Dominance Constraints with Set Operators

Denys Duchier and Joachim Niehren

Programming Systems Lab, Universität des Saarlandes Saarbrücken

Abstract. Dominance constraints are widely used in computational linguistics as a language for talking and reasoning about trees. In this paper, we extend dominance constraints by admitting set operators. We present a solver for dominance constraints with set operators, which is based on propagation and distribution rules, and prove its soundness and completeness. From this solver, we derive an implementation in a constraint programming language with finite sets and prove its faithfullness.

1 Introduction

The dominance relation of a tree is the ancestor relation between its nodes. Logical descriptions of trees via dominance were investigated in computer science since the beginning of the sixties, for instance in the logics (W)SkS [15, 16]. In computational linguistics, the importance of dominance based tree descriptions for deterministic parsing was discovered at the beginning of the eighties [9]. Since then, tree descriptions based on dominance constraints have become increasingly popular [14, 1]. Meanwhile, they are used for tree-adjoining and D-tree grammars [17, 13, 3], for underspecified representation of scope ambiguities in semantics [12, 4] and for underspecified descriptions of discourse structure [5].

A dominance constraint describes a finite tree by conjunctions of literals with variables for nodes. A dominance literal $x \triangleleft^* y$ requires x to denote one of the ancestors of the denotation of y. A labeling literal $x{:}f(x_1, \ldots, x_n)$ expresses that the node denoted by x is labeled with symbol f and has the sequence of children referred to by $x_1, \ldots, x_n$. Solving dominance constraints is an essential service required by applications in e.g. semantics and discourse. Even though satisfiability of dominance constraints is NP-complete [8], it appears that dominance constraints occurring in these applications can be solved rather efficiently [2, 7].

For a typical application of dominance constraints in semantic underspecification of scope we consider the sentence: *every yogi has a guru.* This sentence is semantically ambiguous, even though its syntactic structure is uniquely determined. The trees in Figure 1 specify both meanings: either there exists a common guru for every yogi, or every yogi has his own guru. Both trees (and thus meanings) can be represented in an underspecified manner through the dominance constraint in Figure 2.

In this paper, we propose to extend dominance constraints by admitting set operators: union, intersection, and complementation can be applied to the relations of dominance $\triangleleft^*$ and inverse dominance $\triangleright^*$. Set operators contribute a controlled form of disjunction and negation that is eminently well-suited for

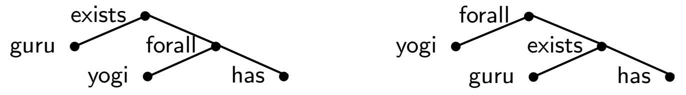
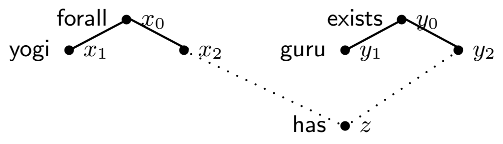


Fig. 1. Sets of trees represent sets of meanings.



$$x_0\text{:forall}(x_1, x_2) \wedge$$
$$y_0\text{:exists}(y_1, y_2) \wedge$$
$$x_1\text{:yogi} \wedge x_2 \lhd^* z \wedge$$
$$y_1\text{:guru} \wedge y_2 \lhd^* z \wedge$$
$$z\text{:has}$$

Fig. 2. A single tree description as underspecified representation of all meanings.

constraint propagation while less expressive than general Boolean connectives. Set operators allow to express proper dominance, disjointness, nondisjointness, nondominance, and unions thereof. Such a rich set of relations is important for specifying powerful constraint propagation rules for dominance constraints as we will argue in the paper.

We first present a system of abstract saturation rules for propagation and distribution, which solve dominance constraints with set operators. We illustrate the power of the propagation rules and prove soundness, completeness, and termination in nondeterministic polynomial time. We then derive a concrete implementation in a constraint programming language with finite sets [11, 6] and prove its faithfulness to the abstract saturation rules. The resulting solver is not only well suited for formal reasoning but also improves in expressiveness on the saturation based solver for pure dominance constraints of [8] and produces smaller search trees than the earlier set based implementation of [2] because it requires less explicit solved forms. For omitted proofs, we globally refer to the extended version of this paper available from `http://www.ps.uni-sb.de/Papers/`.

2 Dominance Constraints

We first define tree structures and then dominance constraints with set operators which are interpreted in the class of tree structures. We assume a signature Σ of 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. We are interested in finite constructor trees that can be seen as ground terms over Σ such as $f(g(a, b))$ in Fig. 3.

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

A *(finite) constructor tree* τ is a triple (V, E, L) consisting of a tree (V, E), and *labelings* $L : V \to \Sigma$ for nodes and $L : E \to N$ for edges, such that any node $u \in V$ has exactly one outgoing edge with label k for each $1 \leq k \leq \mathsf{ar}(\sigma(\pi))$, and no other outgoing edges. We draw constructor trees as in Fig. 3, by annotating nodes with their labels and ordering the edges by increasing labels from left to right. If $\tau = (V, E, L)$, we write $V_\tau = V$, $E_\tau = E$, $L_\tau = L$.

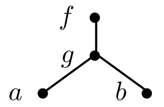


Fig. 3. $f(g(a, b))$

Definition 1. *The* tree structure $\mathcal{M}^\tau$ *of a finite constructor tree* τ *over* Σ *is the first-order structure with domain* V_τ *which provides the* dominance relation $\lhd^{*\tau}$ *and a* labeling relation *of arity* $\mathsf{ar}(f) + 1$ *for each function symbol* $f \in \Sigma$. *These relations are defined such that for all* $u, v, u_1, \ldots, u_n \in V_\tau$*:*

$$\begin{array}{ll} u \lhd^{*\tau} v & \textit{iff there is a path from } u \textit{ to } v \textit{ with egdes in } E_\tau; \\ u{:}f^\tau(v_1, \ldots, v_n) & \textit{iff } L_\tau(u) = f, \mathsf{ar}(f) = n, \textit{ and } L(u, v_i) = i \textit{ for all } 1 \leq i \leq n \end{array}$$

We consider the following *set operators* on binary relations: inversion $^{-1}$, union $\cup$, intersection $\cap$, and complementation $\neg$. We write $\rhd^{*\tau}$ for the inverse of dominance $\lhd^{*\tau}$, equality $=^\tau$ for the intersection $\lhd^{*\tau} \cap \rhd^{*\tau}$, inequality $\neq^\tau$ for the complement of equality, proper dominance $\lhd^{+\tau}$ as dominance but not equality, $\rhd^{+\tau}$ for the inverse of proper dominance, and disjointness $\perp^\tau$ for $\neg\lhd^{*\tau} \cap \neg\rhd^{*\tau}$. Most importantly, the following partition holds in all tree structures $\mathcal{M}^\tau$.

$$V_\tau \times V_\tau \quad = \quad \uplus\{=^\tau, \lhd^{+\tau}, \rhd^{+\tau}, \perp^\tau\}$$

Thus, all relations that set operators can generate from dominance $\lhd^{*\tau}$ have the form $\cup\{r^\tau \mid r \in R\}$ for some set of relation symbols $R \subseteq \{=, \lhd^+, \rhd^+, \perp\}$.

For defining the constraint language, we let x, y, z range over an infinite set of node variables. A *dominance constraints with set operators* φ has the following abstract syntax (that leaves set operators implicit).

$$\varphi ::= x\, R\, y \;\mid\; x{:}f(x_1, \;\ldots\;, x_n) \;\mid\; \varphi \wedge \varphi' \;\mid\; \mathsf{false}$$

where $R \subseteq \{=, \lhd^+, \rhd^+, \perp\}$ is a set of relation symbols and $n = \mathsf{ar}(f)$. Constraints are interpreted in the class of tree structures over Σ. For instance, a constraint $x\ \{=, \perp\}\ y$ expresses that the nodes denoted by x and y are either equal or lie in disjoint subtrees. In general, a set R of relation symbols is interpreted in $\mathcal{M}^\tau$ as the union $\cup\{r^\tau \mid r \in R\}$.

We write $\mathsf{Vars}(\varphi)$ for the set of variables occurring in φ. A *solution* of a constraint φ consists of a tree structure $\mathcal{M}^\tau$ and a variable assignment $\alpha : \mathsf{Vars}(\varphi) \to V_\tau$. We write $(\mathcal{M}^\tau, \alpha) \models \varphi$ if all constraints of φ are satisfied by $(\mathcal{M}^\tau, \alpha)$ in the usual Tarskian sense. For convenience we admit syntactic sugar and allow to write constraints of the form xSy where S is a set expression:

$$S ::= R \mid = \mid \lhd^* \mid \rhd^* \mid = \mid \neq \mid \lhd^+ \mid \rhd^+ \mid \perp \mid \neg S \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S^{-1}$$

Clearly, every set expression S can be translated to a set R of relation symbols denoting the same relation. In all tree structures, $x\, \neg S\, y$ is equivalent to $\neg\; x\, S\, y$ and $x\, S_1 \cup S_2\, y$ to $x\, S_1\, y \vee x\, S_2\, y$. Thus our formalism allows a controlled form of negation and disjunction without admitting full Boolean connectives.

Propagation Rules:

(Clash)	$x\emptyset y \quad\rightarrow\quad \mathbf{false}$	
(Dom.Refl)	$\varphi \quad\rightarrow\quad x\lhd^* x$	(x occurs in φ)
(Dom.Trans)	$x\lhd^* y \wedge y\lhd^* z \quad\rightarrow\quad x\lhd^* z$	
(Eq.Decom)	$x{:}f(x_1,\ldots,x_n) \wedge y{:}f(y_1,\ldots,y_n) \wedge x{=}y \quad\rightarrow\quad \bigwedge_{i=1}^n x_i{=}y_i$	
(Lab.Ineq)	$x{:}f(\ldots) \wedge y{:}g(\ldots) \quad\rightarrow\quad x{\neq}y$	if $f \neq g$
(Lab.Disj)	$x{:}f(\ldots,x_i,\ldots,x_j,\ldots) \quad\rightarrow\quad x_i\perp x_j$	where $1 \leq i < j \leq n$
(Lab.Dom)	$x{:}f(\ldots,y,\ldots) \quad\rightarrow\quad x\lhd^+ y$	
(Inter)	$xR_1y \wedge xR_2y \quad\rightarrow\quad xRy$	if $R_1{\cap}R_2 \subseteq R$
(Inv)	$xRy \quad\rightarrow\quad yR^{-1}x$	
(Disj)	$x\perp y \wedge y\lhd^* z \quad\rightarrow\quad x\perp z$	
(NegDisj)	$x\lhd^* z \wedge y\lhd^* z \quad\rightarrow\quad x\neg\perp y$	
(Child.up)	$x\lhd^* y \wedge x{:}f(x_1,\ldots,x_n) \wedge \bigwedge_{i=1}^n x_i\neg\lhd^* y \quad\rightarrow\quad y{=}x$	

Distribution Rules:

(Distr.Child)	$x\lhd^* y \wedge x{:}f(x_1,\ldots,x_n) \quad\rightarrow\quad x_i\lhd^* y \vee x_i\neg\lhd^* y$	$(1 \leq i \leq n)$
(Distr.NegDisj)	$x\neg\perp y \quad\rightarrow\quad x\lhd^* y \vee x\neg\lhd^* y$	

Fig. 4. Saturation rules D of the Base Solver

3 A Saturation Algorithm

We now present a solver for dominance constraints with set operators. First, we give a base solver which saturates a constraint with respect to a set of propagation and distribution rules, and prove soundness, completeness, and termination of saturation in nondeterministic polynomial time. Second, we add optional propagation rules, which enhance the propagation power of the base solver.

The base solver is specified by the rule schemes in Figure 4. Let D be the (infinite) set of rules instantiating these schemes. Each rule is an implication between a constraint and a disjunction of constraints. We distinguish *propagation* rules $\varphi_1 \rightarrow \varphi_2$ which are deterministic and *distribution* rules $\varphi_1 \rightarrow \varphi_2 \vee \varphi_3$ which are nondeterministic.

Proposition 1 (Soundness). *The rules of D are valid in all tree structures.*

The inference system D can be interpreted as a saturation algorithm which decides the satisfiability of a constraint. A propagation rule $\varphi_1{\rightarrow}\varphi_2$ applies to a constraint φ if all atomic constraints in φ_1 belong to φ but at least one of the atomic constraints in φ_2 does not. In this case, saturation proceeds with $\varphi \wedge \varphi_2$. A distribution rule $\varphi_1{\rightarrow}\varphi_2 \vee \varphi_3$ applies to a constraint φ if both rules $\varphi_1{\rightarrow}\varphi_2$ and $\varphi_1{\rightarrow}\varphi_3$ could be applied to φ. In this case, one of these two rules is non-deterministically chosen and applied. A constraint is called D-saturated if none of the rules in D can be applied to it.

Proposition 2 (Termination). *The maximal number of iterated D-saturation steps on a constraint is polynomially bounded in the number of its variables.*

Proof. Let φ be a constraint with m variables. Each D-saturation step adds at least one new literal to φ. Only a $O(m^2)$ literals can be added since all of them have the form xRy where $x, y \in \mathsf{Vars}(\varphi)$ and R has 16 possible values.

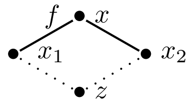


Fig. 5. (Neg.Disj)

Next, we illustrate prototypical inconsistencies and how D-saturation detects them. We start with the constraint $x{:}f(x_1, x_2) \wedge x_1 \lhd^* z \wedge x_2 \lhd^* z$ in Fig. 5 which is unsatisfiable since siblings cannot have a common descendant. Indeed, the disjointness of the siblings $x_1 \perp x_2$ can be derived from (Lab.Disj) whereas $x_1 \neg\perp x_2$ follows from (NegDisj) since x_1 and x_2 have the common descendant z.

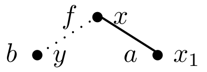


Fig. 6. (Distr.Child)

To illustrate the first distribution rule, we consider the unsatisfiable constraint $x{:}f(x_1) \wedge x \lhd^* y \wedge x_1{:}a \wedge y{:}b$ in Fig. 6 where $a \neq b$. We can decide the position of y with respect to x by applying rule (Distr.Child) which either adds $x_1 \lhd^* y$ or $x_1 \neg\lhd^* y$. (1) If $x_1 \neg\lhd^* y$ is added, propagation with (Child.up) yields $x{=}y$. As x and y carry distinct labels, rule (Lab.Ineq) adds $x{\neq}y$. Now, we can deduce $x\emptyset y$ by intersecting equality and inequality (Inter). Thus, the (Clash) rule applies. (2) If $x_1 \lhd^* y$ is added then (Child.up) yields $x_1{=}y$ which again clashes because of distinct labels.

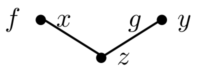


Fig. 7. (Distr.NegDisj)

The second distribution rules helps detect the inconsistency of $x{:}f(z) \wedge y{:}g(z)$ in Fig 7 where $f{\neq}g$. In a first step one can infer from (Lab.Dom) that $x \lhd^+ z$ and $y \lhd^+ z$. As the (Inter) rule allows to weaken relations, we also have $x \lhd^* z$ and $y \lhd^* z$, i.e. $x \neg\perp y$ by (NegDisj), so that (Distr.NegDisj) can deduce either $x \lhd^* y$ or $x \neg\lhd^* y$. Consider the case $x \lhd^* y$, from $y \lhd^+ z$ derive $z \neg\lhd^* y$ by (Inv, Inter), and (Child.up) infers $y{=}x$ resulting in a clash due to the distinct labels. Similarly for the other case.

Definition 2. *A* D-solved form *is a D-saturated constraint without* false.

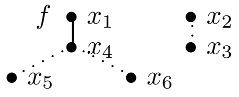


Fig. 8. D-solved form

The intuition is that a D-solved form has a backbone which is a *dominance forest*, i.e. a forest with child and dominance edges. For instance, Fig 8 shows the dominance forest underlying $x_1{:}f(x_4) \wedge x_4 \lhd^* x_5 \wedge x_4 \lhd^* x_6 \wedge x_2 \lhd^* x_3 \wedge x_5 \perp x_6$ which becomes D-solved when D-propagation.

We would like to note that the set based solver for dominance constraints of [2] insists on more explicit solved forms: for each two variables, one of the relations $\{=, \lhd^+, \rhd^+, \perp\}$ must be selected. For the dominance forest in Fig. 8, this leads to 63 explicit solutions instead of a single D-solved form. The situation is even worse for the formula $x_1 \lhd^* x_2 \wedge x_2 \lhd^* x_3 \wedge \ldots \wedge x_{n-1} \lhd^* x_n$. This constraint can be deterministically D-solved by D-propagation whereas the implementation of [2] computes a search tree of size 2^n.

Proposition 3 (Completeness). *Every D-solved form has a solution.*

(Child.down) $x \lhd^+ y \wedge x{:}f(x_1,\ldots,x_n) \wedge \bigwedge_{i=1,i\neq j}^{n} x_i \neg\lhd^* y \quad\rightarrow\quad x_j \lhd^* y$
(NegDom) $\quad x \perp y \wedge y \neg\perp z \quad\rightarrow\quad x \neg\lhd^* z$

Fig. 10. Some Optional Propagation Rules O

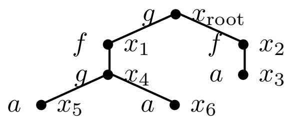


Fig. 9. A solution.

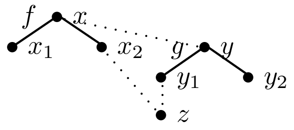
The proof is given in the Section 4. The idea for constructing a solution of a D-solved form is to turn its underlying dominance forest into a tree, by adding labels such that dominance children are placed at disjoint positions whenever possible. For instance, a solution of the dominance forest in Fig. 8 is drawn in Fig. 9. Note that this solution does also satisfy $x_5 \perp x_6$ which belongs to the above constraint but not to its dominance forest. This solution is obtained from the dominance forest in Fig. 8 by adding a root node and node labels by which all dominance edges are turned into child edges.

Theorem 1. *Saturation by the inference rules in D decides the satisfiability of a dominance constraint with set operators in non-deterministic polynomial time.*

Proof. Let φ be a dominance constraint with set operators. Since all rules in D are sound (Proposition 1) and terminate (Proposition 2), φ is equivalent to the disjunction of all D-solved forms reachable from φ by non-deterministic D-saturation. Completeness (Proposition 3) yields that φ is satisfiable iff there exists a D-solved reachable from φ.

We can reduce the search space of D-saturation by adding optional propagation rules O. Taking advantage of set operators, we can define rather powerful propagation rules. The schemes in Fig 10, for instance, exploit the complementation set operators, and are indeed supported by the set based implementation of Section 6. For lack of space, we omit further optional rules that can be expressed by set operators and are also implemented by our solver. Instead, we illustrate O in the situation below which arises naturally when resolving scope ambiguities as in Figure 2.

$x{:}f(x_1,x_2) \wedge y{:}g(y_1,y_2) \wedge x_2 \lhd^* z \wedge y_1 \lhd^* z \wedge x \lhd^* y$



We derive $x \lhd^+ y$ by (Lab.Ineq,Inter), $x_1 \perp x_2$ by (Lab.Disj), and $x_2 \neg\perp y$ by (Dom. Trans,NegDisj). We combine the latter two using optional rule (NegDom) into $x_1 \neg\lhd^* y$. Finally, optional rule (Child.down) yields $x_2 \lhd^* y$ whereby the situation is resolved.

4 Completeness Proof

We now prove Proposition 3 which states completeness in the sense that every D-solved form is satisfiable. We proceed in two steps. First, we identify *simple*

D-solved forms and show that they are satisfiable (Proposition 4). Then we show how to extend every D-solved form into a simple D-solved form by adding further constraints (Proposition 5).

Definition 3. *A variable x is* labeled in φ *if* $x{=}y$ in φ *and* $y{:}f(y_1,\ldots,y_n)$ in φ *for some variable y and term $f(y_1,\ldots,y_n)$. A variable y is a* root variable *for φ if* $y{\lhd^*}z$ in φ *for all $z \in \mathsf{Vars}(\varphi)$. We call a constraint φ* simple *if all its variables are labeled, and if there is a root variable for φ.*

Proposition 4. *A simple D-solved form is satisfiable.*

Proof. By induction on the number of literals in a simple D-solved form φ. φ has a root variable z. Since all variables in φ are labeled there is a variable z' and a term $f(z_1,\ldots,z_n)$ such that $z{=}z' \wedge z'{:}f(z_1,\ldots,z_n) \in \varphi$. We pose:

$$V = \{x \in \mathsf{Vars}(\varphi) \mid x{=}z \in \varphi\} \text{ and } V_i = \{x \in \mathsf{Vars}(\varphi) \mid z_i{\lhd^*}x \in \varphi\}$$

for all $1 \leq i \leq n$. To see that $\mathsf{Vars}(\varphi)$ is covered by $V \cup V_1 \cup \ldots \cup V_n$, let $x \in \mathsf{Vars}(\varphi)$ such that $z_i{\lhd^*}x \notin \varphi$ for all $1 \leq i \leq n$. Saturation with (Distr.Child) derives either $z_i{\lhd^*}x$ or $z_i{\neg\lhd^*}x$; but $z_i{\lhd^*}x \notin \varphi$ by assumption, therefore $z_i{\neg\lhd^*}x \in \varphi$ for all $1 \leq i \leq n$. (Child.up) infers $z{=}x \in \varphi$, i.e. $x \in V$. For a set $W \subseteq \mathsf{Vars}(\varphi)$ we define $\varphi_{|W}$ to be the conjunction of all literals $\psi \in \varphi$ with $\mathsf{Vars}(\psi) \subseteq W$.

$$\varphi \models\!\mid \varphi' \quad \text{holds where} \quad \varphi' =_{\text{def}} \varphi_{|V} \wedge z{:}f(z_1,\ldots,z_n) \wedge \varphi_{|V_1} \wedge \ldots \wedge \varphi_{|V_n}$$

$\varphi \models \varphi'$ follows from $\varphi' \subseteq \varphi$. To show $\varphi' \models \varphi$ we prove that each literal in φ is entailed by φ'

1. Case $x{:}g(x_1,\ldots,x_m) \in \varphi$ for some variable x and term $g(x_1,\ldots,x_m)$: If $x \in V_i$, i.e. $z_i{\lhd^*}x \in \varphi$ for some $1 \leq i \leq n$ then $x{:}g(x_1,\ldots,x_m) \in \varphi_{|V_i}$ since φ is saturated under (Lab.Dom, Dom.Trans). Otherwise $x \in V$, i.e. $z{=}x \in \varphi$, and thus $z{=}x \in \varphi_{|V}$. Since φ is clash free and saturated under (Lab.Ineq,Clash), $f{=}g$ and $n{=}m$ must hold. Saturation with respect to (Eq.Decom) implies $z_i{=}x_i \in \varphi$ for all $1 \leq i \leq n$ and hence $z_i{=}x_i \in \varphi_{|V_i}$. All together, the right hand side φ' contains $z{=}x \wedge z{:}f(z_1,\ldots,z_n) \wedge \wedge_{i=1}^{n} z_i{=}x_i$ which entails $x{:}g(x_1,\ldots,x_m)$ as required.
2. Case $xRy \in \varphi$ for some variables x, y and relation set $R \subseteq \{=, \lhd^+, \rhd^+, \perp\}$. Since $x, y \in V \cup V_1 \cup \ldots \cup V_n$ we distinguish 4 possibilities:
   (a) $x \in V_i$, $y \in V_j$, where $1 \leq i \neq j \leq n$. Here, $x{\perp}y \in \varphi$ by saturation under (Lab.Disj, Inv, Disj). Clash-freeness and saturation under (Inter, Clash) yield $\perp \in R$. Finally, φ' entails $z_i{\perp}z_j$ and thus $x{\perp}y$ which in turn entails xRy.
   (b) When $x, y \in V$ (resp. V_i), by definition $xRy \in \varphi_{|V}$ (resp. $\varphi_{|V_i}$)
   (c) $x \in V$ and $y \in V_i$. Here, $x{\lhd^+}y \in \varphi$ by saturation under (Lab.Dom, Dom.Trans). Thus $\lhd^+ \in R$ by saturation under (Inter, Clash) and clash-freeness of φ. But φ' entails $z{\lhd^+}z_i$ and thus $x{\lhd^+}y$ which in turn entails xRy.
   (d) The case $x \in V$ and $y \in V_i$ is symmetric to the previous one.

Next note that all $\varphi_{|V_i}$ are simple D-solved forms. By induction hypothesis there exist solutions $(\mathcal{M}^{\tau_i}, \alpha_i) \models \varphi_{|V_i}$ for all $1 \leq i \leq n$. Thus $(\mathcal{M}^{f(\tau_1,\ldots,\tau_n)}, \alpha)$ is a solution of φ if $\alpha_{|V_i} = \alpha_i$ and $\alpha(x) = \alpha(z)$ is the root node of $f(\tau_1, \ldots, \tau_n)$ for all $x \in V$. $\square$

An *extension of a constraint* φ is a constraint of the form $\varphi \wedge \varphi'$ for some φ'. Given a constraint φ we define a partial ordering $\prec_\varphi$ on its variables such that $x \prec_\varphi y$ holds if and only if $x \triangleleft^* y$ *in* φ but not $y \triangleleft^* x$ *in* φ. If x is unlabeled then we define the set $\mathsf{con}_\varphi(x)$ of variables *connected to* x *in* φ as follows:

$$\mathsf{con}_\varphi(x) = \{y \mid y \text{ is } \prec_\varphi \text{ minimal with } x \prec_\varphi y\}$$

Intuitively, a variable y is connected to x if it is a "direct dominance child" of x. So for example, $\mathsf{con}_{\varphi_1}(x) = \{y\}$ and $\mathsf{con}_{\varphi_1}(y) = \{z\}$ for:

$$\varphi_1 := x \triangleleft^* x \wedge x \triangleleft^* y \wedge x \triangleleft^* z \wedge y \triangleleft^* z,$$

Definition 4. *We call* $V \subseteq \mathsf{Vars}(\varphi)$ *a* φ-disjointness *set if for any two distinct variables* $y_1, y_2 \in V$, $y_1 \neg\perp y_2$ not in φ.

The idea is that all variables in a φ-disjointness set can safely be placed at disjoint positions in at least one of the trees solving φ.

Lemma 1. *Let* φ *be D-saturated,* $x \in \mathsf{Vars}(\varphi)$*. If* V *is a maximal* φ*-disjointness set in* $\mathsf{con}_\varphi(x)$ *then for all* $y \in \mathsf{con}_\varphi(x)$ *there exists* $z \in V$ *such that* $y{=}z$ in φ*.*

Proof. If $y \neg\perp z$ *not in* φ for all $z \in V$ then $\{y\} \cup V$ is a disjointness set; thus $y \in V$ by maximality of V. Otherwise, there exists $z \in V$ such that $y \neg\perp z$ *in* φ. Saturation of φ with respect to rules (Distr.NegDisj, Inter) yields $y \triangleleft^* z$ *in* φ or $z \triangleleft^* y$ *in* φ. In both cases, it follows that $z{=}y$ *in* φ since z and y are both $\prec_\varphi$ minimal elements in the set $\mathsf{con}_\varphi(x)$.

Lemma 2 (Extension by Labeling). *Every D-solved form* φ *with an unlabeled variable* x *can be extended to a D-solved form with strictly fewer unlabeled variables, and in which* x *is labeled.*

Proof. Let $\{x_1, \ldots, x_n\}$ be a maximal φ-disjointness set included in $\mathsf{con}_\varphi(x)$. Let f be a function symbol of arity n in Σ, which exists w.l.o.g. Otherwise, f can be encoded from a constant and a symbol of arity ≥ 2 whose existence in Σ we assumed. We define the following extension $\mathsf{ext}(\varphi)$ of φ:

$$\begin{aligned}\mathsf{ext}(\varphi) =_{\text{def}}\ & \varphi \wedge x{:}f(x_1, \ldots, x_n) \wedge \\ & \textstyle\bigwedge \{xRz \wedge zR^{-1}x \mid \triangleleft^+ \in R,\ x_i \triangleleft^* z \ in\ \varphi,\ 1 \leq i \leq n\} \wedge \qquad (1) \\ & \textstyle\bigwedge \{yRz \mid \perp \in R,\ x_i \triangleleft^* y \ in\ \varphi,\ x_j \triangleleft^* z \ in\ \varphi,\ 1 \leq i {\neq} j \leq n\} \qquad (2)\end{aligned}$$

Note that x is labeled in $\mathsf{ext}(\varphi)$ since $x{=}x \in \varphi$ by saturation under (Dom.Refl). We have to verify that $\mathsf{ext}(\varphi)$ is D-solved, i.e. that none of the D-rules can be applied to $\mathsf{ext}(\varphi)$. We give the proof only for two of the more complex cases.

1. (Distr.Child) cannot be applied to $x{:}f(x_1, \dots, x_n)$: suppose $x \lhd^* y$ *in* φ and consider the case $y \lhd^* x$ *not in* φ. Thus $x \prec_\varphi y$ and there exists $z \in \mathsf{con}_\varphi(x)$ with $z \lhd^* y$ *in* φ. Lemma 1 and the maximality of the φ-disjointness set $\{x_1, \dots, x_n\}$ yield $x_j{=}z$ *in* φ for some $1 \leq j \leq n$. Thus, $x_j \lhd^* y$ *in* φ by (Dom.Trans) and (Distr.Child) cannot be applied with x_j. For all such $1 \leq i \neq j \leq n$ we can derive $x_i \perp y$ by (Lab.Dom, Disj, Inv), thus $x_i \neg\lhd^* y$ by (Inter) and (Distr.Child) cannot be applied with x_i either.
2. (Inter) applies when $R_1 \cap R_2 \subseteq R$, yR_1z *in* $\mathsf{ext}(\varphi)$, and yR_2z *in* $\mathsf{ext}(\varphi)$. We prove yRz *in* $\mathsf{ext}(\varphi)$ for the case where yR_1z *in* φ and yR_2z is contributed to $\mathsf{ext}(\varphi)$ by (2). Thus, $\perp \in R_2$ and there exists $1 \leq i{\neq}j \leq n$ such that $x_i \lhd^* y$ *in* φ and $x_j \lhd^* z$ *in* φ. It is sufficient to prove $\perp \in R_1$ since then $\perp \in R_1 \cap R_2 \subseteq R$ which implies yRz *in* φ. We assume $\perp \notin R_1$ and derive a contradiction. If $\perp \notin R_1$ then $R_1 \subseteq \{=, \lhd^+, \rhd^+\}$. Thus, weakening yR_1z *in* φ with (Inter) yields $y \neg\perp z$ *in* φ. Next, we can apply (Distr.NegDisj) which proves either $y \lhd^* z$ *in* φ or $y \neg\lhd^* z$ *in* φ.
   (a) If $y \lhd^* z$ *in* φ then $x_i \lhd^* z$ *in* φ follows from (Dom.Trans) and $x_i \neg\perp x_j$ *in* φ from (NegDisj). This contradicts our assumption that $\{x_1, \dots, x_n\}$ is a φ-disjointness set.
   (b) If $y \neg\lhd^* z$ *in* φ then we have $y \neg\lhd^* z$ *in* φ and $y \neg\perp z$ *in* φ from which on can derive $y \rhd^* z$ *in* φ with (Inter) and $z \lhd^* y$ *in* φ with (Inv). From (Dom.Trans) we derive $x_j \lhd^* y$ *in* φ. Since we already know $x_j \lhd^* y$ *in* φ we can apply (NegDisj) which shows $x_i \neg\perp x_j$ *in* φ. But again, this contradicts that $\{x_1, \dots, x_n\}$ is a φ-disjointness set. □

Proposition 5. *Every D-solved form can be extended to a simple D-solved form.*

Proof. Let φ be D-solved. W.l.o.g., φ has a root variable, else we choose a fresh variable x and consider instead the D-solved extension $\varphi \wedge \bigwedge\{xRy \wedge yR^{-1}x \mid \lhd^+ \in R,\ y \in \mathsf{Vars}(\varphi)\}$. By Lemma 2, we can successively label all its variables. □

5 Constraint Programming with Finite Sets

Current constraint programming technology provides no support for our D-saturation algorithm. Instead, improving on [2], we reformulate the task of finding solutions of a tree description as a constraint satisfaction problem solvable by constraint programming [11, 6]. In this section, we define our target language. Its propagation rules are given in Fig 12 and are used in proving correctness of implementation. Distribution rules, however, are typically problem dependent and we assume that they can be programmatically stipulated by the application. Thus, the concrete solver of Section 6 specifies its distribution rules in Figure 13.

Let $\Delta = \{1 \ \dots \ \mu\}$ be a finite set of integers for some large practical limit μ such as 134217726. We assume a set of *integer variables* with values in Δ and ranged over by I and a set of *set variables* with values in 2^Δ and ranged over by S. Integer and finite set variables are also both denoted by X.

$$\begin{array}{l} \mathcal{B} ::= \mathsf{false} \mid X_1{=}X_2 \mid I \in D \mid i \in S \mid i \notin S \qquad (D \subseteq \Delta) \\ \mathcal{C} ::= \mathcal{B} \mid S_1 \cap S_2{=}\emptyset \mid S_3 \subseteq S_1 \cup S_2 \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \mathcal{C}_1 \,\mathbf{or}\, \mathcal{C}_2 \end{array}$$

Fig. 11. Finite Domain and Finite Set Constraints

Equality: $$X_1{=}X_2 \wedge \mathcal{B}[X_j] \quad \rightsquigarrow_{\mathrm{P}} \quad \mathcal{B}[X_k] \qquad \{j,k\} = \{1,2\} \qquad \text{(eq.subst)}$$

Finite domain integer constraints:

$$\begin{array}{rcll} I \in D_1 \wedge I \in D_2 & \rightsquigarrow_{\mathrm{P}} & I \in D_1 \cap D_2 & \text{(fd.conj)} \\ I \in \emptyset & \rightsquigarrow_{\mathrm{P}} & \mathsf{false} & \text{(fd.clash)} \end{array}$$

Finite sets constraints:

$$\begin{array}{rclll} i \in S \wedge i \notin S & \rightsquigarrow_{\mathrm{P}} & \mathsf{false} & & \text{(fs.clash)} \\ S_1 \cap S_2{=}\emptyset \wedge i \in S_j & \rightsquigarrow_{\mathrm{P}} & i \notin S_k & \{j,k\} = \{1,2\} & \text{(fs.disjoint)} \\ S_3 \subseteq S_1 \cup S_2 \wedge i \notin S_1 \wedge i \notin S_2 & \rightsquigarrow_{\mathrm{P}} & i \notin S_3 & & \text{(fs.subset.neg)} \\ S_3 \subseteq S_1 \cup S_2 \wedge i \in S_3 \wedge i \notin S_j & \rightsquigarrow_{\mathrm{P}} & i \in S_k & \{j,k\} = \{1,2\} & \text{(fs.subset.pos)} \end{array}$$

Disjunctive propagators:

$$\frac{\mathcal{B} \wedge \mathcal{C} \quad \rightsquigarrow_{\mathrm{P}}^{\circledast} \quad \mathsf{false}}{\mathcal{B} \wedge (\mathcal{C} \,\mathbf{or}\, \mathcal{C}') \quad \rightsquigarrow_{\mathrm{P}} \quad \mathcal{C}'} \qquad \frac{\mathcal{B} \wedge \mathcal{C}' \quad \rightsquigarrow_{\mathrm{P}}^{\circledast} \quad \mathsf{false}}{\mathcal{B} \wedge (\mathcal{C} \,\mathbf{or}\, \mathcal{C}') \quad \rightsquigarrow_{\mathrm{P}} \quad \mathcal{C}} \qquad \text{(commit)}$$

Fig. 12. Propagation Rules

The abstract syntax of our language is given in Fig 11. We distinguish between *basic constraints* $\mathcal{B}$, directly representable in the constraint store, and *non-basic constraints* $\mathcal{C}$ acting as propagators and amplifying the store. The declarative semantics of these constraints is obvious (given that $\mathcal{C}_1 \,\mathbf{or}\, \mathcal{C}_2$ is interpreted as disjunction). We write $\beta \models \mathcal{C}$ if β is an assignment of integer variables to integers and set variables to sets which renders $\mathcal{C}$ true (where set operators and Boolean connectives have the usual meaning).

We use the following abbreviations: we write $I{\neq}i$ for $I \in \Delta \setminus \{i\}$, $S_1 \parallel S_2$ for $S_1 \cap S_2{=}\emptyset$, $S{=}D$ for $\wedge\{i \in S \mid i \in D\} \wedge \{i \notin S \mid i \in \Delta \setminus D\}$, $S_1 \subseteq S_2$ for $S_1 \subseteq S_2 \cup S_3 \wedge S_3{=}\emptyset$, and $S = S_1 \uplus S_2$ for $S_1 \parallel S_2 \wedge S \subseteq S_1 \cup S_2 \wedge S_1 \subseteq S \wedge S_2 \subseteq S$

The propagation rules $\rightsquigarrow_{\mathrm{P}}$ for inference in this language are summarized in Fig 12. The expression $\mathcal{C}_1 \,\mathbf{or}\, \mathcal{C}_2$ operates as a *disjunctive propagator* which does not invoke any case distinction. The propagation rules for disjunctive propagators use the saturation relation $\rightsquigarrow_{\mathrm{P}}^{\circledast}$ induced by $\rightsquigarrow_{\mathrm{P}}$ which in turn is defined by recursion through $\rightsquigarrow_{\mathrm{P}}^{\circledast}$. Clearly, all propagation rules are valid formulas when seen as implications or as implications between implications in case of (commit).

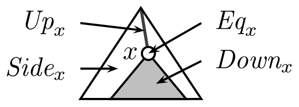
6 Reduction to Finite Set Constraints

We now reduce dominance constraints with set operators to finite set constraints of the language introduced above. This reduction yields a concrete implementation of the abstract dominance constraint solver when realized in a constraint programming system such as [11, 6].

The underlying idea is to represent a literal $x\,R\,y$ by a membership expression $y{\in}R(x)$ where $R(x)$ is a set variable denoting a finite set of nodes in a tree. This idea is fairly general in that it does not depend on the particular relations interpreting the relation symbols. Our encoding consists of 3 parts:

$$[\![\varphi]\!] = \bigwedge_{x\in\mathsf{Vars}(\varphi)} \mathsf{A}_1(x) \bigwedge_{x,y\in\mathsf{Vars}(\varphi)} \mathsf{A}_2(x,y) \quad\wedge\quad \mathsf{B}[\![\varphi]\!]$$

$\mathsf{A}_1(\,\cdot\,)$ introduces a node representation per variable, $\mathsf{A}_2(\,\cdot\,)$ axiomatizes the treeness of the relations between these nodes, and $\mathsf{B}[\![\varphi]\!]$ encodes the specific restrictions imposed by φ.



Representation. When observed from a specific node x, the nodes of a solution tree (hence the variables that they interpret) are partitioned into 4 regions: x itself, all nodes above, all nodes below, and all nodes to the side. The main idea is to introduce corresponding set variables.

Let MAX be the maximum constructor arity used in φ. For each formal variable x in φ we choose a distinct integer ι_x to represent it, and introduce $7+$ MAX constraint set variables written Eq_x, Up_x, $Down_x$, $Side_x$, $Equp_x$, $Eqdown_x$, $Parent_x$, $Down_x^i$ for $1 \le i \le \text{MAX}$, and one constraint integer variable $Label_x$. First we state that $x = x$:

$$\iota_x \in Eq_x \tag{3}$$

Eq_x,Up_x,$Down_x$,$Side_x$ encode the set of variables that are respectively equal, above, below, and to the side (i.e. disjoint) of x. Thus, posing $\mathcal{I} = \{\iota_x \mid x \in \mathsf{Vars}(\varphi)\}$ for the set of integers encoding $\mathsf{Vars}(\varphi)$, we have:

$$\mathcal{I} = Eq_x \uplus Down_x \uplus Up_x \uplus Side_x$$

We can improve propagation by introducing $Eqdown_x$ and $Equp_x$ as intermediate results. This improvement is required by (Dom.Trans):

$$\mathcal{I} = Eqdown_x \uplus Up_x \uplus Side_x \tag{4}$$
$$\mathcal{I} = Equp_x \uplus Down_x \uplus Side_x \tag{5}$$
$$Eqdown_x = Eq_x \uplus Down_x \tag{6}$$
$$Equp_x = Eq_x \uplus Up_x \tag{7}$$

$Down_x^i$ encodes the set of variables in the subtree rooted at x's ith child (empty if there is no such child):

$$Down_x = \uplus\{Down_x^i \mid 1 \le i \le \text{MAX}\} \tag{8}$$

We define $\mathsf{A}_1(x)$ as the conjunction of the constraints introduced above:

$$\mathsf{A}_1(x) = (3) \wedge (4) \wedge (5) \wedge (6) \wedge (7)$$

Wellformedness. Posing $\mathbf{Rel} = \{=, \lhd^+, \rhd^+, \perp\}$. In a tree, the relationship that obtains between the nodes denoted by x and y must be one in $\mathbf{Rel}$. We introduce an integer variable C_{xy}, called a choice variable, to explicitly represent it and contribute a well-formedness clause $\mathsf{A}_3[\![x\ r\ y]\!]$ for each $r \in \mathbf{Rel}$. Freely indentifying the symbols in $\mathbf{Rel}$ with the integers 1,2,3,4, we write:

$$\mathsf{A}_2(x,y) \quad = \quad C_{xy} \in \mathbf{Rel} \wedge \bigwedge \{\mathsf{A}_3[\![x\ r\ y]\!] \mid r \in \mathbf{Rel}\} \tag{9}$$

$$\mathsf{A}_3[\![x\ r\ y]\!] \quad \equiv \quad \mathsf{D}[\![x\ r\ y]\!] \wedge C_{xy} = r \ \textbf{ or } \ C_{xy} \neq r \wedge \mathsf{D}[\![x\ \neg r\ y]\!] \tag{10}$$

For all $r \in \mathbf{Rel}$, it remains to define $\mathsf{D}[\![x r y]\!]$ and $\mathsf{D}[\![x \neg r y]\!]$ encoding the relations $x\ r\ y$ and $x\ \neg r\ y$ resp. by set constraints on the representations of x and y.

$$\begin{array}{rcl}
\mathsf{D}[\![x = y]\!] & = & \mathit{Eq}_x = \mathit{Eq}_y \wedge \mathit{Up}_x = \mathit{Up}_y \wedge \mathit{Down}_x = \mathit{Down}_y \wedge \mathit{Side}_x = \mathit{Side}_y \\
 & & \wedge \mathit{Eqdown}_x = \mathit{Eqdown}_y \wedge \mathit{Equp}_x = \mathit{Equp}_y \\
 & & \wedge \mathit{Parent}_x = \mathit{Parent}_y \wedge \mathit{Label}_x = \mathit{Label}_y \underset{i}{\wedge} \mathit{Down}^i_x = \mathit{Down}^i_y \\
\mathsf{D}[\![x \neg\!= y]\!] & = & \mathit{Eq}_x \parallel \mathit{Eq}_y \\
\mathsf{D}[\![x \lhd^+ y]\!] & = & \mathit{Eqdown}_y \subseteq \mathit{Down}_x \wedge \mathit{Equp}_x \subseteq \mathit{Up}_y \wedge \mathit{Side}_x \subseteq \mathit{Side}_y \\
\mathsf{D}[\![x \neg\lhd^+ y]\!] & = & \mathit{Eq}_x \parallel \mathit{Up}_y \wedge \mathit{Down}_x \parallel \mathit{Eq}_y \\
\mathsf{D}[\![x \perp y]\!] & = & \mathit{Eqdown}_x \subseteq \mathit{Side}_y \wedge \mathit{Eqdown}_y \subseteq \mathit{Side}_x \\
\mathsf{D}[\![x \neg\perp y]\!] & = & \mathit{Eq}_x \parallel \mathit{Side}_y \wedge \mathit{Side}_x \parallel \mathit{Eq}_y
\end{array}$$

Problem specific constraints. The third part $\mathsf{B}[\![\varphi]\!]$ of the translation forms the additional problem-specific constraints that further restrict the admissibility of wellformed solutions and only accept those that are models of φ. The translation is given by clauses (11,12,13).

$$\mathsf{B}[\![\varphi \wedge \varphi']\!] \quad = \quad \mathsf{B}[\![\varphi]\!] \wedge \mathsf{B}[\![\varphi']\!] \tag{11}$$

A pleasant consequence of the introduction of choice variables C_{xy} is that any dominance constraint $x\ R\ y$ can be translated as a restriction on the possible values of C_{xy}. For example, $x \lhd^* y$ can be encoded as $C_{xy} \in \{1,2\}$. More generally:

$$\mathsf{B}[\![x\ R\ y]\!] \quad = \quad C_{xy} \in R \tag{12}$$

Finally the labelling constraint $x : f(y_1\ \ldots\ y_n)$ requires a more complex treatment. For each constructor f we choose a distinct integer ι_f to encode it.

$$\begin{array}{rcl}
\mathsf{B}[\![x : f(y_1\ \ldots\ y_n)]\!] & = & \mathit{Label}_x = \iota_f \wedge_{j=n+1}^{j=\text{MAX}} \mathit{Down}^j_x = \emptyset \\
 & & \wedge_{j=1}^{j=n} \mathit{Parent}_{y_j} = \mathit{Eq}_x \wedge \mathit{Down}^j_x = \mathit{Eqdown}_{y_j} \wedge \mathit{Up}_{y_j} = \mathit{Equp}_x
\end{array} \tag{13}$$

Definition of The Concrete Solver. For each problem φ we define a search strategy specified by the distribution rules of Figure 13. These rules correspond precisely to (Distr.Child, Distr.NegDisj) of algorithm-D and are to be applied in the same non-deterministic fashion. Posing $\rightsquigarrow\ =\ \rightsquigarrow_{\text{P}} \cup \rightsquigarrow_{\text{D}}$, we define our concrete solver as the non-deterministic saturation $\rightsquigarrow^{\circledast}$ induced by $\rightsquigarrow$ and write $\varphi_1 \rightsquigarrow^{\circledast} \varphi_2$ to mean that φ_2 is in a $\rightsquigarrow^{\circledast}$ saturation of φ_1. While the abstract solver left this point open, in order to avoid unnecessary choices, we further require that a $\rightsquigarrow_{\text{D}}$ step be taken only if no $\rightsquigarrow_{\text{P}}$ step is possible.

$$\begin{array}{lll} C_{xy} \in \{=, \lhd^+\} & \rightsquigarrow_{\mathrm{D}} & C_{x_i y} \in \{=, \lhd^+\} \vee C_{x_i y} \notin \{=, \lhd^+\} \quad \text{for } x{:}f(x_1, \dots, x_n) \text{ in } \varphi \\ C_{xy} \neq \bot & \rightsquigarrow_{\mathrm{D}} & C_{xy} \in \{=, \lhd^+\} \vee \ C_{xy} \notin \{=, \lhd^+\} \end{array}$$

Fig. 13. Problem specific distribution rules

7 Proving Correctness of Implementation

We now prove that $[\![\varphi]\!]$ combined with the search strategy defined above yields a sound and complete solver for φ. Completeness is demonstrated by showing that the concrete solver obtained by $[\![\varphi]\!]$ provides at least as much propagation as specified by the rules of algorithm D, i.e. whenever $x\,R\,y$ is in a $\rightarrow^{\circledast}$ saturation of φ then $C_{xy} \in R$ is in a $\rightsquigarrow^{\circledast}$ saturation of $[\![\varphi]\!]$.

Theorem 2. *$[\![\varphi]\!]$ is satisfiable iff φ is satisfiable.*

This follows from Propositions 6 and 7 below.

Proposition 6. *if φ is satisfiable then $[\![\varphi]\!]$ is satisfiable.*

We show how to construct a model β of $[\![\varphi]\!]$ from a model $(\mathcal{M}^\tau, \alpha)$ of φ. We define the variable assignment β as follows: $\beta(Up_x) = \{\iota_y \mid \alpha(y) \lhd^+ \alpha(x)\}$ and similarly for $Eq_x, Down_x, Side_x, Eqdown_x, Equp_x$, $\beta(Parent_x) = \{\iota_y \mid \exists k\ \alpha(y)k = \alpha(x)\}$, $\beta(Down^k_x) = \{\iota_y \mid \alpha(y) \rhd^* \alpha(x)k\}$, $\beta(Label_x) = \iota_{L_\tau(\alpha(x))}$ and $\beta(C_{xy}) = R$ if $\alpha(x)\ R\ \alpha(y)$ in $\mathcal{M}^\tau$. We have that if $(\mathcal{M}^\tau, \alpha) \models \varphi$ then $\beta \models [\![\varphi]\!]$.

Proposition 7. *if $[\![\varphi]\!]$ is satisfiable, then φ is satisfiable.*

We prove this by reading a D-solved form off a model β of $[\![\varphi]\!]$.

$$\varphi' \quad \equiv \quad \varphi \wedge \bigwedge_{x,y} \bigwedge_{R' \supseteq R} x\ R'\ y \qquad \text{where } R = \beta(C_{xy})$$

φ' is a D-solved form containing φ: all relationships between variables are fully resolved and all their generalizations have been added. The only possibility is that D-rules might derive a contradiction. However, if $\varphi' \rightarrow^{\circledast}_{\mathrm{P}} \mathsf{false}$ then $[\![\varphi']\!] \rightsquigarrow^{\circledast}_{\mathrm{P}} \mathsf{false}$ (Lemma 4) which would contradict the existence of a solution β. Therefore φ' is a O-solved form and φ is satisfiable.

We distinguish propagation and distribution rules; in algorithm D they are written $\rightarrow_{\mathrm{P}}$ and $\rightarrow_{\mathrm{D}}$, and in our concrete solver $\rightsquigarrow_{\mathrm{P}}$ and $\rightsquigarrow_{\mathrm{D}}$. We write $\varphi'' \preccurlyeq \varphi'$ for φ'' *is stronger than* φ' and define it as the smallest relation that holds of atomic constraints and such that $\mathsf{false} \preccurlyeq \mathsf{false}$ and $x\ R\ y \ \preccurlyeq\ x\ R'\ y$ iff $R \subseteq R'$.

Proposition 8 (Stronger Propagation). *For each rule $\varphi \rightarrow_{\mathrm{P}} \varphi'$ of algorithm D, there exists $\varphi'' \preccurlyeq \varphi'$ such that $[\![\varphi]\!] \rightsquigarrow^{\circledast}_{\mathrm{P}} [\![\varphi'']\!]$.*

The proof technique follows this pattern: each φ' is of the form $x\ R\ y$ and we choose $\varphi'' = xR'y$ where $R' \subseteq R$. Assume $[\![\varphi]\!]$ as a premise. Show that $[\![\varphi]\!] \wedge \mathcal{C} \rightsquigarrow^{\circledast}_{\mathrm{P}} \mathsf{false}$. Notice that a clause $\mathcal{C}$ **or** $\mathcal{C}'$ is introduced by $[\![\varphi]\!]$ as required by (10). Thus $\mathcal{C}'$ follows by (commit). Then show that $[\![\varphi]\!] \wedge \mathcal{C}' \rightsquigarrow^{\circledast}_{\mathrm{P}} [\![\varphi'']\!]$. For want of space, we include here only the proof for rule (NegDisj).

Lemma 3. $[\![x \lhd^* y]\!] \leadsto_{\mathrm{P}}^{\circledast} \iota_y \in \mathit{Eqdown}_x$ *(proof omitted)*

Proposition 9. $[\![x \lhd^* z \wedge y \lhd^* z]\!] \leadsto_{\mathrm{P}}^{\circledast} [\![x \neg\bot\, y]\!]$

Proof. From the premises $[\![x \lhd^* z]\!]$ and $[\![y \lhd^* z]\!]$, i.e. $C_{xz} \in \{=, \lhd^+\}$ and $C_{yz} \in \{=, \lhd^+\}$, we must show $[\![x \neg\bot\, y]\!]$ i.e. $C_{xy} \neq \bot$. By Lemma 3 we obtain $\iota_z \in \mathit{Eqdown}_x$ and $\iota_z \in \mathit{Eqdown}_y$. Since $\mathcal{I} = \mathit{Eqdown}_y \uplus \mathit{Up}_y \uplus \mathit{Side}_y$, we have $\iota_z \notin \mathit{Side}_y$. Now consider the non-basic constraint $\mathit{Eqdown}_x \subseteq \mathit{Side}_y$ which occurs in $\mathsf{D}[\![x \perp y]\!]$: from $\iota_z \in \mathit{Eqdown}_x$ it infers $\iota_z \in \mathit{Side}_y$ which contradicts $\iota_z \notin \mathit{Side}_y$. Therefore, the well-formedness clause $\mathsf{D}[\![x \perp y]\!] \wedge C_{xy} = \bot$ **or** $C_{xy} \neq \bot \wedge \mathsf{D}[\![x \neg\bot\, y]\!]$ infers its right alternative by rule (commit). Hence $C_{xy} \neq \bot$ □

Lemma 4. *(1) if $\varphi \rightarrow_{\mathrm{P}}^{\circledast} \varphi'$, then there exists $\varphi'' \preccurlyeq \varphi'$ such that $[\![\varphi]\!] \leadsto_{\mathrm{P}}^{\circledast} [\![\varphi'']\!]$. (2) if $\varphi \rightarrow_{\mathrm{P}}^{\circledast} \varphi_1$ and $\varphi_1 \rightarrow_{\mathrm{D}} \varphi_2$, then there exists $\varphi_1' \preccurlyeq \varphi_1$ such that $[\![\varphi]\!] \leadsto_{\mathrm{P}}^{\circledast} [\![\varphi_1']\!]$ and $[\![\varphi_1']\!] \leadsto_{\mathrm{D}} [\![\varphi_2]\!]$.*

(1) follows from Proposition 8, and (2) from (1) and the fact that the concrete distribution rules precisely correspond to those of algorithm D.

Proposition 10 (Simulation). *The concrete solver simulates the abstract solver: if $\varphi \rightarrow^{\circledast} \varphi'$ then there exists $\varphi'' \preccurlyeq \varphi'$ such that $[\![\varphi]\!] \leadsto^{\circledast} [\![\varphi'']\!]$.*

Follows from Lemma 4.

Theorem 3. *(1) every $\leadsto^{\circledast}$ saturation of $[\![\varphi]\!]$ corresponds to a D-solved form of φ and (2) for every D-solved form of φ there is a corresponding $\leadsto^{\circledast}$ saturation of $[\![\varphi]\!]$.*

(1) from Proposition 10. (2) Consider a $\leadsto^{\circledast}$ saturation of $[\![\varphi]\!]$. As in Proposition 7, we can construct a D-solved form φ' of φ by reading off the current domains of the choice variables C_{xy}. If φ' was not D-solved, then $\rightarrow^{\circledast}$ could infer a new fact, but then by Proposition 10 so could $\leadsto^{\circledast}$ and it would not be a saturation.

8 Conclusion

In this paper, we extended dominance constraints by admitting set operators. Set operators introduce a controlled form of disjunction and negation that is less expressive than general Boolean connectives and remains especially well-suited for constraint propagation. On the basis of this extension we presented two solvers: one abstract, one concrete.

The design of the abstract solver is carefully informed by the needs of practical applications: it stipulates inference rules required for efficiently solving dominance constraints occurring in these applications. The rules take full advantage of the extra expressivity afforded by set operators. We proved the abstract solver sound and complete and that its distribution strategy improves over [2] and may avoid an exponential number of choice points. This improvement accrues from admitting less explicit solved forms while preserving soundness.

Elaborating on the technique first presented in [2], the concrete solver realizes the desired constraint propagation by reduction to constraint programming using set constraints. We proved that the concrete solver faithfully simulates the abstract one, and thereby shed new light on the source of its observed practical effectiveness. The concrete solver has been implemented in the concurrent constraint programming language Oz [10], performs efficiently in practical applications to semantic underspecification, and produces smaller search trees than the solver of [2].

References

1. 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.
2. D. Duchier and C. Gardent. A constraint-based treatment of descriptions. In *Int. Workshop on Computational Semantics*, Tilburg, 1999.
3. D. Duchier and S. Thater. Parsing with tree descriptions: a constraint-based approach. In *Int. Workshop on Natural Language Understanding and Logic Programming*, Las Cruces, New Mexico, 1999.
4. M. Egg, J. Niehren, P. Ruhrberg, and F. Xu. Constraints over lambda-structures in semantic underspecification. In *Joint Conf. COLING/ACL*, pages 353–359, 1998.
5. C. Gardent and B. Webber. Describing discourse semantics. In *Proceedings of the 4th TAG+ Workshop*, Philadelphia, 1998.
6. C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997.
7. A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. Technical report, Programming Systems Lab, Universität des Saarlandes, Apr. 2000. Submitted.
8. A. Koller, J. Niehren, and R. Treinen. Dominance constraints: Algorithms and complexity. In *Logical Aspects of Comp. Linguistics 98*, 2000. To appear in LNCS.
9. M. P. Marcus, D. Hindle, and M. M. Fleck. D-theory: Talking about talking about trees. In *21st ACL*, pages 129–136, 1983.
10. Mozart. The mozart programming system. `http://www.mozart-oz.org/`.
11. T. Müller and M. Müller. Finite set constraints in Oz. In F. Bry, B. Freitag, and D. Seipel, editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München, 1997.
12. R. Muskens. Order-Independence and Underspecification. In J. Groenendijk, editor, *Ellipsis, Underspecification, Events and More in Dynamic Semantics.* DYANA Deliverable R.2.2.C, 1995.
13. O. Rambow, K. Vijay-Shanker, and D. Weir. D-tree grammars. In *Proceedings of ACL'95*, pages 151–158, MIT, Cambridge, 1995.
14. J. Rogers and K. Vijay-Shanker. Reasoning with descriptions of trees. In *Annual Meeting of the Association for Comp. Linguistics (ACL)*, 1992.
15. 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.
16. W. Thomas. Automata on Infinite Objects. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science, Formal Models and Semantics*, volume B, chapter 4, pages 133–191. The MIT Press, 1990.
17. K. Vijay-Shanker. Using descriptions of trees in a tree adjoining grammar. *Computational Linguistics*, 18:481–518, 1992.