Ordering Constraints over Feature Trees with Ordered Sorts

Martin Müller

Programming Systems Lab, Universität des Saarlandes,
66041 Saarbrücken, Germany, `mmueller@ps.uni-sb.de`

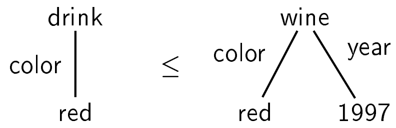
Abstract. Feature constraints have been used to accommodate records in constraint programming and record like structures in computational linguistics. The constraint system $FT_{\leq}$ [23] provides information ordering constraints interpreted over feature trees. Here, we generalise $FT_{\leq}$ to $FT_{\leq}(\mathrm{sort})$ by assuming a partial order on the sorts. This extension has important applications in the context of computational linguistics and program analysis which we sketch in the paper. A feature tree τ_1 has less information than a feature tree τ_2 if and only if τ_1 has fewer edges than τ_2, and node labelled with smaller sorts. We investigate the satisfiability problem of $FT_{\leq}(\mathrm{sort})$ constraints. Its complexity depends on the order on sorts: We give a criterion for polynomial complexity based on the coherence property of partial orders, and we apply it to special cases including trees and partial orders with finitely many maximal elements. We also obtain the cubic complexity result of $FT_{\leq}$ as a corollary. We recall from [23] the reduction of the satisfiability problem of Dörre's weak-subsumption constraints to the satisfiability problem of $FT_{\leq}$.

Keywords: feature constraints, partially ordered sorts, tree orderings, weak subsumption

1 Introduction

Feature constraints have been used for describing records in constraint programming [1, 35, 34] and of record like structures in computational linguistics [15, 14, 31]. Following [2, 4, 3] we consider feature constraints as predicate logic formulae that are interpreted in the structure of feature trees.

A feature tree is a possibly infinite tree with unordered and labelled edges and with possibly labelled nodes. Edge labels are functional in that the labels of distinct edges departing from the same node must be distinct. A feature tree represents symbolic information where, if τ_1 and τ_2 are feature trees, τ_1 represents less information than τ_2 if τ_1 has fewer edges than τ_2 and nodes with smaller (*i. e.*, less specific) sorts. In this case, we say that τ_1 is smaller than τ_2 with respect to the information ordering and write $\tau_1 \leq \tau_2$. For example, assuming two sorts drink and wine where drink is smaller than wine, we have:



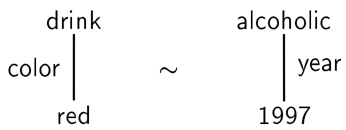
In [23], we have introduced the constraint system $FT_{\leq}$ of information ordering constraints over feature trees, which in turn is obtained from the constraint system FT [2] by adding ordering constraints.[1] Here, we generalise $FT_{\leq}$ by allowing the sorts to be partially ordered themselves, where we assume every two sorts to have (*i*) a common lower bound, and (*ii*) a least upper bound if they have some upper bound. The syntax of $FT_{\leq}$(sort) constraints φ is defined by

$$\varphi \;::=\; x \leq x' \quad | \quad x[f]x' \quad | \quad a(x) \quad | \quad \varphi \wedge \varphi'$$

where x and x' are variables, f is a feature, and a is a sort. The semantics of $FT_{\leq}$ is given by interpretation over feature trees where the symbol $\leq$ is interpreted as information ordering on feature trees. The semantics of $x[f]y$ is defined as in FT. The semantics of $a(x)$ says that the root node of the denotation of x must have a sort greater than a. For instance, both trees above are possible denotations for x in solutions of the constraint $\mathsf{drink}(x) \wedge x[\mathsf{color}]x' \wedge \mathsf{red}(x')$.

In [23], we have given a satisfiability test for $FT_{\leq}$ constraints which takes worst case time $O(n^3)$. This algorithm computes the fixed-point closure of a constraint with respect to a set of axioms which characterises satisfiability of $FT_{\leq}$ constraints. Here, we adapt the algorithm to the sorted case and give a criterion on the partial order of sorts which guarantees the satisfiability problem to remain polynomial. Most importantly, we obtain worst case time complexity $O(n^3)$ if the sorts form a tree, and $O(n^{2k-1})$ if every sort is smaller than one of at most k maximal sorts.

Our algorithm relies on the following derived relation between a finite number of feature trees. Feature trees τ_1 through τ_n are said to be compatible, written $\tau_1 \sim \ldots \sim \tau_n$ or $\sim(\tau_1 \ldots \tau_n)$, if their information content can be combined in a feature tree τ which is greater than all of them. For instance, assuming another sort alcoholic smaller than wine, the following holds.



This is witnessed by the right feature tree at the top of this page, whose root node combines the sorts alcoholic and drink in the sort wine. As a second example, assume sorts $\{\mathsf{a\ or\ b}, \mathsf{b\ or\ c}, \mathsf{a\ or\ c}, \mathsf{a}, \mathsf{b}, \mathsf{c}\}$ ordered by $\mathsf{a\ or\ b} \leq \mathsf{a}$, $\mathsf{a\ or\ b} \leq \mathsf{b}$, *etc.*,

[1] For further related work on feature constraints see [23].

and note that the three disjunctive sorts are incompatible while every two of them are compatible.

$$\mathsf{a\ or\ b} \sim \mathsf{b\ or\ c} \qquad \mathsf{a\ or\ b} \sim \mathsf{a\ or\ c} \qquad \mathsf{b\ or\ c} \sim \mathsf{a\ or\ c}$$

This compatibility relation is captured by additional constraints $\sim (x_1 \ldots x_n)$ whose semantics is defined equivalent to $\exists y(x_1 \leq y \wedge \ldots \wedge x_n \leq y)$. These constraints are essential to ensure termination of our algorithm since they avoid the introduction of fresh variables. Note that for every constraint φ, there are exponentially many compatibility constraints over variables in φ. Often, however, it suffices to compute only a polynomial number of them to decide satisfiability of φ. Whether this is the case depends on the particular partial order on sorts. We distinguish the (generalised) coherence-property of partial orders as essential and show how it is used to estimate the degree of the polynomial.

Plan of the Paper. In Section 2 we sketch two application domains of the constraint system $FT_{\leq}(\text{sort})$, one in computational linguistics and one in program analysis. Section 3 defines $FT_{\leq}(\text{sort})$. We characterise satisfiability of $FT_{\leq}(\text{sort})$ constraints and show decidability. In Section 4 we show that our satisfiability test can be organised to have polynomial complexity; Section 4.1 gives the general criterion, and Section 4.2 applies it to special cases. In Section 5 we show $FT_{\leq}(\text{sort})$ to be a proper generalisation of $FT_{\leq}$, and Section 6 recalls that our algorithm yields an $O(n^3)$ algorithm for satisfiability of weak subsumption constraints (see below). Section A in the appendix contains the proof of Proposition 4.

2 Application Domains

2.1 Natural Language Processing

Constraint-based grammar formalisms (*e. g.*, [15, 14, 32, 31, 27]) make fundamental use of constraint satisfiability. Roughly, the analysis of a sentence in such a formalism accumulates constraints according to which lexical entries, grammar rules, or global principles are applied. The solutions of this set of constraints correspond to the syntactic structure associated to the sentence. Hence, for the sentence to be well-formed, this set of constraints must always be satisfiable.

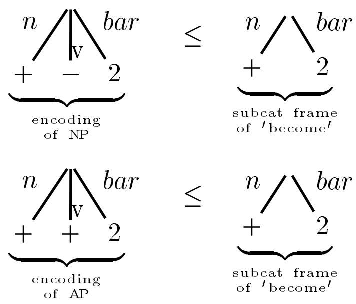
In this context, ordering constraints have been considered by Shieber [31] and Dörre [9] for the analysis of coordinated sentence constituents. We sketch an example from [31] after [9]. The assumed grammar is left implicit.

Pat has become $[_{NP}$ a republican$]$ and $[_{AP}$ very stingy$]$

In this example, the single complement 'a republican and very stingy' of the verb 'to become' consists of the noun phrase (*NP*) 'a republican' and the adjective phrase (*AP*) 'very stingy' coordinated by 'and'. To show this sentence to be well-formed (wrt. the assumed grammar) one must check that the complement meets the requirements of the verb (its *subcategorisation frame*); this is the case if both constituents 'a republican' and 'very stingy' of the coordination do. Following Shieber [31], this can be expressed with ordering constraints in a general rule for constituent coordination:

$$E \;::=\; C \text{ and } D \qquad \text{if } C \leq E \text{ and } D \leq E \tag{1}$$

Here, E denotes the structure associated with the coordination, and C and D those associated with its constituents, both of which are required to be smaller (*i. e.*, contain more information, be more specific) than E. This rule applies if we model the categories *NP* and *AP* and the subcategorisation frame of 'to become' by the feature trees depicted to the right and interpret $\leq$ as the information ordering on feature trees. Note that ordering constraints are essential here. In particular, the accordance between 'a republican' and 'very stingy' cannot be expressed by an equality between the associated structures since both have a different category, namely *NP* and *AP*.



Subsumption and Weak Subsumption. The original interpretation Shieber gave to the ordering constraints in his coordination rule (1) corresponds to the subsumption ordering [10] on the domain of feature algebras [33]. The satisfiability problem of subsumption constraints has been investigated by Dörre and was shown undecidable by Dörre and Rounds [10]. Dörre considered weak subsumption [9] as a weakening of subsumption, and shows the satisfiability problem of weak subsumption constraints to be decidable in time $O(n^5)$.

In [23] we have shown the satisfiability problem of weak subsumption constraints to be subsumed by the one of $FT_{\leq}$ constraints which improves the complexity bound to $O(n^3)$ (see Section 6 for a summary). For the sketched analysis of constituent coordination Dörre argues that weak subsumption often suffices, *i. e.*, that the additional expressivity of subsumption is not needed. Finally note that one can easily imagine an alternative encoding of the coordination phenomenon using sorts. We take this as an indication that the information ordering on sorted feature trees may be interesting in this context.

2.2 Program Analysis

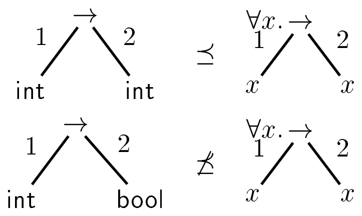
Many type inference problems for programming languages have a natural formulation as the satisfiability problem of certain tree constraint systems. There, the analysis of a program derives constraints for every program statement and requires the resulting set of constraints to be satisfiable. The solutions of this set of constraints correspond to the types of variables and statements in the program, and a program with an inconsistent analysis is rejected as ill-typed.

For instance, type inference for simply typed lambda calculus is equivalent to the satisfiability problem of equations over finite trees [37]. Type inference for a functional language with so-called polymorphic recursion [24] is equivalent to semi-unification [11, 16, 13], *i. e.*, the satisfiability problem of subsumption constraints over finite trees. Type inference problems for languages with an order on types ("subtyping") have been modelled as the satisfiability problem of classes of ordering constraints over trees (see [26, 17, 20, 18, 36] and others). In particular, record subtyping can be modelled by our information ordering when records are modelled as feature trees [35].

In contrast to this paper, most work on tree orderings for program analysis has focused on constructor trees (where at every node the constructor symbol's *arity* determines the number and labeling of departing edges). The choice of constructor trees does not affect the satisfiability problem in comparison with the one over feature trees, but appears to make the entailment problem much harder [23, 28]. Also, types are often modelled by finite trees. We consider infinite trees here since we are interested in analysing recursive data structures, but all results can be adapted to finite trees. More importantly, to model function types one usually considers an order which is contravariant for a binary constructor $\rightarrow$ (*i. e.*, $\tau_1 \leq \tau_1' \wedge \tau_2 \leq \tau_2'$ implies $\tau_1 \rightarrow \tau_2 \leq \tau_1' \rightarrow \tau_2'$), while in this paper the order is covariant for all constructors (*i. e.*: $\tau_1 \leq \tau_1' \wedge \tau_2 \leq \tau_2'$ implies $f(\tau_1\ \tau_2) \leq f(\tau_1'\ \tau_2')$).

Our original motivation for the investigation of ordering constraints stems from type inference problems for the concurrent constraint programming language Oz [30, 34]. Here, an analysis based on a fully covariant ordering seems appropriate since the mechanism for procedural abstraction in Oz is relational rather than functional (see also [21]). The goal of the analysis is to detect as many type errors as possible statically; the means is to approximate the type of program variables, *i. e.*, the set of values that variables may assume at run-time, and to reject a program which contains variables whose type is found to be empty. An expressive language to describe possible types is important and requires sorts at least for the description of basic types like integers, numbers, or booleans. (More expressive operators will be considered in future work.) Ordered trees can be used to approximate sets if a tree τ is taken to denote the set of all greater trees $\{\tau' \mid \tau \leq \tau'\}$. This observation has been used to transfer to the domain of set constraints techniques which have been developed for tree constraints [22, 8].

Another intuition for the use of $FT_{\leq}$(sort) constraints in program analysis is that the information ordering on feature trees approximates the instantiation relation between monomorphic and polymorphic types as in ML [19]. For example, the instances of the polymorphic type $\forall x.x{\rightarrow}x$ are obtained by replacing all occurrences of x in $x \rightarrow x$ with the same type, *e. g.*, $\mathsf{int}{\rightarrow}\mathsf{int}$ and $\mathsf{bool}{\rightarrow}\mathsf{bool}$. As mentioned above, the type inference problem for an ML-style language with polymor-



phic recursion [24] is equivalent to semi-unification [11, 16], and hence undecidable. The reduction of type inference to semi-unification can be seen as a program analysis in terms of subsumption constraints. The corresponding analysis based on $FT_{\leq}(\text{sort})$ constraints will accept more programs (intuitively, by allowing also $\mathsf{int}\to\mathsf{bool}$ as an "instance" of $\forall x.x\to x$), but it will be correct in that rejected programs are indeed ill-typed. This is, we obtain a polynomial plausibility check for a typed language with an otherwise undecidable type inference problem.

3 Ordering Constraints over Feature Trees

The constraint system $FT_{\leq}(\text{sort})$ is defined by a set of constraints and their interpretation over feature trees.

Preliminaries. We assume an infinite set of *variables* $\mathcal{V}$ (ranged over by x, y, z), and two sets of *features* $\mathcal{F}$ (ranged over by f, g) and *sorts* $\mathcal{S}$ (ranged over by a, b, c).[2] The cardinality of a finite set S is denoted by $|S|$. Finite sequences of variables $x_1 \ldots x_n$ are abbreviated by $\overline{x}$ if n is understood or does not matter. The number of variables in a sequence $\overline{x}$ is denoted by $\mathsf{len}(\overline{x})$, and the set of variables themselves is denoted by $\{\overline{x}\}$. Analogous notation is freely used for other syntactic categories.

Partially Ordered Sorts. We assume the sorts $\mathcal{S}$ to be partially ordered by $\sqsubseteq$, *i. e.*, $(\mathcal{S}, \sqsubseteq)$ to be a partially ordered set (poset), where we require that for every two sorts a_1 and a_2 there exists

1. a common lower bound (*i. e.*, an a with $a \sqsubseteq a_1$ and $a \sqsubseteq a_2$), and
2. a least upper bound, denoted by $a_1 \sqcup a_2$, if there exists some common upper bound at all.

Condition 1 is met, *e. g.*, if there is a minimal sort. Condition 2 amounts to saying that $(\mathcal{S}, \sqsubseteq)$ becomes an upper semi-lattice when extended by a maximal sort.

We shall freely identify $(\mathcal{S}, \sqsubseteq)$ and $\mathcal{S}$. Given a partially ordered set $(O, \sqsubseteq)$, we say that a subset $A \subseteq O$ is $\sqsubseteq$-*compatible* if an upper bound of A exists in O ($a \in O$: $a' \sqsubseteq a$ for all $a' \in A$) and $\sqsubseteq$-*incompatible* otherwise. Observe that condition 2 implies the following fact.

Observation 1. *Every finite compatible set of sorts has a least upper bound.*

Order-Sorted Feature Trees. A *path* p is a finite sequence of features. The *empty path* is denoted by ε and the free-monoid concatenation of paths p and p' as pp'; we have $\varepsilon p = p\varepsilon = p$. Given paths p and q, p' is called a *prefix of* p if $p = p'p''$ for some path p''. A *tree domain* is a non-empty prefix closed set of paths. An *order-sorted feature tree* (or simply a *feature tree*) τ is a pair (D, π)

[2] In contrast to [23], we choose two alphabets for features and sorts.

consisting of a tree domain D, and a *sorting* function $\pi : D \to \mathcal{S}$.[3] For a sorting function π (or any other function), we write its *domain* as $\mathcal{D}(\pi)$. We freely identify a function π with the set $\{(x, \pi(x)) \mid x \in \mathcal{D}(\pi)\}$. Given a feature tree τ, we write D_τ and π_τ for its two components.

Given a feature tree τ and $f \in D_\tau$, we denote with $\tau.p$ the subtree of τ at path p:

$$\tau.p \quad = \quad (\{p' \mid pp' \in D_\tau\}, \{(p', a) \mid (pp', a) \in \pi_\tau\})$$

A feature tree τ with sort a and subtrees $\tau_1, \ldots, \tau_n$ at exactly the features $f_1, \ldots, f_n$ is denoted by $a(f_1{:}\tau_1 \ldots f_n{:}\tau_n)$, or simply by a if $n = 0$. For instance, the binary branching tree from the introduction is written as wine(color:red year:1997). A feature tree is called *finite* if its tree domain is finite, and *infinite* otherwise. In this paper we consider possibly infinite feature trees. The set of all feature trees over $\mathcal{F}$ and $\mathcal{S}$ is denoted by $\mathcal{FT}_{(\mathcal{F},\mathcal{S})}$, or $\mathcal{FT}$ if $\mathcal{F}$ and $\mathcal{S}$ are understood.

Syntax. The syntax of $FT_\leq$(*sort*) *constraints* φ is defined as follows.

$$\varphi ::= \; x \leq y \quad | \quad a(x) \quad | \quad x[f]y \quad | \quad \sim\overline{x} \quad | \quad \varphi_1 \wedge \varphi_2$$

An $FT_\leq$(sort) constraint is a conjunction of *basic constraints*, namely *ordering constraints* $x \leq y$, *sorting constraints* $a(x)$, *selection constraints* $x[f]y$, and *compatibility constraints* $\sim\overline{x}$. Note that $FT_\leq$(sort) constraints are just $FT_\leq$ constraints [23] extended by n-ary compatibility constraints $\sim\overline{x}$ instead of just binary ones. For brevity, we refer to $FT_\leq$(sort) constraints also just as as *constraints*. The variables occurring in a constraint φ are written as $\mathcal{V}(\varphi)$. The *size of* φ is the number of symbols in φ. As is common, we identify a constraint with the set of basic constraints in it, and write $\varphi \subseteq \psi$ or $x[f]y \in \varphi$ accordingly.

Semantics. $FT_\leq$(sort) constraints are interpreted in the structure defined by the domain $\mathcal{FT}$ of order-sorted feature trees; the signature containing the binary relation symbol $\leq$ and the n-ary relation symbol $\sim$, a unary relation symbol $a()$ per sort $a \in \mathcal{S}$, and a binary relation symbol $[f]$ per feature $f \in \mathcal{F}$; and an interpretation of the relation symbols over $\mathcal{FT}$ as follows. This structure is called $\mathcal{FT}_{(\mathcal{F},\mathcal{S})}$ or $\mathcal{FT}$ after its domain without raising confusion.

$$\begin{array}{lll}
\tau_1[f]\tau_2 & \text{iff} & \tau_1.f = \tau_2 \\
a(\tau) & \text{iff} & a \sqsubseteq \pi_\tau(\varepsilon) \\
\tau_1 \leq \tau_1 & \text{iff} & D_{\tau_1} \subseteq D_{\tau_2} \text{ and } \pi_{\tau_1} \sqsubseteq \pi_{\tau_2} \\
\sim (\tau_1 \ldots \tau_n) & \text{iff} & \text{exists } \tau \text{ such that } \tau_1 \leq \tau, \; \ldots, \; \tau_n \leq \tau
\end{array}$$

The relation $\pi_1 \sqsubseteq \pi_2$ is defined by lifting the partial order on sorts to sortings.

$$\pi_1 \sqsubseteq \pi_2 \quad \text{iff} \quad \text{for all } p \in \mathcal{D}(\pi_1) : \pi_1(p) \sqsubseteq \pi_2(p)$$

Note that $\leq$ is a partial order on $\mathcal{FT}$ and that that $\sim\overline{\tau}$ holds iff $\{\overline{\tau}\}$ is $\leq$-compatible. Note also that $\sim(x_1 \ldots x_n)$ is equivalent to $\exists z\,(x_1 \leq z \wedge \ldots \wedge x_n \leq z)$.

[3] Unlike in [23], we assume a total function here.

(F1)	$x \le x$ and $x \le y \wedge y \le z \rightarrow x \le z$
(F2)	$y[f]y' \wedge y \le x \wedge x[f]x' \rightarrow y' \le x'$
(F3)	$\sim\overline{x} \rightarrow \sim\overline{y} \qquad$ if $\{\overline{x}\} = \{\overline{y}\}$
(F4)	$y \le x \rightarrow \sim xy$
(F5)	$y \le x \wedge \sim x\overline{z} \rightarrow \sim xy\overline{z}$
(F6)	$\overline{x}[f]\overline{y} \wedge \sim\overline{x}\overline{z} \rightarrow \sim\overline{y}$
(F7)	$\sim(\overline{x}\overline{y}) \wedge \varphi \rightarrow \mathit{false} \qquad$ if $\mathsf{clash}_\varphi(\overline{x})$

Table 1. Satisfiability of $FT_{\le}$(sort) Constraints

A mapping σ from variables into $\mathcal{FT}$ is called a *solution* of a constraint φ if σ satisfies all constraints in φ; *i. e.*, $\sigma(x)[f]\sigma(y)$, $a(\sigma(x))$, $\sigma(x) \le \sigma(y)$, and $\sim(\sigma(x_1)\ldots\sigma(x_n))$ for constraints in φ of the form $x[f]y$, $a(x)$, $x \le y$, or $\sim(x_1 \ldots x_n)$, respectively.

Example 1. Due to the requirement that $(\mathcal{S}, \sqsubseteq)$ have lower bounds, the constraint $x \le y \wedge x \le z \wedge a(y) \wedge b(z)$ is satisfiable for all sorts a, b. *E. g.*, every mapping σ with $\sigma(y) = a$, $\sigma(z) = b$, and $\sigma(x) = c$ where $c \sqsubseteq a$ and $c \sqsubseteq b$ is a solution. $\circ$

3.1 Satisfiability

Table 1 defines an infinite set F of axioms as the instances of axiom schemes $F1$ through $F7$.

Read as logical implications, the axioms F characterise satisfiability of $FT_{\le}$(sort) constraints. Axiom scheme $F1$ makes $\le$ reflexive and transitive. $F2$ states that $\le$ is monotonic wrt. selection. $F3$ makes constraints $\sim\overline{x}$ equivalent up to associativity, commutativity, and idempotence of concatenation, *e. g.*, $\sim(yxy)$ and $\sim(xy)$. $F4$ introduces compatibility constraints for every variable with each of its lower bounds and, recursively, between all its lower bounds at once. $F5$ is a quasi-transitivity axiom for compatibility constraints. For illustration note that

$$\begin{array}{rcl} y \le x \wedge \sim(xz) & \leftrightarrow & y \le x \wedge \exists u(x \le u \wedge z \le u) \\ & \rightarrow & \exists u(y \le u \wedge y \le u \wedge z \le u) \\ & \leftrightarrow & \sim(xyz) \end{array}$$

is valid. $F6$ states that compatible trees have pointwise compatible subtrees. We use the abbreviation $\overline{x}[f]\overline{y}$ for $x_1[f]y_1 \wedge \ldots \wedge x_n[f]y_n$, implicitly assuming variables such that $\overline{x} = x_1 \ldots x_n$ and $\overline{y} = y_1 \ldots y_n$. Axiom scheme $F7$ may derive logical inconsistency *false* from a clash. We say that a constraint φ contains a *clash* wrt. a sequence $\overline{x}$ of variables, if $\mathsf{clash}_\varphi(\overline{x})$ holds, where $\mathsf{clash}_\varphi(\overline{x})$ is defined by

$$\mathsf{clash}_\varphi(x_1 \dots x_n) \quad \text{iff} \quad \{a \mid a(x_i) \,\mathsf{in}\, \varphi \text{ f. s. } i, 1 \leq i \leq n\} \text{ is } \sqsubseteq\text{-incompatible}$$

Example 2. Assume a and b to be incompatible and consider the inconsistent constraint $a(y) \wedge b(z) \wedge x[f]y \wedge x[f]z$. We can derive *false* from this constraint as follows: Obtain $x \leq x$ from $F1$, $\sim(xx)$ from $F4$, $\sim(yz)$ from $F6$, and finally *false* from $F7$. ∘

Proposition 2. *The structure $\mathcal{FT}$ is a model of the axioms in F.*

Proof. By a routine check of the axioms. □

These axioms coincide with the ones given for $FT_{\leq}$ [23] when compatibility constraints are restricted to binary form $\sim(xy)$. (See also Section 5 for a comparison.) In our more general setting with ordered sorts, a restriction to binary compatibility constraints makes the axioms incomplete.

Example 3. Assume as sorts the non-empty subsets of $\{a, b, c\}$ ordered by inverse set inclusion (*i. e.*, $(\mathcal{S}, \sqsubseteq) = (\mathcal{P}(\{a, b, c\}) \setminus \emptyset, \supseteq)$), and consider the non-satisfiable constraint φ

$$\{a, b\}(x) \wedge \{b, c\}(x') \wedge \{a, c\}(x'') \wedge x \leq y \wedge x' \leq y \wedge x'' \leq y$$

The axioms F, restricted to binary compatibility constraints, and φ do not imply *false*. ∘

Fixed-Point Algorithm. The axioms also induce an *algorithm* F which repeatedly, and until a fixed-point is reached, adds to a constraint φ all constraints which are logically implied wrt. $F1 - F6$, or replaces φ with *false* if $F7$ applies. (Recall that *false* is not a constraint.) In computing fixed points, we always make sure to identify $\sim\overline{x}$ and $\sim\overline{y}$ whenever $\{\overline{x}\} = \{\overline{y}\}$. A constraint φ is called *F-closed wrt. a set of variables V* if $\psi \subseteq \varphi$ for all its F-consequences ψ with $\mathcal{V}(\psi) \subseteq V$; or simply *$F$-closed* if $V = \mathcal{V}$. The F-closure (wrt. V) of a constraint φ is defined accordingly. The F-closure of a constraint is infinite due to reflexivity of ordering and compatibility, and since $\mathcal{V}$ is infinite. In contrast, the F-closure of a constraint φ *wrt.* $\mathcal{V}(\varphi)$ is finite.

Proposition 3. *For any constraint φ, the computation of its F-closure wrt. $\mathcal{V}(\varphi)$ terminates.*

Proof. If φ has size n then $|\mathcal{V}(\varphi)| \leq n$ and there are at most 2^n sequences $\overline{x}$ for distinct sets $\{\overline{x}\} \subseteq \mathcal{V}(\varphi)$. Note that no sort constraint $a(x)$ is ever added by the algorithm. Hence, there are at most $n^2 + 2^n + n$ constraints of the form $x \leq y$, $\sim\overline{x}$, and $a(x)$, respectively, in the closure of φ wrt. $\mathcal{V}(\varphi)$. □

Proposition 4. *Every F-closed constraint is satisfiable.*

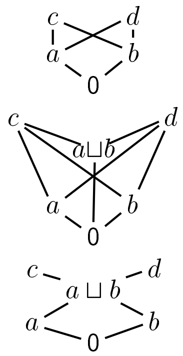
The proof is contained in Section A of the Appendix. An implication of Proposition 4 is that algorithm F can be understood as a reduction of the satisfiability problem of $FT_{\leq}$(sort)-constraints to the satisfiability problem of only ordering, compatibility, and labelling constraints ($x \leq y$, $\sim\overline{x}$, and $a(x)$; see also Proposition 17). To see this note that for every constraint there is an equivalent $F1-F6$-closed constraint, and that an $F1-F6$-closed constraint is F-closed if and only if $F7$ is non-applicable. Applicability of $F7$, however, can be decided without considering any selection constraint.

For Proposition 4 to hold, Observation 1 is crucial which guarantees least upper bounds for finite compatible sets of sorts.

Example 4. For further illustration, consider the following constraint φ:

$$a(y) \wedge y \leq x \wedge b(z) \wedge z \leq x \wedge x \leq u \wedge c(u) \wedge x \leq v \wedge d(v)$$

and let φ' be the $F1-F6$-closure of φ. To verify that φ' is also $F7$-closed it suffices to consider the largest compatibility constraints $\sim(x_1 \ldots x_n)$ in φ' such that for x_1 through x_n there are sorting constraints in φ'; these are $\sim(yzu)$ and $\sim(yzv)$. Now consider the three posets depicted to the right, only the third of which respects condition 2. With respect to all three posets, $\{a, b, c\}$ and $\{a, b, d\}$ are compatible such that neither $\mathsf{clash}_{\varphi'}(yzu)$ nor $\mathsf{clash}_{\varphi'}(yzv)$ holds. However, φ is not satisfiable wrt. the first two posets since both lack a sort which is both smaller than c and d and larger than a and b: *i. e.*, there is no valid sorting for x. The third poset however allows, *e. g.*, any solution σ where $\sigma(u) = c$, and $\sigma(v) = d$, $\sigma(x) = a \sqcup b$, $\sigma(y) = a$, and $\sigma(z) = b$. ◦



Theorem 5 Decidability. *The satisfiability of $FT_{\leq}$(sort) constraints is decidable.*

Proof. By F-closure of φ wrt. $\mathcal{V}(\varphi)$. Termination follows from Proposition 3. By Proposition 2, φ is non-satisfiable if algorithm F terminates with *false*. If F terminates with a constraint ψ (*i. e.*, not with *false*) then, by Proposition 2, φ and ψ are equivalent. By Proposition 4, ψ is satisfiable, and hence φ is. □

Theorem 6 Canonicity. *If an $FT_{\leq}$(sort)-constraint φ is satisfiable over some model of the axioms in F then φ is satisfiable over $\mathcal{FT}$.*

Proof. Assume a model $\mathcal{A}$ of the axioms in F. If a constraint φ is satisfiable in $\mathcal{A}$ then it is equivalent to an F-closed constraint φ' (and not *false*). By Proposition 4 this implies that φ' is satisfiable and hence φ is. □

4 Polynomial Complexity

The termination argument in the proof of Proposition 3 gives an exponential upper bound, due to the *a priori* exponential number of compatibility constraints over a fixed set of variables. Fortunately, it is often not necessary to compute all of these. Rather, depending on the poset of sorts, one can give a polynomial upper bound on the number of compatibility constraints necessary to decide satisfiability, and organise the algorithm F such that it has polynomial complexity.

4.1 The Coherence Property

In this section we give a criterion for polynomial worst-case time complexity based on the (generalised) *coherence*-property [5] of partial orders. The following Section 4.2 applies the criterion to special cases.

Definition 7 Coherence. A poset $(O, \sqsubseteq)$ has the *coherence property* (or: is *coherent*) if for all $A \subseteq O$ the following holds.

If $\{a_1, a_2\}$ is $\sqsubseteq$-compatible for all $a_1, a_2 \in A$, then A is $\sqsubseteq$-compatible.

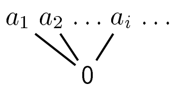
Generally, $(O, \sqsubseteq)$ has the *coherence property of kth order* (or: is *k-coherent*) if for all subsets $A \subseteq O$ the following holds.

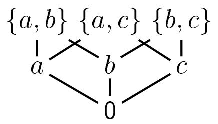
If A' is $\sqsubseteq$-compatible for all $A' \subseteq A$ with $|A'| \leq k$, then A is $\sqsubseteq$-compatible. $\lhd$

Note that k-coherence trivially implies $k{+}1$-coherence.

In the context of type inference for functional languages with subtyping (see [20, 18, 36, 12] and references therein), the related *two-disk* or *Helly-property* of posets [25] has been studied [7, 29]. While the Helly-property implies coherence, a property corresponding to k-coherence has not been considered before.

Based on k-coherence one can give a polynomial bound on the number of sets of variables whose compatibility is essential for satisfiability (Theorem 9).





It remains for every particular poset $(\mathcal{S}, \sqsubseteq)$ to determine the smallest k such that $(\mathcal{S}, \sqsubseteq)$ is k-coherent. For illustration, consider the two depicted posets and observe that the left one is coherent, and that the second one is 3-coherent but not 2-coherent. Section 4.2 contains further examples; in particular note Corollary 13 and Proposition 14. Call a constraint $F^{(k)}$*-closed* if it is $F1$, $F2$, and $F7$-closed, and if it contains all compatibility constraints $\sim\bar{x}$ of size $\leq k$ whenever $\{\bar{x}\} \subseteq \{\bar{x}'\}$ and axioms $F4$ through $F6$ imply $\sim\bar{x}'$.

Proposition 8. *If $(\mathcal{S}, \sqsubseteq)$ is k-coherent, then an $F^{(k)}$-closed constraint is satisfiable.*

Proof. The proof refines the one of Proposition 4 (see Appendix A). $\square$

Theorem 9 Decidability and Complexity. *Let the poset of sorts $(\mathcal{S}, \sqsubseteq)$ be k-coherent such that compatibility of finite $A \subseteq \mathcal{S}$ can be tested in time $|A|$. Then the satisfiability of constraints over $\mathcal{FT}_{(\mathcal{F},\mathcal{S})}$ is decidable in time $O(n^{2k-1})$ and with space $O(n^k)$ where n is the size of φ.*

Proof. Assume a constraint φ with size n, *i. e.*, containing at most n variables, sorts, and features. Due to k-coherence, it suffices to check compatibility (*i. e.*, to generate constraints $\sim\overline{x}$) only for variable sequences $\overline{x}$ of at most k distinct variables, of which there are $O(n^k)$ over $\mathcal{V}(\varphi)$.

We organise the algorithm as a rewriting on pairs (P, S) where P and S are called *pool* and *store*, respectively. P is a multiset of basic constraints and S is a data structure for basic constraints described below. In every step, a basic constraint φ_0 is removed from B and added to S. Further, the basic constraints are computed (compatibility constraints restricted to be at most k-ary) which follow from φ_0 and S in one step of reasoning (*modus ponens*) from the axioms $F1 - F6$; they are added to $B \setminus \{\varphi_0\}$ if they are not yet contained in S. The algorithm terminates if B becomes empty or if φ_0 and S imply *false* wrt. $F7$.

The store is a table which contains per variable x a set of sorts (representing sorting constraints $a(x)$) and a partial mapping from features to variables (representing selection constraints $x[f]y$), and which also contains ordering constraints $x \leq y$ and compatibility constraints of arity $\leq k$. This table has size at most $O(n^k)$. We can implement the store such that it takes (at most) time $O(n)$ to (i) add a basic constraint, (ii) test containment of a basic constraint, (iii) check compatibility of the sorts of x and its upper bounds, and (iv) retrieve the new consequences of φ_0 and S. Notably the last operation is the most costly one.

The basic loop of this algorithm runs at most $O(n^k)$ times, and every iteration takes (amortised) $O(n^{k-1})$ time. Hence, overall time complexity is $O(n^{2k-1})$. This complexity argument can be extended, along the lines of [22], to the "on-line" case where basic constraints are dynamically added and the needed variables, sorts, and features are not known in advance. $\square$

Corollary 10. *The satisfiability of $FT_{\leq}(sort)$ constraints over $\mathcal{FT}_{(\mathcal{F},\mathcal{S})}$ is decidable in time $O(n^3)$, if $\mathcal{S}$ is ordered by $\sqsubseteq = \{(a, a), (\mathsf{0}, a) \mid a \in \mathcal{S}\}$ where $\mathsf{0} \notin \mathcal{S}$.*

Proof. Follows from Theorem 9 and Corollary 13. $\square$

Finally note, that there is room left for improvements in a real implementation. For instance, $\sim\overline{y}$ is in an F-closed constraint whenever $\sim x\overline{y}$ is (Lemma **??**). An explicit representation of the redundant constraint $\sim\overline{y}$ can often be avoided.

4.2 Examples

Proposition 11. *The disjoint sum of two k-coherent posets is k-coherent if $k \geq 2$.*

Proof. Assume $L_i = (\mathcal{S}_i, \sqsubseteq_i)$ for $i = 1, 2$ with $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$ and define $L = (\mathcal{S}, \sqsubseteq)$ by $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$ and $\sqsubseteq = \sqsubseteq_1 \cup \sqsubseteq_2$. Suppose $A \subseteq \mathcal{S}$ where A is $\sqsubseteq$-incompatible. If $A \subseteq \mathcal{S}_i$ for $i = 1$ or $i = 2$, then, by k-coherence of L_i, there exists an $\sqsubseteq$-incompatible $A' \subseteq A$ with cardinality $|A'| = k$. Otherwise, there exist $a_1, a_2 \in A'$ such that $a_1 \in \mathcal{S}_1$ and $a_2 \in \mathcal{S}_2$, and hence $\{a_1, a_2\}$ is $\sqsubseteq$-incompatible. $\square$

Proposition 12. *Assume a k-coherent poset $(\mathcal{S}, \sqsubseteq)$ and an element $a \notin \mathcal{S}$. Then $(\mathcal{S} \cup \{a\}, \sqsubseteq \cup \{a\} \times \mathcal{S})$ is k-coherent.*

Proof. Assume an incompatible set A. If $a \in A$, then $A \setminus \{a\}$ is also incompatible and the claim follows from k-coherence of $(\mathcal{S}, \sqsubseteq)$. Case $a \notin A$ is immediate. $\square$

Corollary 13 Trees. *The disjoint union of trees is a coherent poset.*

Proof. Follows from Propositions 11 and 12 with the trivial fact that singleton sets are coherent. $\square$

Proposition 14. *If $\mathcal{S}$ contains at most k $\sqsubseteq$-maximal sorts, and every $a \in \mathcal{S}$ is smaller than one of those, then $(\mathcal{S}, \sqsubseteq)$ is k-coherent.*

Proof. Let $\{a_1, \ldots, a_k\}$ contain all maximal elements in $(\mathcal{S}, \sqsubseteq)$, and consider an $\sqsubseteq$-incompatible $S \subseteq \mathcal{S}$. Then for every i, $1 \leq i \leq k$, there exists $s_i \in S$ such that $s_i \not\sqsubseteq a_i$. Hence, $\{s_1, \ldots, s_k\}$ is $\sqsubseteq$-incompatible. The claim follows from contraposition. $\square$

Corollary 15. *For all finite A, the poset $(\mathcal{P}(A) \backslash \emptyset, \supseteq)$ of non-empty subsets of A ordered by inverse set inclusion is $|A|$-coherent.*

Proof. Simply observe that the maximal sorts of $(\mathcal{P}(A) \backslash \emptyset, \supseteq)$ are the singletons over A and that every non-empty set is smaller than some singleton. $\square$

Corollary 16. *A finite poset $(\mathcal{S}, \sqsubseteq)$ is $|\mathcal{S}|$-coherent.*

Proposition 17. *If $(\mathcal{S}, \sqsubseteq)$ is k-coherent, then $(\mathcal{FT}_{(\mathcal{F},\mathcal{S})}, \leq)$ is also k-coherent.*

Proof. Induction over paths. $\square$

Note that many infinite posets are not k-coherent for any k.

Observation 18 Failure of Coherence. *If S is infinite, then the poset $(\mathcal{P}(S) \backslash \emptyset, \supseteq)$ is not k-coherent for any k.*

Proof. The counter example for 2-coherence is shown preceding to Theorem 9. For $k = 3$, consider the four triples $\{a, b, c\}, \{a, b, d\}, \{a, c, d\}$, and $\{b, c, d\}$, and note that their intersection is empty, while any three of them has a singleton (*i. e.*, non-empty) intersection. The construction principle for the counter example against k-coherence for arbitrary k should be clear. □

5 The Constraint System $FT_{\leq}$

The constraint system $FT_{\leq}$ is defined in [23] over a domain $\mathcal{F}_0$ of feature trees which slightly differs from $\mathcal{FT}$ as considered here. This section shows that $FT_{\leq}$(sort) is a proper generalisation of $FT_{\leq}$ and obtains the complexity result for the satisfiability test in [23] as Corollary 20 from Theorem 9.

In $FT_{\leq}$, the sorting π_τ for $\tau \in \mathcal{F}_0$ may be a *partial* function on D_τ (*i. e.*, nodes mey be unsorted), and sorts $\mathcal{S}$ are only trivially ordered by $\sqsubseteq = \{(a, a) \mid a \in \mathcal{S}\}$. As already mentioned, $FT_{\leq}$ constraints are $FT_{\leq}$(sort) constraints where only binary compatibility constraints are allowed. The permission of unsorted nodes in $FT_{\leq}$ corresponds to the requirement of a minimal sort $\mathsf{0}$ in $FT_{\leq}$(sort), *i. e.*, assuming sorts $\mathcal{S} \cup \{\mathsf{0}\}$, $\mathsf{0} \notin \mathcal{S}$, ordered by $\sqsubseteq \cup \{\mathsf{0}\} \times \mathcal{S}$. We must be careful since $\mathsf{0}(x)$ is not an $FT_{\leq}$ constraint over sorts $\mathcal{S}$, but since $\mathsf{0}(x)$ is trivially valid we can simply allow $\mathsf{0}(x)$ as additional $FT_{\leq}$ constraint interpreted as *true*.

Proposition 19. *Let $\mathcal{S}$ be partially ordered by $\sqsubseteq = \{(a, a), (\mathsf{0}, a) \mid a \in \mathcal{S}\}$ and define $\mathcal{S}' = \mathcal{S} \setminus \{\mathsf{0}\}$. Then $FT_{\leq}$ constraints over $\mathcal{S}'$ (with $\mathsf{0}(x)$ interpreted as true for all x) and $FT_{\leq}$(sort) constraints over $\mathcal{S}$ have the same first order theory.* □

Proof. The solutions over both domains for first-order formulas over both classes of constraints correspond bijectively to each other where unsorted nodes in $\mathcal{F}$ correspond to nodes $\mathsf{0}$-sorted with in $\mathcal{FT}$. □

Corollary 20. *The satisfiability of an $FT_{\leq}$ constraint φ is decidable in time $O(n^3)$ where n is the size of φ.*

Proof. Follows from Corollary 10 and Proposition 19. □

6 Weak Subsumption Constraints

In this section we recall the result of [23] that the satisfiability problem of $FT_{\leq}$(sort) constraints subsumes the satisfiability problem of weak subsumption constraints as proposed by Dörre [9]. This can be concluded from Proposition 19 combined with Theorem 22 below. Weak subsumption constraints have the same syntax as $FT_{\leq}$ constraints (see previous section), apart from the fact that Dörre does not consider compatibility constraints $\sim\overline{x}$. We allow them here for technical simplicity. Their semantics is given by interpretation over arbitrary feature algebras with their induced weak subsumption ordering as defined below.

A *feature algebra* $\mathcal{A}$ over features $\mathcal{F}$ and *constants* $\mathcal{S}$ consists of a set $\mathsf{dom}^{\mathcal{A}}$ called the *domain* of $\mathcal{A}$, a unary relation $a()^{\mathcal{A}}$ on $\mathsf{dom}^{\mathcal{A}}$ for every constant $a \in \mathcal{S}$, and a binary relation $[f]^{\mathcal{A}}$ on $\mathsf{dom}^{\mathcal{A}}$ for every feature $f \in \mathcal{F}$, which satisfy the following properties for all $\alpha, \alpha', \alpha'' \in \mathsf{dom}^{\mathcal{A}}$, constants $a, a_1, a_2 \in \mathcal{S}$, and features $f \in \mathcal{F}$:[4]

1. if $\alpha[f]^{\mathcal{A}}\alpha'$ and $\alpha[f]^{\mathcal{A}}\alpha''$ then $\alpha' = \alpha''$
2. if $a_1(\alpha)^{\mathcal{A}}$ and $a_2(\alpha)^{\mathcal{A}}$ then $a_1 = a_2$

Lemma 21. *The structure $\mathcal{F}_0$ is a feature algebra.*

Here it comes in handy that we, unlike [23], use two alphabets for the definition of feature trees. Note that $\mathcal{FT}$ is, in general, not a feature algebra, since $a(x) \wedge b(x)$ may be satisfiable for $a \neq b$, for instance if $a \sqsubseteq b$.

Given a feature algebra $\mathcal{A}$, a *simulation for* $\mathcal{A}$ is a binary relation Δ on $\mathsf{dom}^{\mathcal{A}}$ with the following properties for all elements α_1, α_2, α_1', α_2' of $\mathcal{A}$'s domain:

1. if $\alpha_1 \Delta \alpha_2,\ a_1(\alpha_1)^{\mathcal{A}},$ and $a_2(\alpha_2)^{\mathcal{A}}$ then $a_1 = a_2$
2. if $\alpha_1 \Delta \alpha_2,\ \alpha_1[d]^{\mathcal{A}}\alpha_1',$ and $\alpha_2[d]^{\mathcal{A}}\alpha_2'$ then $\alpha_1' \Delta \alpha_2'$

The *weak subsumption ordering* $\underset{\sim}{\sqsubset}$ of $\mathcal{A}$ is the greatest simulation for $\mathcal{A}$. The weak subsumption ordering on $\mathcal{A}$ induces a compatibility relation $\sim^{\mathcal{A}}$ as follows:

$$\sim^{\mathcal{A}}(\alpha_1 \ldots \alpha_n) \ \text{ iff } \ \{\alpha_1, \ldots, \alpha_n\} \text{ is } \underset{\sim}{\sqsubset}\text{-compatible}$$

Every feature algebra $\mathcal{A}$ induces a structure with the same signature as $FT_{\leq}(\text{sort})$ where $\leq$ is interpreted as the weak subsumption ordering $\underset{\sim}{\sqsubset}$, $\sim$ as $\sim^{\mathcal{A}}$, $a()$ as $a()^{\mathcal{A}}$, and $[f]$ as $[f]^{\mathcal{A}}$. Every such structure is a model of the axioms in F.

Theorem 22 [23]. *A weak subsumption constraint φ is satisfiable over the structure $\mathcal{F}_0$ if and only if φ is satisfiable over the structure induced by some feature algebra $\mathcal{A}$.*

Proof. If φ is satisfiable over some feature algebra $\mathcal{A}$, then it is satisfiable over the structure induced by $\mathcal{A}$. Since this structure is a model of the axioms in F, Theorem 6 implies that φ is satisfiable over $\mathcal{F}_0$. If φ is satisfiable over $\mathcal{F}_0$ then, by Lemma 21, is satisfiable over some feature algebra. □

[4] A third property is usually assumed which prohibits the same α to satisfy both $a(\alpha)^{\mathcal{A}}$ and $\alpha[f]^{\mathcal{A}}\alpha'$ for some $a \in \mathcal{S}$, $f \in \mathcal{F}$, and α'. This is discussed in [23].

Acknowledgments. I would like to thank Joachim Niehren for detailed comments and discussion, as well as Werner Nutt, Andreas Podelski and Volker Priebe for valuable comments on the topic of this paper. Joachim Walser also provided helpful feedback. The research reported in this paper has been supported by the the Esprit Working Group CCL II (EP 22457) and the Deutsche Forschungsgemeinschaft through the Graduiertenkolleg Kognitionswissenschaft and the SFB 378 at the Universität des Saarlandes.

References

1. H. Aït-Kaci and A. Podelski. Towards a Meaning of Life. *The Journal of Logic Programming*, 16(3 and 4):195–234, July, Aug. 1993.
2. H. Aït-Kaci, A. Podelski, and G. Smolka. A Feature-based Constraint System for Logic Programming with Entailment. *Theoretical Computer Science*, 122(1–2):263–283, Jan. 1994.
3. R. Backofen. A Complete Axiomatization of a Theory with Feature and Arity Constraints. *The Journal of Logic Programming*, 1995. Special Issue on Computational Linguistics and Logic Programming.
4. R. Backofen and G. Smolka. A Complete and Recursive Feature Theory. *Theoretical Computer Science*, 146(1–2):243–268, July 1995.
5. H. P. Barendregt. *The Type Free Lambda Calculus.* In Barwise [6], 1977.
6. J. Barwise, editor. *Handbook of Mathematical Logic.* Number 90 in Studies in Logic. North–Holland, Amsterdam, Holland, 1977.
7. M. Benke. Efficient Type Reconstruction in the Presence of Inheritance. In *18^{th} International Symposium on Mathematical Foundations of Computer Science*, LNCS 711, pp. 272–280, Gdansk, 1993. Springer-Verlag, Berlin, Germany.
8. W. Charatonik and A. Podelski. The Independence Property of a Class of Set Constraints. In E. C. Freuder, editor, *International Conference on Principles and Practice of Constraint Programming*, LNCS 1118. Springer-Verlag, 1996.
9. J. Dörre. Feature-Logic with Weak Subsumption Constraints. In *Constraints, Languages, and Computation*, pp. 187–203. Academic Press, 1994.
10. J. Dörre and W. C. Rounds. On Subsumption and Semiunification in Feature Algebras. In *5^{th} Symp. on Logic in Computer Science*, pp. 300–310. IEEE Computer Society Press, 1990.
11. F. Henglein. Type Inference with Polymorphic Recursion. *ACM Transactions on Programming Languages and Systems*, pp. 253–289, 1993.
12. M. Hoang and J. C. Mitchell. Lower Bounds on Type Inference with Subtypes. In *22^{nd} Symp. on Principles of Programming Languages*, pp. 176–185. Jan. 1995.
13. S. Jahama and A. J. Kfoury. Type Reconstruction in the Presence of Polymorphic Recursion and Recursive Types. Technical Report bu-cs 93-019, Boston University, Dec. 1993.
14. R. M. Kaplan and J. Bresnan. Lexical-Functional Grammar: A Formal System for Grammatical Representation. pp. 173–381. The MIT Press, 1982.
15. M. Kay. Functional Grammar. In C. Chiarello et al., editor, *Proc. of the 5^{th} Annual Meeting of the Berkeley Linguistics Society*, pp. 142–158, 1979.
16. A. J. Kfoury, J. Tiuryn, and Urzycyn. Type Inference in the Presence of Polymorphic Recursion. *ACM Transactions on Programming Languages and Systems*, pp. 290–311, 1993.

17. D. Kozen, J. Palsberg, and M. I. Schwartzbach. Efficient Inference of Partial Types. *Journal of Computer and System Sciences*, 49(2):306–324, 1994.
18. P. Lincoln and J. Mitchell. Algorithmic Aspects of Type Inference with Subsorts. In *19th Symp. on Principles of Programming Languages*, pp. 293–304. Jan. 1992.
19. R. Milner. A Theory of Type Polymorphism in Programming. *Journal of Computer and System Science*, 17:348–375, 1978.
20. J. C. Mitchell. Coercion and Type Inference (summary). In *11th ACM Symp. on Principles of Programming Languages*, pp. 175–185. ACM, Jan. 1984.
21. M. Müller. Polymorphic Types for Concurrent Constraints. Available at `http://www.ps.uni-sb.de/` in `~mmueller/papers/ptcc.ps.gz`, 1996.
22. M. Müller, J. Niehren, and A. Podelski. Inclusion Constraints over Non-Empty Sets of Trees. In *Tapsoft: Theory and Practice of Software Development*, LNCS. Springer-Verlag, 1997. To appear.
23. M. Müller, J. Niehren, and A. Podelski. Ordering Constraints over Feature Trees. Available at `http://www.ps.uni-sb.de/` in `~mmueller/papers/ftsub.ps.gz` Submitted for publication. 1997.
24. A. Mycroft. Polymorphic Type Schemes and Recursive Definitions. In *International Symposium on Programming*, LNCS 167. Springer-Verlag, 1984.
25. P. Nevermann and I. Rival. Holes in Ordered Sets. *Graphs and Combinatorics*, 1:339–350, 1985.
26. J. Palsberg. Efficient Inference of Object Types. In *9th IEEE Symposium on Logic in Computer Science*, pp. 186–185. IEEE Computer Society Press, 1994.
27. C. Pollard and I. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Cambridge University Press, Cambridge, England, 1994.
28. F. Pottier. Simplifying Subtyping Constraints. *ACM SIGPLAN International Conference on Functional Programming*, pp. 122–133. ACM, 1996.
29. V. Pratt and J. Tiuryn. Satisfiability of Inequalities in a Poset. *Studia Logics*. To appear.
30. Programming Systems Lab, Universität des Saarlandes. The Oz Programming System, 1996. `http://www.ps.uni-sb.de/www/oz/`.
31. S. Shieber. *An Introduction to Unification-based Approaches to Grammar*. CSLI Lecture Notes No. 4. Center for the Study of Language and Information, 1986.
32. S. Shieber, H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson. The formalism and implementation of PATR-II. In J. Bresnan, editor, *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, California, 1983.
33. G. Smolka. Feature Constraint Logics for Unification Grammars. *The Journal of Logic Programming*, 12:51–87, 1992.
34. G. Smolka. The Oz Programming Model. In J. van Leeuwen, editor, *Computer Science Today*, LNCS 1000, pp. 324–343. Springer-Verlag, 1995.
35. G. Smolka and R. Treinen. Records for Logic Programming. *The Journal of Logic Programming*, 18(3):229–258, Apr. 1994.
36. J. Tiuryn. Subtype Inequalities. In *7th IEEE Symposium on Logic in Computer Science*, pp. 308–315, 1992.
37. M. Wand. A Simple Algorithm and Proof for Type Inference. *Fundamenta Informaticae*, 10:115–122, 1987.

A Completeness of the Satisfiability Test

This section gives the proof of Proposition 4 which states that every F-closed constraint is satisfiable, and its refinement towards Proposition 8 which states that an $F^{(k)}$-closed constraint is satisfiable if the sorts form a k-coherent poset. The proof setup closely follows the on of [23].

First we observe the following simple fact which allows us to assume, w.l.o.g., that φ contains a sort constraint for all variables in φ.

Lemma 23. *Let φ be an F-closed constraint. Let a be a common lower bound of all sorts occurring in φ, or arbitrary if no sort occurs in φ, and define φ' as $\varphi \wedge \bigwedge\{a(x) \mid x \in \mathcal{V}(\varphi)\}$. Then φ' is F-closed.*

Proof. If sorts occur in φ at all, then condition 1 on sorts implies that such an a exists. We check that $F7$ is not applicable to φ' if it is not applicable to φ. $\square$

Definition 24 Path Reachability. For all paths p and constraints φ, we define a binary relation $\overset{\varphi}{\leadsto}_p$ between variables, where $x \overset{\varphi}{\leadsto}_p y$ reads as "y is reachable from x over path p in φ":

$$\begin{array}{lll} x \overset{\varphi}{\leadsto}_\varepsilon y & \text{if} & y \leq x \text{ in } \varphi \\ x \overset{\varphi}{\leadsto}_f y & \text{if} & x[f]y \text{ in } \varphi, \\ x \overset{\varphi}{\leadsto}_{pq} y & \text{if} & x \overset{\varphi}{\leadsto}_p z \text{ and } z \overset{\varphi}{\leadsto}_q y. \end{array}$$

For all p and φ, we define a binary relation $\overset{\varphi}{\leadsto}_p$ between variables and sorts where $x \overset{\varphi}{\leadsto}_p a$ reads as "a can be reached from x over path p in φ":

$$x \overset{\varphi}{\leadsto}_p a \ \text{ if } \ x \overset{\varphi}{\leadsto}_p y \text{ and } a(y) \text{ in } \varphi$$

Definition 25 Path Consistency. For every constraint φ, variable sequence $\overline{x}$ and path p, let $A_p^\varphi = \{a \mid x \overset{\varphi}{\leadsto}_p a, x \in \{\overline{x}\}\}$. We call a constraint φ *path consistent* if, for all $\overline{x}$, $\overline{y}$, and p, $\sim(\overline{x}\overline{y})$ in φ implies $\sim A_p^\varphi(\overline{x})$.

Lemma 26. *Every $F1$, $F2$, $F4$-closed and path consistent constraint is satisfiable.*

Proof. Let φ be $F1$, $F2$, and $F4$-closed and path consistent. Define $A_p^\varphi(x) =_{def} \{a \mid x \overset{\varphi}{\leadsto}_p a\}$ for all x and paths p, and let

$$D_x =_{def} \{p \mid x \overset{\varphi}{\leadsto}_p\} \qquad \text{and} \qquad \pi_x =_{def} \{(p, \sqcup A_p^\varphi(x)) \mid p \in D_x\}$$

By Lemma 23, A_p^φ is non-empty for all x and $p \in D_x$. By path consistency of φ, π_x is well-defined and hence a total sorting on D_x, thus (D_x, π_x) is a feature tree. We define the variable assignment sol into feature trees by setting, for all x,

$$\mathsf{sol}(x) =_{def} (D_x, \pi_x)$$

We now verify that sol satisfies all primitive constraints in φ and thus φ.

$x \leq y$ in φ**:** For all x', if $x \overset{\varphi}{\rightsquigarrow}_p x'$ then $y \overset{\varphi}{\rightsquigarrow}_p x'$ by the definition of path reachability. Thus, $D_{\mathsf{sol}(x)} \subseteq D_{\mathsf{sol}(y)}$. For all a, if $x \overset{\varphi}{\rightsquigarrow}_p a$ then $y \overset{\varphi}{\rightsquigarrow}_p a$ by the definition of path reachability. Thus $A_p^\varphi(x) \subseteq A_p^\varphi(y)$ which implies $\sqcup A_p^\varphi(x) \leq \sqcup A_p^\varphi(y)$, and hence $\pi_{\mathsf{sol}(x)} \sqsubseteq \pi_{\mathsf{sol}(y)}$; in combination we obtain $\mathsf{sol}(x) \leq \mathsf{sol}(y)$.

$x[f]y$ in φ**:** This case is unchanged w.r.t. the corresponding case in [23].

$a(x)$ in φ**:** Reflexivity, *i. e.*, $F1$-closedness, implies $x \leq x$ in φ. Thus $x \overset{\varphi}{\rightsquigarrow}_\varepsilon a$, and hence $a \leq \pi_x(\varepsilon)$.

$\sim\overline{x}$ in φ**:** For every partition of $\overline{x} = y_1 \ldots y_n \overline{z}$ and all paths p we must show that $\{\pi_{y_1}(p), \ldots, \pi_{y_n}(p)\}$ are compatible (if undefined, let $\pi_{y_i}(p)$ be equal to the sort chosen in Lemma 23). Let $\tilde{y_i}$, for all i, be the sequence of all lower bounds of y_i in φ, *i. e.*, of all y' such that $y_i \overset{\varphi}{\rightsquigarrow}_\varepsilon y'$. Note that $y_i \in \{\tilde{y_i}\}$ for all i. Then, by $F5$-closedness, $\sim(\tilde{y_1} \ldots \tilde{y_n}\overline{z})$ in φ. From path consistency and the definition of π_{y_i} we get $\sim A_p^\varphi(y)$ for all p, where $A_p^\varphi(y) = \bigcup_{i=1}^n A_p^\varphi(y_i)$. □

Lemma 27. *Every $F4 - F7$-closed constraint is path consistent.*

Proof. Let φ be $F4 - F7$-closed. The proof of path consistency is by induction over paths p. Assume $\overline{x}$, $\overline{y}$, and let $A_p^\varphi(\overline{x}) = \{a \mid x \overset{\varphi}{\rightsquigarrow}_p a, x \in \{\overline{x}\}\}$. We have to show that $\sim A_p^\varphi(\overline{x})$. The induction is a generalisation of the corresponding one in [23] where the inducation base is proved with $F4, F5$- and $F7$-closedness, and the induction step with $F4, F5$- and $F6$-closedness. □

Proof of Proposition 8. We assume a k-coherent $(S, \sqsubseteq)$ and an $\mathsf{F}^{(k)}$-closed constraint φ. Now let $\tilde{\varphi}$ be the F-closure of φ. The following two properties hold.

1. Either $\tilde{\varphi} = \mathit{false}$, or every primitive constraint in $\tilde{\varphi}$ which is not contained in φ is an incompatibility constraint.
2. If $\sim(\overline{x})$ in $\tilde{\varphi}$ but $\sim(\overline{x})$ not in φ, and $\overline{y}$ is a subsequence of $\overline{x}$, then $\mathsf{clash}_\varphi(\overline{y})$ does not hold.

The first property is simple. The second one can be shown by induction over the length of the F-derivation of $\sim(\overline{x})$ from φ. In combination, both properties imply that $\tilde{\varphi}$ is not $\perp$ and hence satisfiable, so φ is satisfiable. □