The First-Order Theory of Ordering Constraints over Feature Trees

Martin Müller and Joachim Niehren
Programming Systems Lab
Universität des Saarlandes
Saarbrücken, Germany
www.ps.uni-sb.de/~{mmueller,niehren}

Ralf Treinen
Laboratoire de Recherche en Informatique
Université de Paris-Sud
Orsay, France
www.lri.fr/~treinen

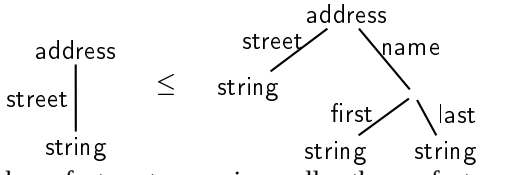
Abstract

The system $FT_{\leq}$ of ordering constraints over feature trees has been introduced as an extension of the system FT of equality constraints over feature trees. We investigate the first-order theory of $FT_{\leq}$ and its fragments, both over finite trees and over possibly infinite trees. We prove that the first-order theory of $FT_{\leq}$ is undecidable, in contrast to the first-order theory of FT which is well-known to be decidable. We determine the complexity of the entailment problem of $FT_{\leq}$ with existential quantification to be PSPACE-complete, by proving its equivalence to the inclusion problem of non-deterministic finite automata. Our reduction from the entailment problem to the inclusion problem is based on a new alogrithm that, given an existential formula of $FT_{\leq}$, computes a finite automaton which accepts all its logic consequences.

1 Introduction

Feature constraints have been used for describing records in constraint programming [28, 33] and record-like structures in computational linguistics [11, 21, 24]. Feature constraints also occur naturally in type inference for programming languages with object types or record types [4, 20, 22].

Following [1–3], we consider feature constraints as predicate logic formulas interpreted in the structure of feature trees. A feature tree is a tree with unordered edges labeled by features and with possibly labeled nodes. Features are functional in that the features labeling the edges departing from the same node must be pairwise different. The structure of feature trees gives rise to an ordering in a very natural way which is called *weak subsumption ordering* in [6]. Consider for example:



Intuitively, a feature tree τ_1 is smaller than a feature tree τ_2 if τ_1 has fewer edges and node labels than τ_2. More precisely, this means that every word of features in the tree domain of τ_1 belongs to the tree domain of τ_2, and that the (partial) labeling function of τ_1 is contained in the labeling function of τ_2. In this case we write $\tau_1 \leq \tau_2$.

We consider the system $FT_{\leq}$ of ordering constraints over feature trees [15, 17, 18]. Its constraints φ are given by the following abstract syntax:

$$\varphi ::= x{\leq}x' \mid x[f]x' \mid a(x) \mid \varphi \wedge \varphi'$$

The constraints of $FT_{\leq}$ are interpreted in the structure of feature trees with the weak subsumption ordering. We distinguish two cases, the structure of finite feature trees and the structure of possibly infinite feature trees. A constraint $x{\leq}x'$ holds if the denotation of x weakly subsumes the denotation of x', $x[f]x'$ is valid if the denotation of x has an edge at the root that is labeled with the feature f and leads to the denotation of x', and $a(x)$ means that the root of x is labeled with a.

The constraint system $FT_{\leq}$ is an extension of the well-investigated constraint system FT, which provides for equality constraints $x{=}y$ rather than more general ordering constraints $x{\leq}y$. The full first-order theory of FT is decidable [3] and has non-elementary complexity [34]. The decidability question for the first-order theory of $FT_{\leq}$ has been raised in [17]. There, two indications in favour of decidability have been formulated: its analogy to FT and its relationship to second-order monadic logic (we will discuss this below). In contrast, we show in this paper that the *the first-order theory of $FT_{\leq}$ is undecidable*. Our result holds in the structure of possibly infinite feature trees and, more surprisingly, even in the structure of finite feature trees. Our proof is based on an encoding the Post Correspondence Problem using a technique of [30].

Once the undecidability of the first-order theory of $FT_{\leq}$ is settled, it remains to distinguish decidable fragments and their complexity. It is well-known that the satisfiability problem of FT, its entailment problem $\varphi \models \varphi'$, and its entailment problem with existential quantifiers $\varphi \models \exists x_1 \ldots \exists x_n \varphi'$ can be solved in quasi-linear time. The investigation of ordering constraints was initiated

by Dörre [6] who gave an $O(n^5)$-algorithm for deciding satisfiability of $FT_{\leq}$-constraints. This result was improved to $O(n^3)$ in [18], where also the entailment problem of $FT_{\leq}$ concerning *quantifier-free* judgements $\varphi \models \varphi'$ was shown decidable in cubic time. The next step towards larger fragments of the theory of $FT_{\leq}$ was to consider entailment judgments with existential quantification $\varphi \models \exists x_1 \ldots \exists x_n \varphi'$ which are equivalent to unsatisfiability judgments $\varphi \wedge \neg \exists x_1 \ldots \exists x_n \varphi'$ with quantification below negation. As shown in [17], this problem is decidable, coNP-hard in case of finite trees, and PSPACE-hard in case of arbitrary trees. Decidability is proved by reduction to the entailment problem with existential quantifiers in the related structure of so-called *sufficiently labeled* feature trees. Since the full first-order theory of ordering constraints that over sufficiently labeled (finite) feature trees can easily be encoded in (weak) second order monadic logic, decidability of entailment follows from the classical results on (W)S2S [23, 29].

This paper contributes the exact complexity of the entailment problem of $FT_{\leq}$ with existential quantification. We prove PSPACE-completeness, both in structures of finite trees and of possibly infinite trees. This result is obtained by reducing the entailment problem of $FT_{\leq}$ with existential quantifiers to the inclusion problem of nondeterministic finite automata (NFA), and vice versa. Our reduction of entailment is based on the following idea: Given an existential formula $\exists \bar{x} \varphi$ we construct an automaton that accepts all its consequences in form of so called path constraints. The inverse reduction has already been presented in [17] in the case of possibly infinite trees. Surprisingly, we can adapt this reduction to the structure of finite trees by inverting all ordering constraints used there.

Applications and Related Work. The application domains of ordering constraints over feature trees are quite diverse. They have been used to describe so-called coordination phenomena in natural language [6] but also for the analysis of concurrent constraint programming languages [16]. The less general equality constraints over feature trees are central to constraint based grammars, and they provide record constraints for logic programming [28] or concurrent constraint programming [13, 25]. In concurrent constraint programming, entailment with existential quantification is needed for deciding the satisfaction of conditional guards. As mentioned above, our results are also relevant for constraint-based inference of record types and object types. In this context, the entailment test has recently received some attention as a justification for constraint simplification and as a means to check type interfaces [4, 9, 10, 14, 22, 32].

Originally, weak subsumption has been introduced as a weakening of subsumption. The subsumption ordering between feature structures [5, 12, 26] is omnipresent in linguistic theories like HPSG (head-driven phrase structure grammar) [21]. According to the more general view of [6, 27], the subsumption ordering and the weak subsumption ordering are definable between elements of an arbitrary feature algebra (not only between feature structures). This logical perspective enables the definition of subsumption (resp. weak subsumption) constraints [7] which are interpreted with respect the subsumption (resp. weak subsumption) ordering of arbitrary feature algebras. Syntactically, subsumption constraints, weak subsumption constraints, and $FT_{\leq}$ constraints coincide but semantically they differ. As proved in [7], the satisfiability problem of subsumption constraints is undecidable. The satisfiability problem of weak subsumption constraints is equivalent to the satisfiability problem of $FT_{\leq}$ constraints [6, 18] and hence decidable in cubic time.

For the full version containing all proofs see [19].

2 Ordering Constraints

The constraint system $FT_{\leq}$ is defined by a set of constraints, the structure of feature trees, and an interpretation of constraints over feature trees. We assume an infinite set $\mathcal{V}$ of *variables* ranged over by x, y, z, a set $\mathcal{F}$ of at least two *features* ranged over by f, g and a set $\mathcal{L}$ of *labels* ranged over by a, b.

Feature Trees. A *path* π is a word of features. The *empty path* is denoted by ε and the free-monoid concatenation of paths π and π' as $\pi\pi'$. A path π' is called a *prefix of* π if $\pi = \pi'\pi''$ for some path π''. A *tree domain* is a non-empty prefix closed set of paths.

A *feature tree* τ is a pair (D, L) consisting of a tree domain D and a partial function $L : D \rightharpoonup \mathcal{L}$ that we call *labeling function* of τ. Given a feature tree τ, we write D_τ for its tree domain and L_τ for its labeling function. For instance, $\tau_0 = \overset{\cdot}{\underset{a}{|}} f$ $\tau_0 = (\{\varepsilon, f\}, \{(f, a)\})$ is a feature tree with domain $D_{\tau_0} = \{\varepsilon, f\}$ and $L_{\tau_0} = \{(f, a)\}$. A feature tree is *finite* if its tree domain is finite, and *infinite* otherwise. A *node of* τ is an element of D_τ. A node π of τ is *labeled with* a if $(\pi, a) \in L_\tau$. A node of τ is unlabeled if it is not labeled by any a. The *root* of τ is the node ε. The *root label* of τ is $L_\tau(\varepsilon)$, and $f \in \mathcal{F}$ is a *root feature* of τ if $f \in D_\tau$. Given a tree τ with $\pi \in D_\tau$, we write as $\tau[\pi]$ the subtree of τ at path π; formally $D_{\tau[\pi]} = \{\pi' \mid \pi\pi' \in D_\tau\}$ and $L_{\tau[\pi]} = \{(\pi', a) \mid (\pi\pi', a) \in L_\tau\}$.

Syntax and Semantics. An $FT_{\leq}$ *constraint* φ is defined by the abstract syntax

$$\varphi \quad ::= \quad x \leq y \mid a(x) \mid x[f]y \mid \varphi_1 \wedge \varphi_2$$

An $FT_{\leq}$ constraint is a conjunction of *basic constraints* which are either *ordering constraints* $x{\leq}y$, *labeling constraints* $a(x)$, or *selection constraints* $x[f]y$.

We define the structure $FT_{\leq}$ over feature trees in which we interpret $FT_{\leq}$ constraints. Its universe consists of the set of all feature trees. The constraints are interpreted as follows:

$$\begin{array}{rll} \tau_1{\leq}\tau_2 & \text{iff} & D_{\tau_1} \subseteq D_{\tau_2} \text{ and } L_{\tau_1} \subseteq L_{\tau_2} \\ \tau_1[f]\tau_2 & \text{iff} & D_{\tau_2} = \{\pi \mid f\pi \in D_{\tau_1}\} \text{ and} \\ & & L_{\tau_2} = \{(\pi, a) \mid (f\pi, a) \in L_{\tau_1}\} \\ a(\tau) & \text{iff} & (\varepsilon, a) \in L_{\tau} \end{array}$$

The substructure of $FT_{\leq}$ whose universe contains only the finite trees is denoted by $FT_{\leq}^{fin}$.

First-Order Formulas. If not specified otherwise, a formula is said to be valid (satisfiable) if it is valid (satisfiable) both in $FT_{\leq}$ and $FT_{\leq}^{fin}$. Let Φ and Φ' be first-order formulas built from $FT_{\leq}$ constraints with the usual first-order connectives. We say that Φ *entails* Φ', written $\Phi \models \Phi'$, if $\Phi \rightarrow \Phi'$ is valid, and that Φ is *equivalent* to Φ' if $\Phi \leftrightarrow \Phi'$ is valid. We denote with $\mathcal{V}(\Phi)$ the set of variables occurring free in Φ, and with $\mathcal{F}(\Phi)$ and $\mathcal{L}(\Phi)$ the set of features and labels occurring in Φ.

3 Expressiveness of the Theory $FT_{\leq}$

In this section we introduce some abbreviations of formulas needed in Section 4. We use the usual abbreviations for ordering constraints, for instance we write $x{<}y$ for $x{\leq}y \wedge x \neq y$, and $x{\geq}y$ for $y{\leq}x$.

3.1 Minimal and Maximal Values

We can construct, for any formula φ, formulas $\mu x\varphi$ and $\nu x\,\varphi$ expressing that *x is minimal (maximal) with the property* φ:

$$\begin{array}{rcl} \mu x\varphi & := & \varphi \wedge \neg\exists y\,(\varphi[y/x] \wedge y{<}x) \\ \nu x\,\varphi & := & \varphi \wedge \neg\exists y\,(\varphi[y/x] \wedge y{>}x) \end{array}$$

Here, y is a fresh variable not occurring in φ, and $\varphi[y/x]$ denotes the formula where every free occurrence of x is replaced by y. Typically, x occurs free in φ but this is not required. Note that, in contrast to $\forall x$ and $\exists x$, μx and νx are no variable binders that restrict the scope of the variable x; hence x is free in $\mu x\varphi$ and in $\nu x\,\varphi$ if it is free in φ. The formulas $\mu x\varphi$ and $\nu x\,\varphi$ do *not* state that x denotes the smallest (resp. greatest) tree satisfying φ, in fact such a tree may not exist. This difference is important for the formula $\mathsf{atom}(x)$ defined below.

Example 1 *The sentence* $\exists x\,(\mu x\,\mathit{true})$ *is valid in* $FT_{\leq}$ *and in* $FT_{\leq}^{fin}$ *(there even exists a smallest tree, namely* $(\{\varepsilon\}, \{\})$.

Example 2 *The formula* $\mu x\,(x[0]x \wedge x[1]x)$ *is satisfied in* $FT_{\leq}$ *by the full binary, everywhere unlabeled tree, and is not satisfiable in* $FT_{\leq}^{fin}$ *since* $FT_{\leq}^{fin}$ *contains no infinite trees.*

We can now express that x denotes an atom in the lattice-theoretic sense, i.e. that it is a minimal tree strictly greater than the smallest tree $(\{\varepsilon\}, \{\})$, by:

$$\begin{array}{ll} \mathsf{one\text{-}dist}(x,y) & := (x{<}y \wedge \neg\exists z\,(x{<}z{<}y)) \\ \mathsf{atom}(x) & := \exists y\,((\mu y\,\mathit{true}) \wedge \mathsf{one\text{-}dist}(y,x)) \end{array}$$

3.2 Label Restrictions

The formula $x \sim y$ reads *x and y are consistent*:

$$x \sim y := \exists z\,(x{\leq}z \wedge y{\leq}z)$$

For any label $a \in \mathcal{L}$ we write $x \sim a$ to express that the root of x is either unlabeled or labeled with a:

$$x \sim a := \exists y\,(x{\leq}y \wedge a(y))$$

The following formula expresses that the root of a tree is unlabeled:

$$\mathsf{not\text{-}root\text{-}labeled}(x) := x \sim a \wedge x \sim b$$

where a and b are two arbitrary different label symbols. We obtain a first-class status of labels by encoding a label a as the feature tree $(\{\varepsilon\}, \{(\varepsilon, a)\})$.

$$\mathsf{label\text{-}atom}(x) := \mathsf{atom}(x) \wedge \neg\mathsf{not\text{-}root\text{-}labeled}(x)$$

We can now express that x and y either have the same root label or are both unlabeled at the root by

$$\begin{array}{l} \mathsf{same\text{-}root\text{-}label}(x,y) := \\ \qquad \forall z\,(\mathsf{label\text{-}atom}(z) \rightarrow (x \sim z \leftrightarrow y \sim z)) \end{array}$$

3.3 Arity Restrictions

We can simulate a first-class status of feature symbols by encoding a feature f by the tree $(\{\varepsilon, f\}, \emptyset)$.

$$\mathsf{feature\text{-}atom}(x) := \mathsf{atom}(x) \wedge \mathsf{not\text{-}root\text{-}labeled}(x)$$

The following formula expresses that x has exactly the root features $f_1, \ldots, f_n$:

$$\begin{array}{l} x\{f_1, \ldots, f_n\} := \exists x_1, \ldots, x_n \Big(\bigwedge_{i=1}^{n} x[f_i]x_i \wedge \\ \qquad \forall y\,(\bigwedge_{i=1}^{n} y[f_i]x_i \wedge \mathsf{same\text{-}root\text{-}label}(x,y) \rightarrow x{\leq}y)\Big) \end{array}$$

These so-called *arity constraints* have been introduced in [28]. A decidable feature logic where feature and label symbols have first class status has been investigated in [31]. The next formula is crucial for our undecidability proof. A tree τ satisfies this formula iff $\{\varepsilon, \mathsf{c}\} \subseteq D_\tau \subseteq \{\mathsf{c}\}^*$ and all its nodes are unlabeled:

$$\mathsf{string\text{-}c}(x) := x\{\mathsf{c}\} \wedge \mathsf{not\text{-}root\text{-}labeled}(x) \wedge \exists y\,(x[\mathsf{c}]y \wedge y \leq x)$$

In general, we have that

Lemma 3.1 *The formula $\exists y\,(x[f]y \wedge y \leq x)$ is satisfied by τ iff $f \in D_\tau$ and whenever $n \geq m$, $f^n, f^m \in D_\tau$, then*

$$\tau[f^n] \leq \tau[f^m]$$

4 Undecidability Results

Theorem 4.1 *The first-order theories of $FT_\leq^{fin}$ and $FT_\leq$ are undecidable.*

The result holds for arbitrary (even empty) $\mathcal{L}$ and for $\mathcal{F}$ of cardinality ≥ 2, we use however, for the sake of clarity, distinct label symbols a, b and pairwise distinct feature symbols $\mathsf{s}, \mathsf{c}, \mathsf{p}, \mathsf{l}, \mathsf{r}$. We prove Theorem 4.1 by reduction of the Post Correspondence Problem (PCP). See [30] for a discussion of the proof technique employed in this chapter.

An instance of PCP is a finite sequence $P = ((p_i, q_i))_{i=1,\ldots,m}$ of pairs of words from $\{\mathsf{a}, \mathsf{b}\}^*$. Such an instance is *solvable* if there is a nonempty sequence $(i_1, \ldots, i_n)$, $1 \leq i_j \leq m$, such that $p_{i_1} \cdots p_{i_n} = q_{i_1} \cdots q_{i_n}$. According to a classical result due to Post, it is undecidable whether an instance of the PCP is solvable.

In the following, let $P = ((p_i, q_i))_{i=1,\ldots,m}$ be a fixed instance of PCP. We say that a pair (v, w) is P-constructed from a pair of words (v', w') if, for some j, $v = p_j v'$ and $w = q_j w'$. To encode solvability of P into the theory of $FT_\leq^{fin}$, resp. $FT_\leq$, we employ the following equivalent definition of solvability:

Proposition 4.2 *P is solvable iff there is a set X of pairs of words containing a pair (w, w) with $w \neq \varepsilon$, such that every pair in X is either $(\varepsilon, \varepsilon)$ or is P-constructed from some other pair in X.*

4.1 Words and Trees

There is an obvious one-to-one encoding function γ from words in $\{\mathsf{a}, \mathsf{b}\}^*$ to feature trees: $\gamma(w) = (D_w, L_w)$ where $D_w = \{\varepsilon, \ldots, \mathsf{s}^{|w|}\}$, $L_w(\mathsf{s}^j) = w.j$ for $0 \leq j \leq |w| - 1$, and $L_w(\mathsf{s}^{|w|})$ undefined.

We define a left-inverse function $\overline{\gamma}$, that is $\overline{\gamma}(\gamma(w)) = w$, from finite feature trees to words in $\{\mathsf{a}, \mathsf{b}\}^*$ as follows: If τ does not have root feature s, or if its root is unlabeled or has label different from a and from b then $\overline{\gamma}(\tau) = \varepsilon$. Otherwise let τ' be such that $\tau[\mathsf{s}]\tau'$. We define $\overline{\gamma}(\tau) = \mathsf{a} \cdot \overline{\gamma}(\tau')$ if τ has root label a, and $\overline{\gamma}(\tau) = \mathsf{b} \cdot \overline{\gamma}(\tau')$ if τ has root label b.

To express that y denotes the fixed word π appended with the denotation of x, we define for any $\pi \in \{\mathsf{a}, \mathsf{b}\}^*$ a formula $\mathsf{app}_\pi(x, y)$, such that $\mathsf{app}_\pi[\tau, \tau']$ iff $\overline{\gamma}(\tau') = \pi\overline{\gamma}(\tau)$, by induction on π:

$$\begin{aligned}
\mathsf{app}_\varepsilon(x, y) &:= x = y \\
\mathsf{app}_{\mathsf{a}\pi}(x, y) &:= \mathsf{a}(y) \wedge \exists z\,(y[\mathsf{s}]z \wedge \mathsf{app}_\pi(x, z)) \\
\mathsf{app}_{\mathsf{b}\pi}(x, y) &:= \mathsf{b}(y) \wedge \exists z\,(y[\mathsf{s}]z \wedge \mathsf{app}_\pi(x, z))
\end{aligned}$$

Furthermore, we define $\mathsf{eps}(x)$, expressing that x denotes a tree τ with $\overline{\gamma}(\tau) = \varepsilon$, by

$$\mathsf{eps}(x) := \neg\exists y\, x[\mathsf{s}]y \vee \neg\mathsf{a}(x) \vee \neg\mathsf{b}(x)$$

Finally, the following formula expresses that x denotes a finite string:

$$\mathsf{finite}(x) := \neg\exists y\,(y[\mathsf{s}]y \wedge y \leq x)$$

In case of $FT_\leq^{fin}$ this formula is, of course, equivalent to *true*.

4.2 P-Constructions

Provided an appropriate encoding of sets of pairs of words and a predicate $\mathsf{in}(x_l, x_r, s)$, expressing that the pair (x_l, x_r) is member of the set s, we can express that s is a P-constructible set of pairs of words and that P is solvable:

$$\begin{aligned}
&\mathsf{construction}_P(s) := \forall y, y'\,(\mathsf{in}(y, y', s) \rightarrow \\
&\quad (\mathsf{eps}(y) \wedge \mathsf{eps}(y') \vee \exists z, z'\,(\mathsf{in}(z, z', s) \wedge \\
&\quad \textstyle\bigvee_{j=1\ldots m}(\mathsf{app}_{p_j}(z, y) \wedge \mathsf{app}_{q_j}(z', y')))))) \\
&\mathsf{solvable}_P := \exists s\,(\exists x\,(\mathsf{in}(x, x, s) \wedge \neg\mathsf{eps}(x) \wedge \mathsf{finite}(x)) \\
&\quad \wedge \mathsf{construction}_P(s))
\end{aligned}$$

Lemma 4.3 *For all predicates $\mathsf{in}(x, y, z)$, if $\mathsf{solvable}_P$ is valid then the instance P of the Post Correspondence Problem is solvable.*

Proof. Let σ be a fixed value for s such that $\mathsf{solvable}_P$ holds. We show by induction that if $\mathsf{in}(\tau, \tau', \sigma)$ is satisfied, then $(\overline{\gamma}(\tau), \overline{\gamma}(\tau'))$ is finitely constructible according to the instance P of the Post Correspondence Problem. □

Lemma 4.4 *There is a predicate $\mathsf{in}(x, y, z)$ such that if the instance P of the Post Correspondence Problem is solvable then $\mathsf{solvable}_P$ is valid.*

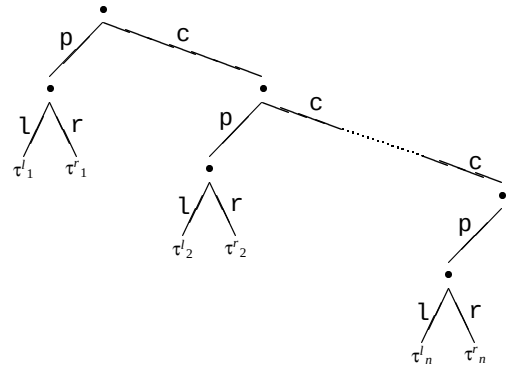


Figure 1: Representation of a sequence of pairs of trees $((\tau^l{}_i, \tau^r{}_i))_{i=1,\ldots,n}$

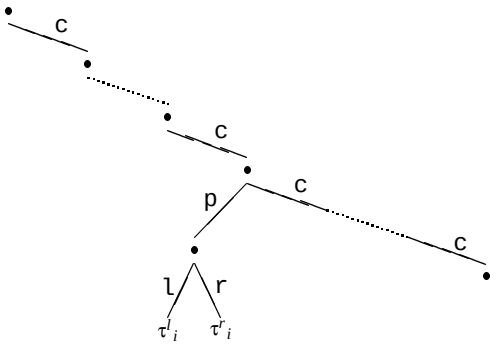


Figure 2: A possible value for x' such that $\mathsf{one\text{-}branch}(x,x')$, where x is as in Figure 1.

Proof. Since we know already how to encode words as trees, we now have to define an appropriate encoding of an arbitrary set of pairs of trees as a feature tree, together with a corresponding formula in. The representation of a sequence $(\tau^l{}_i, \tau^r{}_i)_i$ is given in Figure 1. We define, for any formula φ, a formula $\mu!x\,\varphi$ expressing that x denotes the *smallest* element satisfying φ (this formula must not be confused with $\mu x\varphi$):

$$\mu!x\,\varphi := \varphi \wedge \forall y\,(\varphi[y/x] \rightarrow x \leq y)$$

If x denotes a tree as given in Figure 1, then the formula $\mathsf{one\text{-}branch}(x,x')$ given below expresses that x' denotes a tree as given in Figure 2.

$$\begin{aligned}\mathsf{one\text{-}branch}(x,x') := \exists x_c\,(&\forall x_c\,(\mathsf{string\text{-}c}(x_c) \wedge x_c \leq x) \\ &\wedge \forall x'\,(x' \leq x \wedge \exists z\,(\mu!z\,(x_c < z \leq x'))))\end{aligned}$$

In this formula, x' is smaller than x but is strictly greater than the c-spine x_c of x. The tree x' can have only one of the p-edges of x since the set of trees between x_c and x' must have a smallest element. By the maximality of x', the tree x' contains x_c plus exactly one of the subtrees of x starting with a p-edge (see Figure 2). The following formula $\mathsf{select}(\tau^l, \tau^r, \sigma)$, where σ is as in Figure 2, expresses that τ^l is the tree $\tau^l{}_i$ and τ^r is the tree $\tau^r{}_i$:

$$\begin{aligned}\mathsf{select}(y_l, y_r, x) \quad := \quad &\exists x'\,(\mu x'(x \leq x' \wedge \exists x''\,(x'[c]x'' \wedge x'' \leq x')) \\ &\wedge \exists z\,(x'[\mathsf{p}]z \wedge z[\mathsf{l}]y_l \wedge z[\mathsf{r}]y_r))\end{aligned}$$

From a tree σ as given in Figure 2, we get the tree σ' (denoted by x') containing at all nodes c^j with $j \leq i$ a pair $(\tau^{l'}{}_j, \tau^{r'}{}_j)$ such that $\tau^l{}_i \leq \tau^{l'}{}_j$ and $\tau^r{}_i \leq \tau^{r'}{}_j$ (by Lemma 3.1). By the minimality of σ' we get that $\tau^l{}_i = \tau^{l'}{}_j$ and $\tau^r{}_i = \tau^{r'}{}_j$ for all j, hence in particular for $j = 0$ (see Figure 3). Combination of the two formulas yields

$$\mathsf{in}(y_l, y_r, x) \quad := \quad \exists x'(\mathsf{one\text{-}branch}(x,x') \wedge \mathsf{select}(y_l, y_r, x'))$$

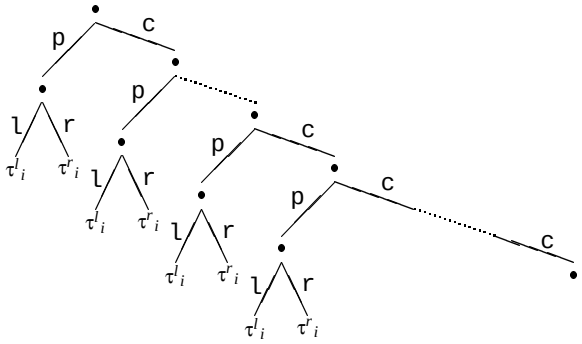


Figure 3: The value of x' in the formula $\mathsf{select}(y_l, y_r, x)$ where x is as in Figure 2.

which completes the proof. □

5 Entailment with Existential Quantifiers

In [17] it is shown that the entailment problem of $FT_{\leq}$ with existential quantifiers $\varphi \models \exists \bar{x}\varphi'$ is decidable, PSPACE-hard in the case of infinite trees and coNP-hard in the case of finite trees. We settle the precise complexity of this entailment problem in both cases.

Theorem 5.1 *Entailment of $FT_{\leq}$ with existential quantification $\varphi \models \exists \bar{x}\varphi'$ is PSPACE-complete for both structures $FT_{\leq}$ and $FT^{fin}_{\leq}$.*

In Section 5.3 we modify the PSPACE-hardness proof given in [17] for the case of infinite trees such that it proves PSPACE-hardness for both cases (Proposition 5.3). In particular, we show that we can encode the Kleene-star operator without need for infinite trees. Containment in PSPACE is shown (Proposition 5.8) by reducing in polynomial time the entailment problem to an inclusion problem

between the languages accepted by nondeterministic finite state automata (NFA). Language equivalence for NFA (and hence inclusion, since $A \subseteq B \leftrightarrow B = A \cup B$) is known to be PSPACE-complete if the alphabet contains at least two distinct symbols [8].

5.1 Path Constraints

We characterize existential $FT_{\leq}$ formulas $\exists \bar{x} \varphi$ by equivalent sets of path constraints (where sets are interpreted as conjunctions). The abstract syntax of *path constraints* ψ is defined as follows:

$$\psi ::= \quad a(x[\pi]) \mid x?[\pi]{\sim}a \mid x?[\pi]{\leq}y[\pi'] \mid x?[\pi]{\sim}y?[\pi']$$

The semantics of path constraints is given by extension of the structure $FT_{\leq}$ through the following predicates.

$a(\tau[\pi])$	iff	$(\pi, a) \in L_\tau$
$\tau?[\pi]{\sim}a$	iff	$\pi \in D_\tau$ implies $\tau[\pi]{\sim}a$
$\tau?[\pi]{\leq}\tau'[\pi']$	iff	$\pi' \in D_{\tau'}$ and if $\pi \in D_\tau$ then $\tau[\pi] \leq \tau'[\pi']$
$\tau?[\pi]{\sim}\tau'?[\pi']$	iff	if $\pi \in D_\tau$ and $\pi' \in D_{\tau'}$ then $\tau[\pi]{\sim}\tau'[\pi']$

Lemma 5.2 *For every path constraint ψ there exists an ordering constraint φ and variables $x_1, \ldots, x_n$ such that $\psi \models\!| \exists x_1 \ldots \exists x_n \varphi$.*

In the Section 5.2, we use path constraints for presenting typical examples of entailment judgements. Path constraints are also helpful for proving PSPACE-hardness in Section 5.3. In Section 5.5 we will construct a finite automaton that accepts all path constraints ψ entailed by $\exists \bar{x} \varphi$ and thereby reduce the entailment with existential quantification to the inclusion problem of finite automata.

5.2 Examples

A major difficulty in testing entailment with existential quantifiers is that there exist many equivalent $FT_{\leq}$ constraints of quite distinct syntactic shape. This makes it very difficult (if not impossible) to apply a standard technique for deciding entailment, which performs a comparison of constraints in some¡ syntactic normal form [1, 18, 28]. A first rather simple case is:

Example 3 *The formula $\exists y(x{\leq}y \wedge a(y))$ is equivalent to $x?[\varepsilon]{\sim}a$ which is equivalent to $\exists y \exists z(x{\leq}y \wedge z{\leq}y \wedge a(z))$.*

We next illustrate a more complex case of equivalent constraints with distinct syntactic shape.

Example 4 *(Figure 4) Both of the following formulas are equivalent to $x?[fg]{\sim}a$ and hence equivalent to each other:*

$$\begin{array}{ll} & \exists y \exists y' \exists z \exists z' (x{\leq}y \wedge y[f]y' \wedge y'{\leq}z \wedge z[g]z' \wedge a(z')) \\ \models\!| & \exists y \exists y' \exists z \exists z' (x{\leq}y \wedge y[f]y' \wedge z{\leq}y' \wedge z[g]z' \wedge a(z')) \end{array}$$

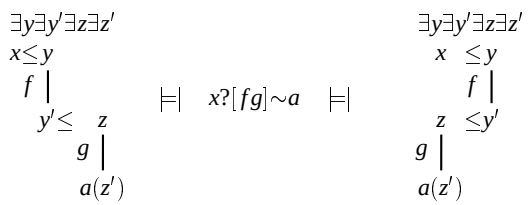


Figure 4: Graphical Presentation of Example 4

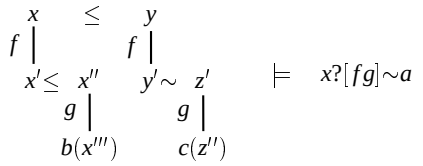


Figure 5: Graphical Presentation of Example 5

In the next example, a constraint is given that entails $x?[fg]{\sim}a$ for all a. Note that this constraint thus also entails the constraints given in the previous example.

Example 5 *(Figure 5) If $b \neq c$ then for all a the judgement*

$$\left.\begin{array}{l} x[f]x' \wedge x'{\leq}x'' \wedge x''[g]x''' \wedge b(x''') \wedge \\ x{\leq}y \wedge y[f]y' \wedge y'{\leq}u \wedge u{\geq}z' \wedge z'[g]z'' \wedge c(z'') \end{array}\right\} \models x?[fg]{\sim}a$$

holds. In other words, if α is a solution of the constraint displayed on the left hand side and if $fg \in D_{\alpha(x)}$ then fg is unlabeled in $\alpha(x)$ and hence compatible with any label a.

Example 6 *(Figure 6) The following situation illustrates a non-trivial example for entailment of selection constraints without existential quantifiers.*

$$\left.\begin{array}{l} x \leq v \wedge v[f]v' \wedge v' \leq y \wedge \\ y \leq u' \wedge u[f]u' \wedge u \leq x \end{array}\right\} \quad \models \quad x[f]y$$

The right-hand side $x[f]y$ is equivalent to the conjunction $x?[f]{\leq}y[\varepsilon] \wedge y?[\varepsilon]{\leq}x[f]$ of path constraints which are entailed by the first and second line of the left-hand side, respectively.

5.3 Entailment is PSPACE-hard

We next show that entailment is PSPACE-hard in both the finite and the infinite tree case. PSPACE-hardness follows from Proposition 5.3, which claims a polynomial

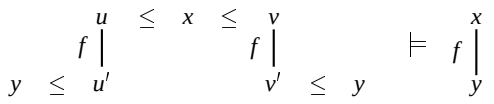


Figure 6: Graphical Representation of Example 6

reduction of the inclusion problem between regular languages over the alphabet $\mathcal{F}$ to an entailment problem between two existential $FT_\leq$ formulas. Notice that we have assumed $\mathcal{F}$ to contain at least two features.

Our PSPACE-hardness proof is based on the fact that a satisfiable ordering constraint φ may entail an infinite conjunction of path constraints, even in case of finite trees:

Example 7

1. $\forall n : x[f]y \wedge y{\leq}x \wedge a(y) \models x?[f^n]{\sim}a$.
2. $\forall n,m : x[f]x \models x?[f^m]{\leq}x[f^n]$.

For this reason the entailment problem for $FT_\leq^{fin}$ does not necessarily reduce to an inclusion problem between finite regular languages (which is decidable in coNP [8]). We consider regular expressions of the following form

$$R \quad ::= \quad \varepsilon \mid f \mid R^* \mid R_1 \cup R_2 \mid R_1 R_2$$

where $f \in \mathcal{F}$. Define, for all regular expressions R and variables x and y, the existential formula $\Theta(x,R,y)$ recursively as follows.

$$\begin{array}{lcl}
\Theta(x,\varepsilon,y) & = & x{\leq}y \\
\Theta(x,f,y) & = & \exists z\,(x{\leq}z \wedge z[f]y) \\
\Theta(x,S_1 \cup S_2,y) & = & \Theta(x,S_1,y) \wedge \Theta(x,S_2,y) \\
\Theta(x,S^*,y) & = & \exists z\,(x{\leq}z \wedge \Theta(z,S,z) \wedge z{\leq}y) \\
\Theta(x,S_1S_2,y) & = & \exists z\,(\Theta(x,S_1,z) \wedge \Theta(z,S_2,y))
\end{array}$$

Apparently, $\Theta(x,R,y)$ has size linear in the size of R.

Proposition 5.3 *For all variables x,y and for every pair of regular expressions R_1 and R_2: $\Theta(x,R_1,y) \models \Theta(x,R_2,y)$ is equivalent to $\mathcal{L}(R_2) \subseteq \mathcal{L}(R_1)$.*

Proof. It is sufficient to prove for every R that $\Theta(x,R,y)$ is equivalent to $\bigwedge\{x?[\pi]{\leq}y[\varepsilon] \mid \pi \in \mathcal{L}(R)\}$. This is done by structural induction over R, closely along the lines of the corresponding proof in [17]. □

In comparison to [17], the surprising insight here is that it needs only a minor modification to modify the PSPACE-hardness proof given for the case of infinite trees such that it also works in the case of finite trees: Here, every word in $\mathcal{L}(R)$ constrains x at the associated path *provided it exists*; in contrast, according to the encoding in [17], a word in $\mathcal{L}(R)$ constrains x at the associated path and *requires it to exist*. The encoding in [17] uses formulas $\Theta'(x,R,y)$ with the property that $\Theta'(x,R,y) \models\mid \bigwedge\{y?[\varepsilon]{\leq}x[\pi] \mid \pi \in \mathcal{L}(R)\}$ for all R.[1] Hence, every solution of $\Theta'(x,R,y)$ maps x to an infinite tree if R denotes an infinite language.

5.4 Satisfiability Test

In this section we recall the satisfiability test for $FT_\leq$ introduced in [18], which we will also need as a preprocessing step in our entailment test in Section 5.5. Clearly, satisfiability (and hence entailment) depends on the choice of finite or infinite trees. For instance, $x[f]x$ is unsatisfiable in $FT_\leq^{fin}$ but satisfiable in $FT_\leq$.

Let an *extended constraint* be a conjunction of constraints φ and (atomic) compatibility constraints $x{\sim}y$. From now on, we will only deal with extended constraints and freely call them constraints for simplicity.

In the case of infinite trees, we say that an (extended) constraint φ is F-*closed* if it satisfies the following properties for all x,y,z,x',y',f,a,b.

F1.1	$x{\leq}x \in \varphi$	if	$x \in \mathcal{V}(\varphi)$
F1.2	$x{\leq}z \in \varphi$	if	$x{\leq}y \in \varphi$ and $y{\leq}z \in \varphi$
F2	$x'{\leq}y' \in \varphi$	if	$x[f]x' \in \varphi,\ x{\leq}y \in \varphi,\ y[f]y' \in \varphi$
F3.1	$x{\sim}y \in \varphi$	if	$x{\leq}y \in \varphi$
F3.2	$x{\sim}z \in \varphi$	if	$x{\leq}y \in \varphi$ and $y{\sim}z \in \varphi$
F3.3	$x{\sim}y \in \varphi$	if	$y{\sim}x \in \varphi$
F4	$x'{\sim}y' \in \varphi$	if	$x[f]x' \in \varphi,\ x{\sim}y \in \varphi,\ y[f]y' \in \varphi$
F5	$a = b$	if	$a(x) \in \varphi,\ x{\sim}y \in \varphi,\ b(y) \in \varphi$

The rules of F1 and F2 require that φ is closed with respect to reflexivity, transitivity, and decomposition of $\leq$. The rules in F3 and F4 require that φ contains all compatibility constraints that it entails (this is proved in [18]), and F5 requires φ to be clash-free.

In the case of finite trees, we say that a constraint φ is F-*closed* if it satisfies F1-F5 and the additional *occurs check property* F6 for all $n \geq 1$, $x_1,\ldots,x_{n+1},y_1,\ldots y_n,f_1,\ldots,f_n$:

$$\mathsf{F6} \quad x_1{\leq}x_{n+1} \notin \varphi \quad \text{if} \quad x_i[f_i]y_i \wedge x_{i+1}{\leq}y_i \in \varphi \text{ for all } 1 \leq i \leq n$$

The following result is proved in [18]. It holds in both cases, for finite trees and for possibly infinite trees, but with the respective notion of F-closedness.

Proposition 5.4 *There exists a cubic time algorithm that, given a constraint φ, computes an F-closed constraint containing φ or proves its unsatisfiability. Every F-closed constraint is satisfiable.*

[1] In comparison to [17], all ordering symbols have been turned around, and in the clause $\Theta(x,f,y)$ we have exchanged the ordering and the selection constraint.

5.5 An Automaton for Path Constraints

In this section we show that for every F-closed constraint φ there is a non-deterministic automaton $\mathcal{A}_\varphi$ of size polynominal in the size of φ which accepts the set of all path constraints which are entailed by φ and which mention only symbols from a fixed set of variables, labels, and features. Note that F-closedness is a necessary assumption for our automaton construction. Note also that the automaton does not differ in the case of finite and infinite trees, only the assumed version of F-closedness differs.

Path Constraints as Words. The automaton accepts all words $\langle\psi\rangle$ associated with a path constraint ψ over some finite sub-alphabet of $\mathcal{F} \cup \mathcal{L} \cup \mathcal{V} \cup \{\leq, \sim, ?, [,], (,)\}$. In first approximation, let $\langle\psi\rangle$ be the *concrete syntax* of ψ. There is however a serious problem with recognizing the concrete syntax of entailed constraints:

Example 8 *The set of path constraints entailed by $x{\leq}x$ is $\{x?[\pi]{\sim}x?[\pi] \mid \pi \in \mathcal{F}^*\} \cup \{x?[\varepsilon]{\leq}x[\varepsilon]\}$ (when restricted to the variables in $x{\leq}x$). If $\langle\varphi\rangle$ denotes the abstract syntax of φ then the set $\{\langle\psi\rangle \mid x{\leq}x \models \psi\}$ is not regular.*

We therefore have to alter the definition of $\langle\varphi\rangle$ slightly but fundamentally. The trick is to "factor out" the maximal common suffix of the two paths in a path constraint of the form $x?[\pi_1]{\sim}y?[\pi_2]$. More exactly, we add the symbol # to the alphabet and alter the definition of $\langle\psi\rangle$ such that:

$$\langle x?[\pi_1]{\sim}y?[\pi_2]\rangle = x?[\pi]{\sim}y?[\pi']\#\pi''$$

where π'' is the longest common suffix of π_1 and π_2 such that $\pi_1 = \pi\pi''$ and $\pi_2 = \pi'\pi''$; *i.e.*, in $x?[\pi]{\sim}y?[\pi']\#\pi''$ we require that either π and π' end with distinct feature symbols or that at least one of them is the empty path. This solves the problem of Example 8: With respect to the new definition of $\langle\psi\rangle$ the set $\{\langle\varphi\rangle \mid x{\leq}x \models \psi\}$ is regular:

$$\{x?[\varepsilon]{\sim}x?[\varepsilon]\#\pi \mid \pi \in \mathcal{F}^*\} \cup \{x?[\varepsilon]{\leq}x[\varepsilon]\}$$

The definition of $\langle\psi\rangle$ does also adjust some more difficult regularity problems raised by trivial consequences. All these consequences are raised by the following valid entailment judgement:

$$x?[\pi]{\sim}y?[\pi'] \models x?[\pi\pi'']{\sim}y?[\pi'\pi'']$$

Example 9 *The set $\{\langle\psi\rangle \mid x?[gf]{\sim}y?[ff] \models \psi\}$ restricted to words with features f, g and variables x, y is:*

$$\begin{array}{ll} & \{x?[g]{\sim}y?[f]\#f\pi \mid \pi \in \{f,g\}^*\} \\ \cup & \{z?[\varepsilon]{\sim}z?[\varepsilon]\#\pi \mid z \in \{x,y\}\} \\ \cup & \{z?[\varepsilon]{\leq}z[\varepsilon] \mid z \in \{x,y\}\} \end{array}$$

By Lemma 5.2 there exists an existential formula equivalent to $x?[gf]{\sim}y?[ff]$; in fact, there are many of them.

Overall Structure of the Automaton. If we consider an entailment problem of the form $\varphi \models \exists\bar{x}\varphi'$ then we construct two non-deterministic automata $\mathcal{A}_\varphi$ and $\mathcal{A}^{\bar{x}}_{\varphi'}$ with the alphabet $\mathcal{F}(\varphi \wedge \varphi') \cup \mathcal{L}(\varphi \wedge \varphi') \cup \mathcal{V}(\varphi \wedge \exists\bar{x}\varphi') \cup \{\leq, \sim, ?, [,], (,), \#\}$.

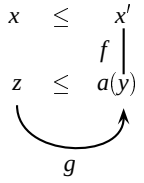
Each automaton constructed falls into four parts (sharing only the initial state q_s and the accepting state q_f), corresponding to the four kinds of path constraints. We just explain the construction of $\mathcal{A}_\varphi$ (and equally $\mathcal{A}_{\varphi'}$) for the quantifier free. The automaton $\mathcal{A}'^{\bar{x}}_\varphi$ for the constraint $\exists\bar{x}\varphi'$ is easily obtained from $\mathcal{A}_{\varphi'}$ by filtration of all words containing variables in $\bar{x}$ that is by removing all transitions labeled with a symbol from $\bar{x}$. Note that the local variables in $\bar{x}$ do not occur in the alphabet of $\mathcal{A}^{\bar{x}}_{\varphi'}$; they do only matter for the definition of its states.

The construction of the automaton $\mathcal{A}_\varphi$ is given in Figure 7. It is completely spelled out except of one additional symmetry rule (40) which can be expressed through a dozen further transitions. In the rest of this section we explain this construction.

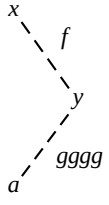
Constraints as Graphs. Our construction of the automaton is motivated by considering constraints as graphs. For instance, the constraint of Example 4

$$x{\leq}x' \wedge x'[f]y \wedge a(y) \wedge z{\leq}y \wedge z[g]y$$

can be depicted as the following graph, where variables are represented as nodes.



Intuitively, when the automaton $\mathcal{A}_\varphi$ accepts a word $\langle\psi\rangle$ it traverses the constraint graph associated with φ where ψ is associated a certain traversal pattern. We will depict such traversal patterns graphically; for instance, the above graph allows for the following traversal:

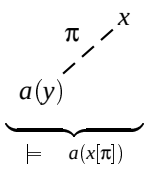


In these pictures, the horizontal dimension corresponds to the ordering $\leq$ (left to right) and the vertical one corresponds to feature selection (top to bottom).

1	q_s	$\xrightarrow{a(x[}$	$/x[a]$	
2	$/x[a]$	$\xrightarrow{\varepsilon}$	$/y[a]$	$x \geq y \in \varphi$
3	$/x[a]$	$\xrightarrow{f}$	$/y[a]$	$x[f]y \in \varphi$
4	$/x[a]$	$\xrightarrow{])}$	q_f	$a(x) \in \varphi$
5	q_s	$\xrightarrow{x?[}$	$\backslash x$	
6	$\backslash x$	$\xrightarrow{\varepsilon}$	$\backslash y$	$x \leq y \in \varphi$
7	$\backslash x$	$\xrightarrow{f}$	$\backslash y$	$x[f]y \in \varphi$
8	$\backslash x$	$\xrightarrow{]\leq y[}$	$/y[\backslash x]$	
9	$/x[\backslash z]$	$\xrightarrow{\varepsilon}$	$/y[\backslash z]$	$x \geq y \in \varphi$
10	$/x[\backslash z]$	$\xrightarrow{f}$	$/y[\backslash z]$	$x[f]y \in \varphi$
11	$/x[\backslash x]$	$\xrightarrow{]}$	q_f	
12	q_s	$\xrightarrow{x?[}$	$\backslash\backslash xx$	
13	$\backslash\backslash xx'$	$\xrightarrow{\varepsilon}$	$\backslash\backslash yy'$	$x \leq y, x' \leq y' \in \varphi$
14	$\backslash\backslash xx'$	$\xrightarrow{f}$	$\backslash\backslash yy'$	$x[f]y, x'[f]y' \in \varphi$
15	$\backslash\backslash xx'$	$\xrightarrow{\varepsilon}$	$\rangle\backslash yx'$	$x \sim y \in \varphi$
16	$\rangle\backslash xx'$	$\xrightarrow{\varepsilon}$	$\rangle\backslash yy'$	$x \geq y, x' \leq y' \in \varphi$
17	$\rangle\backslash xx'$	$\xrightarrow{f}$	$\rangle\backslash yy'$	$x[f]y, x'[f]y' \in \varphi$
18	$\rangle\backslash xx'$	$\xrightarrow{\varepsilon}$	$\rangle\rangle xy'$	$x' \sim y' \in \varphi$
19	$\rangle\rangle xx'$	$\xrightarrow{\varepsilon}$	$\rangle\rangle yy'$	$x \geq y, x' \geq y' \in \varphi$
20	$\rangle\rangle xx'$	$\xrightarrow{f}$	$\rangle\rangle yy'$	$x[f]y, x'[f]y' \in \varphi$
21	$\rangle\rangle xx'$	$\xrightarrow{]\sim a}$	q_f	$a(x) \in \varphi$
22	$\rangle\rangle xx'$	$\xrightarrow{]\sim c}$	q_f	$a(x), b(x') \in \varphi, a \neq b$
23	q_s	$\xrightarrow{x?[}$	$\backslash x[\varepsilon]$	
24	$\backslash x[h]$	$\xrightarrow{\varepsilon}$	$\backslash y[h]$	$x \leq y \in \varphi$
25	$\backslash x[h]$	$\xrightarrow{f}$	$\backslash y[f]$	$x[f]y \in \varphi$
26	$\backslash x[h]$	$\xrightarrow{]\sim y?[}$	$\backslash y[\backslash x, h, \varepsilon]$	
27	$\backslash x[\backslash z, h, g]$	$\xrightarrow{\varepsilon}$	$\backslash y[\backslash z, h, g]$	$x \leq y \in \varphi$
28	$\backslash x[\backslash z, h, g]$	$\xrightarrow{f}$	$\backslash y[\backslash z, h, f]$	$x[f]y \in \varphi$
29	$\backslash x[\backslash z, h, g]$	$\xrightarrow{]\#}$	$\backslash x\backslash z$	$h \neq g \vee h = g = \varepsilon$
30	$\backslash x[\backslash z, h, g]$	$\xrightarrow{\varepsilon}$	$\rangle y[\backslash z, h, g]$	$x \sim y \in \varphi$
31	$\rangle x[\backslash z, h, g]$	$\xrightarrow{\varepsilon}$	$\rangle y[\backslash z, h, g]$	$x \geq y \in \varphi$
32	$\rangle x[\backslash z, h, g]$	$\xrightarrow{f}$	$\rangle y[\backslash z, h, f]$	$x[f]y \in \varphi$
33	$\rangle x[\backslash z, h, g]$	$\xrightarrow{]\#}$	$\backslash x\rangle z$	$h \neq g \vee h = g = \varepsilon$
34	$\backslash x\backslash x'$	$\xrightarrow{\varepsilon}$	$\backslash y\backslash y'$	$x \leq y, x' \leq y' \in \varphi$
35	$\backslash x\backslash x'$	$\xrightarrow{f}$	$\backslash y\backslash y'$	$x[f]y, x'[f]y' \in \varphi$
36	$\backslash x\backslash x'$	$\xrightarrow{\varepsilon}$	$\backslash x\rangle y'$	$x' \sim y' \in \varphi$
37	$\backslash x\rangle x'$	$\xrightarrow{\varepsilon}$	$\backslash y\rangle y'$	$x \leq y, x' \geq y' \in \varphi$
38	$\backslash x\rangle x'$	$\xrightarrow{f}$	$\backslash y\rangle y'$	$x[f]y, x'[f]y' \in \varphi$
39	$\backslash x\rangle x$	$\xrightarrow{\mathcal{F}^*}$	q_f	
40	$\dfrac{x?[\pi]\sim y?[\pi']\#\pi'' \in \mathcal{L}(\mathcal{A}_\varphi)}{y?[\pi']\sim x?[\pi]\#\pi'' \in \mathcal{L}(\mathcal{A}_\varphi)}$			

Figure 7: The automaton $\mathcal{A}_\varphi$ recognizing the path constraints implied by φ.

Labeling Path Constraints. The subautomaton comprising rules 1–4 recognizes all the constraints $a(x[\pi])$ entailed by φ. The associated pattern looks as follows.



In particular, note that rules 2 and 3 allow to derive every constraint of the form $y?[\varepsilon] \leq x[\pi]$ entailed by φ:

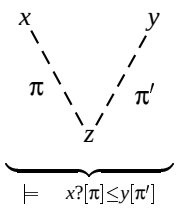
$$\varphi \models y?[\varepsilon] \leq x[\pi] \quad \text{if} \quad /x[a] \xrightarrow{\pi} /y[a]$$

for any label a. Here, we memorize in the state the label a read at the beginning of the constraint (rule 1) to check it against a labeling constraint in φ (rule 4).

Ordering Path Constraints. The next group of rules 5–11 serves to recognize constraints of the form $x?[\pi] \leq y[\pi']$. Note that $\varphi \models x?[\pi] \leq y[\pi']$ iff there is a z such that

$$\varphi \models x?[\pi] \leq z[\varepsilon] \quad (1)$$
$$\varphi \models z?[\varepsilon] \leq y[\pi'] \quad (2)$$

The associated graph pattern is this one:



Condition (1) is verified by the rules 6 and 7, while condition (2) is checked by rules 9 and 10 which are analogous to 2 and 3 except that we do not memorize a label but the variable z found in condition (1).

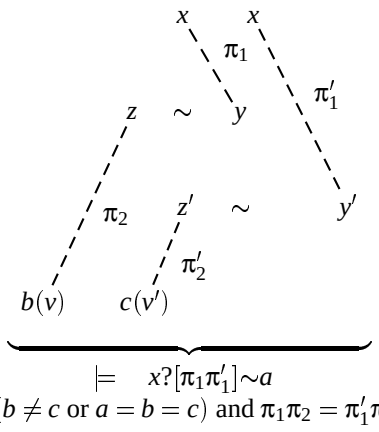
Example 10 *The constraint from Example 6 entails $x[f]y$. This selection constraint is equivalent to the conjunction of the two path constraints $x?[f]{\leq}y[\varepsilon]$ and $y?[\varepsilon]{\leq}x[f]$.*

$$\begin{aligned} q_s &\xrightarrow{x?[} \backslash x \xrightarrow{\varepsilon} \backslash v \xrightarrow{f} \backslash v' \\ &\xrightarrow{\varepsilon} \backslash y \xrightarrow{]\leq y[} /y[\backslash y] \xrightarrow{]} q_f \\ q_s &\xrightarrow{y?[} \backslash y \xrightarrow{\varepsilon} \backslash u' \xrightarrow{]\leq x[} /x[\backslash u'] \\ &\xrightarrow{\varepsilon} /u[\backslash u'] \xrightarrow{f} /u'[\backslash u'] \xrightarrow{]} q_f \end{aligned}$$

Label Compatibility. Rules 12–22 check constraints of the kind $x?[\pi]{\sim}a$. Note that $\varphi \models x?[\pi]{\sim}a$ iff there are y,y',z,z',v,v',b,c and $\pi_1,\pi_1',\pi_2,\pi_2'$ with $\pi = \pi_1\pi_2 = \pi_1'\pi_2'$ such that:

$$\begin{aligned} \varphi &\models x?[\pi_1]{\leq}y[\varepsilon] \wedge y{\sim}z && (3)\\ \varphi &\models x?[\pi_1']{\leq}y'[\varepsilon] \wedge y'{\sim}z' && (4)\\ \varphi &\models v?[\varepsilon]{\leq}z[\pi_2], && (5)\\ \varphi &\models v'?[\varepsilon]{\leq}z'[\pi_2'] && (6)\\ \varphi &\models b(v) \wedge c(v') \text{ and } (b \neq c \text{ or } a = b = c) && (7) \end{aligned}$$

The associated pattern look as follows.



$\models \quad x?[\pi_1\pi_1']{\sim}a$
if $(b \neq c \text{ or } a = b = c)$ and $\pi_1\pi_2 = \pi_1'\pi_2'$

We check the conditions (3) and (4) as well as (5) and (6) in parallel where we assume, by symmetry, that π_1 is a prefix of π_1'. With the names used above, the automaton consumes π_1 by rules 13–14, switches y to z with rule 15, then consumes π_2 minus its suffix π_2' (which is identical to π_1' minus its prefix π_1) by rules 16–17, switches form y' to z' in rule 18 and consumes π_2' in rules 19–20. Finally rules 21–22 check the label constraints (7).

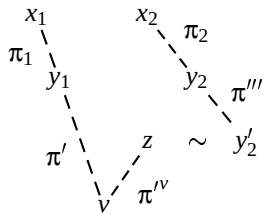
Example 11 *In the case of Example 5 we obtain*

$$\begin{aligned} q_s &\xrightarrow{x?[} \backslash\backslash xx \xrightarrow{\varepsilon} \backslash\backslash xy \xrightarrow{f} \backslash\backslash x'y' \\ &\xrightarrow{\varepsilon} \rangle\backslash x''y' \xrightarrow{\varepsilon} \rangle\rangle x''z' \xrightarrow{g} \rangle\rangle x'''z'' \\ &\xrightarrow{]\sim a} q_f \end{aligned}$$

Path Compatibility. Rules 23–39 check for constraints $x_1?[\pi_1]{\sim}x_2?[\pi_2']\#\pi'\pi''$. One possible justification for $\varphi \models x_1?[\pi_1]{\sim}x_2?[\pi_2]\#\pi'\pi''$ is that there are variables $y_1,y_2,y_2'{\sim}z,v$ and paths π''',π'^{v} such that $\pi' = \pi'''\pi'^{v}$ and

$$\begin{aligned} \varphi &\models x_1?[\pi_1]{\leq}y_1[\varepsilon] && (8)\\ \varphi &\models y_1?[\pi']{\leq}v[\varepsilon] && (9)\\ \varphi &\models x_2?[\pi_2]{\leq}y_2[\varepsilon] && (10)\\ \varphi &\models y_2?[\pi''']{\leq}y_2'[\varepsilon] && (11)\\ \varphi &\models v?[\varepsilon]{\leq}z[\pi'^{v}] && (12) \end{aligned}$$

These situations correspond to the following pattern:



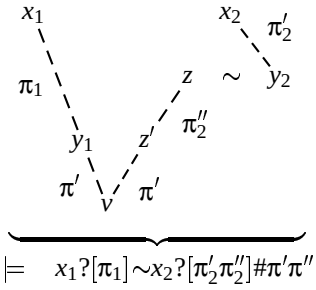
$\models \quad x_1?[\pi_1]{\sim}x_2?[\pi_2]\#\pi'''\pi'^{v}\pi''$

The rules 23–29 and 34–39 deal with this situation: Rules 23–29 and 34–35 consume π_1 and π_2, rules 37 and 38 the common suffix π' that is explicit in the constraint, and rule 39 an arbitrary common suffix π''': In order to guarantee that π_1 and π_2 have no common suffix themselves, the automaton must memorize the last feature in π_1 and π_2 and check them to be distinct before switching in rule 33. In order to allow for π_1 and/or π_2 to equal ε, the automaton also memorizes whether or not a feature has been consumed at all (rule 26). Slightly abusing notation, we allow for h and g in these rules to denote either a feature symbol or ε.

The second justification is similar but contains the switch through the compatibility constraint $\sim$ within π_2 instead of π'; *i.e.*, there are variables $y_1,y_2,y_2{\sim}z,z',v$ and paths π_2',π_2'' such that $\pi_2 = \pi_2'\pi_2''$ and

$$\begin{aligned} \varphi &\models x_1?[\pi_1]{\leq}y_1[\varepsilon] && (13)\\ \varphi &\models y_1?[\pi']{\leq}v[\varepsilon] && (14)\\ \varphi &\models x_2?[\pi_2']{\leq}y_2[\varepsilon] && (15)\\ \varphi &\models z'?[\varepsilon]{\leq}z[\pi_2''] && (16)\\ \varphi &\models v?[\varepsilon]{\leq}z'[\pi'] && (17) \end{aligned}$$

The associated pattern is:



For the traversal of this pattern we need the additional rules 30–33.

For both situations there also is the symmetric one which contains the switch through the compatibility constraint $\sim$ in the branch for x_1. We do not detail the automaton checking for these possibilities since its definition is completely symmetric to the rules 23–39.

5.6 Deciding Entailment in PSPACE

In order to decide $\varphi \models \exists\bar{x}\varphi'$, we test satisfiability of φ and $\varphi \wedge \exists\bar{x}\varphi'$. By Proposition 5.4, this can be done in time $O(n^3)$ where n is the size of the entailment problem. If one of the tests fails, entailment is trivial. Otherwise, we compute the F-closures of φ and of φ' and define the associated automaton $\mathcal{A}_\varphi$ and $\mathcal{A}^{\bar{x}}_{\varphi'}$ in time $O(n^3)$. By Proposition 5.8, $\varphi \models \exists\bar{x}\varphi'$ if and only if $\mathcal{L}(\mathcal{A}^{\bar{x}}_{\varphi'}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$. This inclusion is decidable in PSPACE [8].

Proposition 5.5 (Correctness) *If* $\langle\psi\rangle \in \mathcal{L}(\mathcal{A}^{\bar{x}}_\varphi)$ *then* $\exists\bar{x}\varphi \models \psi$.

Proof. By induction over the paths mentioned in ψ. □

Lemma 5.6 (Key) *For all paths* $\mu_1, \mu_2, \pi_1, \pi_2$, *variables* x, y_1, y_2, *and constraints* φ*:*

1. *If* $\langle y_1?[\mu_1]{\leq}x[\pi_1]\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$ *and* $\langle a(x[\pi_1\pi_2])\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$ *then* $\langle y_1?[\mu_1\pi_2]{\sim}a\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$
2. *If* $\langle y_1?[\mu_1]{\leq}x[\pi_1]\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$ *and* $\langle y_2?[\mu_2]{\leq}x[\pi_1\pi_2]\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$ *then* $\langle y_1?[\mu_1\pi_2]{\sim}y_2?[\mu_2]\rangle \in \mathcal{L}(\mathcal{A}_\varphi)$.

The path constraints are indeed expressive enough to capture all logic consequences of a constraint with existential quantifiers:

Proposition 5.7 (Characterization) *If* φ *is an* F*-closed* $FT_\leq$ *constraint, then*

$$\exists\bar{x}\varphi \quad \models \quad \bigwedge\{\psi \mid \langle\psi\rangle \in \mathcal{L}(\mathcal{A}^{\bar{x}}_\varphi)\}\,.$$

Proof. The direction from left to right (correctness) follows from Proposition 5.5. For the direction from right to left assume a solution α of $\bigwedge\{\psi \mid \langle\psi\rangle \in \mathcal{L}(\mathcal{A}^{\bar{x}}_\varphi)\}$. Define an extension α' of α by setting, for all $x \in \mathcal{V}(\exists\bar{x}\varphi)$: $\alpha'(x) = \alpha(x)$, and for all $x \in \{\bar{x}\}$: $\alpha'(x) = \tau$ where

$$\begin{aligned} D_{\alpha'(x)} = \;& \{\pi \mid \langle x?[\pi]{\leq}x[\pi]\rangle \in \mathcal{L}(\mathcal{A}_{\varphi'})\} \cup \\ & \{\pi\pi'' \mid z \in V,\ \pi'\pi'' \in D_{\alpha(z)}, \\ & \qquad \langle z?[\pi']{\leq}x[\pi]\rangle \in \mathcal{L}(\mathcal{A}_{\varphi'})\} \\ L_{\alpha'(x)} = \;& \{(\pi, a) \mid a(x[\pi]) \in \mathcal{L}(\mathcal{A}_{\varphi'})\} \cup \\ & \{(\pi\pi'', a) \mid z \in V,\ (\pi'\pi'', a) \in L_{\alpha(z)}, \\ & \qquad \langle z?[\pi']{\leq}x[\pi]\rangle \in \mathcal{L}(\mathcal{A}_{\varphi'})\} \end{aligned}$$

It is obvious that $D'_\alpha(x)$ is prefix closed. The Key Lemma 5.6 implies that $L'_\alpha(x)$ is a partial function. It can be shown by a case distinction that α' satisfies all basic constraints in φ'. For details see the full paper [19]. □

Proposition 5.8 (Reduction) *Let* φ *and* φ' *be closed* $FT_\leq$ *constraints and* $\bar{x}$ *a sequence of variables such that all free variables in* $\exists\bar{x}\varphi'$ *occur in* φ*. Further assume that* $\varphi \wedge \exists\bar{x}\varphi'$ *is satisfiable. Then*

$$\varphi \models \exists\bar{x}\varphi' \qquad \textit{iff} \qquad \mathcal{L}(\mathcal{A}^{\bar{x}}_{\varphi'}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)\,.$$

Proof. The direction from right to left follows directly from Proposition 5.7. For the inverse direction we show that for all ψ with $\mathcal{V}(\psi) \subseteq \mathcal{V}(\varphi)$ it holds that $\langle\psi\rangle \notin \mathcal{L}(\mathcal{A}_\varphi)$ implies $\langle\psi\rangle \notin \mathcal{L}(\mathcal{A}^{\bar{x}}_\varphi)$. For more details, see the full paper [19]. □

Acknowledgments. The research reported in this paper has been supported by the BMBF (FKZ ITW 9601), the Esprit Working Group CCL II (EP 22457), and the DFG (SFB 378). Special thanks are due to Denys Duchier for providing the initial zigzag macros.

References

[1] H. Aït-Kaci, A. Podelski, G. Smolka. A feature-based constraint system for logic programming with entailment. *Theoretical Computer Science*, 122(1–2):263–283, 1994.

[2] R. Backofen. A complete axiomatization of a theory with feature and arity constraints. *J. of Logic Programming*, 24(1&2):37–71, 1995.

[3] R. Backofen, G. Smolka. A complete and recursive feature theory. *Theoretical Computer Science*, 146(1–2):243–268, 1995.

[4] F. Bourdoncle, S. Merz. Type checking higher-order polymorphic multi-methods. In *24th ACM Symposium on Principles of Programming Languages*, 302–315, Paris, France, Jan. 1997. ACM Press, New York.

[5] B. Carpenter. *The Logic of Typed Feature Structures - with Applications to Unification Grammars, Logic Programs and Constraint Resolution*. No. 32 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.

[6] J. Dörre. Feature logics with weak subsumption constraints. In *Annual Meeting of the Association of Computational Linguistics*, 256–263, 1991.

[7] J. Dörre, W. C. Rounds. On subsumption and semiunification in feature algebras. In *5th LICS*, 300–310, 1990.

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

[9] F. Henglein, J. Rehof. The complexity of subtype entailment for simple types. In *12th LICS*, Warsaw, Poland, 1997. IEEE Computer Society Press.

[10] F. Henglein, J. Rehof. Constraint automata and the complexity of recursive subtype entailment. In K. Larsen (ed.), *25th Internat. Conf. on Automata, Languages, and Programming*, LNCS, Aalborg, Denmark, 1998. Springer-Verlag, Berlin, Germany. to appear.

[11] R. M. Kaplan, J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, 173–281. The MIT Press, 1982.

[12] R. T. Kasper, W. C. Rounds. A logical semantics for feature structures. In *Annual Meeting of the Association of Computational Linguistics*, 257–265, 1986.

[13] M. J. Maher. Logic semantics for a class of committed-choice programs. In J.-L. Lassez (ed.), *Internat. Conf. on Logic Programming*, 858–876. The MIT Press, 1987.

[14] S. Marlow, P. Wadler. A practical subtyping system for Erlang. In *2nd ACM SIGPLAN Internat. Conf. on Functional Programming*, 136–149. ACM Press, New York, 1997.

[15] M. Müller. Ordering constraints over feature trees with ordered sorts. In P. Lopez, S. Manandhar, W. Nutt (eds.), *Computational Logic and Natural Language Understanding*, LNAI, to appear. Springer-Verlag, Berlin, Germany.

[16] M. Müller. *Set-based Failure Diagnosis for Concurrent Constraint Programming*. Doctoral Dissertation. Universität des Saarlandes, Tech. Fak., Germany, 1998. In preparation.

[17] M. Müller, J. Niehren. Ordering constraints over feature trees expressed in second-order monadic logic. In T. Nipkow (ed.), *Internat. Conf. on Rewriting Techniques and Applications*, no. 1379 in LNCS, 196–210, Tsukuba, Japan, 1998.

[18] M. Müller, J. Niehren, A. Podelski. Ordering constraints over feature trees. In G. Smolka (ed.), *3rd Internat. Conf. on Principles and Practice of Constraint Programming*, vol. 1330 of *LNCS*, 297–311, 1997.

[19] M. Müller, J. Niehren, R. Treinen. The first-order theory of ordering constraints over feature trees. `http://www.ps.uni-sb.de/Papers/abstracts/FTSubTheory-98.html`.

[20] J. Palsberg. Efficient inference of object types. In *9th LICS*, 186–185, 1994.

[21] C. Pollard, I. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Cambridge University Press, Cambridge, England, 1994.

[22] F. Pottier. Simplifying subtyping constraints. In *ACM SIGPLAN Internat. Conf. on Functional Programming*, 122–133. ACM Press, 1996.

[23] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[24] W. C. Rounds. Feature logics. In J. v. Benthem, A. ter Meulen (eds.), *Handbook of Logic and Language*. Elsevier Science Publishers B.V. (North Holland), 1997.

[25] V. A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, 1993.

[26] S. Shieber. *An Introduction to Unification-based Approaches to Grammar*. CSLI Lecture Notes No. 4. Center for the Study of Language and Information, 1986.

[27] G. Smolka. Feature constraint logics for unification grammars. *J. of Logic Programming*, 12:51–87, 1992.

[28] G. Smolka, R. Treinen. Records for logic programming. *J. of Logic Programming*, 18(3):229–258, 1994.

[29] J. W. Thatcher, J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[30] R. Treinen. A new method for undecidability proofs of first order theories. *J. of Symbolic Computation*, 14:437–457, 1992.

[31] R. Treinen. Feature trees over arbitrary structures. In P. Blackburn, M. de Rijke (eds.), *Specifying Syntactic Structures*, chap. 7, 185–211. CSLI Publications and FoLLI, 1997.

[32] V. Trifonov, S. Smith. Subtyping constrained types. In R. Cousot, D. A. Schmidt (eds.), *3rd Internat. Static Analysis Symposium*, vol. 1145 of *LNCS*, 349–365, Aachen, 1996. Springer-Verlag, Berlin, Germany.

[33] P. Van Roy, M. Mehl, R. Scheidhauer. Integrating efficient records into concurrent constraint programming. In *Internat. Symposium on Programming Language Implementation and Logic Programming*, Aachen, Germany, Sep. 1996. Springer-Verlag.

[34] S. Vorobyov. An improved lower bound for the elementary theories of trees. In *Internat. Conf. on Automated Deduction*, vol. 1104 of *LNCS*, 275–287, 1996.