AN EFFICIENT GRAPH ALGORITHM FOR DOMINANCE CONSTRAINTS*

ERNST ALTHAUS†, DENYS DUCHIER‡, ALEXANDER KOLLER§, KURT MEHLHORN†, JOACHIM NIEHREN‡, AND SVEN THIEL†

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 *normal* dominance constraints and present an efficient graph algorithm for testing their satisfiablity in deterministic polynomial time. Previously, no polynomial time algorithm was known.

1. Introduction. The dominance relation of a tree is the ancestor relation between its nodes. Dominance constraints are logical descriptions of trees talking about the dominance relation.

Dominance based tree descriptions were first used in automata theory in the sixties [TW67], rediscovered in computational linguistics in the early eighties [MHF83], and investigated from a logical point of view in the early nineties [BRVS95]. Since then, they have found numerous applications in computational linguistics: they have been used for grammar formalisms [VS92, RVSW95, DT99, Per00], in natural language semantics [Mus95, ENRX98], and for discourse analysis [GW98].

The two most important computational tasks for dominance constraints are *satisfiability testing* – does the constraint describe a tree? – and *enumerating solutions*, i.e. the described trees. But as shown recently [KNT01], testing satisfiability is an NP-complete problem. Earlier attempts at processing dominance constraints [Cor94, VSWR95, DN00] all suffer from this fact. This has shed doubt on their practical usefulness.

In this article, we identify *normal* dominance constraints, a natural subclass of dominance constraints whose restrictions should be unproblematic for many applications. We present an efficient graph algorithm that decides satisfiability of normal dominance constraints in deterministic polynomial time. Previously, no polynomial time algorithm was known.

We derive the graph algorithm for testing satisfiability as follows. First, we introduce dominance graphs and define their configuration problem (investigated in [ADK+01]). Second, we show that the configurability of dominance graphs is linear time equivalent to the satisfiability of normal dominance constraints (first shown in [KMN00]). Third, we characterize the configurability of dominance graphs as the absence of certain cycles, which we finally test for by reduction to a matching problem.

We also discuss how to use the efficient satisfiability test to enumerate solutions. We apply a choice rule exhaustively while checking for satisfiability after each step. Both procedures have been implemented in C++ using the LEDA library [MN99] and applied to scope ambiguities in natural language semantics in the CLLS framework [ENRX98, EKN01, EKN02].

*This article joins the results from two conference publications [ADK+01] and [KMN00]. The authors are partially supported by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and by the Collaborative Research Centre (SFB) 378 of the Deutsche Forschungsgemeinschaft.

†Max-Planck-Institute for Computer Science, Saarbrücken, Germany

‡Programming Systems Lab, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany

§Department of Computational Linguistics, Universität des Saarlandes, Saarbrücken, Germany

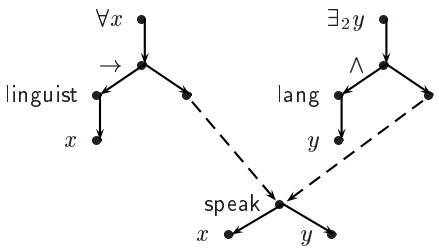


FIG. 2.1. *A simple dominance constraint.*

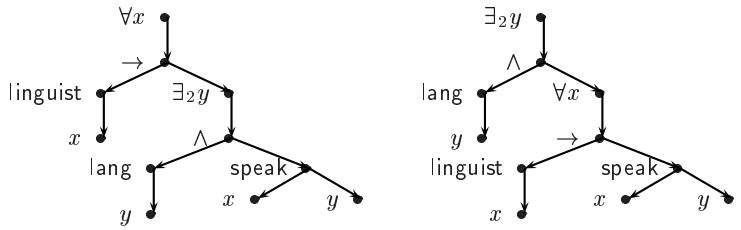


FIG. 2.2. *Readings represented by the constraint in Fig. 2.1.*

To complement our results, we finally investigate a close variant of the configuration problem of dominance graphs where closed leaves are permitted in addition. This variant is more general but also relevant for applications in computational linguistics [Bos96, CFS97]. We show that configurability is already NP-complete for the more general dominance graphs. Nevertheless, the presented algorithms can still help to solve this alternative problem more efficiently.

Plan of the paper. The first part of the paper introduces dominance constraints: We motivate using them in computational linguistics in Section 2; then we define them in Section 3, discuss their satisfiability problem, and introduce the concepts of *normal* dominance constraints and of *solved forms*. In the second part of the paper, we turn to a discussion of dominance graphs. We define them and relate them to normal dominance constraints in Section 4. Section 5 presents a basic algorithm for enumerating the solved forms of a dominance graph. Then we derive the above-mentioned characterization of configurability in Section 6, show how to test for this property efficiently in Section 7, and plug this efficient algorithm into the enumeration algorithm in Section 8. In the final part of the article, we apply this efficient enumeration algorithm back to normal dominance constraints and discuss an implementation (Section 9), and prove that the more general configurability with closed leaves is NP-complete (Section 10). Section 11 concludes and discusses further work.

2. Motivation. As one of the many applications of dominance constraints in computational linguistics, we will give a brief introduction to *scope underspecification* [EKN01, AC92, Rey93, Bos96].

This application is concerned with coping with ambiguous sentences such as the following:

(2.1) Every linguist speaks two languages.

Sentence (2.1) is ambiguous because it has two different possible meanings, indicated by the continuations

(2.2) ... namely, English and Chinese.

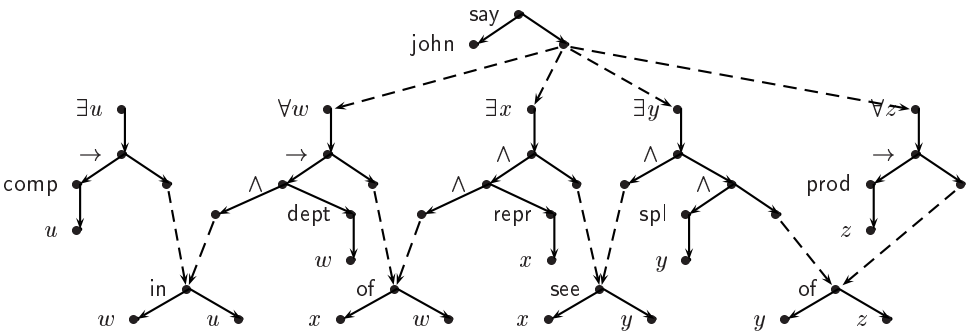


FIG. 2.3. *A dominance constraint describing the meaning of (2.6).*

(2.3) ... but not necessarily the same ones.

In the first reading, each linguist must speak the same two languages. In the second, no two linguists necessarily speak a common language, but each speaks at least two. We can represent the two possible meanings logically as the following first-order formulas, which can be represented as trees as in Fig. 2.2.

(2.4) $\forall x.(\mathsf{linguist}(x) \rightarrow \exists_2 y.\mathsf{lang}(y) \wedge \mathsf{speak}(x,y))$

(2.5) $\exists_2 y.\mathsf{lang}(y) \wedge \forall x.(\mathsf{linguist}(x) \rightarrow \mathsf{speak}(x,y))$

Ambiguity is a real problem to language processing because the number of readings of a sentence grows quickly with the number of "quantifiers" such as "every linguist" and "two languages", and interacts with other sources of ambiguity besides. The sentence (2.6) has already 56 readings, and larger examples are easy to construct.[5]

(2.6) John says that some representative of every department in a company saw a sample of each product.

The key observation to scope underspecification is that the differences between the readings are very systematic; all contain the same "semantic material" (e.g. representations of the constituents "every linguist", "two languages", and "speak"), which is only combined in different ways. The constraints on these combinations can be specified using dominance constraints.

An example is Fig. 2.1. This *constraint graph* is a description of the two readings of (2.1), shown in Fig. 2.2; it can be seen as a graphical representation of a dominance constraint. Similarly, the 56 readings of (2.6) can be represented by the graph in Fig. 2.3. In the paper, we will use constraint graphs to link the (logic) work on dominance constraints to graph algorithms.

Pictures as in Fig. 2.3 are being drawn in most modern approaches to scope underspecification. However, they are not always interpreted as dominance constraints [Rey93, Bos96]. The subtle difference in meaning has the surprising effect of making these other approaches NP-complete even when the graphs fall into the class where dominance constraints have polynomial satisfiability. We will show this in Section 10.

3. Satisfiability of Dominance Constraints. In this section, we define the the language of dominance constraints and recall known results on satisfiability. The variant of dominance constraints we employ describes constructor trees – ground

[5] The following sentence from [Hob83], which is interesting both in form and in content, has around 200 readings: "Many people feel that most sentences exhibit too few quantifier scope ambiguities for much effort to be devoted to this problem, but a casual inspection of several sentences from any text should convince almost everyone otherwise."

terms over a signature of function symbols – rather than feature trees as considered in [Smo92, BRVS95, ST94, MNP00].

3.1. Trees and Constructor Trees. We assume a finite or infinite signature Σ with function symbols $f, g, \ldots$, each of which is equipped with an arity $\mathsf{ar}(f) \geq 0$. Constants are function symbols of arity 0 denoted by a, b. We assume that Σ contains at least one constant and one symbol of arity at least 2.

A constructor tree can be defined either as a term or equivalently on the basis of directed graphs. The ground term $f(g(a, a))$, for instance, corresponds to the directed graph in Figure 3.1. Throughout this article, we will employ the graph based definition.

An *(unlabeled) tree* is a forest with exactly one root. A *forest* is a finite directed graph (V, E) where V is a finite set of *nodes* denoted by u, v, w, and $E \subseteq V \times V$ a set of *edges* such that the indegree of each node is at most 1 and there is no cycle. Each forest has at least one *root*, i.e. a node with indegree 0. We call the nodes with outdegree 0 the *leaves* of the forest.

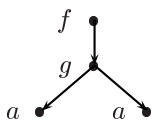


FIG. 3.1. $f(g(a, a))$

A *(finite) constructor tree* τ is a triple (V, E, L) consisting of a tree (V, E) and a labeling function $L : E \cup V \to \Sigma \cup \mathbb{N}$ s.t. $L(E) \subseteq \mathbb{N}$ (edge labels) and $L(V) \subseteq \Sigma$ (node labels). The edge labels in a constructor tree determine the order of the children of a node: for each node $u \in V$ and each natural number $1 \leq k \leq \mathsf{ar}(L(u))$, there is exactly one edge $(u, v) \in E$ with $L((u, v)) = k$.

We draw constructor trees as in Figure 3.1, by annotating nodes with their labels and ordering the edges such that their labels increase from left to right.

3.2. Constraint Language. The language of dominance constraints is a logical language that is interpreted over the class of tree structures. Tree structures are first-order model structures which specify certain relations between the nodes of a construtor tree.

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

DEFINITION 3.1 (Tree Structure). *The* tree structure *of a constructor tree τ with node set V is a first-order structure with domain V which provides the* dominance relation $\triangleleft^*$ *of τ and the* labeling relation *of τ for each function symbol $f \in \Sigma$.*

Let Vars be an infinite set of (node) variables $X, Y, Z, \ldots$ A *dominance constraint* φ is a conjunction of *dominance*, *inequality*, and *labeling* literals of the following form where $\mathsf{ar}(f) = n$:

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

We freely identify a constraint with the set of its literals. Let $\mathsf{Var}(\varphi)$ be the set of variables of φ. A pair of a tree structure τ with node set V and a *variable assignment* $\alpha : \mathsf{Var}(\varphi) \to V$ *satisfies* φ iff it satisfies all its literal in the obvious way. We say that (τ, α) is a *solution* of φ in this case; φ is *satisfiable* if it has a solution.

For instance, the following constraint that happens to be unsatisfiable:

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

It requires that node values of X_1 and X_2 are sisters that are both ancestors of the node value of Y. This is clearly impossible in a tree, since trees cannot branch upwards.

3.3. Constraint Graphs. We usually draw a dominance constraint as a *constraint graph*. For instance, the unsatisfiable constraint from above is drawn in Figure 3.2. It illustrates clearly that the constraint requires upward branching and thus cannot be satisfied by any tree.

The nodes of a constraint graph are the variables of the corresponding constraint. Labeling constraints relate to solid edges called *tree edges*. Dominance constraints are drawn as dashed lines called *dominance edges*. As for trees, we annotate labels to nodes of the graph and order tree edges from left to right. Note that we ignore inequalities in constraint graphs.

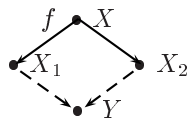


FIG. 3.2. *The unsatisfiable constraint* $X{:}f(X_1, X_2) \wedge X_1 \triangleleft^* Y \wedge X_2 \triangleleft^* Y$

We sometimes annotate variable names to the graph nodes. This is not always necessary since all occurences of the same variable are always represented by a single node in a constraint graph. We may thus freely omit variable names. In the motivating example (Figure 2.3), for instance, we have omitted all variable names.

Constraint graphs motivate the following notions to talk about constraints. We call a variable X *labeled* in a constraint φ if there exists a literal $X{:}f(\ldots)$ in φ. A *(solid) fragment* of constraint φ is a maximal set of variables in φ that are pairwise connected by labeling literals. A variable X is called a *root of a fragment* in φ if it does not occur in child position of a labeling literal in φ, i.e. if there is no Z such that $Z{:}f(\ldots X \ldots)$ belongs to φ. A *hole of a fragment* is a variable in φ that is unlabeled in φ. A *leaf of a fragment* is either a hole or a variable labeled by a constant, i.e. a variable X with $X{:}a$ in φ.

3.4. Satisfiability. We are interested in two natural problems concerning dominance constraints that are both motivated by our application: first of all we would like to test satisfiablity, and second, we would like to enumerate all solutions of a satisfiable dominance constraint.

The complexity of the satisfiability problem of dominance constraints was investigated in [KNT01] and shed doubts on their usefulness.

THEOREM 3.2. *Satisfiablitiy of dominance constraints is NP-complete.*

Deciding satisfiability in non-deterministic polynomial time is quite simple: In a first step one guesses whether $X \triangleleft^* Y$ or $\neg X \triangleleft^* Y$ for each two variables X, Y in a given constraint. In a second step, one tests the consistency of these relationships.

The NP-hardness proof relies on the fact that solid fragments of a constraint graph may *overlap* in a solution. This means that distinct labeled variables may be assigned to the same node of a tree.

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 graph is shown in Figure 3.3. Every solution must map X to the same node as either Y or Y_1. We say that X overlaps with Y or Y_1 in a solution of this constraint.

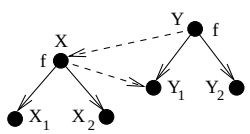


FIG. 3.3. *Overlap*

We call an overlap *proper* if it involves two labeled variables. In the applications in computational linguistics, we typically do not want proper overlap (but may accept overlaps of roots with holes).

The subclass of dominance constraints that excludes proper overlap (and fixes some minor inconveniences) is the class of *normal* dominance constraints.

3.5. Normal Dominance Constraints. We next distinguish a fragment of dominance constraints which we will show to have a polynomial time satisfiability problem.

DEFINITION 3.3 (Normal dominance constraint). *A dominance constraint φ is called* normal *iff for all variables $X, Y \in \mathsf{Var}(\varphi)$:*

1. *There is no proper overlap in solutions of φ: $X \neq Y$ in φ if X and Y are distinct variables that are labeled in φ.*
2. *Solid fragments are tree shaped or cyclic: every variable in φ appears at most once as a parent and at most once as a child in a labeling literal of φ.*
3. *Dominance edges go from holes to roots: if $X \triangleleft^* Y$ in φ then X is unlabeled in φ whereas Y is labeled but does not occur in child position in φ.*
4. *There are no empty fragments: every hole of φ occurs in some child position.*

Conditions 1 and 4 say that only roots and holes have the permission to overlap in a solution of a normal constraint. Distinct holes cannot overlap since they must have distinct parents, which are labeled variables that cannot overlap. For a similar reason, it is impossible that a hole overlaps with a labeled node that has a parent.

Condition 2 requires acyclic fragments to be tree shaped. This excludes many constraints, as for instance $X{:}f(Y,Y)$, $X{:}f(Y_1,Y_2) \wedge X{:}f(Z_1,Z_2)$ or $Y{:}f(X) \wedge Z{:}f(X)$. The last two examples are particularly difficult to treat when subsumed by a larger constraint: they entail equations ($Y_1{=}Z_1$, $Y_2{=}Z_2$, respectively $Y{=}Z$) whose global consequences are difficult to predict. W.l.o.g we can always restrict ourselves to normal constraint with acyclic fragments. Other constraints are unsatisfiable anyway.

Condition 3 forbids to express equality through two side dominance: $X \triangleleft^* Y \wedge Y \triangleleft^* X$ is not normal since a variable cannot be at the same time a root and a hole. Condition 3 is also violated by the dominance edge $X \triangleleft^* Y_1$ in the constraint from overlap example in Fig. 3.3. It goes from a root to a hole, instead vice versa.

In the following theorem we state the main result of this article, which will follow from the results presented in the succeeding sections.

THEOREM 3.4. *Satisfiability of normal dominance constraints can be decided in deterministic polynomial time.*

3.6. Solved Forms. We stated above that we would like to have an algorithm that enumerates all solutions of a given normal dominance constraint. Interpreting this proposition literally makes not much sense as the reader already might have noticed. For instance, we can solve the constraint $X{:}a$ by all trees that have a node labeled by a. Indeed, every satisfiable constraint has an infinite number of solutions, so that we probably do not want to enumerate all of them.

What we want to do is to enumerate all solved forms of a normal dominance constraint instead of all solutions. The idea behind a solved form is that it should be similar to a solution but not describe its irrelevant parts. For instance, $X{:}a$ is a perfect solved form since all its solutions can be easily read off from this constraint.

We will now define an appropriate notion of solved forms. In particular, it should hold that a normal dominance constraint has a solution if and only if it has a solved form. Given a constraint φ we define a relation R_φ on the variables of φ that we call the *reachability relation* of φ. This relation is the transitive closure of the following relation:

$$\{(X,Y) \mid X{:}f(\ldots,Y,\ldots) \in \varphi \text{ or } X \triangleleft^* Y \in \varphi\}$$

We say that Y can be reached from X if $(X,Y) \in R_\varphi$. In this case, it clearly holds that φ entails the dominance $X \triangleleft^* Y$.

DEFINITION 3.5 (Solved Form). *A normal dominance constraint φ is in* solved form *if it satisfies the following two properties for all variables X, Y, Z in $\mathsf{Var}(\varphi)$:*

1. *Dominance edges do not branch upwards: if X and Y are distinct then not both $X \triangleleft^* Z$ in φ and $Y \triangleleft^* Z$ in φ.*
2. *The graph of φ is acyclic: $(X,X) \notin R_\varphi$.*

In other words, a normal dominance constraint φ is in solved form if and only if its constraint graph is a forest.

A *solved form of a normal constraint* φ is a normal constraint φ' that is in solved form, contains the same labeling literals as φ, and has a stronger reachability relation, which means $R_\varphi \subseteq R_{\varphi'}$.

LEMMA 3.6. *Every normal dominance constraint in solved form has a solution.*

Proof. We have to construct a tree solution for a solved form φ. The idea is that the constraint graph of a solved form is already a forest. It is sufficient to transform this forest into a tree without dominance edges. This is quite simple given the transformation illustrated in Figure 3.4. (Note that we assumed Σ to contain a function symbol f of arity at least two and a constant a.)

In the first step, we turn the forest into a tree by adding a top most fragment. Let $Y_1, \ldots, Y_m$ be the minimal elements in the reachability order R_φ which exist since R_φ is acyclic. We can then define a new solved form φ_1 with a top most fragment, by adding a new fragment to φ with a single hole with dominance edges towards all roots $Y_1, \ldots, Y_m$.

In the second step we repeatedly transform dominance edges into tree edges. We stop, once no dominance edge is left. The idea is illustrated in Figure 3.4. Recall that we assumed that our signature contains a constant a and a function symbol f of arity $n \geq 2$. We will also use a function dist on constraints, which is defined for all constraints φ' by $\mathsf{dist}(\varphi') = \varphi' \wedge \bigwedge\{X \neq Y \mid X, Y \in \mathsf{Var}(\varphi') \text{ distinct}\}$. Suppose that there still exists a hole X in φ_1 from where dominance edges start. Let $Y_1, \ldots, Y_m$ be all the roots of φ_1 such that the dominance literal $X \triangleleft^* Y_i$ belongs to φ_1 for $i = 1, \ldots, m$. We construct φ_2 by removing all these literals from φ_1 and distinguish two cases:

- If $m > n$, we fix a fresh variable Z and define:

$$\varphi_3 = \mathsf{dist}(\varphi_2 \wedge X{:}f(Y_1, \ldots, Y_{n-1}, Z) \wedge \textstyle\bigwedge_{i=n}^{m} Z \triangleleft^* Y_i)$$

  φ_3 is a solved form that entails φ, and it contains $n-1$ dominance literals less than φ_1.
- If $m \leq n$, we fix fresh variables $Z_{m+1}, \ldots, Z_n$ and define:

$$\varphi_3 = \mathsf{dist}(\varphi_1 \wedge X{:}f(Y_1, \ldots, Y_m, Z_{m+1}, \ldots, Z_n) \wedge \textstyle\bigwedge_{j=m+1}^{n} Z_j{:}a)$$

  φ_3 is a solved form that entails φ, and it contains m dominance literals less than φ_1.

By applying the above transformation repeatedly, we obtain a solved form φ^* which entails φ and contains no dominance literals.

In the third step we can easily satisfy φ^* by the constructor tree that corresponds to φ^* itself. □

In the construction of a solution of φ we had to "invent" variables that are not present φ. Thus the constructor tree in the solution contains nodes that do not

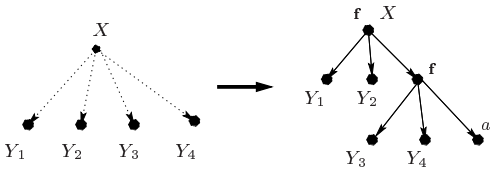


FIG. 3.4. *Transforming dominance edges into tree edges. Here, 4 dominance edges are transformed while using a function symbol of arity 3.*

correspond to variables of φ. In the following lemma we will show that every satisfiable normal constraint has a solved form and that it can essentially be obtained by removing the invented material.

LEMMA 3.7. *Every solution of a normal dominance constraint φ also satisfies some solved form of φ.*

Proof. Let (α, τ) be a solution of φ. In order to construct a solved form, we define a partial function hole on the root variables of φ. Consider a root Y, the function is defined if there is a hole X with $\alpha(X) \triangleleft^* \alpha(Y)$. Since τ is a tree, there is hole Z such that $\alpha(Z) \triangleleft^* \alpha(Y)$ and $\alpha(X) \triangleleft^* \alpha(Z)$ for all holes X with $\alpha(X) \triangleleft^* \alpha(Y)$[6]. We set $\mathsf{hole}(Y) = Z$. Let φ_l denote the conjunction of the labeling and inequality literals of φ, then the following is a solved form of φ:

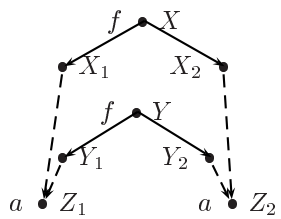
$$\varphi' = \varphi_l \wedge \bigwedge \{\mathsf{hole}(Y) \triangleleft^* Y \mid Y \text{ is a root for which } \mathsf{hole} \text{ is defined}\}$$

Clearly, φ' is a normal constraint in solved form. We have to show $R_\varphi \subseteq R_{\varphi'}$. Since both constraints have the same labeling literals, it suffices to prove for every dominance literal $X \triangleleft^* Y$ in φ that $(X, Y) \in R_{\varphi'}$. We will show a stronger statement: if X is a hole and Y is a root with $\alpha(X) \triangleleft^* \alpha(Y)$, then $(X, Y) \in R_{\varphi'}$.

We proceed by induction on the length of the path from $\alpha(X)$ to $\alpha(Y)$. For $Z = \mathsf{hole}(Y)$, we have $\alpha(X) \triangleleft^* \alpha(Z)$. If $X = Z$, the claim holds. Otherwise, let R denote the root of the fragment of Z. The hole X can only overlap with a root, so we get $\alpha(X) \triangleleft^* \alpha(R)$. As $\alpha(R) \neq \alpha(Z)$, we can apply the induction hypothesis and obtain $(X, R) \in R_{\varphi'}$. Since (R, Z) and (Z, Y) belong to $R_{\varphi'}$, the claim follows from the transitivity of the reachability relation. □

The validity of Lemma 3.7 heavily depends on the absence of proper overlaps (Condition 1 of Def. 3.3). This is illustrated by the following example:

$$X{:}f(X_1, X_2) \wedge Y{:}f(Y_1, Y_2) \wedge \bigwedge_{i=1}^{2} (X_i \triangleleft^* Z_i \wedge Y_i \triangleleft^* Z_i \wedge Z_i{:}a)$$



This constraint satisfies all normality conditions except for the overlap restriction. It also has a solution but no solved form. The reason for this problem is that X and Y overlap properly in all solutions of this constraint.

The combination of Lemmas 3.6 and 3.7 yields the following proposition, which justifies computing with solved forms instead of solutions:

PROPOSITION 3.8. *A normal dominance constraint has a solved form if and only if it is satisfiable.*

[6]So either $\alpha(Z) = \alpha(Y)$ (i.e. Y is plugged into Z) or $\alpha(Z)$ is the lowest proper ancestor of $\alpha(Y)$ for which α^{-1} is defined (i.e. which is not "invented").

4. Configurability of Dominance Graphs. For the satisfiability problem it turns out that we do not have to consider any labels. If we delete all labels from the constraint graph but keep the information whether an edge is a tree or a dominance edge, we get a dominance graph (given two minor assumptions):

DEFINITION 4.1 (Dominance Graph). *A* dominance graph *is a directed graph* $G = (V, E \dot\cup D)$ *satisfying the following two conditions:*

1. *The graph* $G = (V, E)$ *defines a collection* T *of node disjoint trees of height at least 1.*
2. *Each edge in* D *goes from a leaf of some tree in the collection to the root of some tree in the collection.*

We will use analogous notions for dominance and constraint graphs: The edges in E are called *tree edges*, and the edges in D are called *dominance edges*. A *leaf* is a node with no outgoing tree edge and a *root* is a node with no incoming tree edge. A dominance edge $d = (v, w)$ is *redundant* if there is a path from v to w in $G \setminus d$. The *reachability relation* R_G of a dominance graph $G = (V, E \dot\cup D)$ is the set of all pairs (u, v) such that there is a (directed) path from u to v in G, i.e. R_G is the transitive closure of the binary relation induced by the edge set of G.

We need a new notation that replaces the notion of a solution in the logical sense. Now the idea is that we want to assemble the trees in T by plugging roots into leaves.

DEFINITION 4.2. *We say that a dominance graph* G *is a* configuration *iff it is a forest and the edges in* D *form a matching, i.e. every node of* G *is incident to at most one edge of* D.[7]

We call a dominance graph $G' = (V', E' \dot\cup D')$ a *configuration of* G iff $V = V'$, $E = E'$, G' is a configuration, and $R_G \subseteq R_{G'}$, i.e. G' realizes all dominance edges in G. A dominance graph is *configurable* if it has a configuration.

4.1. Solved forms. In the sequel we will prove the equivalence of the configurability problem for dominance graphs and the satisfiability problem for normal dominance constraints (Lemma 4.5). In order to do so, we extend the notion of solved forms to dominance graphs.

DEFINITION 4.3. *A dominance graph* G *is* in solved form *iff it is a forest.*

By definition, every configuration is a solved form. Unlike a configuration, a solved form does not require its dominance edges to form a matching.

We call a dominance graph $G' = (V', E' \dot\cup D')$ a *solved form of* G iff $V = V'$, $E = E'$, G' is a solved form and $R_G \subseteq R_{G'}$. A dominance graph is *solvable* if it has a solved form. The following lemma shows that configurability and solvability are equivalent for dominance graphs.

LEMMA 4.4. *Every dominance graph in solved form is configurable.*

Proof. For the proof, we define a *problem leaf* to be a leaf with more than one outgoing dominance edge; our aim will be to eliminate problem leaves from solved forms.

The proof is by induction on weights (d, a) of graphs G, where d is the negative minimum depth of a problem leaf of G (or $-\infty$ if there is none), and a is the total number of dominance edges emanating from problem leaves of minimum depth (potentially 0). We consider the lexicographic order on these weights.

Solved forms without problem leaves (i.e. with weight $(-\infty, 0)$) are configurations, so the lemma is trivially true in this case. So let G be a solved form that does have

[7] And hence D defines a partial function from roots to holes which specifies for every matched root where to plug it.

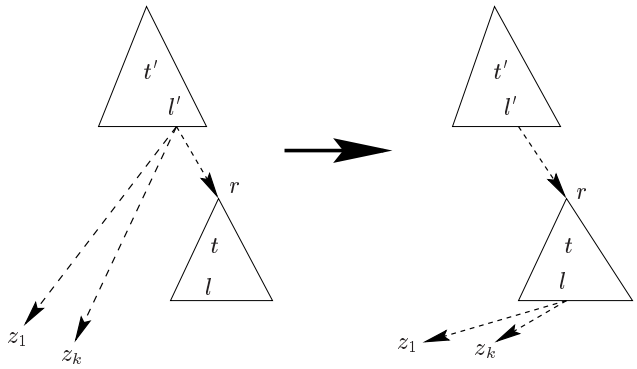


FIG. 4.1. *Application of Rule 1: All dominance edges of l' except for (l', r) are shifted down to the leaf l.*

problem leaves. Let G have weight (d, a), and assume that we know that all solved forms of lower weight do have configurations. Then we can apply the following rule to a problem leaf l' of minimum depth:

SIMPLIFICATION RULE 1. *Let $e = (l', r)$ be a dominance edge from the leaf l' of a tree t' to the root r of a tree t. Let l be an arbitrary leaf of t. Change any dominance edge (l', z) with $z \neq r$ into (l, z), see Figure 4.1.*

The result G' is still in solved form, and its weight is strictly lower than that of G; so by the induction hypothesis, G' has a configuration G_c. But G_c also realizes all dominance edges of G. This is obvious for (l', r) and for all dominance edges which do not start in l'. For an edge (l', z) with $z \neq r$ we note that this edge is realized because there is a path from l' to l in G' and G_c realizes the edge (l, z). So G has a configuration as well. □

4.2. Dominance Graphs of Normal Constraints. We now map normal dominance constraints to dominance graphs while ignoring labelings.

Let φ be a normal dominance constraint. We define a graph $G(\varphi) = (\mathsf{Var}(\varphi), E \dot\cup D)$, where the set of *tree edges* E and *dominance edges* D are defined as follows:

$$\begin{aligned} E &= \{(X, X_i) \mid X{:}f(X_1, \ldots, X_n) \text{ in } \varphi,\ 1 \leq i \leq n\} \\ D &= \{(X, Y) \mid X \lhd^* Y \text{ in } \varphi\} \end{aligned}$$

LEMMA 4.5. *For a normal dominance constraint φ the following holds if none of its roots is labeled by a constant and if all its fragments are acyclic.*

1. *The graph $G(\varphi)$ is a dominance graph.*
2. *The relations R_φ and $R_{G(\varphi)}$ are equal.*
3. *φ is in solved form iff $G(\varphi)$ is.*
4. *If φ' is a solved form of φ then $G(\varphi')$ is a solved form of $G(\varphi)$ and vice versa.*

Proof. We will proof the statements one by one:

1. Conditions 1 and 2 of Definition 3.3 ensure that all acyclic fragments of φ are trees. Since we assumed all fragments to be acyclic, it follows that $(\mathsf{Var}(\varphi), E)$ is a collection of node-disjoint trees. The roots of these trees must also be roots of fragments in φ since there are no empty fragments by Condition 4. This shows that the height of all trees in $G(\varphi)$ is at least one. And finally, from Condition 3 we can conclude that dominance edges can only go from leaves to roots.

2. We have equality since both relations are defined as the transitive closure of the same set.
3. Only the roots of the normal constraint φ may have more than one incoming edge in $G(\varphi)$ all of which are dominance edges. But every root in a solved form has by definition at most one incoming dominance edge. Since solved forms are acyclic by definition, it follows that their graphs are always forests and thus in solved form. The converse implication is obvious.
4. This an immediate consequence of the previous statements and the definitions of solved form for normal dominance constraints and dominance graphs.

□

We call a dominance graph G *arity consistent* if for all nodes v of G there exists a function symbol $f \in \Sigma$ such that the number of tree edges emanating from v is equal to the arity of f.

LEMMA 4.6. *For every arity consistent dominance graph G there exists a normal dominance constraint φ such that $G(\varphi) = G$.*

This lemma is obvious. The following theorem summarizes our results of the previous two lemmas:

THEOREM 4.7. *The following four problems are linear time equivalent:*

1. *Satisfiability of normal dominance constraints.*
2. *Existence of solved forms for normal dominance constraints.*
3. *Existence of solved forms for dominance graphs.*
4. *Configurability of dominance graphs.*

Proof. We have to check the application conditions of Lemmas 4.5 and 4.6.

1. We can always assume fragments to be acyclic. The existence of cycles can be checked easily in linear time, and normal constraints with cycles are unsatisfiable anyway.
2. We can assume that normal dominance constraints do not contain roots that are labeled by constants. Otherwise, we can replace the fragment $X{:}a$ by $X{:}f(X_1, \ldots, X_n) \wedge \bigwedge_{i=1}^{n} X_i{:}a$ for some fresh variables $X_1, \ldots, X_n$ and sufficiently many inequalities. This operation requires constant time and clearly preserves normality,but also satisfiability as the inner structure of fragments is anyway irrelevant for the satisfiability of normal constraints.
3. We can make all dominance graphs arity consistent with a similar transformation as in the proof of Lemma 3.6 (see also Figure 3.4). We use the fact that we have a function symbol of arity $n \geq 2$ and a constant symbol (which has arity 0). Let u denote a node in a dominance graph G with children $v_1, \ldots, v_m$. If $m \neq n$ and $m \neq 0$, we can apply one of the following transformations:
   - Case $0 < m < n$:
     Add new nodes $v_{m+1}, \ldots, v_n$ and the edges $(u, v_{m+1}), \ldots, (u, v_n)$ to G. Then u and the newly added nodes are arity consistent.
   - Case $m > n$:
     Add a new node u' and make u' a child of u by adding the edge (u, u'). Then shift the children $v_n, \ldots, v_m$ of u down to u' by replacing the edges $(u, v_n), \ldots, (u, v_m)$ with the edges $(u', v_n), \ldots (u', v_m)$. After that u is arity consistent, and the out-degree of u' is less than that of u. If u' is not arity-consistent, we apply the appropriate transformation to u'.

   We see that the transformations preserve satisfiability because only the inner structure of the fragments changes, but the reachability relation (restricted to

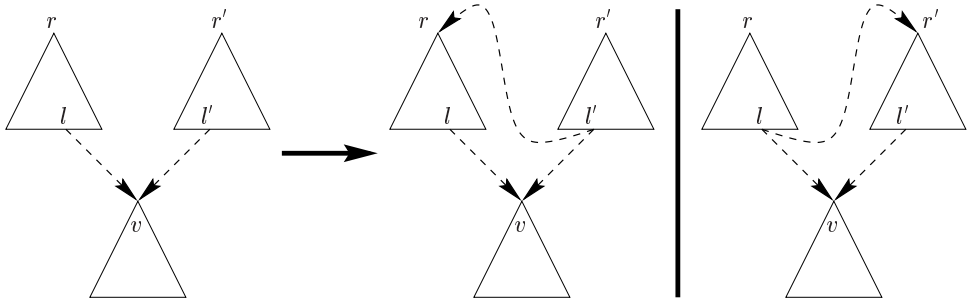


FIG. 5.1. *Two graphs H, H' are generated by applying the choice rule to the graph G on the left hand side.*

the original nodes) remains the same. Since n is a constant, the time required to make u arity consistent (including recursive transformations) is $O(m)$.

□

5. Enumeration of Minimal Solved Forms. Now that we have reduced the problem of solving normal dominance constraints to the problem of finding solved forms of dominance graphs, we show in this section how to enumerate solved forms of a dominance graph G. We are interested in solved forms that contain no unnecessary dominance edges. Let G' be a solved form of G that is transitively reduced. We call G' a *minimal solved form of* G if there is no solved form of G whose reachability relation is strictly contained in $R_{G'}$. Our algorithm below enumerates exactly the minimal solved forms of G. However, the algorithm may take exponential time to produce even a single solved form because it blindly enumerates all cases. Using the efficient configurability test we derive in Sections 6 and 7, we will optimize this algorithm to enumerate solved forms in polynomial time per solved form (Section 8).

The enumeration algorithm applies the following simplification rules:

SIMPLIFICATION RULE 2 (Redundancy Elimination). *All redundant dominance edges, i.e. edges that are implied by transitivity, can be removed. In particular, parallel edges can be combined into one.*

SIMPLIFICATION RULE 3 (Choice). *Let v be a root with at least two incoming dominance edges (l, v) and (l', v) and let r and r' be the roots of the trees containing leaves l and l', respectively. Generate two new graphs H and H' by adding either (l', r) or (l, r') to D, see Figure 5.1.*

The enumeration of the solved forms can be carried out by a recursive algorithm:

1. Make the graph reduced, i.e. apply Rule 2.
2. If the graph contains a (directed) cycle, terminate this recursion since the graph has no configuration.
3. If the graph is in solved form, report it and terminate this recursion.
4. Otherwise, apply the choice rule and apply the algorithm to the two newly generated graphs.

We will now prove that the algorithm is correct. First we observe that if the algorithm does not report a solved form in the third step, then the graph is acyclic, but not a forest. And hence, there must be node v with two incoming edges. It is easy to see that this can only be root, which implies that the choice rule can be applied to v.

Now we analyze the effect of the simplification rules on the set of solved forms.

The removal of redundant edges (Rule 2) does not change the reachability relation, which implies that the solved forms remain the same, too. An application of the choice rule (Rule 3) increases the reachability relation and partitions the set of solved forms in two disjoint sets: In a solved form S of G the nodes l and l' are both ancestors of v and therefore either l' is ancestor of l and hence of r or vice versa. This implies that S is either a solved form of H or of H'.

From this we conclude that every enumerated solved form is indeed a solved form of the original graph. On the other hand, we see that for every solved form S there exists exactly one solved form S' such that S' is enumerated by the algorithm and its reachability relation has the property $R_{S'} \subseteq R_S$. If S is minimal, we get $R_{S'} = R_S$, and since both graphs are transitively reduced and acyclic, we obtain $S' = S$. Thus the algorithm enumerates at least all minimal solved forms.

What remains to show is that all enumerated solved forms are minimal. So let S be solved form that is reported by the algorithm. Assume that S is not minimal, i.e. there exists a solved form S' with $R_{S'} \subset R_S$. Since both S and S' are enumerated there must be a step in the computation which "separated" the two graphs. This means there is an application of the choice rule which generated two graphs H and H' such that $R_H \subseteq R_S$ and $R_{H'} \subseteq R_{S'}$. From $R_{S'} \subset R_S$ we infer $R_{H'} \subseteq R_S$. Since R_S cannot contain both $R_{H'}$ and R_H, we have a contradiction.

To prove termination, we derive an upper bound for the maximum recursion depth. We reconsider the reachability relation R_G of a graph G. If G is acyclic, the cardinality of R_G is at most $\binom{n}{2} \leq n^2$, where n is the number of nodes in G. Thus, whenever the size of the relation becomes greater than $\binom{n}{2}$, the recursion terminates immediately. But if we apply the choice rule to a reduced, acyclic dominance graph, the size of the relation increases strictly, i.e. $|R_G| < \min(|R_H|, |R_{H'}|)$. This is because $R_H \supseteq R_G$, and $(l', r) \in R_H$ but (l', r) cannot be in R_G, otherwise (l', v) would have been redundant. A similar argument holds for H'.

6. Characterization of the Existence of Solved Forms. We give a graph theoretic characterization of solvability; as this is equivalent to configurability by Lemma 4.4, the result carries over to configurability.

The *undirected dominance graph* $G_u = (V, E_u \dot\cup D_u)$ corresponding to the dominance graph $G = (V, E \dot\cup D)$ is the undirected graph obtained by making all edges of G undirected. More precisely, we set $E_u = \{(\{u, v\}, \mathit{tree}) \; ; \; (u, v) \in E\}$ and define $D_u = \{(\{u, v\}, \mathit{dom}) \; ; \; (u, v) \in D\}$. This explicit distinction is important. Consider the dominance graph consisting of the two nodes r and l, the tree edge (r, l) and the dominance edge (l, r). If there were no explicit distinction, then both edges would correspond to the same undirected edge. However, when we talk about undirected edges in the sequel, we will only state the first component because it will be clear whether we refer to a tree or a dominance edge.

Now, we want to define the notion of a *cycle* in an undirected graph, which may differ from the reader's usual notion. A cycle C in an undirected graph is a sequence $[v_0, e_0, v_1, e_1, \ldots, e_{n-1}, v_n]$ of nodes $v_0, \ldots, v_n$ and edges $e_0, \ldots, e_{n-1}$ with $n \geq 2$ such that $v_0 = v_n$ and for $i = 0, \ldots, n-1$ the edge e_i is incident to v_i and v_{i+1}, and $e_i \neq e_{(i+1) \bmod n}$. This means we require a cycle to consist of at least 2 edges, we do not exclude that a cycle uses a node or an edge more than once, however any two consecutive edges must be distinct. We call C *edge-simple* if the edges in the sequence are pairwise different. C is said to be *simple* if all the visited nodes $v_0, \ldots, v_{n-1}$ (and hence also the edges) are pairwise different. Usually, we are not interested in the sequence of nodes and we identify C with the sequence $e_0 \circ e_1 \circ \ldots \circ e_{n-1}$ of its edges.

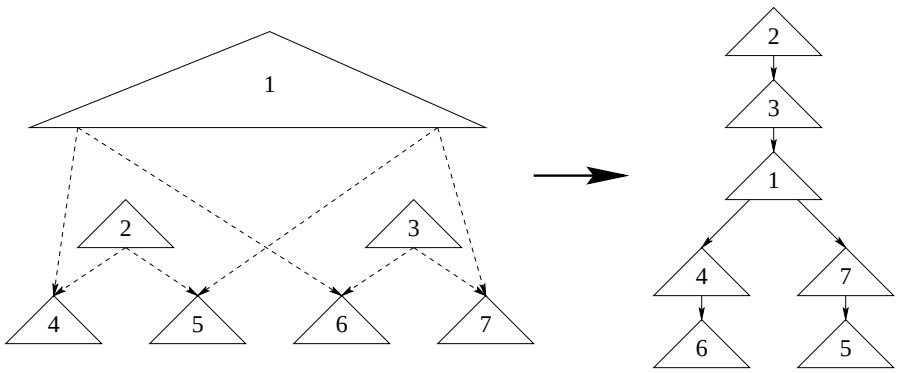


FIG. 6.1. *A solvable dominance graph and one of its solved forms. The graph contains an undirected cycle, but no hypernormal cycle.*

6.1. Hypernormal Dominance Graphs.

Let us first investigate a simpler subproblem of the solvability problem. A dominance graph $G = (V, E \dot\cup D)$ is *hypernormal* if for every leaf l in (V, E) there is at most one dominance edge $(l, .)$ in D.

PROPOSITION 6.1. *Let $G = (V, E \dot\cup D)$ be a hypernormal dominance graph. If G_u contains a cycle then G is unsolvable.*

Proof. The proof is by induction on the minimal number k of dominance edges in a simple cycle C of G_u. Clearly, the case $k = 0$ cannot occur. If $k = 1$ then there exists a dominance edge from a leaf l to the root of the fragment of l, and hence G is not solvable. For $k > 1$, assume that we know the result to be true for $k - 1$. C either does not contain any nodes at which its edges change directions; then it is also a cycle in G and hence, G is clearly unsolvable. Or C does change directions, then it must contain two dominance edges (l, r) and (l', r) into the same root. Both results of applying the choice rule produce graphs with a simple cycle containing $k-1$ dominance edges, so both are unsolvable. But then, G must be unsolvable as well. □

The converse of the above proposition is also true. If G is not solvable, then G_u contains a cycle. This statement will be a corollary of Theorem 6.2, which we will prove below.

6.2. Dominance Graphs.

The Proposition 6.1 does not carry over literally to the general case: Figure 6.1 is a counterexample. In order to state our theorem for the general case, we call a subgraph H_u of G_u *hypernormal* if the corresponding directed subgraph H of G is hypernormal. In particular, a *hypernormal cycle* in G_u is a cycle that contains for every leaf l at most one incident dominance edge.

THEOREM 6.2. *Let $G = (V, E \dot\cup D)$ be a dominance graph.*

(a) G is solvable iff G_u does not contain a hypernormal cycle.

(b) G is solvable iff every hypernormal subgraph of G is.

Note that this implies that a graph G is configurable iff G_u has no hypernormal cycle, by Lemma 4.4.

Proof. Part (b) follows immediately from part (a). If some hypernormal subgraph of G is unsolvable, G is unsolvable. If every hypernormal subgraph of G is solvable, G_u contains no hypernormal cycle, and hence G is solvable by part (a). We turn to part (a).

Assume first that G_u contains a hypernormal cycle C. Let D' be the dominance edges of G corresponding to edges in C. Then $G' = (V, E \dot\cup D')$ is a hypernormal dominance graph such that G'_u contains C. By Proposition 6.1, G' is unsolvable and hence G is unsolvable.

It remains to prove the converse: If G is unsolvable, G_u contains a simple hypernormal cycle. Suppose we run the algorithm of Section 5 on G. The computation of this algorithm can be modeled as a binary tree. We label every node x with a dominance graph $D(x)$. The root of the computation tree is labelled with G. Whenever the algorithm applies the choice rule and generates two new dominance graphs, we grow the tree by attaching two new nodes at the current node and label each node with one of the two new dominance graphs. Since G is unsolvable, the leaves of the tree are labelled with graphs that contain directed cycles.

For every node x in the tree we will show that $D(x)_u$ contains a simple hypernormal cycle. We prove this by induction on the height of x in the computation tree. If the height is 0, then x is a leaf and the claim clearly holds, for any simple directed cycle translates to an (undirected) simple hypernormal cycle.
Assume now, that the height is greater than 0. So x has two children labelled with the graphs H and H'. The two graphs have been generated by an application of the choice rule to $D(x)$. And by the induction hypothesis we may assume that both H_u and H'_u contain a simple hypernormal cycle. Suppose that v is the root and that (l, v) and (l', v) are the edges considered in the above application of the choice rule. Let r be the root of the tree with the leaf l and r' be the root of the tree with the leaf l'. We have a simple hypernormal cycle C_1 in H_u which uses the dominance edge $\{r, l'\}$. So the tree edge $\{l', r'\}$ must also belong to C_1, and hence we may assume $C_1 = \{r, l'\} \circ \{l', r'\} \circ P_1$. Similarly, H'_u contains a simple hypernormal cycle $C_2 = \{r', l\} \circ \{l, r\} \circ P_2$.

If P_1 or P_2 visits v, we can construct a hypernormal cycle in G_u. Suppose $P_1 = P' \circ P''$ such that P' ends in v. Then $P' \circ \{v, l'\} \circ \{l', r'\}$ is a simple hypernormal cycle because P' avoids l'. (If P_2 visits v, we can apply a similar argument.)

So we may assume that both C_1 and C_2 avoid v. Let w denote the first node on P_1 different from r' that also lies on P_2. (The node w may be equal to r.) We decompose P_1 and P_2 such that $P_i = Q_i \circ R_i$, Q_i ends at w and R_i starts at w. Q_1 is a path from r' to w, and R_2 is a path from w to r'. By the choice of w, we have that $Q_1 \circ R_2$ is a simple cycle.
If it is not hypernormal, then we are in the situation of Figure 6.2: w is a leaf, Q_1 ends with a dominance edge and R_2 starts with a dominance edge. Since P_2 is hypernormal, Q_2 ends with the tree edge incident to w. Now we consider the cycle $C = Q_2 \circ Q_1^{\text{rev}} \circ \{r', l'\} \circ \{l', v\} \circ \{v, l\} \circ \{l, r\}$. Obviously, any two consecutive edges on C are hypernormal. So it remains to prove that C is simple, i.e. we have to show that Q_1 and Q_2 avoid l' and l. Since P_1 avoids l' and P_2 avoids l, we have $l' \neq w \neq l$, Q_1 avoids l' and Q_2 avoids l. We show now that Q_1 also avoids l. Assume the hypernormal cycle C_1 visits l, then it must use the tree edge $e = \{l, r\}$. As C_1 is simple, P_1 ends with e. Recall that Q_1 ends with w. Since $w \neq l$ and $w \neq r$ (w is a leaf), the prefix Q_1 of P_1 ends before l is visited by P_1. A similar argument proves that Q_2 avoids l'. Thus C is a simple hypernormal cycle in $D(x)_u$. □

7. Testing for Hypernormal Cycles. Now we show how to test for the presence of hypernormal cycles in a dominance graph in polynomial time. This immediately gives us a polynomial algorithm for testing solvability (and hence, configurability) of dominance graphs.

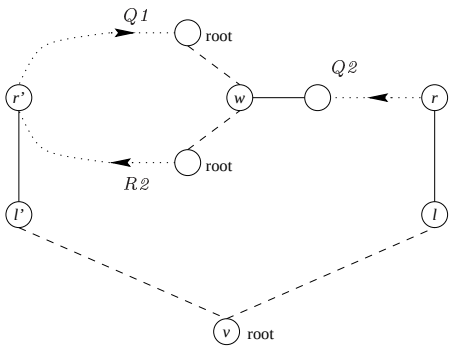


FIG. 6.2. *Construction of a simple hypernormal cycle in the proof of Theorem 6.2.*

We reduce the problem to a matching problem in an auxiliary graph. First, we want to recall some basic definitions from matching theory. A *matching* M in a graph H is a set of edges of H such that every node of H is incident to at most one edge in M. We call the edges in M *matching edges* and the other edges *non-matching edges*. An *alternating path* with respect to M is a (simple) path which alternately uses a matching edge and a non-matching edge. An *alternating cycle* is a cycle (of even length) which is an alternating path.

For the test we construct the following auxiliary graph A. For every edge $e = \{v, w\} \in G_u$ we have two nodes $e_v = (\{v, w\}, v)$ and $e_w = (\{v, w\}, w)$ in A. Before we define the edges of A, we want to introduce some more definitions. Let v denote a node of G_u, we call a pair of distinct edges $e = \{u, v\}$ and $f = \{v, w\}$ incident to v a *bend at* v and denote it by $\langle e, v, f\rangle$. The bend is called a *hypernormal bend* if either v is not a leaf or v is a leaf and either e or f is the tree edge incident to v. Now we are ready to define the edge set of A. We have two types of edges:

(a) For every edge $e = \{v, w\}$ we have the edge $a(e) = \{e_v, e_w\}$.

(b) For every hypernormal bend $\langle e, v, f\rangle$ we have the edge $b(\langle e, v, f\rangle) = \{e_v, f_v\}$.

Clearly, the edges of type (a) form a perfect matching M in A. The following lemma shows how hypernormal cycles in G_u are related to the auxiliary graph A:

LEMMA 7.1. *The graph G_u contains a hypernormal cycle iff the graph A contains an alternating cycle with respect to M.*

Proof. Suppose first that G_u contains a hypernormal cycle C. We may assume that C is simple. Every pair of consecutive edges on C is a hypernormal bend. Suppose $C = e_0 \circ e_1 \circ \ldots \circ e_{n-1}$, where $e_i = \{v_i, v_{(i+1) \bmod n}\}$ for $i = 0, \ldots, n-1$. Then $C' = a(e_0) \circ b(\langle e_0, v_1, e_1\rangle) \circ a(e_1) \circ \ldots \circ a(e_{n-1}) \circ b(\langle e_{n_1}, v_0, e_0\rangle)$ is an alternating cycle in A.

Suppose next that A contains an alternating cycle C'. Look at an edge of type (b) and its neighboring edges of type (a): $a(e) \circ b(\langle f, v, g\rangle) \circ a(h)$. From the construction of A we can conclude that either $e = f$ and $h = g$, or $e = g$ and $h = f$. Hence $\langle e, v, h\rangle$ is a hypernormal bend at node v. So if we delete all the edges of type (b) from C' we get a sequence $a(e_0) \circ \ldots \circ a(e_{n-1})$ of type (a) edges. Then $C = e_0 \circ \ldots \circ e_{n-1}$ is an edge-simple cycle and any pair of consecutive edges is a hypernormal bend. Now fix any leaf l visited by C. Every hypernormal bend at l contains the tree edge incident to l, and since C is edge-simple, we can conclude that C visits l only once and hence contains only one dominance edge incident to l. This proves that C is a hypernormal cycle in G_u. □

Gabow et al. [GKT01] gave an algorithm which can decide whether A contains an alternating cycle with respect to M in time $O(m')$, where m' is the number of edges in A. Now, we want to bound the size of the auxiliary graph A. We assume that all non-leaves in the dominance graph G have outdegree at most two[8]. Observe that we have one edge of type (a) for every edge of G. For the edges of type (b) we count the number of hypernormal bends at a node v. For a leaf we get outdeg_v, and for a non-leaf we have $\binom{\text{deg}_v}{2}$. Thus the auxiliary graph A has $n' = 2m$ nodes and $m' = O(m + \sum_{v \in V}(\text{indeg}_v + 2)^2)$ edges. Let us assume that G is transitively reduced. Then we have no parallel edges, and hence a root r with indegree greater than n must have two dominance edges from different leaves of the same tree to r, which is trivial to recognize in time $O(m)$. So we can assume that the indegree of any root is at most n. Let us say that we have $r \leq n$ roots and let d_i be the indegree of the i-th root. We have $\sum_{i=1}^{r} d_i \leq m$ and $d_i \leq n$. What is the maximum value of $S = \sum_i (2 + d_i)^2$? We have $S = O(n+m) + \sum_i d_i^2$. The sum $\sum_i d_i^2$ is maximized if we make the d_is as unequal as possible. So we attain the maximum if we set m/n of the d_is equal to n and all others equal to zero. Thus $\sum_{v \in V}(2 + \text{indeg}_v)^2 = O(n + m) + O(m/n \cdot n^2) = O(nm)$. The dominance graphs that arise in our linguistic applications have the following properties: $m = O(n)$ and the indegrees are bounded (the outdegrees are not), and hence the auxiliary graph has $n' = O(n)$ nodes and $m' = O(n)$ edges.

We summarize the results of this chapter in the following theorem.

THEOREM 7.2. *The existence of a hypernormal cycle in a dominance graph can be decided in time $O(m')$, where $m' = O(m) + \sum_{v \in V}(2 + \text{indeg}_v)^2 = O(nm)$.*

8. Efficient Enumeration. A first application of the solvability test from the previous section is to make the enumeration of solved forms more efficient. We modify the enumeration algorithm from Section 5 by testing for (undirected) hypernormal cycles in step 2 instead of directed arbitrary cycles. The recursion will terminate immediately once the graph becomes unconfigurable, and we know that the recursion depth is bounded by n^2. Thus:

COROLLARY 8.1. *A solved form of a solvable dominance graph can be constructed in time $O(n^3 m)$. If a dominance graph has N minimal solved forms, they can be enumerated in time $O(Nn^3 m)$.*

Note that N can still be exponential in n. Also note that we can get configurations instead of solved forms in the same asymptotic time, by applying Lemma 4.4: Simplification Rule 1 can only be applied at most n^2 times either, by a similar argument about the reachability relation.

8.1. Incremental Redundancy Elimination. The enumeration algorithm in Section 5 has to maintain the transitive reduction of the dominance graph G. This can be done in time $O(nm)$ (see [GK79, Sim88]). But for all recursive calls of the algorithm the reduction can be computed much faster, only the top-level needs to do the full-fledged reduction. This is because the instances on which recursive calls work are just reduced graphs where one irredundant edge has been added.

So we are faced with the following problem: We are given a reduced dominance graph G and an irredundant dominance edge $d = (s, t)$ which is not contained in G, and we are to compute all edges of G which become redundant by the insertion of d into G. An edge $e = (v, w)$ of G becomes redundant iff there is a path P from v to

[8] We can replace each non-leaf with outdegree more than two and its children by a small binary tree. This construction increases the number of nodes and the number of edges only by a constant factor.

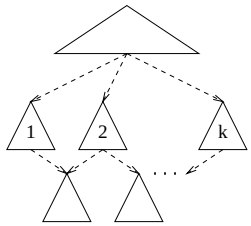


FIG. 9.1. *Embedded chain of length k.*

k	N	Time (new)	Time (old)
3	5	20	180
4	14	190	670
5	42	1210	5900
6	132	4130	12740
7	429	16630	46340
8	1430	255000	n/a

FIG. 9.2. *Runtimes on embedded chains of length k. N is the respective number of configurations. Times are in milliseconds CPU time.*

w in the graph $G \cup d$. Since G is reduced, P must use the edge d. And hence e is redundant iff in G there is a path from v to s and a path from t to w. Further, we observe that if $G \cup d$ is cyclic then any cycle must use the edge d. And hence $G \cup d$ is cyclic iff there is a node v such that in G there is a path from v to s and a path from t to v. Thus we can make $G \cup d$ reduced and test its acyclicity with the following algorithm:

- Start a depth first search in G at the node t and color all reachable nodes *red*.
- Start a depth first search in G_{rev} at the node s and color all reachable nodes *green*. (G_{rev} is obtained from G by reversing all the edges.)
- If there is a two-colored node v, report that $G \cup d$ is cyclic and stop.
- Delete all edges with *red* target node and *green* source node, and insert the edge d.

It is easy to see that the running time of the algorithm is $O(n + m)$. Note that this improvement does not lead to a better asymptotic running time of the enumeration algorithm, but it has shown considerable impact in practice.

9. Implementation and Evaluation. Going back to the application in computational linguistics described in the introduction, the algorithm for enumerating solved forms that we have just sketched gives us a straightforward algorithm for enumerating the minimal solved forms[9] of a normal dominance constraint φ: We only need to run it on $G(\varphi)$ and translate the solved forms back to solved forms of the constraint. We have implemented this algorithm, and this gives us a significant improvement in runtimes over earlier solvers for dominance constraints. By way of example, consider the dominance graph in Figure 9.1. This graph is an *embedded chain* of length k. Such graphs appear in the application; for instance, the graph for "John says that every linguist speaks two languages" is an embedded chain of length 2. Runtimes for enumerating all configurations of embedded chains of various lengths (on a 550 MHz Pentium III) are displayed in Figure 9.2. In the table, "new" refers to the algorithm

[9] A solved form φ' of φ is called *minimal* iff $G(\varphi')$ is a minimal solved form of $G(\varphi)$.

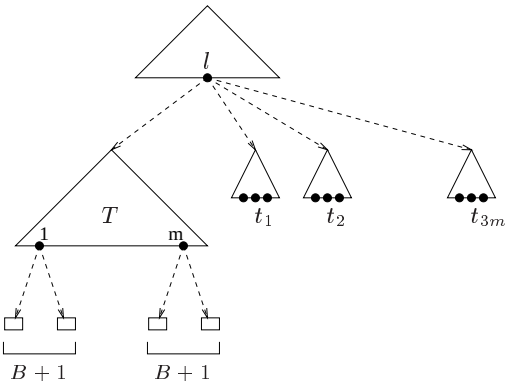


FIG. 10.1. *The dominance graph constructed in the reduction of 3-partition.*

sketched above; "old" refers to the dominance constraint solver described in [DG01].

10. Dominance Graphs with Closed Leaves. A slight extension of the configuration problem by *closed leaves* becomes NP-complete again. This is interesting in its own right because it shows where the frontier between polynomial and NP-complete is. It is also interesting in the application to computational linguistics because some approaches interpret graphs as in Fig. 2.3 as dominance graphs with closed leaves instead of dominance constraints.

A dominance graph with closed leaves is given by a dominance graph $G = (V, E \dot\cup D)$ and a set L of leaves. The members of L are called *closed*, all other leaves are called *open*. Closed leaves cannot be the source of dominance edges. A solved form of (G, L) with closed leaves L is a solved form $G' = (V, E \cup D')$ of G which has the additional property that there is no edge $(l, v) \in E'$ with $l \in L$, but there is an edge $(l, v) \in D'$ for every $l \notin L$. In other words, it is not allowed to attach a tree to a closed leaf, and every open leaf must be "plugged" with some other tree. We show that the configuration problem of dominance graphs with closed leaves is NP-complete by reducing the 3-partition problem to it.[10]

FACT 1 (3-partition). *Let A denote a multiset $\{a_1, \ldots, a_{3m}\}$ of integers and $B \in \mathbb{N}$ such that $B/4 < a_i < B/2$ for all i; and $\sum_{i=1}^{3m} a_i = mB$. The question is whether there is a partition $A_1 \uplus \ldots \uplus A_m$ of A such that for all i, $\sum_{a \in A_i} a = B$. The problem is NP-complete in the strong sense [GJ79, problem SP15, page 224].*

We describe the reduction now, which is shown in Figure 10.1. The tree T has m leaves. Each leaf wants to dominate $B+1$ closed subtrees (i.e., subtrees which have only closed leaves). T is required to be the child of some node l. This node l also wants to dominate the trees $t_1, \ldots, t_{3m}$. For all i, the tree t_i has $a_i + 1$ open leaves.

THEOREM 10.1. *The configurability problem for dominance graphs with closed leaves is NP-complete.*

Proof. Consider an instance (A, B) of the 3-partition problem and the dominance graph G constructed in the reduction. We show that the instance (A, B) has a solution iff G is configurable.

Assume first that the 3-partition problem has a solution. Observe that each of the sets A_i must have cardinality three. Let $A_i = \{a_{x_i}, a_{y_i}, a_{z_i}\}$ be one of the sets

[10] Exactly the same reduction works if we do not require open leaves to have outgoing dominance edges in solved forms; so this modified problem is NP-complete as well.

in the partition. Then $a_{x_i} + a_{y_i} + a_{z_i} = B$. We plug t_{x_i} as child into the i-th leaf of T, t_{y_i} into some leaf of t_{x_i} and t_{z_i} into some leaf of t_{y_i}. Then the tree T has $a_{x_i} + 1 + a_{y_i} + 1 + a_{z_i} + 1 - 2 = B + 1$ open leaves below its i-th leaf. These leaves are plugged with the $B + 1$ closed subtrees which the i-th leaf of T wants to dominate. Finally, we plug l with T and obtain a configuration of G.

Assume next that the dominance graph G has a configuration. Consider the subtree plugged to the i-th leaf of T. It contains a subset A_i of the trees $\{t_1, \ldots, t_{3m}\}$. We must have $\sum_{t_j \in A_i}(a_j + 1) \geq B + 1 + |A_i| - 1$, which can be seen as follows. $B + 1$ closed subtrees must be plugged into some open leaf. Every subtree in A_i also requires an open leaf where its root can be plugged. And one of these leaves is the i-th leaf of T.

We next show that $|A_i| \geq 3$ for all i. It is clear that A_i cannot be empty (since $B > 0$). If A_i is a singleton, i.e. $A_i = \{t_x\}$, we have a contradiction since t_x has $a_x + 1 < B/2 + 1 \leq B + 1$ leaves. Now consider the case, where A_i consists of two elements t_x and t_y. By attaching t_x and t_y below the i-th leaf of T, we obtain $a_x + 1 + a_y + 1 - 1 < B/2 + 1 + B/2 = B + 1$ open leaves, which is also a contradiction. Since each set A_i has cardinality at least three, since we have m sets, and since there are $3m$ elements to distribute, we conclude that $|A_i| = 3$ for all i. Thus $\sum_{t_j \in A_i} a_j \geq B$ for every i. Finally, we observe that we have equality since $\sum_{a \in A} a = mB$. Thus we also have a solution for the 3-partition problem. □

Note that for *solvability* of dominance graphs with closed leaves, Theorem 6.2 still holds. That is, solvability is still a polynomial problem. The difference with the unrestricted problem is that Lemma 4.4 breaks down: All the graphs we construct in the encoding of 3-partition are in solved form, but they may well be unconfigurable.

The relevance of this result is again in its relation to computational linguistics. There are alternative approaches to scope [Bos96] which require that the holes and roots of the trees must be paired uniquely: The roots must be "plugged" into the holes, and every hole must be plugged. This corresponds to making the holes open leaves, and all others closed leaves. Hence, we can show that the satisfiability problems of these alternative approaches must be NP-complete as well.

11. Conclusion. We have have distinguished the large and natural fragment of normal dominance constraints and shown that it has a polynomial time satisfiability problem. We have shown that this satisfiability problem is equivalent to the configuration problem of dominance graphs, we introduced. Configurability was then reduced to the existence of particular cycles, which could finally be checked by solving a weighted matching problem.

The efficient graph algorithm we presented eliminates any doubts about the computational practicability of dominance constraints which were raised by the NP-completeness result for the general language [KNT01] and expressed e.g. in [WM99]. First experiments confirm the efficiency of the new algorithm – it is superior to the NP algorithms especially on larger constraints.

Two main directions are to be pursued in the future. On the one hand side, one might want to extend the presented graph algorithm to more expressive languages than normal dominance constraints. In particular, one might try to find a polynomial time fragment of parallelism constraints [EKN02] that are also useful for computational linguistics [EKN01]. On the other hand side, it might be worthwhile in general to investigate graph algorithms for other problems in the areas constraint programming or computational linguistics.

[AC92] H. Alshawi and R. Crouch. Monotonic semantic interpretation. In *Proceedings of the 30th Annual Meeting of the Association of Computational Linguistics*, pages 32–39, 1992.

[ADK+01] E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. An efficient algorithm for the configuration problem of dominance graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 815–824, 2001.

[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.

[CFS97] A. Copestake, D. Flickinger, and I. Sag. Minimal Recursion Semantics. An Introduction. Manuscript, `ftp://csli-ftp.stanford.edu/linguistics/sag.ps`, 1997.

[Cor94] Thomas Cornell. On determining the consistency of partial descriptions of trees. In *Proceedings of the 32th Annual Meeting of the Association of Computational Linguistics*, pages 163–170, 1994.

[DG01] Denys Duchier and Claire Gardent. Tree descriptions, constraints and incrementality. In Harry Bunt, Reinhard Muskens, and Elias Thijsse, editors, *Computing Meaning, Volume 2*, volume 77 of *Studies In Linguistics And Philosophy*, pages 205–227. Kluwer Academic Publishers, December 2001.

[DN00] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic*, number 1861 in Lecture Notes in Computer Science, pages 326–341. Springer-Verlag, July 2000.

[DT99] Denys Duchier and Stefan Thater. Parsing with tree descriptions: a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming*, pages 17–32, December 1999.

[EKN01] Markus Egg, Alexander Koller, and Joachim Niehren. The constraint language for lambda structures. *Journal of Logic, Language, and Information*, 10:457–485, 2001.

[EKN02] Katrin Erk, Alexander Koller, and Joachim Niehren. Processing underspecified semantic representations in the constraint language for lambda structures. *Research on Language and Computation*, 1(1), 2002. In Press.

[ENRX98] Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *Proceedings of the Joint 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association of Computational Linguistics*, pages 353–359, 1998.

[GJ79] M.R. Garey and D.S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H. Freeman and Company, 1979.

[GK79] A. Goralcikova and V. Koubek. A reduct-and-closure algorithm for graphs. In *Proceedings of the 8th Symposium on Mathematical Foundations of Computer Science*, volume 74 of *LNCS*, pages 301–307. Springer, 1979.

[GKT01] H.N. Gabow, H. Kaplan, and R.E. Tarjan. Unique maximum matching algorithms. *Journal of Algorithms*, 40:159–183, 2001.

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

[Hob83] Jerry Hobbs. An improper treatment of quantification in ordinary english. In *Proceedings of the 21st annual meeting of the Association of Computational Linguistics*, pages 57–63, 1983.

[KMN00] Alexander Koller, Kurt Mehlhorn, and Joachim Niehren. A polynomial-time fragment of dominance constraints. In *Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics*, pages 368–375, October 2000.

[KNT01] Alexander Koller, Joachim Niehren, and Ralf Treinen. Dominance constraints: Algorithms and complexity. In *Proceedings of the Third International Conference on Logical Aspects of Computational Linguistics (Dec. 1998)*, volume 2014 of *Lecture Note in Artificial Intelligence*, pages 106–125. Springer-Verlag, 2001.

[MHF83] Mitchell P. Marcus, Donald Hindle, and Margaret M. Fleck. D-theory: Talking about talking about trees. In *Proceedings of the 21st annual meeting of the Association of Computational Linguistics*, pages 129–136, 1983.

[MN99] Kurt Mehlhorn and Stefan Näher. *LEDA: A platform for combinatorial and geometric computing*. Cambridge University Press, Cambridge, November 1999.

[MNP00] Martin Müller, Joachim Niehren, and Andreas Podelski. Ordering constraints over feature trees. *Constraints, an International Journal*, 5(1–2):7–42, January 2000. Special Issue on CP'97.

[Mus95] 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.

[Per00] Guy Perrier. From intuitionistic proof nets to interaction grammars. In *Proceedings of the 5th TAG+ Workshop*, Paris, 2000.

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

[RVSW95] O. Rambow, K. Vijay-Shanker, and D. Weir. D-Tree grammars. In *Proceedings of the 33rd annual meeting of the Association of Computational Linguistics*, pages 151–158, 1995.

[Sim88] Klaus Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58(1–3):325–346, 1988.

[Smo92] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.

[ST94] Gert Smolka and Ralf Treinen. Records for logic programming. *Journal of Logic Programming*, 18(3):229–258, April 1994.

[TW67] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, August 1967.

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

[VSWR95] K. Vijay-Shanker, David Weir, and Owen Rambow. Parsing D-Tree Grammars. In *Proceedings of the International Workshop on Parsing Technologies*, 1995.

[WM99] A. Willis and S. Manandhar. Two accounts of scope availability and semantic underspecification. In *Proceedings of the 37th annual meeting of the Association of Computational Linguistics*, pages 293–300, 1999.