch})(\lambda x\ (\mathsf{every\ plan})(\lambda y\ (\mathsf{have}\ x)\ y))$

(3.3) $(\mathsf{every\ plan})(\lambda x\ (\mathsf{a\ catch})(\lambda y\ (\mathsf{have}\ x)\ y))$

Accordingly, the λ-structures for the two readings only differ in one point: in the λ-structure for (3.2), the tree fragment for *a catch* dominates the fragment for *every plan*, and in the λ-structure for (3.3), it is the other way round.

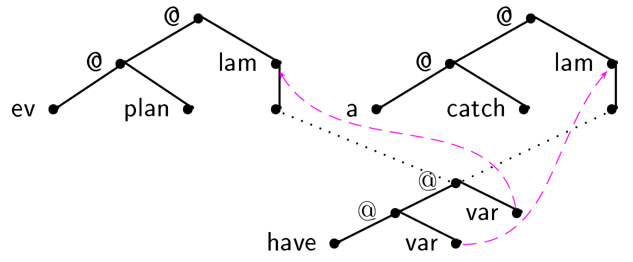


FIG. 6. Constraint for *Every plan has a catch.*

We can represent both λ-structures by one constraint, which models the underspecified semantics of (3.1); it is the constraint in Fig. 6. This constraint does not specify whether the fragment for *a catch* dominates that for *every plan* or vice versa. But it does state that one of these two cases must hold: the two fragments both dominate the fragment for *have*. But constraints describe trees, and trees do not branch upwards. Note, by the way, that each quantifier of the sentence is modeled by a separate fragment in the constraint.

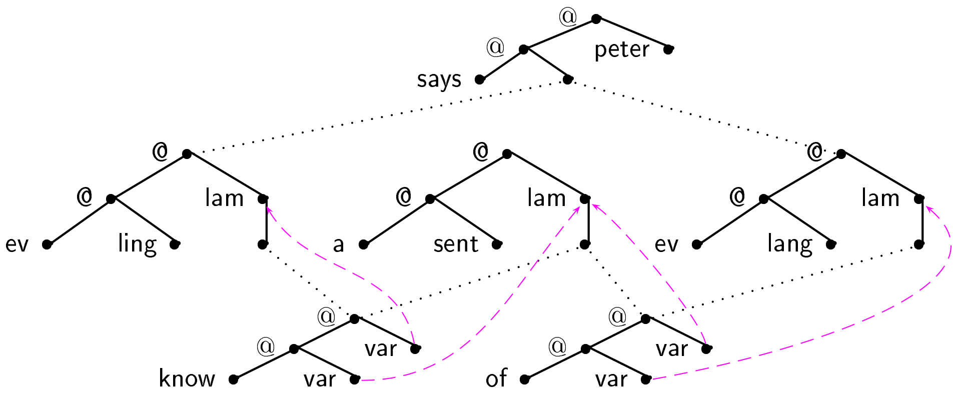


FIG. 7. Constraint for *Peter says every linguist knows a sentence of every language.*

In general, constraints representing the underspecified semantics of a sentence can be somewhat more intricate than the simple one in Fig. 6, though. For example, Fig. 7 models the meaning of the sentence

(3.4) Peter says every linguist knows a sentence of every language.

Here, the fragments for *every linguist*, *know*, *a sentence*, *of*, *every language* form a *chain*, an alternating sequence of "upper" and "lower" fragments. See [28] for a treatment of chains as well as an analysis of the general shape of constraints that represent the meaning of sentences.

4 Normal Dominance and Binding Constraints

Now we are ready to begin discussing processing issues. There are two questions we are interested in: first, to test the satisfiability of a constraint, and second, to enumerate its solutions. In this section, we recall the result that the satisfiability problem of DB constraints in general is NP-complete [29]. Then we define *normal* DB constraints, a fragment with polynomial satisfiability [26]. Finally, we show how to enumerate the solutions of a normal constraint by enumerating its solved forms in the sense defined below (Def. 4.4).

4.1 Complexity of DB constraints

Although dominance constraints are a very simple language, their satisfiability problem is surprisingly hard [29]:

THEOREM 4.1
Satisfiability of dominance constraints is NP-complete.

NP-hardness is shown by encoding the Boolean Satisfiability problem. The main idea of the encoding is to force fragments to overlap, thereby expressing disjunction. An example of how to do this is shown in Fig. 8; deviating from our drawing convention for constraint graphs, this graph is intended to represent a constraint that is *not* overlap-free (and does not contain any inequality literals). This constraint entails $X{=}Y \vee X{=}Y_1$; that is, X must overlap either with Y or with Y_1.

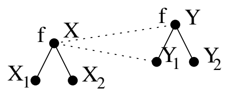


FIG. 8. Overlap

Satisfiability of first-order formulas over dominance constraints is strictly more complex than of dominance constraints, but still decidable. This problem is non-elementary, i.e. it is not in d-times exponential time for any d [29].

4.2 *Normal constraints*

When restricted to overlap-free constraints, the proof of Theorem 4.1 breaks down; and indeed, if we require a slightly stronger restriction than overlap-freeness, we get polynomial satisfiability.

To describe it, we need a notion of *reachability* within a constraint, which we define inductively. Given a constraint φ and variables $X, Y \in \mathsf{Vars}(\varphi)$, *$Y$ is reachable from X in φ* iff

- $X{:}f(\ldots Y \ldots)$ is in φ for some f, or
- $X \triangleleft^* Y$ is in φ, or
- there exists a Z such that Z is reachable from X, and Y is reachable from Z.

DEFINITION 4.2
A DB constraint φ is called *normal* iff for all variables $X, Y, Z \in \mathsf{Vars}(\varphi)$,

1. φ is overlap-free;
2. labeling constraints form tree-like fragments: each variable occurs at most once in a mother and once in a child position of a labeling literal in φ;
3. dominance edges only go from holes to roots of fragments: if $X \triangleleft^* Y$ is in φ, then for all Z, f, neither $X{:}f(\ldots)$ nor $Z{:}f(\ldots Y \ldots)$ is in φ;
4. there are no empty fragments: if $X \triangleleft^* Y$ in φ, then there are Z and f such that $Z{:}f(\ldots X \ldots)$ in φ;
5. no node variable caries two binding requirements in φ: for all variables X there exists at most one lambda binding literal $\lambda(X) = Y$ and most one anaphoric binding literal $\mathsf{ante}(X) = Y$ in φ.
6. lambda binders can always be satisfied: if $\lambda(X) = Y$ in φ then $X{:}\mathsf{var} \wedge Y{:}\mathsf{lam}(Y')$ in φ for some Y', and X is reachable from Y in φ.
7. anaphoric binders can always be satisfied: if $\mathsf{ante}(X) = Y$ in φ then $X{:}\mathsf{ana}$ in φ.

Basically, a normal DB constraint is a graph of tree-shaped fragments, connected by dominance edges. The constraints in Fig. 6 and Fig. 7 are both normal, and indeed, all constraints needed to model scope (not parallelism) fall into this class.

THEOREM 4.3
Satisfiability of normal DB constraints is in deterministic polynomial time.

PROOF. For pure dominance constraints satisfying properties 1–4, this theorem is proved in [26]. The idea behind the non-trivial proof is to check satisfiability by testing for the existence of *hypernomal cycles* in the constraint graph. This cycle test can then be reduced to a weighted matching problem.

Now let φ be a normal DB constraint such that its dominance part φ' is satisfiable. Let (τ', α) be a solution of φ' with $\tau' = (V, E, L, \dots)$. We define partial functions $\lambda, \mathsf{ante} : V \rightsquigarrow V$ by $\lambda(\alpha(X)) = \alpha(Y)$ iff $\lambda(X){=}Y$ in φ and $\mathsf{ante}(\alpha(X)) = \alpha(Y)$ iff $\mathsf{ante}(X){=}Y$ in φ. We have to show that both functions are well-defined. For λ-binding, this means that for each node $v \in V$, if φ contains λ-binding literals $\lambda(X){=}Y$ and $\lambda(X'){=}Y'$ with $\alpha(X) = \alpha(X') = v$, then $\alpha(Y)$ must be equal to $\alpha(Y')$. So suppose we have two such λ-binding literals. By condition 6, X and X' must be var-labeled in φ. But then the overlap-freeness of φ (condition 1) implies that X and X' must be the same variable. (Otherwise, $X{\neq}X'$ in φ such that $\alpha(X) \neq \alpha(X')$, which contradicts our assumption.) Now by condition 5, there is at most one lambda binding requirement per variable. So the literals $\lambda(X){=}Y$ and $\lambda(X'){=}Y'$ must thus be the same. The well-definedness of ante can be shown the same way.

We now check that $\tau = (V, E, L, \lambda, \mathsf{ante})$ is a partial lambda structure. It then clearly follows that (τ, α) satisfies φ except that τ is partial. But a constraint that can be satisfied by a partial lambda structure can also be satisfied by a total one. It is sufficient to add a lam-labeled node at the root of the partial lambda structure, whose purpose is to bind all previously unbound nodes labeled by var and ana.

We now check all conditions of Def. 2.3 for τ, except for totality of λ and ante. The domain of λ is a subset of $L^{-1}(\mathsf{var})$, and its range is a subset of $L^{-1}(\mathsf{lam})$, by condition 6. The domain of ante is a subset of $L^{-1}(\mathsf{ana})$ by condition 7. Finally, each var-labeled node is situated below its binder by the reachability demanded in condition 6. It is easy to show by structural induction (using the definition of reachability) that whenever Y is reachable from X in φ, $\alpha(X)$ dominates $\alpha(Y)$ in τ' and in τ. ∎

4.3 Enumeration of solved forms

Now that we know how to check whether a normal constraint has a solution, we would like to *enumerate* all solutions. Unfortunately, this is impossible, as every satisfiable DB constraint has an infinite number of solutions: There may be any number of additional nodes between the ends of a dominance edge, and there may be additional nodes above the "root" of the constraint graph as well.

This is why we will consider a slightly revised enumeration problem: that of enumerating all (finitely many) solved forms of the normal constraint. Intuitively, a constraint is in solved form if it is easy to read off its solutions. Solved forms may have more than one solution, but the differences between these solutions should be "irrelevant".

DEFINITION 4.4
We say that a normal constraint is in *solved form* if its constraint graph is "tree shaped", i.e. it is acyclic and does not contain any nodes with two dominance parents (called "triangles" below).

For example, the two right-hand constraints schematically drawn in Fig. 9 and, more concretely, the constraint in Fig. 5 are in solved form; the leftmost constraint in Fig. 9 and the constraint in Fig. 6 are not. A satisfying λ-structure for a constraint in solved form can be obtained by identifying the end nodes of any remaining dominance edge; but as we have seen in Section 3.2, these dominance literals can also be satisfied with an arbitrary number of nodes between the ends. Solved forms abstract away from this effect, which is not interesting for the application to scope.

A different formalization of solved forms (with respect to a saturation system) will be defined in Section 5.3. Although it is completely different on the surface, the underlying intuition of capturing the relevant aspects of a solution will be the same.

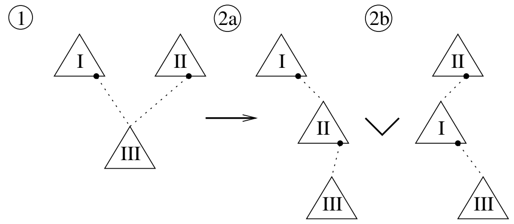


FIG. 9. Solving "triangles"

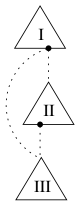


FIG. 10: Redundant edge

Given that a constraint is satisfiable (in particular, has a cycle-free graph), we can eliminate all triangles – and hence, obtain solved forms – by successive applications of the *triangle rule* illustrated in Fig. 9. This rule requires a *choice*: Either the hole of fragment I is moved above the root of fragment II, or the hole of fragment II is moved above the root of fragment I.

After each choice, we can apply the above satisfiability test to check if the branch is worth pursuing. Thus we eventually arrive at solved forms, and we only need polynomial time for each solved form, as the search tree spanned by the choices has polynomial depth. All the solved forms we find are different, and each solution of the original constraint satisfies one of them.

A final complication is that we must take care not to apply the triangle rule to *redundant* dominance edges. If we applied the choice rule to a trivial triangle as in Fig. 10, we would not make any progress. Fortunately, constraint graphs can be made irredundant very cheaply, so we can add this as a preprocessing step before each application of the choice rule.

5 A solver for CLLS without parallelism

We now present an algorithm that tests the satisfiability of *unrestricted* DB constraints in non-deterministic polynomial time. The same algorithm can also be used to enumerate the solved forms of a DB constraint. While it performs worse on normal constraints than the graph-based algorithm from Section 4.2, it has the advantage that it is more general and can be extended to a satisfiability test for CLLS as a whole, which we will do in the second part of the paper.

Our algorithm saturates a given constraint with respect to a set of propagation and distribution rules. The idea is to delay case distinctions (distribution) for as long as deterministic inferences (propagation) can make progress. This setup is known as the computational paradigm of constraint programming [35]. In practice, it often helps avoid the worst case complexity when solving combinatoric problems.

We proceed as follows. First, we extend CLLS by set operators adding a restricted form of negation and disjunction in Sections 5.1 and 5.2. After discussing saturation in general (Section 5.3), we present a saturation algorithm for dominance constraints with set operators in Section 5.4. Finally, in Section 5.5, we extend this saturation algorithm to also take binding constraints into account.

5.1 Set operators

For the sake of generality and to permit more powerful propagation, we formulate our saturation rules for an extension of CLLS which allows for set operators in dominance constraints. This language subsumes several variants of dominance constraints in the literature [29, 12], whose special-purpose literals are simply special cases of the set-operator literals. Dominance constraints with set operators were first proposed by Cornell [8], who however did not consider labeling constraints. Recently, they were rediscovered independently and investigated without further restrictions [13].

In the fragment of CLLS considered so far, we can talk about the following relations beside labeling: dominance $\triangleleft^*$, inverse dominance $\triangleright^*$, equality $=$, inequality $\neq$, proper dominance $\triangleleft^+$, and inverse proper dominance $\triangleright^+$.

In our extension, we are interested in all relations that can be generated from the dominance relation and *set operators*: union $\cup$, intersection $\cap$, complementation $\neg$, and inversion $^{-1}$. We can generate non-dominance $\neg\triangleleft^*$ and inverse non-dominance $\neg\triangleright^*$ but also the important *disjointness* relation $\perp$, which holds whenever neither dominance nor inverse dominance holds: $\perp$ means $\neg\triangleleft^* \cap \neg\triangleright^*$. Between two arbitrary nodes of a tree structure, exactly one of the relations $\triangleleft^+, \triangleright^+, =, \perp$ holds. We get the following partition:

$$V \times V \quad = \quad (\triangleleft^+ \cup \triangleright^+ \cup = \cup \perp)$$

As a consequence, there are exactly 16 relations that set operators can generate from dominance constraints; each of these relations is a finite union $\cup R$ where R is a subset of $\{\triangleleft^+, \triangleright^+, =, \perp\}$.

5.2 *Constraint language with set operators*

A *dominance and binding constraint with set operators* (for short, *DB constraint with set operators*) φ has the following abstract syntax, where R is a subset of relation symbols in $\{=, \triangleleft^+, \triangleright^+, \perp\}$.

$$\varphi \quad ::= \quad X\,R\,Y \mid X{:}f(X_1\ \ldots\ X_n) \mid \lambda(X){=}Y \mid \mathsf{ante}(X) = Y \mid \varphi \wedge \varphi' \mid \mathsf{false}$$

The interpretation of a set R in a tree structure is the union of the interpretations of the relations symbols in R. For instance, a constraint $X\ \{=, \perp\}\ Y$ states that the values of X and Y are either equal or disjoint. For convenience, we admit false as a constraint. To accommodate explicit set operators, we allow for syntactic sugar, writing constraints of the form XSY where S is a set expression:

$$S \quad ::= \quad \triangleleft^* \mid \triangleright^* \mid = \mid \neq \mid \triangleleft^+ \mid \triangleright^+ \mid \perp \mid \neg S \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S^{-1}$$

Set operators permit us to express a controlled form of negation and disjunction. For instance, the constraint $X \neg SY$ is equivalent to the negated formula $\neg\ X\,SY$, whereas $X\ S_1 \cup S_2\ Y$ is equivalent to the disjunction $X\ S_1\ Y \vee X\ S_2\ Y$. However, there is no way to express $X \triangleleft^* Y \vee X \triangleleft^* Z$ using set operators.

5.3 *Saturation and Solved Forms*

The basic idea of saturation is to interpret a rule system as an accumulation procedure. Starting with a set of literals (or other items), more and more literals are added to the set according to some saturation rules. Slightly abusing notation, we freely identify a constraint with the set of its literals. This way, subset inclusion defines a partial order $\subseteq$ on constraints.

The *saturation rules* we work with are implications of the following form:

$$\varphi_0 \rightarrow \vee_{i=1}^{n} \exists V_i \varphi_i$$

where $n \geq 1$ and $\mathsf{Vars}(\varphi_i) - \mathsf{Vars}(\varphi_0) \subseteq V_i$ for all $1 \leq i \leq n$. A rule is called a *propagation rule* if $n = 1$ and a *distribution rule* otherwise. A rule $\varphi \rightarrow \Phi$ is *sound* if $\varphi \models \Phi$. The critical rules with respect to termination are those with local variables on their right hand side, that is, those where at least one of the sets V_i is non-empty.

Given a set V of variables and a constraint φ, we call a constraint $\theta\varphi$ a *V-variant of* φ if $\theta : V \to \mathsf{Vars}$ is some substitution of the variables in V. We call this variant *fresh* if $\theta(V)$ is disjoint from $\mathsf{Vars}(\varphi)$.

A *saturation algorithm* repeatedly applies saturation rules to an input constraint, which is extended by each rule application. A rule $\varphi_0 \to \vee_{i=1}^{n} \exists V_i \varphi_i$ is applied to a constraint φ by selecting a disjunct $\exists V_i \varphi_i$ on the right hand side, selecting a fresh V_i-variant of φ_i say φ_i', and returning $\varphi \wedge \varphi_i'$. A saturation step can be applied only if the following two conditions are satisfied:

1. inferences must be valid: the left-hand side φ_0 is contained in φ, i.e. $\varphi_0 \subseteq \varphi$.
2. inference must add new information: no variant of any disjunct on the right hand side belongs to φ, i.e. for all $1 \leq i \leq n$ and for all V_i-variants φ_i' of φ_i: $\varphi_i' \not\subseteq \varphi_0$.

Let *S* be a set of saturation rules. We call a constraint *saturated* (under *S*) if no further rule of *S* applies to it. We say that a constraint is in *S-solved form* if it is saturated under *S* and clash-free (i.e. it does not contain **false**). If the set *S* is clear from the context, we also write *solved form* instead of *S*-solved form. Finally, a constraint φ' is *a solved form of* the constraint φ iff it is in solved form, and $\varphi \subseteq \varphi'$ in the above sense.

This is the second definition of a solved form after that in Section 4.3. We will continue to use this second definition for the rest of the paper, as it works more generally than the first one. Although this second definition looks very different from the first one, it will prove to be quite similar for the concrete saturation rules that we present next: these saturation rules will also generate "tree-shaped" solved forms in a way.

5.4 *Saturation of dominance constraints*

We now present a saturation algorithm for dominance constraints, which we extend in Section 5.5 such that it also deals with binding constraints. This algorithm consists of a set *D* of saturation rules which decides the satisfiability of a dominance constraint and also enumerates its *D*-solved forms.

We proceed in three steps. First we present a subset of saturation rules from *D* that, given a satisfiable, normal dominance constraint, can enumerate possible solved forms, but does not check for satisfiability (Fig. 12). In the second step, we add rules that detect unsatisfiability (Fig. 15). Finally, we introduce rules that can deal with unrestricted dominance constraints (Fig. 17).

5.4.1 Enumerating solved forms

The core rules of algorithm *D* are shown in Fig. 12. These rules can enumerate the solved forms of a satisfiable normal constraint. Consider for instance the leftmost constraint in Fig. 11. Its graph has the same shape as the underspecified representation of the simple scope ambiguity in Fig. 6.

$X_1{\neq}X_2 \wedge$
$X_1{:}f(X_2) \wedge X_2{\lhd^*}Z \wedge$
$Y_1{:}g(Y_2) \wedge Y_2{\lhd^*}Z \wedge$
$Z{:}a$

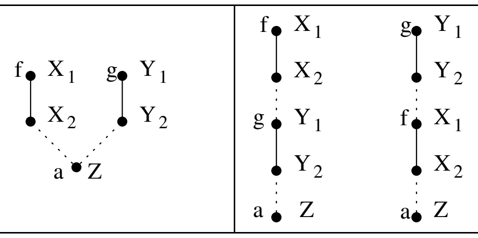


FIG. 11. The shape of a simple scope ambiguity

This constraint contains 2 upper fragments with variables X_1, X_2 and Y_1, Y_2, respectively, and a lower fragment, consisting of the variable Z. The constraint should have the two solved forms drawn to the right in Fig. 11.

(D.NegDisj)	$X{\lhd^*}Z \wedge Y{\lhd^*}Z \rightarrow X{\neg\bot}Y$	
(D.Distr.NegDisj)	$X{\neg\bot}Y \rightarrow X{\lhd^*}Y \vee Y{\lhd^*}X$	
(D.Inter)	$X R_1 Y \wedge X R_2 Y \rightarrow X R Y$	if $R_1 \cap R_2 \subseteq R$
(D.Child.Ineq)	$X{\neq}Y \wedge X{:}f(\dots, X', \dots) \wedge Y{:}g(\dots, Y', \dots) \rightarrow X'{\neq}Y'$	
(D.Parent.Ineq)	$X{\lhd^+}Z \wedge Y{:}f(\dots, Z, \dots) \rightarrow X{\lhd^*}Y$	

FIG. 12. Main rules

This is indeed the case, as we can see by going through the necessary rule applications. X_2 and Y_2 dominate a common variable Z; so they cannot be in disjoint positions. This inference is performed by the rule (D.NegDisj), which adds the literal $X_2{\neg\bot}Y_2$. Non-disjointness implies dominance or inverse dominance. This case distinction is made by saturation with (D.Distr.NegDisj). Applied to the example constraint, it can add either $X_2{\lhd^*}Y_2$ or $Y_2{\lhd^*}X_2$. We only consider the case of $X_2{\lhd^*}Y_2$, the other case is symmetric.

What remains to do is to infer $X_2{\lhd^*}Y_1$. We use (D.Child.Ineq), which states that children of distinct nodes are distinct, and the fact that $X_1{\neq}Y_1$, to infer $X_2{\neq}Y_2$. We now use rule (D.Inter) to intersect $X_2{\neq}Y_2$ with $X_2{\lhd^*}Y_2$, which gives us $X_2{\lhd^+}Y_2$. To see this, we resolve the syntactic sugar behind set expressions: $\lhd^*$ stands for $\{\lhd^+, =\}$ whereas $\neq$ abbreviates $\{\lhd^+, \rhd^+, \bot\}$; the intersection of both sets is $\{\lhd^+\}$. Rule (D.Parent.Ineq) states that a node that strictly dominates another also dominates its mother. A saturation step with this rules yields $X_2{\lhd^*}Y_1$, as expected.

5.4.2 Detecting Unsatisfiability

The rules we have seen so far cannot test even normal constraints for satisfiability. For this purpose, we need the additional rules in Figure 13.

(D.Clash)	$X \emptyset Y \to \mathsf{false}$	
(D.Lab.Disj)	$X{:}f(\ldots, X_i, \ldots, X_j, \ldots) \to X_i \bot X_j$	where $1 \leq i < j \leq n$
(D.Dom.Refl)	$\varphi \to X \lhd^* X$ $\quad$ X occurs in φ	
(D.Inv)	$XRY \to YR^{-1}X$	
(D.Dom.Trans)	$X \lhd^* Y \wedge Y \lhd^* Z \to X \lhd^* Z$	
(D.Lab.Dom)	$X{:}f(\ldots, Y, \ldots) \to X \lhd^+ Y$	

FIG. 13. Testing satisfiability of normal constraints

To illustrate the rules, we go through two simple examples of unsatisfiable normal constraints.

First, we have to deal with cycles Consider the constraint in Fig. 14.

$$X{:}f(Y) \wedge Y \lhd^* Z \wedge Z{:}f(X)$$

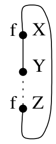


FIG. 14. An unsatisfiable constraint

With rule (D.Lab.Dom) we can derive $X \lhd^+ Y$. The weaker constraint $X \lhd^* Y$ can be added by the intersection rule (D.Inter) – the capability of weakening is built into the rule. Transitivity of dominance (D.Dom.Trans) gives us $X \lhd^* Z$. On the other hand, we can derive $Z \lhd^+ X$ from $Z{:}f(X)$ with (D.Lab.Dom). By inversion (D.Inv), we get $X \rhd^+ Z$. Finally, we use (D.Inter) to intersect $X \rhd^+ Z$ and $X \lhd^+ Z$ and obtain $X \emptyset Z$, which clashes by (D.Clash).

The second example is the unsatisfiable normal constraint in the left picture of Fig. 15. Here, we need case distinction to detect unsatisfiability. We sketch one of the four cases – the one depicted in the right picture of Fig. 15. As X_2 and X_5 both dominate X_7, we apply (D.NegDisj) and (D.Distr.NegDisj) to find that either $X_2 \lhd^* X_5$ or $X_5 \lhd^* X_2$ must hold. Suppose we choose $X_2 \lhd^* X_5$. Then by (D.Child.Ineq) and (D.Parent.Ineq) we derive $X_2 \lhd^* X_4$ and hence $X_2 \lhd^* X_6$ and $X_2 \lhd^* X_8$ by (D.Lab.Dom) and (D.Dom.Trans). But then we get $X_2 \neg\bot X_3$ by (D.NegDisj), which together with the fact that $X_2 \bot X_3$ – (D.Lab.Disj) – results in a clash.

5.4.3 Non-normal constraints

In non-normal constraints, we need to deal with two further phenomena: fragments that are not tree shaped, and fragments that overlap. If a constraint contains a fragment that is not tree-shaped, it is not satisfiable. As a simple example, consider the

$X_1 \neq X_2 \wedge$
$X_1 : f(X_2) \wedge X_2 \lhd^* Z$
$Y_1 : g(Y_2) \wedge Y_2 \lhd^* Z$
$Z : a$

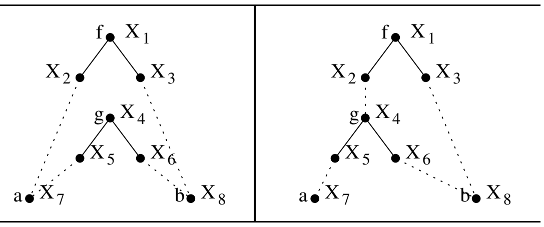


FIG. 15. An unsatisfiable normal constraint

constraint in Fig. 16.

$X : f(Y, Y)$

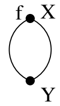


FIG. 16. A constraint with a fragment that is not tree-shaped

This constraint can already be dealt with by the rules of Fig. 13. As Y is the first as well as the second child of X, we derive $Y \bot Y$ by (D.Lab.Disj). Reflexivity of dominance (D.Dom.Refl) yields $Y \lhd^* Y$. We can intersect both relations with (D.Inter) and obtain $Y \emptyset Y$, which clashes by (D.Clash).

The more interesting case is that of overlapping fragments, as in Fig. 8. To handle cases like this, we need the additional saturation rules in Fig. 17. With these rules, we can first apply (D.Distr.Children), as we have both $Y : f(Y_1, Y_2)$ and $Y \lhd^* X$. This yields four cases. Two of them result in unsatisfiable constraints. The remaining cases are on the one hand $Y_1 \lhd^* X \wedge Y_2 \neg\lhd^* X$, and on the other hand $Y_1 \neg\lhd^* X \wedge Y_2 \neg\lhd^* X$. In the first case, we now have $X = Y_2$. In the second case, (D.Children.Up) gives us $Y = X$.

5.4.4 Soundness and Completeness

Procedure *D* is the collection of saturation rules in Fig. 12, 13, and 17. They are also repeated in a complete list of all saturation rules in the appendix. The procedure is *sound* and *complete*. We call a saturation procedure *S sound* if each rule of *S* is sound and each *S*-solved form of a constraint is satisfiable. We call it *complete* if for each solution (τ, α) of a constraint φ, *S* computes an *S*-solved form of φ of which (τ, α) is a solution.

Completeness is rather easy to prove: Saturation always terminates, and whenever we make a choice, we can easily pick one of the two options by inspecting the solution. Soundness is somewhat harder, the main lemma being the following.

(D.Eq.Decom)	$X{:}f(X_1,\dots,X_n) \wedge Y{:}f(Y_1,\dots,Y_n) \wedge X{=}Y \rightarrow \bigwedge_{i=1}^{n} X_i{=}Y_i$
(D.Children.Up)	$X \lhd^* Y \wedge X{:}f(X_1,\dots,X_n) \wedge \bigwedge_{i=1}^{n} X_i \neg\lhd^* Y \rightarrow Y{=}X$
(D.Distr.Children)	$X \lhd^* Y \wedge X{:}f(X_1,\dots,X_n) \rightarrow X_i \lhd^* Y \vee X_i \neg\lhd^* Y \qquad (1 \leq i \leq n)$
(D.Disj)	$X \perp Y \wedge Y \lhd^* Z \rightarrow X \perp Z$

FIG. 17. Dealing with overlaps

PROPOSITION 5.1
Every dominance constraint φ in *D*-solved form is satisfiable. Moreover, if φ contains two variables X and Y, $\varphi \models X{=}Y$ if and only if $X{=}Y$ belongs to φ.

The proposition is shown in [13] by constructing a concrete solution (τ, α) for a constraint φ in *D*-solved form. A special property of the construction is that if the literal $X{=}Y$ does not belong to φ, the solution always satisfies $\alpha(X) \neq \alpha(Y)$, which proves the second claim.

THEOREM 5.2 (Soundness and completeness)
Algorithm *D* is sound and complete for dominance constraints.

A particularly nice property of the solved forms enumerated by the algorithm is that they are *minimal* with respect to the inclusion order; that is, we cannot remove any literals without becoming non-solved.

5.5 *Saturation of binding constraints*

Finally, we have to deal with binding constraints. This can be done by adding the set B of saturation rules in Fig. 18 to the set D from above. The algorithm that is the union of the rules *D* and *B* is called *DB*.

(B.λ.Func)	$\lambda(X){=}Y \ \wedge\ \lambda(U){=}V \ \wedge\ X{=}U \rightarrow Y{=}V$
(B.λ.Dom)	$\lambda(X){=}Y \rightarrow Y \lhd^* X$
(B.λ.var)	$\lambda(X){=}Y \rightarrow X{:}var$
(B.λ.lam)	$\lambda(X){=}Y \rightarrow \exists Z\ (Y{:}\mathsf{lam}(Z))$
(B.ante.Func)	$\mathtt{ante}(X){=}Y \ \wedge\ \mathtt{ante}(U){=}V \ \wedge\ X{=}U \rightarrow Y{=}V$
(B.ante.ana)	$\mathtt{ante}(X){=}Y \rightarrow X{:}\mathsf{ana}$

FIG. 18. Rules for lambda and anaphoric binding

Rules (B.λ.Func) and (B.ante.Func) make sure that λ and ante are partial functions. (B.λ.Dom) expresses the fact that a var-labeled node must be bound by one of its ancestors. The three remaing rules check that variable, anaphoric, and lambda nodes bear the appropriate labels.

The rule (B.λ.lam) is the first rule we propose that introduces a new variable. This variable represents the root of the body of a λ-abstraction.

PROPOSITION 5.3
Every DB constraint in *DB*-solved form is satisfiable.

PROOF. Let φ be a DB constraint in *DB*-solved form. Let φ' be the pure dominance part of φ, which is clearly *D*-solved. By Prop. 5.1, there exists a solution $((V, E, L, \dots), \alpha)$ of φ' such that whenever $X{=}Y$ is not in φ, then $\alpha(X){\neq}\alpha(Y)$.

As in the proof of Theorem 4.3, we define partial functions $\lambda, \mathsf{ante} : V \rightsquigarrow V$ as follows: for all $X, Y \in \mathsf{Vars}(\varphi)$, $\lambda(\alpha(X)) = \alpha(Y)$ iff $\lambda(X){=}Y$ in φ, and $\mathsf{ante}(\alpha(X)) = \alpha(Y)$ iff $\mathsf{ante}(X){=}Y$ in φ. These functions are well-defined because φ is saturated under (B.λ.Func) and (B.ante.Func), and because we have chosen the model τ' such that whenever $X{=}Y$ is not in φ, then $\alpha(X){\neq}\alpha(Y)$.

Again, we check that $\tau = (V, E, L, \lambda, \mathsf{ante})$ is a partial lambda structure. It then clearly follows that (τ, α) satisfies φ except that τ is partial, which is irrelevant (as argued in the proof of Theorem 4.3). The domain of λ is a subset of $L^{-1}(\mathsf{var})$ by the saturation of φ under (B.λ.var), and its range is a subset of $L^{-1}(\mathsf{lam})$ by (B.λ.lam). The domain of ante is a subset of $L^{-1}(\mathsf{ana})$ by (B.ante.ana). Finally, each var-labeled node is situated below its binder by the saturation of φ under (B.λ.Dom). ∎

DB-saturation terminates for all DB constraints, since the only rule introducing additional variables, (B.λ.lam), can be applied only a finite number of times. The reason for this is that *DB*-saturation never adds any λ-binding literals. Hence completeness follows as above.

COROLLARY 5.4
Algorithm *DB* is sound and complete for DB constraints.

The solved forms that *DB* enumerates are again minimal, but not simply with respect to the set inclusion order. Instead, we need a notion of minimality that is set inclusion modulo some operations on new, existentially quantified variables. The exact definition is technically somewhat involved and thus exceeds the scope of this paper; it can be found in [19].

This concludes the first part of the article. The second part explains how to extend the system of saturation rules for dominance and binding to parallelism constraints.

6 The constraint language for lambda structures

Now that we have shown how to process dominance and binding constraints, we can discuss the full constraint language for lambda structures (CLLS), which extends DB constraints by *parallelism constraints*, and present a complete solver. Parallelism constraints are used in [14, 17] to model the meaning of VP ellipsis and its interaction with scope and anaphora. A more recent application is underspecified beta reduction[3, 4].

The definition of parallelism constraints consists of two parts: a fairly straightforward condition on the tree part of a lambda structure, and a slightly more involved condition on the binding functions. We will define the two parts separately. In this section, we present some basic intuitions and then define the syntax and semantics of parallelism without binding. Then we present the four core rules of the solution procedure in Section 7. We add the axioms for the binding part of parallelism in Section 8; these axioms will also serve as further rules in the solution procedure. In Section 9, we add a final group of rules and state completeness results.

6.1 *The Constraint Language over Lambda Structures*

The constraint language over lambda structures, CLLS, is obtained by extending dominance and binding constraints with parallelism literals:[1]

$$\begin{array}{lll} \varphi & ::= & X\,R\,Y \mid X{:}f(X_1\ \ldots\ X_n) \mid \lambda(X){=}Y \mid \mathsf{ante}(X) = Y \mid \varphi \wedge \varphi' \mid \mathsf{false} \\ & \mid & X_1/X_2{\sim}Y_1/Y_2 \end{array}$$

The constraints in the first line are those we have discussed in the first part of this article. The second line adds parallelism literals of the form $X_1/X_2{\sim}Y_1/Y_2$.

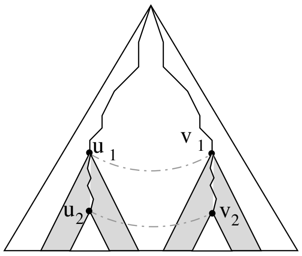


FIG. 19. Parallel tree segments: $u_1/u_2{\sim}v_1/v_2$

We interpret CLLS in λ-structures that are conservatively extended by a parallelism relation. The semantics of parallelism literals will be defined more precisely in Section 6.1, but the intuition is that parallelism relations hold between structurally isomorphic *segments* of the lambda structure. In Fig. 19, the u_i and v_i are nodes in a lambda structure. The shaded areas are the segments u_1/u_2 and v_1/v_2, respectively. We will state the isomorphism between the two segments by mapping the nodes of the two segments to each other. This *correspondence* is indicated in the picture by the dotted lines.

6.2 *Application to VP Ellipis*

The primary application of parallelism constraints to natural language semantics is the analysis of ellipsis, of which the following is a very simple example:

[1] It might be more precise to call this language "CLLS with set operators", as set operators were not part of the original definition [17].

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

The *target* sentence *so does Mary* means *Mary sleeps*. So the meaning of this sentence can be represented as the λ-structure in Fig. 20.

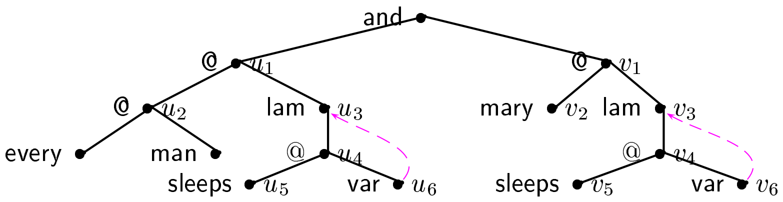


FIG. 20. λ-structure for *Every man sleeps, and so does Mary.*

The segments u_1/u_2 and v_1/v_2 of this λ-structure are structurally isomorphic: for each $1 \leq i \leq n$, u_i corresponds to v_i. Without having defined this, it should be intuitively clear that the binding structure is parallel in both segments.

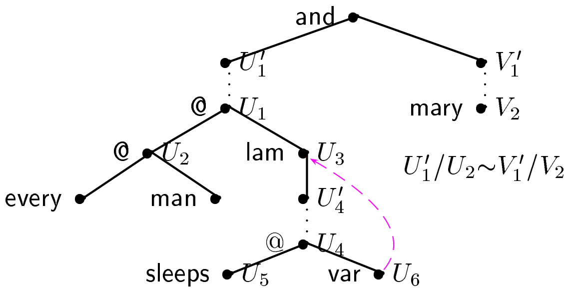


FIG. 21. Constraint for *Every man sleeps, and so does Mary.*

This λ-structure, call it τ for short, can be described canonically by the constraint in Fig. 21. With the variable assignment α given by $\alpha(U_i) = \alpha(U_i') = u_i$ and $\alpha(V_i) = \alpha(V_i') = v_i$ for all $1 \leq i \leq 6$, the pair (τ, α) is a solution of the constraint. The subgraph below U_1' represents the source sentence *every man sleeps*, while the target sentence is described by the subgraph below V_1'. These are all the constraints we need to describe the meaning of sentence (6.1). The parallelism literal $U_1'/U_2{\sim}V_1'/V_2$ asserts that the source context between U_1' and U_2 has the same structure as the target context between V_1' and V_2. So for the target, we only need to specify the exception, *mary*.

In the application to semantics, the syntax-semantics interface would derive the constraint in Fig. 21. Then this constraint would have to be solved in order to reconstruct the meaning of the target sentence.

6.3 Parallelism relations

In the previous paragraph, we have introduced the notions of *segments* and *correspondence functions*. We now make them precise.

DEFINITION 6.1
A *(tree) segment* u/u' in a lambda-structure τ consists of two nodes u and u' of τ satisfying $u\lhd^* u'$. The upper node u is called the *root* of the segment and the lower node u' its *hole*. The nodes in the tree segment are those nodes of the lambda structure that lie below the root but not properly below the hole.

$$\mathsf{b}(u/u') := \{w \in V_\tau \mid u\lhd^* w \text{ and } w(\lhd^* \cup \bot)u'\}.$$

The set of nodes that properly belong to a segment are all these nodes except the hole:

$$\mathsf{b}^-(u/u') := \mathsf{b}(u/u') - \{u'\}$$

In some applications, it is useful to have tree segments with more than one hole [3].

DEFINITION 6.2
A *correspondence function* between two tree segments u/u' and v/v' is a bijective mapping $c : \mathsf{b}(u/u') \to \mathsf{b}(v/v')$ such that for all nodes $w \in \mathsf{b}^-(u/u')$ and every label f of arity n in Σ it holds that:

$$w{:}f(w_1, \ldots, w_n) \Leftrightarrow c(w){:}f(c(w_1), \ldots c(w_n)).$$

Correspondence functions map roots to roots and holes to holes; but the two corresponding holes may carry different labels. Whenever it exists, the correspondence function between u/u' and v/v' is unique; we write $\mathrm{co}(\begin{smallmatrix} u & v \\ u' & v' \end{smallmatrix})$ for it.

Using the notions of segments and correspondence, we can now define parallelism relations.

DEFINITION 6.3
Parallelism in a λ-structure is the four-place relation $u_1/u_2{\sim}v_1/v_2$ which holds on a tuple of nodes iff the segments u_1/u_2 and v_1/v_2 have the same tree structure and parallel binding structures.

Two tree segments in a lambda structure have the *same tree structure* iff there exists a correspondence function between them.

The definition of "parallel binding structure" is somewhat more complex; it is presented in Section 8.1. Note that it would be sufficient to require that every tuple in the parallelism relation satisfies the two conditions; this would allow "parallelism relations" that are arbitrary subsets of the above relation. Choosing this more liberal definition makes no difference to satisfiability, but is sometimes useful in proofs.

7 Saturation for Parallelism Without Binding

In this section, we present the core saturation rules for solving parallelism literals, which spell out the equality of the tree structures on both sides, and we go through an example in more depth than above. The rules in this section form part of the sound and complete procedure in Section 9.

7.1 Abbreviations and auxiliary constraints

To be able to talk about nodes being inside, or properly inside, a segment, we define two constraint abbreviations. These are just shortcuts for conjunctions of CLLS constraints, so they do not add to our language.

$$\begin{array}{rcl} Z{\in}\mathsf{b}(X_1/X_2) & =_{\text{def}} & X_1{\lhd^*}Z \wedge Z(\lhd^* \cup \perp)X_2 \\ Z{\in}\mathsf{b}^-(X_1/X_2) & =_{\text{def}} & X_1{\lhd^*}Z \wedge Z(\lhd^+ \cup \perp)X_2 \end{array}$$

So a pair (τ, α) satisfies $Y{\in}\mathsf{b}(X_1/X_2)$ iff the node $\alpha(Y)$ is an element of the set $\mathsf{b}(\alpha(X_1)/\alpha(X_2))$ for τ. Note that we do not give meaning to the term $\mathsf{b}(X_1/X_2)$ as such.

We extend our constraint language by an auxiliary type of literals, *correspondence literals*, for speaking about correspondence functions.

$$\varphi \quad ::= \quad \ldots \mid \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U){=}V$$

Such a literal states that $X_1/X_2{\sim}Y_1/Y_2$ holds as well as $X_2{\in}\mathsf{b}(U/X_1)$ and $Y_2{\in}\mathsf{b}(V/Y_1)$, and that U corresponds to V with respect to the correspondence function for $X_1/X_2{\sim}Y_1/Y_2$.

We also define symmetric variants of the parallelism, again as constraint abbreviations:

$$\begin{array}{rcl} X_1/X_2 \overset{\mathrm{s}}{\sim} Y_1/Y_2 & =_{\text{def}} & X_1/X_2{\sim}Y_1/Y_2 \;\vee\; Y_1/Y_2{\sim}X_1/X_2 \\ \mathrm{co}^{\mathrm{s}}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U) = V & =_{\text{def}} & \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U) = V \;\vee\; \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(V) = U \end{array}$$

Note that we do not consider correspondence literals as part of CLLS as such; they are more of a calculus-internal bookkeeping mechanism. The saturation procedures we present below will only be complete for CLLS as defined in Section 6.1, not for arbitrary conjunctions of CLLS with auxiliary literals.

7.2 The Core Rules

Now we are ready to introduce the four core rules of the saturation procedure. First, the roots and holes of parallel segments correspond:

$$\text{(P.Root)} \quad X_1/X_2{\sim}Y_1/Y_2 \rightarrow \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(X_1){=}Y_1 \;\wedge\; \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(X_2){=}Y_2$$

Second, each node in a segment corresponds to some node in the parallel segment:

(P.New) $X_1/X_2 \overset{s}{\sim} Y_1/Y_2 \ \wedge\ U \in \mathsf{b}(X_1/X_2) \rightarrow \exists V\ \mathsf{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U) = V$

The function of (P.New) is to introduce new variables: Applied exhaustively to a constraint which contains a parallelism literal $X_1/X_2 \sim Y_1/Y_2$, it creates corresponding variables for all variables U that satisfy either $U \in \mathsf{b}(X_1/X_2)$ or $U \in \mathsf{b}(Y_1/Y_2)$. Note that by 5.3, (P.New) is applied only to variables that do not yet possess a correspondent with respect to this correspondence function, i.e. if there is no V' such that the constraint already contains $\mathsf{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U) = V'$.

Third, correspondence and inverse correspondence are homomorphisms: Corresponding nodes that properly lie in parallel segments carry the same labels and have corresponding children.

(P.Copy.Label) $\bigwedge_{i=0}^{n} \mathsf{co}^s(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U_i) = V_i \ \wedge\ U_0{:}f(U_1, \ldots, U_n) \ \wedge\ U_0 \in \mathsf{b}^-(X_1/X_2) \rightarrow V_0{:}f(V_1, \ldots, V_n)$

Fourth, the relation between two variables in a segment carries over to the corresponding variables in parallel segments.

(P.Copy.Dom) $U_1 R U_2 \ \wedge\ \bigwedge_{i=1}^{2} \mathsf{co}^s(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U_i) = V_i \rightarrow V_1 R V_2$

These last two axioms copy labeling, dominance, inequality and disjointness literals between corresponding variables.

7.3 Example: Solving parallelism without binding

We illustrate the four core rules by solving a case of quantifier parallelism. We use the a Hirschbhler sentence [22], in which scope and ellipsis interact:

(7.1) Every linguist attends a workshop, and every computer scientist does, too.

The sentence has three distinct readings: Either one single workshop is attended by everybody; or all linguists attend one common workshop, and all computer scientists visit a (potentially different) common workshop; or each linguist, and also each computer scientist, has some workshop of their own that they are travelling to. There are no "mixed" readings in which, e.g., all linguists gather at one workshop, while the computer scientists disperse to different workshops.

Fig. 22 shows a simplified version of a constraint describing the meaning of (7.1); we have compressed some fragments into single nodes, and we have omitted all binding literals.

Now let us apply our preliminary algorithm to solve this constraint. Throughout the example, we write co as a shortcut for $\mathsf{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})$, as this is the only correspondence function that appears here.

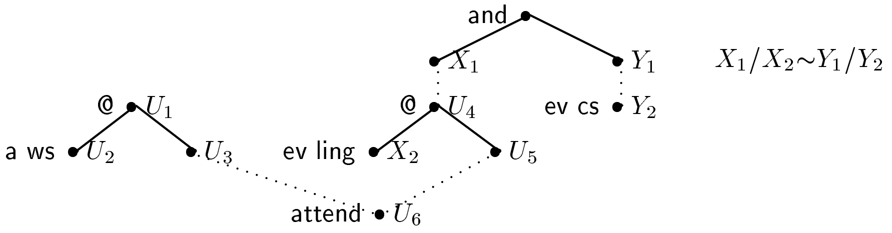


FIG. 22: Simplified constraint for *Every linguist attends a workshop, and every computer scientist does, too.*

First, let us work with the variables that we know (through saturation with the DB algorithm) to take values in one of the two parallel segments: X_1, X_2, U_4, U_5, U_6. We apply (P.Root) to record $\mathsf{co}(X_1){=}Y_1$, and $\mathsf{co}(X_2){=}Y_2$. Then, we make up new variables for the remaining correspondents. By applying (P.New) three times, we obtain new variables V_4, V_5, V_6 constrained by $\bigwedge_{i=4}^{6} \mathsf{co}(U_i){=}V_i$. Now we copy constraints from the source to the target sentence. For example, knowing that $U_4{:}@(X_2, U_5)$ and that $\mathsf{co}(U_4){=}V_4$, $\mathsf{co}(X_2){=}Y_2$, and $\mathsf{co}(U_5){=}V_5$, we get $V_4{:}@(Y_2, V_5)$ by (P.Copy.Label). Continuing in this manner, we arrive at the constraint pictured in Fig. 23.

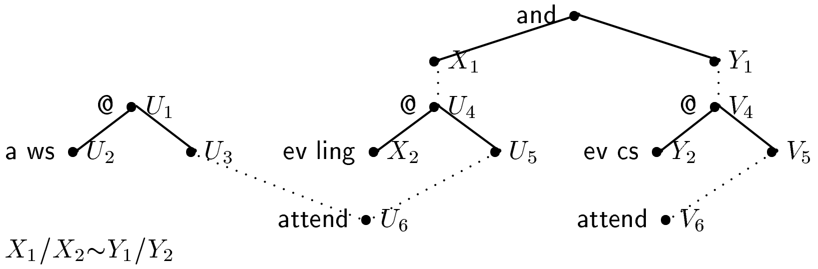


FIG. 23. Partial saturation of the constraint in Fig. 22

Now the parallelism procedure is stuck. The variables U_1, U_2, U_3 could end up either inside the source sentence (below X_1) or outside the ellipsis (above the conjunction), so we do not know whether we can apply (P.New) to these variables. To solve the constraint, we must now apply (D.NegDisj) and (D.Distr.NegDisj), e.g. to U_1 and U_4, which both dominate U_6.

We continue with the branch where we add $U_4 \triangleleft^* U_1$; the opposite case is similar. With $U_4 \triangleleft^* U_1$, the *DB* algorithm can derive $U_i {\in} \mathsf{b}(X_1/X_2)$ for $1 \leq i \leq 3$. That is, the three variables denote nodes inside the left parallel segment, and we can proceed as above, copying the variables with (P.New), and the literals that mention them with (P.Copy.Label) and (P.Copy.Dom). The result is shown in Fig. 24.

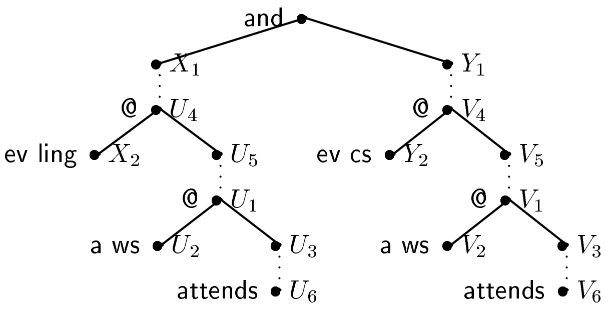


FIG. 24. A solved form for the constraint in Fig. 22

Note in particular that the algorithm does not allow mixed readings. In Fig. 24, V_1 corresponds to U_1, and we have $U_5 \triangleleft^* U_1$. But in that case we can immediately derive $V_5 \triangleleft^* V_1$ by (P.Copy.Dom). That is, because "a workshop" has received narrow scope in the source sentence, it is forced to get narrow scope in the target sentence as well.

8 Axioms and Saturation for Parallelism With Binding

Now we complete the definition of parallelism relations by specifying what "parallel binding structure" means. This time, we give the definition in the form of a set of *axioms* that parallelism relations must satisfy. The variables in the axioms range over all nodes of the λ-structure concerned. The advantage of that is that we can, by a slight abuse of notation, reuse these axioms as saturation rules for our solution procedure. After stating the axioms/saturation rules, we go through an example for illustration.

8.1 Parallel binding structure

DEFINITION 8.1
We say that two segments in a lambda have *parallel binding structures* if their correspondence function satisfies the following set of axioms.

For a var-labeled node bound within the segment, the corresponding node is bound correspondingly.

$$\text{(C.}\lambda\text{.Copy)} \quad \lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)(U_i){=}V_i \wedge U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow \lambda(V_1){=}V_2$$

For a var-node bound outside the segment, the corresponding node has the same binder.

$$\text{(C.}\lambda\text{.Above)} \quad \lambda(U_1){=}Y \ \wedge\ \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)(U_1){=}V_1 \ \wedge\ U_1{\in}\mathsf{b}^-(X_1/X_2) \ \wedge\ Y{\triangleleft^+}X_1 \ \rightarrow\ \lambda(V_1){=}Y$$

There are no ’hanging λ-binders’.

(C.λ.Hang) $\lambda(U_1){=}U_2 \;\wedge\; X_1/X_2 \overset{\mathrm{s}}{\sim} Y_1/Y_2 \;\wedge\; U_2{\in}\mathsf{b}^-(X_1/X_2) \rightarrow X_2 \neg\triangleleft^* U_1$

If an ana-node is bound within the segment, there are two possible antecedents for its corresponding node, matching the strict and the sloppy reading:

(C.ante.StrictSloppy) $\mathrm{ante}(U_1){=}U_2 \;\;\wedge\;\; \bigwedge_{i=1}^{2} \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U_i){=}V_i \;\;\wedge\;\; U_1{\in}\mathsf{b}^-(X_1/X_2) \;\rightarrow$ $\mathrm{ante}(V_1){=}U_1 \;\vee\; \mathrm{ante}(V_1){=}V_2$

If an ana-node is bound outside the segment, then its correspondent has the same anaphoric binder.

(C.ante.Above) $\mathrm{ante}(U_1){=}U_2 \wedge \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U_1){=}V_1 \wedge U_2(\triangleleft^+{\cup}\bot)X_1 \wedge U_1{\in}\mathsf{b}^-(X_1/X_2)$ $\rightarrow \mathrm{ante}(V_1){=}U_2$

(C.ante.Below) $\mathrm{ante}(U_1){=}U_2 \;\wedge\; \mathrm{co}(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix})(U_1){=}V_1 \;\wedge\; X_2\triangleleft^+U_2 \;\wedge\; U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow$ $\mathrm{ante}(V_1){=}U_2$

Axioms (C.λ.Copy), (C.λ.Above) and (C.λ.Hang) regulate the interaction of parallelism and variable binding, while axioms (C.ante.StrictSloppy), (C.ante.Above) and (C.ante.Below) deal with the interaction of parallelism and anaphoric binding. Most axioms should be self-explanatory, with two exceptions. (C.ante.StrictSloppy) will be explained in more detail in Section 9.3. (C.λ.Hang) prevents a certain overgeneration problem; its (linguistic) motivation is explained in [17].

8.2 *Example: Solving parallelism with binders*

As we have announced above, we now re-use the axioms from the previous paragraph as saturation rules. We use these rules on two examples to illustrate how the constraint solving procedure works.

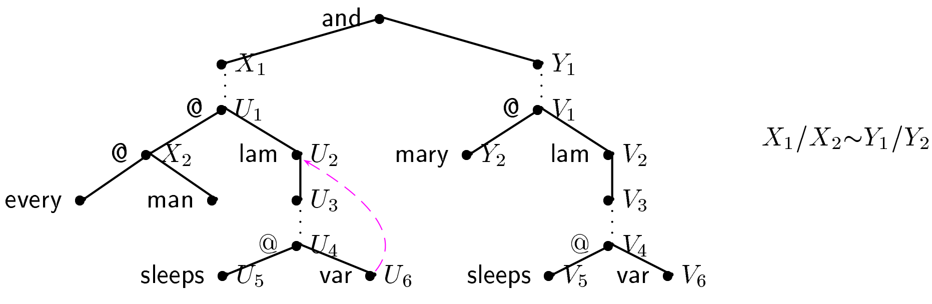


FIG. 25. Partial saturation of the constraint in Fig. 21; λ-link still missing

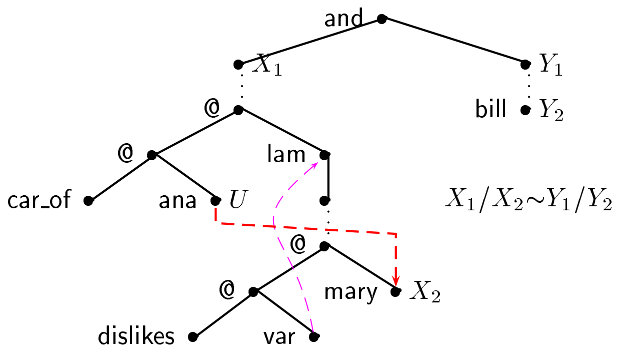


FIG. 26. Constraint for *Mary*$_1$ *dislikes her*$_1$ *car, and so does Bill.*

We first demonstrate the λ-binding rules. Fig. 21 shows the constraint for the sentence *Every man sleeps, and so does Mary*. To find a solved form for this constraint, we first proceed as for the Hirschbühler example in Section 7.2; this yields the constraint in Fig. 25. It remains to copy the binding literal $\lambda(U_6)=U_2$. This is achieved by applying (C.λ.Copy). We have $\mathsf{co}(U_6)=V_6$ and $\mathsf{co}(U_2)=V_2$, so we may add $\lambda(V_6)=V_2$.

As an example with anaphora, consider the following sentence:

(8.1) Mary$_1$ dislikes her$_1$ car, and so does Bill.

This sentence is an example of a *strict/sloppy ambiguity*. In the strict reading of this sentence, it is Mary's car that Bill does not like, while in the *sloppy* reading, Bill disapproves of his own car. The semantics of this sentence is described by the constraint in Fig. 26.

We saturate this constraint exactly as we did the two previous ones; among other things, we introduce a correspondent V for the anaphoric node U. It remains to handle the anaphoric binding. The anaphoric link $\mathsf{ante}(U)=X_2$ is entirely inside the source context, so it is copied using the distribution rule (C.ante.StrictSloppy). As the constraint contains $\mathsf{co}(X_2)=Y_2$, this rule adds either $\mathsf{ante}(V)=U$ or $\mathsf{ante}(V)=Y_2$.

Fig. 27 shows the first possibility, corresponding to the strict reading. V is bound by U, which in turn is bound by X_2, so we can reach $mary$ from V via a *link chain* [24]. The second possibility, $\mathsf{ante}(V)=Y_2$, corresponds to the sloppy reading.

9 Soundness and Completeness of the CLLS Solver

Finally, we present a sound and complete solution procedure for CLLS. We first introduce a relation on tree nodes that helps us compute with correspondences. Then we present the saturation rules we have left out before. Finally we state the soundness and completeness results and sketch a proof.

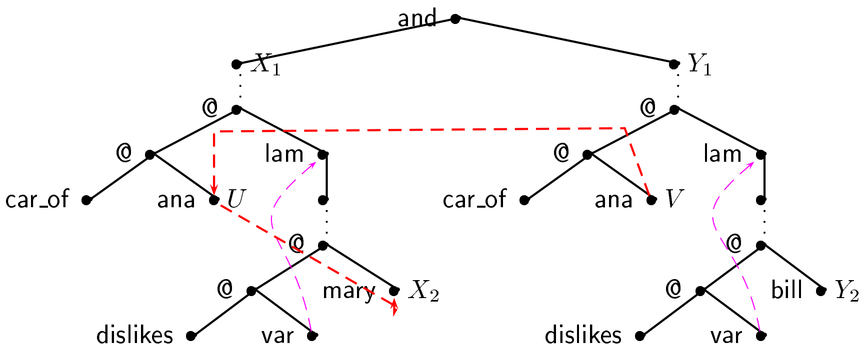


FIG. 27. Solved form for Fig. 26 (strict reading)

9.1 Path equality

We first introduce another relation on nodes of a λ-structure, the *path equality* relation. It states, informally speaking, that two paths in a λ-structure are equal in length and in the node labels passed.

DEFINITION 9.1
Let $\tau = (V, E, L, \lambda, \mathsf{ante})$ be a λ-structure. A *path equality* $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u & v \end{smallmatrix})$ holds in τ iff

- either $u{=}u_1$ and $v{=}v_1$,
- or E contains edges $(u', u), (v', v)$ such that $L(u') = L(v')$, and $L(u', u) = L(v', v)$, and $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u' & v' \end{smallmatrix})$ holds in τ.

The interesting point about path equality is that it can express correspondence:

LEMMA 9.2
Let τ be a lambda structure with nodes u_1, u_2, v_1, v_2 such that the correspondence function $\mathsf{co}(\begin{smallmatrix} u_1 & v_1 \\ u_2 & v_2 \end{smallmatrix})$ exists. The following property then holds for all nodes u, v of τ:

$$\mathsf{co}(\begin{smallmatrix} u_1 & v_1 \\ u_2 & v_2 \end{smallmatrix})\ (u){=}v \text{ iff } u \in \mathsf{b}(u_1/u_2) \text{ and } \ \mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u & v \end{smallmatrix})$$

PROOF. Let $\tau = (V, E, L, \lambda, \mathsf{ante})$. We abbreviate $\mathsf{co}(\begin{smallmatrix} u_1 & v_1 \\ u_2 & v_2 \end{smallmatrix})$ by c. For the $\Rightarrow$ direction, $c(u_1) = v_1$, and $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u_1 & v_1 \end{smallmatrix})$ always holds. Suppose $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ w & c(w) \end{smallmatrix})$ has been shown for $w \in \mathsf{b}^-(u_1/v_1)$, and $w{:}f(w_1, \ldots, w_n)$ holds. Then $c(w){:}f(c(w_1), \ldots c(w_n))$ by Def. 6.2, so $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ w_i & c(w_i) \end{smallmatrix})$ holds for $1 \leq i \leq n$.

Now for the $\Leftarrow$ case. We must have $c(u_1) = v_1$, so the case $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u_1 & v_1 \end{smallmatrix})$ is trivial. Now suppose $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u & v \end{smallmatrix})$ holds for $u{\neq}u_1$. Then by Def. 9.1 there are u', v' such that $\mathsf{p}(\begin{smallmatrix} u_1 & v_1 \\ u' & v' \end{smallmatrix})$ holds, u' and v' bear the same label, and u is the i-th child of u' and v the i-th child of v' for some i. Hence if $c(u') = v'$ holds, so does $c(u) = v$ by Def. 6.2. ∎

We extend our constraint language by *path equality* literals, which are interpreted by the path equality relation. Like correspondence literals, they are auxiliary literals

(P.Path.Corr)	$\mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)(U){=}V \to \mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ U & V \end{smallmatrix}\right) \ \wedge\ U{\in}\mathsf{b}(X_1/X_2)$
(P.Path.Sym)	$\mathsf{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \to \mathsf{p}\left(\begin{smallmatrix} Y & X \\ V & U \end{smallmatrix}\right)$
(P.Path.Dom)	$\mathsf{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \to X{\lhd^*}U \ \wedge\ Y{\lhd^*}V$
(P.Path.Eq.1)	$\mathsf{p}\left(\begin{smallmatrix} X_1 & X_3 \\ X_2 & X_4 \end{smallmatrix}\right) \ \wedge\ \bigwedge_{i=1}^{4} X_i{=}Y_i \to \mathsf{p}\left(\begin{smallmatrix} Y_1 & Y_3 \\ Y_2 & Y_4 \end{smallmatrix}\right)$
(P.Path.Eq.2)	$\mathsf{p}\left(\begin{smallmatrix} X & X \\ U & V \end{smallmatrix}\right) \to U{=}V$
(P.Trans.H)	$\mathsf{p}\left(\begin{smallmatrix} X & Y \\ U & V \end{smallmatrix}\right) \ \wedge\ \mathsf{p}\left(\begin{smallmatrix} Y & Z \\ V & W \end{smallmatrix}\right) \to \mathsf{p}\left(\begin{smallmatrix} X & Z \\ U & W \end{smallmatrix}\right)$
(P.Trans.V)	$\mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right) \ \wedge\ \mathsf{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right) \to \mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right)$
(P.Diff.1)	$\mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right) \ \wedge\ \mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right) \ \wedge\ X_2{\lhd^*}X_3 \ \wedge\ Y_2{\lhd^*}Y_3 \to \mathsf{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right)$
(P.Diff.2)	$\mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_3 & Y_3 \end{smallmatrix}\right) \ \wedge\ \mathsf{p}\left(\begin{smallmatrix} X_2 & Y_2 \\ X_3 & Y_3 \end{smallmatrix}\right) \ \wedge\ X_1{\lhd^*}X_2 \ \wedge\ Y_1{\lhd^*}Y_2 \to \mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)$

FIG. 28. Rules for correspondence functions and path equalities

rather than a proper part of CLLS.

$$\varphi \quad ::= \quad \ldots \mid \ \mathsf{p}\left(\begin{smallmatrix} X_1 & Y_1 \\ Y_1 & Z_1 \end{smallmatrix}\right)$$

9.2 *Further Rules*

The saturation rules we have presented so far are not even complete for CLLS without binding constraints. This is not surprising, as we have not shown any rules that talk about properties of correspondence functions. We now amend this by the rules in Fig. 28.

(P.Path.Corr) is justified by Lemma 9.2. The next four rules state obvious truths on path equalities. At the same time, these rules together with (P.New) ensure that correspondence functions are actually bijective functions. Rules (P.Trans.H) through (P.Diff.2) handle the proper interaction of correspondence functions when there is more than one parallelism.

The final ingredient to the solver are three distribution rules, given in Fig. 29. They are necessary to make the saturation strong enough to achieve completeness.

(P.Distr.Seg)	$X_1/X_2 \overset{\mathrm{s}}{\sim} Y_1/Y_2 \ \wedge\ X_1{\lhd^*}X \to X{\in}\mathsf{b}(X_1/X_2) \ \vee\ X_2{\lhd^+}X$
(P.Distr.Project)	$\varphi \to X{=}Y \ \vee\ X{\neq}Y$ where $X, Y \in \mathsf{Vars}(\varphi)$
(C.ante.Distr)	$\mathrm{ante}(U_1){=}U_2 \ \wedge\ X_1/X_2 \overset{\mathrm{s}}{\sim} Y_1/Y_2 \ \wedge\ U_1{\in}\mathsf{b}^-(X_1/X_2) \to$ $X_1{\lhd^*}U_2 \ \vee\ U_2{\lhd^+}X_1 \ \vee\ U_2{\perp}X_1$

FIG. 29. Distribution rules

The first rule guesses whether or not to put a variable into the sphere of action of a parallelism literal. The second rule guesses (in)equalities between variables. The third rule is only necessary when there are anaphoric linking constraints; it allows us to decide when we need to apply (C.ante.StrictSloppy), (C.ante.Above), or (C.ante.Below). The second rule in particular is very powerful and can lead to an explosion of the search space. Fortunately, it seems that this rule is not needed for linguistic examples (see Section 11.2).

9.3 Soundness and completeness

Combining all the rules we have discussed so far (which are all listed in Appendix A), we obtain the procedure *D* plus *P*, *DP* for short, for CLLS without binding constraints, and the procedure *DB* plus *P* and *C*, *DBPC* for short, for CLLS. These procedures are both sound and complete, in the sense defined in Section 5.4.4.

THEOREM 9.3
The procedure *DP* is sound and complete for CLLS without binding constraints. The procedure *DBPC* is sound and complete for CLLS.

For the soundness part, [19] construct a concrete solution (τ, α) for a given constraint φ in *DP*-solved form. They then check that all literals are indeed satisfied, which requires a tedious case distinction.

Concerning completeness, [19] show that *DP* computes all *minimal solved forms* of a constraint, with respect to the notion of minimality we have sketched at the end of Section 5.5. The proof of completeness in [19] only takes recourse to few of the actual saturation rules and is thus easily extended to *DBPC*.

10 The expressiveness of CLLS

The procedure for solving CLLS constraints is sound and complete, but it does not necessarily terminate, even when there are no binding constraints.

An example is shown in Fig. 30, Picture 1. This constraint describes two parallel segments that *overlap* without one of them being properly nested in the other: Y_1 must denote a node inside the segment described by X_1/X_2, and X_2 a node inside Y_1/Y_2. The constraint is unsatisfiable: there is no finite tree which, seen from the root, has an infinite number of f-labeled nodes followed by an infinite number of g-labeled nodes. If we run our algorithm on this constraint, it will keep copying f-labeled and g-labeled variables forever. Pictures 2 and 3 of Fig. 30 show two stages of the infinite copying process.

This behaviour is not surprising, as CLLS is at least as expressive as context unification (CU), a problem from unification theory whose decidability is open [48]. Decidability of CU is interesting on its own because CU is a specialization of (undecidable) second-order unification, but the best known lower bound is NP-hard, by encoding string unification [33].

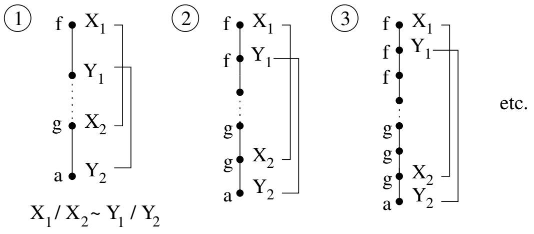


FIG. 30. A constraint for which the procedure never terminates

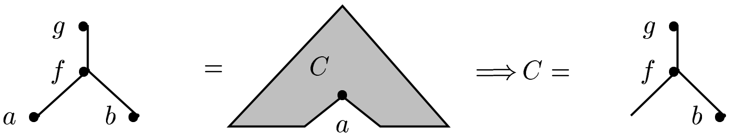


FIG. 31. $g(f(a,b)) = C(a)$

We devote the rest of this section to exploring the connection between CLLS and CU, generalizing an earlier equivalence result [41] and presenting an alternative encoding of CU into CLLS by example. The main idea is that we can see an *occurrence* of a context as a segment or, vice versa, we identifiy a context with the *structure* of a segment.

A *Context Unification problem* is a conjunction of equations between terms containing *context variables*. Given a set of function symbols $f, g, \dots$ and a set of context variables ranged over by C, context terms have the following formal syntax:

$$t ::= f(t_1, \dots, t_n) \mid C(t).$$

Usually, the syntax of context terms also comprises first-order variables, but this adds nothing to the expressiveness [42]. A context variable C denotes a ground term with exactly one *hole*, e.g. $g(f(\bullet, b))$. Alternatively, a context variable can be seen as a *context function*, a function that maps terms to terms, like $\lambda X.g(f(X,b))$.

A *solution* of a CU problem is a mapping of the context variables to context functions. For example, the CU problem

$$g(f(a,b)) = C(a)$$

has exactly one solution: C must be mapped to the context function $\lambda X.g(f(X,b))$ (Fig. 31). This context function corresponds to the term $g(f(\bullet, b))$ with exactly one hole. As before, we regard ground terms as constructor trees.

Now we can take *CLLS without binding* to be the sublanguage of CLLS that contains conjunctions only of labeling, dominance, and parallelism literals, and prove the following theorem.

THEOREM 10.1
The satisfiability problem of CLLS without binding is equivalent to context unification.

PROOF. On the one hand, every context unification problem can be encoded in *equality up-to constraints* [39], which can be rewritten using labeling and parallelism constraints [41]. On the other hand, [41] shows that any conjunction of labeling, dominance, and parallelism constraints can be written as a context unification problem. ∎

It is an open question whether path equality and binding constraints can be expressed in CU. The problem with binding is that the interaction of binding and parallelism (Def. 8.1) is rather intricate and makes use of correspondence functions; and correspondence literals may or may not exceed the expressiveness of CU.

One drawback of this proof is that the first direction (encoding of CU in CLLS) is very indirect. Here we present a direct encoding. We only discuss the example

$$g(f(C(a), b)) = C(g(f(a, b))),$$

but the encoding generalizes in a straightforward manner.

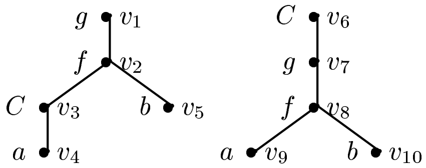


FIG. 32. Trees for $g(f(C(a), b))$ and $C(g(f(a, b)))$

First of all, we read the terms $g(f(C(a), b))$ and $C(g(f(a, b)))$ as finite constructor trees whose node labels are either ordinary function symbols or context variables. The result of this first step is shown in Fig. 32.

Now we form a constraint from the trees. We use one variable per tree node, say $X_1, \ldots, X_{10}$. (For ease of notation, we just use the same index for a tree node and the matching variable here.) For every tree node labeled by a function symbol, we put the matching labeling literal in our constraint – e.g. $X_1{:}g(X_2)$ and $X_2{:}f(X_3, X_4)$ for the first two nodes of the first tree. Now for every two nodes labeled by the same context variable, we put a parallelism literal in our constraint. v_3 and v_6 are both labeled C; they have v_4 and v_7 as their children, respectively. We translate this into the parallelism literal $X_3/X_4{\sim}X_6/X_7$.

Finally, we express equality of the two context terms by adding $X_1{=}X_6$ to our constraint: X_1 and X_6 are the variables for the two root nodes. The end result is shown in Fig. 33. It is satisfiable, as is the CU problem. Our example generalizes easily; conjunctions of equations are no problem either, as any conjunction $s_1{=}t_1 \wedge \ldots \wedge s_n{=}t_n$ of equations can be turned into an equivalent single equation $f(s_1, \ldots, s_n){=}f(t_1, \ldots, t_n)$.

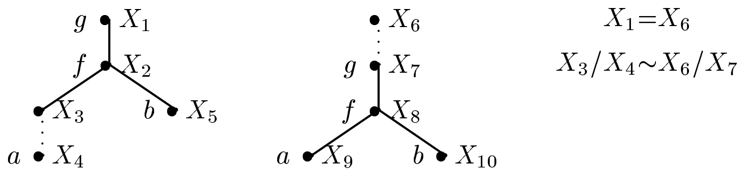


FIG. 33. Constraint for the problem $g\big(f(C(a), b)\big) = C\big(g(f(a, b))\big)$

11 Implementations

In this section, we give a brief overview of implemented solvers for DB and CLLS constraints. All mentioned implementations are available over the web at `http://www.coli.uni-sb.de/sfb378/projects/CHORUS-en.html`.

11.1 Solvers for dominance and binding constraints

The approach in [13] describes a solver for DB constraints that uses *concurrent constraint programming*. The idea is to encode dominance constraints into constraints over finite sets of integers and disjunctive propagators, as provided by the Mozart Programming System for the language Oz [49, 43]. The saturation rules of Section 5 can thereby be reduced to Mozart's propagation and distribution mechanisms.

Interestingly, the propagation power of this solver is so strong that on the examples from our application, it seems never to try a wrong choice; all choices made merely distinguish between the different solutions. Failure free search is a very much uncharacteristic behaviour when solving an NP-complete problem; it indicates that a polynomial time subproblem could be used in practice. In our case, such a fragment exists indeed: it is the fragment of normal dominance constraints presented in Section 4.2.

A polynomial time solver for *normal* DB constraints is desribed in [26]. It uses the graph algorithm from the proof of Theorem 4.3 that we have implemented in LEDA [36, 30], a library of common data structures and algorithms.

The general observation is that the graph algorithm is faster than the constraint implementation, which in turn is faster than the straightforward implementation of the saturation algorithm. This is illustrated in Table 16 on "chains" of growing lengths. A chain [28] is a type of constraint that is very common in scope underspecification; it alternates "upper" fragments (quantifiers) and "lower" fragments (nuclear scopes), as sketched in Fig. 34. Concrete examples in this paper include Fig. 6, a chain of length two, and the lower part of Fig. 7, which is a chain of length three.

The table displays the runtimes in milliseconds on a 350 MHz Pentium III and the numbers of minimal solved forms.

Runtimes in milliseconds

Length	Sols	graph (normal DB)	constraint (DB)	naive saturation (CLLS)
2	2	20	30	40
3	5	30	110	270
4	14	60	400	1510
5	42	170	1700	7000
6	132	610	5400	35200
7	429	2700	20700	> 150000
8	1430	9000	61000	> 500000

TABLE 16. Runtimes of the three DB implementations on chains of growing lengths.

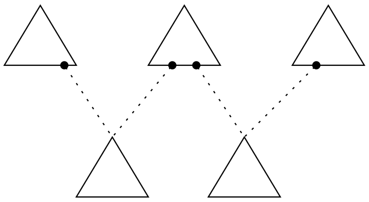


FIG. 34. A chain of length three

11.2 Solver for CLLS including parallelism

The best implementation for the CLLS solver that is known so far is the naive interpretation of the saturation system. To gain some efficiency, this implementation never applies (P.Distr.Project). Distribution rules are only applied to constraints that are saturated under all propagation rules except (P.New), and (P.New) is only applied to constraints that are saturated with all other rules.

Runtimes

Sentence	CU complete	CU incomplete	CLLS naive saturation
(11.1)	40 sec	1 sec	40 ms
(11.2)	n/a	15 sec	270 ms
(11.3)	2+ h	1 sec	4 sec

TABLE 17. Comparison of CU and CLLS implementations

A baseline against which we can compare this implementation are the (formally equivalent) solvers for context unification from the literature – a complete one from [40], and one where the most expensive rules were removed to improve efficiency [25]. Table 17 compares some runtimes, on the following sentences:

(11.1) Every man loves a woman.

(11.2) Every researcher of a company saw most samples.

(11.3) Peter likes Mary. John does, too.

The result of the comparison is that the CU solvers suffer badly in the examples with more complex scope ambiguities; the complete solver took several hours for the five-reading sentence. The CLLS solver, which can separate scope and ellipsis more cleanly, is much more efficient because it simply uses the DB rules. Another result is that the incomplete CU implementation can resolve the simple ellipsis (11.3) faster than the CLLS implementation can. This is because the current CLLS implementation has not been optimized at all; it is actually promising that the runtime is on the same order of magnitude as for the optimized CU solver. However, it is clearly still necessary to improve the implementation, e.g. by limiting the redundancy produced by the saturation system, or by sharing parts of constraints instead of copying them.

12 Conclusion and Outlook

In this paper, we have investigated algorithms and complexity results for CLLS as a whole and for the sublanguage of dominance and binding constraints. We have illustrated the algorithms on examples arising in an underspecified analysis of scope, anaphora, and ellipsis.

Because parallelism constraints are equivalent to context unification, one cannot expect that our saturation procedure terminates in general. This is, of course, unfortunate from a processing point of view. However, we can again look for fragments of the full language that are large enough for the linguistic application and on which our procedure does terminate. One possibility is to exclude models with parallel segments that overlap without being properly nested (see Section 10).

On the other hand, the results presented in this paper can act as a platform on which further useful extensions of the constraint language can be pursued. One such extension is *group parallelism* [3]; this is a generalization of parallelism to *sequences* of segments. Group parallelism is interesting because it can be used to represent beta reduction in an underspecified way [3, 4]. The saturation procedure presented here serves as a foundation for the saturation procedure for group parallelism.

References

[1] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In *Proceedings of the 30th ACL*, pages 32–39, Kyoto, 1992.

[2] R. Backofen, J. Rogers, and K. Vijay-Shanker. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language, and Information*, 4:5–39, 1995.

[3] Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Beta reduction constraints. In *Proc. 12th Rewriting Techniques and Applications*, Utrecht, 2001. To appear.

[4] Manuel Bodirsky, Katrin Erk, Alexander Koller, and Joachim Niehren. Underspecified beta reduction. Submitted. Available at `http://www.ps.uni-sb.de/Papers/abstracts/usp-beta.html`, 2001.

[5] Johan Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.

[6] Hubert Comon. Completion of rewrite systems with membership constraints. In *Coll. on Automata, Languages and Programming*, volume 623 of *LNCS*, 1992.

[7] Ann Copestake, Dan Flickinger, and Ivan Sag. Minimal Recursion Semantics. An Introduction. Manuscript, available at `ftp://csli-ftp.stanford.edu/linguistics/sag/mrs.ps.gz`, 1997.

[8] Thomas Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the 32nd ACL*, pages 163–170, 1994.

[9] Richard Crouch. Ellipsis and quantification: A substitutional approach. In *Proceedings of the 7th EACL*, pages 229–236, Dublin, 1995.

[10] Mary Dalrymple, Stuart Shieber, and Fernando Pereira. Ellipsis and higher-order unification. *Linguistics & Philosophy*, 14:399–452, 1991.

[11] Denys Duchier. A model-eliminative treatment of quantifier-free tree descriptions. In D. Heylen, A. Nijholt, and G. Scollo, editors, *Algebraic Methods in Language Processing, AMILP 2000, TWLT 16*, Twente Workshop on Language Technology (2nd AMAST Workshop on Language Processing), pages 55–66, Iowa City, USA, May 2000. Universiteit Twente, Faculteit Informatica.

[12] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In *Proceedings of IWCS-3*, Tilburg, 1999.

[13] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, LNCS. Springer, July 2000.

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

[15] Markus Egg. *Reinterpretation by Underspecification*. Habilitation thesis, University of the Saarland, 2000. To appear.

[16] Markus Egg. Reinterpretation from a synchronic and a diachronic point of view. In Regine Eckhart and Klaus von Heusinger, editors, *Meaning Change - Meaning Variation*, number 106 in Arbeitspapier der FG Sprachwissenschaft. Universität Konstanz, 2000.

[17] Markus Egg, Alexander Koller, and Joachim Niehren. The constraint language for lambda structures. *Journal of Logic, Language, and Information*, 2001. To appear.

[18] Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING/ACL'98)*, pages 353–359, Montreal, Canada, August 1998.

[19] Katrin Erk and Joachim Niehren. Parallelism constraints. In *International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Norwich, U.K., 2000. Springer-Verlag, Berlin.

[20] Claire Gardent and Michael Kohlhase. Higher-order coloured unification and natural language semantics. In *Proceedings ACL'96*, 1996.

[21] Claire Gardent and Bonnie Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.

[22] Paul Hirschbühler. VP deletion and across the board quantifier scope. In J. Pustejovsky and P. Sells, editors, *NELS 12*, Univ. of Massachusetts, 1982.

[23] Andrew Kehler. A discourse copying algorithm for ellipsis and anaphora resolution. In *Proceedings of EACL*, 1993.

[24] Andrew Kehler. *Interpreting Cohesive Forms in the Context of Discourse Inference*. PhD thesis, Harvard University, 1995.

[25] Alexander Koller. Evaluating context unification for semantic underspecification. In Ivana Kruijff-Korbayová, editor, *Proceedings of the Third ESSLLI Student Session*, pages 188–199, Saarbrücken, Germany, 1998.

[26] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th ACL*, Hong Kong, 2000.

[27] Alexander Koller and Joachim Niehren. On underspecified processing of dynamic semantics. In *Proceedings of COLING-2000*, 2000.

[28] Alexander Koller, Joachim Niehren, and Kristina Striegnitz. Relaxing underspecified semantic representations for reinterpretation. *Grammars*, 2000. Special Issue on MOL'99. To appear.

[29] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, 1998. To appear as LNCS.

[30] LEDA. The LEDA Library. Web page at `http://www.mpi-sb.mpg.de/LEDA/`, 2000.

[31] Jordi Lévy. Linear second order unification. In *International Conference on Rewriting Techniques and Applications*. Springer-Verlag, 1996.

[32] Jordi Lévy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In *International Conference on Rewriting Techniques and Applications*. Springer-Verlag, 2000.

[33] G.S. Makanin. The problem of solvability of equations in a free semigroup. *Soviet Akad. Nauk SSSR*, 223(2), 1977.

[34] Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*, pages 129–136, 1983.

[35] K. Mariott and P.J. Stuckey. *Programming with Constraints*. Kluwer Academic Publisher, 1998.

[36] Kurt Mehlhorn and Stefan Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 1999.

[37] Richard Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974.

[38] R.A. Muskens. Order-Independence and Underspecification. In J. Groenendijk, editor, *Ellipsis, Underspecification, Events and More in Dynamic Semantics*. DYANA Deliverable R.2.2.C, 1995.

[39] J. Niehren, M. Pinkal, and P. Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In *Proceedings 14th CADE*. Springer-Verlag, Townsville, 1997.

[40] J. Niehren, M. Pinkal, and P. Ruhrberg. A uniform approach to underspecification and parallelism. In *Proceedings ACL'97*, pages 410–417, Madrid, 1997.

[41] Joachim Niehren and Alexander Koller. Dominance Constraints in Context Unification. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, France, 1998. To appear as LNCS.

[42] Joachim Niehren, Ralf Treinen, and Sophie Tison. On rewrite constraints and context unification. *Information Processing Letters*, 74(1-2):25–40, April 2000.

[43] Oz Development Team. The Mozart Programming System web pages. `http://www.mozart-oz.org/`., 1999.

[44] Manfred Pinkal. Radical underspecification. In *Proceedings of the 10th Amsterdam Colloquium*, pages 587–606, 1996.

[45] Manfred Pinkal. On underspecification. In *Proceedings of the 5th International Workshop on Computational Semantics*, Tilburg, 1999. ITK, Tilburg University.

[46] Owen Rambow, K. Vijay-Shanker, and David Weir. D-Tree Grammars. In *Proceedings of the 33rd ACL*, 1995.

[47] Uwe Reyle. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.

[48] Decidability of context unification. The RTA list of open problems, number 90, `www.lri.fr/~rtaloop/`, 1998.

[49] Gert Smolka. The Oz Programming Model. In Jan van Leeuwen, editor, *Computer Science Today*, pages 324–343. Springer-Verlag, 1995.

[50] Kees van Deemter and Stanley Peters. *Semantic Ambiguity and Underspecification*. CSLI, Stanford, 1996.

[51] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.

[52] K. Vijay-Shanker, David Weir, and Owen Rambow. Parsing D-Tree Grammars. In *Proceedings of the Intl. Workshop on Parsing Technologies*, 1995.

A Solving CLLS Constraints: All Saturation Rules

In this appendix, we gather all the saturation rules we have mentioned throughout the paper.

Solving Dominance Constraints: Rule system D

Main rules

(D.NegDisj)	$X \triangleleft^* Z \wedge Y \triangleleft^* Z \rightarrow X \neg\bot Y$
(D.Distr.NegDisj)	$X \neg\bot Y \rightarrow X \triangleleft^* Y \vee Y \triangleleft^* X$
(D.Inter)	$X R_1 Y \wedge X R_2 Y \rightarrow X R Y$ if $R_1 \cap R_2 \subseteq R$
(D.Child.Ineq)	$X \neq Y \wedge X{:}f(\ldots, X', \ldots) \wedge Y{:}g(\ldots, Y', \ldots) \rightarrow X' \neq Y'$
(D.Parent.Ineq)	$X \triangleleft^+ Z \wedge Y{:}f(\ldots, Z, \ldots) \rightarrow X \triangleleft^* Y$

Testing satisfiability of normal constraints

(D.Clash)	$X \emptyset Y \rightarrow \mathsf{false}$
(D.Dom.Refl)	$\varphi \rightarrow X \triangleleft^* X$ X occurs in φ
(D.Inv)	$X R Y \rightarrow Y R^{-1} X$
(D.Dom.Trans)	$X \triangleleft^* Y \wedge Y \triangleleft^* Z \rightarrow X \triangleleft^* Z$
(D.Lab.Disj)	$X{:}f(\ldots, X_i, \ldots, X_j, \ldots) \rightarrow X_i \bot X_j$ where $1 \leq i < j \leq n$
(D.Lab.Dom)	$X{:}f(\ldots, Y, \ldots) \rightarrow X \triangleleft^+ Y$
(D.Disj)	$X \bot Y \wedge Y \triangleleft^* Z \rightarrow X \bot Z$

Dealing with overlaps

(D.Eq.Decom)	$X{:}f(X_1, \ldots, X_n) \wedge Y{:}f(Y_1, \ldots, Y_n) \wedge X{=}Y \rightarrow \bigwedge_{i=1}^n X_i{=}Y_i$
(D.Children.up)	$X \triangleleft^* Y \wedge X{:}f(X_1, \ldots, X_n) \wedge \bigwedge_{i=1}^n X_i \neg\triangleleft^* Y \rightarrow Y{=}X$
(D.Distr.Children)	$X \triangleleft^* Y \wedge X{:}f(X_1, \ldots, X_n) \rightarrow X_i \triangleleft^* Y \vee X_i \neg\triangleleft^* Y$ $(1 \leq i \leq n)$

Binding for Dominance Constraints: Rule System B

(B.λ.Func)	$\lambda(X){=}Y \ \wedge\ \lambda(U){=}V \ \wedge\ X{=}U \rightarrow Y{=}V$
(B.λ.Dom)	$\lambda(X){=}Y \rightarrow Y \triangleleft^* X$
(B.λ.var)	$\lambda(X){=}Y \rightarrow X{:}var$
(B.λ.lam)	$\lambda(X){=}Y \rightarrow \exists Z\ (Y{:}\mathsf{lam}(Z))$
(B.ante.Func)	$\mathrm{ante}(X){=}Y \ \wedge\ \mathrm{ante}(U){=}V \ \wedge\ X{=}U \rightarrow Y{=}V$
(B.ante.ana)	$\mathrm{ante}(X){=}Y \rightarrow X{:}\mathsf{ana}$

Solving CLLS Constraints: Rule System *P*

Main Rules

(P.Root) $X_1/X_2{\sim}Y_1/Y_2 \rightarrow \mathrm{co}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(X_1){=}Y_1 \;\wedge\; \mathrm{co}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(X_2){=}Y_2$

(P.New) $X_1/X_2\stackrel{s}{\sim}Y_1/Y_2 \;\wedge\; U{\in}\mathsf{b}(X_1/X_2) \rightarrow \exists V\; \mathrm{co}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U){=}V$

(P.Copy.Label) $\bigwedge_{i=0}^{n} \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_i){=}V_i \;\wedge\; U_0{:}f(U_1,\ldots,U_n) \;\wedge\; U_0{\in}\mathsf{b}^-(X_1/X_2) \rightarrow V_0{:}f(V_1,\ldots,V_n)$

(P.Copy.Dom) $U_1 R U_2 \;\wedge\; \bigwedge_{i=1}^{2} \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_i){=}V_i \rightarrow V_1 R V_2$

Properties of Path Equality Constraints

(P.Path.Corr) $\mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U){=}V \rightarrow \mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ U & V\end{smallmatrix}\right) \;\wedge\; U{\in}\mathsf{b}(X_1/X_2)$

(P.Path.Sym) $\mathsf{p}\left(\begin{smallmatrix}X & Y\\ U & V\end{smallmatrix}\right) \rightarrow \mathsf{p}\left(\begin{smallmatrix}Y & X\\ V & U\end{smallmatrix}\right)$

(P.Path.Dom) $\mathsf{p}\left(\begin{smallmatrix}X & Y\\ U & V\end{smallmatrix}\right) \rightarrow X{\triangleleft^*}U \;\wedge\; Y{\triangleleft^*}V$

(P.Path.Eq.1) $\mathsf{p}\left(\begin{smallmatrix}X_1 & X_3\\ X_2 & X_4\end{smallmatrix}\right) \;\wedge\; \bigwedge_{i=1}^{4} X_i{=}Y_i \rightarrow \mathsf{p}\left(\begin{smallmatrix}Y_1 & Y_3\\ Y_2 & Y_4\end{smallmatrix}\right)$

(P.Path.Eq.2) $\mathsf{p}\left(\begin{smallmatrix}X & X\\ U & V\end{smallmatrix}\right) \rightarrow U{=}V$

Interaction between correspondence functions

(P.Trans.H) $\mathsf{p}\left(\begin{smallmatrix}X & Y\\ U & V\end{smallmatrix}\right) \;\wedge\; \mathsf{p}\left(\begin{smallmatrix}Y & Z\\ V & W\end{smallmatrix}\right) \rightarrow \mathsf{p}\left(\begin{smallmatrix}X & Z\\ U & W\end{smallmatrix}\right)$

(P.Trans.V) $\mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right) \;\wedge\; \mathsf{p}\left(\begin{smallmatrix}X_2 & Y_2\\ X_3 & Y_3\end{smallmatrix}\right) \rightarrow \mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_3 & Y_3\end{smallmatrix}\right)$

(P.Diff.1) $\mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right) \;\wedge\; \mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_3 & Y_3\end{smallmatrix}\right) \;\wedge\; X_2{\triangleleft^*}X_3 \;\wedge\; Y_2{\triangleleft^*}Y_3 \rightarrow \mathsf{p}\left(\begin{smallmatrix}X_2 & Y_2\\ X_3 & Y_3\end{smallmatrix}\right)$

(P.Diff.2) $\mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_3 & Y_3\end{smallmatrix}\right) \;\wedge\; \mathsf{p}\left(\begin{smallmatrix}X_2 & Y_2\\ X_3 & Y_3\end{smallmatrix}\right) \;\wedge\; X_1{\triangleleft^*}X_2 \;\wedge\; Y_1{\triangleleft^*}Y_2 \rightarrow \mathsf{p}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)$

Distribution Rules

(P.Distr.Seg) $X_1/X_2\stackrel{s}{\sim}Y_1/Y_2 \;\wedge\; X_1{\triangleleft^*}X \rightarrow X{\in}\mathsf{b}(X_1/X_2) \;\vee\; X_2{\triangleleft^+}X$

(P.Distr.Project) $\varphi \rightarrow X{=}Y \;\vee\; X{\neq}Y$ where $X, Y \in \mathsf{Vars}(\varphi)$

Binding in CLLS Constraints: Rule System *C*

(C.λ.Copy) $\lambda(U_1){=}U_2 \wedge \bigwedge_{i=1}^{2} \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_i){=}V_i \wedge U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow \lambda(V_1){=}V_2$

(C.λ.Above) $\lambda(U_1){=}Y \;\wedge\; \mathrm{co}^{\mathrm{s}}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_1){=}V_1 \;\wedge\; U_1{\in}\mathsf{b}^-(X_1/X_2) \;\wedge\; Y{\triangleleft^+}X_1 \rightarrow \lambda(V_1){=}Y$

(C.λ.Hang) $\lambda(U_1){=}U_2 \;\wedge\; X_1/X_2\stackrel{s}{\sim}Y_1/Y_2 \;\wedge\; U_2{\in}\mathsf{b}^-(X_1/X_2) \rightarrow X_2{\neg\triangleleft^*}U_1$

(C.ante.StrictSloppy) $\mathrm{ante}(U_1){=}U_2 \;\wedge\; \bigwedge_{i=1}^{2} \mathrm{co}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_i){=}V_i \;\wedge\; U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow \mathrm{ante}(V_1){=}U_1 \;\vee\; \mathrm{ante}(V_1){=}V_2$

(C.ante.Above) $\mathrm{ante}(U_1){=}U_2 \wedge \mathrm{co}\left(\begin{smallmatrix}X_1 & Y_1\\ X_2 & Y_2\end{smallmatrix}\right)(U_1){=}V_1 \wedge U_2({\triangleleft^+}{\cup}{\perp})X_1 \wedge U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow \mathrm{ante}(V_1){=}U_2$

(C.ante.Below) $\mathrm{ante}(U_1){=}U_2 \wedge \mathrm{co}\left(\begin{smallmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{smallmatrix}\right)(U_1){=}V_1 \wedge X_2{\lhd^+}U_2 \wedge U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow \mathrm{ante}(V_1){=}U_2$

(C.ante.Distr) $\mathrm{ante}(U_1){=}U_2 \wedge X_1/X_2 \overset{s}{\sim} Y_1/Y_2 \wedge U_1{\in}\mathsf{b}^-(X_1/X_2) \rightarrow X_1{\lhd^*}U_2 \vee U_2{\lhd^+}X_1 \vee U_2{\perp}X_1$

Received March 2001.