Theoretical Computer Science 122 (1994) 263-283
Elsevier

A feature constraint system for logic programming with entailment

Hassan Aït-Kaci and Andreas Podelski

Paris Research Laboratory, Digital Equipment Corporation, 85 avenue Victor Hugo, 92500 Rueil Malmaison, France

Gert Smolka

Deutsches Forschungszentrum für Künstliche Intelligenz, Universität des Saarlandes, Stuhlsatzenhausweg 3, W-6600 Saarbrücken 11, Germany

Abstract

Aït-Kaci, H., A. Podelski and G. Smolka, A feature constraint system for logic programming with entailment, Theoretical Computer Science 122 (1994) 263-283.

We introduce a constraint system called *FT*. This system offers a theoretical and practical alternative to the usual Herbrand system of constraints over constructor trees. Like Herbrand, *FT* provides a universal data structure based on trees. However, the trees of *FT* (called feature trees) are more general than the constructor trees of Herbrand, and the constraints of *FT* are of finer grain and of different expressiveness. The essential novelty of *FT* is provided by functional attributes called features which allow representing data as extensible records, a more flexible way than that offered by Herbrand's fixed arity constructors. The feature tree structure determines an algebraic semantics for *FT*. We establish a logical semantics, thanks to three axiom schemes presenting the first-order theory *FT*. We propose using *FT* as a constraint system for logic programming. We provide a test for constraint unsatisfiability, and a test for constraint entailment. The former corresponds to unification and the latter to matching. The combination of the two is needed for advanced control mechanisms. We use the concept of relative simplification of constraints, a normalization process that decides entailment and unsatisfiability simultaneously. The two major technical contributions of this work are: (i) an incremental system performing relative simplification for *FT* that we prove to be sound and complete; and (ii) a proof showing that *FT* satisfies independence of negative constraints, the property that conjoined negative constraints may be handled independently.

Résumé

Nous présentons un système de contraintes appelé *FT*. Ce système constitue une alternative théorique et pratique à Herbrand, le système usuel de contraintes sur les arbres à constructeurs.

Correspondence to: H. Aït-Kaci, Paris Research Laboratory, Digital Equipment Corporation, 85 avenue Victor Hugo, 92500 Rueil Malmaison, France. Email addresses of the authors: hak@prl.dec.com, podelski@prl.dec.com, smolka@dfki.uni-sb.de.

Comme Herbrand, FT fournit une structure de données d'arbres. Cependant, les arbres de FT (appelés arbres à traits) sont plus généraux que les arbres à constructeurs de Herbrand, et les contraintes de FT sont d'une granularité plus fine et d'expressivité différente. L'innovation essentielle de FT est dûe à des attributs fonctionnels appelés traits qui permettent de représenter les données sous forme de structure d'enregistrement extensible, de manière plus flexible que celle offerte par les constructeurs d'arité fixe de Herbrand. La structure d'arbre à traits détermine une sémantique algébrique pour FT. Nous établissons une sémantique logique, grâce à trois schémas d'axiomes présentant la théorie du premier ordre FT. Nous proposons d'utiliser FT comme un système de contraintes pour la programmation logique. Nous produisons un critère de satisfaisabilité de contrainte, et un critère de validation d'implication de contrainte. Le premier correspond à l'unification et le deuxième au filtrage. La combinaison des deux est nécessaire pour des mécanismes de contrôle avancés. Nous utilisons le concept de simplification relative, un processus de normalisation qui décide simultanément la validation d'implication et la non-satisfaisabilité. Les deux contributions techniques majeures de ce travail sont: (i) un système incrémental effectuant la simplification relative pour FT, que nous démontrons être cohérent et complet; et (ii) une preuve montrant que FT jouit de l'indépendance des contraintes négatives, propriété qui permet à des contraintes négatives conjointes d'être traitées séparément.

1. Introduction

An important structural property of many logic programming systems is the fact that they factorize into a constraint system and a relational facility. Colmerauer's Prolog II [10] is an early language design making explicit use of this property. CLP (constraint logic programming [12]), ALPS [18], CCP (concurrent constraint programming [23]), and KAP (Kernel Andorra Prolog [11]) are recent logic programming frameworks that exploit this property to its full extent by being parameterized with respect to an abstract class of constraint systems. The basic operation that these frameworks require of a constraint system is a test for unsatisfiability. In addition, ALPS, CCP, and KAP require a test for entailment between constraints, which is needed for advanced control mechanisms such as delaying, coroutining, synchronization, committed choice, and deep constraint propagation. LIFE [5, 6], formally a CLP language, employs a related, but limited, suspension strategy to enforce deterministic functional application. Given this situation, constraint systems are a central issue in research on logic programming.

The constraint systems of most existing logic programming languages are variations and extensions of Herbrand [16], the constraint system underlying Prolog. The individuals of Herbrand are trees corresponding to ground terms, and the atomic constraints are equations between terms. Seen from the perspective of programming, Herbrand provides a universal data structure as a logical system.

This paper presents a constraint system FT, which we feel is an intriguing alternative to Herbrand both theoretically and practically. Like Herbrand, FT provides a universal data structure based on trees. However, the trees of FT (called feature trees) are more general than the trees of Herbrand (called constructor trees), and the constraints of FT are of a finer grain and of different expressiveness. The essential novelty of FT is due to functional attributes called features, which provide for record-like descriptions of data avoiding the overspecification, intrinsic in Herbrand's constructor-based descriptions. For the special case of constructor trees, features amount to argument selectors for constructors.

Constructor trees are useful for structuring data in modern symbolic programming languages; e.g., Prolog and ML. This gives the more flexible feature trees an interesting potential. More precisely, feature trees model extensible record structures. They form the semantics of record calculi like [1], which are used in symbolic programming languages [5] and in computational linguistics (see e.g. [3, 24, 8]). Generally, these extensible record structures allow hierarchical representation of partial knowledge. They lend themselves to object-oriented programming techniques [3].

Let us suppose that we want to say that x is a wine whose grape is riesling and whose color is white. To do this in Herbrand, one may write the equation:

$$\mathtt{x} = \mathtt{wine}(\mathtt{riesling}, \mathtt{white}, \mathtt{y}_1, \ldots, \mathtt{y}_n)$$

with the implicit assumption that the first argument of the constructor wine carries the "feature" grape, the second argument carries the "feature" color, and the remaining arguments $\mathtt{y}_1, \ldots, \mathtt{y}_n$ carry the remaining "features" of the chosen representation of wines. The obvious difficulty with this description is that it says more than we want to say; namely, that the constructor wine has $n+2$ arguments and that the "features" grape and color are represented as the first and the second argument.

The constraint system FT avoids this overspecification by allowing the description

$$\mathtt{x} : \mathtt{wine}[\mathtt{grape} \Rightarrow \mathtt{riesling},\ \mathtt{color} \Rightarrow \mathtt{white}] \tag{1}$$

saying that x has sort wine, its feature grape is riesling, and its feature color is white. Nothing is said about other features of x, which may or may not exist.

The individuals of FT are feature trees. A feature tree is a possibly infinite tree whose nodes are labeled with symbols called sorts, and whose edges are labeled with symbols called features. The labeling with features is deterministic in that all edges departing from a node must be labeled with distinct features. Thus, every direct subtree of a feature tree can be identified by the feature labeling the edges leading to it. The constructor trees of Herbrand can be represented as feature trees whose edges are labeled with natural numbers indicating the corresponding argument positions.

Examples of feature trees are shown in Fig. 1. All but the second and third feature tree in Fig. 1 satisfy the description (1).

The constraints of FT are ordinary first-order formulae taken over a signature that accommodates sorts as unary predicates and features as binary predicates. Thus the description (1) is actually syntactic sugar for the formula:

$$\begin{aligned} &\mathit{wine}(x) \wedge \exists y(\mathit{grape}(x, y) \wedge \mathit{riesling}(y)) \\ &\qquad \wedge \exists y(\mathit{color}(x, y) \wedge \mathit{white}(y)). \end{aligned}$$

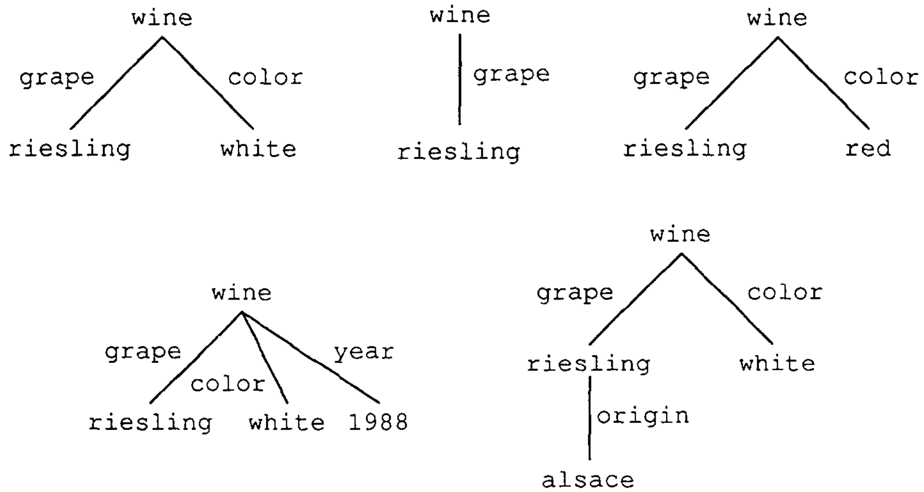

Fig. 1. Examples of Feature Trees.

The set of all rational feature trees is made into a corresponding logical structure $\mathcal{T}$ by letting $A(x)$ hold if and only if the root of x is labeled with the sort A, and letting $f(x, y)$ hold if and only if x has y as direct subtree via the feature f. The feature tree structure $\mathcal{T}$ fixes an algebraic semantics for FT.

We will also establish a logical semantics, which is given by three axiom schemes fixing a first-order theory FT. Backofen and Smolka [7] show that $\mathcal{T}$ is a model of FT and that FT is in fact a complete theory, which means that FT is exactly the theory induced by $\mathcal{T}$. However, we will not use the completeness result in the present paper, but show explicitly that entailment with respect to $\mathcal{T}$ is the same as entailment with respect to FT.

The two major technical contributions of this paper are (i) an incremental simplification system for entailment that is proven to be sound and complete, and (ii) a proof showing that the "independence of negative constraints" property [9, 16, 17] holds for FT.

The incremental entailment simplification system is the prerequisite for FT's use with either of the constraint programming frameworks ALPS, CCP, KAP or LIFE mentioned at the beginning of this section. Roughly, these systems are concurrent thanks to a new effective discipline for procedure parameter-passing that we could describe as "call-by-constraint-entailment" (as opposed to Prolog's call-by-unification).

The independence property is important since it means that *negative constraints* on feature trees can be solved (exactly like in Colmerauer's work on disequations over infinite trees [9]). Namely, thanks to independence, a conjunction with more than one negated constraints $\phi \wedge \neg\phi_1 \wedge \cdots \wedge \neg\phi_n$ can be solved by testing separately each negated constraint ϕ_i for entailment, for $i = 1, \ldots, n$. This, of course, is done by our simplification system for entailment.

One origin of FT is Aït-Kaci's ψ-term calculus [1], which is at the heart of the programming language LOGIN [3] and further extended in the language LIFE [5] with functions over feature structures, thanks to a generalization of the concept of residuation of Le Fun [4].[1] Other precursors of FT are the feature descriptions found in unification grammars [15, 14] developed for natural language processing, and also the formalisms of Mukai [19, 20] (for a thorough survey of precursors in this field, cf. [8]). These early feature structure formalisms were presented in a nonlogical form. Major steps in the process of their understanding and logical reformulation are the articles [22, 25, 13, 24]. Feature trees, the feature tree structure $\mathcal{T}$, and the axiomatization of $\mathcal{T}$ were first given in [7]. The idea of relative simplification of constraints was first introduced and used in [6] to explain the behavior of functions as passive constraints in LIFE.

The paper is organized as follows. Section 2 defines the basic notions and discusses the differences in expressivity between Herbrand and FT. Section 3 gives a basic simplification system that decides satisfiability of positive constraints. The material of Section 4 is not limited to FT but discusses the notion of incremental entailment checking and its connection with the independence property and negation. Section 5 gives the entailment simplification system, proves it sound, complete and terminating, and also proves that FT satisfies the independence property.

2. Feature trees and constraints

To give a rigorous formalization of feature trees, we first fix two disjoint alphabets $\mathcal{S}$ and $\mathcal{F}$, whose symbols are called *sorts* and *features*, respectively. The letters A, B, C will always denote sorts, and the letters f, g, h will always denote features. Words over $\mathcal{F}$ are called paths. The concatenation of two paths v and w results in the path vw. The symbol ε denotes the empty path, $v\varepsilon = \varepsilon v = v$, and $\mathcal{F}^\star$ denotes the set of all paths.

A *tree domain* is a nonempty set $D \subseteq \mathcal{F}^\star$ that is prefix-closed; that is, if $vw \in D$, then $v \in D$. Thus, it always contains the empty path.

A *feature tree* is a mapping $t : D \to \mathcal{S}$ from a tree domain D into the set of sorts. The paths in the domain of a feature tree represent the nodes of the tree; the empty path represents its root. The letters s and t are used to denote feature trees.

When convenient, we may consider a feature tree t as a relation, i.e., $t \subseteq \mathcal{F}^\star \times \mathcal{S}$, and write $(w, A) \in t$ instead of $t(w) = A$. (Clearly, a relation $t \subseteq \mathcal{F}^\star \times \mathcal{S}$ is a feature tree if and only if $D = \{w \mid \exists A : (w, A) \in t\}$ is a tree domain and t is functional; that is, if $(w, A) \in t$ and $(w, B) \in t$, then $A = B$.) As relations, i.e., as subsets of $\mathcal{F}^\star \times \mathcal{S}$, feature trees are

[1] Le Fun [4] is an extension of Prolog seen as a constraint logic programming system over Herbrand terms extended with applicative expressions. Le Fun's constraint solver achieves implicit coroutining thanks to an automatic suspension mechanism called "residuation" delaying equations with insufficiently instantiated function arguments. Resumption is triggered asynchronously by function argument matching.

partially ordered by set inclusion. We say that s is *smaller than* (or, *is a prefix-subtree of*; or, *subsumes*; or, *approximates*) t if $s \subseteq t$.

The *subtree* wt of a feature tree t at one of its nodes w is the feature tree defined by (as a relation):

$$wt := \{(v, A) \mid (wv, A) \in t\}.$$

If D is the domain of t, then the domain of wt is the set $w^{-1}D = \{v \mid wv \in D\}$. Thus, wt is given as the mapping $wt : w^{-1}D \to \mathcal{S}$ defined on its domain by $wt(v) = t(wv)$. A feature tree s is called a *subtree* of a feature tree t if it is a subtree $s = wt$ at one of its nodes w, and a direct subtree if $w \in \mathcal{F}$.

A feature tree t with domain D is called *rational* if (i) t has only finitely many subtrees and (ii) t is finitely branching; that is: for every $w \in D$, $w\mathcal{F} \cap D = \{wf \in D \mid f \in \mathcal{F}\}$ is finite. Assuming (i), the condition (ii) is equivalent to saying that there exist finitely many features $f_1, \ldots, f_n$ such that $D \subseteq \{f_1, \ldots, f_n\}^\star$.

Constraints over feature trees will be defined as first-order formulae. We first fix a first-order signature $\mathcal{S} \uplus \mathcal{F}$ by taking sorts as unary and features as binary relation symbols. Moreover, we fix an infinite alphabet of *variables* and adopt the convention that x, y, z always denote variables. Under this signature, every term is a variable and an *atomic formula* is either a feature constraint xfy ($f(x, y)$ in standard notation), a sort constraint Ax ($A(x)$ in standard notation), an equation $x \doteq y$, $\perp$ ("false"), or $\top$ ("true"). Compound formulae are obtained as usual by the connectives $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\neg$ and the quantifiers $\exists$ and $\forall$. We use $\tilde{\exists}\phi$ and $\tilde{\forall}\phi$ to denote the existential and universal closure of a formula ϕ, respectively. Moreover, $\mathcal{V}(\phi)$ is taken to denote the set of all variables that occur free in a formula ϕ. The letters ϕ and ψ will always denote formulae. In the following we will not make a distinction between formulae and constraints; that is, a *constraint* is a formula as defined above.

$\mathcal{S} \uplus \mathcal{F}$-structures and *validity* of formulae in $\mathcal{S} \uplus \mathcal{F}$-structures are defined as usual. Since we consider only $\mathcal{S} \uplus \mathcal{F}$-structures in the following, we will simply speak of structures. A *theory* is a set of closed formulae. A *model* of a theory is a structure that satisfies every formula of the theory. A formula ϕ is a *consequence* of a theory T ($T \models \phi$) if $\tilde{\forall}\phi$ holds in every model of T. A formula ϕ is *satisfiable* in a structure $\mathcal{A}$ if $\tilde{\exists}\phi$ holds in $\mathcal{A}$. Two formulae ϕ, ψ are *equivalent* in a structure $\mathcal{A}$ if $\tilde{\forall}(\phi \leftrightarrow \psi)$ holds in $\mathcal{A}$. We say that a formula ϕ *entails* a formula ψ in a structure $\mathcal{A}$ [theory T] and write $\phi \models_{\mathcal{A}} \psi$ [$\phi \models_T \psi$] if $\tilde{\forall}(\phi \rightarrow \psi)$ holds in $\mathcal{A}$; i.e., $\mathcal{A} \models \tilde{\forall}(\phi \rightarrow \psi)$ [is a consequence of T; i.e., $FT \models \tilde{\forall}(\phi \rightarrow \psi)$]. A theory T is *complete* if for every closed formula ϕ either ϕ or $\neg\phi$ is a consequence of T.

The *feature tree structure* $\mathcal{T}$ is the $\mathcal{S} \uplus \mathcal{F}$-structure defined as follows:

- the domain of $\mathcal{T}$ is the set of all rational feature trees;
- $t \in A^{\mathcal{T}}$ if and only if $t(\varepsilon) = A$ (t's root is labeled with A);
- $(s, t) \in f^{\mathcal{T}}$ if and only if $f \in D_s$ and $t = fs$ (t is the subtree of s at f).

Roughly, the Herbrand constraint $y = A(x_1, x_2)$, where A is a binary constructor symbol, and the feature constraint $Ay \wedge y1x_1 \wedge y2x_2$, where A is a sort and $1, 2, \ldots$ are features, correspond to each other. (We will see later that this correspondence is a formal one for satisfiability, but not for entailment.) Now it becomes clear what we mean by saying that feature constraints are finer grained. Also, feature trees are more general in the sense that they satisfy more constraints. For example, no constructor tree y satisfies both $y=A(x_1,x_2)$ and $y=A(x_1,x_2,x_3)$.

Next we discuss the expressivity of our constraints with respect to feature trees (that is, with respect to the feature tree structure $\mathcal{T}$) by means of examples. The constraint:

$$\neg\exists y(xfy)$$

says that x has no subtree at f; i.e., that there is no edge departing from x's root that is labeled with f. To say that x has subtree y at path $f_1\ldots f_n$, we can use the constraint:

$$\exists z_1\ldots\exists z_{n-1}(xf_1z_1\wedge z_1f_2z_2\wedge\cdots\wedge z_{n-1}f_ny).$$

Now let us look at statements we cannot express. One simple unexpressible statement is "y is a subtree of x" (i.e. "$\exists w\colon y=wx$"). Moreover, we cannot express that x is smaller than y. Finally, if we assume that the alphabet $\mathcal{F}$ of features is infinite, we cannot say that x has subtrees at features $f_1,\ldots,f_n$ but no subtree at any other feature. In particular, we then cannot say that x is a primitive feature tree; that is, has no proper subtree.

The theory FT_0 is given by the following two axiom schemes:

(Ax1) $\forall x\forall y\forall z(xfy\wedge xfz\rightarrow y\doteq z)$ (for every feature f),

(Ax2) $\forall x(Ax\wedge Bx\rightarrow\bot)$ (for every two distinct sorts A and B).

The first axiom scheme says that features are functional and the second scheme says that sorts are mutually disjoint. Clearly, $\mathcal{T}$ is a model of FT_0. Moreover, FT_0 is incomplete (for instance, $\exists x(Ax)$ holds in $\mathcal{T}$ but not in other models of FT_0). We will see in the next section that FT_0 plays an important role with respect to basic constraint simplification.

Next we introduce some additional notation needed in the rest of the paper. This notation will also allow us to state a third axiom scheme that, as shown in [7], extends FT_0 to a complete axiomatization of $\mathcal{T}$.

Throughout the paper we assume that conjunction of formulae is an associative and commutative operator that has $\top$ as neutral element. This means that we identify $\phi\wedge(\psi\wedge\theta)$ with $\theta\wedge(\psi\wedge\phi)$, and $\phi\wedge\top$ with ϕ (but not, e.g., $xfy\wedge xfy$ with xfy). A conjunction of atomic formulae can thus be seen as the finite multiset of these formulae, where conjunction is multiset union, and $\top$ (the "empty conjunction") is the empty multiset. We will write $\psi\subseteq\phi$ (or $\psi\in\phi$, if ψ is an atomic formula) if there exists a formula ψ' such that $\psi\wedge\psi'=\phi$.

We will use an additional atomic formula $xf\uparrow$ ("f undefined on x") that is taken to be equivalent to $\neg\exists y(xfy)$, for some variable y (other than x).

Only for the formulation of the third axiom we introduce the notion of a *solved-clause*, which is either $\top$ or a conjunction ϕ of atomic formulae of the form xfy, Ax or $xf\uparrow$ such that the following conditions are satisfied:

(1) if $Ax\in\phi$ and $Bx\in\phi$, then $A=B$;

(2) if $xfy \in \phi$ and $xfz \in \phi$, then $y = z$;
(3) if $xfy \in \phi$, then $xf\uparrow \notin \phi$.

Given a solved-clause ϕ, we say that a variable x is *dependent* in ϕ if ϕ contains a constraint of the form Ax, xfy or $xf\uparrow$, and use $\mathscr{DV}(\phi)$ to denote the set of all variables that are dependent in ϕ.

The theory FT is obtained from FT_0 by adding the axiom scheme:
(Ax3) $\tilde{\forall} \exists X \phi$ (for every solved-clause ϕ and $X = \mathscr{DV}(\phi)$).

Theorem 2.1. *The feature tree structure $\mathscr{T}$ is a model of the theory FT.*

Proof. We will only show that $\mathscr{T}$ is a model of the third axiom. Let X be the set of dependent variables of the solved-clause ϕ, $X = \mathscr{DV}(\phi)$. Let α be any $\mathscr{T}$-valuation defined on $\mathscr{V}(\phi) - X$; we write the tree $\alpha(y)$ as t_y. We will extend α on X such that $\mathscr{T}, \alpha \models \phi$.

Given $x \in X$, we define the "punctual" tree $t_x = \{(\varepsilon, A)\}$, where $A \in \mathscr{S}$ is the sort such that $Ax \in \phi$, if it exists, and arbitrary, otherwise. Now we are going to use the notion of *tree sum* of Nivat [21], where $w^{-1}t = \{(wv, A) | (v, A) \in t\}$ ("the tree t translated by w"), and we define:

$$\alpha(x) = \biguplus \{w^{-1} t_y | x \overset{w}{\rightsquigarrow} y \textit{ for some } y \in \mathscr{V}(\phi), w \in \mathscr{F}^{\star}\}.$$

Here the relation $\overset{w}{\rightsquigarrow}$ is given by: $x \overset{\varepsilon}{\rightsquigarrow} x$, and $x \overset{wf}{\rightsquigarrow} y$ if $x \overset{w}{\rightsquigarrow} y'$ and $y'fy \in \phi$, for some $y' \in \mathscr{V}(\phi)$ and some $f \in \mathscr{F}$. Since:

$$\alpha(x) = \bigcup \{w^{-1} \alpha(y) | \dots\}$$

and, for a node w of $\alpha(x)$, $w\alpha(x) = \alpha(y)$, it follows that $\alpha(x)$ is a rational tree and that $\mathscr{T}, \alpha \models \phi$. $\square$

For another proof of this theorem see [7], which also proves that FT is a complete theory if the alphabets of sorts and features are infinite.

A practical motivation for the assumption on the infiniteness of $\mathscr{F}$ (and of $\mathscr{S}$ as well) is the need to account for dynamic record field updates. It turns out that this semantical point of view has advantages in efficiency as well. Thus, the algorithms we present in this paper for entailment and for solving negative constraints on feature trees rely on the infiniteness of $\mathscr{F}$ and $\mathscr{S}$.

3. Basic simplification

A *basic constraint* is either $\perp$ or a possibly empty conjunction of atomic formulae of the form Ax, xfy, and $x \doteq y$. The following five *basic simplification* rules constitute a simplification system for basic constraints, which, as we will see, decides whether a basic constraint is satisfiable in $\mathscr{T}$.

$$(1)\qquad \frac{xfy \wedge xfz \wedge \phi}{xfz \wedge y \doteq z \wedge \phi}$$

$$(2)\qquad \frac{Ax \wedge Bx \wedge \phi}{\perp}\quad A \neq B$$

$$(3)\qquad \frac{Ax \wedge Ax \wedge \phi}{Ax \wedge \phi}$$

$$(4)\qquad \frac{x \doteq y \wedge \phi}{x \doteq y \wedge \phi[x \leftarrow y]}\quad x \in \mathscr{V}(\phi) \text{ and } x \neq y$$

$$(5)\qquad \frac{x \doteq x \wedge \phi}{\phi}$$

The notation $\phi[x \leftarrow y]$ is used to denote the formula that is obtained from ϕ by replacing every occurrence of x with y. We say that a constraint ϕ *simplifies to* a constraint ψ by a simplification rule ρ if ϕ/ψ is an instance of ρ. We say that a constraint ϕ *simplifies to* a constraint ψ if either $\phi = \psi$ or ϕ simplifies to ψ in finitely many steps each licensed by one of the five simplification rules given above.

Example 3.1. In order to check whether the two feature descriptions $x[f \Rightarrow u : A]$ and $y[f \Rightarrow v : A]$ are unifiable, in the sense of [3], we will simplify the basic constraint $xfu \wedge yfv \wedge Au \wedge Av \wedge z \doteq x \wedge y \doteq z$.

The following basic simplification chain leads to a solved constraint (which, as shown in [24, 5], exhibits unifiability):

$$\begin{array}{lll}
& xfu \wedge yfv \wedge Au \wedge Av \wedge z \doteq x \wedge y \doteq z & \\
\Rightarrow & xfu \wedge yfv \wedge Au \wedge Av \wedge z \doteq x \wedge y \doteq x & \text{(by rule 4)} \\
\Rightarrow & xfu \wedge xfv \wedge Au \wedge Av \wedge z \doteq x \wedge y \doteq x & \text{(by rule 4)} \\
\Rightarrow & xfv \wedge Au \wedge Av \wedge u \doteq v \wedge z \doteq x \wedge y \doteq x & \text{(by rule 1)} \\
\Rightarrow & xfv \wedge Av \wedge Av \wedge u \doteq v \wedge z \doteq x \wedge y \doteq x & \text{(by rule 4)} \\
\Rightarrow & xfv \wedge Av \wedge u \doteq v \wedge z \doteq x \wedge y \doteq x & \text{(by rule 3)}
\end{array}$$

Using the same steps up to the last one, the constraint $xfu \wedge yfv \wedge Au \wedge Bv \wedge z \doteq x \wedge y \doteq z$ simplifies to $\perp$ (in the last step, rule 2 instead of rule 3 is applied).

Proposition 3.2. *If the basic constraint ϕ simplifies to ψ, then $FT_0 \models \phi \leftrightarrow \psi$.*

Proof. The rules 3, 4 and 5 perform equivalence transformations with respect to every structure. The rules 1 and 2 correspond exactly to the two axiom schemes of FT_0 and perform equivalence transformations with respect to every model of FT_0. $\square$

We say that a basic constraint ϕ *binds* a variable x to y if $x \doteq y \in \phi$ and x occurs only once in ϕ. At this point it is important to note that we consider equations as ordered; that is, assume that $x \doteq y$ is different from $y \doteq x$ if $x \neq y$. We say that a variable x is *eliminated*, or *bound by* ϕ, if ϕ binds x to some variable y.

Proposition 3.3. *The basic simplification rules are terminating.*

Proof. First observe that the simplification rules do not add new variables and preserve eliminated variables. Furthermore, Rule 4 increases the number of eliminated variables by one. Hence, we know that if an infinite simplification chain exists, we can assume without loss of generality that it only employs the rules 1, 3 and 5. Since rule 1 decreases the number of feature constraints "xfy", which is not increased by rules 3 and 5, we know that if an infinite simplification chain exists, we can assume without loss of generality that it only employs rules 3 and 5. Since this is clearly impossible, an infinite simplification chain cannot exist. □

A basic constraint is called *normal* if none of the five simplification rules applies to it. A constraint ψ is called a *normal form* of a basic constraint ϕ if ϕ can be simplified to ψ and ψ is normal. A *solved constraint* is a normal constraint that is different from $\perp$.

So far we know that we can compute for any basic constraint ϕ, a normal form ψ by applying the simplification rules as long as they are applicable. Although the normal form ψ may not be unique for ϕ, we know that ϕ and ψ are equivalent in every model of FT_0. It remains to show that every solved constraint is satisfiable in $\mathcal{T}$.

Every basic constraint ϕ has a unique decomposition $\phi = \phi_N \wedge \phi_G$ such that ϕ_N is a possibly empty conjunction of equations "$x \doteq y$" and ϕ_G is a possibly empty conjunction of feature constraints "xfy" and sort constraint "Ax". We call ϕ_N the *normalizer* and ϕ_G the *graph* of ϕ.

Proposition 3.4. *A basic constraint $\phi \neq \perp$ is solved if and only if the following conditions hold*:

(1) *an equation $x \doteq y$ appears in ϕ only if x is eliminated in ϕ;*
(2) *the graph of ϕ is a solved clause;*
(3) *no primitive constraint appears more than once in ϕ.*

Proposition 3.5. *Every solved constraint is satisfiable in every model of FT.*

Proof. Let ϕ be a solved constraint and $\mathcal{A}$ be a model of FT. Then we know by axiom scheme Ax3 that the graph ϕ_G of a solved constraint ϕ is satisfiable in an FT-model $\mathcal{A}$. A variable valuation α into $\mathcal{A}$ such that $\mathcal{A}, \alpha \models \phi_G$ can be extended on all eliminated variables simply by $\alpha(x) = \alpha(y)$ if $x \doteq y \in \phi$, such that $\mathcal{A}, \alpha \models \phi$. □

The following theorem states that basic simplification yields a decision procedure for satisfiability of basic constraints.

Theorem 3.6. *Let ψ be a normal form of a basic constraint ϕ. Then ϕ is satisfiable in $\mathscr{T}$ if and only if $\psi \neq \bot$.*

Proof. Since ϕ and ψ are equivalent in every model of FT_0 and $\mathscr{T}$ is a model of FT_0, it suffices to show that ψ is satisfiable in $\mathscr{T}$ if and only if $\psi \neq \bot$. To show the nontrivial direction, suppose $\psi \neq \bot$. Then ψ is solved and we know by the preceding proposition that ψ is satisfiable in every model of FT. Since $\mathscr{T}$ is a model of FT, we know that ψ is satisfiable in $\mathscr{T}$. $\square$

The next theorem implies the elementary equivalence of all models of FT with respect to satisfiability of basic constraints. Namely, satisfiability in any of the models of FT means satisfiability in all of them. Also, it is sufficient to test satisfiability in the model $\mathscr{T}$ alone. Finally, only the first two axioms are relevant for satisfiability.

Theorem 3.7. *For every basic constraint ϕ the following statements are equivalent:*

$$\mathscr{T} \models \tilde{\exists}\phi \;\Leftrightarrow\; \exists \textit{ model } \mathscr{A} \textit{ of } FT_0\colon \mathscr{A} \models \tilde{\exists}\phi \;\Leftrightarrow\; FT \models \tilde{\exists}\phi.$$

Proof. The implication $1 \Rightarrow 2$ holds since $\mathscr{T}$ is a model of FT_0. The implication $3 \Rightarrow 1$ follows from the fact that $\mathscr{T}$ is a model of FT. It remains to show that $2 \Rightarrow 3$.

Let ϕ be satisfiable in some model of FT_0. Then we can apply the simplification rules to ϕ and compute a normal form ψ such that ϕ and ψ are equivalent in every model of FT_0. Hence, ψ is satisfiable in some model of FT_0. Thus $\psi \neq \bot$, which means that ψ is solved. Hence, we know by the preceding proposition that ψ is satisfiable in every model of FT. Since ϕ and ψ are equivalent in every model of $FT_0 \subseteq FT$, we have that ϕ is satisfiable in every model of FT. $\square$

4. Entailment, independence and negation

In this section we discuss some general properties of constraint entailment. This prepares the ground for the next section, which is concerned with entailment simplification in the feature tree constraint system.

Throughout this section we assume that $\mathscr{A}$ is a structure, γ and ϕ are formulae that can be interpreted in $\mathscr{A}$, and that X is a finite set of variables.

We say that γ *disentails* ϕ in $\mathscr{A}$ if γ entails $\neg\phi$ in $\mathscr{A}$. If γ is satisfiable in $\mathscr{A}$, then γ cannot both entail and disentail $\exists X\phi$ in $\mathscr{A}$. We say that γ *determines* ϕ in $\mathscr{A}$ if γ either entails or disentails ϕ in $\mathscr{A}$.

Given γ, ϕ and X, we want to determine in an *incremental* manner whether γ entails or disentails $\exists X\phi$. Typically, γ will *not* determine $\exists X\phi$ when $\exists X\phi$ is considered first, but this may change when γ is strengthened to $\gamma \wedge \gamma'$. To this end, we use the concept of *relative simplification* of constraints first introduced in [6]. The basic idea leading to an incremental entailment checker is to simplify ϕ with respect to (relatively to) the *context* γ and the *local variables* X. Given γ, X and ϕ, simplification must yield a formula ψ such that:

$$\gamma \models_{\mathscr{A}} \exists X\phi \leftrightarrow \exists X\psi.$$

The following facts provide some evidence that this is the appropriate invariant for entailment simplification.

Proposition 4.1. *Let* $\gamma \models_{\mathscr{A}} \exists X\phi \leftrightarrow \exists X\psi$. *Then*
(1) $\gamma \models_{\mathscr{A}} \exists X\phi$ *if and only if* $\gamma \models_{\mathscr{A}} \exists X\psi$;
(2) $\gamma \models_{\mathscr{A}} \neg\exists X\phi$ *if and only if* $\gamma \models_{\mathscr{A}} \neg\exists X\psi$;
(3) *if* $\psi = \bot$, *then* $\gamma \models_{\mathscr{A}} \neg\exists X\phi$;
(4) *if* $\exists X\psi$ *holds in* $\mathscr{A}$, *then* $\gamma \models_{\mathscr{A}} \exists X\phi$.

Statements (1) and (2) say that it does not matter whether entailment and disentailment are decided for ϕ or ψ. Statement (3) gives a local condition for disentailment, and statement (4) gives a local condition for entailment. The entailment simplification system for feature trees given in the next section will in fact decide entailment and disentailment by simplifying such that the condition of statement (4) is met in the case of entailment, and that the condition of statement (3) is met in the case of disentailment.

In practice, one can ensure by variable renaming that no variable of X occurs in γ. The next fact says that then it suffices if entailment simplification respects the more convenient invariant:

$$\mathscr{A} \models \gamma \wedge \phi \leftrightarrow \gamma \wedge \psi.$$

This is the invariant respected by our system (cf. Proposition 5.4).

Proposition 4.2. *Let* $X \cap \mathscr{V}(\gamma) = \emptyset$. *Then,*
(1) *if* $\mathscr{A} \models \gamma \wedge \phi \leftrightarrow \gamma \wedge \psi$, *then* $\gamma \models_{\mathscr{A}} \exists X\psi$,
(2) $\gamma \models_{\mathscr{A}} \neg\, \exists X\phi$ *if and only if* $\gamma \wedge \phi$ *is unsatisfiable in* $\mathscr{A}$.

That is, the conjunction $\gamma \wedge \phi$ is satisfiable if and only if γ either entails $\exists X\phi$, or it does not determine $\exists X\phi$.

The independence of negative constraints [9, 16, 17] is an important property of constraint systems. If it holds, simplification of conjunctions of positive and negative constraints can be reduced to entailment simplification of conjunctions of positive constraints. In order to see why, observe that $\gamma \wedge \neg\phi_1 \wedge \cdots \wedge \phi_n$ is unsatisfiable if and only if γ entails $\phi_1 \vee \cdots \vee \phi_n$.

To define the independence property, we assume that a constraint system is a pair consisting of a structure $\mathscr{A}$ and a set of basic constraints. From basic constraints one can build more complex constraints using the connectives and quantifiers of predicate logic. We say that a constraint system satisfies the *independence property* if:

$$\gamma \models_{\mathscr{A}} \exists X_1 \phi_1 \vee \cdots \vee \exists X_n \phi_n \text{ if and only if } \exists i\colon \gamma \models_{\mathscr{A}} \exists X_i \phi_i$$

for all basic constraints $\gamma, \phi_1, \ldots, \phi_n$ and all finite sets of variables $X_1, \ldots, X_n$.

Proposition 4.3. *If a constraint system satisfies the independence property, then the following statements hold* (γ, ϕ *and* $\phi_1, \ldots, \phi_n$ *are basic constraints*):

(1) $\gamma \wedge \neg \exists X_1 \phi_1 \wedge \cdots \wedge \neg \exists X_n \phi_n$ *unsatisfiable in* $\mathscr{A}$ *if and only if* $\exists i\colon \gamma \models_{\mathscr{A}} X_i \phi_i$;

(2) *if* $\gamma \wedge \neg \exists X_1 \phi_1 \wedge \cdots \wedge \neg \exists X_n \phi_n$ *is satisfiable in* $\mathscr{A}$, *then* $\gamma \wedge \neg \exists X_1 \phi_1 \wedge \cdots \wedge \neg \exists X_n \phi_n \models_{\mathscr{A}} \exists X \phi$ *if and only if* $\gamma \models_{\mathscr{A}} \exists X \phi$.

5. Entailment simplification

We will now use the general setting of the previous section for the specific case of feature tree constraints. Throughout this section we assume that γ is a solved constraint and X is a finite set of variables not occurring in γ. We will call γ the *context*, the variables in X *local*, and all other variables *global*. Relative simplification is always carried out with respect to the context.

If T is a theory and ϕ and ψ are possibly open formulae, we write $\phi \models_T \psi$ (read: ϕ entails ψ in T) if $\tilde{\forall}(\phi \rightarrow \psi)$ holds in T.

The next theorem expresses the same observations stated before Theorem 3.7 regarding disentailment rather than satisfiability.

Theorem 5.1. *For every basic constraint* ϕ, *the following equivalences hold*:

$$\gamma \models_{\mathscr{T}} \neg \exists X \phi \text{ if and only if } \gamma \models_{FT_0} \neg \exists X \phi \text{ if and only if } \gamma \models_{FT} \neg \exists X \phi.$$

Proof. Implication "2⇒3" holds since $FT_0 \subseteq FT$. Implication "3⇒1" holds since $\mathscr{T}$ is a model of FT. To show implication "1⇒2", suppose $\gamma \models_{\mathscr{T}} \neg \exists X \phi$. Then we know by Proposition 4.2 that $\gamma \wedge \phi$ is unsatisfiable in $\mathscr{T}$. Thus, we know by Theorem 3.7 that $\gamma \wedge \phi$ is unsatisfiable in every model of FT_0. Hence, we know by Proposition 4.2 that $\gamma \models_{FT_0} \neg \exists X \phi$. □

For every basic constraint ϕ and every variable x we define

$$\phi x := \begin{cases} y, & \text{if } x \doteq y \in \phi \text{ and } x \text{ is eliminated,} \\ x, & \text{otherwise.} \end{cases}$$

A basic constraint ϕ is *X-oriented* if $x \doteq y \in \phi$ always implies $x \in X$ or $y \notin X$. A basic constraint ϕ is *pivoted* if $x \doteq y \in \phi$ implies that x is eliminated in ϕ (and then y is a "pivot").

The following *entailment simplification rules* simplify basic constraints to basic constraints with respect to a context γ and local variables X.

$$(1)\qquad \frac{xfu \wedge \phi}{u \doteq v \wedge \phi} \quad yfv \in \gamma \wedge \phi,\ \phi y = x$$

$$(2)\qquad \frac{\phi}{\phi u \doteq \phi v \wedge \phi} \quad \begin{cases} xfu \wedge yfv \subseteq \gamma, \\ \phi x = \phi y,\ \ \phi u \neq \phi v, \\ \phi X\text{-}oriented\ and\ pivoted \end{cases}$$

$$(3)\qquad \frac{\phi}{\perp} \quad Ax \wedge By \subseteq \gamma \wedge \phi,\ \phi x = \phi y,\ A \neq B$$

$$(4)\qquad \frac{Ax \wedge \phi}{\phi} \quad Ay \in \gamma \wedge \phi,\ \phi y = x$$

$$(5)\qquad \frac{x \doteq y \wedge \phi}{x \doteq y \wedge \phi[x \leftarrow y]} \quad \begin{cases} x \neq y,\ \ x \in y, x \in \mathscr{V}(\phi), \\ (x \in X \text{ or } y \notin X) \end{cases}$$

$$(6)\qquad \frac{x \doteq y \wedge \phi}{y \doteq x \wedge \phi} \quad x \notin X,\ y \in X$$

$$(7)\qquad \frac{\phi}{\phi[x \leftarrow y]} \quad x \doteq y \in \gamma,\ x \in \mathscr{V}(\phi)$$

$$(8)\qquad \frac{x \doteq x \wedge \phi}{\phi}$$

We say that a basic constraint ϕ *simplifies* to a constraint ϕ *with respect to* γ *and* X if $\phi = \psi$ or ϕ simplifies to ψ in finitely many steps each licensed by one of the eight simplification rules given above. The notions of *normal* and *normal form with respect to* γ are defined accordingly.

Example 5.2. Assume, in the context of functions in LIFE [6] (the case of guarded Horn clauses [18] is quite similar), that a function *fun* is defined in the form $fun(z, z) \rightarrow \ldots$, and that it is called as $fun(x[f \Rightarrow u : A], y[f \Rightarrow v : B])$. That is, the actual parameter pair of feature descriptions $(x[f \Rightarrow u : A], y[f \Rightarrow v : B])$ has to be tested upon matching of (and incompatibility with) the formal parameter pair (z, z). (This is in order to know whether that function call fires, fails, or *residuates*.) As shown in [24, 5], this corresponds to testing whether the *context* $\gamma = xfu \wedge yfv \wedge Au \wedge Bv$ entails the *guard* $\exists z(x \doteq z \wedge y \doteq z)$.

Let $X = \{z\}$. Then we have the following simplification chain with respect to γ and X:

$$x \doteq z \wedge y \doteq z$$

$$\Rightarrow_{\gamma, X} \quad z \doteq x \wedge y \doteq z \qquad \text{by rule E6}$$

$$\Rightarrow_{\gamma,X}\ z \doteq x \wedge y \doteq x \qquad \text{by rule E5}$$

$$\Rightarrow_{\gamma,X}\ u \doteq v \wedge z \doteq x \wedge y \doteq x \qquad \text{by rule E2}$$

$$\Rightarrow_{\gamma,X}\ \perp \qquad \text{by rule E3}$$

Let us now take as context $\tilde{\gamma} = xfu \wedge yfv \wedge Au$. Then $\tilde{\phi} = u \doteq v \wedge z \doteq x \wedge y \doteq x$ is normal with respect to $\tilde{\gamma}$ and X. We shall see that this normal form tells us that $\tilde{\gamma}$ does not determine $\tilde{\phi}$. If $\tilde{\gamma}$ gets strengthened either to $\tilde{\gamma} \wedge Bv$ (as above), or to $\tilde{\gamma} \wedge x \doteq y$, then the strengthened context does determine: it disentails in the first and entails in the second case. The basic normal form of $\tilde{\gamma} \wedge x \doteq y$ is $yfu \wedge Au \wedge v \doteq u \wedge x \doteq y$; with respect to this context $\tilde{\phi}$ simplifies to $z \doteq y$.

In the previous example, $\phi = z \doteq x \wedge y \doteq x$ simplifies to $\phi_1 = u \doteq v \wedge z \doteq x \wedge y \doteq x$ with respect to $\gamma = xfu \wedge yfv \wedge Au \wedge Bv$ and $X = \{z\}$. This corresponds to a basic simplification as follows:

$$\begin{array}{lll}
 & \gamma \wedge \phi & \\
= xfu \wedge yfv \wedge Au \wedge Bv & \wedge z \doteq x \wedge y \doteq x & \\
\Rightarrow xfu \wedge xfv \wedge Au \wedge Bv & \wedge z \doteq x \wedge y \doteq x & \text{(by rule B4)} \\
\Rightarrow xfv \wedge Au \wedge Bv \wedge u \doteq v & \wedge z \doteq x \wedge y \doteq x & \text{(by rule B1)} \\
= & \gamma' \wedge \phi'_1 &
\end{array}$$

We observe that $\gamma \wedge \phi_1$ is equal to $\gamma' \wedge \phi'_1$, modulo renaming y by $\phi_1 y = x$ and u by $\phi_1 u = v$, and modulo the repetition of xfv.

Lemma 5.3. *Let ϕ simplify to ϕ_1 with respect to γ and X, not using rule E6 (in an entailment simplification step). Then $\gamma \wedge \phi$ simplifies to some $\gamma' \wedge \phi'_1$ which is equal to $\gamma \wedge \phi_1$ up to variable renaming and repetition of conjuncts.*

Proof. Clearly, each entailment simplification rule, except for E6, corresponds directly to a basic simplification rule (namely, E1 and E2 to B1, E3 to B2, E4 to B3, E5 and E7 to B4, and E8 to B5).

If the application of the entailment simplification rule to ϕ relies on a condition of the form $\phi x = y$ or $\phi x = \phi y$, where $x \neq \phi x$ or $y \neq \phi y$, then $x \doteq \phi x \in \phi$ or $y \doteq \phi y \in \phi$, and rule B4 is first applied to $\gamma \wedge \phi$, eliminating x by ϕx(y by ϕy).

When comparing $\gamma \wedge \phi_1$ and $\gamma' \wedge \phi'_1$, renamings take account of these variable eliminations. Note that, if the rule applied to ϕ is E2, then γ' has one feature constraint xfv less than γ – which, after renaming, has a repetition of exactly this constraint. □

Proposition 5.4. *If ϕ simplifies to ψ with respect to γ and X, then $\gamma \wedge \phi$ and $\gamma \wedge \psi$ are equivalent in every model of FT_0.*

Proof. Follows from Lemma 5.3 and Proposition 3.2.

Proposition 5.5. *The entailment simplification rules are terminating, provided γ and X are fixed.*

Proof. First we strengthen the statement by weakening the applicability conditions $\phi y = x$ in rules E1 and E4 to $\phi y = \phi x$. Then from Lemma 5.3 follows: (*) Each entailment simplification rule applies to ϕ_1 with respect to γ and X if and only if it applies to ϕ'_1 with respect to γ' and X – except possibly for E5, when the corresponding variable has already been eliminated in an "extra" basic simplification step.

If γ' has one conjunct of the form xfu less than γ, then (*) still holds; regarding a new application of E2 this is ensured by its (therefore so complicated ...) applicability condition.

With Condition (*), it is possible to prove by induction on n: For every entailment simplification chain $\phi, \phi_1, \ldots, \phi_n$ with respect to γ and X, there exists a "basic plus rule E6" simplification chain $\gamma \wedge \phi, \gamma_1 \wedge \phi'_1, \ldots, \gamma_{n+k} \wedge \phi'_{n+k}$, where $k \geqslant 0$ is the number of "extra" variable elimination steps. Since, according to Proposition 3.3, basic simplification chains are finite, so are entailment simplification chains. □

So far we know that we can compute for any basic constraint ϕ a normal form ψ with respect to γ and X by applying the simplification rules as long as they are applicable. Although the normal form ψ may not be unique, we know that $\gamma \wedge \phi$ and $\gamma \wedge \psi$ are equivalent in every model of FT_0.

Proposition 5.6. *For every basic constraint ϕ one can compute a normal form ψ with respect to γ and X. Every such normal form ψ satisfies: $\gamma \models_{\mathscr{T}} \exists X\phi$ if and only if $\gamma \models_{\mathscr{T}} \exists X\psi$, and $\gamma \models_{FT} \exists X\phi$ if and only if $\gamma \models_{FT} \exists X\psi$.*

Proof. Follows from Propositions 5.4, 5.5, 4.2 and 4.1. □

In the following we will show that from the entailment normal form ψ of ϕ with respect to γ it is easy to tell whether we have entailment, disentailment or neither. Moreover, the basic normal form of $\gamma \wedge \phi$ is exactly $\gamma \wedge \psi$ in the first case (and in the second, where $\gamma \wedge \bot = \bot$), and "almost" in the third case (cf. Lemma 5.3).

Proposition 5.7. *A basic constraint $\phi \neq \bot$ is normal with respect to γ and X if and only if the following conditions are satisfied:*

(1) *ϕ is solved, X-oriented, and contains no variable that is bound by γ;*
(2) *if $\phi x = y$ and $xfu \in \gamma$, then $yfv \notin \phi$ for every v;*
(3) *if $\phi x = \phi y$ and $xfu \in \gamma$ and $yfv \in \gamma$, then $\phi u = \phi v$;*
(4) *if $\phi x = y$ and $Ax \in \gamma$, then $By \notin \phi$ for every B;*
(5) *if $\phi x = \phi y$ and $Ax \in \gamma$ and $By \in \gamma$, then $A = B$.*

Lemma 5.8. *If $\phi \neq \bot$ is normal with respect to γ and X, then $\gamma \wedge \phi$ is satisfiable in every model of FT.*

Proof. Let $\phi \neq \bot$ be normal with respect to γ and X. Furthermore, let $\gamma = \gamma_N \wedge \gamma_G$ and $\phi = \phi_N \wedge \phi_G$ be the unique decompositions into normalizer and graph. Since the variables bound by γ_N occur neither in γ_G nor in ϕ, it suffices to show that $\gamma_G \wedge \phi_N \wedge \phi_G$ is satisfiable in every model of FT.

Let $\phi_N(\gamma_G)$ be the basic constraint that is obtained from γ_G by applying all bindings of ϕ_N. Then $\gamma_G \wedge \phi_N \wedge \phi_G$ is equivalent to $\phi_N \wedge \phi_N(\gamma_G) \wedge \phi_G$ and no variable bound by ϕ_N occurs in $\phi_N(\gamma_G) \wedge \phi_G$. Hence, it suffices to show that $\phi_N(\gamma_G) \wedge \phi_G$ is satisfiable in every model of FT. With conditions 2–5 of the preceding proposition it is easy to see that $\phi_N(\gamma_G) \wedge \phi_G$ is a solved clause. Hence, we know by axiom scheme Ax3 that $\phi_N(\gamma_G) \wedge \phi_G$ is satisfiable in every model of FT. □

The following theorem states that relative simplification yields a decision procedure for disentailment of basic constraints.

Theorem 5.9 (Disentailment). *Let ψ be a normal form of ϕ with respect to γ and X. Then $\gamma \models_{\mathcal{T}} \neg \exists X \phi$ if and only if $\psi = \bot$.*

Proof. Suppose $\psi = \bot$. Then $\gamma \models_{\mathcal{T}} \neg \exists X \psi$ and hence $\gamma \models_{\mathcal{T}} \neg \exists X \phi$ by Proposition 5.7. To show the other direction, suppose $\gamma \models_{\mathcal{T}} \neg \exists X \phi$. Then $\gamma \models_{\mathcal{T}} \neg \exists X \psi$ by Proposition 5.7 and hence $\gamma \wedge \psi$ unsatisfiable in $\mathcal{T}$ by Proposition 4.2. Since $\mathcal{T}$ is a model of FT (Theorem 2.1), we know by the preceding lemma that $\psi = \bot$ (since ψ is assumed to be normal). □

We say that a variable x is *dependent* in a solved constraint ϕ if ϕ contains a constraint of the form Ax, xfy or $x \doteq y$. (Recall that equations are ordered; thus y is not dependent in the constraint $x \doteq y$.) We use $\mathcal{DV}(\phi)$ to denote the set of all variables that are dependent in a solved constraint ϕ.

In the following we will assume that the underlying signature $\mathcal{S} \uplus \mathcal{F}$ has at least one sort and at least one feature that does not occur in the constraints under consideration. This assumption is certainly satisfied if the signature has infinitely many sorts and infinitely many features.

Lemma 5.10. *Let $\phi_1, \ldots, \phi_n$ be basic constraints different from $\bot$, and $X_1, \ldots, X_n$ be finite sets of variables disjoint from $\mathcal{V}(\gamma)$. Moreover, for every $i = 1, \ldots, n$, let ϕ_i be normal with respect to γ and X_i, and let ϕ_i have a dependent variable that is not in X_i. Then $\gamma \wedge \neg \exists X_1 \phi_1 \wedge \cdots \wedge \neg \exists X_n \phi_n$ is satisfiable in every model of FT.*

Proof. Let $\gamma = \gamma_N \wedge \gamma_G$ be the unique decomposition of γ into normalizer and graph. Since the variables bound by γ_N occur neither in γ_G nor in any ϕ_i, it suffices to show that $\gamma_G \wedge \neg\exists X_1 \phi_1 \wedge \cdots \wedge \neg\exists X_n \phi_n$ is satisfiable in every model of FT. Thus, it suffices to exhibit a solved clause δ such that $\gamma_G \subseteq \delta$ and, for every $i = 1, \ldots, n$, $\mathscr{V}(\delta)$ is disjoint with X_i and $\delta \wedge \phi_i$ is unsatisfiable in every model of FT.

Without loss of generality we can assume that every X_i is disjoint with $\mathscr{V}(\gamma)$ and $\mathscr{V}(\phi_j) - X_j$ for all j. Hence, we can pick in every ϕ_i a dependent variable x_i such that $x_i \notin X_j$ for any j.

Let $z_1, \ldots, z_k$ be all variables that occur on either side of equation $x_i \doteq y \in \phi_i$, $i = 1, \ldots, n$ (recall that x_i is fixed for i). None of these variables occurs in any X_j, since every ϕ_i is X_i-oriented. Next we fix a feature g and a sort B such that neither occurs in γ or any ϕ_i.

Now δ is obtained from γ by adding constraints as follows: if $Ax_i \in \phi_i$, then add Bx_i; if $x_i f y \in \phi_i$, then add $x_i f\uparrow$; to enforce that the variables $z_1, \ldots, z_k$ are pairwise distinct, add:

$$z_k g z_{k-1} \wedge \cdots \wedge z_2 g z_1 \wedge z_1 g\uparrow.$$

It is straightforward to verify that these additions to γ yield a solved clause δ as required. □

Proposition 5.11. *If ϕ is solved and $\mathscr{DV}(\phi) \subseteq X$, then $FT \models \tilde{\forall}\exists X\phi$.*

Proof. Let $\phi = \phi_N \wedge \phi_G$ be the decomposition of ϕ in normalizer and graph. Since every variable bound by ϕ is in X, it suffices to show that $\tilde{\forall}\exists X\phi_G$ is a consequence of FT. This follows immediately from axiom scheme Ax3 since ϕ_G is a solved clause. □

The following theorem states that relative simplification yields a decision procedure for entailment of basic constraints.

Theorem 5.12 (Entailment). *Let ψ be a normal form of ϕ with respect to γ and X. Then $\gamma \models_{\mathscr{T}} \exists X\phi$ if and only if $\psi \neq \bot$ and $\mathscr{DV}(\psi) \subseteq X$.*

Proof. Suppose $\gamma \models_{\mathscr{T}} \exists X\phi$. Then we know $\gamma \models_{\mathscr{T}} \exists X\psi$ by Proposition 5.6, and thus $\gamma \wedge \neg\exists X\psi$ is unsatisfiable in $\mathscr{T}$. Since γ is solved, we know that γ is satisfiable in $\mathscr{T}$ and hence that $\gamma \wedge \exists X\psi$ is satisfiable in $\mathscr{T}$. Thus $\psi \neq \bot$. Since $\gamma \wedge \neg\exists X\psi$ is unsatisfiable in $\mathscr{T}$ and $\mathscr{T}$ is a model of FT, we know by Lemma 5.10 that $\mathscr{DV}(\psi) \subseteq X$.

To show the other direction, suppose $\psi \neq \bot$ and $\mathscr{DV}(\psi) \subseteq X$. Then $FT \models \tilde{\forall}\exists X\psi$ by Proposition 5.11, and hence $\mathscr{T} \models \tilde{\forall}\exists X\psi$. Thus $\gamma \models_{\mathscr{T}} \exists X\psi$, and hence $\gamma \models_{\mathscr{T}} \exists X\phi$ by Proposition 5.6. □

The next theorem shows that it does not matter whether entailment of basic constraints is interpreted in the algebraic semantics (i.e., in the feature tree structure $\mathscr{T}$) or in the logical semantics (given by the axioms of FT). Now, of course, the third axiom is necessary (take, for example, as context the true constraint $\top$).

Theorem 7. *Let ϕ be a basic constraint. Then $\gamma \models_{\mathscr{T}} \exists X\phi$ if and only if $\gamma \models_{FT} \exists X\phi$.*

Proof. One direction holds since $\mathscr{T}$ is a model of FT. To show the other direction, suppose $\gamma \models_{\mathscr{T}} \exists X\phi$. Without loss of generality we can assume that ϕ is normal with respect to γ and X. Hence, we know by Theorem 5.12 that $\phi \neq \perp$ and $\mathscr{DV}(\psi) \subseteq X$. Thus, $FT \models \tilde{\forall} \exists X\phi$ by Proposition 5.11 and hence $\gamma \models_{FT} \exists X\phi$. $\square$

We finally show that our constraint system enjoys the property which allows one to solve conjunctions of negative constraints through relative simplification.

Theorem 5.14 (independence). *Let $\phi_1, \ldots, \phi_n$ be basic constraints, and $X_1, \ldots, X_n$ be finite sets of variables. Then,*

$$\gamma \models_{\mathscr{T}} \exists X_1 \phi_1 \vee \cdots \vee \exists X_n \phi_n \text{ if and only if } \exists i\colon \gamma \models_{\mathscr{T}} \exists X_i \phi_i.$$

Proof. To show the nontrivial direction, suppose $\gamma \models_{\mathscr{T}} \exists X_1 \phi_1 \vee \cdots \vee \exists X_n \phi_n$. Without loss of generality we can assume that, for all $i = 1, \ldots, n$, X_i is disjoint from $\mathscr{V}(\gamma)$, ϕ_i is normal with respect to γ and X_i, and $\phi_i \neq \perp$. Since $\gamma \wedge \neg\exists X_1 \phi_1 \wedge \cdots \wedge \neg\exists X_n \phi_n$ is unsatisfiable in $\mathscr{T}$ and $\mathscr{T}$ is a model of FT, we know by Lemma 5.10 that $\mathscr{DV}(\phi_k) \subseteq X_k$ for some k. Hence, $\gamma \models_{\mathscr{T}} \exists X_k \phi_k$ by Theorem 5.12. $\square$

6. Conclusion

We have presented a constraint system FT for logic programming providing a universal data structure based on rational feature trees. FT accommodates record-like descriptions. We think that these are superior to the constructor-based descriptions of Herbrand in that they allow expressing partial knowledge in a more flexible way.

The declarative semantics of FT is specified both algebraically (the feature tree structure $\mathscr{T}$) and logically (the first-order theory FT given by three axiom schemes).

The operational semantics for FT is given by an incremental constraint simplification system, which can check satisfiability of and entailment between constraints. Since FT satisfies the independence property, the simplification system can also check satisfiability of conjunctions of positive and negative constraints.

We see four directions for further research.

First, FT should be strengthened such that it subsumes the expressivity of rational constructor trees [9, 10]. As is, FT cannot express that x is a tree having direct subtrees at exactly the features $f_1, \ldots, f_n$. It turns out that the system CFT [26] obtained from FT by adding the primitive constraint

$$x\{f_1, \ldots, f_n\}$$

(x has direct subtrees at exactly the features $f_1 \ldots, f_n$) has the same nice properties as FT. In contrast to FT, CFT can express constructor constraints; for instance, the constructor constraint $x \doteq A(y,z)$ can be expressed equivalently as $Ax \wedge x\{1,2\} \wedge x1y \wedge x2z$, if we assume that A is a sort and the numbers 1,2 are features.

Second, it seems attractive to extend FT such that it can accommodate a sort lattice as used in [1, 3, 4, 5, 25]. One possibility to do this is to assume a partial order $\leqslant$ on sorts and replace sort constraints Ax with *quasi-sort constraints* $[A]x$ whose declarative semantics is given as

$$[A]x \equiv \bigvee_{B \leqslant A} Bx.$$

Given the assumption that the sort ordering $\leqslant$ has greatest lower bounds if lower bounds exist, it seems that the results and the simplification system given for FT carry over with minor changes.

Third, the worst-case complexity of entailment of basic constraints checking in FT should be established. We conjecture it to be quasi-linear in the size of γ and ϕ, provided the available features (finitely many) are fixed a priori.

Lastly, implementation techniques for FT at the level of the Warren abstract machine [2] need to be developed.

Acknowledgment

Gert Smolka was supported in part by the Bundesminister für Forschung und Technologie under contract ITW 9105. We wish to thank Kathleen Milsted and Ralf Treinen for their constructive comments. We are also grateful to the anonymous referee who provided *à propos* comments and suggestions in an extensive review. Last but not least, we thank yet and again Jean-Christophe Patat, PRL's librarian, for his careful proofreading.

References

[1] H. Aït-Kaci, An algebraic semantics approach to the effective resolution of type equations, *Theoret. Comput. Sci.* **45** (1986) 293–351.

[2] H. Aït-Kaci, *Warren's Abstract Machine: A Tutorial Reconstruction* (The MIT Press, Cambridge, MA, 1991).

[3] H. Aït-Kaci and R. Nasr, LOGIN: A logic programming language with built-in inheritance, *J. Logic Programming* **3** (1986) 185–215.

[4] H. Aït-Kaci and R. Nasr, Integrating logic and functional programming, *Lisp Symbolic Computation* **2** (1989) 51–89.

[5] H. Aït-Kaci and A. Podelski, Towards a meaning of LIFE, in: J. Maluszyński and M. Wirsing, eds., in: *Proc. 3rd International Symposium on Programming Language Implementation and Logic Programming*, Passau, Germany, Lecture Notes in Computer Science, Vol. 528 (Springer, Berlin, 1991) 255–274.

[6] H. Aït-Kaci and A. Podelski, Functions as passive constraints in LIFE, PRL Research Report 13, Digital Equipment Corporation, Paris Research Laboratory, Rueil-Malmaison, France, June 1991 (Revised, November 1992).

[7] R. Backofen and G. Smolka, A complete and decidable feature theory, DFKI Research Report RR-30-92, German Research Center for Artificial Intelligence, Saarbrücken, Germany, 1992.

[8] B. Carpenter, *The Logic of Typed Feature Structures*, Cambridge Tracts in Theoretical Computer Science, Vol. 32 (Cambridge University Press, Cambridge, UK, 1992).

[9] A. Colmerauer, Equations and inequations on finite and infinite trees, in: *Proc. 2nd Internat. Conf. on Fifth Generation Computer Systems* (1984) 85–99.

[10] A. Colmerauer, H. Kanoui and M.V. Caneghem, Prolog, theoretical principles and current trends, *Tech. Sci. Informatics* 2 (1983) 255–292.

[11] S. Haridi and S. Janson, Kernel Andorra Prolog and its computation model, in: D. Warren and P. Szeredi, eds., in: *Logic Programming, Proc. 7th Internat. Conf.* (The MIT Press, Cambridge, MA, 1990), 31–48.

[12] J. Jaffar and J.L. Lassez, Constraint logic programming, in: *Proc. 14th ACM Symp. on Principles of Programming Languages*, Munich, Germany (1987) 111–119.

[13] M. Johnson, *Attribute-Value Logic and the Theory of Grammar*. CSLI Lecture Notes 16 (Center for the Study of Language and Information, Stanford University, CA, 1988).

[14] R.M. Kaplan and J. Bresnan, Lexical-functional grammar: a formal system for grammatical representation, in: J. Bresnan, ed., *The Mental Representation of Grammatical Relations* (The MIT Press, Cambridge, MA, 1982) 173–281.

[15] M. Kay, Functional grammar, in: *Proc. 5th Ann. Meeting of the Berkeley Linguistics Society* (Berkeley Linguistics Society, Berkeley, CA, 1979).

[16] J.L. Lassez, M. Maher and K. Marriot, Unification revisited, in: Jack Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988).

[17] J.L. Lassez and K. McAloon, A constraint sequent calculus, in: *5th Ann. IEEE Symp. on Logic in Computer Science* (1990) 52–61.

[18] M. Maher, Logic semantics for a class of committed-choice programs, J.-L. Lassez, ed., in: *Proc. 4th Internat. Conf. Logic Programming* (The MIT Press, Cambridge, MA, 1987) 858–876.

[19] K. Mukai, Partially specified terms in logic programming for linguistic analysis, in: *Proc. 6th Internat. Conf. on Fifth Generation Computer Systems* (1988).

[20] K. Mukai, Constraint logic programming and the unification of information. Ph.D. Thesis, Tokyo Institute of Technology, Tokyo, Japan, 1991.

[21] M. Nivat, Elements of a theory of tree codes, in: M. Nivat and A. Podelski, eds., *Tree Automata (Advances and Open Problems)* (Elsevier, Amsterdam, NE, 1992).

[22] W.C. Rounds and R.T. Kasper, A complete logical calculus for record structures representing linguistic information, in: *Proc. 1st IEEE Symp. on Logic in Computer Science*, Boston, MA (1986) 38–43.

[23] V. Saraswat and M. Rinard, Concurrent constraint programming, in: *Proc. 7th Ann. ACM Symp. on Principles of Programming Languages*, San Francisco, CA (1990) 232–245.

[24] G. Smolka, Feature constraint logics for unification grammars, *J. Logic Programming*, 12 (1992) 51–87.

[25] G. Smolka and H. Aït-Kaci, Inheritance hierarchies: semantics and unification, *J. Symbolic Comput.* 7 (1989) 343–370.

[26] G. Smolka and R. Treinen, Relative simplification for and independence of CFT. Draft, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, to appear.