A Model Eliminative Treatment of Quantifier-Free Tree Descriptions

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

Abstract

Tree descriptions are widely used in computational linguistics for talking and reasoning about trees. For practical applications, it is essential to be able to decide satisfiability and enumerate solutions efficiently. This challenge cannot realistically be met by brute force enumeration. However it can be addressed very effectively by constraint propagation as provided by modern constraint technology.

Previously, we studied the conjunctive fragment of tree descriptions and showed how the problem of finding minimal models of a conjunctive tree description could be transformed into a constraint satisfaction problem (CSP) on finite set variables.

In this paper, we extend our account to the fragment that admits both negation and disjunction, but still leaves out quantification. Again we provide a reduction to a CSP. While our previous encoding introduced the reader to set constraints and disjunctive propagators, we now extend our arsenal with selection propagators.

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).

A tree description consists of 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. For practical applications, it is essential to be able to decide satisfiability and find solutions of tree descriptions efficiently. While the satisfiability problem was shown to be NP-complete (Koller et al., 2000), we have described previously how to derive efficient solvers, based on constraint programming, by transformation to a constraint satisfaction problem (Duchier and Gardent, 1999; Duchier, 1999b; Duchier and Niehren, 2000).

In (Duchier and Niehren, 2000), we described dominance constraints with set operators. Set operators contribute a controlled form of disjunction and negation that remains eminently well-suited for constraint propagation. The additional expressivity is essential both for applications and for specifying a complete system of inference rules. The language of dominance constraint with set operators is less expressive than general Boolean connectives: it merely admits as operators all relations that can be generated from dominance $\lhd^*$ by union, intersection, complementation, and inversion.

In the present paper, we propose to extend the account to a language with all Boolean connectives. In so doing, we combine and subsume previous work on dominance constraint as well as on feature trees (Smolka and Treinen, 1994).

In Section 2 we review previous work that attended to the conjunctive fragment of dominance logic and motivate a change of formalism in order to additionally accommodate negation. In Section 3 we present a new formalism for tree descriptions that admits all boolean connectives and formally provide its semantics. In Section 4 we precisely define the constraint programming language that we require in order to solve tree descriptions, and list all its constraint propagation rules. In particular, we give precise semantics for disjunctive propagators and selection propagators. Finally, in Section 5 we describe an encoding that turns a tree description into a constraint program in the language of Section 4.

2 Conjunctive Fragment

In this section, we review briefly the tree description formalisms that we studied previously, discuss the difficulties that accrue from admitting negation, and argue that they can be resolved by generalizing the language of tree descriptions in the style of the feature tree logic CFT (Smolka and Treinen, 1994).

A classical tree description ϕ is a conjunction of literals $x \lhd^* y$ and $x : f(x_1 \ \ldots \ x_n)$ where variables denote nodes in a solution tree:

$$\phi \quad ::= \quad x \lhd^* y \ \mid \ x : f(x_1 \ \ldots \ x_n) \ \mid \ \phi \wedge \phi'$$

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 $x_1, \ldots, x_n$.

The semantics of tree decriptions are given by interpretation over finite tree structures. A model $(\mathcal{M}^\tau, \alpha)$ of a description ϕ consists of a tree structure $\mathcal{M}^\tau$ and a variable assignment (or interpretation) α that maps each variable in ϕ to a node in $\mathcal{M}^\tau$, and such that $(\mathcal{M}^\tau, \alpha) \models \phi$ in the usual Tarskian sense. We elaborate further in section 3.

Tree descriptions with set operators. In (Duchier and Niehren, 2000), we presented an extended language of descriptions which generalizes $x \lhd^* y$ into $x \, R \, y$, where R denotes any one of the possible dominance relationships between two nodes.

$$\phi \quad ::= \quad x \, R \, y \ \mid \ x : f(x_1 \ \ldots \ x_n) \ \mid \ \phi \wedge \phi'$$
$$\text{where } R \subseteq \{=, \lhd^+, \rhd^+, \bot\}$$

Two nodes are either equal ($=$), one is a proper ancestor of the other ($\lhd^+$), or the other way around ($\rhd^+$), or they lie in disjoint subtrees ($\bot$). We write $x \, R \, y$ to say that the nodes denoted by x and y must be in a relationship corresponding to one of the relation symbols in R. In other words, $(\mathcal{M}^\tau, \alpha) \models x \, R \, y$ iff $(\mathcal{M}^\tau, \alpha) \models x \, r \, y$ for some $r \in R$. For instance, the literal $x \, \{=, \bot\} \, y$ expresses that the nodes denoted by x and y must either be equal or lie in disjoint subtrees.

Set expressions. For convenience, we admit syntactic sugar and allow to write $x \, S \, y$ where S is a set expression given by:

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

Obviously, every set expression can be translated to an equivalent set R of relation symbols. 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, this extended formalism allows a controlled form of negation and disjunction without admitting full boolean connectives.

Accommodating negation. While tree descriptions with set operators gave us a limited form of disjunction and negation, we are now interested in permitting arbitrary disjunctions and negations. For negation, we need to consider the 3 cases of the abstract syntax:

1. $\neg(x\ R\ y)$ is equivalent to $x\ \neg R\ y$ and thus already handled by complementation of R.
2. $\neg(\phi_1 \wedge \phi_2)$ is equivalent to $\neg\phi_1 \vee \neg\phi_2$, thus will be handled by our treatment of disjunction.
3. Only $\neg(x : f(x_1\ \ldots\ x_n))$ is problematic: it is satisfied when either x is not labeled by f, or does not have arity n, or its ith child is not x_i.

We address the latter problem by expressing $x : f(x_1\ \ldots\ x_n)$ as a conjunction of simpler constraints in the style of CFT (Smolka and Treinen, 1994):

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

$|x| = n$ is an arity constraint, $x{:}f$ is a label or sort constraint, and $x[i] = y$ is a feature constraint. The problematic case becomes:

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

Koller et al. (2000) also briefly consider possible extensions of dominance constraint with finite signatures to admit disjunctions and negations. They suggest that negation can be handled by expansion into the disjunction of all other cases:

$$\neg(x : f(x_1\ \ldots\ x_n)) \quad \equiv \quad (\bigvee_{g \neq f \in \Sigma} x : g(x'_1\ \ldots\ x'_{\mathsf{ar}(g)})) \vee (x : f(x''_1\ \ldots\ x''_n) \wedge \bigvee_{1 \le i \le n} x''_i\ \neg{=}\ x_i)$$

We reject their suggestion as impractical: in order to achieve the same expressivity as ours, they must solve much larger problems. Furthermore, they do not propose an efficient way to process disjunctions, but merely suggest that non-deterministic search could in principle handle it. They are of course well aware that for a problem of such combinatorial complexity, pure non-deterministic search is utterly impractical. One essential contribution of the present paper is to demonstrate how to reduce a disjunctive description to a form amenable to effective model elimination through constraint propagation.

3 Full Boolean Fragment

We propose a new language of tree descriptions inspired both by the language with set operators of Duchier and Niehren (2000) and the feature tree logic CFT of Smolka and Treinen (1994). Its abstract syntax is given by:

$$\phi ::= x\ R\ y \ \mid\ |x| = n \ \mid\ x : f \ \mid\ x[i] = y \ \mid\ \phi_1 \wedge \phi_2 \ \mid\ \phi_1 \vee \phi_2 \ \mid\ \neg\phi$$
$$\text{where } R \subseteq \{=, \lhd^+, \rhd^+, \perp\}$$

The admission of disjunction and negation makes the set operator extension of (Duchier and Niehren, 2000) obsolete since $x\,R_1 \cup R_2\,y \equiv x\,R_1\,y \vee x\,R_2\,y$ and $x\,\neg R\,y \equiv \neg(x\,R\,y)$. Nonetheless, we choose to retain the notion since it facilitates the reduction to a constraint satisfaction problem.

The semantics of tree descriptions are given by interpretation over tree structures and we make this idea formal in the remainder of the present section.

We assume a signature Σ of function symbols written $f, g, a, b, \ldots$ and equipped with an arity $\mathsf{ar}(f) \geq 0$. We assume that Σ contains at least one constant and one function symbol of arity ≥ 2.

Tree, nodes, and dominance. 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 is a pair (D, L) of a finite tree domain D and a labeling function $L : D \to \Sigma$ with the property that

$$\begin{array}{l}\mathcal{B} ::= \mathsf{true} \mid \mathsf{false} \mid X_1 = X_2 \mid I \in D \mid i \in S \mid i \notin S \mid \mathcal{B}_1 \wedge \mathcal{B}_2 \qquad (D \subseteq \Delta)\\ \mathcal{C} ::= \mathcal{B} \mid I_1 \leq I_2 \mid S_1 \parallel S_2 \mid S_3 \subseteq S_1 \cup S_2 \mid \mathcal{C}_1 \wedge \mathcal{C}_2 \mid \\ \qquad \mathcal{C}_1 \ \mathbf{or} \ \mathcal{C}_2 \mid I = \langle I_1 \ \ldots \ I_n \rangle[J] \mid S = \langle S_1 \ \ldots \ S_n \rangle[J]\end{array}$$

Figure 1: Constraint Language

$$\begin{array}{rcll}
\multicolumn{4}{l}{\textbf{equality}}\\
X_1 = X_2 \wedge \mathcal{B}[X_j] & \rightarrow & \mathcal{B}[X_k] & \{j,k\} = \{1,2\}\\
\multicolumn{4}{l}{\textbf{finite domain integer constraints}}\\
I \in D_1 \wedge I \in D_2 & \rightarrow & I \in D_1 \cap D_2 & \\
I \in \emptyset & \rightarrow & \mathsf{false} & \\
I_1 \leq I_2 \wedge I_1 \in \{n \,..\, m\} & \rightarrow & I_2 \in \Delta \setminus \{1 \,..\, n-1\} & \\
I_1 \leq I_2 \wedge I_2 \in \{n \,..\, m\} & \rightarrow & I_1 \in \Delta \setminus \{m+1 \,..\, \mu\} & \\
\multicolumn{4}{l}{\textbf{finite set constraints}}\\
i \in S \wedge i \notin S & \rightarrow & \mathsf{false} & \\
S_1 \parallel S_2 \wedge i \in S_j & \rightarrow & i \notin S_k & \{j,k\} = \{1,2\}\\
S_3 \subseteq S_1 \cup S_2 \wedge i \notin S_1 \wedge i \notin S_2 & \rightarrow & i \notin S_3 & \\
S_3 \subseteq S_1 \cup S_2 \wedge i \in S_3 \wedge i \notin S_j & \rightarrow & i \in S_k & \{j,k\} = \{1,2\}
\end{array}$$

Figure 2: Main Propagation Rules

all $\pi \in D$ and $k \geq 1$ satisfy $\pi k \in D$ iff $k \leq \mathsf{ar}(L(\pi))$, i.e. that each node has precisely as many children as required by the arity of the function symbol with which it is labeled. If τ is a tree, we write D_τ for its domain and L_τ for its labeling function.

Tree structure and set operators. A tree structure is a first-order structure $\mathcal{M}^\tau$ representing a tree τ by the relations between its nodes. The domain of $\mathcal{M}^\tau$ is the tree domain D_τ. For each function symbol $f \in \Sigma$, $\mathcal{M}^\tau$ contains a relation $:f$ of arity $\mathsf{ar}(f)+1$ such that (using infix notation) $\pi{:}f(\pi_1 \ \ldots \ \pi_n)$ holds in $\mathcal{M}^\tau$ iff $L_\tau(\pi) = f$ and $\pi_i = \pi i$ for all $1 \leq i \leq n = \mathsf{ar}(f)$.

We consider tree structures with all relations generated from dominance $\lhd^*$ by inversion $^{-1}$ (i.e. argument swapping), union $\cup$, intersection $\cap$, and complementation $\neg$. We define inverse dominance $\rhd^*$ by $\lhd^{*-1}$, equality $=$ by $\lhd^* \cap \rhd^*$, inequality $\neq$ by $\neg=$, proper dominance $\lhd^+$ by $\lhd^* \cap \neq$, inverse proper dominance $\rhd^+$ by $\lhd^{+-1}$, and disjointness $\perp$ by $\neg\lhd^* \cap \neg\rhd^*$.

Semantics of tree descriptions. Tree descriptions are interpreted in the class of tree structure over Σ. For instance a description $x \ \{=, \perp\} \ y$ is satisfied by a tree structure where the nodes denoted by x and y are either equal or lie in disjoint subtrees. In general, a relation symbol R is interpreted as the relation $\cup R$ of $\mathcal{M}^\tau$ for some tree τ. We write $\mathsf{Vars}(\phi)$ for the set of variables occurring in ϕ. A solution of a description ϕ is a pair $(\mathcal{M}^\tau, \alpha)$ of a tree structure $\mathcal{M}^\tau$ and a variable assignment $\alpha : \mathsf{Vars}(\phi) \to D_\tau$. We write $(\mathcal{M}^\tau, \alpha) \models \phi$ if ϕ is satisfied by $(\mathcal{M}^\tau, \alpha)$ in the usual Tarskian way, where the CFT-style descriptions are interpreted as follows:

$$\begin{array}{lcl}
(\mathcal{M}^\tau, \alpha) \models |x| = n & \equiv & \mathsf{ar}(L_\tau(\alpha(x))) = n\\
(\mathcal{M}^\tau, \alpha) \models x : f & \equiv & L_\tau(\alpha(x)) = f\\
(\mathcal{M}^\tau, \alpha) \models x[i] = y & \equiv & \alpha(y) = \alpha(x)i
\end{array}$$

4 Constraint Language

Our approach for solving a tree description is to transform it into an equivalent constraint satisfaction problem. More precisely, we transform a description ϕ into a constraint $[\![\phi]\!]$ of a constraint language. Solutions of $[\![\phi]\!]$ can then be enumerated efficiently by alternating steps of *propagation*

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

Figure 3: Disjunctive Propagator

finite domain selection constraint		
$I = \langle I_1 \;\ldots\; I_n \rangle[J]$	$\rightarrow$	$J \in \{1..n\}$
$I = \langle I_1 \;\ldots\; I_n \rangle[J] \;\wedge\; J \in D \;\wedge\; \forall j \in D \; I_j \in D_j$	$\rightarrow$	$I \in \cup\{D_j \mid j \in D\}$
$I = \langle I_1 \;\ldots\; I_n \rangle[J] \;\wedge\; J \in D \;\wedge\; I_j \in D_j \;\wedge\; D \cap D_j = \emptyset$	$\rightarrow$	$j \neq J$
$I = \langle I_1 \;\ldots\; I_n \rangle[J] \;\wedge\; j = J$	$\rightarrow$	$I = I_j$
finite set selection constraint		
$S = \langle S_1 \;\ldots\; S_n \rangle[I]$	$\rightarrow$	$I \in \{1..n\}$
$S = \langle S_1 \;\ldots\; S_n \rangle[I] \;\wedge\; I \in D \;\wedge\; \forall i \in D \; j \in S_i$	$\rightarrow$	$j \in S$
$S = \langle S_1 \;\ldots\; S_n \rangle[I] \;\wedge\; I \in D \;\wedge\; \forall i \in D \; j \notin S_i$	$\rightarrow$	$j \notin S$
$S = \langle S_1 \;\ldots\; S_n \rangle[I] \;\wedge\; j \in S \;\wedge\; j \notin S_i$	$\rightarrow$	$i \neq I$
$S = \langle S_1 \;\ldots\; S_n \rangle[I] \;\wedge\; j \notin S \;\wedge\; j \in S_i$	$\rightarrow$	$i \neq I$
$S = \langle S_1 \;\ldots\; S_n \rangle[I] \;\wedge\; i = I$	$\rightarrow$	$S = S_i$

Figure 4: Selection Propagator

and *distribution*. In this section, we present a constraint language sufficient for our purposes and specify the constraint propagation behavior we require as a system of inference rules. The concurrent constraint programming language Oz is a practical implementation of such a language (Smolka, 1995; Mozart, 1999).

Let $\Delta = \{1 \;\ldots\; \mu\}$ be a integer interval 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, J, or K, and a set of *set variables* with values in 2^Δ and ranged over by S. Integer and set variables are also both denoted by X. We write D for a *domain*, i.e. a given fixed subset of Δ.

Basic and non-basic constraints. We distinguish between *basic constraints* $\mathcal{B}$ which can be represented directly in the constraint store and *non-basic constraints* $\mathcal{C}$ which act as propagators (see Fig 1). A propagator implements a set of inference rules that derive new basic constraints. For example, the non-basic *disjointness* constraint $S_1 \parallel S_2$ implements the rules:

$$S_1 \parallel S_2 \;\wedge\; i \in S_j \quad \rightarrow \quad i \notin S_k \qquad \text{for } \{j,k\} = \{1,2\}$$

Whenever a basic constraint $i \in S_1$ (resp. $i \in S_2$) is derived, the disjointness propagator infers $i \notin S_2$ (resp. $i \notin S_1$). The rules for finite domain and finite set constraints are given in (Fig 2).

Abbreviations. $S_1 \parallel S_2$ has the semantics $S_1 \cap S_2 = \emptyset$. We write $I = i$ for $I \in \{i\}$, $I \neq i$ for $I \in \Delta \setminus \{i\}$, $S = D$ for $\wedge\{i \in S \mid i \in D\} \wedge \{i \notin S \mid i \in \Delta \setminus D\}$, $S_1 \subset S_2$ for $S_1 \subseteq S_2 \cup S_3 \;\wedge\; S_3 = \emptyset$, and $S = S_1 \uplus S_2$ for $S_1 \parallel S_2 \;\wedge\; S \subseteq S_1 \cup S_2 \;\wedge\; S_1 \subseteq S \;\wedge\; S_2 \subseteq S$ (i.e. $\uplus$ represents disjoint union aka partition).

Propagation and distribution. Propagation performs cheap deterministic inference, but is not complete. In order to enumerate the solutions of a constraint $\mathcal{C}$, search is required and may be specified by application-dependent *distribution* rules. For our present purpose, we need only consider distribution rules of the form:

$$I \in D_1 \uplus D_2 \quad \rightarrow \quad I \in D_1 \;\vee\; I \in D_2 \qquad \text{for } D_1, D_2 \neq \emptyset \qquad (\delta_1)$$

$$S \subset D \quad \rightarrow \quad i \in S \;\vee\; i \notin S \qquad \text{for } i \in D \qquad (\delta_2)$$

The disjunction in a distribution rule is interpreted non-deterministically. Thus, rule (δ_1) non-deterministically infers either $I \in D_1$ or $I \in D_2$. A constraint program consists of a constraint $\mathcal{C}$ and a set of distribution rules. Solutions are derived by alternating steps of propagation (i.e. saturation under propagation rules) and distribution (i.e. the non-deterministic application of a distribution rule). The choice of distribution rule, of variable I, and of domain partition $D_1 \uplus D_2$ is determined by a *search strategy*, but this is outside the scope of the present paper.

Disjunctive propagator. In Logic Programming, disjunction is handled solely by the non-deterministic exploration of alternatives. For problems of high combinatorial complexity, such a strategy of early commitment is highly undesirable. Modern constraint programming offers a remarkable alternative: the possibility to consider disjunction not as a choice point but as a constraint.

A disjunctive propagator ($\mathcal{C}$ **or** $\mathcal{C}'$) infers $\mathcal{C}'$ if it can be shown that $\mathcal{C}$ is inconsistent with the basic constraints derived so far. The precise semantics of the disjunctive propagator are given in (Fig 3), 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.

Selection propagator. A very common form of disjunction is selection out of a finite collection of alternative values. It can be given more specific and effective support in the form of a constraint which we write:

$$X = \langle X_1 \ \ldots \ X_n \rangle[I]$$

where $\langle X_1 \ \ldots \ X_n \rangle$ represents a sequence and the notation above was chosen for its intuitive similarity with array-lookup. The declarative semantics are simply $X{=}X_I$. Such a constraint might be implemented with an n-ary disjunctive propagator:

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

However, it is possible to extract more precise information concerning X out of the remaining (not yet inconsistent) alternatives of the disjunction. For example, from:

$$\begin{array}{ll} & (X = k_1 \wedge I = 1) \\ \textbf{or} & (X = k_2 \wedge I = 2) \\ \textbf{or} & (X = k_3 \wedge I = 3) \end{array}$$

and $I \neq 2$, we should be able to derive $X \in \{k_1, k_3\}$. This is known as *constructive disjunction*. While difficult 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 (Fig 4), we give the propagation rules for both sequences of finite domain variables and sequences of finite set variables.

5 Reduction To A Constraint Satisfaction Problem

Our approach transforms a tree description ϕ into a constraint $[\![\phi]\!]$ in the language presented above in section 4, and thereby turns the task of finding solutions of ϕ into an equivalent constraint satisfaction problem (CSP). $[\![\phi]\!]$ consists of 3 parts:

$$[\![\phi]\!] = \bigwedge_{x \in \mathsf{Vars}(\phi)} \mathsf{A}_1(x) \bigwedge_{x,y \in \mathsf{Vars}(\phi)} \mathsf{A}_2(x,y) \quad \wedge \ \mathsf{A}_3[\![\phi]\!]$$

$\mathsf{A}_1(x)$ introduces a node representation for each variable x in ϕ, $\mathsf{A}_2(x,y)$ axiomatizes the well-formedness (i.e. treeness) of the relations between these nodes, and $\mathsf{A}_3[\![\phi]\!]$ encodes the specific restrictions imposed by ϕ.

In a solution $(\mathcal{M}^\tau, \alpha)$ of ϕ, every variable x is mapped to a node $\alpha(x)$ of $\mathcal{M}^\tau$. Thus from the point of view of x, the set $\mathsf{Vars}(\phi)$ is partitioned into 4 disjoint subsets: all variables equal to x (i.e. mapped to the same node $\alpha(x)$), all ancestors, all descendents, and all others (i.e. mapped to nodes in disjoint subtrees). Our technique is based on introducing explicit variables for these sets and expressing the constraints that they must satisfy.

5.1 Representation

We assume that we look for solutions in a tree structure with finite signature Σ. Let MAX be the maximum constructor arity in Σ. For each formal variable x in ϕ we choose a distinct integer ι_x to represent it and introduce $7 + 2 \times \text{MAX}$ constraint variables written Eq_x, Up_x, $Down_x$, $Side_x$, $Equp_x$, $Eqdown_x$, $Child_x^i$ and $Down_x^i$ for $1 \leq i \leq \text{MAX}$, and , $Genesis_x$, as well as 3 integer constraint variables $Label_x$, $Arity_x$. First, we state $x = x$:

$$\iota_x \in Eq_x \tag{1}$$

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 $\mathcal{I} = \{\iota_x \mid x \in \mathsf{Vars}(\phi)\}$ for the set of integers encoding $\mathsf{Vars}(\phi)$, we have:

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

As described in (Duchier and Niehren, 2000), we can and should improve propagation by introducing $Eqdown_x$ and $Equp_x$ as intermediate results:

$$\mathcal{I} = Eqdown_x \uplus Up_x \uplus Side_x \tag{2}$$

$$\mathcal{I} = Equp_x \uplus Down_x \uplus Side_x \tag{3}$$

$$Eqdown_x = Eq_x \uplus Down_x \tag{4}$$

$$Equp_x = Eq_x \uplus Up_x \tag{5}$$

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

$$Down_x = \biguplus_{1 \leq i \leq \text{MAX}} Down_x^i \tag{6}$$

$Child_x^i$ encodes the set of variables occurring as x's ith child:

$$Child_x^i \subseteq Down_x^i \tag{7}$$

We choose a bijection $\ell : \Sigma \to \{1 \,..\, |\Sigma|\}$ and encode a constructor $f \in \Sigma$ by the integer $\ell(f)$. The intent is that the literal $x : f$ should correspond to the constraint $Label_x = \ell(f)$.

$$Label_x \in \{1 \,..\, |\Sigma|\} \tag{8}$$

The arity of a node depends on its constructor. Thus, $Arity_x$ and $Label_x$ are related by the following selection constraint:

$$Arity_x = \langle \mathsf{ar}(\ell^{-1}(1)), \ldots, \mathsf{ar}(\ell^{-1}(|\Sigma|)) \rangle [Label_x] \tag{9}$$

A node must have precisely as many children as required by the arity of its constructor:

$$Arity_x < i \Rightarrow |Down_x^i| = 0 \tag{10}$$

Where we define:

$$I < i \Rightarrow \mathcal{C} \quad \equiv \quad (I < i \ \wedge \ \mathcal{C}) \ \textbf{or} \ (I \geq i)$$

(10) is not an equivalence because we do not want to eliminate solutions that require more nodes than there are variables in ϕ.

$\mathsf{A}_1(x)$ is simply the conjunction of the constraints presented above:

$$\mathsf{A}_1(x) \equiv (1) \wedge \ \ldots \ \wedge (10)$$

$$C_{xy} \in \{1,2,3,4\} \tag{11}$$
$$\mathsf{B}[\![x = y]\!] \wedge C_{xy} = 1 \;\; \textbf{or} \;\; C_{xy} \neq 1 \wedge \mathsf{B}[\![x \neg= y]\!] \tag{12}$$
$$\mathsf{B}[\![x \lhd^+ y]\!] \wedge C_{xy} = 2 \;\; \textbf{or} \;\; C_{xy} \neq 2 \wedge \mathsf{B}[\![x \neg\lhd^+ y]\!] \tag{13}$$
$$\mathsf{B}[\![x \rhd^+ y]\!] \wedge C_{xy} = 3 \;\; \textbf{or} \;\; C_{xy} \neq 3 \wedge \mathsf{B}[\![x \neg\rhd^+ y]\!] \tag{14}$$
$$\mathsf{B}[\![x \perp y]\!] \wedge C_{xy} = 4 \;\; \textbf{or} \;\; C_{xy} \neq 4 \wedge \mathsf{B}[\![x \neg\perp y]\!] \tag{15}$$

Figure 5: Well-formedness clauses

$$\begin{aligned}
\mathsf{B}[\![x = y]\!] &= Eq_x = Eq_y \wedge Up_x = Up_y \wedge Down_x = Down_y \wedge Side_x = Side_y \wedge \\
&\quad Equp_x = Equp_y \wedge Eqdown_x = Eqdown_y \wedge \\
&\quad Label_x = Label_y \wedge Arity_x = Arity_y \wedge Genesis_x = Genesis_y \wedge \\
&\quad Down^i_x = Down^i_y \wedge Child^i_x = Child^i_y \\
\mathsf{B}[\![x \neg= y]\!] &= Eq_x \parallel Eq_y \\
\mathsf{B}[\![x \lhd^+ y]\!] &= Eqdown_y \subseteq Down_x \wedge Equp_x \subseteq Up_y \wedge Side_x \subseteq Side_y \\
\mathsf{B}[\![x \neg\lhd^+ y]\!] &= Eq_x \parallel Up_y \wedge Down_x \parallel Eq_y \\
\mathsf{B}[\![x \perp y]\!] &= Eqdown_x \subseteq Side_y \wedge Eqdown_y \subseteq Side_x \\
\mathsf{B}[\![x \neg\perp y]\!] &= Eq_x \parallel Side_y \wedge Side_x \parallel Eq_y
\end{aligned}$$

Figure 6: Set constraints characteristic of each case

$$Genesis_x \in \mathcal{G} \tag{16}$$
$$\begin{array}{ll} & \iota_{x[i]} \in Genesis_y \wedge Child^i_x = Eq_y \wedge Down^i_x = Eqdown_y \wedge Up_y = Equp_x \\ \textbf{or} & \iota_{x[i]} \notin Genesis_y \wedge Child^i_x \parallel Eq_y \end{array} \tag{17}$$

Figure 7: Genesis clauses

5.2 Well-Formedness

In a tree, the relationship that obtains between the nodes denoted by x and y must be either $=$, $\lhd^+$, $\rhd^+$ or $\perp$. We encode this choice explicitly with a finite domain variable C_{xy} which we call a *choice variable* and contribute the clauses of Fig 5. Where $\mathsf{B}[\![x\ r\ y]\!]$ are the set constraints characteristic of the case $x\ r\ y$ as described in Fig 6.

Finally, we state that for any two variables x, y and integer i, $x[i]{=}y$ either holds or not. That is, for each pair (x, i) with $x \in \mathsf{Vars}(\phi)$ and $1 \leq i \leq \text{MAX}$ we choose a distinct integer $\iota_{x[i]}$ to encode it. We define $\mathcal{G} = \{\iota_{x[i]} \mid x \in \mathsf{Vars}(\phi), 1 \leq i \leq \text{MAX}\}$, and contribute the clauses of Fig 7.

$$\mathsf{A}_2(x, y) \equiv (11) \wedge (12) \wedge (13) \wedge (14) \wedge (15) \wedge (16) \bigwedge_{1 \leq i \leq \text{MAX}} (17)$$

5.3 Problem Specific Constraints

The last part $\mathsf{A}_3[\![\phi]\!]$ of the translation forms the additional problem specific constraints that further restrict the admissibility of well-formed solutions and admits only those which are models of ϕ.

We will make use of the notion of a *restriction* σ which maps constraint variables to restrictions on their domains. For example, each literal shown below contributes a restriction with the property shown to its right.

$$
\begin{array}{ll}
\mathsf{C}[\![\phi_1 \vee \phi_2]\!]^J_\rho & = \langle \varphi_1 \wedge \varphi_2 \wedge I \in \{1,2\} \wedge J\rho I, \langle \sigma_1, \sigma_2 \rangle [I] \rangle \\
\multicolumn{2}{l}{\quad \text{where } I \text{ is a fresh variable, } \langle \varphi_1, \sigma_1 \rangle = \mathsf{C}[\![\phi_1]\!]^I_\leq \text{ and } \langle \varphi_2, \sigma_2 \rangle = \mathsf{C}[\![\phi_2]\!]^I_\geq} \\
\mathsf{C}[\![\phi_1 \wedge \phi_2]\!]^J_\rho & = \langle \varphi_1 \wedge \varphi_2, \sigma_1 \& \sigma_2 \rangle \\
\multicolumn{2}{l}{\quad \text{where } \langle \varphi_1, \sigma_1 \rangle = \mathsf{C}[\![\phi_1]\!]^J_\rho \text{ and } \langle \varphi_2, \sigma_2 \rangle = \mathsf{C}[\![\phi_2]\!]^J_\rho \text{ and } \sigma_1 \& \sigma_2 \text{ is defined in Fig 9}} \\
\mathsf{C}[\![\neg\neg\phi]\!] & = \mathsf{C}[\![\phi]\!] \\
\mathsf{C}[\![\neg(\phi_1 \wedge \phi_2)]\!] & = \mathsf{C}[\![\neg\phi_1 \vee \neg\phi_2]\!] \\
\mathsf{C}[\![\neg(\phi_1 \vee \phi_2)]\!] & = \mathsf{C}[\![\neg\phi_1 \wedge \neg\phi_2]\!] \\
\mathsf{C}[\![x\ R\ y]\!] & = \langle \mathsf{true}, \top[C_{xy} \mapsto \{\mathsf{D}[\![r]\!] \mid r \in R\}] \rangle \\
\mathsf{C}[\![\neg(x\ R\ y)]\!] & = \mathsf{C}[\![x\ \neg R\ y]\!] \\
\mathsf{C}[\![|x| = n]\!] & = \langle \mathsf{true}, \top[\mathit{Arity}_x \mapsto \{n\}] \rangle \\
\mathsf{C}[\![\neg(|x| = n)]\!] & = \langle \mathsf{true}, \top[\mathit{Arity}_x \mapsto \mathsf{ar}(\Sigma) \setminus \{n\}] \rangle \\
\mathsf{C}[\![x : f]\!] & = \langle \mathsf{true}, \top[\mathit{Label}_x \mapsto \{\ell(f)\}] \rangle \\
\mathsf{C}[\![\neg(x : f)]\!] & = \langle \mathsf{true}, \top[\mathit{Label}_x \mapsto \ell(\Sigma) \setminus \{\ell(f)\}] \rangle \\
\mathsf{C}[\![x[i] = y]\!] & = \langle \mathsf{true}, \top[\mathit{Child}^i_x \mapsto \langle \{y\}, \emptyset \rangle] \rangle \\
\mathsf{C}[\![\neg(x[i] = y)]\!] & = \langle \mathsf{true}, \top[\mathit{Child}^i_x \mapsto \langle \emptyset, \{y\} \rangle] \rangle
\end{array}
\qquad
\left\{
\begin{array}{ll}
\mathsf{D}[\![=]\!] & = 1 \\
\mathsf{D}[\![\lhd^+]\!] & = 2 \\
\mathsf{D}[\![\rhd^+]\!] & = 3 \\
\mathsf{D}[\![\bot]\!] & = 4
\end{array}
\right.
$$

Figure 8: Computing restrictions

$$
\begin{array}{ll}
(\sigma_1 \& \sigma_2)(C_{xy}) & = \sigma_1(C_{xy}) \cap \sigma_2(C_{xy}) \\
(\sigma_1 \& \sigma_2)(\mathit{Arity}_x) & = \sigma_1(\mathit{Arity}_x) \cap \sigma_2(\mathit{Arity}_x) \\
(\sigma_1 \& \sigma_2)(\mathit{Label}_x) & = \sigma_1(\mathit{Label}_x) \cap \sigma_2(\mathit{Label}_x) \\
(\sigma_1 \& \sigma_2)(\mathit{Child}^i_x) & = \langle \mathit{In}_1 \cup \mathit{In}_2, \mathit{Out}_1 \cup \mathit{Out}_2 \rangle \\
\multicolumn{2}{r}{\text{where } \langle \mathit{In}_1, \mathit{Out}_1 \rangle = \sigma_1(\mathit{Child}^i_x)} \\
\multicolumn{2}{r}{\langle \mathit{In}_2, \mathit{Out}_2 \rangle = \sigma_2(\mathit{Child}^i_x)}
\end{array}
$$

Figure 9: Accumulating Restrictions

$$
\begin{array}{lll}
x \lhd^* y & \sigma(C_{xy}) & = \{1,2\} \\
x : f & \sigma(\mathit{Label}_x) & = \{\ell(f)\} \\
|x| = n & \sigma(\mathit{Arity}_x) & = \{n\}
\end{array}
$$

Our intention is to translate a description ϕ into a corresponding restriction σ such that the problem specific constraints are all of the form:

$$C_{xy} \in \sigma(C_{xy}) \tag{18}$$

$$\mathit{Label}_x \in \sigma(\mathit{Label}_x) \tag{19}$$

$$\mathit{Arity}_x \in \sigma(\mathit{Arity}_x) \tag{20}$$

Additionally, literals such as $x[i] = y$ and $\neg(x[i] = y)$ give rise to restrictions on variables Child^i_x. Such a restriction is expressed as a pair $\langle \mathit{In}^i_x, \mathit{Out}^i_x \rangle$ where In^i_x represents the set of variables that must be included in Child^i_x and Out^i_x the set of those that must be excluded from it. It yields the problem specific constraint:

$$\mathit{In}^i_x \subseteq \mathit{Child}^i_x \wedge \mathit{Out}^i_x \parallel \mathit{Child}^i_x \tag{21}$$

Disjunctions are handled by means of the selection constraint. If descriptions ϕ_1 and ϕ_2 translate respectively to restrictions σ_1 and σ_2, then $\phi_1 \vee \phi_2$ translates to $\langle \sigma_1, \sigma_2 \rangle [I]$ where I is a new finite domain variable. The selection constraint applied to restrictions is interpreted point-wise as follows:

$$\langle \sigma_1, \sigma_2 \rangle [I](X) = \langle \sigma_1(X), \sigma_2(X) \rangle [I]$$

If approached naïvely, nested disjunctions result in spurious indeterminacy. Consider:

$$(\phi_1 \vee \phi_2) \vee (\phi'_1 \vee \phi'_2)$$

The description above results in a restriction of the form:

$$\langle\langle\sigma_1, \sigma_2\rangle[I], \langle\sigma'_1, \sigma'_2\rangle[I']\rangle[J]$$

where J controls the outermost disjunction, I the nested disjunction on the left, and I' the nested disjunction on the right. If $J = 1$, I' may still take indifferently values 1 or 2 eventhough this choice has become irrelevant. In order to remove such spurious indeterminacy, we add the following constraints:

$$J \leq I \quad \wedge \quad J \geq I'$$

Thus, when $J = 1$ then I' must be 1 and when $J = 2$, I must be 2.

We can now proceed to define formally the translation of a description ϕ into a restriction. For this purpose, we use $\mathsf{C}[\![\phi]\!]^J_\rho$ where ρ is either $\leq$ or $\geq$. J and ρ are used to remove spurious indeterminacies as outlined above. $\mathsf{C}[\![\phi]\!]^J_\rho$ actually returns a pair:

$$\langle\varphi, \sigma\rangle = \mathsf{C}[\![\phi]\!]^J_\rho$$

where φ is a constraint and σ is a restriction. We write $\top$ for the most general restriction, which is defined point-wise as follows:

$$\begin{array}{ll} \top(C_{xy}) & = \{1, 2, 3, 4\} \\ \top(\mathit{Arity}_x) & = \{0 \,..\, \mathrm{MAX}\} \\ \top(\mathit{Label}_x) & = \{1 \,..\, |\Sigma|\} \\ \top(\mathit{Child}^i_x) & = \langle\emptyset, \emptyset\rangle \end{array}$$

and we write $\sigma[X \mapsto V]$ for the restriction such that:

$$\sigma[X \mapsto V](Y) = \begin{cases} V & \text{if } Y \text{ is } X \\ \sigma(Y) & \text{otherwise} \end{cases}$$

We can now put it all together and define $\mathsf{C}[\![\phi]\!]^J_\rho$ in Fig 8. If we pose $\langle\varphi, \sigma\rangle = \mathsf{C}[\![\phi]\!]^1_\leq$ and $\langle \mathit{In}^i_x, \mathit{Out}^i_x\rangle = \sigma(\mathit{Child}^i_x)$, the problem specific constraints are:

$$\begin{array}{lll} \mathsf{A}_3[\![\phi]\!] = \varphi & & \\ \quad \bigwedge_{x,y \in \mathsf{Vars}(\phi)} & C_{xy} \in \sigma(C_{xy}) & \\ \quad \bigwedge_{x \in \mathsf{Vars}(\phi)} & \mathit{Arity}_x \in \sigma(\mathit{Arity}_x) & \\ \qquad\qquad \wedge & \mathit{Label}_x \in \sigma(\mathit{Label}_x) & \\ \qquad\qquad \bigwedge_{1 \leq i \leq \mathrm{MAX}} & \mathit{In}^i_x \subseteq \mathit{Child}^i_x \wedge \mathit{Out}^i_x \parallel \mathit{Child}^i_x & \end{array}$$

5.4 Distribution Rules

There are three types of choices that may have to be made when propagation alone is not sufficient to resolve them automatically: what branch of a disjunction $\phi_1 \vee \phi_2$ to select, in what dominance relationship two variables x and y stand, and whether y is the ith child of x. Thus we need three distribution rule schemas. The first one allows us to pick a non-determined choice variable and make a case distinction:

$$C_{xy} \in D_1 \uplus D_2 \quad \rightarrow \quad C_{xy} \in D_1 \ \vee \ C_{xy} \in D_2 \qquad D_1, D_2 \neq \emptyset$$

The second one allows us to pick a non-determined selector I, introduced when translating $\phi_1 \vee \phi_2$ into $\langle\sigma_1, \sigma_2\rangle[I]$, and try each alternative:

$$I \in \{1, 2\} \quad \rightarrow \quad I{=}1 \ \vee \ I{=}2$$

The third one is an instance of (δ_2) and allows us to pick a set variable $Genesis_y$ and an integer $\iota_{x[i]} \in \mathcal{G}$ to decide whether or not $x[i] = y$ holds:

$$\iota_{x[i]} \in Genesis_y \ \vee \ \iota_{x[i]} \notin Genesis_y$$

Completeness. For every solution $(\mathcal{M}^\tau, \alpha)$ of ϕ there is consistent solved form of $[\![\phi]\!]$. We do not provide formal proof, however it should be clear that the constraints of our encoding are valid in all tree structures, and that, given a solution $(\mathcal{M}^\tau, \alpha)$ of ϕ, we can construct a solved form of $[\![\phi]\!]$ simply by reading off the value for each variable.

Soundness. For every consistent solved form of $[\![\phi]\!]$ there exists a corresponding solution $(\mathcal{M}^\tau, \alpha)$ of ϕ. In other words, propagation is strong enough to derive a contradiction whenever a description is not satisfiable. To prove this, we should exhibit a procedure that constructs a solution $(\mathcal{M}^\tau, \alpha)$ of ϕ from a consistent solved form $[\![\phi]\!]$. This result has not been established yet, but we are working on an approach in the style of Duchier and Niehren (2000).

6 Conclusion

In this paper, we addressed the following open question: is it possible to give a constraint-based treatment to the class of tree descriptions admitting all Boolean connectives? Is it possible to devise a solver for this class that may still perform effective model elimination through constraint propagation? We have shown that this is indeed the case.

We presented a new formalism for tree descriptions that combines the set operators formalism of Duchier and Niehren (2000) with the feature tree logic of Smolka and Treinen (1994), and extends them by additionally admitting disjunction and negation.

Then we showed that every tree description in our formalism can be transformed into a CSP expressible as a constraint program, and thus solvable by contraint programming. We gave precise propagation and distribution rules that we assume of a practical implementation. In particular, we gave precise operational semantics to disjunctive propagators and selection propagators, thus providing secure foundation for an important technical contribution: our treatment of disjunction by reduction to the selection constraint.

Future work. We are currently working on formal proofs of completeness and soundness. We also plan to develop a concrete implementation in Oz and perform an experimental evaluation of the technique presented here.

Acknowledgments. We are greatful to Joachim Niehren. The present paper borrows and greatly benefits from previous joint work with him.

References

Backofen, R., Rogers, J., and 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., and 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., and Berthier, F. (1988). The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88*, pages 693–702, Tokyo, Japan.

Duchier, D. (1999a). Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on Mathematics of Language*, pages 115–126, Orlando, Florida.

Duchier, D. (1999b). Set constraints in computational linguistics – solving tree descriptions. In *Workshop on Declarative Programming with Sets (DPS'99)*, pages 91–98.

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

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

Duchier, D. and 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., and Xu, F. (1998). Constraints over lambda-structures in semantic underspecification. In *Joint Conf. COLING/ACL*, pages 353–359.

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

Koller, A., Niehren, J., and Treinen, R. (2000). 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., and Bennett, P., editors, *Linguistic Theory and Computer Applications*. Academic Press.

Marcus, M. P., Hindle, D., and Fleck, M. M. (1983). D-theory: Talking about talking about trees. In *21st ACL*, pages 129–136.

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

Müller, T. and Müller, M. (1997). Finite set constraints in Oz. In Bry, F., Freitag, B., and Seipel, D., editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München.

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

Rambow, O., Vijay-Shanker, K., and Weir, D. (1995). D-tree grammars. In *Proceedings of ACL'95*, pages 151–158, MIT, Cambridge.

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

Smolka, G. (1995). The Oz Programming Model. In van Leeuwen, J., editor, *Computer Science Today*, pages 324–343. Springer-Verlag, Berlin.

Smolka, G. and Treinen, R. (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.