Dominance Constraints:
Algorithms and Complexity

Alexander Koller[1], Joachim Niehren[2], and Ralf Treinen[3]

[1] Department of Computational Linguistics, Universität des Saarlandes, Saarbrücken, Germany, koller@coli.uni-sb.de

[2] Programming Systems Lab, Universität des Saarlandes, Saarbrücken, Germany, niehren@ps.uni-sb.de

[3] Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay, France, treinen@lri.fr

Abstract. Dominance constraints for finite tree structures are widely used in several areas of computational linguistics including syntax, semantics, and discourse. In this paper, we investigate algorithmic and complexity questions for dominance constraints and their first-order theory. The main result of this paper is that the satisfiability problem of dominance constraints is NP-complete. We present two NP algorithms for solving dominance constraints, which have been implemented in the concurrent constraint programming language Oz. Despite the intractability result, the more sophisticated of our algorithms performs well in an application to scope underspecification. We also show that the positive existential fragment of the first-order theory of dominance constraints is NP-complete and that the full first-order theory has non-elementary complexity.

Keywords. Dominance constraints, complexity, computational linguistics, underspecification, constraint programming.

1 Introduction

Dominance constraints are a popular tool for describing trees throughout computational linguistics. They allow to express both *immediate dominance* (and labeling) relations and general (reflexive, transitive) *dominance* relations between the nodes of a tree. In syntax, they provide for underspecified tree descriptions employed in deterministic parsing [MHF83] and to combine TAG with unification grammars [VS92]. In underspecification of the semantics of scope ambiguities, dominance constraints are omnipresent. While they are somewhat implicit in earlier approaches [Rey93,Bos96], they are used explicitly in two recent formalisms [ENRX98,Mus98]. An application of dominance constraints in discourse semantics has recently been proposed in [GW98], and they have been used to model information growth and partiality [MVK98].

Despite their popularity, there have been no results about the computational complexity of solving these constraints, i.e. of finding a tree that satisfies all given constraints. In this paper, we remedy this situation by proving that the satisfiability problem of dominance constraints is NP-complete. This result holds for all logical fragments between the purely conjunctive fragment and the positive existential fragment. We present two algorithms for solving them; one involves a nondeterministic guessing step, which makes it convenient for a completeness proof, but not for implementation, whereas the other gives priority to deterministic computation steps and enumerates cases only by need. Finally, we show that the first-order theory over dominance constraints with a signature of bounded arity is decidable and has non-elementary complexity. The decidability result is not new – e.g. [Rog94] sketches a proof for a different variant of dominance constraints –, but we work out the details of a transparent proof by encoding into second-order monadic logic for the first time.

Related Work. In [RVS92], it was shown how to solve formulae from the propositional language over (a different variant of) dominance constraints (over a different type of trees). There, tableau-style saturation rules for enumerating models are presented which are quite similar to the ones we use here. This solution procedure terminates, but there are no complexity results. Continuing this line of work, [BRVS95] present a sets of first-order axioms over dominance constraints which capture certain classes of trees.
From an implementation perspective, dominance constraints were approached first in [DG99], which presents an implementation based on *finite set constraints*. A more advanced version of the algorithm presented here and an implementation thereof are given [DN99]. This implementation is also based on finite set constraint programming but improves that of [DG99].

Plan of the paper. In Section 2, we start out by defining the syntax and semantics of dominance constraints. In Section 3, we present the solution algorithms for dominance constraints and prove their soundness, completeness, and NP runtimes. The algorithms are first defined for the (purely conjunctive) language of dominance constraints and extended to the other propositional connectives later. In Section 4, we complement this result by proving NP-hardness of the problem. In fact, we will not really provide the details of the proof, but give a thorough explanation of the proof idea. In Section 5, we turn to the decidability and complexity of the first-order theory over dominance constraints. Section 6 summarizes and concludes the paper. Some of the proofs are only sketched; for more details, we refer the reader to [Kol99].

2 Syntax and Semantics of Dominance Constraints

In this section, we define the syntax and semantics of dominance constraints. To this end, we first introduce the notion of a *tree structure*, the kind of first-order structure we will interpret dominance constraints over. After that, it will be straightforward to define the actual syntax and semantics. Finally, we look briefly at *constraint graphs*, a graphical syntactic alternative.

2.1 Tree structures

Throughout, we take $\mathbb{N}$ to be the set of positive integers and $\mathbb{N}_0$ to be the set of nonnegative integers and assume that Σ is a ranked signature that contains *function symbols* or *tree constructors* $f, g, a, b, \ldots$, which are assigned arities by an arity function $\mathsf{ar} : \Sigma \to \mathbb{N}_0$. We further assume that Σ contains at least two constructors, one of which is nullary, and the other one of arity at least 2.

Intuitively, we want trees to be essentially the same as ground terms over Σ. Formally, we first define a *tree domain* D to be a nonempty prefix-closed subset of $\mathbb{N}^*$; i.e., the elements of D are words of positive integers. These words can be thought of as the paths from the root of a tree to its nodes. We write the concatenation of two words π and π' as juxtaposition $\pi\pi'$.

We define a *constructor tree* τ to be a pair (D_τ, L_τ) of a tree domain D_τ and a *labeling function*

$$L_\tau : D_\tau \to \Sigma,$$

with the additional property that for every $\pi \in D_\tau$, $\pi k \in D_\tau$ iff $1 \leq k \leq \mathsf{ar}(L_\tau(\pi))$. A *finite constructor tree* is a constructor tree whose domain is finite. Throughout, we will simply say "tree" to mean "finite constructor tree".

The *tree structure* $\mathcal{M}^\tau$ over the tree τ is a first-order model structure with the universe D_τ and whose interpretation function assigns relations over D_τ to a set of fixed predicate symbols. We will use the same symbols for the predicate symbols and their interpretations; as the latter are applied to paths and the former are applied to variables, there is no danger of confusion. The interpretation function is fully determined by τ; so to specify a tree structure, it is sufficient to specify the underlying tree.

In detail, the interpretation is as follows. If $f \in \Sigma$ has arity n, the *labeling* relation $\pi{:}f(\pi_1, \ldots, \pi_n)$ is true in $\mathcal{M}^\tau$ iff $L_\tau(\pi) = f$ and for all $1 \leq i \leq n$, $\pi_i = \pi i$. The *dominance* relation $\pi \triangleleft^* \pi'$ is true iff π is a prefix of π'.

2.2 Syntax and Semantics of Dominance Constraints

With these definitions, it is straightforward to define the syntax and semantics of dominance constraints. Assuming a set of (node) variables $X, Y, \ldots$, a *dominance constraint* φ has the following abstract syntax:

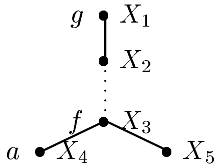
$$\begin{aligned} \varphi ::= {} & X{:}f(X_1, \ldots, X_n) \qquad f \in \Sigma,\ n = \mathsf{ar}(f) \\ \mid {} & X \triangleleft^* Y \\ \mid {} & \varphi \wedge \varphi'. \end{aligned}$$

We use the formula $X = Y$ as an abbreviation for $X \triangleleft^* Y \wedge Y \triangleleft^* X$.

We will start by considering only this (purely conjunctive) *constraint language* and successively allow more logical connectives, until we have arrived at the full first-order language in Section 5.

Satisfaction of an atomic constraint is defined with respect to a pair $(\mathcal{M}^\tau, \alpha)$ of a tree structure $\mathcal{M}^\tau$ and a variable assignment $\alpha : \mathrm{Var} \to D_\tau$ that assigns nodes to the variables. This is extended to satisfaction of arbitrary formulae in the usual Tarskian way.

Because dominance constraints can easily become unreadable, we will use *constraint graphs* as a graphical device to represent them. They are essentially an alternative to the original syntax of the language. Constraint graphs are directed graphs with two kinds of edges: solid and dotted. The nodes of a constraint graph represent variables in a constraint. Labeling constraints are expressed by attaching the constructor to the node and drawing solid edges to the children; dominance constraints are expressed by drawing a dotted edge between the respective nodes. As an example, the graph below is the constraint graph for the constraint to its right.



$X_1{:}g(X_2) \wedge X_2 \triangleleft^* X_3 \wedge$
$X_3{:}f(X_4, X_5) \wedge X_4{:}a.$

3 Solving Dominance Constraints

Now we show that the satisfiability problems of all languages over dominance constraints between the (purely conjunctive) constraint language itself and the positive existential fragment are in NP. We first define an algorithm that decides satisfiability for the constraint language and prove the running time, soundness, and completeness. Then we present an algorithm that does the same thing, but lends itself more easily to implementation. Finally, we extend the results to the other propositional connectives.

It turns out that it's actually easier to define satisfiability algorithms for dominance constraints if we additionally allow atomic constraints of the form $\neg X \triangleleft^* Y$. Hence, we are going to work with this extended language of dominance constraints in Sections 3.1 to 3.3.

3.1 The Algorithm

The first algorithm proceeds in three steps. First, we guess nondeterministically for each pair X, Y of variables in φ if X dominates Y or not, and add the corresponding atomic constraint to φ. This is done by the (Choice) rule, where **or** stands for nondeterministic choice.

$$\text{(Choice)} \quad \textbf{true} \quad \rightarrow \quad X \triangleleft^* Y \textsf{ or } \neg X \triangleleft^* Y$$

In the second step, we saturate φ according to the following deterministic *propagation* rules. We recall that $X = Y$ is an abbreviation for $X \triangleleft^* Y \wedge Y \triangleleft^* X$.

(Refl)	$\mathbf{true} \;\rightarrow\; X \triangleleft^* X$	(X occurs in φ)
(Trans)	$X \triangleleft^* Y \wedge Y \triangleleft^* Z \;\rightarrow\; X \triangleleft^* Z$	
(Decomp)	$X{=}Y \wedge X{:}f(X_1, \ldots, X_n) \wedge Y{:}f(Y_1, \ldots, Y_n) \;\rightarrow\; \bigwedge_{i=1}^{n} X_i{=}Y_i$	
(Disj)	$X{:}f(\ldots, X_i, \ldots, X_k, \ldots) \;\rightarrow\; \neg X_i \triangleleft^* X_k \quad (1 \leq i \neq k \leq n)$	
(Dom)	$X{:}f(\ldots, Y, \ldots) \;\rightarrow\; X \triangleleft^* Y$	
(Parent)	$X{=}Y \wedge X'{:}f(\ldots, X, \ldots) \wedge Y'{:}g(\ldots, Y, \ldots) \;\rightarrow\; X'{=}Y'$	
(Child)	$X \triangleleft^* Y \wedge X{:}f(X_1, \ldots, X_n) \wedge \bigwedge_{i=1}^{n}(\neg X_i \triangleleft^* Y) \;\rightarrow\; Y \triangleleft^* X$	

In the Parent rule, f and g need not be different.
In the third step, we detect unsatisfiable constraints by applying the following *clash* rules.

(Clash1)	$X{:}f(\ldots) \wedge Y{:}g(\ldots) \wedge X{=}Y \;\rightarrow\; \mathbf{false},$	if $f \neq g$
(Clash2)	$X \triangleleft^* Z \wedge Y \triangleleft^* Z \wedge \neg X \triangleleft^* Y \wedge \neg Y \triangleleft^* X \;\rightarrow\; \mathbf{false}$	
(Clash3)	$X \triangleleft^* Y \wedge \neg X \triangleleft^* Y \;\rightarrow\; \mathbf{false}$	
(Clash4)	$X{:}f(X_1, \ldots, X_i, \ldots, X_n) \wedge X_i \triangleleft^* X \;\rightarrow\; \mathbf{false}$	

After the initial guessing step, the algorithm applies all instances of all propagation and clash rules. We call a constraint to which no clash rule can be applied *clash-free*, the result of applying all possible rules to a constraint for as long as the constraint is clash-free its *saturation*, and a constraint which is its own saturation *saturated*. The algorithm outputs that its input is satisfiable if it can find a clash-free saturation (that is, can apply the guessing step in such a way that subsequent propagation and clash rules won't produce **false**); otherwise, it outputs that the input is unsatisfiable.
An example for application of these rules is to prove the unsatisfiability of

$$X{:}a \wedge X \triangleleft^* Y \wedge \neg Y \triangleleft^* X,$$

where a is a nullary symbol. Application of the (Child) rule adds the new constraint $Y \triangleleft^* X$. But this makes the (Clash3) rule applicable, so the algorithm finds a clash. A really tricky example is to prove the unsatisfiability of

$$Y{:}f(Z) \wedge X{:}g(U) \wedge U \triangleleft^* Z \wedge \neg X \triangleleft^* Y.$$

The (Dom) and (Trans) rules will give us $Y \triangleleft^* Z$, $X \triangleleft^* U$, and $X \triangleleft^* Z$. Now for the saturation to be clash-free, (Choice) must have guessed $Y \triangleleft^* X$; for if it had chosen $\neg Y \triangleleft^* X$, we would get a clash with (Clash2). Similarly, (Choice) must have guessed $\neg Z \triangleleft^* X$, for if it had chosen $Z \triangleleft^* X$, we could derive $U \triangleleft^* X$, which produces a clash with (Clash4). But in this case, we can derive $X \triangleleft^* Y$ with the (Child) rule, which causes a clash with (Clash3).
It's easy to see that the algorithm terminates in NP time. As we have guessed the dominance relations between all variables in the first step, the second step can never consistently add a new constraint; either the constraint is already known, or it clashes, by the Clash3 rule. So we will only spend deterministic polynomial time with the application of propagation and clash rules. Note that one major change of the second algorithm below will be to allow the propagation rules to be more productive.

Proposition 1 (Soundness). *A satisfiable dominance constraint has a clash-free saturation.*

Proof. Assume that the constraint φ is satisfiable. Clearly, the guessing step of the algorithm can add a choice of (possibly negated) dominance constraints such that their conjunction φ' with φ is satisfiable as well; we only have to read off whether the denotations of two variables dominate each other in a fixed solution of φ. Now all propagation rules maintain satisfiability, and the preconditions of all clash rules are unsatisfiable. Hence, φ' is saturated and clash-free. $\square$

3.2 Completeness

As usual, proving completeness is slightly more involved than proving soundness. Here, we proceed in two steps: First we show that a special class of saturated, clash-free constraints is satisfiable; then we show that every saturated, clash-free constraint can be extended by some additional conjuncts to a saturated, clash-free constraint of the restricted class. Together, this shows completeness:

Proposition 2 (Completeness). *A saturated and clash-free constraint is satisfiable.*

Incidentally, the proof also shows how to obtain a model for a clash-free constraint. But first, some terminology. We call $V \subseteq V(\varphi)$ an *equality set for* φ if $Y_1 \lhd^* Y_2$ *in* φ for all $Y_1, Y_2 \in V$. All variables in an equality set must be mapped to the same node in a solution of φ. A variable X is *labeled in* φ if there is an X' such that $\{X, X'\}$ is an equality set for φ and $X'{:}f(X'_1, \ldots, X'_n)$ *in* φ for some term $f(X'_1, \ldots, X'_n)$. We call a constraint φ *simple* if all its variables are labeled, and if there is a so-called *root variable* Y for φ such that $Y \lhd^* Z$ *in* φ for all $Z \in V(\varphi)$.

Lemma 1 (Satisfiability of Simple Constraints). *A simple, saturated, and clash-free constraint is satisfiable.*

Proof. It is not difficult to show that for any $Z \in V(\varphi)$, there is a unique sequence of maximal equality sets $E_1, \ldots, E_n$ that connect the root of φ to Z via labeling constraints. From this, we can read off the satisfying tree structure and variable assignment in a straightforward way. $\square$

It remains to show that we can restrict our attention to simple constraints. An *extension of a constraint* φ is a constraint of the form $\varphi \wedge \varphi'$ for some φ'. We will show how to extend a saturated, clash-free constraint to a simple, saturated, and clash free constraint.
We define the set $\mathsf{con}_\varphi(X)$ of variables *connected to* X *in* φ as follows:

$$\mathsf{con}_\varphi(X) = \left\{ Y \;\middle|\; \begin{array}{l} X \lhd^* Y \textit{ in } \varphi, Y \lhd^* X \textit{ not in } \varphi, \text{ not exists } Z \text{ s.t.} \\ X \lhd^* Z, Z \lhd^* Y \textit{ in } \varphi, Z \lhd^* X, Y \lhd^* Z \textit{ not in } \varphi \end{array} \right\}$$

Note that for this algorithm, $X \lhd^* Y$ *not in* φ is the same as $\neg X \lhd^* Y$ *in* φ; it does, however, make a difference for the second algorithm below. Intuitively, a variable is connected to X if it is a "minimal dominance child" of X. So for example, in

$$\varphi_1 := X \lhd^* X \wedge X \lhd^* Y \wedge \neg Y \lhd^* X \wedge X \lhd^* Z \wedge \neg Z \lhd^* X \wedge Y \lhd^* Z \wedge \neg Z \lhd^* Y,$$

$\mathsf{con}_{\varphi_1}(X) = \{Y\}$ and $\mathsf{con}_{\varphi_1}(Y) = \{Z\}$.
We call $V \subseteq V(\varphi)$ a *disjointness set for* φ if for any two distinct variables $Y_1, Y_2 \in V$, $Y_1 \lhd^* Y_2$ *not in* φ. The idea is that all variables in a disjointness set can safely be placed at disjoint positions of a tree.

Lemma 2. *If φ is saturated and $X \in V(\varphi)$ then for all $Y_1, Y_2 \in \mathsf{con}_\varphi(X)$, the set $\{Y_1, Y_2\}$ is either an equality or disjointness set for φ.*

For a constraint φ and a variable X of φ, Lemma 2 implies the existence of maximal disjointness sets $V \subseteq \mathsf{con}_\varphi(X)$ for φ. Such a set is constructed by choosing one representative from every maximal equality set contained in $\mathsf{con}_\varphi(X)$.
Now we can state and prove the key lemma of the completeness proof.

Lemma 3 (Extension by Labeling). *Let φ be a constraint and X a variable that is unlabeled in φ. Let $\{X_1, \ldots, X_n\} \subseteq \mathsf{con}_\varphi(X)$ be a disjointness set for φ that is maximal among all disjointness sets that are subsets of $\mathsf{con}_\varphi(X)$, and let f be a function symbol of arity n. If φ is saturated and clash-free, then $\varphi \wedge X{:}f(X_1, \ldots, X_n)$ is also saturated and clash-free.*

Proof. Let $\varphi' = \varphi \wedge X{:}f(X_1, \ldots, X_n)$. Since we have not introduced new variables or dominance relations, φ' inherits saturation with respect to the nondeterministic guessing rule, (Refl), and (Trans) and clash-freeness with respect to (Clash2) and (Clash3) from φ'. The rule (Decomp) is not applicable to φ'; otherwise, X would have been labeled in φ. By the same argument, the (Clash1) rule is not applicable to φ'. The only new way to apply the (Dom) rule is to the new labeling constraint; but the dominances (Dom) can derive are already in φ. The (Clash4) rule is not applicable, either: No $X_i \lhd^* X$ can be in φ' because $X_i \in \mathsf{con}_\varphi(X)$. The arguments for the remaining rules are more interesting:

(Disj) The only new way in which the (Disj) might apply is as follows:

$$X{:}f(X_1, \ldots, X_n) \to \neg X_i \lhd^* X_j \quad (i \neq j)$$

By assumption, $\{X_1, \ldots, X_n\}$ is a disjointness set for φ. Hence $\neg X_i \lhd^* X_j$ *in* φ, i.e. φ' is saturated under (Disj).

(Child) The only possible case in which the (Child) rule might newly apply looks as follows:

$$X \lhd^* Y \wedge X{:}f(X_1, \ldots, X_n) \wedge \bigwedge_{i=1}^{n} (\neg X_i \lhd^* Y) \to Y \lhd^* X$$

But since we have chosen $\{X_1, \ldots, X_n\}$ to be a maximal disjointness set of $\mathsf{con}_\varphi(X)$, it follows from $X \lhd^* Y$ *in* φ that either $Y \lhd^* X$ is already in φ, or there is a $1 \leq i \leq n$ such that $X_i \lhd^* Y$ *in* φ (compare Lemma 2). Hence, the (Child) rule is not applicable in any new way either.

(Parent) Finally, the only non-trivial new way in which to apply the (Parent) rule looks as follows.

$$X_i{=}Z \wedge X{:}f(\ldots, X_i, \ldots) \wedge Y{:}g(\ldots, Z, \ldots) \rightarrow X{=}Y$$

We show that this is not possible either: $X_i{=}Z \wedge Y{:}g(\ldots, Z, \ldots)$ may not belong to φ. We distinguish four cases depending on the positive and negative dominance relations between X and Y.

1. $X \lhd^* Y$ *in* φ:
   (a) $Y \lhd^* X$ *in* φ: This implies that X would have been labeled in φ which contradicts the assumption of the lemma.
   (b) $\neg Y \lhd^* X$ *in* φ: Because of (Dom) and (Trans), $Y \lhd^* X_i$ *in* φ. Saturation under (Trans) and (Clash4) implies $\neg X_i \lhd^* Y$ *in* φ. But this contradicts $X_i \in \mathsf{con}_\varphi(X)$, as Y could take the role of Z in the second line of the definition of $\mathsf{con}_\varphi(X)$.
2. $\neg X \lhd^* Y$ *in* φ:
   (a) $Y \lhd^* X$ *in* φ: We show that all immediate children of Y (i.e. Z and all of its siblings in the labeling constraint) do not dominate X; then we can apply the (Child) rule to conclude $X \lhd^* Y$ *in* φ, which contradicts $\neg X \lhd^* Y$ *in* φ, i.e. the clash-freeness of φ with respect to rule (Clash3).
   Clearly, $Z \lhd^* X$ *not in* φ; for otherwise, (Trans) would imply that $X_i \lhd^* X$ *in* φ, which contradicts the $\mathsf{con}_X(\varphi)$ condition for X_i. So let Z' be a sibling of Z in the labeling constraint above. If $Z' \lhd^* X$ *in* φ, then $Z' \lhd^* Z$ *in* φ (by the Trans rule); but by the (Disj) rule, $\neg Z' \lhd^* Z$ *in* φ, in contradiction to (Clash3).
   (b) $\neg Y \lhd^* X$ *in* φ: Saturation under (Trans) implies $X \lhd^* Z$ *in* φ, and saturation under (Dom) yields $Y \lhd^* Z$ *in* φ. This means that (Clash2) is applicable on φ, in contradiction to the clash-freeness of φ. □

Corollary 1 (Reduction to Simple Constraints). *Every saturated, clash-free constraint has a simple, saturated, clash-free extension.*

Proof. Let φ be saturated and clash-free. Without loss of generality, φ has a root variable (otherwise, we choose a fresh variable X and consider $\varphi \wedge \bigwedge\{X \lhd^* Y | Y \in V(\varphi)\}$ instead of φ).

By Lemma 3, we can successively label all variables in φ. The only problem is that the signature might not contain a function symbol for an arity we need; but we can get around this (artificial) problem by encoding these symbols with a nullary symbol and one symbol of arity ≥ 2, whose existence we have assumed. □

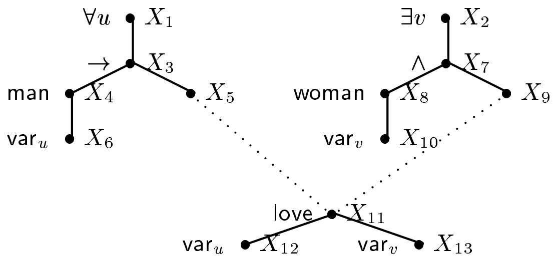


Fig. 1. An underspecified representation of *Every man loves a woman.*

3.3 A More Practical Solution Algorithm

The algorithm we have just presented is convenient from a theoretical perspective, but from a practical perspective, it's totally useless. Let's consider an example from scope underspecification for illustration. The constraint graph in Fig. 1 represents a dominance constraint describing the readings of the ambiguous sentence (1).

(1) *Every man loves a woman.*

This constraint has 14 variables; so the algorithm will consider 2^{196} or about 10^{59} alternatives in the guessing step, which of course is way too much to search through deterministically.
A more practically feasible satisfiability algorithm is inspired by *constraint programming* [KN99a]. We replace the nondeterministic guessing rule (Choice) by two *distribution* rules. The application strategy is to apply propagation and clash rules for as long as possible; only when no such rule is applicable, a single application of a distribution rule takes place.

$$\left.\begin{array}{llll}\text{(Refl)} & \text{(Trans)} & \text{(Decomp)} & \text{(Child)} \\ \text{(Disj)} & \text{(Dom)} & \text{(Parent)} & \end{array}\right\} \text{ Propagation rules from Sect. 3.1}$$

$$\left.\begin{array}{ll}\text{(Clash1)} & \text{(Clash2)} \\ \text{(Clash3)} & \text{(Clash4)}\end{array}\right\} \text{ Clash rules from Sect. 3.1}$$

$$\begin{array}{lll}\text{(Distr1)} & X \lhd^* Z \wedge Y \lhd^* Z \quad \rightarrow & X \lhd^* Y \textbf{ or } Y \lhd^* X \\ \text{(Distr2)} & X \lhd^* Y \wedge X{:}f(X_1, \ldots, X_n) \quad \rightarrow & Y \lhd^* X \textbf{ or } \bigvee_{i=1}^{n} X_i \lhd^* Y\end{array}$$

Proving soundness of the modified algorithm is a trivial extension of the original soundness proof. The completeness proof has exactly the same structure as the one above, but the details of the proofs of Lemmas 1 and 3 have to be changed.

On the example (1), the new algorithm will alternate between propagation steps and single applications of the (Distr1) rule. The first application of this rule will be to X_5 and X_9. The algorithm terminates after a total of three applications of (Distr1); this means that the search space has a size of at most eight.

As it stands, the second algorithm can be implemented with reasonable efficiency, but it's still not perfect for the application to scope underspecification, mainly because it is based on the wrong set of atomic constraints ($\triangleleft^*$ and $\neg\triangleleft^*$); using $\triangleleft^*$, inequality $\neq$, and disjointness $\perp$ (two nodes are disjoint if they don't dominate each other) would be better. A more advanced version of the second algorithm which takes this into account and runs even more efficiently is defined in [DN99].

Alternatively, an implementation of a dominance constraint solver can be based on *finite set constraints*. This was first suggested in [DG99] and is worked out e.g. in [KN99b]. [DN99] compares these two fundamental alternatives to implementing dominance constraint solvers in terms of runtimes and search spaces. An implementation of the set constraint based solver can be found on the WWW at `http://www.coli.uni-sb.de/cl/projects/chorus/demo.html`.

3.4 Larger logical languages

The algorithms we have defined so far solve dominance constraints that are pure conjunctions of labeling, dominance, and negated dominance constraints. We now extend them to allow in addition disjunctions and, later, negations in formulae over these atomic constraints. Positive occurrences of existential quantifiers can always be added, as they make no difference for satisfiability. The proofs for this section are simple and will be omitted.

In a nondeterministic setting, it is easy to deal with *disjunctions*; all we have to do is to go through the formula recursively and guess for each disjunction which disjunct can be satisfied. In this way, we produce a conjunction that we can feed into the original algorithm. It is easily shown that a formula φ of disjunctions and conjunctions over dominance constraints is satisfiable iff there is a choice of disjuncts that has a clash-free saturation.

The only difficulty in handling *negations* is to deal with negated labeling constraints, as a formula containing negations can clearly be reduced to an equivalent one where the only negations are single negations of atomic formulae, and we already know what to do with negated dominance constraints. We can get rid of negated labeling constraints as well by replacing them with satisfiability equivalent formulae that do not contain negated labeling constraints. If the signature is finite, we can replace a constraint $\neg X{:}f(X_1, \ldots, X_n)$ by

$$\left(\bigvee_{g \neq f \in \Sigma} X{:}g(X'_1, \ldots, X'_{\mathsf{ar}(g)}) \right) \;\vee\; \left(X{:}f(X''_1, \ldots, X''_n) \wedge \bigvee_{i=1}^{n} \neg X''_i {=} X_i \right),$$

where the X'_i and X''_i are fresh variables. Now all we need to show is that a negated labeling constraint φ is satisfied by a pair $(\mathcal{M}, \alpha)$ iff its encoding φ' is satisfied by $(\mathcal{M}, \alpha')$, where α' agrees with α on α's domain.

This construction does not work for infinite signatures since the first disjunction would become infinite. Except in pathological cases, however, negated labeling constraints can be eliminated in this case as well, at the price of additional case distinctions.

Taking all the results from this section together, we have shown:

Proposition 3. *The satisfiability problem of the positive existential fragment over dominance constraints (and, of course, all smaller languages) is in NP.*

4 NP-Hardness of Dominance Constraints

As we have just seen, the satisfiability problems of all languages over dominance constraints which are sublanguages of the positive existential fragment are in NP. In this section, we complement this result by showing that even for purely conjunctive dominance constraints, this problem is NP-hard. To this end, we reduce the 3SAT problem to the satisfiability problem of dominance constraints. We only sketch the proof, as the main construction is quite intuitive, and further details provide no further insight. Together with the result from the previous section, we obtain the following result:

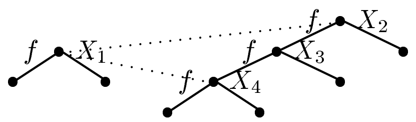
Theorem 1. *The satisfiability problems of all logical languages over dominance constraints between the (purely conjunctive) constraint language and the positive existential fragment are NP-complete.*

3SAT, a classical NP-hard problem, is the satisfiability problem of propositional formulae in conjunctive normal form where every conjunct is a disjunction of exactly three literals. This special type of conjunctive normal form is called 3-CNF.

$$\begin{array}{lr} \text{formulae} & \psi = C_1 \wedge \ldots \wedge C_m \\ \text{clauses} & C_i = L_{i1} \vee L_{i2} \vee L_{i3} \\ \text{literals} & L_{ij} = X_k \text{ or } \neg X_k. \end{array}$$

We assume that the variables that occur in ψ are $X_1, \ldots, X_n$.

The reduction is by encoding formulae in 3-CNF as satisfaction equivalent dominance constraints. The central problem that we must overcome is to model clauses without using disjunctions. We do this by using *dominance triangles*, subconstraints whose graphs look like this:



If $(\mathcal{M}^\tau, \alpha)$ is a solution of such a constraint, then α must map exactly one of the variables X_2, X_3, X_4 to the same node as X_1 because $\alpha(X_2)$ must be a prefix of $\alpha(X_1)$, which in turn must be a prefix of $\alpha(X_4)$. We can exploit this effect to model three-way disjunction – just what we need to encode a clause.

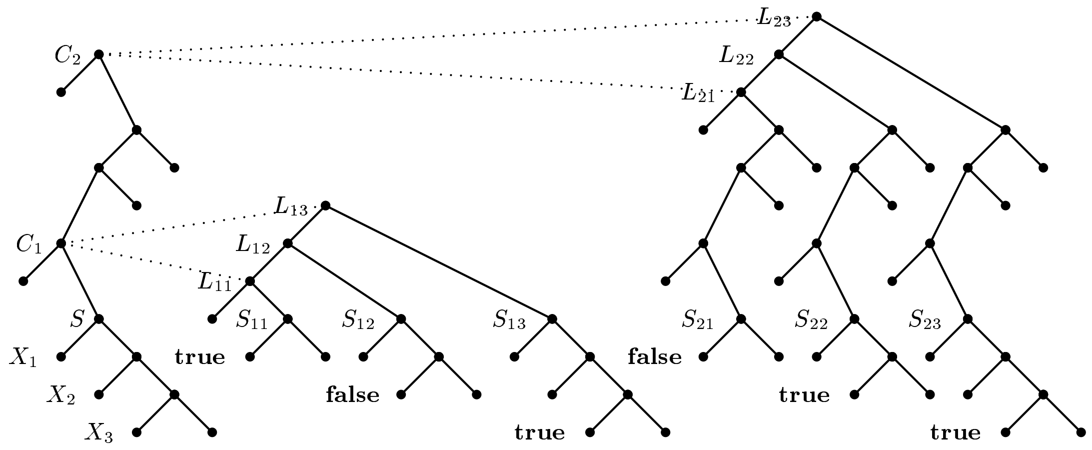


Fig. 2. An encoding of $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3)$ as a dominance constraint.

4.1 An Example of the Reduction

As an example of a 3-CNF formula, consider the following formula ψ:

(2) $(X_1 \vee \neg X_2 \vee X_3) \wedge (\neg X_1 \vee X_2 \vee X_3)$

The constraint graph in Fig. 2 represents the dominance constraint φ which is the encoding of ψ. We are drawing the constraint graph in a somewhat simplified manner by leaving away all labels of inner nodes and most variable names; all inner nodes should be read as being labeled with a fixed binary constructor f. The signature we use is $\{f{:}2, \mathbf{true}{:}0, \mathbf{false}{:}0\}$, but the proof remains valid if the signature contains more labels.

We claim that φ is satisfiable iff ψ is satisfiable. To understand this, let us take a closer look at the various parts of the diagram.

The lower left part of the graph (below the node S) holds a variable assignment: For each of the variables X_k that occur in ψ, there is one node. In a solution, each of these nodes must be labeled with either **true** or **false**, but not both.

We can view ψ as a constraint on admissibility of variable assignments by calling a variable assignment admissible if it satisfies ψ. Each clause imposes such a restriction on the variable assignments; within a clause, we have a choice between three different options for satisfying the constraint.

The dominance constraint expresses the very same thing.

Because it is part of a dominance triangle, C_1 must be identified with one of the L_{1j} in any solution. But once we have identified C_1 with one of the three L_{1j}, we have decided which of the clause C_1's literals we want to satisfy: The right daughter of the chosen L_{1j} node is identified with S, some entries in the variable assignment subtree may be skipped, and then a value restriction is imposed on one of the variables X_k. In the example, L_{11} forces the label of X_1 to be **true**; L_{12} forces the label of X_2 to be **false**; etc. We have imposed a constraint on the variable assignment that is obviously equivalent to that imposed by the first clause in ψ.

The second clause is represented in a similar way: The dominance triangle between L_{21}, C_2, and L_{23} allows a choice which literal of this clause we want to satisfy. Whichever literal we pick, its right daughter first skips the entry for C_1 (identifying S with one of the S_{2j}), and then it selects a variable entry and imposes a value constraint.

An important detail of this encoding is the presence of more nodes than just the C_i in the main branch of the graph (for example, there are two additional nodes between C_1 and C_2 in the constraint graph). These nodes are "rubbish dumps" which can be used to store unneeded material in such a way that it won't interfere with anything else. Suppose we identified C_1 and L_{12} in a solution of φ. Then L_{11} will be identified with the left daughter of C_1, and L_{13} will be identified with the mother of C_1. Clearly, we do not want any other part of the constraint to say anything about the right child of C_1's mother because otherwise, we might run into unnecessarily unsatisfiable dominance constraints. This means that above each C_i node, we need two additional nodes to drop material from the identification process. We do not need any additional nodes below the C_i because the unnecessary material is then a left child of the selected literal node and can safely be stored below C_i's left daughter.

4.2 The Reduction in the General Case

Now that we have made the intuition clear, we define the encoding in a more systematic way.

We build the constraint graph that corresponds to φ from the "building blocks" in Fig. 3. Larger building blocks can include several copies of smaller building blocks. For most of the building blocks, we have specified with arrows an upper and a lower attachment site where it can be composed with other blocks by identifying the two attachment sites; we write such compositions as trees whose labels are the two building blocks. Furthermore, we take a block with a superscript s (such as Skip with superscript $i-1$ in X_i) to mean s-fold composition of building blocks. So we want the X_i block to consist of $i-1$ occurrences of Skip blocks and two additional nodes that are immediate children of the lowest attachment site in the sequence of Skips, the left of which is labeled with **true**.

It is easy to see that the constraint graph from the previous section was built according to this scheme. The overall structure consists of m entries for the clauses, below which n Skip blocks hold a variable assignment. Within each

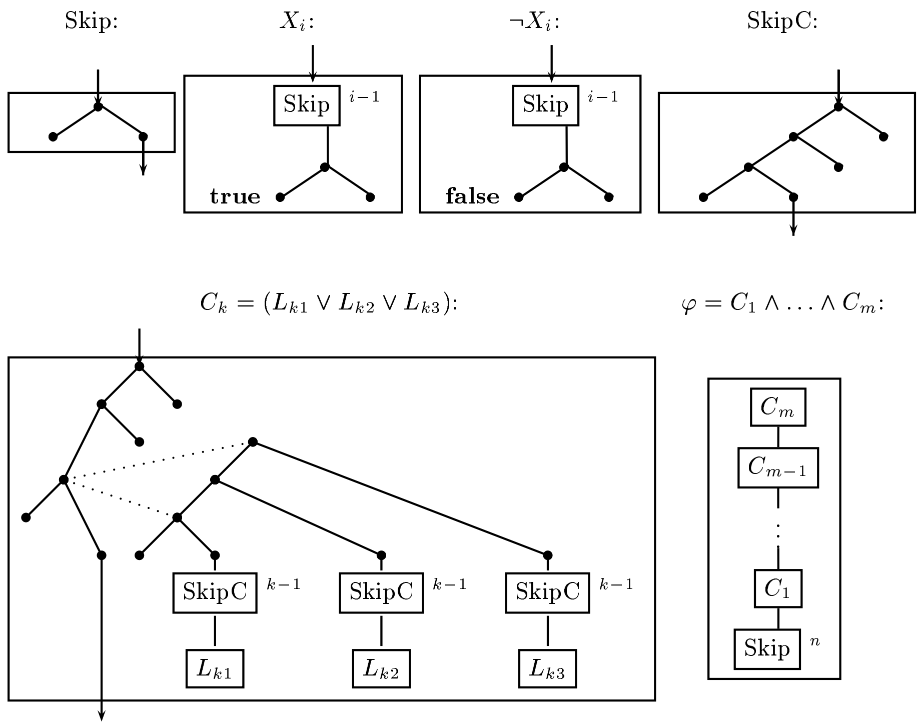


Fig. 3. Building blocks for the encoding of 3SAT as a dominance constraint.

C_i block, there is a dominance triangle that allows the selection of a literal, together with a sufficient number of SkipC blocks to skip lower clauses. Finally, the encoding of a literal selects a propositional variable and imposes a value restriction.

The intuitive explanation from the beginning of this section should make clear that this encoding is correct. For a formal proof, we can encode valuations satisfying a 3-CNF formula ψ as tree structures and variable assignments satisfying the encoding of ψ, and vice versa. The gory details of this construction can be found in [Kol99].

5 The First-Order Theory of Dominance Constraints

In the sections so far, we have focused on propositional languages over dominance constraints. Now we allow all propositional and first-order connectives (over the same set of atomic constraints) and consider the validity problem of first-order formulae Φ over dominance constraints. We first show a direct proof of the decidability of this problem for the case of bounded arity by reduction to second-order monadic logic. Afterwards, we show that the problem has non-elementary complexity by reducing the equivalence problem of regular languages with complement to it. Both results hold true for validity both over finite and over arbitrary trees.

5.1 Reduction to Second-Order Monadic Logic

In this section we assume a (finite or infinite) signature with a bound n on the arities of the function symbols. We show how to reduce the theory of dominance constraints to $S(n+1)S$, the second-order monadic theory of $n+1$ successor functions. The reduction carries over verbatim to $WS(n+1)S$, so finiteness of the trees doesn't make a difference.

The language of $S(n+1)S$ contains first-order variables $x, y, z, \dots$ and second-order monadic variables (i.e., variables denoting sets) $X, Y, Z, \dots$, a binary relation symbol $\in$, a constant ϵ, and for every $0 \leq i \leq n$, a unary function symbol i. The universe of the corresponding structure is the set $\{0, \dots, n\}^*$ of words over the alphabet $\{0, \dots, n\}$, where ϵ denotes the empty word, and a function symbol i is interpreted as $i(w) = wi$. The formula $x \in X$ is true iff the denotation of x is contained in the denotation of X. The *theory* $S(n+1)S$ is the set of all closed formulae valid in this structure. The theory $WS(n+1)S$ is defined by restricting the denotation of monadic second-order variables to *finite* sets. The decidability of these theories has been established in [TW68] for the case of $WS(n+1)S$ and [Rab69] for the case of $S(n+1)S$. A function application $i(x)$ is usually written as $x.i$.

We encode tree structures as the denotations of set variables X. Both are sets of words, and we can easily express closure under prefix and left brother; the only challenge is how to encode labels and arities. Below, we will first write down an $S(n+1)S$ formula that characterizes (encodings of) tree structures. Once this is done, we can easily encode dominance and labeling constraints.

We assume that the function symbols of a given arity k are numbered $1, 2, \dots$ if there are infinitely many symbols of arity k and $1, \dots, a_k$ if there are only finitely many. If there are infinitely many function symbols of arity k, we write $a_k = \infty$.

We encode a tree τ as the following finite subset of $\{0, \dots, n\}^*$:

$$T_\tau = \{\pi 0^i \mid \pi \in D_\tau, i \leq \tau(\pi)\}$$

In words, we represent τ as a set T_τ of words in $S(n+1)S$ by first requiring that all words in the tree domain of τ are also in T_τ. Then we add 0-successors to signify the label: The label of the node π of τ is represented by the arity of the node π together with the length of the string of 0's attached to π in T_τ. That is, instead of requiring that the label should determine the arity of a node, we only have to make sure that the label which is indicated by the arity and the zeroes really exists (i.e. there are at most a_k zeroes).

We encode a closed first-order formula Φ over dominance constraints as a closed monadic second-order formula containing a second-order variable X which denotes the encoding T_τ of a tree model τ. To this end, we first axiomatize the general structure of an T_τ:

$$
\begin{aligned}
tree(X) = {} & \forall x \bigwedge_{i=0,\dots,n} (x.i \in X \rightarrow x \in X) && (3)\\
& \wedge\ \forall x \bigwedge_{i=2,\dots,n} (x.i \in X \rightarrow x.(i-1) \in X) && (4)\\
& \wedge\ \forall x((x \in X \wedge \neg\exists y\, x = y.0) \rightarrow x.0 \in X) && (5)\\
& \wedge\ \forall x(\bigwedge_{i=1,\dots,n} x.0.i \notin X) && (6)\\
& \wedge\ \forall x(x.1 \notin X \rightarrow x.0^{a_0+1} \notin X) && (7)\\
& \wedge\ \forall x \bigwedge_{\substack{i=1,\dots,n\\ a_i<\infty}} (x.i \in X \wedge x.(i+1) \notin X \rightarrow x.0^{a_i+1} \notin X) && (8)
\end{aligned}
$$

(In this formula, the expression $x.(n+1) \notin X$ should be read as **true**.)
The formula first says that X is a tree, that is, prefixed-closed (formula 3) and closed under left brother (formula 4). By formula 5, every "proper" tree node, that is, a word not ending on 0, has to be labeled. Formula 6, together with 3, ensures that if $x0w \in X$, then $w \in \{0\}^*$. Finally, the consistency of the label of a node with its number of children (in the sense discussed above) is expressed by formula 7 for the case of nullary function symbols, and by formula 8 for non-nullary function symbols.
To complete our translation, we need another auxiliary predicate, $treenode(x)$, which expresses that x is a "proper tree node" (i.e. not part of the encoding of a label). More exactly, under the assumption that $tree(X)$ holds, we will get that $treenode(x)$ holds exactly when $x \in X$ and $x \in \{1,\dots,n\}^*$:

$$treenode(x) \quad := \quad x \in X \wedge \neg\exists y\, x = y.0$$

The prefix predicate is expressed in $S(n+1)S$ as usual:

$$prefix(x,y) \quad := \quad \forall X\Big((y \in X \wedge \bigwedge_{i=0,\dots,n} (\forall z.z.i \in X \rightarrow z \in X)) \rightarrow x \in X\Big)$$

Finally, we encode an arbitrary first-order formula Φ over the dominance constraints as an $S(n+1)S$-formula Φ_X by

1. relativizing all quantifiers by the predicate $treenode(\cdot)$;
2. replacing $x \triangleleft^* y$ by $prefix(x,y)$;
3. replacing $x{:}f(x_1,\dots,x_n)$ by $label_f(x,x_1,\dots,x_n)$,

where we define

$$label_f(x,x_1,\dots,x_n) = \bigwedge_{i=1,\dots,n} x_i = x.i \wedge x.0^f \in X \wedge x.0^{f+1} \notin X \wedge x.(n+1) \notin X$$

Theorem 2. *A first-order formula Φ over dominance constraints is valid in the class of all (resp. finite) tree structures iff the following formula is valid in $S(n+1)S$ (resp. $WS(n+1)S$):*

$$\forall X(tree(X) \rightarrow \Phi_X)$$

Using the signature consisting of a constant a and a binary f, the following formula is valid over finite trees but not valid over infinite trees:

$$\forall X, Y, Z(X{:}f(Y, Z) \rightarrow \exists V(V{:}a \wedge X \triangleleft^* V))$$

Correspondingly, the following $S3S$-formula is valid in $WS3S$ but not in $S3S$:

$$\forall X(tree(X) \rightarrow \forall x, y, z(treenode(x) \wedge treenode(y) \wedge treenode(z) \wedge \\ label_f(x, y, z) \rightarrow \exists v(treenode(v) \wedge label_a(v) \wedge prefix(x, v))))$$

Corollary 2. *The theory of first-order formulae over dominance constraints with a signature of bounded arity is decidable.*

5.2 The First-Order Theory is Non-Elementary

We recall that a problem has *non-elementary complexity* if there is no algorithm for it running in time bounded by $exp_k(n)$ for any k, where $exp_0(n) = n$ and $exp_{k+1}(n) = 2^{exp_k(n)}$ (see, for instance, [Joh90] for a survey).

Theorem 3. *The first-order theory of dominance constraints has non-elementary complexity, both in the case of the finite tree models and in the case of arbitrary tree models.*

Recall that we have assumed the signature to contain at least a constant and a binary function symbol. We show this theorem by a reduction of the following classical problem:

Theorem 4 (Stockmeyer and Meyer, [SM73]). *The problem whether two regular expressions formed with $1, 2$, concatenation, union and complement (interpreted with respect to $\{1, 2\}^*$) denote the same set is non-elementary.*

We define our syntax of regular expressions as

$$R \ ::= \ 1 \ | \ 2 \ | \ R \cup R \ | \ RR \ | \ \neg R$$

The language defined by the regular expression R is called $\mathcal{L}_R$.
Given two variables X and Y, we translate a regular expression R of this class into a formula $\varphi_{[R]}(X, Y)$ with free variables $\{X, Y\}$. Roughly, $\varphi_{[R]}(X, Y)$ expresses that in any of its solutions, X dominates Y, and the path between the two nodes is in $\mathcal{L}_R$.

$$\varphi_{[1]}(X,Y) = \exists Z(X{:}f(Y,Z))$$
$$\varphi_{[2]}(X,Y) = \exists Z(X{:}f(Z,Y))$$
$$\varphi_{[R_1 \cup R_2]}(X,Y) = \varphi_{[R_1]}(X,Y) \vee \varphi_{[R_2]}(X,Y)$$
$$\varphi_{[R_1 R_2]}(X,Y) = \exists Z(\varphi_{[R_1]}(X,Z) \wedge \varphi_{[R_2]}(Z,Y))$$
$$\varphi_{[\neg R]}(X,Y) = X \triangleleft^* Y \wedge \neg \varphi_{[R]}(X,Y)$$

The following formula says that a tree is (an approximation of) the full binary tree. More precisely, if τ is a tree satisfying Bin, all of its inner nodes must be labeled with f and all of its leaves must be labeled with a.

$$Bin = \forall X\,(X{:}a \vee \exists Y \exists Z\ X{:}f(Y,Z))$$

Proposition 4. *For any tree structure $\mathcal{M}$ and variable assignment α,*

$$M, \alpha \models Bin \wedge \varphi_{[R]}(X,Y) \textit{ iff exists } v \in \mathcal{L}_R : \alpha(Y) = \alpha(X) \cdot v.$$

Proof. This is proven by induction on R. The only non-trivial case is the complement. Let $(\mathcal{M}, \alpha)$ satisfy Bin. Then

$M, \alpha \models \varphi_{[\neg R]}(X,Y)$
$\Leftrightarrow M, \alpha \models X \triangleleft^* Y \wedge \neg \varphi_{[R]}(X,Y)$ by definition
$\Leftrightarrow \alpha(X) \triangleleft^* \alpha(Y)$ and $\neg \exists v \in \mathcal{L}_R\ \alpha(Y) = \alpha(X) \cdot v$ by induction
$\Leftrightarrow \exists v \in \mathcal{L}_{[\neg R]} \alpha(Y) = \alpha(X) \cdot v$ $(*)$

The step $(*)$ is justified by the fact that in a solution of Bin, $\alpha(X) \triangleleft^* \alpha(Y)$ iff there is a word $v \in \{1,2\}^*$ with $\alpha(Y) = \alpha(X) \cdot v$, and that such a v is *unique*. □

We can finally reduce the equivalence problem of regular expressions to the theory of dominance constraints:

Lemma 4. *Let R_1 and R_2 be regular expressions, $\mathcal{I}$ the class of infinite tree models and $\mathcal{F}$ the class of finite tree models. Then we have that*

$$\mathcal{L}_{R_1} = \mathcal{L}_{R_2} \tag{9}$$
$$\Leftrightarrow \mathcal{I} \models Bin \rightarrow (\forall X \forall Y(\varphi_{[R_1]}(X,Y) \leftrightarrow \varphi_{[R_2]}(X,Y)) \tag{10}$$
$$\Leftrightarrow \mathcal{F} \models Bin \rightarrow (\forall X \forall Y(\varphi_{[R_1]}(X,Y) \leftrightarrow \varphi_{[R_2]}(X,Y) \tag{11}$$

Proof. $\underline{(9) \Rightarrow (10)}$: If $\mathcal{L}_{R_1} = \mathcal{L}_{R_2}$ then, by Proposition 4, the formulae $\varphi_{[R_1]}$ and $\varphi_{[R_2]}$ are equivalent in any model of Bin.
$\underline{(10) \Rightarrow (11)}$: Immediate since $\mathcal{F} \subseteq \mathcal{I}$.
$\underline{(\neg 9) \Rightarrow (\neg 11)}$: Let $v \in \mathcal{L}_{R_1} - \mathcal{L}_{R2}$, and let τ be the binary tree of depth $length(v)$; that is, all nodes of depth $< length(v)$ are labeled with f, and all nodes of depth $length(v)$ are labeled with a. $\mathcal{M}^\tau$, together with the variable assignment $\{X \mapsto \epsilon, Y \mapsto v\}$, satisfies $\varphi_{[R_1]}$ but not $\varphi_{[R_2]}$, as a consequence of Proposition 4. □

6 Conclusion

In this paper, we have analyzed the complexity of various logical languages over dominance constraints. We have shown that the satisfiability problems of all languages between the purely conjunctive constraint language and the positive existential fragment are NP-complete. We have presented two algorithms for solving dominance constraints, the second of which has been implemented and works well for real-world examples. Finally, we have presented a new proof of the decidability of the first-order theory of dominance constraints with signatures of bounded arity by encoding it into $(W)SnS$, and we have shown that its complexity is non-elementary.

From a practical perspective, the most important of the logical languages over dominance constraints we have considered here is the purely conjunctive constraint language. Despite the NP-hardness result we have derived for this language, there are implementations that decide satisfiability very efficiently for constraints from scope underspecification. These implementations are either advanced variants of the second algorithm presented here or based on finite set constraints; either way, they follow the "propagate and distribute" strategy advocated by constraint programming.

The observation that the general intractability does not affect the behaviour of actual implementations suggests that the linguistically relevant dominance constraints all belong to a subclass with an easier satisfiability problem, and that our algorithm exploits this automatically. The NP-hardness proof in this paper can be invalidated by only considering *normal* constraints, which contain an *inequality constraint* between any two variables for which they contain a labeling constraint; but inequality constraints can be used to build another NP-hardness proof. So an exact characterization of this subclass is an open problem.

Acknowledgments. We would like to thank Manuel Bodirsky for implementing dominance constraints in Oz and Denys Duchier for suggesting to do so via finite set constraints. It is a pleasure to thank all members (student or not) of the CHORUS project. The research reported here was supported by the SFB 378 at the Universität des Saarlandes and the Esprit Working Group CCL II (EP 22457).

7 References

[Bos96] Johan Bos. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*, pages 133–143, 1996.

[BRVS95] 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.

[DG99] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In *Proceedings of IWCS-3*, Tilburg, 1999.

[DN99] Denys Duchier and Joachim Niehren. Solving dominance constraints with finite set constraint programming. Technical report, Universität des Saarlandes, Programming Systems Lab, 1999. `http://www.ps.uni-sb.de/Papers/abstracts/DomCP99.html`.

[ENRX98] M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over Lambda-Structures in Semantic Underspecification. In *Proceedings COLING/ACL'98*, Montreal, 1998.

[GW98] Claire Gardent and Bonnie Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998. University of Pennsylvania.

[Joh90] David S. Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, vol A: Algorithms and Complexity*, chapter 2, pages 67–161. Elsevier, 1990.

[KN99a] Alexander Koller and Joachim Niehren. Constraint programming in computational linguistics. Submitted. `http://www.coli.uni-sb.de/~koller/cpcl.html`, 1999.

[KN99b] Alexander Koller and Joachim Niehren. Scope underspecification and processing. Lecture Notes, ESSLLI '99, Utrecht, 1999. `http://www.coli.uni-sb.de/~koller/papers/esslli99.html`.

[Kol99] Alexander Koller. Constraint languages for semantic underspecification. Diplom thesis, Universität des Saarlandes, Saarbrücken, Germany, 1999. `http://www.coli.uni-sb.de/~koller/papers/da.html`.

[MHF83] 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.

[Mus98] Reinhard Muskens. Underspecified semantics. Technical Report 95, Universität des Saarlandes, Saarbrücken, 1998. To appear.

[MVK98] Wilfried Meyer-Viol and Ruth Kempson. Sequential construction of logical forms. In *Proceedings of the Third Conference on Logical Aspects of Computational Linguistics*, Grenoble, France, 1998.

[Rab69] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Rey93] Uwe Reyle. Dealing with ambiguities by underspecification: construction, representation, and deduction. *Journal of Semantics*, 10:123–179, 1993.

[Rog94] James Rogers. *Studies in the Logic of Trees with Applications to Grammar Formalisms.* PhD thesis, University of Delaware, 1994.

[RVS92] James Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Proceedings of the 30th ACL*, pages 72–80, University of Delaware, 1992.

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *5th Annual ACM Symposium on the Theory of Computing*, pages 1–9, 1973.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2:57–68, 1968.

[VS92] K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.