Records for Logic Programming

Gert Smolka and Ralf Treinen

German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D 66123 Saarbrücken, Germany
{smolka, treinen}@dfki.uni-sb.de

Abstract

CFT is a new constraint system providing records as logical data structure for constraint (logic) programming. It can be seen as a generalization of the rational tree system employed in Prolog II, where finer-grained constraints are used, and where subtrees are identified by keywords rather than by position.

CFT is defined by a first-order structure consisting of so-called feature trees. Feature trees generalize the ordinary trees corresponding to first-order terms by having their edges labeled with field names called features. The mathematical semantics given by the feature tree structure is complemented with a logical semantics given by five axiom schemes, which we conjecture to comprise a complete axiomatization of the feature tree structure.

We present a decision method for CFT, which decides entailment and disentailment between possibly existentially quantified constraints. Since CFT satisfies the independence property, our decision method can also be employed for checking the satisfiability of conjunctions of positive and negative constraints. This includes quantified negative constraints such as $\forall y \forall z(x \neq f(y,z))$.

The paper also presents an idealized abstract machine processing negative and positive constraints incrementally. We argue that an optimized version of the machine can decide satisfiability and entailment in quasi-linear time.

Journal of Logic Programming, 18(3), April 1994, pp. 229–258.

A short version appeared in Krzysztof Apt, ed., Proceedings of the Joint International Conference and Symposium on Logic Programming, Washington D.C., 9–12 Nov 1992, MIT Press, pp. 240–254.

A previous version appeared as DFKI Research Report RR-92-23, Aug 1992.

Contents

1 Introduction

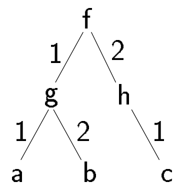
Records are an important data structure in programming languages. They appeared first with imperative languages such as ALGOL 68 and Pascal, but are now also present in modern functional languages such as SML. A major reason for providing records is the fact that they serve as the canonical data structure for expressing object-oriented programming techniques.

In this paper we will show that records can be incorporated into logic programming in a straightforward and natural manner. We will model records with a constraint system CFT, which can serve as the basis of future constraint (logic) programming languages.[1] Since CFT is a conservative extension of Prolog II's rational tree system [12, 13], the familiar term notation can still be used.[2]

1.1 Records are Feature Trees

We model records as feature trees [7, 9]. A feature tree (examples are shown in Figure 1) is a tree whose edges are labeled with symbols called features, and whose nodes are labeled with symbols called sorts. The features labeling the edges correspond to the field names of records. As one would expect, the labeling with features must be deterministic, that is, every direct subtree of a feature tree is uniquely identified by the feature of the edge leading to it. Feature trees without subtrees model atomic values (e.g., numbers). Feature trees may be finite or infinite. Infinite feature trees provide for the convenient representation of cyclic data structures. The last example in Figure 1 gives a finite graph representation of an infinite feature tree, which may arise as the representation of the recursive type equation $\mathsf{nat} = \mathsf{0} + \mathsf{s}(\mathsf{nat})$.

A ground term, say $\mathsf{f}(\mathsf{g}(\mathsf{a}, \mathsf{b}), \mathsf{h}(\mathsf{c}))$, can be seen as a feature tree whose nodes are labeled with function symbols and whose arcs are labeled with numbers:



Thus the trees corresponding to first-order terms are in fact feature trees

[1] Such languages can, for instance, be obtained as instances of the frameworks CLP [18], ALPS [25] and CC [27].

[2] We haven chosen to admit infinite trees so that cyclic data structures can be represented directly. However, a set-up admitting only finite trees as in the original Horn clause model is also possible.

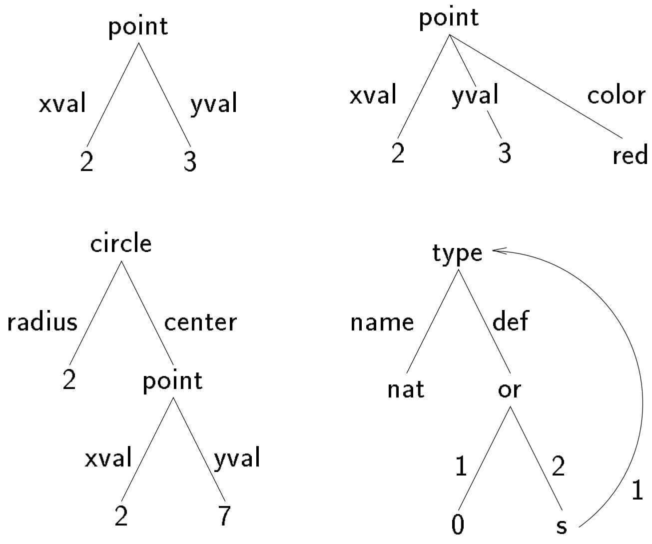


Figure 1: Examples of Feature Trees.

observing certain restrictions (e.g., the features departing from a node must be consecutive positive integers).

1.2 Record Descriptions

In CFT, records (i.e., feature trees) are described by first-order formulae. To this purpose, we set up a first-order structure $\mathcal{T}$ (CFT's standard model) whose universe is the set of all feature trees (over given alphabets of features and sorts), and whose descriptive primitives are defined as follows:

- Every sort symbol A is taken as a unary predicate, where a *sort constraint* $x\colon A$ holds if and only if the root of the tree x is labeled with A.
- Every feature symbol f is taken as a binary predicate, where a *feature constraint* $x[f]y$ holds if and only if the tree x has the direct subtree y at feature f.
- Every finite set F of features is taken as a unary predicate, where an *arity constraint* xF holds if and only if the tree x has direct subtrees exactly at the features appearing in F.

The descriptions or constraints of CFT are now exactly the first-order formulae obtained from the primitive forms specified above, where we include equations "$x = y$" between variables.

A feature constraint $x[f]y$ corresponds to field selection for records. A more familiar notation for $x[f]y$ might be $y = x.f$ or $y = x[f]$. Note that the field selection function "$x.f$" is partial since not every record has a field f.

Next we note that the familiar term notation can still be used in CFT if a little syntactic sugar is provided. For instance, the equational constraint

$$\mathsf{X} \quad = \quad \mathsf{point}(\mathsf{Y}, \mathsf{Z})$$

employing the binary constructor point translates into the conjunction

$$\mathsf{X}\colon \mathsf{point} \;\wedge\; \mathsf{X}\{1,2\} \;\wedge\; \mathsf{X}[1]\mathsf{Y} \;\wedge\; \mathsf{X}[2]\mathsf{Z}.$$

Note that constructors and features are dual in the sense that features are argument selectors for constructors.

CFT can also express constructors that identify their arguments by keywords rather than by position. For instance, the equation

$$\mathsf{P} \quad = \quad \mathsf{point}(\mathsf{xval}\colon \mathsf{X},\ \mathsf{yval}\colon \mathsf{Y},\ \mathsf{color}\colon \mathsf{Z})$$

can be taken as an abbreviation for

$$\mathsf{P}\colon \mathsf{point} \;\wedge\; \mathsf{P}\{\mathsf{xval}, \mathsf{yval}, \mathsf{color}\} \;\wedge\; \mathsf{P}[\mathsf{xval}]\mathsf{X} \;\wedge\; \mathsf{P}[\mathsf{yval}]\mathsf{Y} \;\wedge\; \mathsf{P}[\mathsf{color}]\mathsf{Z}.$$

Using nesting, which can be expressed in CFT with existentially quantified auxiliary variables, we can give the following description of the infinite feature tree shown in Figure 1:

$$\mathsf{X} \quad = \quad \mathsf{type}(\mathsf{name}\colon \mathsf{nat},\ \mathsf{def}\colon \mathsf{or}(0, \mathsf{s}(\mathsf{X}))).$$

Compared to the standard tree constraint systems, the major expressive flexibility provided by CFT is the possibility to access a feature without saying anything about the existence of other features. The constraint

$$\mathsf{X}[\mathsf{color}]\mathsf{Y}$$

says that X must have a color field whose value is Y, but nothing else. Hence we can express properties of the color of X without knowing whether X is a circle, triangle, car or something else. Using constructor constraints, we would have to write a disjunction

$$\mathsf{X} = \mathsf{circle}(\ldots, \mathsf{Y}, \ldots) \;\vee\; \mathsf{X} = \mathsf{triangle}(\ldots, \mathsf{Y}, \ldots) \;\vee\; \ldots$$

which means that we have to know statically which alternatives are possible dynamically. Moreover, disjunctions are expensive computationally. In contrast, feature constraints like X[color]Y allow for efficient constraint simplification, as we will see in this paper.

Descriptions leaving the arity of a record open are also essential for knowledge representation, where a description like

$$\mathsf{X}\colon \mathsf{person}[\mathsf{father}\colon \mathsf{Y},\ \mathsf{employer}\colon \mathsf{Y}]$$

should not disallow other features. In CFT this description can be expressed by simply *not* imposing an arity constraint:

$$\mathsf{X}\colon \mathsf{person} \;\wedge\; \mathsf{X}[\mathsf{father}]\mathsf{Y} \;\wedge\; \mathsf{X}[\mathsf{employer}]\mathsf{Y}.$$

1.3 Constraint Simplification

The major technical contribution of this paper is the presentation and verification of a constraint simplification method for CFT. This method provides for incremental entailment and disentailment checking as it is needed for advanced constraint programming frameworks [25, 27]. We show how the decision method can be realized as an abstract machine processing positive and negative constraints incrementally.

To state our technical results precisely, let a simple constraint be a formula in the fragment

$$[x{:}\, A,\ x[f]y,\ xF,\ x = y,\ \bot,\ \top]_{\wedge,\exists}$$

obtained by closing the atomic formulae under conjunction and existential quantification. Let γ and ϕ be simple constraints. We give a method that decides simultaneously entailment $\gamma \models_{\mathrm{CFT}} \phi$ and disentailment $\gamma \models_{\mathrm{CFT}} \neg\phi$. This method can be implemented by an incremental algorithm having quasi-linear complexity, provided the features possibly occurring in γ and ϕ are restricted a priori to some finite set. We also prove that CFT satisfies the independence property,[3] that is,

$$\gamma \models_{\mathrm{CFT}} \phi_1 \vee \ldots \vee \phi_n \quad \Longleftrightarrow \quad \exists i{:}\ \ \gamma \models_{\mathrm{CFT}} \phi_i.$$

Hence, our decision method can decide the satisfiability of conjunctions of positive and negative simple constraints since

$$\gamma \wedge \neg\phi_1 \wedge \ldots \wedge \neg\phi_n \models_{\mathrm{CFT}} \bot$$

is equivalent to

$$\gamma \models_{\mathrm{CFT}} \phi_1 \vee \ldots \vee \phi_n.$$

All results are obtained under the assumption that the alphabets of sorts and features are infinite.

1.4 Related Work

CFT can be viewed as the minimal combination of Colmerauer's rational tree system [12, 13] with the feature constraint system FT [7]. In fact, CFT is obtained from FT by simply adding arity constraints as new descriptive primitive. However, the addition of arity constraints requires a nontrivial extension of FT's relative simplification method [7], which can be seen from the fact that the entailment

$$x = f(x, y) \wedge y = f(y, y) \models_{\mathrm{CFT}} x = y$$

[3] Since we allow for existential quantification in simple constraints, our independence result is in fact stronger than what is usually stated in the literature [13, 23, 24]. See also the discussion at the end of Section 5.4.

holds in CFT. (It of course also holds in Colmerauer's rational tree system.)

Our operational investigations are based on congruences and normalizers of constraints, two straightforward notions providing for an elegant presentation of the results.[4] We improve on Colmerauer's [13] results for rational trees since our constraints are closed under existential quantification. For instance, our algorithm is complete for negative quantified constraints such as $\neg\exists y\exists z(x = f(y, z))$.

Feature descriptions have a long and winded history. One root are the unification grammar formalisms FUG [21] and LFG [20] developed for applications in computational linguistics (see [11] for a more recent paper in this area). Another, independent root is Aït-Kaci's ψ-term calculus [1, 2], which is the basis of several constraint programming languages [4, 5, 6]. Smolka [29] gives a unified logical view of most earlier feature formalisms and studies an expressive feature constraint logic.

Feature trees appeared only recently with the work on FT [9, 7]. To our knowledge the notion of an arity constraint is new. Carpenter's [11] extensional types are somewhat related in that they fix an arity for all elements of a type. Feature constraints with first class features have been considered in [31].

A short version of this paper not containing the proofs and the description of the abstract machine has appeared before [30].

1.5 Organization of the Paper

Section 2 gives a formal definition of the feature tree structure, thus fixing syntax and semantics of CFT. Section 3 defines a first-order theory by means of five axiom schemes, which we conjecture to be a complete axiomatization of the feature tree structure. In Section 4 we show that CFT is indeed a conservative extension of the theory of constructor trees. Section 5 presents the decision method and states its properties. The proofs follow in Section 6. Section 7 shows how the decision method can be realized as an abstract machine processing positive and negative constraints incrementally.

2 The Feature Tree Structure

This section gives a formal definition of CFT's standard model $\mathcal{T}$. $\mathcal{T}$ is a first-order structure whose universe consists of all feature trees obtainable from given alphabets of sorts and features.

[4] Huet [17] uses the related notion of "équivalence simplifiable" in his study of rational tree unification.

From now on we assume that an infinite alphabet SOR of symbols called **sorts** and an infinite alphabet FEA of symbols called **features** are given. For several results of this paper (e.g., independence) it is essential that both alphabets are infinite. The letters A, B will always denote sorts, the letters f, g will always denote features, and the letters F, G will always denote *finite* sets of features.

We also assume an infinite alphabet of variables, ranged over by the letters x, y, z. From the alphabets of sorts, features and variables we define the following first-order language with equality:

1. Every sort symbol A is a unary predicate.
2. Every feature symbol f is a binary predicate.
3. Every finite set F of features is a unary predicate, called an *arity predicate*.
4. The equality symbol $\doteq$ is a binary predicate that is always interpreted as identity.
5. There is no function symbol, and there is no predicate symbol other than the ones above.

Every formula and every structure in this paper will be taken with respect to this signature. Note that under this signature every term is a variable.

For convenience, we will write Ax, xfy and xF for $A(x)$, $f(x, y)$ and $F(x)$, respectively. (In Section 1 we have used yet another, Prolog compatible syntax: X:a for sort and X[f]Y for feature constraints.) We assume the usual connectives and quantifiers. We write $\bot$ for "false" and $\top$ for "true". We use $\tilde{\exists}\phi$ [$\tilde{\forall}\phi$] to denote the existential [universal] closure of a formula ϕ. Moreover, $\mathcal{V}(\phi)$ is taken to denote the set of all variables occurring free in a formula ϕ.

A **path** is a word (i.e., a finite, possibly empty sequence) over the set of all features. The symbol ε denotes the empty path, which satisfies $\varepsilon p = p = p\varepsilon$ for every path p. A path p is called a **prefix** of a path q, if there exists a path p' such that $pp' = q$. We use $\mathrm{FEA}^\star$ to denote the set of all paths.

A **tree domain** is a nonempty set $D \subseteq \mathrm{FEA}^\star$ that is **prefix-closed**, that is, if $pq \in D$, then $p \in D$. Note that every tree domain contains the empty path.

A **feature tree** is a partial function $\sigma\colon \mathrm{FEA}^\star \rightsquigarrow \mathrm{SOR}$ whose domain is a tree domain. The paths in the domain of a feature tree represent the nodes of the tree; the empty path represents its root. We use D_σ to denote the domain of a feature tree σ. A feature tree is called **finite** [**infinite**] if its domain is finite [infinite]. The letters σ and τ will always denote feature trees.

The **subtree** $p\sigma$ of a feature tree σ at a path $p \in D_\sigma$ is the feature tree defined (in relational notation) by:

$$p\sigma := \{(q, A) \mid (pq, A) \in \sigma\}.$$

We now define the **feature tree structure** $\mathcal{T}$ as follows:

- The universe of $\mathcal{T}$ is the set of all feature trees;
- $\sigma \in A^{\mathcal{T}}$ iff $\sigma(\varepsilon) = A$;
- $(\sigma, \tau) \in f^{\mathcal{T}}$ iff $f \in D_\sigma$ and $\tau = f\sigma$;
- $\sigma \in F^{\mathcal{T}}$ iff $D_\sigma \cap \mathrm{FEA} = F$.

Note that $\mathcal{T}$ contains all infinite feature trees, where nodes may have infinitely many features. Another option is to admit only those infinite feature trees that are rational (i.e., have only finitely many subtrees and where all nodes are finitely branching). For the results of this paper this would not make a difference. We also conjecture that the rational feature tree structure and $\mathcal{T}$ are elementarily equivalent, analogous to the situation with constructor trees [26].

3 The Theory CFT

We will now define a first-order theory CFT having the feature tree structure $\mathcal{T}$ as one of its models. All results of this paper actually hold for every model of CFT. We conjecture that CFT is a complete axiomatization of the feature tree structure $\mathcal{T}$ and expect that this can be shown with a quantifier elimination technique similar to the one used in [9].

We briefly review the notion of a theory. A **theory** is a set of closed formulae. We say that a structure $\mathcal{A}$ is a **model** of a theory T ($\mathcal{A} \models T$) if $\mathcal{A}$ satisfies each formula of T. A formula ϕ is a **consequence** of a theory T ($T \models \phi$) if $\tilde{\forall}\phi$ is valid in every model of T. A formula ϕ is **unsatisfiable** in a theory T if $\neg\phi$ is a consequence of T.

A formula ϕ **entails** a formula ψ in a structure $\mathcal{A}$ ($\phi \models_{\mathcal{A}} \psi$) if $\mathcal{A}$ satisfies $\tilde{\forall}(\phi \rightarrow \psi)$. A formula ϕ **entails** a formula ψ in a theory T ($\phi \models_T \psi$) if ϕ entails ψ in every model of T, that is, if $\phi \rightarrow \psi$ is a consequence of T. Two formulae ϕ, ψ are **equivalent** in a theory T ($\phi \models\!\!\!\!\dashv_T \psi$) if they are equivalent in every model $\mathcal{A}$ of T, that is, if $\phi \leftrightarrow \psi$ is a consequence of T. A formula ϕ **disentails** a formula ψ in a theory T if ϕ entails $\neg\psi$ in T. For convenience, we will omit the index $\emptyset$ for the empty theory, that is, write $\models$ for $\models_\emptyset$.

CFT is defined by five axiom schemes. The first four schemes are straightforward:

$$\begin{array}{lll} (S) & \tilde{\forall}\,(Ax \wedge Bx \rightarrow \bot) & \text{if } A \neq B \\ (F) & \tilde{\forall}\,(xfy \wedge xfz \rightarrow y \doteq z) & \\ (A1) & \tilde{\forall}\,(xF \wedge xfy \rightarrow \bot) & \text{if } f \notin F \\ (A2) & \tilde{\forall}\,(xF \rightarrow \exists y(xfy)) & \text{if } f \in F. \end{array}$$

The first two axiom schemes say that sorts are pairwise disjoint, and that features are functional. The last two schemes say that, if x has arity F, exactly the features $f \in F$ are defined on x.

To formulate the remaining axiom scheme, we need the notion of a determinant. A **determinant for** x is a formula

$$Ax \wedge x\{f_1, \ldots, f_n\} \wedge xf_1y_1 \wedge \ldots \wedge xf_ny_n$$

which we will write more conveniently as

$$x \doteq A(f_1\colon y_1, \ldots, f_n\colon y_n).$$

(It is understood that all the feature symbols f_i are different.) As we have pointed out before, a determinant as the one above is similar to a constructor equation $x \doteq f(y_1, \ldots, y_n)$. A **determinant for** pairwise distinct variables $x_1, \ldots, x_n$ is a conjunction

$$x_1 \doteq D_1 \;\wedge\; \ldots \;\wedge\; x_n \doteq D_n$$

of determinants for $x_1, \ldots, x_n$. If δ is a determinant, we use $\mathcal{D}(\delta)$ to denote the set of variables determined by δ. In terms of constructor tree logic this corrresponds to the systems of regular equations in [14] or to the rational solved forms in [12, 26].

The remaining axiom scheme will say that every determinant determines a unique solution for its determined variables. To this purpose we define the quantifier $\exists! x\phi$ ("there exists a unique x such that") as an abbreviation for

$$\exists x\phi \wedge \forall x, y(\phi \wedge \phi[x \leftarrow y] \rightarrow x \doteq y).$$

($\phi[x \leftarrow y]$ denotes the formula obtained from ϕ by replacing every free occurrence of x with y while possibly renaming bound variables in order to avoid capturing.) The more general form $\exists! X\phi$, where X is a finite set of variables, is defined accordingly. The quantifier $\exists!$ satisfies

$$\exists! X\phi \wedge \exists X(\phi \wedge \psi) \quad \models_{\mathcal{A}} \quad \phi \rightarrow \psi \tag{1}$$

for every structure $\mathcal{A}$ and all formulae ϕ, ψ.

Now we can state the fifth axiom scheme:

$$(D) \quad \tilde{\forall}\,(\exists!\mathcal{D}(\delta)\,\delta) \qquad \text{if } \delta \text{ is a determinant.}$$

An example of an instance of scheme (D) is:

$$\forall u, v, w\, \exists!\; x, y, z \left(\begin{array}{l} x \doteq A(f\colon v,\, g\colon y) \wedge \\ y \doteq B(f\colon x,\, g\colon z,\, h\colon u) \wedge \\ z \doteq A(f\colon w,\, g\colon y,\, h\colon z) \end{array} \right).$$

The theory CFT is the set of all sentences that can be obtained as instances of the axiom schemes (S), (F), (A1), (A2) and (D).

Proposition 3.1 *The feature tree structure $\mathcal{T}$ is a model of* CFT *. Moreover, the substructure of $\mathcal{T}$ containing only the rational feature trees is also a model of* CFT *.*

Proof. That the first four axioms schemes are satisfied is obvious. To show that $\mathcal{T}$ satisfies the fifth axiom, one assumes arbitrary feature trees for the universally quantified variables and constructs feature trees for the existentially quantified variables. $\square$

Proposition 3.2 *Let δ be a determinant and ϕ any formula. Then:*

$$\delta \models_{\mathrm{CFT}} \phi \quad \iff \quad \mathrm{CFT} \models \exists\mathcal{D}(\delta)(\delta \wedge \phi).$$

Proof. The direction "$\Rightarrow$" follows immediately from Axiom Scheme (D). The other direction follows by Axiom Scheme (D) and (1). $\square$

4 Relationship to Constructor Trees

In this section we show that the theory CFT can be seen as a conservative extension of the theory RT . Let Σ be a fixed infinite constructor signature. The axioms set RT [26] is defined by the following axiom schemes:

$(RT1)$	$\tilde{\forall}\,(f(\bar{x}) \doteq f(\bar{y}) \rightarrow \bar{x} \doteq \bar{y})$	$f \in \Sigma$
$(RT2)$	$\tilde{\forall}\,\neg(f(\bar{x}) \doteq f(\bar{y}))$	$f, g \in \Sigma, f \neq g$
$(RT3)$	$\tilde{\forall}\exists!\bar{x}\bar{x} \doteq \bar{t}$	$\bar{x} \doteq \bar{t}$ is a rational solved form

A rational solved form is a set of equations $x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n$ where all x_i are different variables and no term t_i is a variable. [26] shows that RT is a complete set of axioms.

Given Σ, we define the signature Σ^F of CFT as FEA $= \Sigma$ and SOR $= \{1, 2, \ldots\}$. We present an effective translation σ^F of an Σ-formula σ into an Σ^F-formula σ^F such that RT $\models \sigma$ iff CFT $\models \sigma^F$. Since we may assume without loss of generality that σ contains only flat equations $x \doteq f(x_1, \ldots, x_n)$, we can define the translation as the homomorphic extension of

$$[x \doteq f(x_1, \ldots, x_n)]^F \;:=\; fx \wedge x\{1, \ldots, n\} \wedge x1x_1 \wedge \ldots \wedge xnx_n \,.$$

Every Σ^F-model $\mathcal{A}$ of CFT translates into a Σ-model A^C with same domain by

$$\begin{array}{lll}(a_1, \ldots, a_n, a) \in f^{\mathcal{A}^C} & \text{iff} & a \in f^{\mathcal{A}} \text{ and } a \in \{1, \ldots, n\}^{\mathcal{A}} \text{ and} \\ & & (a, a_i) \in i^{\mathcal{A}} \text{ for every } i \in \{1, \ldots, n\} \,.\end{array}$$

By axiom scheme *(D)* of CFT , $f^{\mathcal{A}}$ is indeed a function. An easy inductive argument yields

Proposition 4.1 *For all Σ^F-models $\mathcal{A}$ with $\mathcal{A} \models$ CFT and for all Σ-formulae σ we have $(\sigma^F)^{\mathcal{A}} = \sigma^{(\mathcal{A}^C)}$ and $\mathcal{A}^C \models$ RT .*

Theorem 4.2 *For every Σ-formula σ: RT $\models \sigma$ iff CFT $\models \sigma^F$.*

Proof. For the first direction, let $\mathcal{A}$ be a model of CFT . By Proposition 4.2, $\mathcal{A}^C$ is a model of RT , hence $\mathcal{A}^C \models \sigma$, and $\mathcal{A} \models \sigma^F$ follows from Proposition 4.2.

For the other direction, let CFT $\models \sigma^F$. Since RT is complete and consistent, either RT $\models \sigma$ or RT $\models \neg\sigma$ holds. By assumption $\mathcal{T} \models \sigma^F$, hence $\mathcal{T}^C \models \sigma$ by Proposition 4.2. Since $\mathcal{T}^C$ is a model of RT , we conclude RT $\models \sigma$. □

5 The Decision Method

In this section we develop in several steps a method for deciding simultaneously entailment and disentailment in CFT. The proofs of the results stated here will follow in the next section.

A **basic constraint** is a possibly empty conjunction of atomic constraints (i.e., Ax, xfy, xF, $x \doteq y$). The empty conjunction is the formula $\top$. We assume that the conjunction of formulae is associative and commutative, and that it satisfies $\phi \wedge \top = \phi$. We can thus see a basic constraint equivalently as a finite multiset of atomic constraints, where $\wedge$ corresponds to multiset union and $\top$ to the empty multiset. For basic constraints ϕ, ψ, we will write $\psi \subseteq \phi$ (or $\psi \in \phi$, if ψ is an atomic constraint) if there exists a basic constraint ψ' such that $\psi \wedge \psi' = \phi$.

Let γ, ϕ be basic constraints and X, Y be finite sets of variables. We will eventually arrive at an incremental method for deciding

$$\begin{array}{lll}\exists Y \gamma & \models_{\mathrm{CFT}} & \exists X \phi \\ \exists Y \gamma & \models_{\mathrm{CFT}} & \neg\exists X \phi\end{array}$$

simultaneously. We will also see that the equivalences

$$\exists Y\gamma \models_{\mathrm{CFT}} \exists X\phi \iff \exists Y\gamma \models_{\mathcal{A}} \exists X\phi \tag{2}$$
$$\exists Y\gamma \models_{\mathrm{CFT}} \neg\exists X\phi \iff \exists Y\gamma \models_{\mathcal{A}} \neg\exists X\phi \tag{3}$$

hold for every model $\mathcal{A}$ of the theory CFT.

We say that a basic constraint **clashes** if it simplifies to $\perp$ with one of the following rules:

(SCl) $\quad \dfrac{Ax \wedge Bx \wedge \phi}{\perp} \quad A \neq B$

(ACl) $\quad \dfrac{xF \wedge xG \wedge \phi}{\perp} \quad F \neq G$

(FCl) $\quad \dfrac{xF \wedge xfy \wedge \phi}{\perp} \quad f \notin F$

We call a basic constraint **clash-free** if it does not clash.

Proposition 5.1 *A clashing basic constraint is unsatisfiable in* CFT*.*

Proof. For rule (SCl) the claim follows from axiom scheme (S), for rule (FCl) from axiom scheme (A1), and for rule (ACl) the claim follows from schemes (A1) and (A2). □

Consider the basic constraint

$$x \doteq y \wedge x\,f\,x' \wedge y\,f\,y' \wedge A\,x' \wedge B\,y', \tag{4}$$

where A, B are distinct sorts. Clearly, this constraint is unsatisfiable in CFT: If there was a solution, it would have to identify x' and y' (since features are functional), which is impossible since A and B are disjoint. This suggests that a constraint simplification method must infer all equalities between variables that are induced by the functionality of features (axiom scheme (F)). This observation leads us to the central notions of congruences and normalizers of constraints.

5.1 Congruences and Normalizers

We call an equivalence relation $\approx$ between variables a **congruence** of a basic constraint ϕ if:

- if $x \doteq y \in \phi$, then $x \approx y$;
- if xfy, $x'fy' \in \phi$ and $x \approx x'$, then $y \approx y'$.

It is easy to see that the set of congruences of a basic constraint is closed under intersection. Since the equivalence relation identifying all variables is a congruence of every basic constraint, every basic constraint has a least congruence. We use $\langle\phi\rangle$ to denote the **least congruence** of a basic constraint ϕ. Note that we have the equivalence $x \langle\phi\rangle\ y \iff \phi \models x \doteq y$ in the special case where ϕ is a conjunction of equations.

The least congruence of the basic constraint (4) has two nontrivial equivalence classes: $\{x, y\}$ and $\{x', y'\}$.

Technically, it will be most convenient to represent congruences as idempotent substitutions mapping variables to variables. We call a substitution θ a **normalizer** of an equivalence relation $\approx$ on the set of all variables if

1. θ maps variables to variables;
2. θ is idempotent (that is, $\theta\theta = \theta$);
3. $\theta x = \theta y$ if and only if $x \approx y$ (for all variables x, y).

Given $\approx$, we can obtain a normalizer of $\approx$ by choosing a canonical member for every equivalence class and mapping every variable to the canonical member of its class.

Let θ be a substitution. We use $\mathcal{D}om(\theta)$ (the **domain of** θ) to denote the set of all variables x such that $\theta x \neq x$. A substitution is called **finite** if its domain is finite. A finite substitution θ with the domain $\mathcal{D}om(\theta) = \{x_1, \ldots, x_n\}$ can be represented as an equation system

$$x_1 \doteq \theta x_1 \wedge \ldots \wedge x_n \doteq \theta x_n.$$

For convenience, we will simply use θ to denote this formula. Now, if θ is a substitution and ϕ is a quantifier-free formula, we have

$$\theta \wedge \phi \quad \models\!\!\!\!\!\!\;=\!\!| \quad \theta \wedge \theta\phi,$$

where the application of θ to ϕ is defined as one would expect.

We call a substitution θ a **normalizer** of a basic constraint ϕ if θ is a normalizer of the least congruence of ϕ. Every basic constraint ϕ has a finite normalizer since its least congruence can only identify variables occurring in ϕ.

The least congruence of the basic constraint (4) has two nonsingleton equivalence classes: $\{x, y\}$ and $\{x', y'\}$. Hence the constraint (4) has 4 normalizers, each representing a different choice for the normal forms of identified variables. One possible normalizer is the substitution $\{x \mapsto y, x' \mapsto y'\}$.

Let θ be a normalizer of ϕ. Then $\langle\theta\rangle = \langle\phi\rangle$ and $x \langle\theta\rangle\ y \iff \theta x = \theta y$ for all variables x, y ($\langle\theta\rangle$ is the least congruence of the equational representation of θ).

Let ϕ and ψ be basic constraints. We write $\phi - \psi$ for the constraint that is obtained from ϕ by deleting all constraints occurring in ψ. We write $\overline{\phi}$ for the formula obtained from ϕ by deleting all equations "$x \doteq y$". We call a basic constraint ϕ **equation-complete** if $\langle\phi\rangle = \langle\phi - \overline{\phi}\rangle$ (that is, the least congruence of ϕ coincides with the least congruence of the equations contained in ϕ).

Theorem 5.2 *Let $\mathcal{A}$ be a model of* CFT *, ϕ a basic constraint, and θ a normalizer of ϕ. Then:*

1. *ϕ is unsatisfiable in $\mathcal{A}$ if and only if $\theta\overline{\phi}$ clashes;*
2. *$\phi \models\!\mid_{\mathrm{CFT}}\ \theta \wedge \theta\overline{\phi}$ and $\theta \wedge \theta\overline{\phi}$ is equation-complete.*

The first statement of the theorem gives us a method for deciding the satisfiability of basic constraints, provided we have a method for computing normalizers. The second statement gives us a solved form for satisfiable basic constraints. Since the first statement implies that a basic constraint is satisfiable in one model of CFT if and only if it is satisfiable in every model of CFT , we know that the theory CFT is satisfaction complete [18].

Let ϕ be the basic constraint (4) and θ be the normalizer $\{x \mapsto y, x' \mapsto y'\}$. Then $\theta\overline{\phi}$ is the clashing constraint

$$y\ f\ y' \wedge y\ f\ y' \wedge A\ y' \wedge B\ y'.$$

The following simplification rules for basic constraints provide a method for computing normalizers:

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

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

$$\text{(Elim)} \qquad \frac{x \doteq y \wedge \phi}{x \doteq y \wedge \phi[x \leftarrow y]} \quad x \neq y,\ x \in \mathcal{V}(\phi)$$

($\phi[x \leftarrow y]$ denotes the formula obtained from ϕ by replacing every free occurrence of x with y while possibly renaming bound variables in order to avoid capture.) Each of these rules is an equivalence transformation for CFT (rule (Cong) corresponds to axiom scheme (F)). It is also easy to see that the rules preserve the congruences of a constraint, and hence its least congruence. Furthermore, the rules are terminating. Hence we can compute for every basic constraint ϕ a normal form that has exactly the same normalizers as ϕ. The next proposition says that normal constraints exhibit a normalizer (a constraint is normal with respect to a set of rules if none of the rules applies to it):

Proposition 5.3 *Let ϕ be a basic constraint that is normal with respect to the rules (Triv), (Cong) and (Elim). Then the unique substitution θ such that $\phi = \theta \wedge \overline{\phi}$ is a normalizer of ϕ satisfying $\overline{\phi} = \theta\overline{\phi}$.*

5.2 Entailment without $\exists$

Next we will give a method for deciding entailment $\gamma \models_{\mathrm{CFT}} \phi$ between basic constraints. The constraint γ will be required to have a special form called saturated graph.

A basic constraint γ is called a **graph** if it is clash-free, contains no equation, and satisfies $xfy \in \gamma \wedge xfz \in \gamma \Rightarrow y = z$. Hence a clash-free basic constraint γ not containing equations is a graph if and only if the identity substitution is the only normalizer of γ.

A basic constraint ϕ is called **saturated** if for every arity constraint $xF \in \phi$ and every feature $f \in F$ there exists a feature constraint $xfy \in \phi$.

We call a variable x **determined** in a basic constraint ϕ if ϕ contains a determinant for x (see Section 3). We use $\mathcal{D}(\phi)$ to denote the set of all variables determined in ϕ. We say that an equation $x \doteq y$ is **determined** in ϕ if x and y are both determined in ϕ.

The next theorem says that in a satisfiable and equation-complete basic constraint we can delete determined equations without losing information.

Theorem 5.4 (Determined Equations) *Let η be a conjunction of equations and ϕ be a basic constraint such that $\eta \wedge \phi$ is equation-complete and satisfiable in* CFT*. Then $\eta \wedge \phi \models\!\mid_{\mathrm{CFT}} \phi$, provided every equation in η is determined in ϕ.*

Theorem 5.5 *Let $\mathcal{A}$ be a model of* CFT*, γ a saturated graph, ϕ a basic constraint, and let θ be a normalizer of $\gamma \wedge \phi$. Then:*

1. *$\gamma \models_{\mathcal{A}} \neg\phi$ if and only if $\theta(\gamma \wedge \overline{\phi})$ clashes;*
2. *$\gamma \models_{\mathcal{A}} \phi$ if and only if*
   (a) *$\theta(\gamma \wedge \overline{\phi})$ is clash-free and*
   (b) *$\theta\overline{\phi} \subseteq \theta\gamma$ and*
   (c) *every equation in θ is determined in γ.*

The first statement follows immediately from Theorem 5.2 (since for every structure $\mathcal{A}$, $\gamma \models_{\mathcal{A}} \neg\phi$ iff $\gamma \wedge \phi$ is unsatisfiable in $\mathcal{A}$). The second statement is nontrivial. Note that deciding entailment and disentailment is straightforward once a normalizer is computed.

To see an example, let us verify

$$x \doteq A(f\colon x,\, g\colon y) \,\wedge\, y \doteq A(f\colon y,\, g\colon y) \quad \models_{\mathrm{CFT}} \quad x \doteq y \tag{5}$$

with the method provided by Theorem 5.5. Without syntactic sugar we have

$$Ax \,\wedge\, x\{f,g\} \,\wedge\, xfx \,\wedge\, xgy \,\wedge\, Ay \,\wedge\, y\{f,g\} \,\wedge\, yfy \wedge ygy \quad \models_{\mathrm{CFT}} \quad x \doteq y.$$

The left-hand side γ is in fact a saturated graph. If we apply the simplification rule (Elim) to $\gamma \wedge \phi$ (ϕ is the right-hand side $x \doteq y$), we obtain (up to duplicates) the normal and clash-free constraint

$$x \doteq y \,\wedge\, Ay \,\wedge\, y\{f,g\} \,\wedge\, yfy \,\wedge\, ygy.$$

Hence $\theta := \{x \mapsto y\}$ is a normalizer of $\gamma \wedge \phi$. Since $\overline{\phi} = \top$ and $x \doteq y$ is determined in γ, we know by Theorem 5.5 that γ entails ϕ in every model of CFT .

5.3 Entailment with $\exists$

We now extend Theorem 5.5 to the general case $\exists Y \gamma \models_{\mathrm{CFT}} \exists X \phi$.

First we note that, after possibly renaming quantified variables, we have

$$\exists Y \gamma \models_{\mathrm{CFT}} \exists X \phi \quad \iff \quad \gamma \models_{\mathrm{CFT}} \exists X \phi.$$

Hence it suffices to consider the case where only the right-hand side has existential quantifiers.

Next we will see that we can assume without loss of generality that γ is a saturated graph. Given a basic constraint γ, we can first apply the simplification rules (Triv), (Cong) and (Elim) and obtain an equivalent normal form $\theta \wedge \gamma'$, where θ is a normalizer and γ' either clashes or is a graph. If γ' clashes, then $\gamma \models_{\mathrm{CFT}} \exists X \phi$ trivially holds. Otherwise, we can assume without loss of generality that $\theta \wedge \gamma'$ and X have no variable in common. Thus we have

$$\gamma \models_{\mathrm{CFT}} \exists X \phi \iff \theta \wedge \gamma' \models_{\mathrm{CFT}} \exists X \phi \iff \gamma' \models_{\mathrm{CFT}} \exists X (\theta\phi)$$

since θ is idempotent and $\theta\gamma' = \gamma'$. Now we know by axiom scheme (A2) that there exists a saturated graph γ'' such that $\gamma' \models\!\!\mid_{\mathrm{CFT}} \exists Y \gamma''$ for some set Y of new variables. Thus we have

$$\gamma \models_{\mathrm{CFT}} \exists X \phi \iff \exists Y \gamma'' \models_{\mathrm{CFT}} \exists X (\theta\phi) \iff \gamma'' \models_{\mathrm{CFT}} \exists X (\theta\phi).$$

Hence it suffices to exhibit a decision method for the case $\gamma \models_{\mathrm{CFT}} \exists X \phi$, where γ is a saturated graph and X is disjoint from $\mathcal{V}(\gamma)$.

We say that a variable x is **constrained** in a basic constraint ϕ if ϕ contains an atomic constraint of the form $x \doteq y$, Ax, xF or xfy. We write $\mathcal{C}(\phi)$ for the set of all variables that are constrained in a basic constraint ϕ. The basic constraint (4), for instance, constrains the variables x, y, x' and y'.

In the following X will be a finite set of variables. We write $-X$ for the complement of X. We call a normalizer θ X**-oriented** if $\theta(-X) \subseteq -X$. Given an equivalence relation between variables, we can obtain an X-oriented normalizer by choosing the canonical member of a class from $-X$ whenever the class contains an element that is not in X. To compute X-oriented normalizers, it suffices to add the rule

$$\text{(Orient)} \quad \frac{y \doteq x \wedge \phi}{x \doteq y \wedge \phi} \quad \text{if } x \in X \text{ and } y \notin X$$

to the simplification rules (Triv), (Cong) and (Elim). With this additional rule normal forms will always exhibit an X-oriented normalizer.

The **restriction** $\theta|_X$ of a normalizer θ to a set X of variables is the substitution that agrees with θ on X and is the identity on $-X$.

Theorem 5.6 (Entailment) *Let $\mathcal{A}$ be a model of* CFT*, γ a saturated graph, ϕ a basic constraint, X a finite set of variables not occurring in γ, and let θ be an X-oriented normalizer of $\gamma \wedge \phi$. Then:*

1. *$\gamma \models_{\mathcal{A}} \neg\exists X \phi$ if and only if $\theta(\gamma \wedge \overline{\phi})$ clashes;*
2. *$\gamma \models_{\mathcal{A}} \exists X \phi$ if and only if*
   - *(a) $\theta(\gamma \wedge \overline{\phi})$ is clash-free and*
   - *(b) $\mathcal{C}(\theta\overline{\phi} - \theta\gamma) \subseteq X$ and*
   - *(c) every equation in $\theta|_{-X}$ is determined in γ.*

Theorem 5.5 is obtained from the Entailment Theorem as the special case where $X = \emptyset$. Since the criteria of Theorem 5.6 do not depend on the particular model $\mathcal{A}$, we obtain the claims (2) and (3) stated at the beginning of this section.

5.4 Independence

Theorem 5.7 (Independence) *Let $\phi, \phi_1, \ldots, \phi_n$ be basic constraints and $X_1, \ldots, X_n$ be finite sets of variables. Then:*

$$\phi \models_{\mathcal{A}} \exists X_1 \phi_1 \vee \ldots \vee \exists X_n \phi_n \quad \Longleftrightarrow \quad \exists i: \quad \phi \models_{\mathcal{A}} \exists X_i \phi_i$$

for every model $\mathcal{A}$ of CFT*.*

The Independence Theorem does not hold for finite alphabets of sorts and features. For finitely many sorts $A_1, \ldots, A_n$ we have

$$\top \quad \models_{\mathcal{T}} \quad A_1 x \vee \ldots \vee A_n x,$$

and for finitely many features $f_1, \ldots, f_n$ we have

$$\top \quad \models_{\mathcal{T}} \quad x\{\} \vee \exists y(x f_1 y) \vee \ldots \vee \exists y(x f_n y).$$

Since we allow for existential quantification, our Independence Theorem is stronger than what is usually stated in the literature [13, 23, 24]. Independence of existentially quantified constraints has been shown for a class of Boolean constraint systems in [16] and for finite and rational constructor trees over an infinite signature in [26]. In fact, independence for existentially quantified constraints over finite or rational constructor trees does not hold if the alphabet of constructors is finite. To see this, note that the disjunction

$$\exists \overline{y}_1 (x = f_1(\overline{y}_1)) \;\vee\; \ldots \;\vee\; \exists \overline{y}_n (x = f_n(\overline{y}_n))$$

is valid if there are no other constructors but $f_1, \ldots, f_n$.

6 The Proofs

We now give the proofs of the results stated in the preceding section.

6.1 Congruences and Normalizers

We first study the properties of the simplification system given by the rules (Triv), (Cong), (Elim), and (Orient). Since the rule (Orient) is not applicable for $X = \emptyset$, the subsystem (Triv), (Cong), (Elim) is in fact a special case of the full system.

A basic constraint is called a **graph constraint** if it contains no equation. Note that a graph constraint is a graph if and only if it is equation-complete and clash-free.

We say that a congruence $\approx$ **contains** an equation $x \doteq y$ if $x \approx y$.

Proposition 6.1 *Let $\theta \wedge \gamma$ be a normal form of a basic constraint ϕ with respect to the rules (Triv), (Cong), (Elim), (Orient), where θ is a set of equations and where γ is a graph constraint. Then:*

1. $\phi \models\!\mid_{\mathrm{CFT}} \theta \wedge \gamma$;
2. *θ is an X-oriented normalizer of ϕ;*

3. $\gamma = \theta\gamma$.

Proof. It is obvious that the rules perform equivalence transformations in CFT, so ϕ and $\theta \wedge \gamma$ are equivalent in CFT.

The rule (Elim) forces all variables occurring at the left side of an equation to occur only once. Hence, θ is an idempotent substitution, and $\theta(-X) \subseteq -X$ by (Orient). Since $\mathcal{D}om(\theta)$ is disjoint from $\mathcal{V}(\gamma)$, the third claim follows.

To prove that θ is a normalizer of ϕ, it remains to show that $\langle\theta\rangle$ is the least congruence of ϕ. To this end, we first show that the simplification rules preserve congruences. So assume ϕ simplifies to ψ with one of the rules. We have to show that an equivalence relation between variables is a congruence of ϕ iff it is a congruence of ψ. For the rules (Triv) and (Orient) this is trivial.

If $\approx$ is a congruence of $xfy \wedge xfz \wedge \phi$, then it is as well a congruence of $xfz \wedge \phi$, and $\approx$ contains $y \doteq z$ since θ is a congruence of $xfy \wedge xfz$. If $\approx$ is a congruence of $y \doteq z \wedge xfz \wedge \phi$, then $y \approx z$, hence $\approx$ is a congruence of $xfy \wedge xfz \wedge \phi$. This proves that application of (Cong) preserves congruences.

For the case of (Elim), every congruence of $x \doteq y \wedge \phi$ is a congruence of $x \doteq y \wedge \phi[x \leftarrow y]$, and vice versa, since in either case every congruence must contain $x \doteq y$.

Now we show by contradiction that $\langle\theta\rangle$ is a congruence of $\theta \wedge \gamma$. By definition, $\langle\theta\rangle$ contains all equations of θ. Hence, if $\langle\theta\rangle$ is not a congruence of $\theta \wedge \gamma$, then there must be $xfy, x'fy' \in \gamma$ with $x \langle\theta\rangle x'$, $y \neq y'$ and not $y \langle\theta\rangle y'$.

If $x = x'$, then (Cong) applies, which contradicts the normal form assumption. If x and x' are different variables, then at least one of them is contained in $\mathcal{D}om(\theta)$ since $\theta x = \theta x'$. Hence (Elim) applies, which again contradicts the normal form assumption.

Since every congruence of $\theta \wedge \gamma$ must contain θ, we conclude that $\langle\theta\rangle$ is in fact the least congruence of $\theta \wedge \gamma$. Since the simplification rules preserve congruences, $\langle\theta\rangle$ is the least congruence of ϕ. □

Proof of Proposition 5.3. Follows from Proposition 6.1. □

We say that a variable x is **eliminated** in a basic constraint ϕ if ϕ contains an equation $x \doteq y$ and x occurs in ϕ only once.

Proposition 6.2 *The simplification system consisting of (Triv), (Cong), (Elim) and (Orient) is terminating.*

Proof. Obviously, there cannot be a derivation using (Triv) or (Cong) infinitely often. Hence, it suffices to show that the rules (Elim) and (Orient) terminate.

(Elim) and (Orient) do not introduce new variables. For a given basic constraint ϕ, consider the lexicographically ordered cross-product (see, e.g., [15]) of the following measures:

1. the number of variables in $X \cap \mathcal{V}(\phi)$ that are not eliminated in ϕ,
2. the number of equations $x \doteq y$ such that $x \notin X$,
3. the number of variables in $-X \cap \mathcal{V}(\phi)$ that are not eliminated in ϕ.

Application of the rule (Elim) with $x \in X$ decreases the first component in this lexicographic ordering, while application of (Orient) does not increase the first component but decreases the second. Application of (Elim) with $x \notin X$ does not increase the first or second component and decreases the third. $\square$

Proposition 6.3 *For every normalizer θ of a basic constraint ϕ:*

$$\phi \models\!\!\!\!\!=\!\!\!\!\mid_{\mathrm{CFT}} \ \theta \wedge \theta\overline{\phi}.$$

Proof. It is easy to show that two normalizers of a basic constraint, when considered as formulas, are equivalent in every structure. By Proposition 6.2 and Proposition 6.1 there is a normalizer ρ of ϕ satisfying $\phi \models_{\mathrm{CFT}} \rho$, hence

$$\phi \models_{\mathrm{CFT}} \theta.$$

Let η be the equational part of ϕ. Then

$$\theta \models_{\mathrm{CFT}} \eta$$

since the least congruence of ϕ, that is $\langle\theta\rangle$, contains all equations of ϕ. Hence $\phi \models\!\!\!\!\!=\!\!\!\!\mid_{\mathrm{CFT}} \ \theta \wedge \phi \models\!\!\!\!\!=\!\!\!\!\mid_{\mathrm{CFT}} \ \theta \wedge \eta \wedge \overline{\phi} \models\!\!\!\!\!=\!\!\!\!\mid_{\mathrm{CFT}} \ \theta \wedge \overline{\phi} \models\!\!\!\!\!=\!\!\!\!\mid_{\mathrm{CFT}} \ \theta \wedge \theta\overline{\phi}$. $\square$

Proposition 6.4 *If θ is a normalizer of a congruence of a basic constraint ϕ, then $\theta\overline{\phi}$ either clashes or is a graph.*

Proof. Obvious. $\square$

We say that the feature f is **realized** for a variable x in a basic constraint ϕ if ϕ contains a feature constraint xfy for some variable y.

Proposition 6.5 *Let ϕ be a graph and let $\mathcal{C}(\phi) \subseteq X$. Then* CFT $\models \tilde{\forall}\exists X \phi$.

Proof. Since ϕ is a graph, the following implications hold:

1. $Ax, Bx \in \phi \Rightarrow A = B$;

2. $xF, xfy \in \phi \Rightarrow f \in F$;

3. $xF, xG \in \phi \Rightarrow F = G$;

4. $xfy, xfz \in \phi \Rightarrow y = z$.

Furthermore we may assume without loss of generality that ϕ does not contain any multiple occurrence of an atomic constraint. We will construct a determinant $\delta \supseteq \phi$ with $\mathcal{D}(\delta) = X$. Then

$$\mathrm{CFT} \models \tilde{\forall}\exists X \delta$$

by axiom (D), which proves the claim since $\delta \models \phi$.

For each $x \in X$, let F_x denote the set of feature symbols that are realized for x in ϕ. We define the determinant δ by adding to ϕ for each variable $x \in X$ the following atomic constraints:

- Ax, provided there is no sort constraint for x in ϕ;
- xF_x, provided there is no arity constraint for x in ϕ;
- xfx, provided there is an arity constraint $xF \in \phi$ and $f \in F$ is not realized for x in ϕ. □

Lemma 6.6 *Let $\mathcal{A}$ be a model of* CFT *and θ a normalizer of the basic constraint ϕ. Then the following statements are equivalent:*

1. *$\theta\overline{\phi}$ is clash-free;*
2. *ϕ is satisfiable in every model of* CFT *;*
3. *ϕ is satisfiable in $\mathcal{A}$.*

Proof. By Proposition 6.3, $\phi \models\!\!\!\!\!\!=|_{\mathrm{CFT}} \ \theta \wedge \theta\overline{\phi}$. Since θ is an idempotent substitution, $\theta \wedge \theta\overline{\phi}$ is satisfiable in a structure iff $\theta\overline{\phi}$ is satisfiable in this structure.

Hence for any model $\mathcal{B}$ of CFT, ϕ is satisfiable in $\mathcal{B}$ iff $\theta\overline{\phi}$ is. By Proposition 6.4, $\theta\overline{\phi}$ is either a graph or clashes. Hence, if $\theta\overline{\phi}$ is clash-free, then (2) and (3) follow by Proposition 6.5. Otherwise (2) and (3) do not hold by Proposition 5.1. □

Proof of Theorem 5.2. The first statement of Theorem 5.2 follows immediately from Lemma 6.6. The second statement is a consequence of Proposition 6.3. □

Proposition 6.7 *Let ψ, ϕ be basic constraints, X a finite set of variables not occurring in ψ, and θ a normalizer of $\psi \wedge \phi$. Then*

$$\psi \models_{\mathrm{CFT}} \exists X\, \phi \leftrightarrow \exists X\, (\theta \wedge \theta\overline{\phi}).$$

Proof. The claim follows from the following equivalence:

$$\begin{array}{rlll}
\psi \wedge \exists X\, \phi & \models\!\!\!\dashv_{\mathrm{CFT}} & \psi \wedge \exists X\, (\psi \wedge \phi) & \text{since } X \text{ disjoint from } \mathcal{V}(\psi) \\
& \models\!\!\!\dashv_{\mathrm{CFT}} & \psi \wedge \exists X\, (\theta \wedge \theta\overline{\psi} \wedge \theta\overline{\phi}) & \text{by Proposition 6.3} \\
& \models\!\!\!\dashv_{\mathrm{CFT}} & \psi \wedge \exists X\, (\theta \wedge \theta\overline{\phi}) & \text{since } \theta \wedge \psi \models \theta\overline{\psi}.
\end{array}$$

□

Proposition 6.8 *Let $\mathcal{A}$ be a model of* CFT *, ψ, ϕ basic constraints, θ a normalizer of $\phi \wedge \psi$ and X a finite set of variables disjoint from $\mathcal{V}(\psi)$. Then the following statements are equivalent:*

1. $\psi \models_{\mathcal{A}} \neg\exists X\, \phi$ *;*
2. $\psi \models_{\mathcal{A}} \neg\exists X\, (\theta \wedge \theta\overline{\phi})$ *;*
3. $\psi \models_{\mathcal{A}} \neg\exists X\, (\theta \wedge \overline{\phi})$ *;*
4. $\theta(\overline{\psi} \wedge \overline{\phi})$ *clashes;*
5. $\theta(\psi \wedge \overline{\phi})$ *clashes.*

Proof. (1) and (2) are equivalent by Proposition 6.7, and the equivalence of (2) and (3) is a basic property of substitutions. The equivalence of (1) and (4) can be seen as follows:

$$\begin{array}{rll}
\psi \models_{\mathcal{A}} \neg\exists X\, \phi & \Leftrightarrow & \mathcal{A} \models \tilde{\forall}(\psi \rightarrow \neg\exists X\, \phi) \\
& \Leftrightarrow & \mathcal{A} \models \tilde{\forall}\neg\exists X\, (\psi \wedge \phi) \\
& \Leftrightarrow & \mathcal{A} \models \neg\tilde{\exists}(\psi \wedge \phi) \\
& \Leftrightarrow & \theta(\overline{\psi} \wedge \overline{\phi}) \text{ clashes} \qquad\qquad \text{by Lemma 6.6.}
\end{array}$$

Finally, (4) and (5) are equivalent, since by definition of normalizers $\theta(\overline{\psi} \wedge \overline{\phi})$ and $\theta(\psi \wedge \overline{\phi})$ differ only by trivial equations $x \doteq x$. □

6.2 Determined Equations

We use $\mathcal{V}(\theta)$ to denote the set of all variables occurring in the equational representation of a substitution θ.

Lemma 6.9 *Let γ be a graph constraint and let θ be a normalizer of some congruence of γ. If $\theta\gamma$ is clash-free and if $\mathcal{V}(\theta) \subseteq \mathcal{D}(\gamma)$, then*

$$\gamma \models_{\mathrm{CFT}} \theta.$$

Proof. Suppose $\theta\gamma$ is clash-free and $\mathcal{V}(\theta) \subseteq \mathcal{D}(\gamma)$. Then γ contains a determinant δ such that $\mathcal{D}(\delta) = \mathcal{V}(\theta)$. Hence it suffices to prove that

$$\delta \models_{\mathrm{CFT}} \theta. \tag{6}$$

Since $\theta\delta$ is clash-free, we know by Proposition 6.4 that $\theta\delta$ is a graph. Since $\mathcal{C}(\theta\delta) \subseteq \mathcal{D}(\delta) \cup \mathcal{V}(\theta) = \mathcal{D}(\delta)$, we know by Proposition 6.5 that CFT $\models \tilde{\forall}\exists\mathcal{D}(\delta)\, \theta\delta$. Hence, since θ is idempotent

$$\mathrm{CFT} \models \tilde{\forall}\exists\mathcal{D}(\delta)\ (\theta \wedge \delta).$$

Thus we have (6) by Proposition 3.2. □

Lemma 6.10 *Let η, η' be sets of equations, and let γ be a graph constraint such that $\eta \wedge \eta' \wedge \gamma$ is equation-complete and satisfiable in* CFT *. If $\mathcal{V}(\eta') \subseteq \mathcal{D}(\gamma)$, then*

$$\eta \wedge \gamma \models_{\mathrm{CFT}} \eta'.$$

Proof. Let θ be a normalizer of η. First note that, since θ is an idempotent substitution,

$$\theta \wedge \phi \models_{\mathcal{A}} \psi \quad \Longleftrightarrow \quad \theta\phi \models_{\mathcal{A}} \theta\psi \tag{7}$$

for any structure $\mathcal{A}$ and basic constraints ϕ, ψ. Since $\eta \models\!\!\!\dashv \theta$, we know by our assumptions that $\theta \wedge \eta' \wedge \gamma$ is equation-complete and satisfiable in CFT . We first show that

$$\theta\eta' \wedge \theta\gamma \text{ is equation-complete.} \tag{8}$$

Assume that $\theta x f \theta x', \theta y f \theta y' \in \theta\gamma$ and $\theta x\ \langle\theta\eta'\rangle\ \theta y$. By (7) we have $\theta \wedge \eta' \models x \doteq y$. Since $xfx', yfy' \in \gamma$ and $\eta' \wedge \theta \wedge \gamma$ is equation complete, we have $x'\ \langle\theta \wedge \eta'\rangle\ y'$ and thus $\theta x'\ \langle\theta\eta'\rangle\ \theta y'$ by (7), which completes the proof of (8).

Now let θ' be a normalizer of $\theta\eta'$. As a consequence of (8), θ' is normalizer of some congruence of $\theta\gamma$. Since $\theta \wedge \eta' \wedge \gamma$ is satisfiable in CFT , $\theta' \wedge \theta\gamma$ is satisfiable in CFT and we know by Lemma 6.6 that $\theta'\theta\gamma$ is clash-free. Furthermore, $\mathcal{V}(\theta') = \mathcal{V}(\theta\eta') \subseteq \mathcal{D}(\theta\gamma)$, since by assumption $\mathcal{V}(\eta') \subseteq \mathcal{D}(\gamma)$. Hence

$$\theta\gamma \models_{\mathrm{CFT}} \theta'$$

by Lemma 6.9. Since we have $\eta \models\!\!\!\dashv \theta$ and $\theta' \models\!\!\!\dashv \theta\eta'$, we obtain

$$\eta \wedge \gamma \models_{\mathrm{CFT}} \eta'$$

using (7). □

Proof of Theorem 5.4. Follows immediately from Lemma 6.10. □

6.3 Entailment and Independence

The next lemma is the key to the proofs of the Entailment and the Independence Theorems of Section 5.

Lemma 6.11 *Let γ be a saturated graph, and for every i, $1 \leq i \leq n$, ϕ_i a basic constraint, X_i a finite set of variables disjoint from $\mathcal{V}(\gamma)$, and θ_i an X_i-oriented normalizer of $\gamma \wedge \phi_i$. If for each i*

$$\mathcal{C}(\theta_i\overline{\phi}_i - \theta_i\gamma) \not\subseteq X_i \quad \textit{or} \quad \mathcal{V}(\theta_i|_{-X_i}) \not\subseteq \mathcal{D}(\gamma),$$

then

$$\text{CFT} \models \tilde{\exists}(\gamma \wedge \neg\exists X_1(\theta_1 \wedge \overline{\phi}_1) \wedge \ldots \wedge \neg\exists X_n(\theta_n \wedge \overline{\phi}_n)).$$

Proof. We may assume without loss of generality that $\theta_i(\gamma\wedge\overline{\phi}_i)$ is clash-free for all i, since otherwise by Proposition 6.8

$$\gamma \wedge \neg\exists X_i(\theta_i \wedge \overline{\phi}_i) \models\!\!\!\mid_{\text{CFT}} \gamma.$$

We will construct a graph $\zeta \supseteq \gamma$ such that ζ disentails each $\exists X_i(\theta_i \wedge \overline{\phi}_i)$ in CFT. This proves the claim since ζ is a graph and hence is satisfiable in CFT (Proposition 6.5).

Let Z be the set of all variables x such that there exists an i such that $x \notin X_i$ and

1. $Ax \in \theta_i\overline{\phi}_i - \theta_i\gamma$ for some A or
2. $xF \in \theta_i\overline{\phi}_i - \theta_i\gamma$ for some F or
3. $xfy \in \theta_i\overline{\phi}_i - \theta_i\gamma$ for some f, y or
4. $x \in \mathcal{V}(\theta_i|_{-X_i}) - \mathcal{D}(\gamma)$.

By the assumptions, to each i at least one of these cases applies. Now we fix for every variable $x \in Z$

- a sort A_x not occurring in γ or in any of the ϕ_i, and
- a feature f_x not occurring in γ or in any of the ϕ_i (neither as a feature constraint nor as element of an arity constraint).

It is understood that $A_x \neq A_y$ and $f_x \neq f_y$ if $x \neq y$. This is possible, since we have assumed that the alphabets of sorts and features are infinite.

For every $x \in Z$ let F_x be the set of features that are realized for x in γ. Now we are ready to define the graph ζ:

$$\begin{array}{rcl}\zeta & := & \gamma \\ & \cup & \{A_x x \mid x \in Z,\ \gamma \text{ contains no sort constraint for } x\} \\ & \cup & \{x f_x x \mid x \in Z,\ \gamma \text{ contains no arity constraint for } x\} \\ & \cup & \{x(F_x \cup \{f_x\}) \mid x \in Z,\ \gamma \text{ contains no arity constraint for } x\}.\end{array}$$

It remains to show that ζ disentails $\exists X_i(\theta_i \wedge \overline{\phi_i})$ in CFT for every i. By Proposition 6.8, it suffices to show that each $\theta_i(\zeta \wedge \overline{\phi_i})$ contains a clash. To this end we take a closer look at the four cases in the definition of Z. Recall that for every i at least one case applies.

1. $Ax \in \theta_i\overline{\phi_i} - \theta_i\gamma$ and $x \notin X_i$.

   Since $\theta_i(\gamma \wedge \overline{\phi_i})$ is clash-free, $\theta_i\gamma$ does not contain a sort constraint for x. Since $x \in \mathcal{V}(\theta_i\overline{\phi_i})$ and θ_i is idempotent, $x = \theta_i x$, thus γ also does not contain a sort constraint for x. Hence by the definition of ζ, $A_x x \in \zeta$ with $A_x \neq A$, which causes a clash in $\theta_i(\zeta \wedge \overline{\phi_i})$.

2. $xF \in \theta_i\overline{\phi_i} - \theta_i\gamma$ and $x \notin X_i$.

   Since $\theta_i(\gamma \wedge \overline{\phi_i})$ is clash-free, $\theta_i\gamma$ does not contain an arity constraint for x. Since $x \in \mathcal{V}(\theta_i\overline{\phi_i})$ and θ_i is idempotent, we have $x = \theta_i x$ and thus γ does not contain an arity constraint for x. Hence $x f_x x \in \zeta$ and $f_x \notin F$, which causes a clash in $\theta_i(\zeta \wedge \overline{\phi_i})$.

3. $xfy \in \theta_i\overline{\phi_i} - \theta_i\gamma$ and $x \notin X_i$.

   Since θ_i is a normalizer of $\gamma \wedge \phi_i$, there is no z such that $xfz \in \theta_i\gamma$, that is, $\theta_i\gamma$ does not realize f for x. Since $x \in \mathcal{V}(\theta_i\overline{\phi_i})$ and θ_i is idempotent, $x = \theta_i x$, thus γ also does not realize f for x. By assumption γ is saturated, hence γ does not contain an arity constraint for x, since any arity constraint for x would exclude f for x and therefore would lead to a clash in $\theta_i(\gamma \wedge \overline{\phi_i})$. Hence $x(F_x \cup \{f_x\}) \in \zeta$ and $f \notin F_x \cup \{f_x\}$, which implies that $\theta_i(\zeta \wedge \overline{\phi_i})$ contains a clash.

4. $x \in \mathcal{V}(\theta_i|_{-X_i}) - \mathcal{D}(\gamma)$.

   There must be an equation $x \doteq y$ or $y \doteq x$ in θ_i. Since θ_i is X_i-oriented, we know that $y \notin X_i$. Hence either $y \in \mathcal{D}(\gamma)$ or $y \in Z$, which means that both x and y are determined in ζ.

   If either x or y has no sort constraint in γ, then $\theta_i\zeta$ contains a sort clash. Otherwise, either x or y has no arity constraint in γ since x and y are not both determined in γ and γ is saturated by assumption. Hence $\theta_i\zeta$ contains an arity clash. □

Proposition 6.12 *Let $\mathcal{A}$ be a model of CFT, γ a saturated graph, ϕ a basic constraint, X a finite set of variables disjoint from $\mathcal{V}(\gamma)$, and θ an X-oriented normalizer of $\gamma \wedge \phi$. Then $\gamma \models_{\mathcal{A}} \exists X\phi$ iff*

1. *$\theta(\gamma \wedge \overline{\phi})$ is clash-free and*
2. *$\mathcal{C}(\theta\overline{\phi} - \theta\gamma) \subseteq X$ and*
3. *$\mathcal{V}(\theta|_{-X}) \subseteq \mathcal{D}(\gamma)$.*

Proof. Suppose that $\gamma \models_{\mathcal{A}} \exists X\phi$. Then (1) follows from Proposition 6.8 since the graph γ is satisfiable in $\mathcal{A}$ (Proposition 6.5). The claims (2) and (3) follow with Lemma 6.11.

For the other direction, first observe that

$$\gamma \models_{\mathcal{A}} \theta|_{-X}$$

follows with Lemma 6.9 from the assumptions (1) and (3). Since $\mathcal{V}(\gamma)$ is disjoint from X, $\theta\gamma = (\theta|_{-X})\gamma$, hence,

$$\gamma \ \models_{\mathcal{A}} \ (\theta|_{-X} \wedge \gamma) \ \models \ \theta|_{-X} \wedge \theta\gamma.$$

Since $\theta(\gamma \wedge \overline{\phi})$ is clash-free, we know by Proposition 6.4 that $\theta\overline{\phi} - \theta\gamma$ is a graph. Thus

$$\tilde{\forall}\exists X(\theta\overline{\phi} - \theta\gamma)$$

by Proposition 6.5 and assumption (2). Hence,

$$\begin{array}{lll}
\gamma & \models_{\mathcal{A}} & \theta|_{-X} \wedge \theta\gamma \wedge \exists X(\theta\overline{\phi} - \theta\gamma) \\
& \models_{\mathcal{A}} & \exists X(\theta|_{-X} \wedge (\theta\overline{\phi} - \theta\gamma) \wedge \theta\gamma) \quad \text{since } X \text{ is disjoint from} \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{V}(\theta|_{-X}) \text{ and } \mathcal{V}(\gamma) \\
& \models_{\mathcal{A}} & \exists X(\theta|_{-X} \wedge \theta\overline{\phi}) \\
& \models_{\mathcal{A}} & \exists X(\theta|_{-X} \wedge \theta|_X \wedge \theta\overline{\phi}) \qquad\quad \text{since } \theta \text{ is idempotent} \\
& & \qquad\qquad\qquad\qquad\qquad\qquad\quad \text{and } X\text{-oriented} \\
& \models_{\mathcal{A}} & \exists X(\theta \wedge \theta\overline{\phi}).
\end{array}$$

□

Proof of Theorem 5.6. The first part of Theorem 5.6 is Proposition 6.8, the second part is Proposition 6.12. □

Proof of Theorem 5.7. The implication from right to left is trivial. It remains to show that for every model $\mathcal{A}$ of CFT, basic constraints $\phi, \phi_1, \ldots, \phi_n$ and finite sets $X_1, \ldots, X_n$ of variables,

$$\phi \models_{\mathcal{A}} \exists X_1\phi_1 \vee \ldots \vee \exists X_n\phi_n \quad \Rightarrow \quad \exists i: \ \phi \models_{\mathcal{A}} \exists X_i\phi_i.$$

Without loss of generality we can assume that ϕ is a saturated graph, and that no X_i has a variable in common with ϕ. By Proposition 6.7, we may decompose each ϕ_i into $\theta_i \wedge \theta_i \overline{\phi_i}$ for some X_i-oriented normalizer θ_i of $\phi_i \wedge \phi$. We may assume without loss of generality that $\theta_i(\phi \wedge \overline{\phi}_i)$ is clash-free for any i, since otherwise by Proposition 6.8

$$\phi \wedge \neg\exists X_i(\theta_i \wedge \overline{\phi}_i) \models\!\!\mid_{\mathcal{A}} \phi.$$

Moreover, it follows by Lemma 6.11 that $\mathcal{C}(\theta_i\overline{\phi}_i - \theta_i\phi) \subseteq X$ and $\mathcal{V}(\theta_i|_{-X_i}) \subseteq \mathcal{D}(\phi)$ for some i. Hence, the claim follows with Proposition 6.12. □

7 The Abstract Machine

The decision method developed in Section 5 is abstract and does not provide directly for a discussion of important algorithmic aspects such as worst-case complexity and incrementality. We will now present an algorithmic formulation of the method showing how constraints can be processed incrementally, an aspect that is of crucial importance for a constraint system to be used in a "real" constraint programming system. The algorithmic formulation will also provide for an upper bound on the computational complexity of entailment checking.

To keep the presentation of the algorithm manageable, we will assume that the features that can actually occur in constraints are restricted to some *a priori known* finite set. Note that this assumption only restricts the set of inputs formulae of the algorithm, it does not affect the theory under consideration. This assumption can certainly not be made in practise, but our idealized algorithm nevertheless illustrates important techniques that do carry over to the general case. We will see that our algorithm decides entailment and disentailment in at most quasi-linear time. The development of truly efficient implementation techniques for the general case is not straightforward and will require further research.

The algorithm is presented as an abstract machine consuming a conjunction of possibly negated basic constraints

$$\gamma_1 \wedge \neg\exists X_1\phi_1 \wedge \gamma_2 \wedge \neg\exists X_2\phi_2 \wedge \gamma_3 \wedge \ldots$$

from left to right and detecting unsatisfiability as early as possible. The abstract machine is *incremental* in the sense that it avoids redoing work when further constraints arrive. This means that already processed information must be stored in a simplified form allowing for maximal reuse of work already done.

Let $\gamma = \gamma_1 \wedge \gamma_2 \wedge \ldots$ be the conjunction of the positive constraints seen so far. By the Independence Theorem we know that the conjunction of the positive and negated constraints seen so far is satisfiable if and only if (1) γ is satisfiable and (2) no negated constraint $\exists X_i\phi_i$ is entailed by γ. Moreover, a negated constraint $\exists X_i\phi_i$ can be discarded if it is disentailed by γ. But what do we do with negated constraints that are neither entailed nor disentailed by γ? These *undetermined* negated constraints pose two questions concerning incrementality: Given a further positive constraint γ_k, which of the undetermined negated constraints $\exists X_i\phi_i$ need to be reconsidered? And, if a negated constraint must be reconsidered, how can previous work be reused? Both questions will be answered in the following.

Our abstract machine for CFT has been inspired by Warren's abstract machine for Prolog [3] and the actual implementations of SICStus Prolog [10] and AKL [19].

7.1 The Heap

The algorithm employs a variable-centered representation of basic constraints. The represented constraint is kept in a form exhibiting a suitably oriented normalizer. The representation is built stepwise by including one atomic constraint at a time. Inclusion of an atomic constraint corresponds to application of the simplification rules (Triv), (Cong), (Elim) and (Orient). Whenever the represented constraint is extended, satisfiability is checked by means of the clash rules.

The representation is variable-centered in that an atomic constraint is always stored with the variable it is constraining (see Subsection 4.3). We assume that some finite enumeration type *feature* is given having as elements the features that can be used in constraints. The definition of the type *variable* appears in Figure 2. An equation $x \doteq y$ is represented by having the field *ref* of x point to y. The field *isglobal* is *false* if the variable is existentially quantified in a negated constraint, and *true* otherwise. Sort and arity constraints are represented as one would expect. A feature constraint xfy is represented by having the field *subtree*[f] of the variable x point to the variable y. If no feature constraint is known for x and f, then *subtree*[f] = *nil*. A new, completely unconstrained variable is created by the function *newvar*, also shown in Figure 2.

The collection of all variable records in the store is called the *heap*. From what we have said it is clear that the heap represents a basic constraint. The heap always satisfies three invariants:

1. the graph defined by the *ref*-pointers is acyclic (which means that it is a forest, where the *ref*-pointers are directed towards the roots)

2. the mapping obtained by dereferencing a variable to the root of the *ref*-pointer tree it appears in is an X-oriented normalizer of the repre-

```
arity    = set of feature
variable = record
               isglobal : bool
               ref      : ↑variable
               sort     : sort ⊎ {none}
               arity    : arity ⊎ {none}
               subtree  : array [feature] of ↑variable
           end

function newvar(is_global: bool) : ↑variable
    var x: ↑variable
    new(x)
    with x↑ do
        isglobal ← is_global
        ref ← nil    sort ← none    arity ← none
        for every f ∈ feature do subtree[f] ← nil
    return x
end newvar

procedure deref(var x: ↑variable)
    while x↑.ref ≠ nil do x ← x↑.ref
end deref
```

Figure 2: Representation, creation and dereferencing of variables.

sented constraint (where X is the set of all local variables)

3. the represented constraint is saturated.

The first invariant ensures that the procedure *deref* defined in Figure 2 always terminates.

7.2 Imposing Positive Constraints

For every atomic constraint there is a procedure imposing it on the heap:

Ax	$putsort(x, A)$
xfy	$putfeature(x, f, y)$
xF	$putarity(x, F)$
$x \doteq y$	$unify(x, y)$.

The procedures are given in Figure 3 and 4. They are justified by the simplification and clash rules of Section 5. If a clash is discovered, control jumps to the label *failure* (see Figure 5). It is easy to verify that the constraint imposition procedures preserve the heap invariants. If no clash is discovered, the constraint represented by the heap is satisfiable.

```
procedure putsort(x: ↑variable; A: sort)
    deref(x)
    if x↑.sort = none
    then setsort(x, A)
    else if x↑.sort ≠ A then goto failure
end putsort

procedure setsort(x: ↑variable; A: sort)
    x↑.sort ← A
    if x↑.isglobal then push(trail, "putsort(x,A)")
end setsort

procedure putfeature(x:↑variable; f:feature; y:↑variable)
    deref(x) deref(y)
    if x↑.arity ≠ none ∧ f ∉ x↑.arity
    then goto failure
    else if x↑.subtree[f] ≠ nil
        then unify(x↑.subtree[f],y)
        else setfeature(x,f,y)
end putfeature

procedure setfeature(x:↑variable; f:feature; y:↑variable)
    x↑.subtree[f] ← y
    if x↑.isglobal then push(trail, "putfeature(x,f,y)")
end setfeature

procedure putarity(x: ↑variable; F: arity)
    deref(x)
    if x↑.arity = none
    then setarity(x, F)
        for every f ∈ feature do
            if f ∉ F ∧ x↑.subtree[f] ≠ nil then goto failure
    else if x↑.arity ≠ F then goto failure
end putarity

procedure setarity(x: ↑variable; F: arity)
    x↑.arity ← F
    for every f ∈ F do   % maintain saturation
      if x↑.subtree[f] = nil then setfeature(x,f,newvar(x↑.isglobal))
    if x↑.isglobal then push(trail, "putarity(x,F)")
end setarity
```

Figure 3: Imposing sort, feature and arity constraints.

```
procedure unify(x,y: ↑variable)
    deref(x) deref(y)
    if x ≠ y
    then if x↑.isglobal
         then bind(y,x)
         else bind(x,y)
end unify

procedure bind(x,y: ↑variable)
    setref(x,y)
    if x↑.sort ≠ none then putsort(y,x↑.sort)
    for every f ∈ feature do
       if x↑.subtree[f] ≠ nil then putfeature(y,f,x↑.subtree[f])
    if x↑.arity ≠ none then putarity(y,x↑.arity)
end bind

procedure setref(x,y: ↑variable)
    x↑.ref ← y
    if x↑.isglobal
    then if x↑.sort ≠ none ∧ x↑.arity ≠ none ∧
            y↑.sort ≠ none ∧ y↑.arity ≠ none
         then push(trail, "setref(x,y)")
         else push(trail, "unify(x,y)")
end setref
```

Figure 4: Imposing equality constraints.

```
failure: while ¬ empty(trail) do undo(pop(trail))

procedure undo(e: stackentry)
    case e of
       "putsort(x,A)" : x↑.sort ← none
       "putarity(x,F)" : x↑.arity ← none
       "putfeature(x,f,y)" : x↑.subtree[f] ← nil
       "unify(x,y)" : x↑.ref ← nil
       "setref(x,y)" : x↑.ref ← nil
end undo
```

Figure 5: Restoring the heap after failure.

```
procedure residuate(var script: stack)
    var e: stackentry
    clear(script)
    while ¬ empty(trail) do
        e ← pop(trail)
        undo(e)
        if e ≠ "setref(...)" then push(script,e)
end residuate
```

```
procedure resume(script: stack)
    clear(trail)
    while ¬ empty(script) do execute(pop(script) )
end resume
```

Figure 6: Residuating and resuming negated constraints.

Every change to a global variable is recorded on a stack called *trail*. Note that the procedure *setref* records new equations between global variables differently depending on whether they are determined (*ref(x,y)*) or not (*unify(x,y)*). The reason for this distinction will be given later.

If control jumps to the label *failure* (see Figure 5), the trail is popped and previous changes to global variables are undone. In case there are no local variables, *untrailing* upon failure will in fact delete all constraints from the heap.

So far we have a machinery that can be fed piece by piece with atomic constraints. A new constraint is imposed by applying the appropriate procedure. Control jumps to the label *failure* if and only if the resulting heap is unsatisfiable. After a constraint is imposed without failure, the resulting heap is equivalent to the conjunction of the imposed constraint and the previous heap (provided auxiliary variables introduced by the procedure *setarity* to maintain saturation are quantified existentially). Clearly, the abstract machine presented so far is sound, incremental, and discovers failure as early as possible.

7.3 Imposing Negated Constraints

We will now see how a negated constraint $\neg\exists X\phi$ is processed. First, the trail is cleared (i.e., set to the empty stack). Then ϕ is fed like a positive constraint, where the existentially quantified variables X are created as local variables. If failure occurs, the resulting untrailing undoes all changes to global variables and the negated constraint is discarded (which is sound since in this case $\neg\exists X\phi$ is entailed by the positive constraints γ_i seen so far). If ϕ has been fed completely without causing a failure, the negated constraint is "residuated" by calling the procedure *residuate* of Figure 6, which returns a stack of constraints called a *script*. Residuation untrails and moves constraints from the heap to the script, such that the global part of the heap is restored to what it had been before processing the negated constraint, and such that the equivalence

$$\textit{restored heap} \wedge \textit{script} \quad \models\!\!\dashv_{\mathrm{CFT}} \quad \textit{heap before residuation} \tag{9}$$

holds. This equivalence would be obvious if the *setref*-entries in the trail (recording determined equations between global variables) were pushed as *unif*-entries on the script. Discarding them is however justified by Theorem 5.4 since the heap is equation-complete before residuation.

Next we will see that $\exists X\phi$ is entailed by the positive constraints if and only if the script obtained by residuation is empty. This means that a negative constraint $\neg\exists X_i\phi_i$ causes unsatisfiability of the conjunction

$$\gamma_1 \wedge \neg\exists X_1\phi_1 \wedge \gamma_2 \wedge \neg\exists X_2\phi_2 \wedge \gamma_3 \wedge \ldots$$

if and only if $\exists X_i\phi_i$ is processed without failure and residuates with an empty script.

To see the claim about residuation, suppose $\exists X\phi$ is imposed without failure on a heap whose global variables represent a constraint γ and residuates with a script representing the constraint σ. Moreover, suppose that ψ is the constraint represented by the local variables X in the heap just after residuation. By Equivalence (9) we have $\gamma \wedge \phi \models\!\!\dashv_{\mathrm{CFT}} \gamma \wedge \psi \wedge \sigma$. (This equivalence is slightly simplified since it ignores existentially quantified auxiliary variables introduced to maintain saturation of the heap.) Moreover, $\mathcal{C}(\psi) \subseteq X$, and ψ is satisfiable and equation-complete. Hence we know $\mathrm{CFT} \models \exists X\psi$ by the Entailment Theorem.

1. Suppose the script is empty. Then $\gamma \wedge \phi \models\!\!\dashv_{\mathrm{CFT}} \gamma \wedge \psi$ and hence $\gamma \wedge \exists X\phi \models\!\!\dashv_{\mathrm{CFT}} \gamma \wedge \exists X\psi$. Since $\mathrm{CFT} \models \exists X\psi$, we have $\gamma \models_{\mathrm{CFT}} \exists X\phi$.
2. Suppose the script is nonempty. Then we know by the Entailment Theorem that γ does not entail $\exists X\phi$ since the heap before residuation violates either condition (2.c) (i.e., there is a *unify*-entry on the trail) or condition (2.b) (i.e., there is a *put*-entry on the trail)).

We now know that a negative constraint residuating with a nonempty script is neither entailed nor disentailed by the positive constraints seen so far. Moreover, the script together with the records of the local variables X in the heap represent a simplified form of the negated constraint. This simplified form depends both on the negated constraint and the already seen positive constraints. If more positive information becomes available, the negated constraint must possibly be reconsidered. Rather than imposing the original negated constraint anew, its residuated script is resumed with the procedure *resume* in Figure 6. It suffices to resume a residuated script if one of the following events occurs:

- the script contains an entry *putsort(x,_)* and variable *x* is made a reference or acquires a sort;
- the script contains an entry *putfeature(x,f,_)* and variable *x* is made a reference or acquires feature *f* or an arity;
- the script contains an entry *putarity(x,_)* and variable *x* is made a reference or acquires an arity or a feature;
- the script contains an entry *unify(x,y)* and variable *x* or *y* is made a reference or acquires a sort, an arity, or a feature.

Resumption of a script is handled in the same way a negated constraint is imposed initially. In particular, a resumed script may residuate again with a new script.

7.4 Worst-Case Complexity

We will now see that an optimized version of our abstract machine can decide $\gamma \models_{\mathrm{CFT}} \exists X \phi$ in time at most quasi-linear in the size of γ and ϕ. The necessary optimization concerns the implementation of the forest consisting of the *ref*-pointers by means of an efficient union-find method [22].

For our worst-case analysis we assume that γ and ϕ are fed to the empty machine as a sequence of *newvar*, *put* and *unify* procedure calls. The constraint γ is fed first, then the trail is cleared, then ϕ is fed, and finally the procedure *residuate* is called. If failure occurs while γ is being processed, then γ is unsatisfiable and trivially entails $\exists X \phi$. If failure occurs while ϕ is being processed, then (and only then) γ disentails $\exists X \phi$. If no failure occurs, γ entails $\exists X \phi$ if and only if the script obtained by residuation is empty.

It suffices to show that the machine does not require more than quasi-linear time in the case where failure does not occur. Clearly, the size of the heap built after processing γ and ϕ is linear in the size of γ and ϕ. Since the procedure *bind*, through which all recursion is channelled, always sets a *ref*-pointer whose value was *nil* before, the total number of calls to *putsort*, *putarity*, *putfeature* and *unify* is linear. If we do not count recursive calls, these procedures require constant time plus the time for one or two calls of *deref*. Thus, the entire time needed is linear plus the time for a linear number of calls of *deref*. Hence, if we implement the congruence represented by the *ref*-pointers with an efficient union-find method employing path compression, the abstract machine will run in at most quasi-linear time [22].

Our abstract machine and hence our worst-case analysis assume that the features that can occur in γ and ϕ are restricted to some *a priori known* finite set. Without this assumption, the time for obtaining y given x and f such that xfy is in the heap is no longer constant. In this case entailment checking can certainly be implemented with a complexity not worse than quadratic in the size of γ and ϕ.

8 Summary and Conclusion

We have shown that records can be incorporated into constraint (logic) programming in a straightforward and natural manner. Semantically, records are modeled as feature trees generalizing the trees corresponding to first-order terms. The first-order language we have set up for describing feature trees is richer than the equational language employed with classical trees in that it allows for finer-grained descriptions. The resulting constraint system CFT is a conservative extension of both Prolog II's rational tree system [12, 13] and the feature tree system FT [9, 7]. Thus CFT brings together the work on classical tree constraints (e.g., [17, 12, 13, 23, 26]) and the work on feature descriptions (e.g., [21, 20, 1, 2, 4, 5, 6, 29, 9, 7, 11])—two lines of research that seemed to be rather far apart in the past.

The declarative semantics of CFT was specified both algebraicly (the feature tree structure $\mathcal{T}$) and logically (the first-order theory CFT given by five axiom schemes). For the constraint problems considered in the paper the coincidence of the algebraic and logical semantics was shown. We conjecture that CFT is in fact a complete recursive axiomatization of the feature tree structure.

We have established abstract decision methods for satisfiability and entailment of constraints. Moreover, we have shown that CFT satisfies the Independence Property, which means that our methods can decide the satisfiability of conjunctions of positive and negative constraints.

We have presented an idealized abstract machine processing positive and negative constraints incrementally. The correctness of the machine was verified using the abstract decision method established before. Under the assumption that the features that can appear in constraints are restricted to some a priori known finite set, an optimized version of the machine can decide satisfiability and entailment in quasi-linear time.

Our abstract machine shows that an implementation of CFT will be more complex than an implementation of the classical rational tree system using established Prolog technology [3]. Really efficient implementations of CFT will require further research. However, since the classical rational tree system is a subsystem of CFT, a gracefully degrading implementation of CFT seems feasible, which pays for CFT's extra-expressivity only when non-classical constraints are used.

Acknowledgements

We are grateful to Michael Mehl and Ralf Scheidhauer for having pointed out to the first author how unification and residuation are implemented in SICStus Prolog and AKL. Discussions with Andreas Podelski and Peter van Roy also helped with the design of the abstract machine. Hubert Comon suggested Proposition 3.2. One of the anonymous referees provided helpful comments. Last not least the paper profited from discussions with Joachim Niehren and Jörg Würtz.

The research reported in this paper has been supported by the Bundesminister für Forschung und Technologie under contract ITW 9105, the Esprit Project ACCLAIM (PE 7195) and the Esprit Working Group CCL (EP 6028).

References

[1] H. Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures*. PhD thesis, University of Pennsylvenia, Philadelphia, PA, 1984.

[2] H. Aït-Kaci. An algebraic semantics approach to the effective resolution of type equations. *Theoretical Comput. Sci.*, 45:293–351, 1986.

[3] H. Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction*. Logic Programming. MIT Press, Cambridge, MA, 1991.

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

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

[6] H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. In J. Maluszyński and M. Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming*, Springer LNCS vol. 528, pages 255–274. Springer-Verlag, 1991.

[7] H. Aït-Kaci, A. Podelski, and G. Smolka. A feature-based constraint system for logic programming with entailment. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages

1012–1021, ICOT, Japan, 1992. Association for Computing Machinery. Full version will appear in Theoretical Computer Science.

[8] R. Backofen and G. Smolka. A complete and recursive feature theory. Research Report RR-92-30, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, D 66123 Saarbrücken, Germany, Sept. 1992.

[9] R. Backofen and G. Smolka. A complete and recursive feature theory. In *Proc. of the 31 th ACL*, Columbus, Ohio, 1993. Complete version as [8].

[10] M. Carlsson, J. Widén, J. Andersson, S. Andersson, K. Boortz, H. Nilsson, and T. Sjöland. *SICStus Prolog Users's manual.* SICS, Box 1263, 164 28 Kista, Sweden, 1991.

[11] B. Carpenter. Typed feature structures: A generalization of first-order terms. In Saraswat and Ueda [28], pages 187–201.

[12] A. Colmerauer. Prolog and infinite trees. In K. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 153–172. Academic Press, 1982.

[13] A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the 2nd International Conference on Fifth Generation Computer Systems*, pages 85–99, 1984.

[14] B. Courcelle. Fundamental properties of infinite trees. *Theoretical Comput. Sci.*, 25(2):95–169, 1983.

[15] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[16] R. Helm, K. Marriott, and M. Odersky. Constraint-based query optimization for spatial databases. In *Tenth ACM Symposium on the Principles of Database Systems*, pages 181–191, Denver, CO, May 1991.

[17] G. Huet. Résolution d'equations dans des langages d'ordre $1, 2, \cdots, \omega$. Thèse de Doctorat d'Etat, l'Université Paris VII, Sept. 1976.

[18] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Conference on Principles of Programming Languages*, pages 111–119, Munich, Germany, Jan. 1987. ACM.

[19] S. Janson and S. Haridi. Programming paradigms of the Andorra kernel language. In Saraswat and Ueda [28], pages 167–186.

[20] R. M. Kaplan and J. Bresnan. Lexical-Functional Grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–381. MIT Press, Cambridge, MA, 1982.

[21] M. Kay. Functional grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley, CA, 1979. Berkeley Linguistics Society.

[22] D. C. Kozen. *The Design and Analysis of Algorithms*. Springer-Verlag, 1992.

[23] J.-L. Lassez, M. J. Maher, and K. G. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 15, pages 587–625. Morgan-Kauffman, 1988.

[24] J. L. Lassez and K. McAloon. A constraint sequent calculus. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 52–61, June 1990.

[25] M. J. Maher. Logic semantics for a class of committed-choice programs. In J.-L. Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming*, pages 858–876. MIT Press, 1987.

[26] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 348–357. IEEE Computer Society, 1988.

[27] V. Saraswat and M. Rinard. Concurrent constraint programming. In *Proceedings of the 7th Annual ACM Symposium on Principles of Programming Languages*, pages 232–245, San Francisco, CA, January 1990.

[28] V. Saraswat and K. Ueda, editors. *Logic Programming, Proceedings of the 1991 International Symposium*, San Diego, USA, 1991. The MIT Press.

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

[30] G. Smolka and R. Treinen. Records for logic programming. In K. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 240–254, Washington, USA, Nov. 1992. The MIT Press.

[31] R. Treinen. Feature constraints with first-class features. In A. Borzyszkowski and S. Sokołowski, editors, *Mathematical Foundations of Computer Science*, Lecture Notes in Artificial Intelligence vol. 711, pages 734–743. Springer-Verlag, Sept. 1993.