

# DF Constraint System

Liviu-Virgil Ciortuz\*

April, 1995

## Abstract

This work presents DF, a feature constraint system that starting from the main lines of the two major approaches in the field – the OSF system of Hassan Aït-Kaci, and the CFT system of Gert Smolka – provides for new significant characteristics: a first-class citizen status offered to (both features and) sorts, finer grained constraints using arguments in the feature context, well-type conditioning between feature values and types, and principle-based completion of sort signature.

Our approach was inspired by a less known work of Michael Kifer, namely F-logic [16] which was designed as “a logical foundation for object-oriented and frame-based languages”. An alternative constraint approach is needed in order make F-logic useful as a logic programming language, due to complex description of objects and the large number of inference rules involved. This paper aims to do this job.

---

\* Supported by a grant from the French Ministry of Foreign Affairs.

Address: “A. I. Cuza” University of Iasi, Department of Computer Science, 6600 Iasi, Romania.  
email: ciortuz@uaic.ro.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Related work . . . . .	3
1.2	Why another feature constraint system? . . . . .	4
<b>2</b>	<b>DF Constraints</b>	<b>6</b>
2.1	DF Logic Records . . . . .	6
2.2	DF Logic Semantics . . . . .	8
<b>3</b>	<b>DF Normalization</b>	<b>12</b>
3.1	DF Basic Normalization . . . . .	13
3.2	Reference Signatures . . . . .	15
3.2.1	Simple reference signatures . . . . .	16
3.2.2	Conditional Reference Signatures . . . . .	19
3.3	DF Definite Normalization . . . . .	21
3.4	DF Relative Normalization . . . . .	24
.1	Syntax . . . . .	29
.2	Semantics . . . . .	30

# 1 Introduction

We define DF, a constraint system working on  $\chi$ -terms, higher-order logic records extending H. Ait-Kaci’s  $\psi$ -terms, and thus first-order terms. The present section firstly surveys the DF background and then offers an account on what distinguishes it with respect to well-known feature constraint systems. Section 2 introduces the syntax and logical semantics of DF. Then, Section 3 provides the operational approach for both (positive) DF-clause solving and sort signature completion via simple, respectively definite normalization rules. Later on, relative normalization rules will carry both  $\chi$ -term matching and the satisfiability test of DF-formulae by DF clause entailment. The final aim of this work is to make M. Kifer’s F-logic effective as a logic programming language.

## 1.1 Related work

The interest for feature constraint systems emerged with the seminal work of Gert Smolka [21] that successfully characterized the early unification-grammar formalisms like LFG, FUG, PATR-II [14], [15], [19] in the predicate logic. Constraints in these formalisms were expressed as first-order formulae with equality and a general method to decide their satisfiability was designed.

The way was thus opened to better specify the tree constructor terms in Herbrand, the constraint system underlying Prolog. FT [6], the new system, based on feature terms, emerged then to CFT [23] by incorporating arity constraints, and was finally chosen to underlie the concurrent constraint object-oriented language Oz [20].

On the other side, feature constraints were used to express in a powerful way logic records generalizing first-order terms, namely  $\psi$ -terms [2] [3] [7], allowing for the sound use of the inheritance principle in the area of logic programming.

This paper aims to enhance the previously mentioned work by reconsidering a (more complete) attempt to express the object-orientation paradigm in logic, namely F-logic [16]. We preserve the F-logic high expressivity<sup>1</sup> but replace its rather complicated operational semantics (13 inference rules!) with a simple one based on constraint theory normalization and relative simplification rules. The resulting constraint system is called DF.

The interested reader can find in the Appendix at the end of this paper a brief introduction of F-logic, slightly adapted from [16]. For convenience, we restricted the presentation assuming Datalog instead of Prolog ones as ‘identification’ terms for complex objects.<sup>2</sup> While reading the Appendix is not compulsory for the understanding of the paper, it offers an idea of the way we followed to make F-logic effective as a logic programming language.

A simple synoptic view on the above mentioned feature-constraint systems is given in Figure 1. It considers also the E and EF systems [24] which provides FT, respectively CFT systems with a first-class citizen status offered to features. DF extends this status to sorts.

Specific contributions aimed by the present work are the following:

---

<sup>1</sup>up to restriction to Datalog terms instead of Prolog ones to denote complex objects and classes. We call corresponding subset of F-logic DF-logic. For the sake of uniformity, we do not define atoms on top of terms, as F-logic does. We introduce instead a new type of methods, namely predicative methods in the  $\chi$ -term definition.

<sup>2</sup>We name DF-logic the subset of F-logic determined in this way. All the results are preserved in the new settings.

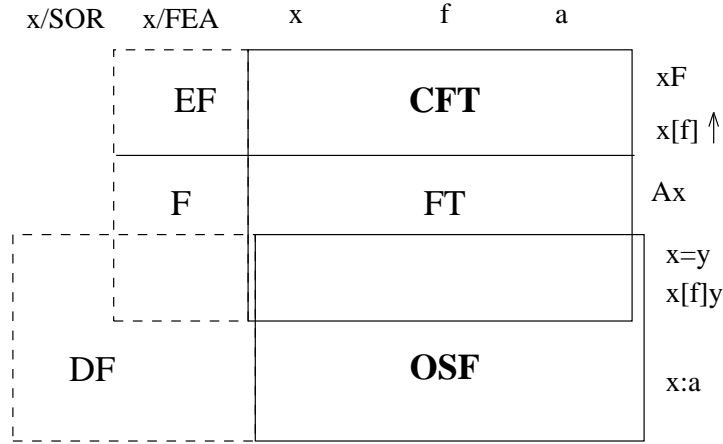


Figure 1: Overview on Feature Constraint Systems

- distinguishing between single- and multi-valued features on one side, and between features values and their types on the other side;
- using of arguments to define the feature application context;
- doing type control via well-type conditioning and type inheritance axioms; and
- allowing for inferential completion of (partially specified, possibly conditional) sort signature.

## 1.2 Why another feature constraint system?

The answer to this question is mainly twofold:

1. The need for a logic programming language working in the F-logic specific settings lead us to a “dedicated” feature constraint system here presented.
2. The OSF system (which is obviously more linked to DF than CFT) has in our opinion two major drawbacks:
  - i. OSF cannot ultimately dissociate between feature values and types.

**Example 1.1** *Given the simple sort signature*

*girl : person boy : person lucy : girl mark : boy jim : boy.*

*shown in Figure 2, an OSF-like declaration*

*lucy[likes ⇒ boy]*

*will make the goal*

*? – lucy[likes ⇒ X]*

*produce the answers*

*X = boy, X = mark, X = jim.*

*We consider that the informational entropy of this response is too wide.*

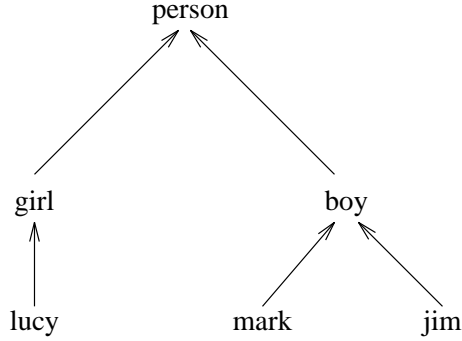


Figure 2: A (ground) reference signature

F-logic, and therefore the DF constraint system make instead a clear distinction between feature values and types. This is done by imposing  $\Rightarrow$  to declare only types (i.e., in the above example: if *lucy* likes something, then that is a *boy*), and introducing the functional correspondence  $\rightarrow$  to denote (only) values. For example, the fact that *lucy* likes *jim* is (explicitly) declared by

$$lucy[likes \rightarrow jim].$$

The correspondence between feature values and types is ensured by the so call *well-type conditioning* principle. In a simplified form, this principle says that

$$\left. \begin{array}{l} object[feature \rightarrow value] \\ object[feature \Rightarrow type] \end{array} \right\} \Rightarrow value : type.$$

ii. OSF assumes a sort signature completely specified at the syntax level. It is often the case that the sort signature we are involved with is not apriori totally known, or at least that a combined syntax-semantic approach is suitable from the user point of view. DF does this just using the above mentioned well-typing principle. To get an idea of how things work in DF, let's take the following DF definite program:

**Example 1.2**

- (C1)  $X[happy] :- X : person[friend \rightarrow Y].$
- (C2)  $person[friend \Rightarrow person].$
- (C3)  $Z[F \rightarrow W] :- W[F : symmetric \rightarrow Z].$
- (C4)  $friend : symmetric.$
- (C5)  $albert : person$
- (C6)  $albert[friend \rightarrow lucy].$

We informally show how the predicative feature *happy* yields the truth value *true* for *lucy*. First, *albert* inherits the *friend* typing method from the *person* class (C2) as he is a *person* (C5). One could visualise the type inheritance effect<sup>3</sup> by explicitly writing

---

<sup>3</sup>In a simplified form, the type inheritance principle can be expressed

$$\left. \begin{array}{l} class[feature \Rightarrow type] \\ object : class \end{array} \right\} \Rightarrow object[feature \Rightarrow type].$$

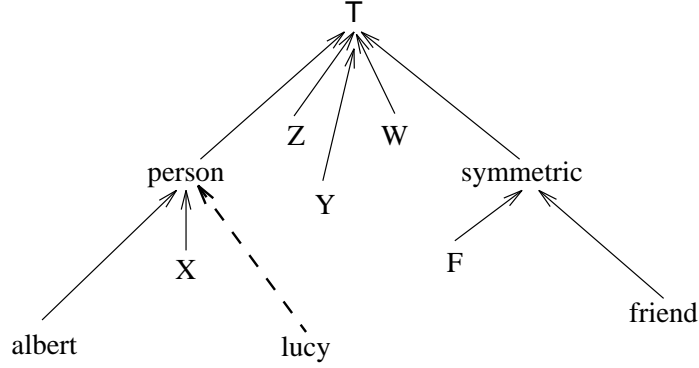


Figure 3: Run-time sort signature completion

(C7)  $albert[friend \Rightarrow person]$ .

Now, due to the the well-typing conditions that link method values and types, it follows from (C6) and (C7) that *lucy* is a *person*.<sup>4</sup> The effect of run-time signature completion for this specific case is shown in Figure 3.

(C8)  $lucy : person$ .

Applying clause (C1) for *X* instantiated to *lucy* will prove that's enough for our goal to prove that *lucy* has a friend.

(C9)  $lucy[friend \rightarrow Y]$ .

She has a friend indeed due to (C3), since *friend* is a symmetric function (C4) and there is someone whose friend is she, namely *albert* (C6). So, finally

(C10)  $lucy[happy]$ .

Note for example in the clause (C3) how DF offers a first-class citizen status to features. That clause says that every symmetric function commutes his argument with its caller.

This example was inspired to us by [12].

## 2 DF Constraints

This section presents the DF system syntax and the underlying constraint semantics.

### 2.1 DF Logic Records

The logic structures described by the DF system are based upon three types of elementary relations: *derivations*, *equations* and *methods*. These relations lead naturally to the DF base terms:

---

<sup>4</sup>Note that OSF, and therefore LOGIN and LIFE are not capable of such an inference. If it was not explicitly declared that *lucy* is a *person*, then the mentioned goal could not be solved.

derivation	$s : t$	1.
equation	$s = t$	2.
functional method	$s[u@w_1, \dots, w_n \rightarrow v]$	3.1
set-valued method	$s[u@w_1, \dots, w_n \rightarrow\!\!\rightarrow v]$	3.2
function-typing method	$s[u@w_1, \dots, w_n \Rightarrow t]$	3.1'
set-typing method	$s[u@w_1, \dots, w_n \Rightarrow\!\!\Rightarrow t]$	3.2'
boolean method	$s[u@w_1, \dots, w_n]$	3.3

The intended meaning of these relations is respectively

- 1:  $s$  is an object of the type  $t$  (or:  $s$  is a sort derived from  $t$ );
- 2:  $s$  is equal to  $t$ ;
- 3.1: The value of the functional feature  $u$  for the object/class  $s$  in the context made of the  $w_1, \dots, w_n$  arguments is  $v$ ;
- 3.2: One of the values taken by the set-valuated feature  $u$  for  $s$  is  $v$  when taking the argument  $w_1, \dots, w_n$ ;
- 3.1' and 3.2': The value(s) of the function-typing, respectively set-typing feature<sup>5</sup>  $u$  for  $s$  in the context  $w_1, \dots, w_n$  should belong to the sort  $t$ ; and
- 3.3: The feature  $u$  is true for  $s$  in the context  $w_1, \dots, w_n$ .

In the above relations,  $s, t, u, v, w_i$  are *references* belonging to  $\mathcal{R} = \mathcal{C} \cup \mathcal{V}$ , where  $\mathcal{C} = \{a, b, c, \dots\}$  is a set of *constants*,<sup>6</sup> and  $\mathcal{V} = \{X, Y, \dots\}$  is a countably infinite set of *variables*.

We individualize among constants a special one  $\top$ , read “top”.

Methods having a same *root*  $s$  are commonly built together into a single frame structure called  $\chi$ -term.

By syntax convenience, the special character  $@$  is omitted when the feature takes no arguments.

**Example 2.1** *The well-known Prolog denotation for lists  $[], [H | T]$  is made explicit by the following DF-terms that define also the type for the append operation on lists:*

$$\begin{aligned}
(T1) \quad & nil : list \\
(T2) \quad & list[ tail \Rightarrow list; \\
& \quad \quad \quad append@list \Rightarrow list]
\end{aligned}$$

(T1) is a derivation term and declares *nil* to be a list. (T2) is a  $\chi$ -term saying that the tail of any list is also of the list type, and that appending a list to another list returns always a list.

By convenience, derivation terms will be allowed to enhance the  $\chi$ -term writing. For instance

$$(T3) \quad nil[append@L:list \rightarrow L]$$

The  $\chi$ -term definition can be extended recursively to allow that  $v$  and  $t$  in the methods' definition be  $\chi$ -terms, as in

---

<sup>5</sup> $\Rightarrow$  corresponds to  $\rightarrow$ , and  $\Rightarrow\!\!\Rightarrow$  to  $\rightarrow\!\!\rightarrow$ .

<sup>6</sup> $\mathcal{C}$  is assumed to be infinite in order to ensure the existential prenex s-equivalent form for each DF-formula.

---

$s \doteq t$	<i>equation constraint</i>
$s : t$	<i>derivation constraint</i>
$s[u_{qt}\bar{v}]t$	<i>non-boolean method constraint</i> ( $qt = \rightarrow, \Rightarrow, \twoheadrightarrow, \text{ or } \Rightarrow\Rightarrow$ )
$s[u_{\bar{v}}\bar{v}]$	<i>boolean method constraint</i>

---

Figure 4: Atomic DF-constraints

$$(T4) \quad L' : list[ \textit{head} \rightarrow H; \\ \textit{tail} \rightarrow T[\textit{append}@L: list \rightarrow Z]; \\ \textit{append}@L \rightarrow Y[ \textit{head} \rightarrow H; \\ \textit{tail} \rightarrow Z]]$$

Now it can be easily seen that  $\psi$ -terms – logic records in the OSF theory – are  $\chi$ -terms containing only functional argumentless methods, and whose features names are constants.<sup>7</sup> By consequence, first-order terms are  $\chi$ -terms.

DF-logic formulae are built on top of DF-terms using logical quantifiers and connectives.

## 2.2 DF Logic Semantics

We introduce now the DF constraint semantics. This is an intensional one, meaning that relations that define predicates are assigned to denotations in the domain of interpretation, not to the symbol names themselves.

The elementary relations introduced in the precedent subsection translate naturally into DF atomic constraints which are shown in Figure 4. The  $\bar{v}$  symbol denotes a finite (possibly empty) sequence  $v_1, \dots, v_k$ . These constraints are built upon  $;$ ,  $\doteq$ , and  $u_{qt}\bar{v}$  as binary predicates (for  $qt$  one of  $\rightarrow, \Rightarrow, \twoheadrightarrow, \Rightarrow\Rightarrow$ ), and  $u_{\bar{v}}\bar{v}$  as unary predicate.

Note that we think  $u_{qt}\bar{v}$  as HiLog [9] predicate. Translation into first-order predicate logic with equality is imediate. The reader could alternatively consider  $[\dots]$  as a multy-arity predicate defined in the contextual first-order predicate calculus.

**Definition 2.1** *Let  $\mathcal{C}$  be a set of constants. A DF-algebra over  $\mathcal{C}$  is a tuple*

$$\mathcal{A} = \langle D^{\mathcal{A}}, \prec_{\mathcal{A}}, I_{\mathcal{C}}^{\mathcal{A}}, (I_{p \rightarrow \bar{q}}^{\mathcal{A}})_{p, \bar{q}}, (I_{p \Rightarrow \bar{q}}^{\mathcal{A}})_{p, \bar{q}}, (I_{p \twoheadrightarrow \bar{q}}^{\mathcal{A}})_{p, \bar{q}}, (I_{p \Rightarrow\Rightarrow \bar{q}}^{\mathcal{A}})_{p, \bar{q}}, (I_{p_{\bar{v}} \bar{q}}^{\mathcal{A}})_{p, \bar{q}} \rangle$$

where

$D^{\mathcal{A}}$  is the domain of the DF-algebra  $\mathcal{A}$ ,  
 $\prec_{\mathcal{A}}$  is a binary relation on  $D^{\mathcal{A}}$ ,  
 $I_{\mathcal{C}}^{\mathcal{A}} : \mathcal{C} \rightarrow D^{\mathcal{A}}$  is the interpretation of constants,  
 $I_{p \rightarrow \bar{q}}^{\mathcal{A}}, I_{p \Rightarrow \bar{q}}^{\mathcal{A}}, I_{p \twoheadrightarrow \bar{q}}^{\mathcal{A}}, I_{p \Rightarrow\Rightarrow \bar{q}}^{\mathcal{A}}$ , indexed on  $p \in D^{\mathcal{A}}$  and  $\bar{q} \in (D^{\mathcal{A}})^* = \bigcup_{n \geq 0} (D^{\mathcal{A}})^n$   
are binary relations on  $D^{\mathcal{A}}$ , and  
 $I_{p_{\bar{v}} \bar{q}}^{\mathcal{A}}$  are unary relations on  $D^{\mathcal{A}}$ .

---

<sup>7</sup>These restrictions lead to the fact that  $\psi$ -terms are defined upon a sort signature  $(\mathcal{C}, :, \wedge)$  which organize  $\mathcal{C}$  as lower semi-lattice,  $\wedge$  denoting here the sort intersection. As [2] shows, such a signature can always be constructed provided that  $:$  is a computably partial order relation on  $\mathcal{C}$ .

Let  $\mathcal{A}$  be a DF-algebra,  $\preceq_{\mathcal{A}}$  the reflexive and transitive closure of  $\prec_{\mathcal{A}}$ , and  $\alpha : \mathcal{V} \rightarrow D^{\mathcal{A}}$  a variable assignment in  $\mathcal{A}$  extended as usually to constants from  $\mathcal{C}$ . Satisfiability of DF-atomic constraints is defined as follows:

$$\begin{aligned} \mathcal{A}, \alpha \models s \doteq t & \quad \text{iff} \quad \alpha(s) = \alpha(t) \\ \mathcal{A}, \alpha \models s : t & \quad \text{iff} \quad \alpha(s) \preceq_{\mathcal{A}} \alpha(t) \\ \mathcal{A}, \alpha \models s[u_{qt}\bar{v}]t & \quad \text{iff} \quad (\alpha(s), \alpha(t)) \in I_{\alpha(u)_{qt}\alpha(\bar{v})}^{\mathcal{A}} \text{ for } qt \neq \bar{b} \\ \mathcal{A}, \alpha \models s[u_{\bar{b}}\bar{v}] & \quad \text{iff} \quad \alpha(s) \in I_{\alpha(u)_{\bar{b}}\alpha(\bar{v})}^{\mathcal{A}} \end{aligned}$$

Satisfiability of complex DF formulae (obtained via logical connectives and quantifier application) is defined as usually.

From now on, for the sake of uniformity, whenever not otherwise said,  $qt$  will denote  $\rightarrow, \Rightarrow, \rightarrow\rightarrow, \Rightarrow\Rightarrow$  and  $\bar{b}$ . By consequence, when writing  $s[u_{\bar{b}}\bar{v}]t$  we will assume by convention that  $t$  is the special symbol  $\top$ , denoting the truth value true. (Boolean methods are seen as a special kind of functional methods.)

Every  $\chi$ -term is interpreted as the conjunction of elementary constraints it is built upon.

**Definition 2.2** *A DF-clause is a finite conjunction of atomic DF-constraints.*

**Example 2.2** *For the  $\chi$ -term (T2) from the precedent example we get the DF clause*

$$list[tail_{\rightarrow}]list \wedge list[append_{\rightarrow} list]list$$

while the  $\chi$ -term (T4) goes into

$$\begin{aligned} L' : list \wedge L'[head_{\rightarrow}]H \wedge L'[tail_{\rightarrow}]T \wedge T[append_{\rightarrow}L]Z \wedge L : list \wedge \\ L'[append_{\rightarrow}L]Y \wedge Y[head_{\rightarrow}]H \wedge Y[tail_{\rightarrow}]Z. \end{aligned}$$

Conjunction will be seen as an idempotent and commutative operation having  $\top$  as neutral element. Thus DF-clauses can be seen equivalently as sets of atomic DF-constraints. The empty DF-constraint clause, denoted  $\top$ , is always true.

It's useful to note clear that any DF-clause  $\phi$  has three main components: the equation part  $\phi_{\doteq}$ , the derivation part  $\phi_{:}$ , and the frame part  $\phi_{[\ ]}$ . All these components of  $\phi$  are defined as one would expect:

$$\begin{aligned} \phi &= \{s : t \mid s : t \in \phi\} \\ \phi_{\doteq} &= \{s \doteq t \mid s \doteq t \in \phi\} \\ \phi_{[\ ]} &= \phi - (\phi_{:} \cup \phi_{\doteq}). \end{aligned}$$

Now we introduce a particular form for DF-clauses which serves to characterize their satisfiability. The next section will give an operational way to get such a form for every DF-clause.

**Definition 2.3** *A (positive) DF-clause  $\phi$  is in solved form (or, simply: is solved) if*

1. *there are no derivation cycles  $s:t_1, t_1:t_2, \dots, t_n:s$  in  $\phi$ ;*
2. *if  $s \doteq t \in \phi$ , then  $s$  occurs only once in  $\phi$ ;*
3. *if  $\left\{ \begin{array}{l} s_1[u_{1\rightarrow}\bar{v}_1]t_1 \in \phi \\ s_2[u_{2\rightarrow}\bar{v}_2]t_2 \in \phi, \text{ and} \\ s_1 \doteq s_2, u_1 \doteq u_2, \bar{v}_1 \doteq \bar{v}_2 \in \phi \end{array} \right\}$  then  $t_1 \doteq t_2 \in \phi$ ,*

- 
- (G)  $\tilde{\forall}(X : \top \wedge \neg \top : X)$
- (AS)  $\tilde{\forall}(X:Y \wedge Y:X \rightarrow X \doteq Y)$
- (F)  $\tilde{\forall}(X[u \rightarrow \bar{v}]Y \wedge X[u \rightarrow \bar{v}]Z \rightarrow Y \doteq Z)$
- (WT)  $\tilde{\forall}(X[u \rightarrow \bar{v}]Y \wedge X[F \Rightarrow \bar{v}]Z \rightarrow Y:Z)$   
 $\tilde{\forall}(X[u \Rightarrow \bar{v}]Y \wedge X[u \Rightarrow \bar{v}]Z \rightarrow Y:Z)$
- (T)  $\tilde{\forall}(X[u \rightarrow \bar{v}]Y \wedge X':X \wedge \bar{v}:\bar{v} \wedge Y:Y' \rightarrow X'[u \rightarrow \bar{v}]Y')$   
 $\tilde{\forall}(X[u \Rightarrow \bar{v}]Y \wedge X':X \wedge \bar{v}:\bar{v} \wedge Y:Y' \rightarrow X'[u \Rightarrow \bar{v}]Y')$
- (C)  $\tilde{\forall}(\Delta\phi \rightarrow \exists C(\phi)\phi)$  for every  $\phi$  solved clause.  
 $\Delta\phi$  is the set  $\{\neg(u \doteq u' \& \bar{v} \doteq \bar{v}') \mid s[u \rightarrow \bar{v}]t, s[u' \rightarrow \bar{v}']t \in \phi\}$ , and  
 $C(\phi)$  is the set of all constrained variables in  $\phi$ .
- 

Figure 5: Axiom schemes for DF-algebras

$$4. \text{ if } \left\{ \begin{array}{l} s_1[u_1 \rightarrow \bar{v}_1]t_1 \in \phi \\ s_2[u_2 \Rightarrow \bar{v}_2]t_2 \in \phi \\ u_1 \doteq u_2 \in \phi, \\ s_1 : s_2 \text{ and } \bar{v}_1 : \bar{v}_2 \in \phi \end{array} \right\} \text{ then } t_1 : t_2 \in \phi, \text{ and}$$

similarly for  $\rightarrow$  in correspondence with  $\Rightarrow$ .

The axiom schemes all DF-algebras have to satisfy are given in Figure 5. There the  $\tilde{\forall}\phi$  formula denotes as usually the universal closure of  $\phi$ .

The (G) axiom expresses the fact that  $\top$  is assumed to be the greatest element in the domain of interpretation. The (AS) axiom corresponds to the antisymmetry of the derivation relation. The (F) axiom states the functionality of single-valued methods. The (WT) axiom gives the well-typing correspondence between values and types for single-valued, and respectively set-valued methods. The typing axiom (T) expresses in a compact form the type inheritance, argument subtyping, and range supertyping principles from the object-orientation paradigm. (One could see the distinct cases if note for instance that  $Y'$  can be just  $X$ , and  $Y'$  can be  $Y$ . In that case the formula states the argument subtyping principle.)

Finally, the (C) axiom is added (as [24] pointed out) in order to avoid empty relation interpretations for the first-order predicates on which DF atomic constraints are built.<sup>8</sup>

**Definition 2.4** *Let  $\phi$  be a DF-formula, and  $\mathcal{A}$  a DF-algebra. A valuation  $\alpha$  in  $\mathcal{A}$  is a solution of  $\phi$  if  $\mathcal{A}, \alpha \models \phi$ .*

The definitions for DF-equivalence, DF-entailment and DF-disentailment relations between formulae are given as expected.

---

<sup>8</sup>  $X$  is constrained in  $\phi$  if there is a constraint  $X : s$ ,  $X[u_q t \bar{v}]t$ , or  $X \doteq t$  in  $\phi$ .

**Definition 2.5** Let  $\mathcal{A}$  be a DF-algebra, and  $\phi$  and  $\psi$  DF-formulae.

$\phi$  and  $\psi$  are equivalent in  $\mathcal{A}$  if  $\mathcal{A} \models \check{\vee}(\phi \leftrightarrow \psi)$ .

$\phi$  entails  $\psi$  in  $\mathcal{A}$  if  $\mathcal{A} \models \check{\vee}(\phi \rightarrow \psi)$ . Notation:  $\phi \models_{\mathcal{A}} \psi$ .

$\phi$  disentails  $\psi$  in  $\mathcal{A}$  if  $\mathcal{A} \models \check{\vee}(\phi \rightarrow \perp)$ . Notation:  $\phi \models_{\mathcal{A}} \neg\psi$ .

$\phi$  determines  $\psi$  in  $\mathcal{A}$  if either  $\phi \models_{\mathcal{A}} \psi$  or  $\phi \models_{\mathcal{A}} \neg\psi$ .

**Definition 2.6** Let  $\phi$  and  $\psi$  be DF-formulae.

$\phi$  is DF-equivalent to  $\psi$  if  $\phi$  and  $\psi$  are equivalent in every DF-algebra  $\mathcal{A}$ .

$\phi$  DF-entails/disentails  $\psi$  if  $\phi$  DF-entails/disentails  $\psi$  in every DF-algebra  $\mathcal{A}$ .

**Proposition 2.1** Every solved DF-clause  $\phi$  is satisfiable.

*Proof:* Let  $\mathcal{R}(\phi)$  be the set of all references occurring in  $\phi$ . Obviously,  $\phi_{\perp}$  determines an equivalence relation  $\equiv_{\phi}$  on  $\mathcal{R}(\phi)$ . Let  $\theta$  be a functional mapping  $\theta : \mathcal{R}(\phi) \rightarrow \mathcal{R}(\phi)$  such that  $\theta(s) = \theta(t)$  iff  $s \equiv_{\phi} t$ . (Such a function can be immediately constructed by choosing one representative for each class in  $\mathcal{R}(\phi)_{/\equiv_{\phi}}$ .)

Let  $\mathcal{A}^{\phi}$  be a DF-algebra  $\mathcal{A}^{\phi} = \langle D^{\phi}, <_{\phi}, I_{\mathcal{C}}^{\phi}, I_{\rightarrow}^{\phi}, I_{\Rightarrow}^{\phi}, I_{\Rightarrow\Rightarrow}^{\phi}, I_{\Rightarrow\Rightarrow}^{\phi}, I_{\bar{b}}^{\phi} \rangle$  such that

$$D^{\phi} = \theta(\mathcal{R}(\phi)),$$

$$s <_{\phi} t \text{ iff } s : t \in \phi,$$

$$I_{\mathcal{C}}(a) = \theta(a), \text{ for every } a \in \mathcal{C},$$

$$I_{qt}^{\phi} \text{ contains } (s, t) \text{ if } s[u_{qt}\bar{v}] \in \phi, \text{ for } qt \neq \bar{b},$$

$$\text{if } I_{\Rightarrow}^{\phi} \text{ contains } (s, t), s' \leq_{\phi} s, \bar{v}' \leq_{\phi} \bar{v}, \text{ and } t \leq_{\phi} t' \text{ then } I_{\Rightarrow}^{\phi} \text{ contains } (s', t'),$$

$$\text{and similarly for } \Rightarrow\Rightarrow.$$

Then the  $\mathcal{A}^{\phi}$ -valuation  $\alpha$  which assigns every  $X$  in  $\mathcal{R}(\phi)$  to  $\theta(X)$  in  $D^{\phi}$  is a solution of  $\phi$  in  $\mathcal{A}^{\phi}$ .  $\square$

The DF-algebra  $\mathcal{A}^{\phi}$  constructed in the proof of the above statement (assuming each  $I_{qt}^{\phi}$  minimal wrt the specified properties) is called in the sequel the **canonical DF-algebra** associated to  $\phi$ . It can be easily converted by isomorphisms to a canonical DF-graph<sup>9</sup> algebra corresponding to  $\phi$ . We show that it generates via DF-homomorphisms

<sup>9</sup>A DF-graph is a tuple

$$G = \langle N, E = E' \cup E'', \mu, r_0 \rangle$$

where

$N$  is a finite set of nodes  $r, s, t, \dots \in \mathcal{R}$ ;

$E \subseteq N \times N$  is the set of derivation edges (making the subset  $E'$ ), and method edges ( $E''$ , which is disjoint from  $E'$ );

$\mu : E'' \rightarrow 2^{\mathcal{C} \times QT \times \mathcal{C}^*}$ , where  $QT = \{\rightarrow, \Rightarrow, \Rightarrow, \Rightarrow\Rightarrow, \bar{b}\}$ , associates labels to method edges; and

$r_0$  is the root of  $G$ .

Every DF-graph  $G$  has to satisfy the following conditions:

(G.AS): there are no derivation cycles in  $G$ , i.e., if  $(s_1, s_2), (s_2, s_3), \dots, (s_{n-1}, s_n) \in E'$  then  $s_n \neq s_1$ ;

(G.F): functional method edges are determinate, i.e., if  $(s, v), (s, v') \in E''$  and  $\mu(s, t) \cap \mu(s, t')$  contains  $(u, \rightarrow, \bar{w})$ , then  $v = v'$ ;

(G.WT): “value” and “type” method edges commute through derivation chains, i.e., if  $(s, v)$  and  $(s, t) \in E''$ ,  $\mu(s, v)$  contains  $(u, \rightarrow, \bar{w})$ , and  $\mu(s, t)$  contains  $(u, \Rightarrow, \bar{w})$ , then there is a derivation chains from  $v$  to  $t$  in  $E'$ ;

(G.T): method edges emerge through derivation edges, i.e., if  $(s, t) \in E''$ ,  $(u, \Rightarrow, \bar{w}) \in \mu(s, t)$ , and there are derivation chains in  $E'$  from  $s'$  to  $s$ , from  $t$  to  $t'$  and from  $\bar{w}$  to  $\bar{w}'$  (obviously, on corresponding components), then  $(s', t') \in E''$  and  $(u, \Rightarrow, \bar{w}') \in \mu(s', t')$ . Similarly, for  $\Rightarrow\Rightarrow$  in correspondence with  $\Rightarrow\Rightarrow$ ;

(G.P): boolean methods edges link only to  $\top$ , which represents the truth value true: if  $(s, t) \in E''$  and  $(u, \bar{b}, \bar{w}) \in \mu(s, t)$ , then  $t = \top$ .

From the implementation point of view, it's conveniently to work with DF-graph generators: graphs which satisfy the conditions (G.AS), (G.F) and (G.P). They are further “lazily” closed through (G.WT) and (G.T).

all solutions of  $\phi$ .

**Definition 2.7** Given  $\mathcal{A}$  and  $\mathcal{B}$  two DF-algebras, a DF-homomorphism from  $\mathcal{A}$  to  $\mathcal{B}$  is an application  $\gamma : D^{\mathcal{A}} \rightarrow D^{\mathcal{B}}$  such that:

$$\begin{aligned} & \gamma(u) <_{\mathcal{B}} \gamma(v), \text{ for every } u, \text{ and } v \text{ in } D^{\mathcal{A}} \text{ such that } u <_{\mathcal{A}} v. \\ & \gamma(I_{\mathcal{C}}^{\mathcal{A}}(a)) = I_{\mathcal{C}}^{\mathcal{B}}(a), \text{ for every } a \in \mathcal{C}; \\ & I_{\gamma(u) \rightarrow \gamma(\bar{v})}^{\mathcal{B}} = \{(\gamma(s), \gamma(t)) \mid (s, t) \in I_{u \rightarrow \bar{v}}^{\mathcal{A}}\}, \text{ for every } u \in D^{\mathcal{A}}, \text{ and } \bar{v} \in (D^{\mathcal{A}})^*; \\ & I_{\gamma(u) \Rightarrow \gamma(\bar{v})}^{\mathcal{B}} \supseteq \{(\gamma(s), \gamma(t)) \mid (s, t) \in I_{u \Rightarrow \bar{v}}^{\mathcal{A}}\}, \text{ for every } u \in D^{\mathcal{A}}, \text{ and } \bar{v} \in (D^{\mathcal{A}})^*; \\ & \text{and similarly for } \Rightarrow, \text{ and } \Rightarrow; \text{ and} \\ & I_{\gamma(u) \text{b} \gamma(\bar{v})}^{\mathcal{B}} \supseteq \{(\gamma(s) \mid s \in I_{u \text{b} \bar{v}}^{\mathcal{A}})\}, \text{ for every } u \in D^{\mathcal{A}}, \text{ and } \bar{v} \in (D^{\mathcal{A}})^*. \end{aligned}$$

The following results justify for the canonicity of the algebra  $\mathcal{A}^{\phi}$  relative to the satisfiability of  $\phi$ . The next section will prove that we can find for every DF-clause  $\phi$  a DF-equivalent solved form.

**Proposition 2.2** Let  $\gamma : \mathcal{A} \rightarrow \mathcal{B}$  be a DF-homomorphism,  $\alpha$  and  $\beta$  valuations in  $\mathcal{A}$ , respectively  $\mathcal{B}$ , and  $\phi$  a DF-constraint. If  $\mathcal{A}, \alpha \models \phi$ , then  $\mathcal{B}, \gamma(\beta) \models \phi$ .

*Proof:* Immediately, due to the definitions of DF-homomorphism and satisfiability of DF-constraints.  $\square$

In fact, the above property is true for  $\phi$  any DF-formula. (To show this it is sufficient to consider the disjunctive normal form of  $\phi$ .)

**Proposition 2.3** If  $\phi$  is a solved DF-clause,  $\mathcal{A}^{\phi}$  is the canonical DF-algebra associated to  $\phi$ ,  $\alpha$  is a solution of  $\phi$  in  $\mathcal{A}^{\phi}$ , and  $\beta$  is a solution of  $\phi$  in the DF-algebra  $\mathcal{A}^{\phi}$  then there is  $\gamma : \mathcal{A}^{\phi} \rightarrow \mathcal{B}$  DF-homomorphism such that  $\beta = \gamma \circ \alpha$ .

*Proof:* Let  $\theta$  be the application defined in the proof of Proposition 2.1. Then  $\gamma : D^{\mathcal{A}^{\phi}} = \mathcal{R}(\phi) \rightarrow D^{\mathcal{B}}$  by  $\gamma(s) = \beta(\theta(s))$  satisfies the stated claim.  $\square$

### 3 DF Normalization

This section gives the operational counterpart for solving DF-clause first, and then for the general case of DF-formulae. All this work is done using rewriting rules.

So, solving (positive) DF-clauses is achieved by simple (or: basic) normalization rules.

We introduce then the notions of definite DF-constraint and definite DF-clause. Definite normalization rules compute a solved form for a given definite DF-clause. Thus the way is prepared for doing relative simplification of a (positive) DF-clause wrt a definite DF-clause, and this task is realized by relative normalization rules.

Relative normalization rules cover both DF-term unification (corresponding in fact to term matching in the OSF idea) and DF resolution, as we allow the clause relative to which the simplification is performed to be a definite one.<sup>10</sup>

One subsection is dedicated to each of these three groups of normalization rules. One more subsection introduces an enhanced syntax for DF-constraints – using multi-sorted variables – together with its operational semantic counterpart, used in writing down in a compact manner the rules in the last two groups.

<sup>10</sup>Unification and resolution in F-logic is thus replaced in our approach by two sets of conditional rewriting rules: the first transforms the program into an equivalent normal form, the second performs simultaneously conditional unification and resolution.

---


$$\begin{array}{l}
(\text{EquE}) \quad \frac{t \doteq t \wedge \phi}{\phi} \\
(\text{VarE}) \quad \frac{V \doteq t \wedge \phi}{V \doteq t \wedge \phi(V/t)} \quad \text{if } V \text{ occurs in } \phi \\
(\text{EquR}) \quad \frac{a \doteq V \wedge \phi}{V \doteq a \wedge \phi} \\
(\text{ConR}) \quad \frac{a \doteq b \wedge \phi}{a \doteq b \wedge \phi(a/b)} \quad \text{if } a \neq b \text{ and } a \text{ occurs in } \phi
\end{array}$$


---

Figure 6: Equational Basic Normalization Rules

### 3.1 DF Basic Normalization

Now we start the presentation of basic (or: simple) **DF-normalization** rules. Given a (positive) DF-clause  $\phi$ , they put it under an equivalent solved form.

There are two groups of DF basic normalization rules. Those in the first group, the so-called **equational** rules concern the equational part of the clause to be normalized and are given in Figure 6. They do eliminate trivial equations (EquE), substitute variables by the references they are equated to, propagate their effect into the whole clause (VarE and EquR), and also rewrite the clause with respect to the congruence relation induced on the alphabet  $\mathcal{C}$  by the equational part of  $\phi$  (ConR).

In writing these rules we denoted by  $\phi(s/t)$  the formula obtained from  $\phi$  by replacing all occurrences of  $s$  by  $t$ . Also, it is assumed that equations are seen as oriented, i.e.,  $a \doteq b$  is not the same with  $b \doteq a$ . This assumption is needed to ensure the termination of normalization process.

The second group of basic normalization rules are the so-called **core** rules, given in Figure 7. They provide for new entries to the equational part of the clause – via antisymmetry (AS) and functionality (F), or to the derivational part of the clause – via well-typing conditions (WT).

Note that normalization acts like a communication process between the three distinct parts of the clause  $\phi$ , namely  $\phi_{=}$ ,  $\phi_{\cdot}$  and  $\phi_{[]}$ . Using the terminology that appears in [20], each normalization rule reads information from one part’s “blackboard” (except for (F) and (WT) that read from two blackboards), and write information on another part’s blackboard (except for (VarE) and (ConR) which write on all blackboards).

Basic normalization rules apply as time as possible on a given DF-clause  $\phi$ , without assuming any preferential application order.

**Proposition 3.1** *Basic DF-normalization rules are finite terminating.*

*Proof:* There is only a finite number of derivations and equations that can be added to  $\phi$  (via (WT), respectively (AS) and (F)) due to the finite number of constraints in  $\phi$  which implies also that there are only finitely many references in  $\phi$ . Only the

---

(AS) 
$$\frac{t : s \wedge s : t \wedge \phi}{s \doteq t \wedge \phi}$$

(F) 
$$\frac{s[u \rightarrow \bar{v}]t \wedge s'[u' \rightarrow \bar{v}']t' \wedge \phi}{t \doteq t' \wedge s[u \rightarrow \bar{v}]t \wedge s'[u' \rightarrow \bar{v}']t' \wedge \phi}$$
  
if  $s \doteq s', u \doteq u', \bar{v} \doteq \bar{v}' \in \phi$  and  $t \doteq t' \notin \phi$

(WT) 
$$\frac{s[u \rightarrow \bar{v}]t \wedge s'[u' \rightarrow \bar{v}']t' \wedge \phi}{t : t' \wedge s[u \rightarrow \bar{v}]t \wedge s'[u' \rightarrow \bar{v}']t' \wedge \phi}$$
  
if  $u \doteq u', s : s', \bar{v} \doteq \bar{v}' \in \phi$ , and  $t : t' \notin \phi$

Similarly for  $\rightarrow$  in correspondence with  $\Rightarrow$ .

---

Figure 7: Core Basic Normalization Rules

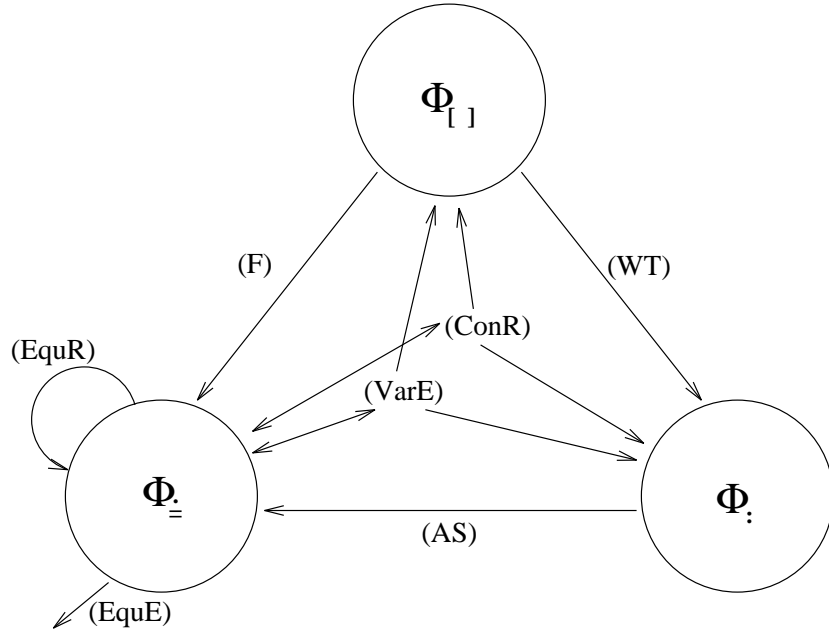


Figure 8: Basic normalization rules as communication processes

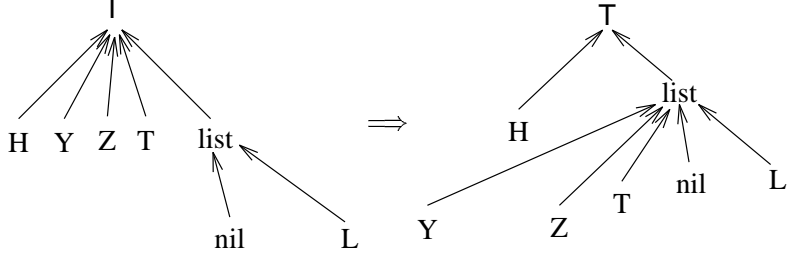


Figure 9: Dynamic typing of variables in DF

derivational and the equational sub-parts of  $\phi$  can increase in cardinality during the normalization process, but this increasing is limited due to the same reasons stated above. Finally, note that only one rule application is possible for every equation constraint in  $\phi$ .  $\square$

**Proposition 3.2** *Every DF-clause  $\phi$  normalizes into an equivalent solved clause  $\phi'$ .*

*Proof:*  $\phi'$  results in a solved form due to the way the normalization rules are defined, and  $\phi \models_{DF} \phi'$ , that is  $\models_{DF} \check{\forall}(\phi \leftrightarrow \phi')$ , due to the (AS), (F) and (FT) normalizations rules – DF-axioms correspondence.  $\square$

**Example 3.1** *The conjunction of the DF-clauses corresponding to the DF-terms (T2) and (T4) in Example 2.2 will gain by normalization the sort specifications*

$$T:\text{list} \wedge Z:\text{list} \wedge Y:\text{list}.$$

*The dynamics of sort characterization of variables through the normalization process is shown in Figure 9. (Prefacing what comes in the next subsection,  $\top$  is assumed to be the greatest element in the is-a hierarchy.)*

*It is important to note that DF allows in this way a fine run-time type checking of variables.*

*The graph corresponding to the normal form of the DF-term conjunction  $T = (T1) \wedge (T2) \wedge (T3) \wedge (T4)$  is shown in Figure 10. Dashed arrows correspond to methods while plain arrows denote the is-a relation. Together with its subgraphs it generates the canonical DF-graph algebra associated to  $T$ .*

### 3.2 Reference Signatures

This rather technical subsection presents unification on (simply first, and then conditional) reference signatures. Its rôle is to ensure a compact formulation of definite and relative normalization rules to be presented in the following subsections. We hope the reader will not get lost into details. If he assumes an intuitive understanding of the just mentioned notions, he can skip that subsection at the first reading, returning eventually later to it.

From now on we use an enhanced syntax for DF-constraints, namely imposing the use of sorted variables (in fact already allowed in the  $\chi$ -term writing). This enhanced syntax allows for a compact formulation of all rewriting rules to be further presented.

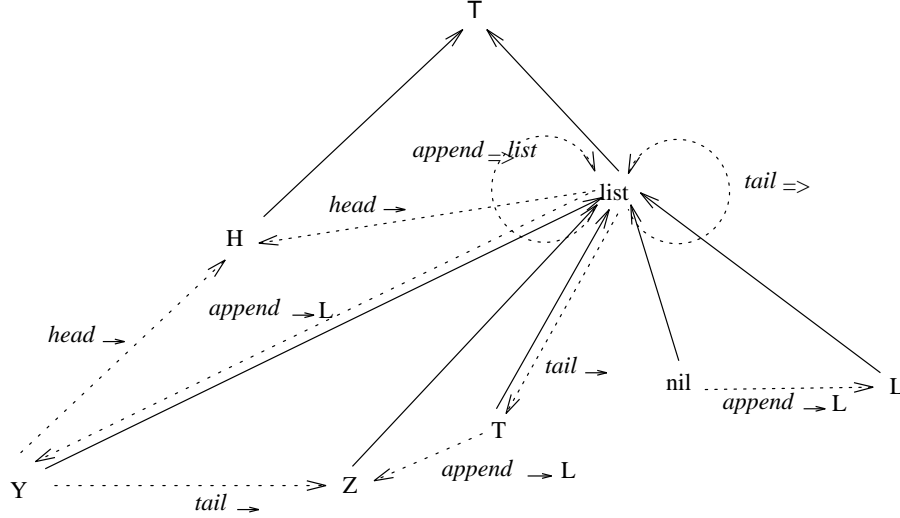


Figure 10: The graph corresponding to the solved form of  $T$

Let  $\mathcal{C} = \{a, b, \dots\}$  be a set of constants, and  $\mathcal{V} = \{X, Y, \dots\}$  a countably infinite set of variables. We assume that  $\mathcal{C}$  contains the special symbol  $\top$  read “top”.

A sort mapping  $\sigma : \mathcal{V} \rightarrow \mathcal{C}^*$  associates to each variable  $X$  as finite non-empty set of constants  $\{a_1, \dots, a_n\}$ . The pair  $(X, \sigma(X))$  is called **sorted variable** and is simply written  $X^{a_1, \dots, a_n}$ . We assume that for each finite set of constants  $S$  there are infinite many variables  $X$  such that  $\sigma(X) = S$ . The variables whom  $\sigma$  associates the top sort  $\top$  are called **simple variables** and are commonly written  $X$  instead of  $X^\top$ .

In the sequel a **reference** is meant to be either a constant or a sorted variable.

Obviously, every DF-clause can be equivalently rewritten in the enhanced DF-syntax, that means using sorted variables.

Note also that the (base) normalization rules presented in the precedent subsection preserve their formulation when the clause under normalization is written using the present enhanced syntax.

Sort characterization for variables acts like using guards to the application of normalization rules.

### 3.2.1 Simple reference signatures

**Definition 3.1** Let  $\mathcal{R} = \mathcal{C} \cup \mathcal{SV}$ . A relation  $<$  on  $\mathcal{R}$  is a subset of  $\mathcal{R} \times \mathcal{R}$ .

The preorder relation determined by  $<$  on  $\mathcal{R}$  is  $\leq$ , the reflexive and transitive closure of  $<$  on  $\mathcal{R}$ . If  $s \leq t$ , then  $s$  is called **descendent** of  $t$ , which in turn is called an **ancestor** of  $s$ .

The notion of **admissible instantiation** on  $\mathcal{R}$  wrt  $<$  is defined as follows:

**Definition 3.2** Let  $<$  be a relation on  $\mathcal{R}$ .

- i. Every  $a \in \mathcal{C}$  is admissible instance of  $a$  wrt  $<$ , and
- ii.  $s$  is an admissible instance of  $X^{a_1, \dots, a_n}$  wrt  $<$  if  $s \leq a_i, i = 1, \dots, n$ .

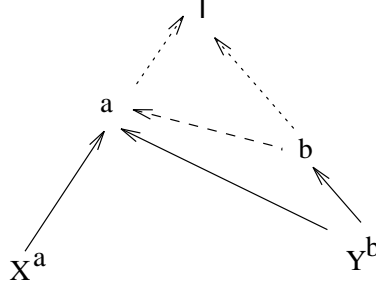


Figure 11: A reference signature

Usually, when speaking about admissible instances of a reference, we will no more mention the relation  $<$  to which we refer, if it is clear enough from the context.

If  $s$  is an admissible instance of  $X^{a_1, \dots, a_n}$ , then  $X^{a_1, \dots, a_n}$  is called **meta-reference** of  $s$ . The idea behind is that  $X^{a_1, \dots, a_n}$  is more general than  $s$ . In fact a sorted variable is thought of as a recipient for common properties of all descendents from  $a_1, \dots$ , and  $a_n$ .

It should be noted that in the DF system a descendent sort inherits from its ancestors only the typing methods. Inheritance of all methods from one class to another is allowed via admissible instantiation.

**Definition 3.3** *A reference signature is a relation  $<$  on  $\mathcal{R}$  such that:*

- i.  $X^{a_1, \dots, a_n} < a_i$ , for  $i = 1, \dots, n$ , for every  $X^{a_1, \dots, a_n} \in \mathcal{SV}$ ;
- ii.  $\top$  is greatest element of  $\mathcal{R}$  wrt  $<$ , i.e.,  $s \leq \top, \forall s \in \mathcal{R}$ , and  $\neg \exists t \in \mathcal{R}, \top < t$ .
- iii.  $<$  is closed under admissible instantiation on  $\mathcal{R}$ , i.e., for all  $s < t$ , if  $s'$  is admissible instance of  $s$ , and  $t'$  admissible instance of  $t$ , it follows that  $s' < t'$ .

It's easy to see that any reference is an admissible instance of itself.

Obviously, one can see the reference signature generated by a subset  $S$  of  $\mathcal{R} \times \mathcal{R}$  as the minimal signature containing  $S$ . Indeed, there is a “total” signature  $\mathcal{R} \times \mathcal{R}$ , and the intersection operation preserves the defining characteristics for reference signatures.

We say that a reference signature  $<$  is finite if it is generated by a finite subset of  $\mathcal{R} \times \mathcal{R}$ .

**Example 3.2** *The relation  $\{(X^a, a), (Y^b, b), (Y^b, a)\}$  is completed up to admissible instantiation by adding the pair  $(b, a)$ , and becomes a reference signature if we extend it considering also  $(a, \top), (b, \top)$ . A graphical representation of this signature is given in the Figure 11.*

*With respect to the above signature,  $X^a$  is a meta-reference of  $Y^b$  since  $Y^b \leq a$ .*

Now we come naturally to the definition of admissible substitution with respect to a given signature on  $\mathcal{R}$ .

**Definition 3.4** *Let  $<$  be a signature on  $\mathcal{R}$ . A mapping  $\sigma : \mathcal{SV} \rightarrow \mathcal{R}$  is an admissible substitution wrt  $<$  if*

- i.  $\sigma(V)$  is an admissible instance of  $V$  for every  $V \in \mathcal{SV}$ ;
- ii. For every  $V \in \mathcal{SV}$  such that  $V \neq \sigma(V)$  there is a ground admissible instance  $a$  of  $\sigma(V)$  in  $\mathcal{R}$ .

We have to say that the second condition in the above definition is added only from reasons of practical convenience. All the results reported in the sequel remain true if we eliminate it. It's rôle is (only) to not allow the proliferation of unsatisfiable constraints during the normalization processes on DF-clauses.

As usually, an admissible substitution  $\sigma$  is considered finite if its domain  $D(\sigma) = \{V \in \mathcal{SV} \mid V \neq \sigma(V)\}$  is finite. In fact, in the sequel we will work only with finite admissible substitutions.

Given  $<$ , a relation on  $\mathcal{R}$ , one can naturally associate it the equivalence relation determined by cycles in  $<$ , namely  $s \equiv t$  iff  $s \leq t$  and  $t \leq s$ .

It's easy to see that given the point *iii.* in the definition for reference signature, it follows that an admissible substitution preserves the equivalence relation between references, i.e., if  $s \equiv t$ , then  $\sigma(s) \equiv \sigma(t)$ .

Obviously, whenever it's possible, we will ask the signature to be processed to come with no cycles. But we will see in the sequel that that is not always the case (in the beginning).

**Definition 3.5** *A normal signature is one for which the determined equivalence relation is the identity on  $\mathcal{R}$ .*

We will define now the unification notion in the new settings. It is a generalization for domain variable unification [17] in that

1. it involves a hierarchy  $<$  over domains and
2. it is in fact an equational unification wrt  $\equiv$ , the equivalence relation determined by  $<$ .

**Definition 3.6** *Let  $<$  be a reference signature on  $\mathcal{R}$ , and  $s$  and  $t$  two references. We say that  $s$  unifies with  $t$  wrt  $<$  if there is an admissible substitution  $\sigma$  such that  $\sigma(s) \equiv \sigma(t)$  (i.e.,  $\sigma(s) = \sigma(t)$ ) or there is a cycle in  $<$  to which both  $\sigma(s)$  and  $\sigma(t)$  belong).*

In the virtue of the above definition two references equivalent wrt  $<$  are always unifiable.

Given  $<$ , a relation on  $\mathcal{R}$ , then  $<$  is naturally rewritten for  $\mathcal{R}/\equiv$ . (We take  $[a] < [b]$  iff  $a \leq b$  and  $[a] \neq [b]$ ). Given  $S \subseteq \mathcal{R}$ , and  $<$  a relation on  $\mathcal{R}$ , we define  $\min(S/\equiv)$  the set of the least elements of  $S/\equiv$  wrt  $<$ .

*Remark:* For every  $X^{a_1, \dots, a_n}$ , and  $Y^{b_1, \dots, b_m} \in \mathcal{SV}$  there is  $Z^{c_1, \dots, c_k} \in \mathcal{SV}$  such that  $Z^{c_1, \dots, c_k}$  is meta-reference for both  $X^{a_1, \dots, a_n}$  and  $Y^{b_1, \dots, b_m}$ . This claim is true for any  $Z^{c_1, \dots, c_k}$  such that  $\{[c_1], \dots, [c_k]\} = \min(\{[a_1], \dots, [a_n]\} \cup \{[b_1], \dots, [b_m]\}/\equiv)$ , and then for every  $W^{d_1, \dots, d_q}$  such that for every  $i$  from 1 to  $k$  there is  $j \in \{1, \dots, q\}$ , with  $d_j < c_i$ .

Let's prove that  $Z^{c_1, \dots, c_k}$  chosen as above is a most general (admissible) unifier of  $X^{a_1, \dots, a_n}$  and  $Y^{b_1, \dots, b_m}$  wrt  $<$ . If  $V^{e_1, \dots, e_q}$  is an admissible instance for both  $X^{a_1, \dots, a_n}$  and  $Y^{b_1, \dots, b_m}$ , then  $V^{e_1, \dots, e_q} \leq X^{a_1, \dots, a_n}$  and  $V^{e_1, \dots, e_q} \leq Y^{b_1, \dots, b_m}$ . It follows, given the way  $c_1, \dots, c_k$  were chosen, that  $V^{e_1, \dots, e_q} \leq c_l$  for  $l = 1, \dots, k$ . Thus  $V^{e_1, \dots, e_q}$  is proven to be an admissible instance of  $Z^{c_1, \dots, c_k}$ .

In connection with the above remark, we should say that the second condition in the admissible substitution definition limits the possibility of reference unifier choice to "effective" candidats wrt the given reference signature.

The definition of more general unifier and most general unifier for two references are given as usually. Also, these notions extend naturally for finite sequences of references, DF-constraints and DF-formulae.

Figure 12 presents an algorithm for admissible unification of references wrt to a given signature on  $\mathcal{R}$ . The correctness of this algorithm is immediate, given the above remark.

**Example 3.3** *With respect to the reference signature in the preceding example,  $X^a$  and  $Y^b$  unify, and one of their most general unifier is  $Y^b$ . Let's take now a little more complicated signature:*

*The reference signature generated by  $\{b : a, Z^c \doteq Y^b\}$  is shown in Figure 13.*

*With respect to it,  $\sigma = \{X^a/Y^b\}$  is a most general unifier of  $X^a$  and  $Y^b$ . There are also other unifiers of them, like  $\theta = \{X^a/Z^c, Y^b/Z^c\}$  and  $\theta = \{X^a/c, Y^b/c\}$ , but they are not most general ( $\theta = \{Y^b/Z^c\} \circ \sigma$ , and  $\theta' = \{Y^b/c\} \circ \sigma = \{Z^c/c\} \circ \sigma$ ). Also,  $b$  unifies with  $c$ , but not with  $a$ .*

Before ending this subsection, we add some considerations that make a link between DF and CFT systems:

Note that every signature  $<$  can be seen as an infinite conjunction of DF derivation constraints.

Conversely, given  $\phi$  a DF-clause, the reference signature defined by  $\phi$  is the signature generated by

$$<_{\phi} = \{(s, t) \mid s : t, s \doteq t \text{ or } t \doteq s \in \phi\}.$$

In the sequel we will deliberately use the same denotation for both  $<_{\phi}$  and the signature it generates.

**Definition 3.7** *Let  $\phi$  be a DF-clause,  $\mathcal{R}(\phi)$ , the set of all references occurring in  $\phi$ , and  $\equiv_{\phi}$  the equivalence relation determined by  $<_{\phi}$  on  $\mathcal{R}(\phi)$ .*

*A normalizer of  $\phi$  is a mapping  $\theta : \mathcal{R}(\phi) \rightarrow \mathcal{R}(\phi)$  such that  $\theta(s) = \theta(t)$  iff  $s \equiv_{\phi} t$ .*

Obviously, we can see every normalizer  $\theta$  of  $\phi$  as an equational DF-clause (i.e., made only of equations), and also as an admissible substitution (wrt  $<_{\phi}$ ).

If  $\phi$  is a solved DF-clause, and  $\theta$  is a normalizer of  $\phi$ , let  $\phi'$  be the formula obtained from  $\phi$  by eliminating all equations from  $\phi$  and replacing every reference  $s$  with  $\theta(s)$ . Then

$$\phi \models_{DF} \theta \wedge \phi'$$

We have thus a DF counterpart for the normalizer notion introduced in [23]. (To see it, one can rewrite the precedent formula by abuse of notation, as  $\phi \models_{DF} \theta \wedge \theta\phi$ .)

Obviously, the normalization rules presented in the precedent subsection compute for every DF-clause  $\phi$  – together with a solved form – a normalizer of it.

### 3.2.2 Conditional Reference Signatures

Now we extend the definitions in the preceding subsection to conditional reference signatures. We denote a DF-formula in disjunctive normal form as a set of positive DF-clauses. The empty DF-clause is considered always  $\top$  (true), while the empty set of DF-clauses is considered always  $\perp$  (false).

---

### Admissible Unification Algorithm

Input:  $s, t \in \mathcal{R}$ , and  
 $<$ , a finite reference signature (in fact a finite generator of it)  
Output:  $\sigma$ , a most general unifier of  $s$  and  $t$ , if  $s$  and  $t$  are unifiable wrt  $<$ , and  
'Fail', otherwise.

Procedure:

```

if  $s \equiv t$ 
  return  $\sigma = \varepsilon$  (the empty substitution);
else
if  $s = a, t = b$ 
  return 'Fail';
else
if  $s = a, t = Y^{b_1, \dots, b_m}$  (or viceversa)
  if  $a \leq b_1, \dots, b_m$ 
    return  $\sigma = \{t/Y^{b_1, \dots, b_m}\}$ ;
  else return 'Fail';
else ( $s = X^{a_1, \dots, a_n}, t = Y^{b_1, \dots, b_m}$ )
  {
  compute  $\{[c_1], \dots, [c_k]\} = \min(\{a_1, \dots, a_n\} \cup \{b_1, \dots, b_m\})/\equiv$ ;
  if  $\exists d \in \mathcal{C}$  such that  $d \leq c_i$ , for  $i = 1, \dots, k$ 
    if  $\{[c_1], \dots, [c_k]\} = \{[a_1], \dots, [a_n]\}$ 
      return  $\sigma = \{Y^{b_1, \dots, b_m}/X^{a_1, \dots, a_n}\}$ 
    else
      if  $\{[c_1], \dots, [c_k]\} = \{[b_1], \dots, [b_m]\}$ 
        return  $\sigma = \{X^{a_1, \dots, a_n}/Y^{b_1, \dots, b_m}\}$ 
      else
        return  $\sigma = \{s/Z^{c_1, \dots, c_k}, t/Z^{c_1, \dots, c_k}\}$ ; ( $Z$  is new variable)
    else
      return 'Fail';
  }

```

---

Figure 12:

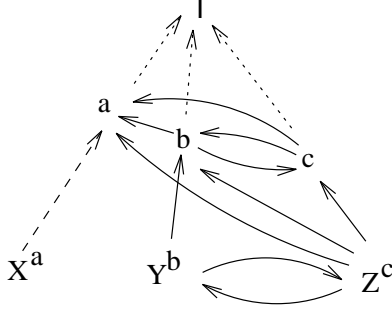


Figure 13:

**Definition 3.8** A conditional reference signature  $(<, \Delta)$  is a reference signature on  $\mathcal{R}$  together with a conditioning mapping  $\Delta : < \rightarrow 2^{\mathcal{D}}$ , (where  $\mathcal{D}$  is the set of all positive DF-clauses over  $\mathcal{R}$ ), such that

- i.  $\Delta(X^{a_1, \dots, a_n}, a_i) = \top$ , for  $i = 1, \dots, n$  and every  $X^{a_1, \dots, a_n} \in \mathcal{SV}$ ;
- ii.  $\Delta(s, \top) = \top$ , for every  $s < \top$ , and
- iii. For all  $s < t$  and every  $\sigma$  admissible substitution wrt  $<$ , it follows that  $\Delta(\sigma(s), \sigma(t)) \supseteq \sigma\Delta(s, t)$ .

Intuitively, a DF-clause  $\delta \in \Delta(s, t)$  denotes a sufficient condition under which the derivation  $s : t$  is true.

Note that the pair  $(\sigma(s), \sigma(t))$  at the point *iii.* in the above definition belongs to the reference signature  $<$  due to the point *iii.* in the (simple) signature definition and to the definition given to the notion of admissible substitution.

Now, the previous introduced notions for a (simple) reference signature generalize naturally to the conditional case:

**Definition 3.9** Let  $(<, \Delta)$  be a conditional reference signature over  $\mathcal{R}$ . A reference  $s$  is a descendent of the reference  $t$  under the condition  $\delta$ , and this fact is denoted by  $s \leq^\delta t$ , if

- $\exists r_1, \dots, r_k \in \mathcal{R}$  such that  $s = r_1 < r_2 < \dots < r_k = t$  and
- $\exists \delta_i \in \Delta(r_i, r_{i+1})$ , for  $i = 1, \dots, k-1$  such that  $\delta = \delta_1 \wedge \dots \wedge \delta_{k-1}$ .

Obviously,  $s \in \mathcal{R}$  is an admissible instance of  $X^{a_1, \dots, a_n}$  wrt  $<$  under the condition  $\delta$  if  $s \leq^{\delta_i} a_i$  for  $i = 1, \dots, n$ , and  $\delta = \delta_1 \wedge \dots \wedge \delta_{k-1}$ .

Now we define the “normality” of conditional reference signatures:

**Definition 3.10** A conditional signature  $(<, \Delta)$  is normal if the its unconditional subset is normal, i.e.,  $s \leq^\top t, t \leq^\top s \Rightarrow s = t$ .

Usually, the fact  $s \leq^\top t$  will be in the sequel written simply  $s \leq t$ , and in the graphical representation associated to a conditional signature, only the non- $\top$  conditions will be explicitly written (as labels on the corresponding edges).

### 3.3 DF Definite Normalization

This subsection extends normalization presented by subsection 3.1 to (DF theories put – via Skolemization – under the form of) definite DF-clauses.

**Definition 3.11** A DF-definite constraint is a disjunction  $A \vee \neg B_1 \vee \dots \vee \neg B_n$ , where  $A, B_1, \dots, B_n$  are DF-atomic constraints. (Notation:  $A :- B_1, \dots, B_n$ .)

**Definition 3.12** A DF-definite clause is a finite conjunction of definite DF-constraints.

A definite DF-constraint can be naturally rewritten using the enhanced syntax formally introduced in the beginning of this section using  $\dashv$  – i.e., copying – in the head of the clause the sort characterization of variables given in the clause body.

**Example 3.4** The definite clauses (C1) and (C3) in Example 1.2 correspond to:

$$\begin{aligned} (C1) \quad & X^{person}[happy] :- X^{person}[friend \dashv] Y \\ (C3) \quad & Z[F^{symmetric} \dashv] W :- W[F^{symmetric} \dashv] Z. \end{aligned}$$

The notion of conditional reference signature determined by a definite DF-clause used in the sequel is easily definable starting from that given for positive DF-clauses.

**Definition 3.13** A DF-definite clause  $\phi$  is solved if

1. the conditional reference signature  $\langle \phi \rangle$  determined by  $\phi$  is normal;
2. if  $s \doteq t \in \phi$  then  $s$  does not have other occurrence in  $\phi$ ;
3. if  $\left\{ \begin{array}{l} s_1[u_1 \dashv \bar{v}_1]t_1 :- \delta_1 \in \phi \\ s_2[u_2 \dashv \bar{v}_2]t_2 :- \delta_2 \in \phi, \text{ and} \\ mgu_{\langle \phi \rangle}(\langle s_1, u_1, \bar{v}_1 \rangle, \langle s_2, u_2, \bar{v}_2 \rangle, \sigma, \delta) \end{array} \right\}$  then  $\sigma(t_1 \doteq t_2 :- \delta \wedge \delta_1 \wedge \delta_2) \in \phi$ ,
4. if  $\left\{ \begin{array}{l} s_1[u_1 \dashv \bar{v}_1]t_1 :- \delta_1 \in \phi \\ s_2[u_2 \dashv \bar{v}_2]t_2 :- \delta_2 \in \phi \\ \text{unify}_{\langle \phi \rangle}(u_1, u_2, \sigma, \delta), \\ \sigma s_1 \leq_{\phi}^{\delta'} \sigma s_2, \sigma \bar{v}_1 \leq_{\phi}^{\delta''} \sigma \bar{v}_2, \text{ and} \\ \sigma \text{ is most general with these properties} \end{array} \right\}$  then  $\sigma(t_2 : t_1 :- \delta \wedge \delta' \wedge \delta'') \in \phi$

Similarly for  $\dashv$  in correspondence with  $\dashv$ .

In the above definition we denoted by  $s \leq_{\phi}^{\delta} t$  the fact that  $s$  is an admissible instance for  $t$  wrt  $\langle \phi \rangle$  under the condition  $\delta$ . Also  $mgu_{\langle \phi \rangle}(s, t, \sigma, \delta)$  denotes the fact that  $\sigma$  is a most general admissible unifier of  $s$  and  $t$  wrt  $\langle \phi \rangle$  under the condition  $\delta$ .

Definite normalization process elaborates equational normalization rules and core normalization rules, shown in Figure 14, respectively Figure 15. They are slightly different versions of their basic counterparts. The differences that had to be covered are due to the (assumed) universal closure of the definite DF-clause.

(It is for this reason that we elaborated the precedent subsection and use now, in writing these rules – and also the rules in the next subsection – the notions of admissible instance and most general admissible unifier wrt the determined signature.)

Two mentions have to be done:

1. Definite antisymmetry dissociates into two rules – (D.AS) and (D.CC) – corresponding to the derivational and respectively the equational part of  $\phi$ . The “congruence closure” rule (D.CC) carries an equation between two references over all (pairs of) admissible instances of them in  $\phi$ .

2. A sound definite normalization is performed if (D.F) and (D.WT) apply only when equational rules first and then (D.AS) and (D.CC) can no more be applied.

Note that the definite constraints introduced by (D.F) and (D.WT) can be renamed apart from the definite clause  $\phi$  before joining it.

---

(D.EquE)  $\frac{t \doteq t : -\delta \wedge \phi}{\phi}$

(D.VarE)  $\frac{X^{a_1, \dots, a_n} \doteq Y^{b_1, \dots, b_m} \wedge \phi}{X = Z^{a_1, \dots, a_n, b_1, \dots, b_m} \wedge Y = Z^{a_1, \dots, a_n, b_1, \dots, b_m} \wedge \phi(X/Z, Y/Z)}$   
if  $X$  occurs in  $\phi$  ( $Z$  is a new variable)

$\frac{X^{a_1, \dots, a_n} \doteq a \wedge \phi}{X^{a_1, \dots, a_n} \doteq a \wedge a : a_1 \wedge \dots \wedge a : a_n \wedge \phi(X^{a_1, \dots, a_n}/a)}$   
if  $X^{a_1, \dots, a_n}$  occurs in  $\phi$

(D.EquR)  $\frac{a \doteq V \wedge \phi}{V \doteq a \wedge \phi}$

(D.ConR)  $\frac{a \doteq b \wedge \phi}{a \doteq b \wedge \phi(a/b)}$  if  $a \neq b$  and  $a$  occurs in  $\phi$

---

Figure 14: Equational Definite Normalization Rules

---

(D.AS)  $\frac{t : s \wedge \phi}{s \doteq t \wedge \phi}$  if  $s \leq_\phi t$

(D.CC)  $\frac{s \doteq t \wedge \phi}{s' \doteq t' :- \delta \wedge s \doteq t \wedge \phi}$  if  $\langle s', t' \rangle \leq_\phi^\delta \langle s, t \rangle$

(D.F)  $\frac{s_1[u_1 \rightarrow \bar{v}_1]t_1 :- \delta_1 \wedge s_2[u_2 \rightarrow \bar{v}_2]t_2 :- \delta_2 \wedge \phi}{\sigma(t_1 \doteq t_2 :- \delta \wedge \delta_1 \wedge \delta_2) \wedge s_1[u_1 \rightarrow \bar{v}_1]t_1 :- \delta_1 \wedge s_2[u_2 \rightarrow \bar{v}_2]t_2 :- \delta_2 \wedge \phi}$   
if  $mg u_{<\phi}(\langle s_1, u_1, \bar{v}_1 \rangle, \langle s_2, u_2, \bar{v}_2 \rangle, \sigma, \delta)$  and  
 $\sigma(t_1 \doteq t_2 :- \delta \wedge \delta_1 \wedge \delta_2) \notin \phi$

(D.WT)  $\frac{s_1[u_1 \rightarrow \bar{v}_1]t_1 :- \delta_1 \wedge s_2[u_2 \rightarrow \bar{v}_2]t_2 :- \delta_2 \wedge \phi}{\sigma(t_2 : t_1 :- \delta \wedge \delta' \wedge \delta'' \wedge \delta_1 \wedge \delta_2) \wedge s_1[u_1 \rightarrow \bar{v}_1]t_1 :- \delta_1 \wedge s_2[u_2 \rightarrow \bar{v}_2]t_2 :- \delta_2 \wedge \phi}$   
if  $unify_{<\phi}(u_1, u_2, \sigma, \delta), \sigma s_1 \leq_\phi^{\delta'} \sigma s_2, \sigma \bar{v}_1 \leq_\phi^{\delta''} \sigma \bar{v}_2,$   
 $\sigma$  is most general with these properties, and  
 $\sigma(t_2 : t_1 :- \delta \wedge \delta' \wedge \delta'' \wedge \delta_1 \wedge \delta_2) \notin \phi$   
Similarly for  $\rightarrow$  in correspondence with  $\Rightarrow$ .

---

Figure 15: Definite (Core) Normalization Rules

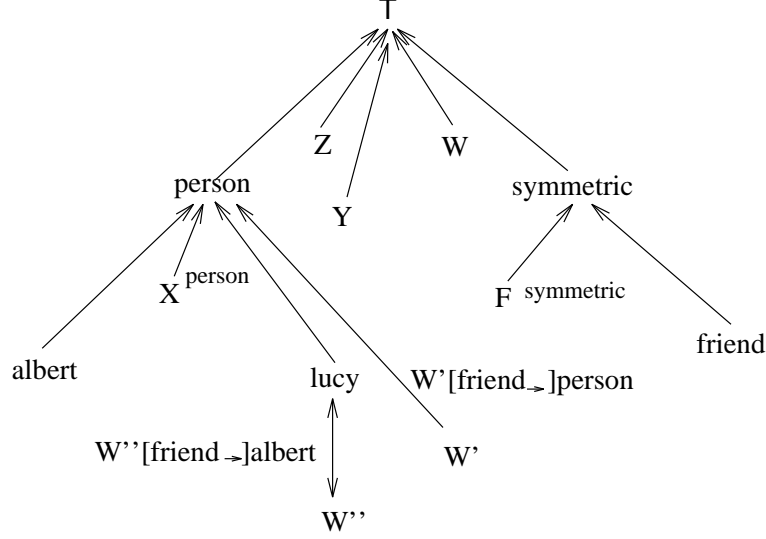


Figure 16:

**Proposition 3.3** *Every definite DF-clause  $\phi$  is satisfiable if it is satisfiable in the DF-graph algebra<sup>11</sup>  $\mathcal{G}$ .*

*Proof:* Classical, as there is a DF-homomorphism from  $\mathcal{G}$  to every DF-algebra  $\mathcal{A}$  and viceversa.  $\square$

**Example 3.5** *Definite normalization of the definite DF-clause  $C$  corresponding to the DF program in Example 1.2 produces the following three new clauses:*

- |   |                |
|---|----------------|
| (D1) $lucy : person$                                      | (C2)+(C6)+(C5) |
| (D2) $W' : person :- W'[friend_{\rightarrow}]person$      | (C3)+(C2)+(C4) |
| (D3) $W'' \doteq lucy :- W''[friend_{\rightarrow}]albert$ | (C3)+(C6)+(C4) |

Figure 16 shows the corresponding conditional signature.

### 3.4 DF Relative Normalization

Relative normalization rules test for the satisfiability of a DF-clause with respect to a given DF-theory. In other words, they check whether the information contained in one (positive) DF clause (possibly representing a  $\chi$ -term) is consistent with information available in another (definite) DF-clause. A substitution  $\theta$  will carry over the matching computation result.

We assume that the two clauses, namely the definite DF-clause  $\Phi$  which “controls” the normalization, and the (positive) DF-clause  $\phi$  which is normalized, do not have variables in common prior relative normalization, and  $\Phi$  is already normalized using the definite normalization rules already given.

<sup>11</sup> $\mathcal{G}$  is defined as expected on the domain of all DF-graphs over  $\mathcal{SV} \cup \mathcal{C}$ , provided that  $E$  and  $\Delta$  (in correspondence with  $E'$ ) are closed under admissible instantiation wrt  $E'$ .

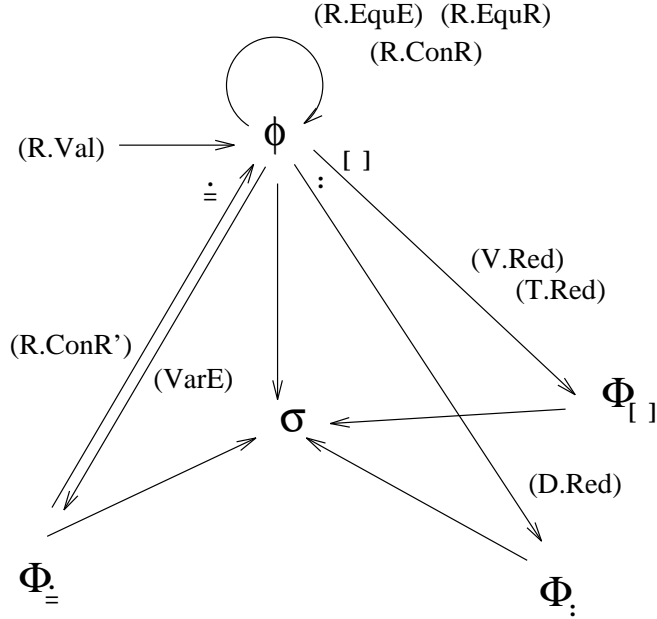


Figure 17: Relative normalization as process communication

During the relative normalization process, the “definite” condition of each reduced constraint enters the clause to be normalized. We can thus achieve goal resumption (over the well-known SLD resolution) in the sense pointed out by [7].

A specific communication process can be thought as underlying the application of the relative normalization rules. Now the rules acting as communication agents first rewrite the information on the  $\phi$  clause blackboards and then try to erase (copies of) them according to the informations they read on the  $\Phi$  clause blackboards, producing instead the unifying substitutions. See Figure 17.

The idea behind it is that after the definite normalization of  $\Phi$  was done, relative normalization tries to match directly derivation (respectively equation and method) constraints in  $\phi$  with derivation (equation, method) constraints in  $\Phi$ .

The relative normalization process starts with the group of **equational relative normalization** rules in Figure 18. Base equational rules are naturally translated in the new settings. New is the rule (R.ConR') which does uniform the use of equated constants between  $\Phi$  and  $\phi$ .

The equational relative normalization rules are applied as many times as possible without any preferential order. Then the DF-clause matching process is done by **reduction relative normalization** rules in Figure 19, tries to “view” the normalized clause  $\phi$  into the normalizing one,  $\Phi$ . Figure 20 gives the final relative normalization rule, which makes the **validation** of the result of relative normalization.

It should be added that the definite constraint in the clause that controls the reduction definite normalization is priorly renamed apart from the clause which is normalized.

If  $\phi$  was totally reduced then the relative normalization process succeeded, and the

---

(R.EquE)  $\Phi \frac{\theta \vdash t \doteq t \wedge \phi}{\theta \vdash \phi}$

(R.VarE)  $\Phi \frac{\theta \vdash X^{a_1, \dots, a_n} \doteq Y^{b_1, \dots, b_m} \wedge \phi}{\{X/Z^{a_1, \dots, a_n, b_1, \dots, b_m}, Y/Z^{a_1, \dots, a_n, b_1, \dots, b_m}\} \circ \theta \vdash \phi(X/Z, Y/Z)}$   
 $Z$  is new variable

$\Phi \frac{\theta \vdash X^{a_1, \dots, a_n} \doteq a \wedge \phi}{\{X^{a_1, \dots, a_n}/a\} \circ \theta \vdash a : a_1 \wedge \dots \wedge a : a_n \wedge \phi(X^{a_1, \dots, a_n}/a)}$

(R.EquR)  $\Phi \frac{\theta \vdash a \doteq V \wedge \phi}{\theta \vdash V \doteq a \wedge \phi}$  if  $V$  is sorted variable

(R.ConR)  $\Phi \frac{\theta \vdash a \doteq b \wedge \phi}{\theta \vdash \phi(a/b)}$  if  $a \neq b$  and  $a$  occurs in  $\phi$

(R.ConR')  $\Phi \frac{\theta \vdash \phi}{\theta \vdash \phi(a/b)}$  if  $a$  occurs in  $\phi$  and  $a \doteq b \in \Phi$

---

Figure 18: Equational Relative Normalization Rules

---

(D.Red)  $\Phi \frac{\theta \vdash \phi \wedge s : t}{\sigma \theta \vdash \sigma(\phi \wedge \delta \wedge \delta')}$   
if  $s' : t' :- \delta' \in \Phi$  and  $mgu_{<\Phi}(\langle s, t \rangle, \langle s', t' \rangle, \sigma, \delta)$

(E.Red)  $\Phi \frac{\theta \vdash \phi \wedge s \doteq t}{\sigma \theta \vdash \sigma(\phi \wedge \delta \wedge \delta')}$   
 $s' \doteq t' :- \delta' \in \Phi$ , and  $mgu_{<\Phi}(\langle s, t \rangle, \langle s', t' \rangle, \sigma, \delta)$

(V.Red)  $\Phi \frac{\theta \vdash \phi \wedge s[u_{qt}\bar{v}]t}{\sigma \theta \vdash \sigma(\phi \wedge \delta \wedge \delta')}$   
if  $qt = \rightarrow, \Rightarrow$  or  $\bar{b}$ , and there is  $s'[u'_{qt'}\bar{v}']t' :- \delta'$  in  $\Phi$  such that  $mgu_{<\Phi}(\langle s, u, \bar{v}, t \rangle, \langle s', u', \bar{v}', t' \rangle, \sigma, \delta)$

(T.Red)  $\Phi \frac{\theta \vdash \phi \wedge s[u_{qt}\bar{v}]t}{\sigma \theta \vdash \sigma(\phi \wedge \delta \wedge \delta_1 \wedge \delta_2 \wedge \delta_3 \wedge \delta')}$   
if  $qt$  is  $\Rightarrow, \Rightarrow\Rightarrow$ , and there is  $s'[u'_{qt'}\bar{v}']t' :- \delta'$  in  $\Phi$  such that  $mgu_{<\Phi}(\theta u, u', \sigma, \delta)$ , and  $\sigma s \preceq_{\Phi}^{\delta_1} \sigma s', \sigma \bar{v} \preceq_{\Phi}^{\delta_2} \sigma \bar{v}', \sigma t' \preceq_{\Phi}^{\delta_3} \sigma t$ .

---

Figure 19: Relative Normalization Rules: Reduction

---


$$(R.Val) \quad \frac{\theta \vdash \phi}{\cdot \vdash \perp}$$

if  $\phi -$  is not the empty DF-clause

---

Figure 20: Relative Normalization Rules: Validation

substitution  $\theta$ , which carried the matching result ( $\theta$  was assumed  $\varepsilon$ , the empty substitution, in the beginning of relative normalization), is now a most general unifying substitution of  $\phi$  into  $\Phi$ .

Note that reduction relative normalization rules introduce (apart from all precedent rules!) true non-determinism: applying one of them could cause the failure of the relative normalization process (i.e., the  $\perp$  clause) without leading necessarily to the conclusion  $\phi$  does not normalize wrt  $\Phi$ . Such a conclusion could be formulated only after all possible trials were done, and it is taken by the relative validation rule, which is applied (only) after all other relative normalization rules can no more be applied.

**Theorem 3.1** (*Soundness and completeness of DF-relative normalization*)

For every  $\Phi$  definite solved DF-clause and  $\phi$  positive DF-clause which have no variable in common,

$\Phi \wedge \neg\phi$  is unsatisfiable iff  $\Phi \models_{DF} \exists\phi$  iff  $\phi$  normalizes to  $\top$  wrt  $\Phi$ , and if this is the case, for every computed admissible substitution  $\theta$  there follows that  $\Phi \models_{DF} \theta\phi$ .

*Proof:* The easiest way to prove this theorem is to refer to the corresponding results in DF-logic, since

- there is an immediate correspondence between DF-logic semantic structures and DF-algebras, and
- any reduction of  $\phi$  to  $\top$  wrt  $\Phi$  corresponds to a refutation from (the DF-logic formula corresponding to)  $\Phi \wedge \phi$  in DF-logic.

In a direct manner, the claim can be proven by considering  $\mathcal{M}$ , minimal DF-model of  $\Phi \rightarrow \phi$ :

- the equality relative normalization rules put  $\phi$  into an  $\mathcal{M}$ -equivalent form, and
- application of any reduction rule is based on the fact that

$\Phi \models_{\mathcal{M}} \exists(\varphi \wedge \phi)$  iff  $\Phi \models_{\mathcal{M}} \exists\sigma(\delta \wedge \delta' \wedge \phi)$  provided that  $\varphi$  is an atomic DF-constraint, there is  $\varphi' : -\delta' \in \Phi$  and  $\sigma$  admissible substitution wrt such that  $\sigma = mgu_{<\Phi}(\varphi, \varphi', \delta)$ .<sup>12</sup> The “if” implication is immediate, while the “only if” part is due to the solved form of  $\Phi$  and the minimality of  $\mathcal{M}$ . □

**Proposition 3.4** *Deciding satisfiability of a DF-clause wrt a solved definite DF-clause is NP-hard.*

*Proof:* Let’s take the simple case of normalizing two positive DF-clauses  $\phi_1$  and  $\phi_2$ . (This covers the case of  $\chi$ -term “into” unification as defined by M.Kifer.) We have to find a mapping  $\lambda : \phi_1 \rightarrow \phi_2$ , where each  $\phi_i$  is seen as set of atomic DF-constraints, such that  $\sigma\varphi = \sigma\lambda(\varphi), \forall\varphi \in \phi_1$ . Such a mapping can be found in NP-time wrt the length of  $\phi_1$  and  $\phi_2$ . □

---

<sup>12</sup>Note that the fact that  $\varphi$  unifies with  $\varphi'$  implies that the two constraints belong to a same category: derivation, equality, or a certain subcategory of method constraints.

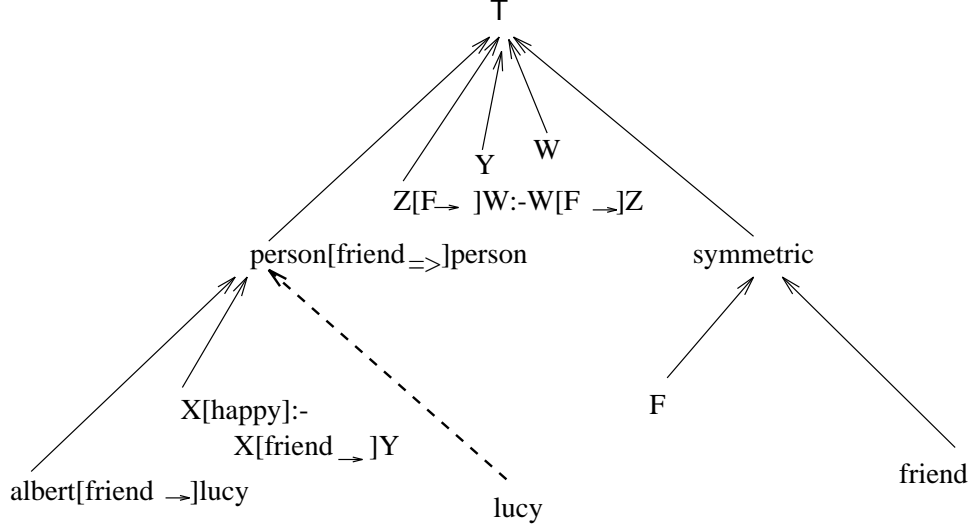


Figure 21:

**Example 3.6** Let consider again the DF program in Example 1.2,

The clauses (D2) and (D3) can be eliminated since relative normalization proves that the condition of (D2) is never satisfied and the head of (D3) is a trivial equation. (Note that the necessity for doing occur check is encountered.)

The reference signature finally obtained, extended with object/class functional specifications is shown in Figure 21. It gives an idea on the DF-graph canonical algebra associated to the program. The solutions of a query are found on such a graph via admissible instantiation, DF-algebra homomorphisms and relative definite normalization.

The goal ? – lucy[happy] is solved – by relative normalization – as it follows:

$$\begin{array}{ll}
 (G0) & lucy[happy] \\
 (G1) & lucy[friend_{\rightarrow}]Y' \\
 (G2) & W'''[friend_{\rightarrow}]lucy \\
 (G3) & \square
 \end{array}
 \quad
 \begin{array}{ll}
 (C1) & \sigma_0 = \{X/lucy\} \\
 (C3) & \sigma_1 = \{Z/lucy, F/friend\} \\
 (C6) & \sigma_2 = \{W'''/albert\}
 \end{array}$$

## Conclusion and further work

DF is a constraint logic system working on  $\chi$ -terms, data frames which offer an increased expressivity over  $\psi$ -terms and feature trees, while maintaining the main results of the CFT and OSF theories, namely clause satisfiability via rewriting rules, the canonicity of a graph algebra, and relative clause normalization as an entailment/disentailment decision method.

Specific points of the present work are first-citizen status offered to both features and sorts, finer grained feature constraints (single- and multi-valued features with arguments), run-time type control and inferential completion of (partially specified, possibly conditional) sort signature via well-type conditioning and type inheritance axioms.

We intend to implement these normalization procedures either using LOGOS – the object-oriented environment we have designed for logic programming languages implementation [11], or – quite better – in Oz, the concurrent constraint object-oriented language built on top of CFT theory. Also, we want to see how the Oz-like concurrent calculi (for instance [20], [22]) will support us towards this goal from a theoretical point of view.

As an application field we intend to implement Head-driven Phrase Structure Grammars [18] for natural language processing.

## Acknowledgements

I have benefit from many helpful discussions with professor Jean-Paul Delahaye during my five months stay at LIFL - University of Lille. My colleagues Dorel Lucanu, Mihaela Juganaru, Mirela Petrea and R. Volanski provided for interesting remarks on precedent versions of this paper. My thanks go equally to all of them. The unknown (yet) faults herein are due only to the author. He simply hopes that during the next stage of his work will be able to cover them and to improve his work style.

## Appendix: Introduction to F-logic

We give here a brief outline of a F-logic, slightly adapted from [16].

For convenience, we restricted the presentation assuming Datalog instead of Prolog ones as ‘identification’ terms for complex objects. We name DF-logic the subset of F-logic determined in this way. All the results are naturally preserved in the new settings.

### .1 Syntax

Let be  $\mathcal{C}$  a set of **constants** ranged over by  $a, b, c, \dots$ , and  $\mathcal{V}$  a set of **variables** denoted  $X, Y, Z, \dots$ . Both constants and variables will be called in the subsequent **references**, and we will range them over by  $s, t, u, v, \dots$ .

DF-terms are syntactic constructions defined by the following BNF grammar:

$$\begin{aligned}
 \textit{is-a-term} & := s : t \\
 \textit{equation} & := s = t \\
 \chi\textit{-term} & := s[\textit{method} ; \textit{method} ; \dots ; \textit{method}] \\
 \textit{method} & := u@v_1, \dots, v_n \textit{quasi-type } t \\
 \textit{quasi-type} & := \rightarrow \mid \Rightarrow \mid \twoheadrightarrow \mid \Rightarrow\Rightarrow \mid \bar{b}
 \end{aligned}$$

The reference  $s$  in the  $\chi$ -term definition is sometimes called the  $\chi$ -term **identification part**. The reference  $u$  in the *method* specification corresponds to the well known notion **feature**, and the possibly empty sequence of **arguments**  $v_1, \dots, v_n$  is commonly referred as the **context** of feature’s application. There are five kinds (we call them quasi-types) of *methods*: functional (corresponding to the *quasi-type*  $\rightarrow$ ), set-valued ( $\twoheadrightarrow$ ), function-typing ( $\Rightarrow$ ), set-typing ( $\Rightarrow\Rightarrow$ ), and boolean ( $\bar{b}$ ) methods.

All methods correspond to partially defined mappings. **Functional** methods are single-valued, while **set-valued** methods are multi-valued mappings. **Function-typing** methods and **set-typing** methods are multi-valued too, and they define the types of values taken by functional methods, respectively **set-valued** methods. **Boolean** methods act like functional methods, but they are restricted to only one value:  $\top[\ ]$ , here a denotation for the truth value **true**. Therefore, the explicit writing of this value will become superfluous.

DF-logic formulae are defined as one would expect, using logical connectives and quantifiers over is-a terms, equations and  $\chi$ -terms.

## .2 Semantics

The declarative DF-semantics makes use of intensional definitions for methods, which means that functions and predicates are associated to reference denotations in the interpretation domain (instead to syntactic symbols, as first-order logic usually does).

A DF-semantic structure over an alphabet  $C$  is a tuple

$$\mathcal{S} = \langle D, \preceq, I_C, I_{\rightarrow}, I_{\Rightarrow}, I_{\twoheadrightarrow}, I_b \rangle$$

where:

- $D$  – the domain of interpretation – is a non-empty set;
- $\preceq$  is a partial order relation on  $D$ ;
- $I_C : C \rightarrow D$  is the interpretation of constant symbols;
- $I_{\rightarrow} : D \rightarrow \prod_{n \geq 0} \text{Partial}(D^{n+1}, D)$  is the interpretation of functional methods.  $\text{Partial}(A, B)$  denotes the set of all partially defined functions from  $A$  to  $B$ .  
 $(I_{\rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$  denotes, when defined, the value taken by the functional method  $m$  when called by the object/class  $u$  with the arguments  $u_1, \dots, u_n$ .
- $I_{\Rightarrow} : D \rightarrow \prod_{n \geq 0} \text{PartialAntimonotone}(D^{n+1}, 2_{\uparrow}^{D^s})$  is the interpretation of fun-typing methods.  
 $2_{\uparrow}^D$  denotes the set made out of all upward-closed subsets of  $D$ . (A subset  $A$  of  $D$  is in  $2_{\uparrow}^D$  iff for every  $x \in A$  there follows that every  $y \geq x$  belongs to  $A$ .  
Given  $A$  and  $B$  two sets partially ordered respectively by  $\leq_A$  and  $\leq_B$ , a function  $f : A \rightarrow B$  is antimonotone iff whenever  $f(x)$  is defined then  $f(y)$  is defined for every  $y \leq_A x$ , and  $f(x) \leq_B f(y)$ . (The antimonotone property is used further for argument subtyping.)  
 $(I_{\Rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$  will say, when defined, that the value taken by the functional method  $m$  called by  $u$  in the context  $u_1, \dots, u_n$  belongs to each one of the classes in the set denoted by  $(I_{\Rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ .
- $I_{\twoheadrightarrow} : D \rightarrow \prod_{n \geq 0} \text{Partial}(D^{n+1}, 2^D)$  is the interpretation of set-valued methods.  
 $2^D$  denotes the powerset of  $d$ , i.e., the set of all subsets of  $D$ .  
 $(I_{\twoheadrightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$  denotes, when defined, the values taken by the set-valued method  $m$  when called by  $u$  in the context of  $u_1, \dots, u_n$ .
- $I_{\twoheadrightarrow} : D \rightarrow \prod_{n \geq 0} \text{PartialAntimonotone}(D^{n+1}, 2_{\uparrow}^D)$  is the interpretation of set-typing methods.  
 $(I_{\twoheadrightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$  will say, when defined, that all the values taken by the set-valued method  $m$  called by  $u$  in the context  $u_1, \dots, u_n$  belong to each one of the classes in the set denoted by  $(I_{\twoheadrightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ .

- $I_b : D^S \rightarrow \prod_{n \geq 0} 2^{D^{n+1}}$  is the interpretation of predicative methods.

For each  $n \geq 0$ , the  $I_b(m)^{(n)}$  component of  $I_b(m)$  denotes the set of all tuples  $\langle u, u_1, \dots, u_n \rangle$  belonging to  $m$ , when  $m$  interpreting  $m$  as a  $(n+1)$ -ary relation.

The well-typing conditions which link the interpretation of method values and types are stated as follows:

if  $q = (I_{\rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ , then  $q \leq r$  for each  $r \in (I_{\Rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ , and

if  $q = (I_{\twoheadrightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ , then  $q \leq r$  for each  $r \in (I_{\Rightarrow}(m))^{(n)}(u, u_1, \dots, u_n)$ .

The notion of variable assignment in a given DF-semantic structure  $\mathcal{S}$  is defined as usually. The satisfiability for DF is-a terms, equations and elementary<sup>13</sup>  $\chi$ -terms is given by the following formulae (assuming that  $\alpha$  is a variable assignment in the DF semantic structure  $\mathcal{S}$ ):

$\mathcal{S}, \alpha \models s : t$  iff  $\alpha(s) \leq \alpha(t)$  in  $D$ ;

$\mathcal{S}, \alpha \models s \doteq t$  iff  $\alpha(s) = \alpha(t)$  in  $D$ ;

$\mathcal{S}, \alpha \models s[m@u, u_1, \dots, u_n \rightarrow v]$  iff  $(I_{\rightarrow}(\alpha(m)))^{(n)}(\alpha(u), \alpha(u_1), \dots, \alpha(u_n))$  is defined and equal to  $\alpha(v)$ ;

$\mathcal{S}, \alpha \models s[m@u, u_1, \dots, u_n \Rightarrow t]$  iff  $(I_{\Rightarrow}(\alpha(m)))^{(n)}(\alpha(u), \alpha(u_1), \dots, \alpha(u_n))$  is defined and contains  $\alpha(t)$ ; similarly for  $\twoheadrightarrow$  and  $\Rightarrow$ ; and finally

$\mathcal{S}, \alpha \models s[m@u, u_1, \dots, u_n]$  iff  $(I_{\rightarrow}(\alpha(m)))^{(n)}$  contains  $\langle \alpha(u), \alpha(u_1), \dots, \alpha(u_n) \rangle$ .

DF-term unification is defined in the case of is-a terms and equations via unification of arrays of Datalog terms. Given  $s$  a flat  $\chi$ -term, it unifies “into” the  $\chi$ -term  $t$ , and  $\theta$  is one of their unifying substitution, if the set of elementary  $\chi$ -subterms of  $t\sigma$  is included into the set of elementary  $\chi$ -subterms of  $s\sigma$ .

DF inference rules, formulated for the case of flat  $\chi$ -terms are the following:

1. Resolution

$$\frac{\neg T \vee C, T' \vee C', \theta = mgu(T, T')}{\theta(C \vee C')}$$

2. Factoring

$$\frac{T \vee T' \vee C, \theta = mgu(T, T')}{\theta(T \vee C')}$$

$$\frac{\neg T \vee \neg T' \vee C, \theta = mgu(T, T')}{\theta(\neg T' \vee C)}$$

3. Paramodulation

$$\frac{L[T] \vee C, (T' \doteq T'') \vee C', \theta = mgu(T, T')}{\theta(L[T''] \vee C \vee C')}$$

4. Is-a Reflexivity

$$X : X$$

---

<sup>13</sup>A  $\chi$ -term is elementary if it has only one method. Of course, any flat  $\chi$ -term is equivalent to a conjunction of elementary  $\chi$ -terms.

5. Is-a Transitivity

$$\frac{P : Q \vee C, Q' : R' \vee C', \theta = mgu(Q, Q')}{\theta(P : R' \vee C \vee C')}$$

6. Is-a Antisymmetry

$$\frac{P : Q \vee C, Q' : P' \vee C', \theta = mgu(\langle P, Q \rangle, \langle P', Q' \rangle)}{\theta((P \doteq Q) \vee C \vee C')}$$

7. Well-Typing

$$\frac{P[M@Q_1, \dots, Q_k \rightarrow R] \vee C, P'[M'@Q'_1, \dots, Q'_k \Rightarrow R'] \vee C' \quad \theta = mgu(\langle P, M, Q_1, \dots, Q_k \rangle, \langle P', M', Q'_1, \dots, Q'_k \rangle)}{\theta(R : R' \vee C \vee C')}$$

Similarly for set valued methods: replace  $\rightarrow$  by  $\rightarrow\rightarrow$ , and  $\Rightarrow$  by  $\Rightarrow\Rightarrow$ .

8. Type Inheritance

$$\frac{P[M@Q_1, \dots, Q_k \Rightarrow T] \vee C, S' : P' \vee C', \theta = mgu(P, P')}{\theta(S'[M@Q_1, \dots, Q_k \Rightarrow T] \vee C' \vee C')}$$

Similarly for set valued methods.

9. Argument Subtyping

$$\frac{P[M@Q_1, \dots, Q_i, \dots, Q_k \Rightarrow R] \vee C, Q''_i : Q'_i \vee C', \theta = mgu(Q_i, Q'_i)}{\theta(P[M@Q_1, \dots, Q''_i, \dots, Q_k \Rightarrow T] \vee C' \vee C')}$$

Similarly for set valued methods.

10. Range Supertype

$$\frac{P[M@Q_1, \dots, Q_k \Rightarrow R] \vee C, R' : R'' \vee C', \theta = mgu(R, R')}{\theta(P[M@Q_1, \dots, Q_k \Rightarrow R''] \vee C' \vee C')}$$

Similarly for set valued methods.

11. Functionality

$$\frac{P[M@Q_1, \dots, Q_k \rightarrow R] \vee C_1, P'[FunM'@Q'_1, \dots, Q'_k \rightarrow R'] \vee C_2 \quad \theta = mgu(\langle P, M, Q_1, \dots, Q_k \rangle, \langle P', FunM', Q'_1, \dots, Q'_k \rangle)}{\theta(R \doteq R' \vee C_1 \vee C_2)}$$

12. Merging

$$\frac{P[\dots] \vee C, P'[\dots] \vee C', \theta = mgu(P, P'), R = merge(\theta(P[\dots]), \theta(P'[\dots]))}{R \vee \theta(C \vee C')}$$

13. Elimination

$$\frac{\neg T[\ ] \vee C}{C}$$

The following result links the declarative and procedural semantics of DF-logic:

**Theorem .2** (Kifer): *The above DF inference rules are sound and complete with respect to the declarative semantics (priorly) defined. I.e., for any unsatisfiable set  $S$  of clauses, there is derivation of the empty clause from  $S$  using the above given rules.*

It is in fact to this result that we owe the development of the DF constraint system. Finally, the correspondence between DF-semantic structures and DF-algebras is quite natural.

Let be the DF-semantic structure

$$\mathcal{S} = \langle D^{\mathcal{S}}, \preceq_{\mathcal{S}}, I_C^{\mathcal{S}}, I_{\rightarrow}^{\mathcal{S}}, I_{\Rightarrow}^{\mathcal{S}}, I_{\Rightarrow\Rightarrow}^{\mathcal{S}}, I_{\bar{b}}^{\mathcal{S}} \rangle .$$

We define the DF-algebra

$$\mathcal{A}(\mathcal{S}) = \langle D^{\mathcal{A}}, \prec_{\mathcal{A}}, I_C^{\mathcal{A}}, (I_{u \rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u \Rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u \Rightarrow\Rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u_{\bar{b}} \bar{v}}^{\mathcal{A}})_{u, \bar{v}} \rangle$$

by taking

$$\begin{aligned} D^{\mathcal{A}} &= D^{\mathcal{S}} \cup \{T\}, \text{ with } T \notin D^{\mathcal{S}}, \\ \prec_{\mathcal{A}} &= \preceq_{\mathcal{S}} \cup \{(u, T) \mid u \text{ is maximal element in } D^{\mathcal{S}}\}, \\ I_C^{\mathcal{A}} &= I_C^{\mathcal{S}}, \\ (I_{m \rightarrow \bar{v}}^{\mathcal{A}})^{(n)}(m) &= \{(u, w) \in D^{\mathcal{A}} \mid (I_{\rightarrow}^{\mathcal{S}}(m))^{(n)}(u, \bar{v}) \text{ is defined and equal to } w\}, \\ &\text{where } n \text{ is the length of the sequence } \bar{v}, \\ (I_{m \Rightarrow \bar{v}}^{\mathcal{A}})^{(n)}(m) &= \{(u, w) \in D^{\mathcal{A}} \mid (I_{\Rightarrow}^{\mathcal{S}}(m))^{(n)}(u, \bar{v}) \text{ is defined and contains } w\}, \\ &\text{similarly for } \Rightarrow\Rightarrow, \text{ and for } \Rightarrow\Rightarrow\Rightarrow, \text{ and finally} \\ (I_{m_{\bar{b}} \bar{v}}^{\mathcal{A}})^{(n)} &= \{u \in D^{\mathcal{A}} \mid (I_{\bar{b}}^{\mathcal{S}}(m))^{(n)} \text{ contains } (u, \bar{v})\}. \end{aligned}$$

Due to the well-typing conditions stated for DF-semantic structures,  $\mathcal{A}(\mathcal{S})$  satisfies all the axioms stated for DF-algebras but (C). Moreover, for every  $\varphi$  DF-logic formula and every  $\alpha$  valuation in  $\mathcal{S}$

$$\text{if } \mathcal{S}, \alpha \models \varphi \text{ then there is } \beta \text{ valuation in } \mathcal{A}(\mathcal{S}) \text{ such that } \mathcal{A}(\mathcal{S}), \beta \models \phi,$$

where  $\phi$  is the DF-constraint formula corresponding to DF-logic formula  $\varphi$ .

Conversely, given a DF-algebra

$$\mathcal{A} = \langle D^{\mathcal{A}}, \prec_{\mathcal{A}}, I_C^{\mathcal{A}}, (I_{u \rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u \Rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u \Rightarrow\Rightarrow \bar{v}}^{\mathcal{A}})_{u, \bar{v}}, (I_{u_{\bar{b}} \bar{v}}^{\mathcal{A}})_{u, \bar{v}} \rangle$$

we can associate it a DF-semantic structure

$$\mathcal{S}(\mathcal{A}) = \langle D^{\mathcal{S}}, \preceq_{\mathcal{S}}, I_C^{\mathcal{S}}, I_{\rightarrow}^{\mathcal{S}}, I_{\Rightarrow}^{\mathcal{S}}, I_{\Rightarrow\Rightarrow}^{\mathcal{S}}, I_{\bar{b}}^{\mathcal{S}} \rangle .$$

defining its components in the following way:

$$\begin{aligned} D^{\mathcal{S}} &= D^{\mathcal{A}}, \\ \preceq_{\mathcal{S}} &= \preceq_{\mathcal{A}} \text{ (} \preceq_{\mathcal{A}} \text{ denotes the reflexive and transitive closure of } \prec_{\mathcal{A}} \text{)}, \\ I_C^{\mathcal{S}} &= I_C^{\mathcal{A}}, \\ (I_{qt}^{\mathcal{S}}(m))^{(n)}(u, \bar{v}) &= w \text{ iff } I_{m_q t \bar{v}}^{\mathcal{A}} \text{ is true for } (u, w), \text{ for } qt \neq \bar{b}. \\ (I_{\bar{b}}^{\mathcal{S}}(m))^{(n)} &\text{ contains } (u, \bar{v}) \text{ iff } I_{u_{\bar{b}} \bar{v}}^{\mathcal{A}} \text{ is true for (i.e, contains) } u, \\ &n \text{ being the length of the sequence denoted by } \bar{u}. \end{aligned}$$

It's not difficult to verify that due to the DF axiom schemes (that  $\mathcal{A}$  verifies)  $\mathcal{S}(\mathcal{A})$  is indeed a DF-semantic structure, and it satisfies the well-typing conditions. Moreover, for every  $\phi$  DF-constraint formula and every valuation  $\alpha$  in  $\mathcal{A}$

$$\text{if } \mathcal{A}, \alpha \models \phi \text{ then there is } \beta \text{ valuation in } \mathcal{A}(\mathcal{S}) \text{ such that } \mathcal{A}(\mathcal{S}), \beta \models \varphi,$$

where  $\varphi$  is the DF logic formula corresponding to DF-constraint formula  $\phi$ .

## References

- [1] H. Abramson, Definite feature grammars for natural language processing. In *Logic Programming and Natural Language Understanding*, vol III, 1990.
- [2] H. Aït-Kaci, R. Nasr, LOGIN: A logic programming language with built-in inheritance, *Journal of Logic Programming*, 1986:3:185-215.
- [3] H. Aït-Kaci, A. Podelski, Towards a meaning of LIFE, *Journal of Logic Programming*, 1993:16:195-234.
- [4] H. Aït-Kaci, A. Podelski, Order-sorted feature theory unification. In Dale Miller (ed.), *Logic Programming, Proceedings of The 1993 International Symposium*, pp 506-524, The MIT Press, 1993.
- [5] H. Aït-Kaci, A. Podelski, Entailment and disentanglement of order-sorted feature constraints. In Andrei Voronkov (ed.), *Proceedings of the 4th International Conference on Logic Programming and Automate Reasoning*, Springer-Verlag, 1993.
- [6] H. Aït-Kaci, A. Podelski, G. Smolka, A feature constraint system for logic programming with entailment, *Theoretical Computer Science* 122, pp 263-283, 1994.
- [7] H. Aït-Kaci, A. Podelski, Functions as passive constraints in LIFE, *ACM Transactions on Programming Languages and Systems*, Vol. 16, pp 1279-1318, 1994.
- [8] R. Backofen, G. Smolka, A complete and recursive theory, DFKI Research Report, 1992.
- [9] W. Chen, M. Kifer, D. S. Warren, HiLog: A foundation for higher-order logic programming, *Journal of Logic Programming*, 1993:15:187-230
- [10] L. V. Ciortuz, Object-oriented inferences in a logical framework for feature-based grammars. In *Grammatical Inferences and Applications*, LNCS 862, R. Carasco, J. Oncina (eds.), Springer-Verlag, pp45-56, 1994.
- [11] L. V. Ciortuz, M. Petrea, LOGOS: An object-oriented framework for logic programming language implementation. In *Proceedings of the 6th Workshop for Logic Programming Environments*, Santa Margherita Ligure, Italy, pp 39-50, 1994.
- [12] S. Constantini, G. Lazarone, A metalogic programming approach: language, semantics and applications, Technical Report, Universita di Milano, 1990.
- [13] G. Gazdar, E. Klein, G. K. Pullum, I. Sag, *Generalized Phrase Structure Grammar*, Harvard University Press, Cambridge, Mass., 1985.
- [14] R. M. Kaplan, J. Bresnan, Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (ed.) *The Mental Representation of Grammatical Relations*, pp 173-381, MIT Press, 1982.
- [15] M. Kay, Parsing in functional-unification grammars. In D. D. Dowty, L. Karttunen (eds.) *Natural Language Parsing*, Cambridge U. P., Cambridge, England, 1985.
- [16] M. Kifer, G. Lausen, J. Wu, A logical foundation of object-oriented and frame-based languages, Research Report, SUNY at Stony Brook, 1990.

- [17] J. Lloyd, *Foundations of Logic Programming*, 2nd ed., Springer-Verlag, 1987.
- [18] C. Pollard, I. Sag, *An information-based syntax and semantics*, vol. I, CSLI, 1987
- [19] S. M. Shieber, H. Uszkoreit, F. C. Pereira, J. Robinson, M. Tyson, The formalism and implementation of PATR-II. In J. Bresnan (ed.) *Research on Interactive Acquisition and Use of Knowledge*, SRI International, Menlo Park, Calif., 1983.
- [20] G. Smolka, M. Henz, J. Würtz, Object-oriented concurrent constraint programming in Oz, DFKI RR-93-16.
- [21] G. Smolka, Feature-constraint logics for unification grammars, *Journal of Logic Programming* 1992:12:51-87.
- [22] G. Smolka, A calculus for higher-order concurrent constraint programming with deep guards, DFKI RR-94-03.
- [23] G. Smolka, R. Treinen, Records for logic programming, *Journal of Logic Programming* 1994:18:229-258.
- [24] R. Treinen, Feature Constraints with first-class features. In A. Borzyszkowski and S. Sokolowski, eds., *Mathematical Foundations of Computer Science*, LNCS 711, pp 734-743, Springer-Verlag, 1993.
- [25] P. van Hentenryck, *Constraint Satisfaction in Logic Programming*, *Programming Logic Series*, The MIT Press, Cambridge, MA, 1989.