Dominance Constraints With Boolean Connectives: A Model-Eliminative Treatment

Denys Duchier

Programming System Lab, Universität des Saarlandes, Saarbrücken
duchier@ps.uni-sb.de

Abstract

Dominance constraints are a language of tree descriptions. Tree descriptions are widely used in computational linguistics for talking and reasoning about trees. While previous research has focused on the conjunctive fragment, we now extend the account to all Boolean connectives and propose a new formalism that combines dominance constraints with a feature tree logic.

Although the satisfiability problem in the conjunctive fragment is known to be NP-complete, we have previously demonstrated that it can be addressed very effectively by constraint propagation: we developed an encoding that transforms a dominance constraint into a constraint satisfaction problem on finite sets solvable by constraint programming. We present a generalization of this encoding for our more expressive formalism, and prove soundness and completeness. Our main contribution is a treatment of disjunction suitable for constraint propagation.

Key words: dominance constraints, tree descriptions, finite set constraints, constraint satisfaction, constraint propagation, constraint programming

1 Introduction

In computational linguistics, theories are frequently concerned with the formulation of constraints or principles restricting the admissibility of tree representations. A large class of structural constraints can be expressed elegantly in the form of tree descriptions, where the 'parent' relation may be relaxed into the 'ancestor', or dominance, relation. Tree descriptions were introduced in (Marcus et al., 1983), motivated by an application to deterministic parsing, and have steadily gained in popularity (Backofen et al., 1995; Rogers and Vijay-Shanker, 1992). Today, they are used in such varied domains as Tree-Adjoining and D-Tree Grammars (Vijay-Shanker, 1992; Rambow et al., 1995;

Duchier and Thater, 1999), for underspecified representation of scope ambiguities in semantics (Muskens, 1995; Egg et al., 1998), and for underspecified descriptions of discourse structure (Gardent and Webber, 1998).

Classical dominance constraints express a tree description as a conjunction of literals $x \lhd^* y$ and $x{:}f(x_1 \ \ldots \ x_n)$ where variables denote nodes in the tree. The symbol $\lhd^*$ notates the dominance relation and $x{:}f(x_1 \ \ldots \ x_n)$ expresses that the node denoted by x is formed from the n-ary constructor f and the sequence of daughter nodes denoted by x_1 through x_n.

While the satisfiability problem was shown to be NP-complete (Koller, Niehren, and Treinen, 2000b), for practical applications it remains essential to be able to decide satisfiability and find solutions of tree descriptions efficiently. This requirement may be addressed either by identifying polynomial fragments, e.g. *normal dominance constraints* (Koller et al., 2000a), or by devising solvers that more effectively deal with the combinatorial complexity of the task. An approach based on constraint propagation has proven particularly successful: efficient constraint programming solvers can be derived by transformation of a dominance constraint into a constraint satisfaction problem on finite sets (Duchier and Gardent, 1999; Duchier, 1999b; Duchier and Niehren, 2000).

Dominance constraints with set operators (Duchier and Niehren, 2000) generalized dominance literals $x \lhd^* y$ into $x\,R\,y$ for any $R \subseteq \{=, \lhd^+, \rhd^+, \perp\}$, where $\lhd^+$ denotes proper dominance and $\perp$ disjointness. R is called a set operator and is interpreted disjunctively, e.g. $x\,\{=,\perp\}\,y$ expresses that the nodes denoted by x and y must either be equal or lie in disjoint subtrees, and $x \lhd^* y$ is now written $x\,\{=,\lhd^+\}\,y$. In all tree structures, we have $x\,\neg R\,y \equiv \neg(x\,R\,y)$ and $x\,(R_1 \cup R_2)\,y \equiv x\,R_1\,y \vee x\,R_2\,y$. Thus the extended formalism allows a controlled form of negation and disjunction without admitting full Boolean connectives, yet remains eminently well-suited to processing based on constraint propagation.

In the present article, we extend the account to a language with Boolean connectives. Our main contribution is a treatment of disjunction suitable for constraint propagation by reduction to the *selection constraint* (Duchier, 1999a). In contrast with previous formalisms based on *constructor trees*, the one proposed here combines dominance constraints with a *feature tree logic* in the style of CFT (Smolka and Treinen, 1994). A literal previously written $x{:}f(x_1, \ldots, x_n)$ is now expressed as a conjunction of simpler constraints:

$$x{:}f(x_1, \ldots, x_n) \quad \equiv \quad x{:}f \ \wedge \ |x|{=}n \ \wedge \ \wedge_{i=1}^{n} x[i] = x_i$$

$|x|{=}n$ is an arity constraint, $x{:}f$ is a label or sort constraint, and $x[i]{=}x_i$ is a feature constraint. The finer granularity of the language facilitates the treatment of negation:

$$\neg\, x{:}f(x_1\ \ldots\ x_n) \quad\equiv\quad \neg\, x{:}f \ \vee\ |x|{\neq}n \ \vee\ \vee_{i=1}^{n} x[i] \neq x_i$$

It also allows us to generalize to all literals the constraint treatment made possible by the set-based disjunctive representation of *set operators*. Where $x\,\{=,\perp\}\,y$, now written $x = y \vee x \perp y$, was treated as constraining the relation holding between x and y to be an element of $\{=,\perp\}$, similarly $x{:}f \vee x{:}g$ can also be treated as constraining the label of x to be an element of $\{f, g\}$.

The increased expressivity permits in particular the direct formulation of constraints that previously required ad hoc extensions to the conjunctive fragment. For example, the *non-intervention* constraint $\neg(x \lhd^* y \lhd^* z)$ proposed by Koller and Niehren (2000) in their application to underspecified processing of dynamic semantics:

$$\neg(x \lhd^* y \lhd^* z) \quad\equiv\quad x \rhd^+ y \ \vee\ x \perp y \ \vee\ y \rhd^+ z \ \vee\ y \perp z$$

In Section 2, we present our formalism and develop a semantic account that sets the stage for the constraint-based treatment. In Section 3, we describe an encoding of dominance constraints into a constraint satisfaction problem on sets, and prove its soundness and completeness in Section 4. The constraint satisfaction problem can be solved effectively using constraint programming and, in Section 5, we formulate precise requirements for the target programming language.

2 Dominance Constraints With Boolean Connectives

We propose a new language of tree descriptions that subsumes the language of dominance constraints with set operators of Duchier and Niehren (2000) and combines it with the feature tree logic CFT of Smolka and Treinen (1994). Its abstract syntax is given below, where x, y range over an infinite set of node variables, $r \in \{=, \lhd^+, \rhd^+, \perp\}$ and f ranges over a finite signature Σ:

$$\phi \quad ::= \quad x\,r\,y \ \mid\ |x| = n \ \mid\ x{:}f \ \mid\ x[i] = y \ \mid\ \phi_1 \wedge \phi_2 \ \mid\ \phi_1 \vee \phi_2 \ \mid\ \neg\phi$$

Without loss of generality, we restrict our attention to a language where only literals may be negated, i.e. where negation has been pushed inward as far as possible. The admission of disjunction subsumes the set operator extension of Duchier and Niehren (2000) since $x\,R\,y \equiv \vee\{x\,r\,y \mid r \in R\}$.

The semantics of dominance constraints with Boolean connectives are given by interpretation over tree structures. We assume a finite signature Σ of function symbols $f, g, a, b, \ldots$ each equipped with an arity $\mathsf{ar}(f) \geq 0$. We write $\textsc{max}$ for the maximum arity in Σ. We further assume that Σ contains at least one constant and one function symbol of arity ≥ 2. We are interested in trees which can be regarded as ground terms over Σ, e.g. $f(g(a, b))$.

Nothing in our presentation critically depends on the existence of the arity function ar: with it we obtain constructor trees, without it traditional feature trees. What is critical is that both Σ and the set of features ($\{1 \mathinner{..} \textsc{max}\}$ for constructor trees) be finite. The finiteness of our encoding in Section 3 depends on this assumption.

We identify a node in a tree with the path that leads to it starting from the root of the tree. A path π is a word (i.e. a sequence) of positive integers. We write ϵ for the empty path and $\pi_1\pi_2$ for the concatenation of π_1 and π_2. π' is a prefix of π iff there exists π'' such that $\pi = \pi'\pi''$. We write $\pi_1 \lhd^* \pi_2$ when π_1 is a prefix of π_2 and say that π_1 dominates π_2. A tree-domain is a non-empty prefix-closed set of paths. A (finite) tree τ is a pair (D_τ, L_τ) of a finite tree-domain D_τ and a labeling function $L_\tau : D_\tau \to \Sigma$ with the property that all $\pi \in D_\tau$ and $k \geq 1$ satisfy $\pi k \in D_\tau$ iff $k \leq \mathsf{ar}(L_\tau(\pi))$, i.e. that each node has precisely as many children as required by the arity of the function symbol with which it is labeled. We write $\pi[i]\pi'$, and say that π' is the daughter of π at feature i, when $\pi' = \pi i$.

The tree structure $\mathcal{M}^\tau$ of a finite tree τ is a first-order structure with domain D_τ and containing, for each $f \in \Sigma$ and $0 \leq n \leq \textsc{max}$, the unary relations $:f^\tau$ and n^τ:

$$\begin{aligned} :f^\tau &= \{\pi \in D_\tau \mid L_\tau(\pi) = f\} \\ n^\tau &= \{\pi \in D_\tau \mid \mathsf{ar}(L_\tau(\pi)) = n\} \end{aligned}$$

and also the binary relations $[i]^\tau$ for each $1 \leq i \leq \textsc{max}$:

$$[i]^\tau \quad = \quad \{(\pi, \pi') \in D_\tau \times D_\tau \mid \pi[i]\pi'\} \quad = \quad \{(\pi, \pi i) \mid \pi i \in D_\tau\}$$

as well as all relations formed from them by inversion $^{-1}$, union $\cup$, intersection $\cap$, complementation $\neg$, composition $\circ$, and reflexive, transitive closure *. For any relations $R, R' \subseteq D_\tau \times D_\tau$, we define $R^{-1} = \{(\pi', \pi) \mid (\pi, \pi') \in R\}$, $R' \circ R = \{(\pi, \pi'') \mid (\pi, \pi') \in R \wedge (\pi', \pi'') \in R'\}$ and $\neg R = D_\tau \times D_\tau \setminus R$. Writing $\uplus$ for disjoint union, we obtain immediate dominance $\lhd^\tau$ by $\uplus\{[i]^\tau \mid 1 \leq i \leq \textsc{max}\}$, dominance $\lhd^{*\tau}$ by $\lhd^{\tau*}$, inverse dominance $\rhd^{*\tau}$ by $\lhd^{*\tau-1}$, equality $=^\tau$ by $\lhd^{*\tau} \cap \rhd^{*\tau}$, inequality $\neq^\tau$ by $\neg{=^\tau}$, proper dominance $\lhd^{+\tau}$ by

$\lhd^{*\tau} \cap \neq^\tau$, inverse proper dominance $\rhd^{+\tau}$ by $\lhd^{+\tau -1}$, and disjointness $\perp^\tau$ by $\neg\lhd^{*\tau} \cap \neg\rhd^{*\tau}$. We also pose $\lhd^{i,*\tau} = \lhd^{*\tau} \circ [i]^\tau$, and say that π' is a descendant of π at feature i when $\pi \lhd^{i,*\tau} \pi'$. The following partition holds in all tree structures:

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

2.2 Proof System And Models

We write V_ϕ for the set of variables occurring in ϕ. Classically, a solution of ϕ consists of a tree structure $\mathcal{M}^\tau$ and a variable assignment $\alpha : V_\phi \to D_\tau$, and we write $(\mathcal{M}^\tau, \alpha) \models \phi$ if the constraint ϕ is satisfied by $(\mathcal{M}^\tau, \alpha)$ in the usual Tarskian way. For technical reasons, in the proofs of Section 4 we need to keep track of which disjuncts are being satisfied. To this end, we introduce simple proof terms with the following abstract syntax:

$$\delta \quad ::= \quad \bullet \;\mid\; \delta \cdot \delta' \;\mid\; \overleftarrow{\delta} \;\mid\; \overrightarrow{\delta}$$

$\bullet$ is the constant proof term assigned to literals, the infix $\cdot$ is a binary proof constructor for conjunctions, and $\overleftarrow{\ }$ (resp. $\overrightarrow{\ }$) is a unary proof constructor for disjunctions, with the arrow pointing in the direction of the selected disjunct.

A solution of ϕ is a triple $(\mathcal{M}^\tau, \alpha, \delta)$. We say that $(\mathcal{M}^\tau, \alpha, \delta)$ satisfies ϕ, and write $(\mathcal{M}^\tau, \alpha, \delta) \models \phi$ iff $(\mathcal{M}^\tau, \alpha) \vdash \delta : \phi$ according to the proof system of Figure 1.

$$\frac{(\mathcal{M}^\tau, \alpha) \vdash \delta_1 : \phi_1 \qquad (\mathcal{M}^\tau, \alpha) \vdash \delta_2 : \phi_2}{(\mathcal{M}^\tau, \alpha) \vdash \delta_1 \cdot \delta_2 : \phi_1 \wedge \phi_2}$$

$$\frac{(\mathcal{M}^\tau, \alpha) \vdash \delta_1 : \phi_1}{(\mathcal{M}^\tau, \alpha) \vdash \overleftarrow{\delta_1} : \phi_1 \vee \phi_2} \qquad \frac{(\mathcal{M}^\tau, \alpha) \vdash \delta_2 : \phi_2}{(\mathcal{M}^\tau, \alpha) \vdash \overrightarrow{\delta_2} : \phi_1 \vee \phi_2}$$

$$\begin{array}{ll}
(\mathcal{M}^\tau, \alpha) \vdash \bullet : x\, r\, y & \text{if } (\alpha(x), \alpha(y)) \in r^\tau \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : \neg(x\, r\, y) & \text{if } (\alpha(x), \alpha(y)) \notin r^\tau \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : |x| = n & \text{if } \alpha(x) \in n^\tau \ \text{ i.e. } \mathsf{ar}(L_\tau(\alpha(x))) = n \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : \neg(|x| = n) & \text{if } \alpha(x) \notin n^\tau \ \text{ i.e. } \mathsf{ar}(L_\tau(\alpha(x))) \neq n \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : x{:}f & \text{if } \alpha(x) \in {:}f^\tau \ \text{ i.e. } L_\tau(\alpha(x)) = f \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : \neg(x{:}f) & \text{if } \alpha(x) \notin {:}f^\tau \ \text{ i.e. } L_\tau(\alpha(x)) \neq f \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : x[i]{=}y & \text{if } \alpha(y) = \alpha(x)i \\
(\mathcal{M}^\tau, \alpha) \vdash \bullet : \neg(x[i]{=}y) & \text{if } \alpha(y) \neq \alpha(x)i
\end{array}$$

Fig. 1. Proof System

Theorem 1 *The satisfiability problem of dominance constraints with Boolean connectives is NP-complete.*

This follows from the result of Koller et al. (2000b) that the satisfiability problem of all logical languages over dominance constraints between the purely conjunctive fragment and the positive existential fragment are NP-complete, and from the idea of their polynomial encoding of negation in the presence of a finite signature, namely to expand $\neg x : f(x_1, \ldots, x_n)$ into the finite disjunction of all other possibilities:

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

where x'_i and x''_i are fresh variables. The same idea may be used to reduce feature constraints $x[k] = y$ to constructor tree constraints:

$$\vee\{x : f(x_1, \ldots, x_{\mathsf{ar}(f)}) \wedge x_k = y \quad | \quad f \in \Sigma \ \wedge \ \mathsf{ar}(f) \geq k\}$$

where the x_i are fresh variables. In this respect, our approach has over theirs the practical advantage of economy.

We now sketch a more direct proof: in Section 3 we describe a polynomial encoding of a dominance constraint with Boolean connectives into a constraint satisfaction problem on variables ranging over elements or subsets of finite domains, and in Section 4 we prove soundness and completeness. This establishes that the satisfiability problem is in NP (we can guess an assignment and check it in polynomial time). NP-completeness can again be shown by encoding propositional satisfaction: we assume a signature that contains the two constants true and false and we consider formulae given by the abstract syntax below, where p ranges over an infinite set of propositional variables:

$$P, P' \quad ::= \quad p \ | \ \neg p \ | \ P \wedge P' \ | \ P \vee P'$$

The satisfiability preserving encoding below transforms a propositional formula P into a dominance constraint $[\![P]\!]$:

$$\begin{array}{rlcrl} [\![p]\!] &= p : \mathsf{true} & \qquad & [\![P \wedge P']\!] &= [\![P]\!] \wedge [\![P']\!] \\ [\![\neg p]\!] &= p : \mathsf{false} & & [\![P \vee P']\!] &= [\![P]\!] \vee [\![P']\!] \end{array}$$

2.3 Functions And Invariants

We consider tree structures equipped with the following functions of type $D_\tau \to 2^{D_\tau}$. For each binary relation $r^\tau \subseteq D_\tau \times D_\tau$ there is a function $\widetilde{r}^\tau$:

$$\widetilde{r}^\tau(\pi) \quad = \quad \{\pi' \mid (\pi, \pi') \in r^\tau\}$$

We write eq_τ for $\widetilde{=}^\tau$, down_τ for $\widetilde{\lhd^+}^\tau$, up_τ for $\widetilde{\rhd^+}^\tau$, side_τ for $\widetilde{\perp}^\tau$, eqdown_τ for $\widetilde{\lhd^*}^\tau$, equp_τ for $\widetilde{\rhd^*}^\tau$, down^i_τ for $\widetilde{\lhd^{i,*}}^\tau$, and child^i_τ for $\widetilde{[i]}^\tau$. Finally, writing $\mathcal{R}$ for the set of relation symbols $\{=, \lhd^+, \rhd^+, \perp\}$, there is a function $\mathsf{rel}_\tau : D_\tau \times D_\tau \to \mathcal{R}$ defined by:

$$\mathsf{rel}_\tau(\pi, \pi') = r \qquad \text{if } (\pi, \pi') \in r^\tau \text{ for some } r \in \mathcal{R}$$

In all tree structures, the invariants of Figure 2 hold, where $\|$ stands for disjointness. These invariants form the basis of our encoding into set constraints. They are not axiomatically minimal, but the redundancy is required to obtain strong propagation in the solver (Duchier and Niehren, 2000).

For all $\pi, \pi' \in D_\tau$ and $1 \leq i \leq \textsc{max}$

$$\begin{array}{rcll}
\{\pi\} & = & \mathsf{eq}_\tau(\pi) & (1)\\
D_\tau & = & \mathsf{eq}_\tau(\pi) \uplus \mathsf{down}_\tau(\pi) \uplus \mathsf{up}_\tau(\pi) \uplus \mathsf{side}_\tau(\pi) & (2)\\
\mathsf{eqdown}_\tau(\pi) & = & \mathsf{eq}_\tau(\pi) \uplus \mathsf{down}_\tau(\pi) & (3)\\
\mathsf{equp}_\tau(\pi) & = & \mathsf{eq}_\tau(\pi) \uplus \mathsf{up}_\tau(\pi) & (4)\\
\mathsf{down}_\tau(\pi) & = & \uplus\{\mathsf{down}^i_\tau(\pi) \mid 1 \leq i \leq \mathsf{ar}(L_\tau(\pi))\} & \\
 & = & \uplus\{\mathsf{down}^i_\tau(\pi) \mid 1 \leq i \leq \textsc{max}\} & (5)\\
\mathsf{child}^i_\tau(\pi) & \subseteq & \mathsf{down}^i_\tau(\pi) & (6)\\
i > \mathsf{ar}(L_\tau(\pi)) & \Rightarrow & \mathsf{down}^i_\tau(\pi) = \emptyset & (7)\\
\pi \neg{=} \pi' & \Rightarrow & \mathsf{eq}_\tau(\pi) \,\|\, \mathsf{eq}_\tau(\pi') & (8)\\
\pi \lhd^+ \pi' & \Rightarrow & \quad \mathsf{eqdown}_\tau(\pi') \subseteq \mathsf{down}_\tau(\pi) & (9)\\
 & & \wedge \quad \mathsf{equp}_\tau(\pi) \subseteq \mathsf{up}_\tau(\pi') & \\
 & & \wedge \quad \mathsf{side}_\tau(\pi) \subseteq \mathsf{side}_\tau(\pi') & \\
\pi \neg{\lhd^+} \pi' & \Rightarrow & \quad \mathsf{eq}_\tau(\pi) \,\|\, \mathsf{up}_\tau(\pi') & (10)\\
 & & \wedge\ \mathsf{down}_\tau(\pi) \,\|\, \mathsf{eq}_\tau(\pi') & \\
\pi \perp \pi' & \Rightarrow & \quad \mathsf{eqdown}_\tau(\pi) \subseteq \mathsf{side}_\tau(\pi') & (11)\\
 & & \wedge\ \mathsf{eqdown}_\tau(\pi') \subseteq \mathsf{side}_\tau(\pi) & \\
\pi \neg{\perp} \pi' & \Rightarrow & \quad \mathsf{eq}_\tau(\pi) \,\|\, \mathsf{side}_\tau(\pi') & (12)\\
 & & \wedge\ \mathsf{side}_\tau(\pi) \,\|\, \mathsf{eq}_\tau(\pi') & \\
\pi' = \pi i & \Rightarrow & \quad \mathsf{child}^i_\tau(\pi) = \mathsf{eq}_\tau(\pi') & (13)\\
 & & \wedge\ \mathsf{down}^i_\tau(\pi) = \mathsf{eqdown}_\tau(\pi') & \\
 & & \wedge \quad \mathsf{up}_\tau(\pi') = \mathsf{equp}_\tau(\pi) & \\
\pi' \neq \pi i & \Rightarrow & \mathsf{eq}_\tau(\pi') \,\|\, \mathsf{child}^i_\tau(\pi) & (14)\\
\pi' = \pi i \pi'' & \Rightarrow & \mathsf{eqdown}_\tau(\pi') \subseteq \mathsf{down}^i_\tau(\pi) & (15)\\
\pi' \neq \pi i \pi'' & \Rightarrow & \mathsf{eq}_\tau(\pi') \,\|\, \mathsf{down}^i_\tau(\pi) & (16)
\end{array}$$

Fig. 2. Invariants in Tree Structures

Invariants of tree structures induce corresponding invariants on the variables they interpret, and, to make this precise, we introduce the constraint language below, where S, S_i are variables denoting sets:

$$C \quad ::= \quad S_1 = S_2 \;\mid\; S_1 \subseteq S_2 \;\mid\; S_1 \parallel S_2 \;\mid\; S = S_1 \uplus \cdots \uplus S_n \;\mid\; C_1 \wedge C_2 \;\mid\; C_1 \vee C_2 \;\mid\; C_1 \Rightarrow C_2$$

We write V_C for the set of variables of C. Given a set D, a D-solution of C is a variable assignment $\sigma : V_C \to 2^D$ that makes C true. We write $\sigma \models_D C$. For any function $\alpha : D_1 \to D_2$, we write $\alpha^{-1} : D_2 \to 2^{D_1}$ for its inverse image and overload it in the usual fashion to obtain $\alpha^{-1} : 2^{D_2} \to 2^{D_1}$ defined by $\alpha^{-1}(S) = \cup\{\alpha^{-1}(x) \mid x \in S\}$.

Proposition 2 *Given two sets D_1 and D_2, a function $\alpha : D_1 \to D_2$, then $\forall\sigma : V_C \to D_2$ if $\sigma \models_{D_2} C$ then $\alpha^{-1} \circ \sigma \models_{D_1} C$.*

The proof follows by induction and by contradiction from the fact that α is a function. We will use this result in Section 4 to transfer invariants about sets of nodes to corresponding invariants about the set of variables they interpret.

3 Reduction to a CSP

Our approach to solving a dominance constraint ϕ is to transform it into an equivalent constraint satisfaction problem (CSP) $[\![\phi]\!]$ involving *finite domain* (FD) variables taking values in finite integer domains and *finite set* (FS) variables ranging over subsets of finite integer domains. Every $x \in V_\phi$ must be encoded by a distinct integer; also every $f \in \Sigma$: in the interest of legibility we will leave all such encodings implicit.

The encoding follows a pattern similar to our earlier work (Duchier and Gardent, 1999; Duchier, 1999b; Duchier and Niehren, 2000). Section 3.1 defines the language for expressing the CSP. Section 3.2 introduces the variables of the CSP and states some basic constraints about them. We are only interested in models which are trees and Section 3.3 formulates *well-formedness* constraints to that effect. Finally Section 3.4 introduces the additional *problem specific constraints* which admit only those trees which actually satisfy ϕ.

3.1 CSP Language

Let $\Delta = \{0\,..\,\mu\}$ be an integer interval for μ sufficiently large. We assume a set of FD variables I, I_k with values in Δ and a set of FS variables S, S_k with values in 2^Δ. FD and FS variables are also generically written X, X_k. We write D for a domain, i.e. a fixed subset of Δ. The syntax of a constraint $\mathcal{C}$ of our target CSP language is:

$$\begin{array}{rcl}\mathcal{C} & ::= & X_1 = X_2 \mid I \in D \mid D \subseteq S \mid S \subseteq D \mid I \in S \mid \\ & & I_1 < I_2 \mid S_1 \subseteq S_2 \mid S_1 \parallel S_2 \mid S = S_1 \uplus \cdots \uplus S_n \mid \\ & & X = \langle X_1, \ldots, X_n \rangle[I] \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \mathcal{C}_1 \vee \mathcal{C}_2\end{array}$$

This language needs not provide implications since, as we shall see, our encoding turns all implications, e.g. those of Figure 2, into disjunctions.

We say that β is a solution of $\mathcal{C}$, and write $\beta \models \mathcal{C}$, if β is an assignment of elements of Δ to FD variables and subsets of Δ to FS variables that makes $\mathcal{C}$ true. The *selection constraint* $X = \langle X_1, \ldots, X_n \rangle[I]$ is a novel element of our encoding which Duchier (1999a) introduced for the efficient constraint-based treatment of ambiguity in dependency grammar. $X = \langle X_1, \ldots, X_n \rangle[I]$ is true iff $X = X_I$, more precisely β makes it true iff $\beta(I) \in \{1 \ldots n\} \wedge \beta(X) = \beta(X_{\beta(I)})$.

3.2 CSP Variables And Invariants

As described in Section 2.3, a tree structure $\mathcal{M}^\tau$ is equipped with a number of functions such as eq_τ, down_τ, etc. The idea of the encoding is to introduce a CSP variable to denote the value that each function takes at each $x \in V_\phi$. More precisely: given a model $(\mathcal{M}^\tau, \alpha, \delta)$ of ϕ, each function $\gamma : D_\tau \to 2^{D_\tau}$ of $\mathcal{M}^\tau$ induces a corresponding function $\Gamma : V_\phi \to 2^{V_\phi}$ as follows:

$$\Gamma = \alpha^{-1} \circ \gamma \circ \alpha$$

For each $x \in V_\phi$, we introduce a set variable $\mathit{\Gamma}_x$ to denote $\Gamma(x)$. Thus, for each $x \in V_\phi$, there are $6 + 2 \times \textsc{max}$ set variables written Eq_x, Up_x, Down_x, Side_x, Equp_x, Eqdown_x and Child^i_x, Down^i_x for $1 \leq i \leq \textsc{max}$.

Similarly, each function $\gamma : D_\tau \to \mathbb{N}$ (resp. $\gamma : D_\tau \to \Sigma$) induces a corresponding function $\Gamma : V_\phi \to \mathbb{N}$ (resp. $\Gamma : V_\phi \to \Sigma$) as follows:

$$\Gamma = \gamma \circ \alpha$$

Thus, for each $x \in V_\phi$, we introduce two integer variables: Arity_x to denote $(\mathsf{ar} \circ L_\tau \circ \alpha)(x)$ and Label_x to denote $(L_\tau \circ \alpha)(x)$. Finally, for every $x, y \in V_\phi$, we introduce an integer variable R_{xy} to denote $\mathsf{rel}_\tau(\alpha(x), \alpha(y))$. For all $x, y \in V_\phi$ and $1 \leq i \leq \textsc{max}$:

$$\begin{array}{c}\mathit{Eq}_x, \mathit{Down}_x, \mathit{Up}_x, \mathit{Side}_x, \mathit{Child}^i_x, \mathit{Down}^i_x, \mathit{Eqdown}_x, \mathit{Equp}_x \quad \subseteq \quad V_\phi \\ \mathit{Arity}_x \in \mathsf{ar}(\Sigma) \qquad \mathit{Label}_x \in \Sigma \qquad R_{xy} \in \mathcal{R}\end{array} \tag{17}$$

As noted in Section 2.3, invariants of tree structures induce corresponding invariants on the variables they interpret. Therefore, we pattern the constraints below after invariants (1–7) of Figure 2:

$$
\begin{aligned}
x &\in Eq_x && (18)\\
V_\phi &= Eq_x \uplus Down_x \uplus Up_x \uplus Side_x && (19)\\
Eqdown_x &= Eq_x \uplus Down_x && (20)\\
Equp_x &= Eq_x \uplus Up_x && (21)\\
Down_x &= \uplus\{Down_x^i \mid 1 \leq i \leq \text{MAX}\} && (22)\\
Child_x^i &\subseteq Down_x^i && (23)\\
i > Arity_x &\Rightarrow |Down_x^i| = 0 && (24)
\end{aligned}
$$

where, writing $i \leq Arity_x$ for $Arity_x \in \{i, \ldots, \mu\}$, (24) stands for:

$$(i > Arity_x \ \wedge \ |Down_x^i| = 0) \ \vee \ i \leq Arity_x$$

By associativity of composition:

$$
\begin{array}{lrcl}
 & (\mathsf{ar} \circ L_\tau \circ \alpha)(x) & = & \mathsf{ar} \circ (L_\tau \circ \alpha)(x) \\
\text{i.e.} & Arity_x & = & \mathsf{ar}(Label_x)
\end{array}
$$

However, the latter equation is inappropriate for constraint propagation since $Label_x$ is an unknown. Instead, we take advantage of the fact that ar is a finite map from function symbols to integers, and encode it using the selection constraint below:

$$Arity_x = \langle \mathsf{ar}(f_1), \ldots, \mathsf{ar}(f_m)\rangle[Label_x] \qquad (25)$$

where $\Sigma = \{f_1, \ldots, f_m\}$ and each f_k is implicitly identified with the integer k that encodes it.

3.3 Well-Formedness Clauses

The well-formedness clauses are concerned with invariants characterizing the shape of the tree: (1) does x stand in relation r to y, (2) is y a daughter of x at feature i, (3) is y a descendent of x at feature i?

(1) For each $r \in \mathcal{R}$, Figure 2 lists invariants of the form:

$$\pi \; r \; \pi' \Rightarrow C_r(\pi, \pi') \qquad \text{and} \qquad \pi \; \neg r \; \pi' \Rightarrow C_{\neg r}(\pi, \pi')$$

Since $\pi \; r \; \pi' \vee \pi \; \neg r \; \pi$ is valid in all tree structures, so is the disjunction below:

$$(\pi \; r \; \pi' \wedge C_r(\pi, \pi')) \quad \vee \quad (\pi \; \neg r \; \pi' \wedge C_{\neg r}(\pi, \pi'))$$

Thus, on the basis of invariants (8–12), we formulate a quadratic number of well-formedness clauses. For each $x, y \in V_\phi$ and $r \in \mathcal{R}$:

$$\mathsf{B}[\![x\ r\ y]\!] \wedge R_{xy} = r \quad \vee \quad R_{xy} \neq r \wedge \mathsf{B}[\![x\ \neg r\ y]\!] \tag{26}$$

where $R_{xy} = r$ corresponds to $\alpha(x)\ r\ \alpha(y)$ and $\mathsf{B}[\![x\ r\ y]\!]$, defined in Figure 3, corresponds to $C_r(\alpha(x), \alpha(y))$ and forms the *characteristic set constraints* of the case $x\ r\ y$.

$$\begin{aligned}
\mathsf{B}[\![x = y]\!] &= Eq_x = Eq_y \wedge Up_x = Up_y \wedge Down_x = Down_y \wedge Side_x = Side_y \wedge \\
&\quad\ Equp_x = Equp_y \wedge Eqdown_x = Eqdown_y \wedge Label_x = Label_y \wedge \\
&\quad\ Arity_x = Arity_y \wedge_{i=1}^{\mathrm{MAX}} Down_x^i = Down_y^i \wedge_{i=1}^{\mathrm{MAX}} Child_x^i = Child_y^i \\
\mathsf{B}[\![x \neg\!= y]\!] &= Eq_x \parallel Eq_y \\
\mathsf{B}[\![x \lhd^+ y]\!] &= Eqdown_y \subseteq Down_x \wedge Equp_x \subseteq Up_y \wedge Side_x \subseteq Side_y \\
\mathsf{B}[\![x \neg\lhd^+ y]\!] &= Eq_x \parallel Up_y \wedge Down_x \parallel Eq_y \\
\mathsf{B}[\![x \perp y]\!] &= Eqdown_x \subseteq Side_y \wedge Eqdown_y \subseteq Side_x \\
\mathsf{B}[\![x \neg\perp y]\!] &= Eq_x \parallel Side_y \wedge Side_x \parallel Eq_y \\
\mathsf{B}[\![y \rhd^+ x]\!] &= \mathsf{B}[\![x \lhd^+ y]\!] \\
\mathsf{B}[\![y \neg\rhd^+ x]\!] &= \mathsf{B}[\![x \neg\lhd^+ y]\!]
\end{aligned}$$

Fig. 3. Characteristic Set Constraints

(2) Similarly, on the basis of invariants (13) and (14), we also formulate clauses concerned with whether or not y is x's daughter at feature i:

$$Child_x^i = Eq_y \wedge Down_x^i = Eqdown_y \wedge Up_y = Equp_x \ \vee\ Child_x^i \parallel Eq_y \tag{27}$$

(3) Finally, y either is or is not a descendent of x at feature i. On the basis of invariants (15) and (16) we obtain:

$$Eqdown_y \subseteq Down_x^i \quad \vee \quad Eq_y \parallel Down_x^i \tag{28}$$

3.4 Problem-Specific Constraints

So far, the constraints of our encoding only served to characterize structures appropriate for interpreting the variables of ϕ: namely tree structures. Now we turn to the last part of the encoding which forms the additional *problem specific* constraints that further restrict the admissibility of tree structures and admits only those which are models of ϕ.

ϕ can be viewed as a finite tree (D_ϕ, L_ϕ) over the finite signature Σ_ϕ containing all symbols of the form $\wedge$, $\vee$, $x\ r\ y$, $\neg(x\ r\ y)$, $|x|{=}n$, $\neg(|x|{=}n)$, $x{:}f$, $\neg(x{:}f)$,

$x[n]{=}y$, $\neg(x[n]{=}y)$, for all x, y and n appearing in ϕ, $f \in \Sigma_\tau$, and $1 \le i \le \textsc{max}$. We write $\phi[\pi]$ for the subformula of ϕ at path π.

The intuition is that each subformula of ϕ stipulates a set of licensed values for $R_{xy}, Arity_x, Label_x, Child_x^i$. Thus for each $\pi \in D_\phi$ we introduce variables $R^\pi_{xy}, Arity^\pi_x, Label^\pi_x$ to represent the values licensed by $\phi[\pi]$ for R_{xy}, $Arity_x$, $Label_x$. We also introduce $Lo^\pi_{i,x}$, $Hi^\pi_{i,x}$ to represent resp. the upper and lower bounds stipulated for $Child^i_x$. We write V^π_ϕ for the set of variables R^π_{xy}, $Arity^\pi_x$, $Label^\pi_x$, $Lo^\pi_{i,x}$, $Hi^\pi_{i,x}$ for $1 \le i \le \textsc{max}$. The problem-specific constraints are then expressed as follows:

$$R_{xy} \in R^\epsilon_{xy} \tag{29}$$
$$Arity_x \in Arity^\epsilon_x \tag{30}$$
$$Label_x \in Label^\epsilon_x \tag{31}$$
$$Child^i_x \supseteq Lo^\epsilon_{i,x} \tag{32}$$
$$Child^i_x \subseteq Hi^\epsilon_{i,x} \tag{33}$$

Each variable $X^\pi \in V^\pi_\phi$ is defined by a constraint and we first present an example to help clarify the intuition. Consider ϕ of the form $(x{:}f \vee x{:}g) \wedge (x{:}h \vee x{:}f)$. For each disjunction $\phi[\pi]$ we introduce a *selector variable* I^π to indicate which alternative is selected: $I^\pi = 1$ for the left disjunct and $I^\pi = 2$ for the right one. For each $\pi \in D_\phi$, variable $Label^\pi_x$ is defined as follows:

$$\begin{aligned}
Label_x &\in Label^\epsilon_x & \qquad Label^1_x &= \langle Label^{1\,1}_x, Label^{1\,2}_x\rangle[I^1] \\
Label^\epsilon_x &= Label^1_x \cap Label^2_x & Label^2_x &= \langle Label^{2\,1}_x, Label^{2\,2}_x\rangle[I^2] \\
Label^{1\,1}_x &= \{f\} & Label^{2\,1}_x &= \{h\} \\
Label^{1\,2}_x &= \{g\} & Label^{2\,2}_x &= \{f\}
\end{aligned}$$

where I^1 and I^2 indicate which disjunct is selected in resp. the 1st and 2nd disjunction. Constraint propagation infers $Label^1_x \subseteq \{f, g\}$ and $Label^2_x \subseteq \{h, f\}$, i.e. $Label^\epsilon_x = \{f\}$, i.e. $Label_x = f$.

For simplicity, we introduce a selector variable I^π for every subformula $\phi[\pi]$ and stipulate $I^\pi \in \{1, 2\}$ if $\phi[\pi]$ is a disjunction and $I^\pi = 0$ otherwise.

Each variable $X^\pi \in V^\pi_\phi$ is defined by a constraint $\Upsilon^\pi_\phi(X^\pi)$. We call Υ^π_ϕ a *restriction* and define it, in Figure 4, $\forall \pi \in D_\phi$ by induction on the syntactic representation of ϕ. $\Upsilon^\pi_{\phi,\wedge}$, $\Upsilon^\pi_{\phi,\vee}$, and $\top^\pi_\phi[\cdot]$ are defined below. The selector variables I^π identify which disjuncts are satisfied by a model $(\mathcal{M}^\tau, \alpha, \delta)$. Intuitively $I^\pi = 1$ (resp. $I^\pi = 2$) if in the proof of $(\mathcal{M}^\tau, \alpha) \vdash \delta : \phi$, $\phi[\pi]$ is assigned a proof term $\overleftarrow{\delta'}$ (resp. $\overrightarrow{\delta'}$).

$\Upsilon^\pi_{\phi,\wedge}$ is the restriction for conjunctive subformulae and is defined as follows,

$\phi[\pi]$	Υ^{π}_{ϕ}
$\phi_1 \wedge \phi_2$	$\Upsilon^{\pi}_{\phi,\wedge}$
$\phi_1 \vee \phi_2$	$\Upsilon^{\pi}_{\phi,\vee}$
$x\ r\ y$	$\top^{\pi}_{\phi}[R_{xy} \mapsto \{r\}]$
$\neg(x\ r\ y)$	$\top^{\pi}_{\phi}[R_{xy} \mapsto \mathcal{R} \setminus \{r\}]$
$\|x\|{=}n$	$\top^{\pi}_{\phi}[\mathit{Arity}_x \mapsto \{n\}]$
$\neg(\|x\|{=}n)$	$\top^{\pi}_{\phi}[\mathit{Arity}_x \mapsto \{0 \mathinner{..} \textsc{max}\} \setminus \{n\}]$
$x{:}f$	$\top^{\pi}_{\phi}[\mathit{Label}_x \mapsto \{f\}]$
$\neg(x{:}f)$	$\top^{\pi}_{\phi}[\mathit{Label}_x \mapsto \Sigma \setminus \{f\}]$
$x[i]{=}y$	$\top^{\pi}_{\phi}[\mathit{Lo}_{i,x} \mapsto \{y\}]$
$\neg(x[i]{=}y)$	$\top^{\pi}_{\phi}[\mathit{Hi}_{i,x} \mapsto V_\phi \setminus \{y\}]$

Fig. 4. Inductive Definition of Restrictions

where $\pi 1$ selects the left conjunct and $\pi 2$ selects the right conjunct:

$$\begin{array}{rcll}
\Upsilon^{\pi}_{\phi,\wedge}(I^{\pi}) & \equiv & I^{\pi} = 0 & \\
\Upsilon^{\pi}_{\phi,\wedge}(\mathit{Lo}^{\pi}_{i,x}) & \equiv & \mathit{Lo}^{\pi}_{i,x} = \mathit{Lo}^{\pi 1}_{i,x} \cup \mathit{Lo}^{\pi 2}_{i,x} & \\
\Upsilon^{\pi}_{\phi,\wedge}(X^{\pi}) & \equiv & X^{\pi 1} \cap X^{\pi 2} & \text{otherwise}
\end{array}$$

$\Upsilon^{\pi}_{\phi,\vee}$ is the restriction for disjunctive subformulae and is defined as follows:

$$\begin{array}{rcll}
\Upsilon^{\pi}_{\phi,\vee}(I^{\pi}) & \equiv & I^{\pi} \in \{1,2\} & \\
\Upsilon^{\pi}_{\phi,\vee}(X^{\pi}) & \equiv & X^{\pi} = \langle X^{\pi 1}, X^{\pi 2}\rangle[I^{\pi}] & \text{otherwise}
\end{array}$$

We write $\top^{\pi}_{\phi}$ for the most general restriction at $\phi[\pi]$ defined as follows:

$$\begin{array}{rcll}
\top^{\pi}_{\phi}(R^{\pi}_{xy}) & \equiv & R^{\pi}_{xy} = \mathcal{R} & \\
\top^{\pi}_{\phi}(\mathit{Arity}^{\pi}_{x}) & \equiv & \mathit{Arity}^{\pi}_{x} = \mathsf{ar}(\Sigma) & \subseteq \{0 \mathinner{..} \textsc{max}\} \\
\top^{\pi}_{\phi}(\mathit{Label}^{\pi}_{x}) & \equiv & \mathit{Label}^{\pi}_{x} = \Sigma & \\
\top^{\pi}_{\phi}(\mathit{Lo}^{\pi}_{i,x}) & \equiv & \mathit{Lo}^{\pi}_{i,x} = \emptyset & \\
\top^{\pi}_{\phi}(\mathit{Hi}^{\pi}_{i,x}) & \equiv & \mathit{Hi}^{\pi}_{i,x} = V_\phi & \\
\top^{\pi}_{\phi}(I^{\pi}) & \equiv & I^{\pi} = 0 &
\end{array}$$

and we write $\top^{\pi}_{\phi}[X \mapsto V]$ for the restriction such that:

$$\top^{\pi}_{\phi}[X \mapsto V](Y^{\pi}) \equiv \begin{cases} X^{\pi} = V & \text{if } Y \text{ is } X \\ \top^{\pi}_{\phi}(Y^{\pi}) & \text{otherwise} \end{cases}$$

The restriction constraints are given by:

$$\begin{array}{l}
\wedge\{\Upsilon^{\pi}_{\phi}(R^{\pi}_{xy}) \wedge \Upsilon^{\pi}_{\phi}(\mathit{Arity}^{\pi}_{x}) \wedge \Upsilon^{\pi}_{\phi}(\mathit{Label}^{\pi}_{x}) \wedge \Upsilon^{\pi}_{\phi}(\mathit{Lo}^{\pi}_{i,x}) \\
\quad \wedge\, \Upsilon^{\pi}_{\phi}(\mathit{Hi}^{\pi}_{i,x}) \wedge \Upsilon^{\pi}_{\phi}(I^{\pi}) \mid \pi \in D_\phi,\ x,y \in V_\phi,\ 1 \le i \le \textsc{max}\}
\end{array} \tag{34}$$

3.5 Assembling The CSP

We now formulate the complete CSP by putting together the translated invariants, the well-formedness clauses, and the problem-specific constraints. We refer to these constraints by means of their integer labels.

$$\begin{aligned}
[\![\phi]\!] \quad \equiv \quad & \wedge\{(17\text{–}25) \mid x, y \in V_\phi \;\; 1 \le i \le \textsc{max}\} \\
& \wedge\{(26\text{–}28) \mid x, y \in V_\phi \;\; 1 \le i \le \textsc{max},\;\; r \in \mathcal{R}\} \\
& \wedge\{(29\text{–}34) \mid x, y \in V_\phi \;\; 1 \le i \le \textsc{max}\}
\end{aligned}$$

4 Soundness And Completeness

Theorem 3 (Soundness) *if $[\![\phi]\!]$ is satisfiable then so is ϕ.*

Theorem 4 (Completeness) *if ϕ is satisfiable then so is $[\![\phi]\!]$.*

4.1 Completeness

We show that, given a model $(\mathcal{M}^\tau, \alpha, \delta)$ of ϕ, we can construct a solution β of $[\![\phi]\!]$. We choose β so that it makes the following assignments:

$$\begin{aligned}
\mathit{\Gamma}_x &\mapsto \Gamma(x) \\
R_{xy} &\mapsto \mathsf{rel}_\tau(\alpha(x), \alpha(y))
\end{aligned}$$

where $\mathit{\Gamma}_x$ and Γ are as described in Section 3.2. By (Proposition 2), CSP constraints (17–25) are satisfied.

If $\alpha(x) = \pi$ and $\alpha(y) = \pi'$ and $\pi \; r \; \pi'$ holds in $\mathcal{M}^\tau$. Then, from invariant $\pi \, r \, \pi' \Rightarrow C_r(\pi, \pi')$, one of (8–12), it follows that $C_r(\pi, \pi')$ holds. Therefore, by (Proposition 2), β satisfies $\mathsf{B}[\![x \, r \, y]\!]$. Also $\beta(R_{xy}) = r$ by construction. Similarly if $\pi \neg r \; \pi'$. Thus the well-formedness clauses (26) are satisfied.

The well-formedness clauses (27) are satisfied: if $\pi' = \pi i$, then $\pi i \in D_\tau$ and invariant (13) applies. Thus, by (Proposition 2), β satisfies the left disjunct of (27). If $\pi' \neq \pi i$, then, by definition of child^i_τ, $\pi' \notin \mathsf{child}^i_\tau(\pi)$. Thus, by (Proposition 2), β satisfies the right disjunct of (27). Similarly for (28).

We now turn to the problem-specific constraints. Given a model $(\mathcal{M}^\tau, \alpha, \delta)$ of ϕ, we write $\Pi(\delta)$ for the set of paths of the subformulae of ϕ involved in the proof of $(\mathcal{M}^\tau, \alpha) \models \delta : \phi$. Π is defined as follows:

$$\begin{array}{ll}\Pi(\bullet) & = \{\epsilon\} \\ \Pi(\delta \cdot \delta') & = \{\epsilon\} \cup 1\Pi(\delta) \cup 2\Pi(\delta') \\ \Pi(\overleftarrow{\delta}) & = \{\epsilon\} \cup 1\Pi(\delta) \\ \Pi(\overrightarrow{\delta}) & = \{\epsilon\} \cup 2\Pi(\delta)\end{array}$$

where we write $i\Pi(\delta)$ for $\{i\pi \mid \pi \in \Pi(\delta)\}$. We now describe how β assigns values to the restriction variables X^π. We are going to proceed by induction on π. π is said to be maximal in $\Pi(\delta)$ if $\pi\pi' \in \Pi(\delta) \Rightarrow \pi' = \epsilon$.

if π is maximal, then $\phi[\pi]$ is a literal and $\Upsilon^\pi_\phi(X^\pi)$ is of the form $X^\pi = V$ where V is a constant value. We pose $\beta(X^\pi) = V$. If π is not maximal, then $\phi[\pi]$ is either a conjunction or a disjunction.

If $\phi[\pi]$ is a conjunction, then by hyp., $\forall i \in \{1, 2\}$, $\beta(X^{\pi i})$ has been established. Since $\Upsilon^\pi_\phi(I^\pi) \equiv I^\pi = 0$, we pose $\beta(I^\pi) = 0$. For X other than I, $\Upsilon^\pi_\phi(X^\pi)$ has the form $X^\pi = X^{\pi 1}\ \rho\ X^{\pi 2}$ for ρ either $\cap$ or $\cup$. We pose $\beta(X^\pi) = \beta(X^{\pi 1})\ \rho\ \beta(X^{\pi 2})$.

If $\phi[\pi]$ is a disjunction, then by hyp., $\forall i \in \{1, 2\}$, $\beta(X^{\pi i})$ has been established. If $\pi i \in \Pi(\delta)$, we pose $\beta(I^\pi) = i$, otherwise we pick i arbitrarily in $\{1, 2\}$. For X other than I, $\Upsilon^\pi_\phi(X^\pi)$ has the form $X^\pi = \langle X^{\pi 1}, X^{\pi 2}\rangle[I^\pi]$. We pose $\beta(X^\pi) = \beta(X^{\pi\beta(I^\pi)})$.

By construction, β satisfies (34). Now it only remains to show that the problem-specific constraints (29–33) are also satisfied by β. We will do this only for (29), i.e. $R_{xy} \in R^\epsilon_{xy}$. The other proofs are similar.

We show that $\forall \pi \in \Pi(\delta)\ \beta \models R_{xy} \in R^\pi_{xy}$. Again we proceed by induction on π. If π is maximal, then $\phi[\pi]$ is a literal ℓ and we distinguish the cases when ℓ is $x\,r\,y$, $\neg(x\,r\,y)$, or something else. Since $\pi \in \Pi(\delta)$, $(\mathcal{M}^\tau, \alpha) \vdash \bullet : \ell$ is satisfied.

If ℓ is $x\,r\,y$, then $(\alpha(x), \alpha(y)) \in r^\tau$ and $\beta(R_{xy}) = r$. By construction $\beta(R^\pi_{xy}) = \{r\}$, which proves $\beta \models R_{xy} \in R^\pi_{xy}$. If ℓ is $\neg(x\ r\ y)$, then $(\alpha(x), \alpha(y)) \notin r^\tau$ and $\beta(R_{xy}) \neq r$. By construction $\beta(R^\pi_{xy}) = \mathcal{R} \setminus \{r\}$, which also proves $\beta \models R_{xy} \in R^\pi_{xy}$. Finally in all other cases, $\beta(R^\pi_{xy}) = \mathcal{R}$ which again proves $\beta \models R_{xy} \in R^\pi_{xy}$.

In π is not maximal, then $\phi[\pi]$ is either a conjunction or a disjunction. If $\phi[\pi]$ is a conjunction, we have $\forall i \in \{1, 2\}\ \pi i \in \Pi(\delta)$ and by hyp. $\beta \models R_{xy} \in R^{\pi i}_{xy}$. Since $R^\pi_{xy} = R^{\pi 1}_{xy} \cap R^{\pi 2}_{xy}$, also $\beta \models R_{xy} \in R^\pi_{xy}$. If $\phi[\pi]$ is a disjunction, we have $\exists i \in \{1, 2\}\ \pi i \in \Pi(\delta)$ and $\beta(I^\pi) = i$. Since $R^\pi_{xy} = R^{\pi\beta(I^\pi)}_{xy}$, also $\beta \models R_{xy} \in R^\pi_{xy}$. Hence by induction $\beta \models R_{xy} \in R^\epsilon_{xy}$.

We show that given a solution β of $[\![\phi]\!]$, we can construct a model $(\mathcal{M}^\tau, \alpha, \delta)$ of ϕ. We define $x \prec_\beta y \equiv \beta(R_{xy}) = \lhd^+$. If $\beta(R_{xy}) = \lhd^+$, then $R_{xy} \neq \lhd^+$ is false and β must satisfy the other alternative of (26), i.e. $\mathsf{B}[\![x \lhd^+ y]\!]$ holds, i.e. $y \in Eqdown_y \subseteq Down_x$, i.e. $Down_x \parallel Eq_y$ is false, i.e. $\mathsf{B}[\![x \neg\lhd^+ y]\!]$ is false. From $\mathsf{B}[\![x \neg\lhd^+ y]\!] = \mathsf{B}[\![y \neg\rhd^+ x]\!]$, we infer that β must satisfy the other alternative of (26), i.e. $R_{yx} = \rhd^+$ holds. Thus we have $\beta(R_{xy}) = \lhd^+$ iff $\beta(R_{yx}) = \rhd^+$. In particular, $\prec_\beta$ is antisymmetric. Further $\prec_\beta$ is transitive: if we have $x \prec_\beta y \wedge y \prec_\beta z$, then $\mathsf{B}[\![x \lhd^+ y]\!]$ and $\mathsf{B}[\![y \lhd^+ z]\!]$ must hold, i.e. $x \in Equp_x \subseteq Up_y \subseteq Equp_y \subseteq Up_z$, which makes $Eq_x \parallel Up_z$, i.e. $\mathsf{B}[\![x \neg\lhd^+ z]\!]$, inconsistent. Therefore the other alternative of clause (26), $\mathsf{B}[\![x \lhd^+ z]\!] \wedge R_{xz} = \lhd^+$ must hold, i.e. $\beta(R_{xz}) = \lhd^+$. Thus $\prec_\beta$ defines a partial ordering on V_ϕ.

We consider $V_\phi^=$ a maximal subset of V_ϕ such that for every distinct $x, x' \in V_\phi^=$, we have $\beta(R_{xx'}) \neq =$. We define:

$$
\begin{aligned}
\lceil S \rceil_\beta &= \{x \in S \mid x \text{ is } \prec_\beta \text{ maximal}\}\\
\lfloor S \rfloor_\beta &= \{x \in S \mid x \text{ is } \prec_\beta \text{ minimal}\}\\
\lfloor x \rfloor_\beta &= \{y \in V_\phi^= \mid x \prec_\beta y \text{ and } y \text{ is } \prec_\beta \text{ minimal}\}\\
\lfloor x \rfloor_\beta^i &= \{y \in \lfloor x \rfloor_\beta \mid y \in \beta(Down_x^i)\}
\end{aligned}
$$

We inductively construct V, D, α, L where $\tau = (D, L)$ is a finite tree, V is a set of variables of ϕ and α is a function from V to D. We assumed that the signature Σ contains at least one constant and one function symbol of arity ≥ 2. Given a constant a and a binary constructor f, n trees (τ_i) can be placed at disjoint positions by forming $f(\tau_1, f(\tau_2 \ldots f(\tau_n, a) \ldots))$. In such a case, for simplicity of presentation, we will pretend that we have a n-ary constructor $\mathtt{cons}_n$ giving us $\mathtt{cons}_n(\tau_1, \ldots, \tau_n)$.

Lemma 5 *Given β such that $\beta \models [\![\phi]\!]$. For any $S \subseteq V_\phi^=$, and every distinct $x, x' \in \lfloor S \rfloor_\beta$, we have $\beta(R_{xx'}) = \bot$. Similarly for every $y \in V_\phi^=$ and every distinct $x, x' \in \lfloor y \rfloor_\beta^i$.*

Proof: by definition of $V_\phi^=$, $\beta(R_{xx'}) \neq =$. By definition of $\lfloor\ \rfloor_\beta$, $\beta(R_{xx'}) \notin \{\lhd^+, \rhd^+\}$ otherwise one of them would not be minimal. Therefore $\beta(R_{xx'}) = \bot$. Similarly for $\lfloor y \rfloor_\beta^i$.

We start with $V = \lfloor V_\phi^= \rfloor_\beta = \{x_1, \ldots, x_n\}$. By Lemma 5, $\beta(R_{x_i x_j}) = \bot$ for $1 \leq i \neq j \leq n$ and we place the x_i at disjoint positions: $D = \{\epsilon, 1, \ldots, n\}$, $\alpha(x_i) = i$, and $L(\epsilon) = \mathtt{cons}_n$. D is a tree domain.

For the inductive step, consider the fringe $\lceil V \rceil_\beta$, i.e. the variables x such that $\alpha(x)$ is currently maximal in D. For each $x \in \lceil V \rceil_\beta$, consider $\lfloor x \rfloor_\beta$, i.e. the $\prec_\beta$ minimal variables below x. It follows from (22,24) that $\lfloor x \rfloor_\beta$ is partitioned by

$\lfloor x\rfloor^i_\beta$ for $1 \le i \le \mathit{Arity}_x$. For each $\lfloor x\rfloor^i_\beta$, there are 3 cases:

(1) $y \in \lfloor x\rfloor^i_\beta \cap \beta(\mathit{Child}^i_x)$. In this case, y is the only element of $\lfloor x\rfloor^i_\beta$. Suppose there is a distinct $y' \in \lfloor x\rfloor^i_\beta$. By (27), $y' \in \mathit{Down}^i_x = \mathit{Eqdown}_y$, i.e. $\beta(R_{yy'}) \ne \bot$ which contradicts Lemma 5. We pose $\alpha(y) = \alpha(x)i$ and add it to D.

(2) $\lfloor x\rfloor^i_\beta \ne \emptyset \wedge \lfloor x\rfloor^i_\beta \cap \beta(\mathit{Child}^i_x) = \emptyset$. Posing $\lfloor x\rfloor^i_\beta = \{y_1, \ldots, y_m\}$, by Lemma 5, $\beta(R_{y_i y_j}) = \bot$ for $1 \le i \ne j \le m$. We place them at disjoint positions. We pose $\alpha(y_k) = \alpha(x)ik$ and $L(\alpha(x)i) = \mathtt{cons}_n$, and add $\alpha(x)i$ and $\alpha(y_k)$ to D.

(3) $\lfloor x\rfloor^i_\beta = \beta(\mathit{Child}^i_x) = \emptyset$. If $i \le \beta(\mathit{Arity}_x)$ we add $\alpha(x)i$ to D and pose $L(\alpha(x)i) = c$ where c is a constant.

We add $\lceil V\rceil_\beta$ to V. D is still a tree domain. We stop when $\forall x \in \lceil V\rceil_\beta$, $\lfloor x\rfloor_\beta = \emptyset$. By induction, it must be the case that $V = V_\phi^=$. For every $x \in V_\phi \backslash V_\phi^=$ there is $y \in V$ such that $\beta(R_{xy}) = {=}$: we pose $\alpha(x) = \alpha(y)$ and add x to V. $\forall x \in V_\phi$ we define $L(\alpha(x)) = \beta(\mathit{Label}_x)$. For all $\pi \in D \setminus \alpha^{-1}(V_\phi)$, $L(\pi)$ was defined by the construction procedure.

Lemma 6 *In the tree constructed above, $\forall r \in \mathcal{R}$, we have $\alpha(x)\, r\, \alpha(y)$ iff $\beta(R_{xy}) = r$.*

The construction method enforces the invariant that $\alpha(x) \lhd^+ \alpha(y)$ in τ iff $\beta(R_{xy}) = \lhd^+$, and $\alpha(x) = \alpha(y)$ iff $\beta(R_{xy}) = {=}$. Therefore, $\forall r \in \mathcal{R}$, we have $\alpha(x)\, r\, \alpha(y)$ iff $\beta(R_{xy}) = r$.

Now that we have a tree τ, and therefore a tree structure $\mathcal{M}^\tau$, as well as a variable assignment α, we turn to the proof term δ such that $(\mathcal{M}^\tau, \alpha) \vdash \delta : \phi$. We pose $\delta = [\epsilon]\phi$, where $[\pi]\phi$ is defined as follows:

$$\begin{array}{rcll}
[\pi](\phi_1 \wedge \phi_2) & = & [\pi 1]\phi_1 \cdot [\pi 2]\phi_2 & \\
[\pi](\phi_1 \vee \phi_2) & = & \overleftarrow{[\pi 1]\phi_1} & \text{if } \beta(I^\pi) = 1 \\
[\pi](\phi_1 \vee \phi_2) & = & \overrightarrow{[\pi 2]\phi_2} & \text{if } \beta(I^\pi) = 2 \\
[\pi]\phi & = & \bullet & \text{otherwise}
\end{array}$$

Lemma 7 *By induction, the problem-specific constraints (29–33) hold not only at ϵ, but at every $\pi \in \Pi(\delta)$.*

We prove this for $R_{xy} \in R^\pi_{xy}$, for $\pi \in \Pi(\delta)$; the other proofs are similar. $\epsilon \in \Pi(\delta)$ and β satisfies (29), i.e. $\beta \models R_{xy} \in R^\epsilon_{xy}$ holds. Assume $\beta \models R_{xy} \in R^\pi_{xy}$ for $\pi \in \Pi(\delta)$. If $\phi[\pi] = \phi_1 \wedge \phi_2$, then $\pi 1, \pi 2 \in \Pi(\delta)$ and $R^\pi_{xy} = R^{\pi 1}_{xy} \cap R^{\pi 2}_{xy}$; therefore $\beta \models R_{xy} \in R^{\pi i}_{xy}$ must hold for both $i \in \{1, 2\}$. If $\phi[\pi] = \phi_1 \vee \phi_2$, then only $\pi i \in \Pi(\delta)$ for $i = \beta(I^\pi)$, and $\beta \models R_{xy} \in R^\pi_{xy} = \langle R^{\pi 1}_{xy}, R^{\pi 1}_{xy}\rangle[I^\pi] = R^{\pi i}_{xy}$; therefore $\beta \models R_{xy} \in R^{\pi i}_{xy}$. Otherwise $\phi[\pi]$ is a literal and we are done.

Lemma 8 *By induction,* $(\mathcal{M}^\tau, \alpha) \vdash [\pi]\phi : \phi[\pi]$ *holds for all* $\pi \in \Pi(\delta)$*.*

For π maximal in $\Pi(\delta)$, $\phi[\pi]$ is a literal. If $\phi[\pi] = x\, r\, y$, by Lemma 7, $\beta \models R_{xy} \in R^\pi_{xy} = \{r\}$ and therefore, by Lemma 6, $\alpha(x)\, r\, \alpha(y)$. If $\phi[\pi] = x[i]{=}y$, by Lemma 7, $\beta \models \mathit{Child}^i_x \supseteq \mathit{Lo}^\pi_{i,x} = \{y\}$ and therefore $y \in \mathit{Child}^i_x$ and by step (1) of the tree construction procedure $\alpha(y) = \alpha(x)i$. Similarly for the other cases.

If π is not maximal in $\Pi(\delta)$, then $\phi[\pi]$ is not a literal. If $\phi[\pi] = \phi_1 \wedge \phi_2$, then both $\pi 1, \pi 2 \in \Pi(\delta)$ and by induction hyp. $(\mathcal{M}^\tau, \alpha) \vdash [\pi i]\phi : \phi[\pi i]$ for both $i \in \{1, 2\}$. Therefore $(\mathcal{M}^\tau, \alpha) \vdash [\pi 1]\phi \cdot [\pi 2]\phi : \phi[\pi]$ according to the proof system, i.e. $(\mathcal{M}^\tau, \alpha) \vdash [\pi]\phi : \phi[\pi]$. If $\phi[\pi] = \phi_1 \vee \phi_2$, then $(\mathcal{M}^\tau, \alpha) \vdash [\pi i]\phi : \phi[\pi i]$ for one of $i \in \{1, 2\}$. Suppose $i = 1$: then we have $(\mathcal{M}^\tau, \alpha) \vdash \overleftarrow{[\pi i]}\phi : \phi[\pi]$, i.e. $(\mathcal{M}^\tau, \alpha) \vdash [\pi]\phi : \phi[\pi]$. Similarly for $i = 2$, which completes the proof.

5 Solving The CSP

Solving the CSP $[\![\phi]\!]$ is a highly combinatorial task and brute force enumeration is not practical. An effective means of pruning the search space is required so that not all possible variable assignments β need be investigated. Constraint propagation is such a technique and has proven very successful in a wide range of application domains, from academic applications in computational linguistics to hard industrial scheduling problems.

A CSP consists of a constraint C on a set of variables V_C. Each $x_i \in V_C$ takes values in a finite domain D_i. A solution of C is a variable assignment β, such that $\beta(x_i) \in D_i$, that makes C true. To solve C using constraint propagation, for each $x_i \in V_C$ we maintain $\widehat{x_i} \subseteq D_i$ representing the remaining set of possible values for x_i: i.e. $\beta(x_i) \in \widehat{x_i}$. Constraint propagation is a process of simple deterministic inference. Its role is to discover and remove from $\widehat{x_i}$ values that are inconsistent with C and cannot lead to a solution. It is said to perform *model elimination* because it prunes candidate models from consideration. When $\widehat{x_i} = \{v_i\}$, we say that x_i is *determined*, and $\beta(x_i) = v_i$.

When constraint propagation alone is insufficient to determine all variables in V_C, then a non-deterministic choice is required. This is known as a *distribution step*. A non-determined variable x_i is chosen: since $\widehat{x_i}$ is not a singleton, $\exists D, D' \neq \emptyset$ such that $\widehat{x_i} = D \uplus D'$. We non-deterministically perform either the update $\widehat{x_i} \leftarrow D$ or the update $\widehat{x_i} \leftarrow D'$. The search for solutions of C proceeds by alternating steps of deterministic constraint propagation and non-deterministic distribution.

Constraint programming (CP) is the computational paradigm which supports problem solving in the manner described above. In order to efficiently solve the CSP $[\![\phi]\!]$ of Section 3, the target programming language must not only support the constraints required by our encoding, but also implement their operational semantics in a way that guarantees strong propagation. In the following, we make these requirements precise: we introduce an abstract CP language and specify the expected propagation as a system of inference rules.

Let $\Delta = \{0\,..\,\mu\}$ be an integer interval for some sufficiently large practical limit μ. We assume a set of integer variables I, I_k, J with values in Δ, and a set of set variables S, S_k with values in 2^Δ. Integer and set variables are also both generically written X_k. We write D for a domain, i.e. a fixed subset of Δ, and i, j, n, m for particular integers in Δ. The syntax of our CP language is given in Figure 5.

$$
\begin{aligned}
\mathcal{B} &::= \mathsf{false} \mid X_1 = X_2 \mid I \in D \mid D \subseteq S \mid S \subseteq D \mid \mathcal{B}_1 \wedge \mathcal{B}_2\\
\mathcal{C} &::= \mathcal{B} \mid I_1 \leq I_2 \mid I \in S \mid S_1 \parallel S_2 \mid S_3 \subseteq S_1 \cup S_2 \mid\\
&\qquad X = \langle X_1 \ \ldots \ X_n\rangle[J] \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \mathcal{C}_1 \ \textbf{or} \ \mathcal{C}_2
\end{aligned}
$$

Fig. 5. CP Language

We distinguish between *basic constraints* $\mathcal{B}$ and non-basic constraints $\mathcal{C}$, or *propagators*. Basic constraints represent $\widehat{x_i}$ for each $x_i \in V_C$. For an integer variable I, $\widehat{I}$ is represented by the basic constraint $I \in D$. For a set variable

equality			
$X_1 = X_2 \wedge \mathcal{B}[X_j]$	$\rightarrow$	$\mathcal{B}[X_k]$	$\{j,k\} = \{1,2\}$
finite domain constraints			
$I \in D_1 \wedge I \in D_2$	$\rightarrow$	$I \in D_1 \cap D_2$	
$I \in \emptyset$	$\rightarrow$	false	
$I_1 \leq I_2 \wedge I_1 \in \{n\,..\,m\}$	$\rightarrow$	$I_2 \in \Delta \setminus \{1\,..\,n-1\}$	
$I_1 \leq I_2 \wedge I_2 \in \{n\,..\,m\}$	$\rightarrow$	$I_1 \in \Delta \setminus \{m+1\,..\,\mu\}$	
finite set constraints			
$I \in S \wedge S \subseteq D$	$\rightarrow$	$I \in D$	
$I \in S \wedge I \in \{i\}$	$\rightarrow$	$\{i\} \subseteq S$	
$D_1 \subseteq S \wedge D_2 \subseteq S$	$\rightarrow$	$D_1 \cup D_2 \subseteq S$	
$S \subseteq D_1 \wedge S \subseteq D_2$	$\rightarrow$	$S \subseteq D_1 \cap D_2$	
$D \subseteq S \wedge S \subseteq D' \wedge D \not\subseteq D'$	$\rightarrow$	false	
$S_1 \parallel S_2 \wedge D_j \subseteq S_j$	$\rightarrow$	$S_k \subseteq \Delta \setminus D_j$	$\{j,k\} = \{1,2\}$
$S_3 \subseteq S_1 \cup S_2 \wedge S_1 \subseteq D_1 \wedge S_2 \subseteq D_2$	$\rightarrow$	$S_3 \subseteq D_1 \cup D_2$	
$S_3 \subseteq S_1 \cup S_2 \wedge D_3 \subseteq S_3 \wedge S_j \subseteq D_j$	$\rightarrow$	$D_3 \setminus D_j \subseteq S_k$	$\{j,k\} = \{1,2\}$

Fig. 6. Main Propagation Rules

$$\frac{\mathcal{B} \wedge \mathcal{C} \ \rightarrow^* \ \mathsf{false}}{\mathcal{B} \wedge (\mathcal{C} \ \textbf{or} \ \mathcal{C}') \ \rightarrow \ \mathcal{C}'} \qquad \frac{\mathcal{B} \wedge \mathcal{C}' \ \rightarrow^* \ \mathsf{false}}{\mathcal{B} \wedge (\mathcal{C} \ \textbf{or} \ \mathcal{C}') \ \rightarrow \ \mathcal{C}}$$

Fig. 7. Disjunctive Propagator

finite domain selection constraint

$$\begin{array}{lcl}
I = \langle I_1 \ \ldots \ I_n \rangle[J] & \rightarrow & J \in \{1..n\} \\
I = \langle I_1 \ \ldots \ I_n \rangle[J] \ \wedge \ J \in D \ \wedge \{I_j \in D_j \mid j \in D\} & \rightarrow & I \in \cup\{D_j \mid j \in D\} \\
I = \langle I_1 \ \ldots \ I_n \rangle[J] \ \wedge \ J \in D \ \wedge \ I_j \in D_j \ \wedge \ D \cap D_j = \emptyset & \rightarrow & J \in \Delta \setminus \{j\} \\
I = \langle I_1 \ \ldots \ I_n \rangle[J] \ \wedge \ J \in \{j\} & \rightarrow & I = I_j
\end{array}$$

finite set selection constraint

$$\begin{array}{lcl}
S = \langle S_1 \ \ldots \ S_n \rangle[I] & \rightarrow & I \in \{1..n\} \\
S = \langle S_1 \ \ldots \ S_n \rangle[I] \ \wedge \ I \in D \ \wedge \{D_i \subseteq S_i \mid i \in D\} & \rightarrow & \cap\{D_i \mid i \in D\} \subseteq S \\
S = \langle S_1 \ \ldots \ S_n \rangle[I] \ \wedge \ I \in D \ \wedge \{S_i \subseteq D_i \mid i \in D\} & \rightarrow & S \subseteq \cup\{D_i \mid i \in D\} \\
S = \langle S_1 \ \ldots \ S_n \rangle[I] \ \wedge \ D \subseteq S \ \wedge \ S_i \subseteq D_i \ \wedge \ D \not\subseteq D_i & \rightarrow & I \in \Delta \setminus \{i\} \\
S = \langle S_1 \ \ldots \ S_n \rangle[I] \ \wedge \ S \subseteq D \ \wedge \ D_i \subseteq S_i \ \wedge \ D_i \not\subseteq D & \rightarrow & I \in \Delta \setminus \{i\} \\
S = \langle S_1 \ \ldots \ S_n \rangle[I] \ \wedge \ I \in \{i\} & \rightarrow & S = S_i
\end{array}$$

Fig. 8. Selection Propagator

S, representing all its possible values in 2^Δ is usually prohibitive both in memory and in processing: instead $\widehat{S}$ is represented by a lower bound and an upper bound, i.e. by basic constraints $D \subseteq S$ and $S \subseteq D'$. A set variable is determined when its lower and upper bounds are equal.

Constraint propagation may be formalized as a process of inferential saturation. The system of inference rules for the CP language of Figure 5 is given in Figures 6, 7, and 8. Alternatively, a propagator may be understood as a concurrent agent implementing a non-basic constraint: it observes the monotonically growing set of basic constraints, called the *store*, and derives new basic constraints according to the declarative semantics of its constraint. In this view, a constraint $\mathcal{C}$ of the CP language can be regarded as a collection of concurrent agents, and the inference rules specify the behavior of these agents.

While finite domain constraints are now standard and finite set constraints (Gervet, 1995; Müller and Müller, 1997; Müller, 2001) are gaining in popularity, our CP language includes two unusual constructs both serving the constraint-based treatment of disjunction: *disjunctive propagators* and *selection constraints.*

5.2 Disjunctive Propagators

In Logic Programming (LP), disjunction is handled solely by the non-deterministic exploration of alternatives. When encountering a disjunction $C_1 \vee$

C_2, a LP system immediately makes the non-deterministic decision of either attempting to solve C_1 or C_2. For problems of high combinatorial complexity, such a strategy of early commitment is usually disastrous. It is often preferable to delay the choice until sufficient information is available to reject one of the alternatives. That is the purpose of disjunctive propagators. They allow disjunction to be treated not as a choice point but as a constraint.

A disjunctive propagator ($\mathcal{C}$ **or** $\mathcal{C}'$) infers $\mathcal{C}'$ when $\mathcal{C}$ becomes inconsistent with the basic constraints derived so far. Its precise semantics are given in Figure 7 where we write $\mathcal{B} \wedge \mathcal{C} \rightarrow^* \mathsf{false}$ to mean that false is in the saturation of $\mathcal{B} \wedge \mathcal{C}$ under the propagation rules. Disjunctive propagators are supported by the concurrent constraint programming language Oz (Smolka, 1995; Mozart, 1999; Schulte, 2000).

5.3 Selection Constraints

A very common form of disjunction is selection out a finite collection of alternative values. It can be given more specific and effective support in the form of a constraint which we write $X = \langle X_1, \ldots, X_n \rangle[I]$. Its declarative semantics is simply $X = X_I$ and is logically equivalent to a n-ary disjunctive propagator:

$$(X = X_1 \wedge I = 1) \ \textbf{or} \ \ldots \ \textbf{or} \ (X = X_n \wedge I = n)$$

but gives you more. Consider the case $X_1 \in \{i_1, i_1'\}$, $X_2 \in \{i_2\}$, $X_3 \in \{i_3\}$, $I \in \{1, 3\}$: the constraint $X = \langle X_1, X_2, X_3 \rangle[I]$ is able to derive $X \in \{i_1, i_1', i_3\}$. This is known as *constructive disjunction*: information is lifted out of the remaining alternatives of the disjunction. While difficult and expensive to implement in the general case, it can be very efficiently supported for selection out of homogeneous sequences.

This powerful idea was first introduced in CHIP (Dincbas et al., 1988) for selection out of a sequence of integer values. Duchier (1999a) extended it to selection out of homogeneous sequences of finite set variables and described its application to the efficient treatment of lexical ambiguity when parsing with a dependency grammar. In Figure 8, we give the propagation rules for both sequences of finite domain variables and sequences of finite set variables. Selection constraints are available for Oz. [1]

[1] `http://www.mozart-oz.org/mogul/info/duchier/select.html`

A solver consists of a constraint program and a distribution strategy. The program provides the deterministic inference for model elimination performed during the constraint propagation step. The distribution strategy implements the non-deterministic search: at each distribution step, it is responsible for choosing a non-determined variable and non-deterministically splitting its domain.

Our encoding $[\![\phi]\!]$ has a direct reading as a CP program by translating $(\mathcal{C} \vee \mathcal{C}')$ to $(\mathcal{C}\,\mathbf{or}\,\mathcal{C}')$ and defining $S = S_1 \uplus S_2$ as $S_1 \parallel S_2 \wedge S_1 \subseteq S \wedge S_2 \subseteq S \wedge S \subseteq S_1 \cup S_2$. The design of a practical distribution strategy is more subtle.

First, note that it may not be necessary to determine all variables in order to guarantee satisfiability; e.g. if $\beta_1 \models \phi_1$, then we need not also assign values to the variables of ϕ_2 in order to guarantee the satisfiability of $\phi_1 \vee \phi_2$. More generally, when searching for a model $(\mathcal{M}^\tau, \alpha, \delta)$ of ϕ, for any $\pi \notin \Pi(\delta)$, the truth value of $\phi[\pi]$ is irrelevant to the truth value of ϕ. Therefore it is both unnecessary to enumerate the possible assignments to the variables X^π of $[\![\phi]\!]$ as well as undesirable to do so since this merely introduces spurious ambiguity.

For these reasons, a practical solver should not search for fully explicit solutions, but rather for *less explicit solved forms*. A solved form is to the satisfaction of a constraint C what a most general unifier is to the satisfaction of an equation $t_1 = t_2$ between first-order terms. A solved form for constraint C, is a function $\widehat{\beta}$ assigning a subset of D_i to $\widehat{x_i}$ for each $x_i \in V_C$, such that $C \wedge \{x_i \in \widehat{\beta}(\widehat{x_i}) \mid x_i \in V_C\} \not\to^* \mathsf{false}$ and satisfying a criterion sufficient to guarantee that it can be extended to a fully explicit solution. The criterion should allow $\widehat{\beta}$ to remain as unspecific as possible. For example, Duchier and Niehren (2000) proved that, for the conjunctive fragment, a solved form need only distinguish between $R_{xy} \in \{=, \lhd^+\}$ and $R_{xy} \notin \{=, \lhd^+\}$ and that doing so could save an exponential number of choice points over the search for fully explicit solutions.

The design of an economical solved form for our formalism will be the next step on the way to a practical implementation, but remains at present an open issue for further research.

6 Conclusions

In this article, we introduced the new formalism of *dominance constraints with Boolean connectives* which combines dominance constraints with a feature tree logic in the style of CFT (Smolka and Treinen, 1994). Our main contribution is a treatment of disjunction suitable for constraint propagation by reduction to the *selection constraint* (Duchier, 1999a).

By abandoning constructor trees in favor of feature trees, we obtained a language with more fine grained expressivity that facilitated the treatment of negation and allowed us to generalize the set-based representation of disjunctive information pioneered by *set operators* (Duchier and Niehren, 2000).

We gave a semantic account for our formalism by interpreting dominance constraints over tree structures. On the basis of that account, we described an encoding procedure to transform a dominance constraint ϕ into a CSP $[\![\phi]\!]$. We proved soundness and completeness of this encoding, i.e. that ϕ is satisfiable iff $[\![\phi]\!]$ is satisfiable.

Finally we described how to obtain a solver by interpreting $[\![\phi]\!]$ as a constraint program. The requirements placed on the target language were made precise by defining an abstract CP language and by stipulating the expected propagation as a system of inference rules. Oz is an example of a constraint programming language satisfying our requirements.

Future work will first involve the design of a criterion for less explicit solved forms that refrains from making unnecessary choices and leads to a solver needing less search. Second, we will evaluate the performance on the conjunctive fragment to determine the practical impact of supporting a more general formalism. Third, we will evaluate our technique on highly disjunctive applications such as parsing with D-tree grammars (Duchier and Thater, 1999).

Acknowledgments: the author is grateful to Alexander Koller and Joachim Niehren for their knowledgeable input and extends special thanks to the anonymous reviewers whose insightful suggestions and attention to detail greatly helped improve this article.

References

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

Blackburn, P., Gardent, C., Meyer-Viol, W., 1993. Talking about trees. In: Proceedings of the European Chapter of the Association of Computational Linguistics. Utrecht.

Dincbas, M., Van Hentenryck, P., Simonis, H., Aggoun, A., Graf, T., Berthier, F., Dec. 1988. The constraint logic programming language CHIP. In: Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88. Tokyo, Japan.

Duchier, D., July 1999a. Axiomatizing dependency parsing using set constraints. In: Sixth Meeting on Mathematics of Language. Orlando, Florida.

Duchier, D., Sep. 1999b. Set constraints in computational linguistics – solving tree descriptions. In: Workshop on Declarative Programming with Sets (DPS'99).

Duchier, D., Gardent, C., 1999. A constraint-based treatment of descriptions. In: Int. Workshop on Computational Semantics. Tilburg.

Duchier, D., Niehren, J., Jul. 2000. Dominance constraints with set operators. In: Proceedings of the First International Conference on Computational Logic (CL2000). LNCS. Springer.

Duchier, D., Thater, S., 1999. Parsing with tree descriptions: a constraint-based approach. In: Int. Workshop on Natural Language Understanding and Logic Programming. Las Cruces, New Mexico.

Egg, M., Niehren, J., Ruhrberg, P., Xu, F., 1998. Constraints over lambda-structures in semantic underspecification. In: Joint Conf. COLING/ACL.

Gardent, C., Webber, B., 1998. Describing discourse semantics. In: Proceedings of the 4th TAG+ Workshop. Philadelphia.

Gervet, C., Sep. 1995. Set Intervals in Constraint-Logic Programming: Definition and Implementation of a Language. Ph.D. thesis, Université de France-Compté, European Thesis.

Koller, A., Mehlhorn, K., Niehren, J., 3–6Oct. 2000a. A polynomial-time fragment of dominance constraints. In: Proceedings of the 38th Annual Meeting of the Association of Computational Linguistics. Hong Kong.

Koller, A., Niehren, J., 2000. On underspecified processing of dynamic semantics. In: Proceedings of COLING-2000. Saarbrücken.

Koller, A., Niehren, J., Treinen, R., 2000b. Dominance constraints: Algorithms and complexity. In: Logical Aspects of Comp. Linguistics 98. To appear in LNCS.

Marcus, M., 1987. Deterministic parsing and description theory. In: Whitelock, P., Wood, M., Somers, H., Johnson, R., Bennett, P. (Eds.), Linguistic Theory and Computer Applications. Academic Press.

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

Mozart, 1999. The Mozart Programming System `http://www.mozart-oz.org/`.

Müller, T., 2001. Constraint propagation in mozart. Doctoral dissertation, Universität des Saarlandes, Fachrichtung Informatik, Saarbrücken, Germany, in preparation.

Müller, T., Müller, M., 1997. Finite set constraints in Oz. In: Bry, F., Freitag, B., Seipel, D. (Eds.), 13. Workshop Logische Programmierung. Technische Universität München.

Muskens, R., 1995. Order-Independence and Underspecification. In: Groenendijk, J. (Ed.), Ellipsis, Underspecification, Events and More in Dynamic

Semantics. DYANA Deliverable R.2.2.C.

Rambow, O., Vijay-Shanker, K., Weir, D., 1995. D-tree grammars. In: Proceedings of ACL'95. MIT, Cambridge.

Rogers, J., Vijay-Shanker, K., 1992. Reasoning with descriptions of trees. In: Annual Meeting of the Association for Comp. Linguistics (ACL).

Schulte, C., 2000. Programming constraint services. Doctoral dissertation, Universität des Saarlandes, Fachrichtung Informatik, Saarbrücken, Germany, to appear in Lecture Notes in Artificial Intelligence, Springer-Verlag.

Smolka, G., 1995. The Oz Programming Model. In: van Leeuwen, J. (Ed.), Computer Science Today. Springer-Verlag, Berlin.

Smolka, G., Treinen, R., Apr. 1994. Records for logic programming. Journal of Logic Programming 18 (3), 229–258.

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