A Polynomial-Time Fragment of Dominance Constraints

Alexander Koller koller@coli.uni-sb.de
Kurt Mehlhorn* mehlhorn@mpi-sb.mpg.de
Joachim Niehren niehren@ps.uni-sb.de
University of the Saarland / *Max-Planck-Institute for Computer Science
Saarbrücken, Germany

Abstract

Dominance constraints are logical descriptions of trees that are widely used in computational linguistics. Their general satisfiability problem is known to be NP-complete. Here we identify the natural fragment of *normal* dominance constraints and show that its satisfiability problem is in deterministic polynomial time.

1 Introduction

Dominance constraints are used as partial descriptions of trees in problems throughout computational linguistics. They have been applied to incremental parsing (Marcus et al., 1983), grammar formalisms (Vijay-Shanker, 1992; Rambow et al., 1995; Duchier and Thater, 1999; Perrier, 2000), discourse (Gardent and Webber, 1998), and scope underspecification (Muskens, 1995; Egg et al., 1998).

Logical properties of dominance constraints have been studied e.g. in (Backofen et al., 1995), and computational properties have been addressed in (Rogers and Vijay-Shanker, 1994; Duchier and Gardent, 1999). Here, the two most important operations are *satisfiability testing* – does the constraint describe a tree? – and *enumerating solutions*, i.e. the described trees. Unfortunately, even the satisfiability problem has been shown to be NP-complete (Koller et al., 1998). This has shed doubt on their practical usefulness.

In this paper, we define *normal* dominance constraints, a natural fragment of dominance constraints whose restrictions should be unproblematic for many applications. We present a graph algorithm that decides satisfiability of normal dominance constraints in polynomial time. Then we show how to use this algorithm to enumerate solutions efficiently.

An example for an application of normal dominance constraints is scope underspecification: Constraints as in Fig. 1 can serve as underspecified descriptions of the semantic readings of sentences such as (1), considered as the structural trees of the first-order representations. The dotted lines signify *dominance relations*, which require the upper node to be an ancestor of the lower one in any tree that fits the description.

(1) Some representative of every department in all companies saw a sample of each product.

The sentence has 42 readings (Hobbs and Shieber, 1987), and it is easy to imagine how the number of readings grows exponentially (or worse) in the length of the sentence. Efficient enumeration of readings from the description is a longstanding problem in scope underspecification. Our polynomial algorithm solves this problem. Moreover, the investigation of graph problems that are closely related to normal constraints allows us to prove that many other underspecification formalisms – e.g. Minimal Recursion Semantics (Copestake et al., 1997) and Hole Semantics (Bos, 1996) – have NP-hard satisfiability problems. Our algorithm can still be used as a preprocessing step for these approaches; in fact, experience shows that it seems to solve all encodings of descriptions in Hole Semantics that actually occur.

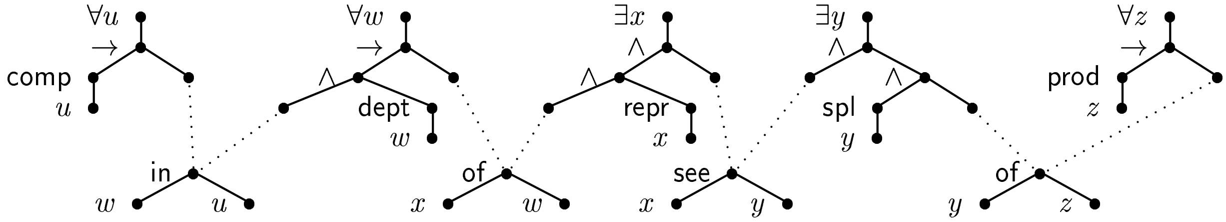

Fig. 1: A dominance constraint (from scope underspecification).

2 Dominance Constraints

In this section, we define the syntax and semantics of dominance constraints. The variant of dominance constraints we employ describes constructor trees – ground terms over a signature of function symbols – rather than feature trees.

So we assume a signature Σ function symbols ranged over by $f, g, \ldots$, each of which is equipped with an arity $\mathsf{ar}(f) \geq 0$. Constants – function symbols of arity 0 – are ranged over by a, b. We assume that Σ contains at least one constant and one symbol of arity at least 2. Finally, let Vars be an infinite set of variables ranged over by X, Y, Z. The variables will denote nodes of a constructor tree. We will consider constructor trees as directed labeled graphs; for instance, the ground term $f(g(a, a))$ can be seen as the graph in Fig. 2.

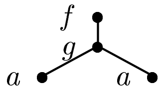

Fig. 2: $f(g(a, a))$

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

A *(finite) constructor tree* τ is a pair (T, L) consisting of a tree $T = (V, E)$, a *node labeling* $L : V \to \Sigma$, and an *edge labeling* $L : E \to \mathbb{N}$, 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 $L((u, v)) = k$.[1] We draw constructor trees as in Fig. 2, by annotating nodes with their labels and ordering the edges along their labels from left to right. If $\tau = ((V, E), L)$, we write $V_\tau = V$, $E_\tau = E$, $L_\tau = L$. Now we are ready to define tree structures, the models of dominance constraints:

[1]The symbol L is overloaded to serve both as a node and an edge labeling.

Definition 2.1. *The* tree structure $\mathcal{M}^\tau$ *of a constructor tree* τ *is a first-order structure with domain* V_τ *which provides the* dominance relation $\triangleleft^{*\tau}$ *and a* labeling relation *for each function symbol* $f \in \Sigma$.

Let $u, v, v_1, \ldots v_n \in V_\tau$ *be nodes of* τ. *The dominance relationship* $u \triangleleft^{*\tau} v$ *holds iff there is a path from* u *to* v *in* E_τ*; the labeling relationship* $u{:}f^\tau(v_1, \ldots, v_n)$ *holds iff* u *is labeled by the* n*-ary symbol* f *and has the children* $v_1, \ldots, v_n$ *in this order; that is,* $L_\tau(u) = f$, $\mathsf{ar}(f) = n$, $\{(u, v_1), \ldots, (u, v_n)\} \subseteq E_\tau$, *and* $L_\tau((u, v_i)) = i$ *for all* $1 \leq i \leq n$.

A *dominance constraint* φ is a conjunction of *dominance*, *inequality*, and *labeling* literals of the following form where $\mathsf{ar}(f) = n$:

$$\begin{array}{rcl} \varphi & ::= & \varphi \wedge \varphi' \mid X \triangleleft^* Y \mid X \neq Y \\ & \mid & X{:}f(X_1, \ldots, X_n) \end{array}$$

Let $\mathsf{Var}(\varphi)$ be the set of variables of φ. A pair of a tree structure $\mathcal{M}^\tau$ and a *variable assignment* $\alpha : \mathsf{Var}(\varphi) \to V_\tau$ *satisfies* φ iff it satisfies each literal in the obvious way. We say that $(\mathcal{M}^\tau, \alpha)$ is a *solution* of φ in this case; φ is *satisfiable* if it has a solution.

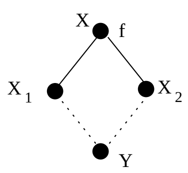

Fig. 3: An unsatisfiable constraint

We usually draw dominance constraints as *constraint graphs*. For instance, the constraint graph for $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge$

$X_2 \triangleleft^* Y$ is shown in Fig. 3. As for trees, we annotate node labels to nodes and order tree edges from left to right; dominance edges are drawn dotted. The example happens to be unsatisfiable because trees cannot branch upwards.

Definition 2.2. *Let φ be a dominance constraint that does not contain two labeling constraints for the same variable.*[2] *Then the* constraint graph *for φ is a directed labeled graph $G(\varphi) = (\mathsf{Var}(\varphi), E, L)$. It contains a* (partial) node labeling $L : \mathsf{Var}(\varphi) \rightsquigarrow \Sigma$ *and an* edge labeling $L : E \to \mathbb{N} \cup \{\triangleleft^*\}$.

The sets of edges E and labels L of the graph $G(\varphi)$ are defined in dependence of the literals in φ: The labeling literal $X{:}f(X_1, \ldots, X_n)$ belongs to φ iff $L(X) = f$ and for each $1 \leq i \leq n$, $(X, X_i) \in E$ and $L((X, X_i)) = i$. The dominance literal $X \triangleleft^* Y$ is in φ iff $(X, Y) \in E$ and $L((X, Y)) = \triangleleft^*$.

Note that inequalities in constraints are not represented by the corresponding constraint graph. We define (solid) *fragments* of a constraint graph to be maximal sets of nodes that are connected over tree edges.

3 Normal Dominance Constraints

Satisfiability of dominance constraints can be decided easily in non-deterministic polynomial time; in fact, it is NP-complete (Koller et al., 1998).

The NP-hardness proof relies on the fact that solid fragments can "overlap" properly. For illustration, consider the constraint $X{:}f(X_1, X_2) \wedge Y{:}f(Y_1, Y_2) \wedge Y \triangleleft^* X \wedge X \triangleleft^* Y_1$, whose constraint graph is shown in Fig. 4. In a solution of this constraint, either Y or Y_1 must be mapped to the same node as X; if $X = Y$, the two fragments *overlap properly.* In the applications in computational linguistics, we typically don't want proper overlap; X should never be identified with Y, only with Y_1. The subclass of dominance constraints that excludes proper overlap (and fixes some minor inconveniences) is the class of *normal* dominance constraints.

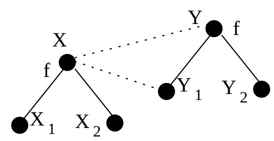

Fig. 4: Overlap

Definition 3.1. *A dominance constraint φ is called* normal *iff for all variables $X, Y, Z \in \mathsf{Var}(\varphi)$,*

1. *$X \neq Y$ in φ iff both $X{:}f(\ldots)$ and $Y{:}g(\ldots)$ in φ, where f and g may be equal* (no overlap)*;*[3]

2. *X only appears once as a parent and once as a child in a labeling literal* (tree-shaped fragments)*;*

3. *if $X \triangleleft^* Y$ in φ, neither $X{:}f(\ldots)$ nor $Z{:}f(\ldots Y \ldots)$ are* (dominances go from holes to roots)*;*

4. *if $X \triangleleft^* Y$ in φ, then there are Z, f such that $Z{:}f(\ldots X \ldots)$ in φ* (no empty fragments)*.*

Fragments of normal constraints are tree-shaped, so they have a unique *root* and *leaves.* We call unlabeled leaves *holes.* If X is a variable, we can define $R_\varphi(X)$ to be the root of the fragment containing X. Note that by Condition 1 of the definition, the constraint graph specifies all the inequality literals in a normal constraint. All constraint graphs in the rest of the paper will represent normal constraints.

The main result of this paper, which we prove in Section 4, is that the restriction to normal constraints indeed makes satisfiability polynomial:

Theorem 3.2. *Satisfiability of normal dominance constraints is $O((k+1)^3 n^2 \log n)$, where n is the number of variables in the constraint, and k is the maximum number of dominance edges into the same node in the constraint graph.*

In the applications, k will be small – in scope underspecification, for instance, it is

[2] Every constraint can be brought into this form by introducing auxiliary variables and expressing $X{=}Y$ as $X \triangleleft^* Y \wedge Y \triangleleft^* X$.

[3] Allowing more inequality literals does not make satisfiability harder, but the pathological case $X \neq X$ invalidates the simple graph-theoretical characterizations below.

bounded by the maximum number of arguments a verb can take in the language if we disregard VP modification. So we can say that satisfiability of the linguistically relevant dominance constraints is $O(n^2 \log n)$.

4 A Polynomial Satisfiability Test

Now we derive the satisfiability algorithm that proves Theorem 3.2 and prove it correct. In Section 5, we embed it into an enumeration algorithm. An alternative proof of Theorem 3.2 is by reduction to a graph problem discussed in (Althaus et al., 2000); this more indirect approach is sketched in Section 6.

Throughout this section and the next, we will employ the following non-deterministic choice rule (Distr), where X, Y are different variables.

$$\begin{array}{lll} \text{(Distr)} & & \varphi \wedge X \triangleleft^* Z \wedge Y \triangleleft^* Z \\ & \rightarrow & \varphi \wedge X \triangleleft^* R_\varphi(Y) \wedge Y \triangleleft^* Z \\ & \vee & \varphi \wedge Y \triangleleft^* R_\varphi(X) \wedge X \triangleleft^* Z \end{array}$$

In each application, we can pick one of the disjuncts on the right-hand side. For instance, we get Fig. 5b by choosing the second disjunct in a rule application to Fig. 5a.

The rule is sound if the left-hand side is normal: $X \triangleleft^* Z \wedge Y \triangleleft^* Z$ entails $X \triangleleft^* Y \vee Y \triangleleft^* X$, which entails the right-hand side disjunction because of conditions 1, 2, 4 of normality and $X \neq Y$. Furthermore, it preserves normality: If the left-hand side is normal, so are both possible results.

We define a normal dominance constraint φ to be in *solved form* iff (Distr) is not applicable to φ and $G(\varphi)$ is cycle-free. Every constraint in solved form is satisfiable.

4.1 Characterizing Satisfiability

In a first step, we characterize the unsatisfiability of a normal constraint by the existence of certain cycles in the undirected version of its graph (Proposition 4.3). Recall that a cycle in a graph is *simple* if it does not contain the same node twice.

Definition 4.1. *A cycle in an undirected constraint graph is called* hypernormal *if it does not contain two adjacent dominance edges that emanate from the same node.*

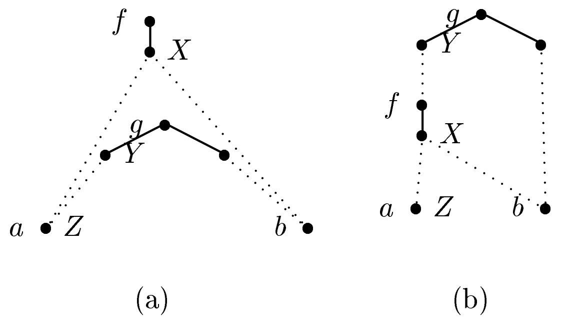

Fig. 5: (a) A constraint that entails $X \triangleleft^* Y$, and (b) the result of trying to arrange Y above X. The cycle in (b) is hypernormal, the one in (a) is not.

For instance, the cycle in the left-hand graph in Fig. 5 is *not* hypernormal, whereas the cycle in the right-hand one is.

Lemma 4.2. *A normal dominance constraint whose undirected graph has a simple hypernormal cycle is unsatisfiable.*

Proof. Let φ be a normal dominance constraint whose undirected graph contains a simple hypernormal cycle. Assume first that it contains a simple hypernormal cycle C that is also a cycle in the directed graph. There is at least one leaf of a fragment on C; let Y be such a leaf. Because φ is normal, Y has a mother X via a tree edge, and X is on C as well. That is, X must dominate Y but is properly dominated by Y in any solution of φ, so φ is unsatisfiable.

In particular, if an undirected constraint graph has a simple hypernormal cycle C with only one dominance edge, C is also a directed cycle, so the constraint is unsatisfiable. Now we can continue inductively. Let φ be a constraint with an undirected simple hypernormal cycle C of length l, and suppose we know that all constraints with cycles of length less than l are unsatisfiable. If C is a directed cycle, we are done (see above); otherwise, the edges in C must change directions somewhere. Because φ is normal, this means that there must be a node Z that has two incoming dominance edges $(X, Z), (Y, Z)$ which are adjacent edges in C. If X and Y are in the same fragment, φ is trivially unsatisfiable. Otherwise, let φ_1 and φ_2 be the two constraints obtained from φ by one application of (Distr) to X, Y, Z. Let C_1 be the sequence of edges we obtain from C by replacing the path from X to $R_\varphi(Y)$ via Z by the edge $(X, R_\varphi(Y))$. C is hypernormal and simple, so no two dominance edges in C emanate from the same node; hence, the new edge is the only dominance edge in C_1 emanating from X, and C_1 is a hypernormal cycle in the undirected graph of φ_1. C_1 is still simple, as we have only removed nodes. But the length of C_1 is strictly less than l, so φ_1 is unsatisfiable by induction hypothesis. An analogous argument shows unsatisfiability of φ_2. But because (Distr) is sound, this means that φ is unsatisfiable too. $\square$

Proposition 4.3. *A normal dominance constraint is satisfiable iff its undirected constraint graph has no simple hypernormal cycle.*

Proof. The direction that a normal constraint with a simple hypernormal cycle is unsatisfiable is shown in Lemma 4.2.

For the converse, we first define an ordering $\varphi_1 \leq \varphi_2$ on normal dominance constraints: it holds if both constraints have the same variables, labeling and inequality literals, and if the reachability relation of $G(\varphi_1)$ is a subset of that of $G(\varphi_2)$. If the subset inclusion is proper, we write $\varphi_1 < \varphi_2$. We call a constraint φ *irredundant* if there is no normal constraint φ' with fewer dominance literals but $\varphi \leq \varphi'$. If φ is irredundant and $G(\varphi)$ is acyclic, both results of applying (Distr) to φ are strictly greater than φ.

Now let φ be a constraint whose undirected graph has no simple hypernormal cycle. We can assume without loss of generality that φ is irredundant; otherwise we make it irredundant by removing dominance edges, which does not introduce new hypernormal cycles.

If (Distr) is not applicable to φ, φ is in solved form and hence satisfiable. Otherwise, we know that both results of applying the rule are strictly greater than φ. It can be shown that one of the results of an application of the distribution rule contains no simple hypernormal cycle. We omit this argument for lack of space; details can be found in the proof of Theorem 3 in (Althaus et al., 2000). Furthermore, the maximal length of a $<$ increasing chain of constraints is bounded by n^2, where n is the number of variables. Thus, applications of (Distr) can only be iterated a finite number of times on constraints without simple hypernormal cycles (given redundancy elimination), and it follows by induction that φ is satisfiable. $\square$

4.2 Testing for Simple Hypernormal Cycles

We can test an undirected constraint graph for the presence of simple hypernormal cycles by solving a *perfect weighted matching problem* on an auxiliary graph $A(G(\varphi))$. Perfect weighted matching in an undirected graph $G = (V, E)$ with edge weights is the problem of selecting a subset E' of edges such that each node is adjacent to exactly one edge in E', and the sum of the weights of the edges in E' is maximal.

The auxiliary graph $A(G(\varphi))$ we consider is an undirected graph with two types of edges. For every edge $e = (v, w) \in G(\varphi)$ we have two nodes e_v, e_w in $A(G(\varphi))$. The edges are as follows:

(Type A) For every edge e in $G(\varphi)$ we have the edge $\{e_v, e_w\}$.

(Type B) For every node v and distinct edges e, f which are both incident to v in $G(\varphi)$, we have the edge $\{e_v, f_v\}$ if either v is not a leaf, or if v is a leaf and either e or f is a tree edge.

We give type A edges weight zero and type B edges weight one. Now it can be shown (Althaus et al., 2000, Lemma 2) that $A(G(\varphi))$ has a perfect matching of positive weight iff the undirected version of $G(\varphi)$ contains a simple hypernormal cycle. The proof is by constructing positive matchings from cycles, and vice versa.

Perfect weighted matching on a graph with n nodes and m edges can be done in time

$O(nm \log n)$ (Galil et al., 1986). The matching algorithm itself is beyond the scope of this paper; for an implementation (in C++) see e.g. (Mehlhorn and Näher, 1999). Now let's say that k is the maximum number of dominance edges into the same node in $G(\varphi)$, then $A(G(\varphi))$ has $O((k+1)n)$ nodes and $O((k+1)^2 n)$ edges. This shows:

Proposition 4.4. *A constraint graph can be tested for simple hypernormal cycles in time $O((k+1)^3 n^2 \log n)$, where n is the number of variables and k is the maximum number of dominance edges into the same node.*

This completes the proof of Theorem 3.2: We can test satisfiability of a normal constraint by first constructing the auxiliary graph and then solving its weighted matching problem, in the time claimed.

4.3 Hypernormal Constraints

It is even easier to test the satisfiability of a *hypernormal* dominance constraint – a normal dominance constraint in whose constraint graph no node has two outgoing dominance edges. A simple corollary of Prop. 4.3 for this special case is:

Corollary 4.5. *A hypernormal constraint is satisfiable iff its undirected constraint graph is acyclic.*

This means that satisfiability of hypernormal constraints can be tested in linear time by a simple depth-first search.

5 Enumerating Solutions

Now we embed the satisfiability algorithms from the previous section into an algorithm for enumerating the *irredundant solved forms* of constraints. A solved form of the normal constraint φ is a normal constraint φ' which is in solved form and $\varphi \leq \varphi'$, with respect to the $\leq$ order from the proof of Prop. 4.3.[4]

Irredundant solved forms of a constraint are very similar to its solutions: Their constraint graphs are tree-shaped, but may still contain dominance edges. Every solution of a constraint is a solution of one of its irredundant solved forms. However, the number of irredundant solved forms is always finite, whereas the number of solutions typically is not: $X{:}a \wedge Y{:}b$ is in solved form, but each solution must contain an additional node with arbitrary label that combines X and Y into a tree (e.g. $f(a,b)$, $g(a,b)$). That is, we can extract a solution from a solved form by "adding material" if necessary.

1. Check satisfiability of φ. If it is unsatisfiable, terminate with failure.
2. Make φ irredundant.
3. If φ is in solved form, terminate with success.
4. Otherwise, apply the distribution rule and repeat the algorithm for both results.

Fig. 6: Algorithm for enumerating all irredundant solved forms of a normal constraint.

The main workhorse of the enumeration algorithm, shown in Fig. 6, is the distribution rule (Distr) we have introduced in Section 4. As we have already argued, (Distr) can be applied at most n^2 times. Each end result is in solved form and irredundant. On the other hand, distribution is an equivalence transformation, which preserves the total set of solved forms of the constraints after the same iteration. Finally, the redundancy elimination in Step 2 can be done in time $O((k+1)n^2)$ (Aho et al., 1972). This proves:

Theorem 5.1. *The algorithm in Fig. 6 enumerates exactly the irredundant solved forms of a normal dominance constraint φ in time $O((k+1)^4 n^4 N \log n)$, where N is the number of irredundant solved forms, n is the number of variables, and k is the maximum number of dominance edges into the same node.*

Of course, the number of irredundant solved forms can still be exponential in the size of the constraint. Note that for hypernor-

[4]In the literature, solved forms with respect to the NP saturation algorithms can contain additional labeling literals. Our notion of an irredundant solved form corresponds to a *minimal solved form* there.

mal constraints, we can replace the quadratic satisfiability test by the linear one, and we can skip Step 2 of the enumeration algorithm because hypernormal constraints are always irredundant. This improves the runtime of enumeration to $O((k+1)n^3N)$.

6 Reductions

Instead of proving Theorem 4.3 directly as we have done above, we can also reduce it to a *configuration problem* of *dominance graphs* (Althaus et al., 2000), which provides a more general perspective on related problems as well. Dominance graphs are unlabeled, directed graphs $G = (V, E \uplus D)$ with *tree edges* E and *dominance edges* D. Nodes with no incoming tree edges are called *roots*, and nodes with no outgoing ones are called *leaves*; dominance edges only go from leaves to roots. A *configuration* of G is a graph $G' = (V, E \uplus E')$ such that every edge in D is realized by a path in G'. The following results are proved in (Althaus et al., 2000):

1. Configurability of dominance graphs is in $O((k+1)^3 n^2 \log n)$, where k is the maximum number of dominance edges into the same node.
2. If we specify a subset $V' \subseteq V$ of *closed leaves* (we call the others *open*) and require that only open leaves can have outgoing edges in E', the configurability problem becomes NP-complete. (This is shown by encoding a strongly NP-complete partitioning problem.)
3. If we require in addition that *every* open leaf has an outgoing edge in E', the problem stays NP-complete.

Satisfiability of normal dominance constraints can be reduced to the first problem in the list by deleting all labels from the constraint graph. The reduction can be shown to be correct by encoding models as configurations and vice versa.

On the other hand, the third problem can be reduced to the problems of whether there is a plugging for a description in Hole Semantics (Bos, 1996), or whether a given MRS description can be resolved (Copestake et al., 1997), or whether a given normal dominance constraints has a *constructive solution*.[5] This reduction is by deleting all labels and making leaves that had nullary labels closed. This means that (the equivalent of) deciding satisfiability in these approaches is NP-hard.

The crucial difference between e.g. satisfiability and constructive satisfiability of normal dominance constraints is that it is possible that a solved form has no constructive solutions. This happens e.g. in the example from Section 5, $X{:}a \wedge Y{:}b$. The constraint, which is in solved form, is satisfiable e.g. by the tree $f(a, b)$; but every solution must contain an additional node with a binary label, and hence cannot be constructive.

For practical purposes, however, it can still make sense to enumerate the irredundant solved forms of a normal constraint even if we are interested only in constructive solution: It is certainly cheaper to try to find constructive solutions of solved forms than of arbitrary constraints. In fact, experience indicates that for those constraints we really need in scope underspecification, all solved forms do have constructive solutions – although it is not yet known why. This means that our enumeration algorithm can in practice be used without change to enumerate constructive solutions, and it is straightforward to adapt it e.g. to an enumeration algorithm for Hole Semantics.

7 Conclusion

We have investigated *normal* dominance constraints, a natural subclass of general dominance constraints. We have given an $O(n^2 \log n)$ satisfiability algorithm for them and integrated it into an algorithm that enumerates all irredundant solved forms in time $O(Nn^4 \log n)$, where N is the number of irredundant solved forms.

[5] A constructive solution is one where every node in the model is the image of a variable for which a labeling literal is in the constraint. Informally, this means that the solution only contains "material" "mentioned" in the constraint.

This eliminates any doubts about the computational practicability of dominance constraints which were raised by the NP-completeness result for the general language (Koller et al., 1998) and expressed e.g. in (Willis and Manandhar, 1999). First experiments confirm the efficiency of the new algorithm – it is superior to the NP algorithms especially on larger constraints.

On the other hand, we have argued that the problem of finding *constructive* solutions even of a normal dominance constraint is NP-complete. This result carries over to other underspecification formalisms, such as Hole Semantics and MRS. In practice, however, it seems that the enumeration algorithm presented here can be adapted to those problems.

Acknowledgments. We would like to thank Ernst Althaus, Denys Duchier, Gert Smolka, Sven Thiel, all members of the SFB 378 project CHORUS at the University of the Saarland, and our reviewers. This work was supported by the DFG in the SFB 378.

References

A. V. Aho, M. R. Garey, and J. D. Ullman. 1972. The transitive reduction of a directed graph. *SIAM Journal of Computing*, 1:131–137.

E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. 2000. An efficient algorithm for the configuration problem of dominance graphs. Submitted. `http://www.ps.uni-sb.de/Papers/abstracts/dom-graph.html`.

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

Johan Bos. 1996. Predicate logic unplugged. In *Proceedings of the 10th Amsterdam Colloquium*.

A. Copestake, D. Flickinger, and I. Sag. 1997. Minimal Recursion Semantics. An Introduction. Manuscript, `ftp://csli-ftp.stanford.edu/linguistics/sag/mrs.ps.gz`.

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

D. Duchier and S. Thater. 1999. Parsing with tree descriptions: a constraint-based approach. In *Proc. NLULP'99*, Las Cruces, New Mexico.

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

Z. Galil, S. Micali, and H. N. Gabow. 1986. An $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs. *SIAM Journal of Computing*, 15:120–130.

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

Jerry R. Hobbs and Stuart M. Shieber. 1987. An algorithm for generating quantifier scopings. *Computational Linguistics*, 13:47–63.

A. Koller, J. Niehren, and R. Treinen. 1998. Dominance constraints: Algorithms and complexity. In *Proceedings of the 3rd LACL*, Grenoble. To appear as LNCS.

M. P. Marcus, D. Hindle, and M. M. Fleck. 1983. D-theory: Talking about talking about trees. In *Proceedings of the 21st ACL*.

K. Mehlhorn and S. Näher. 1999. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge. See also `http://www.mpi-sb.mpg.de/LEDA/`.

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

Guy Perrier. 2000. From intuitionistic proof nets to interaction grammars. In *Proceedings of the 5th TAG+ Workshop*, Paris.

O. Rambow, K. Vijay-Shanker, and D. Weir. 1995. D-Tree grammars. In *Proceedings of the 33rd ACL*, pages 151–158.

J. Rogers and K. Vijay-Shanker. 1994. Obtaining trees from their descriptions: An application to tree-adjoining grammars. *Computational Intelligence*, 10:401–421.

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

A. Willis and S. Manandhar. 1999. Two accounts of scope availability and semantic underspecification. In *Proceedings of the 37th ACL*.