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; Ram-}

bow et al., 1995; Duchier and Thater, 1999), for underspecified representation of scope ambiguities in semantics (Musken, 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 \( x \preceq^* y \) and \( x : f(x_1 \ldots x_n) \) where variables denote nodes in the tree. The symbol \( \preceq^* \) 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 \( \text{NP} \)-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 \( \preceq^* \) 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 Trénien, 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 provides 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 Trénien, 1994).

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

\[
\phi := x \triangleleft^* y \mid x : f(x_1 \ldots x_n) \mid \phi \wedge \phi'
\]

The symbol \( \triangleleft^* \) 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 descriptions are given by interpretation over finite tree structures. A model \( (M^r, \alpha) \) of a description \( \phi \) consists of a tree structure \( M^r \) and a variable assignment (or interpretation) \( \alpha \) that maps each variable in \( \phi \) to a node in \( M^r \), and such that \( (M^r, \alpha) \models \phi \) in the usual Tarskiian sense. We elaborate further in section 3.

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

\[
\phi := x R y \mid x : f(x_1 \ldots x_n) \mid \phi \wedge \phi'
\]

where \( R \subseteq \{=, \triangleleft^+, \triangleright^+, \bot, \}\)

Two nodes are either equal (=), one is a proper ancestor of the other (\( \triangleleft^+ \)), or the other way around (\( \triangleright^+ \)), 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, \( (M^r, \alpha) \models x R y \) iff \( (M^r, \alpha) \models x r y \) 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:

\[
S := R \mid = \mid \triangleleft^+ \mid \triangleright^+ \mid \bot \mid \neg S \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S^{-1}
\]

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 \( x S_1 y \lor x S_2 y \). 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. \(- (x \, R \, y)\) is equivalent to \(x \, \neg R \, y\) and thus already handled by complementation of \(R\).

2. \(- (\phi_1 \land \phi_2)\) is equivalent to \(\neg \phi_1 \lor \neg \phi_2\), thus will be handled by our treatment of disjunction.

3. Only \(- (x : f(x_1 \ldots x_n))\) is problematic: it is satisfied when either \(x\) is not labeled by \(f\), or does not have arity \(n\), or its \(i\)th 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):

\[
x : f(x_1 \ldots x_n) \equiv |x| = n \land x : f \land \bigwedge_{1 \leq i \leq n} x[i] = x_i
\]

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

\[
- (x : f(x_1 \ldots x_n)) \equiv |x| \neq n \lor - (x : f) \lor \bigvee_{1 \leq i \leq n} x[i] \neq x_i
\]

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:

\[
- (x : f(x_1 \ldots x_n)) \equiv \bigvee_{g \neq f \in \Sigma} x : g(x'_1 \ldots x'_{ar(y)}) \land (x : f(x''_1 \ldots x''_n) \land \bigwedge_{1 \leq i \leq n} x''_i \neq x_i)
\]

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:

\[
\phi ::= x \, R \, y \mid |x| = n \mid x : f \mid x[i] = y \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \neg \phi
\]

where \(R \subseteq \{=, \lt^+, \gt^+, \bot\}\)

The admission of disjunction and negation makes the set operator extension of (Duchier and Niehren, 2000) obsolete since \(x \, R_1 \cup R_2 \, y \equiv x \, R_1 \, y \lor x \, R_2 \, y\) 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 \(\Sigma\) of function symbols written \(f, g, a, b, \ldots\) and equipped with an arity \(ar(f) \geq 0\). We assume that \(\Sigma\) contains at least one constant and one function symbol of arity \(\geq 2\).

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 \(\pi\) is a word (i.e. a sequence) of positive integers. We write \(\epsilon\) for the empty path and \(\pi_1 \pi_2\) for the concatenation of \(\pi_1\) and \(\pi_2\). \(\pi\) is a prefix of \(\pi\) iff there exists \(\pi''\) such that \(\pi = \pi' \pi''\). We write \(\pi_1 \lt^+ \pi_2\) when \(\pi_1\) is a prefix of \(\pi_2\) and say that \(\pi_1\) dominates \(\pi_2\). 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
\[ B ::= \text{true} \mid \text{false} \mid X_1 = X_2 \mid I \in D \mid i \in S \mid i \notin S \mid B_1 \land B_2 \quad (D \subseteq \Delta) \]

\[ \mathcal{C} ::= B \mid I_1 \leq I_2 \mid S_1 \parallel S_2 \mid S_3 \subseteq S_1 \cup S_2 \mid \mathcal{C}_1 \land \mathcal{C}_2 \mid \mathcal{C}_1 \lor \mathcal{C}_2 \mid I = \langle I_1 \ldots I_n \rangle[J] \mid S = \langle S_1 \ldots S_n \rangle[J] \]

Figure 1: Constraint Language

equality
\[ X_1 = X_2 \land B[X_j] \rightarrow B[X_k] \quad \{j, k\} = \{1, 2\} \]

finite domain integer constraints
\[ I \in D_1 \land I \in D_2 \rightarrow I \in D_1 \cap D_2 \]
\[ I \in \emptyset \rightarrow \text{false} \]
\[ I_1 \leq I_2 \land I_1 \in \{n \ldots m\} \rightarrow I_2 \in \Delta \setminus \{1 \ldots n - 1\} \]
\[ I_1 \leq I_2 \land I_2 \in \{n \ldots m\} \rightarrow I_1 \in \Delta \setminus \{m + 1 \ldots \mu\} \]

finite set constraints
\[ i \in S \land i \notin \mathcal{S} \rightarrow \text{false} \]
\[ S_1 \parallel S_2 \land i \in S_j \rightarrow i \notin S_k \quad \{j, k\} = \{1, 2\} \]
\[ S_3 \subseteq S_1 \cup S_2 \land i \notin S_1 \land i \notin S_2 \rightarrow i \notin S_3 \]
\[ S_3 \subseteq S_1 \cup S_2 \land i \in S_3 \land i \notin S_j \rightarrow i \in S_k \quad \{j, k\} = \{1, 2\} \]

Figure 2: Main Propagation Rules

all \( \pi \in D \) and \( k \geq 1 \) satisfy \( \pi k \in D \) iff \( k \leq \text{ar}(L(\pi)) \), 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 \( \tau \) is a tree, we write \( D_\tau \) for its domain and \( L_\tau \) for its labeling function.

Tree structure and set operators. A tree structure is a first-order structure \( \mathcal{M}_\tau \) representing a tree \( \tau \) by the relations between its nodes. The domain of \( \mathcal{M}_\tau \) is the tree domain \( D_\tau \). For each function symbol \( f \in \Sigma \), \( \mathcal{M}_\tau \) contains a relation \( :f \) of arity \( \text{ar}(f) + 1 \) such that \( t \) (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 \( 1 \leq i \leq n = \text{ar}(f) \).

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

Semantics of tree descriptions. Tree descriptions are interpreted in the class of tree structure over \( \Sigma \). 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 \( \tau \). We write \( \text{Vars}(\phi) \) for the set of variables occurring in \( \phi \). A solution of a description \( \phi \) is a pair \( (\mathcal{M}_\tau^\phi, \alpha) \) of a tree structure \( \mathcal{M}_\tau^\phi \) and a variable assignment \( \alpha : \text{Vars}(\phi) \rightarrow D_\tau \). We write \( (\mathcal{M}_\tau^\phi, \alpha) \models \phi \) if \( \phi \) is satisfied by \( (\mathcal{M}_\tau^\phi, \alpha) \) in the usual Tarskian way, where the CFT-style descriptions are interpreted as follows:

\[
(\mathcal{M}_\tau^\phi, \alpha) \models \{x\} = n \quad \equiv \quad \text{ar}(L_\tau(\alpha(x))) = n
\]
\[
(\mathcal{M}_\tau^\phi, \alpha) \models x : f \quad \equiv \quad L_\tau(\alpha(x)) = f
\]
\[
(\mathcal{M}_\tau^\phi, \alpha) \models x[i] = y \quad \equiv \quad \alpha(y) = \alpha(x)^i
\]

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 \( \phi \) into a constraint \( [\phi] \) of a constraint language. Solutions of \([\phi]\) can then be enumerated efficiently by alternating steps of propagation.
finite domain selection constraint
\[ I = \{I_1 \ldots I_n\}[J] \quad \rightarrow \quad J \in \{1..n\} \]
\[ I = \{I_1 \ldots I_n\}[J] \quad \land \quad J \in D \quad \land \quad \forall j \in D \quad I_j \in D_j \quad \rightarrow \quad I \in \cup\{D_j \mid j \in D\} \]
\[ I = \{I_1 \ldots I_n\}[J] \quad \land \quad j \in D \quad \land \quad I_j \in D_j \quad \land \quad D \cap D_j = \emptyset \quad \rightarrow \quad j \notin J \]
\[ I = \{I_1 \ldots I_n\}[J] \quad \land \quad j = J \quad \rightarrow \quad I = I_j \]

finite set selection constraint
\[ S = \{S_1 \ldots S_n\}[I] \quad \rightarrow \quad I \in \{1..n\} \]
\[ S = \{S_1 \ldots S_n\}[I] \quad \land \quad I \in D \quad \land \quad \forall i \in D \quad j \in S_i \quad \rightarrow \quad j \notin S \]
\[ S = \{S_1 \ldots S_n\}[I] \quad \land \quad j \notin S \quad \land \quad j \notin S_i \quad \rightarrow \quad i \neq I \]
\[ S = \{S_1 \ldots S_n\}[I] \quad \land \quad i = I \quad \rightarrow \quad S = S_i \]

Figure 3: Disjunctive 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 \( \mu \) such as 134217726. We assume a set of integer variables with values in \( \Delta \) and ranged over by \( I \), \( J \), or \( K \), and a set of set variables with values in \( 2^\Delta \) 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 \( \Delta \).

Basic and non-basic constraints. We distinguish between basic constraints \( B \) which can be represented directly in the constraint store and non-basic constraints \( C \) 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:

\[ S_1 \parallel S_2 \land i \in S_j \quad \rightarrow \quad i \notin S_k \quad \text{for} \{j,k\} = \{1,2\} \]

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 \( \land \{i \in S \mid i \in D\} \land \{i \notin S \mid i \in \Delta \setminus D\} \), \( S_1 \subseteq S_2 \) for \( S_1 \subseteq S_2 \cup S_3 \), \( S_1 \cup S_2 \neq \emptyset \), and \( S = S_1 \cup S_2 \) for \( S_1 \parallel S_2 \land S \subseteq S_1 \cup S_2 \land S_1 \subseteq S \land S_2 \subseteq S \) (i.e. \( \cup \) 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 \( C \), 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 \cup D_2 \quad \rightarrow \quad I \in D_1 \lor I \in D_2 \quad \text{for} \ D_1 \cup D_2 \neq \emptyset \quad \text{(\( \delta_1 \))} \]
\[ S \subseteq D \quad \rightarrow \quad i \in S \lor i \notin S \quad \text{for} \ i \in D \quad \text{(\( \delta_2 \))} \]
The disjunction in a distribution rule is interpreted non-deterministically. Thus, rule $(\delta_1)$ non-deterministically infers either $I \in D_1$ or $I \in D_2$. A constraint program consists of a constraint $C$ 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 $D_1 \not\equiv D_2$ 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 ($C$ or $C'$) infers $C$ if it can be shown that $C$ is inconsistent with the basic constraints derived so far. The precise semantics of the disjunctive propagator are given in (Fig 3), where we write $B \land C \rightarrow^* \text{false}$ to mean that false is in the saturation of $B \land C$ 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-access. The declarative semantics are simply $X = X_I$. Such a constraint might be implemented with an $n$-ary disjunctive propagator:

$$(X = X_1 \land I = 1) \text{ or } \ldots \text{ or } (X = X_n \land I = n)$$

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:

$$(X = k_1 \land I = 1) \quad \text{or} \quad (X = k_2 \land I = 2) \quad \text{or} \quad (X = k_3 \land I = 3)$$

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

$$[\phi] = \bigwedge_{x \in \text{Var}(\phi)} A_1(x) \land A_2(x, y) \land A_3[\phi]$$

$A_1(x)$ introduces a node representation for each variable $x$ in $\phi$, $A_2(x, y)$ axiomatizes the well-formedness (i.e. treelessness) of the relations between these nodes, and $A_3[\phi]$ encodes the specific restrictions imposed by $\phi$. 


In a solution \((\mathcal{M}', \alpha')\) of \(\phi\), every variable \(x\) is mapped to a node \(\alpha(x)\) of \(\mathcal{M}'\). Thus from the point of view of \(x\), the set \(\text{Vars}(\phi)\) is partitioned into 4 disjoint subsets: all variables equal to \(x\) (i.e., mapped to the same node \(\alpha(x)\)), all ancestors, all descendants, 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 \(\Sigma\). Let \(\text{MAX}\) be the maximum constructor arity in \(\Sigma\). For each formal variable \(x\) in \(\phi\) we choose a distinct integer \(\iota_x\) to represent it and introduce \(7 + 2 \times \text{MAX}\) constraint variables written \(\text{Eq}_x, \text{Up}_x, \text{Down}_x, \text{Side}_x, \text{Eq}_{up_x}, \text{Eq}_{down_x}, \text{Child}_x^i\) and \(\text{Down}_x^i\) for \(1 \leq i \leq \text{MAX}\), and \(\text{Genesix}_x\), as well as 3 integer constraint variables \(\text{Label}_x, \text{Arity}_x, \text{A}_{i_j}(x)\). First, we state \(x = x\):

\[
\iota_x \in \text{Eq}_x
\]

\(\text{Eq}_x, \text{Up}_x, \text{Down}_x, \text{Side}_x\) encode the set of variables that are respectively equal, above, below, and to the side (i.e. disjoint) of \(x\). Thus posing \(I = \{\iota_x | x \in \text{Vars}(\phi)\}\) for the set of integers encoding \(\text{Vars}(\phi)\), we have:

\[I = \text{Eq}_x \uplus \text{Up}_x \uplus \text{Down}_x \uplus \text{Side}_x\]

As described in (Duchier and Niehren, 2000), we can and should improve propagation by introducing \(\text{Eq}_{down_x}\) and \(\text{Eq}_{up_x}\) as intermediate results:

\[I = \text{Eq}_{down_x} \uplus \text{Up}_x \uplus \text{Side}_x\]
\[I = \text{Eq}_{up_x} \uplus \text{Down}_x \uplus \text{Side}_x\]

\[\text{Eq}_{down_x} = \text{Eq}_x \uplus \text{Down}_x\]
\[\text{Eq}_{up_x} = \text{Eq}_x \uplus \text{Up}_x\]

\(\text{Down}_x^i\) encodes the set of variables in the subtree rooted at \(x\)'s \(i\)th child (empty if there is no such child):

\[\text{Down}_x = \bigcup_{1 \leq i \leq \text{MAX}} \text{Down}_x^i\]

\(\text{Child}_x^i\) encodes the set of variables occurring as \(x\)'s \(i\)th child:

\[\text{Child}_x^i \subseteq \text{Down}_x^i\]

We choose a bijection \(\ell : \Sigma \rightarrow \{1, \ldots, |\Sigma|\}\) 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 \(\text{Label}_x = \ell(f)\).

\[\text{Label}_x \in \{1, \ldots, |\Sigma|\}\]

The arity of a node depends on its constructor. Thus, \(\text{Arity}_x\) and \(\text{Label}_x\) are related by the following selection constraint:

\[\text{Arity}_x = \langle \text{ar}(\ell^{-1}(1)), \ldots, \text{ar}(\ell^{-1}(|\Sigma|)) \rangle[\text{Label}_x]\]

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

\[\text{Arity}_x < i \Rightarrow |\text{Down}_x^i| = 0\]

Where we define:

\[I < i \Rightarrow C \equiv (I < i \land C) \lor (I \geq i)\]

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

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

\[\text{A}_1(x) \equiv (1) \land \ldots \land (10)\]
\( C_{xy} \in \{1, 2, 3, 4\} \)  \hfill (11)

\( B[x = y] \land C_{xy} = 1 \) or \( C_{xy} \neq 1 \land B[x \neq y] \)  \hfill (12)

\( B[x <^+ y] \land C_{xy} = 2 \) or \( C_{xy} \neq 2 \land B[x <^+ y] \)  \hfill (13)

\( B[x >^+ y] \land C_{xy} = 3 \) or \( C_{xy} \neq 3 \land B[x >^+ y] \)  \hfill (14)

\( B[x \perp y] \land C_{xy} = 4 \) or \( C_{xy} \neq 4 \land B[x \perp y] \)  \hfill (15)

Figure 5: Well-formedness clauses

\[
\begin{align*}
B[x = y] &= Eq_x \land Eq_y \land Up_x = Up_y \land Down_x = Down_y \land Side_x = Side_y \land \\
Equp_x &= Equp_y \land Eqdown_x = Eqdown_y \land \\
Label_x &= Label_y \land Arity_x = Arity_y \land Genesis_x = Genesis_y \land \\
Down^l_x &= Down^l_y \land Child^i_x = Child^i_y
\end{align*}
\]

\( B[x \neq y] = Eq_x \lor Eq_y \)

\( B[x <^+ y] = Eqdown_y \subseteq Down_y \land Equp_x \subseteq Up_y \land Side_x \subseteq Side_y \)

\( B[x \neq^+ y] = Eq_x \lor Eq_y \land Down_x \subseteq Side_y \land Eqdown_y \subseteq Side_x \)

\( B[x \perp y] = Eq_x \lor Eq_y \land Side_y \subseteq Side_x \)

Figure 6: Set constraints characteristic of each case

\[
\begin{align*}
Genesis_x &\in \mathcal{G} \quad (16) \\
\tau_{x[i]} &\in Genesis_y \land Child^i_x = Eq_y \land Down^l_x = Eqdown_y \land Up_y = Equp_x
\end{align*}
\]

\( \text{or} \quad \tau_{x[i]} \notin Genesis_y \land Child^i_x \land Eq_y \)  \hfill (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 \( =, <^+, >^+ \) 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 \( B[x \ y] \) are the set constraints characteristic of the case \( x \ y \) 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 \( x \in \text{Var}(\phi) \) and \( 1 \leq i \leq \text{max} \) we choose a distinct integer \( \tau_{x[i]} \) to encode it. We define \( \mathcal{G} = \{ \tau_{x[i]} \mid x \in \text{Var}(\phi), 1 \leq i \leq \text{max} \} \), and contribute the clauses of Fig 7.

\[
A_2(x, y) \equiv (11) \land (12) \land (13) \land (14) \land (15) \land (16) \land \quad 1 \leq i \leq \text{max} \)
\]

5.3 PROBLEM SPECIFIC CONSTRAINTS

The last part \( A_2(\phi) \) 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 \( \phi \).

We will make use of the notion of a restriction \( \sigma \) 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.
\[ C[\phi_1 \lor \phi_2]_I^I = (\phi_1 \land \phi_2 \land I \in \{1, 2\} \land J \rho I, (\sigma_1, \sigma_2)[I]) \]

where \( I \) is a fresh variable, \( (\phi_1, \sigma_1) = C[\phi_1]_I^I \) and \( (\phi_2, \sigma_2) = C[\phi_2]_I^I \)

\[ C[\phi_1 \land \phi_2]_I^I = (\phi_1 \land \phi_2, \sigma_1 \& \sigma_2) \]

where \( (\phi_1, \sigma_1) = C[\phi_1]_I^I \) and \( (\phi_2, \sigma_2) = C[\phi_2]_I^I \) and \( \sigma_1 \& \sigma_2 \) is defined in Fig 9

\[ C[-\phi] = C[\phi] \]

\[ C[-(\phi_1 \lor \phi_2)] = C[-\phi_1 \lor -\phi_2] \]

\[ C[-(\phi_1 \land \phi_2)] = C[-\phi_1 \land -\phi_2] \]

\[ C[x R y] = (\text{true}, T \{C_{xy} \implies \{D[r] \mid r \in R\}\}) \]

\[ D[0] = 1 \]
\[ D[1] = 2 \]
\[ D[2] = 3 \]
\[ D[3] = 4 \]

\[ C[-(x = n)] = (\text{true}, T \{\text{Arity}_y \implies \{n\}\}) \]

\[ C[-(x : f)] = (\text{true}, T \{\text{Label}_x \implies \{f\}\}) \]

\[ C[-(x[i] = y)] = (\text{true}, T \{\text{Child}_x \implies \{y, \emptyset\}\}) \]

\[ C[-(x[i] = y)] = (\text{true}, T \{\text{Child}_x \implies \{y, \emptyset\}\}) \]

\[ \langle \sigma_1 \& \sigma_2 \rangle (C_{xy}) = \sigma_1 (C_{xy}) \cap \sigma_2 (C_{xy}) \]

\[ \langle \sigma_1 \& \sigma_2 \rangle (\text{Arity}_x) = \sigma_1 (\text{Arity}_x) \cap \sigma_2 (\text{Arity}_x) \]

\[ \langle \sigma_1 \& \sigma_2 \rangle (\text{Label}_x) = \sigma_1 (\text{Label}_x) \cap \sigma_2 (\text{Label}_x) \]

\[ \langle \sigma_1 \& \sigma_2 \rangle (\text{Child}_x) = \{\text{In}_1 \cup \text{Out}_1 \cup \text{Out}_2\} \]

where \( \langle \text{In}_1, \text{Out}_1 \rangle = \sigma_1 (\text{Child}_x) \)

\[ \langle \text{In}_2, \text{Out}_2 \rangle = \sigma_2 (\text{Child}_x) \]

**Figure 8:** Computing restrictions

\[ x <^* y \quad \sigma(C_{xy}) = \{1, 2\} \]

\[ x : f \quad \sigma(\text{Label}_x) = \{f\} \]

\[ |x| = n \quad \sigma(\text{Arity}_x) = \{n\} \]

**Figure 9:** Accumulating Restrictions

Our intention is to translate a description \( \phi \) into a corresponding restriction \( \sigma \) such that the problem specific constraints are all of the form:

\[ C_{xy} \in \sigma(C_{xy}) \quad (18) \]

\[ \text{Label}_x \in \sigma(\text{Label}_x) \quad (19) \]

\[ \text{Arity}_x \in \sigma(\text{Arity}_x) \quad (20) \]

Additionally, literals such as \( x[i] = y \) and \( -x[i] = y \) give rise to restrictions on variables \( \text{Child}_x \).

Such a restriction is expressed as a pair \( (\text{In}_1, \text{Out}_1) \) where \( \text{In}_1 \) represents the set of variables that must be included in \( \text{Child}_x \) and \( \text{Out}_x \) the set of those that must be excluded from it. It yields the problem specific constraint:

\[ \text{In}_1 \subseteq \text{Child}_x \land \text{Out}_x \parallel \text{Child}_x \quad (21) \]

Disjunctions are handled by means of the selection constraint. If descriptions \( \phi_1 \) and \( \phi_2 \) translate respectively to restrictions \( \sigma_1 \) and \( \sigma_2 \), then \( \phi_1 \lor \phi_2 \) 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:

\[ (\sigma_1, \sigma_2)[I](X) = (\sigma_1(X), \sigma_2(X))[I] \]
If approached na"ively, nested disjunctions result in spurious indeterminacy. Consider:

\((\phi_1 \lor \phi_2) \lor (\phi'_1 \lor \phi'_2)\)

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 even though this choice has become irrelevant. In order to remove such spurious indeterminacy, we add the following constraints:

\[ J \leq I \land J \geq I' \]

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 \(\phi\) into a restriction. For this purpose, we use \(C[\phi, \rho]\) where \(\rho\) is either \(\leq\) or \(\geq\). \(J\) and \(\rho\) are used to remove spurious indeterminacies as outlined above. \(C[\phi, \rho]\) actually returns a pair:

\[ \langle \varphi, \sigma \rangle = C[\phi, \rho] \]

where \(\varphi\) is a constraint and \(\sigma\) is a restriction. We write \(T\) for the most general restriction, which is defined point-wise as follows:

\[
\begin{align*}
T(C_{xy}) & = \{1, 2, 3, 4\} \\
T(Arity_x) & = \{0 \ldots \text{MAX}\} \\
T(Label_x) & = \{1 \ldots |\Sigma|\} \\
T(\text{Child}_x^i) & = \{\emptyset, \emptyset\}
\end{align*}
\]

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 \(C[\phi, \rho]\) in Fig. 8. If we pose \(\langle \varphi, \sigma \rangle = C[\phi, \rho] = \langle \text{In}_x^i, \text{Out}_x^i \rangle = \sigma(\text{Child}_x^i)\), the problem specific constraints are:

\[
\begin{align*}
A_1[\phi] = & \quad \varphi \\
& \land_{x,y \in \text{Var}(\phi)} C_{xy} \in \sigma(C_{xy}) \\
& \land_{x \in \text{Var}(\phi)} \land_{1 \leq i \leq \text{MAX}} \text{Arity}_x \in \sigma(\text{Arity}_x) \\
& \land_{x \in \text{Var}(\phi)} \land_{1 \leq i \leq \text{MAX}} \text{Label}_x \in \sigma(\text{Label}_x) \\
& \text{In}_x^i \subseteq \text{Child}_x^i \land \text{Out}_x^i \parallel \text{Child}_x^i
\end{align*}
\]

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 \(\phi_1 \lor \phi_2\) to select, in what dominance relationship two variables \(x, y\) stand, and whether \(y\) is the \(i\)th 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:

\[ C_{xy} \in D_1 \equiv D_2 \rightarrow C_{xy} \in D_1 \lor C_{xy} \in D_2 \quad D_1, D_2 \neq \emptyset \]

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

\[ I \in \{1, 2\} \rightarrow I=1 \lor I=2 \]
The third one is an instance of (δy) and allows us to pick a set variable $\text{Genesis}_y$ and an integer $\delta_{x[i]} \in \mathcal{G}$ to decide whether or not $x[i] = y$ holds:

$$\delta_{x[i]} \in \text{Genesis}_y \lor \delta_{x[i]} \not\in \text{Genesis}_y$$

**Completeness.** For every solution $(\mathcal{M}^*, \alpha)$ of $\phi$ 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 $(\mathcal{M}^*, \alpha)$ of $\phi$, 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 $(\mathcal{M}^*, \alpha)$ of $\phi$. In other words, propagation is strong enough to derive a contradiction whenever a description is not satisfactory. To prove this, we should exhibit a procedure that constructs a solution $(\mathcal{M}^*, \alpha)$ of $\phi$ 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 constraint 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 grateful to Joachim Niehren. The present paper borrows and greatly benefits from previous joint work with him.

**References**


