

## Object-Oriented Inferences in a Logical Framework for Feature Grammars

Liviu-Virgil Ciortuz  
Department of Computer Science  
"A. I. Cuza" University of Iasi  
6600 Iasi, Romania

July, 1994

**Abstract:** This paper deals on defining object-oriented inferences by desining a new unification procedure called  $\chi$ -unification (which leads to a sound and complete resolution) in DF-logic, a significant subset of F-logic [KLW,1990], an useful frame-based logical framework that enables the declarative implementation of feature grammars like LFGs, GPSGs, HPSGs, etc.

### 0. Introduction:

DF-logic is the sub-set of F-logic (defined by M. Kifer and his colleagues from SUNY at Stony Brook in [KLW,1990]) obtained by imposing the use Datalog terms (instead of Prolog terms) as 'identification terms' in the atom defintions.

We provide DF-logic with an effective procedural semantics by replacing the unification procedure originally designed for F-atoms such that most things previously intended to be done at the resolution level sould be placed at the unification level (thus drastically reducing the number of resolution rules.

This idea was previously used in the framework of the first-order predicate logic by H.Ait-Kaci in [AKN,1986] (only) for the type-inheritance principle. He defined LOGIN as a logic programming language which extends Prolog by incorporating inheritance at a "built-in" level, namely in the unification procedure. This was called  $\psi$ -unification and it extends the well-known unification in the first-order predicate calculus.

The  $\chi$ -unification procedure here proposed involves not only (type) inheritance, but also other object-oriented principles like argument sub-typing, range super-typing, functionality and the (so called) well-typing conditions. Moreover, while  $\psi$ -unification in LOGIN leads to uncomplete resolution, our approach preserves both soundness and completeness of implied inference in DF-logic.

The work here reported could be naturally redone for F-logic (see [Cio,1994]). So, the F-logic restriction to DF-logic is intended only for gaining more effectiveness in implementing a top-down inference procedure in these settings, and is motivated by reasons of immediate practical sufficiency.

## 1. Syntax:

*DF-atoms* in DF-logic define frames of methods characterizing classes (and objects as well). Their syntactic form is

$$\text{id}[\text{method}_1; \text{method}_2; \dots; \text{method}_n]$$

where *id* is the "identification" term for that atom, and each  $\text{method}_i$ , for  $i = 1, 2, \dots, n$  has one of the following syntactic forms:

$\text{name}@a_1, a_2, \dots, a_k \rightarrow v$	- functional (or: single valued) method;
$\text{name}@a_1, a_2, \dots, a_k \Rightarrow t_1, t_2, \dots, t_m$	- function-typing method;
$\text{name}@a_1, a_2, \dots, a_k \rightarrow v_1, v_2, \dots, v_s$	- set-valuated method;
$\text{name}@a_1, a_2, \dots, a_k \Rightarrow t_1, t_2, \dots, t_r$	- set-typing method;
$\text{name}@a_1, a_2, \dots, a_k$	- predicative method.

*Is-a atoms*, having the syntactic form  $t_1.t_2$  are seen as *class constraints*: the object (represented by)  $t_1$  must belong to the class  $t_2$ , or (equally valid): the class  $t_1$  is a sub-class in (or, better: is derived\* from) the class  $t_2$ . And these are *explicit* class constraints. Other, *implicit* class constraints will be inferred using object-oriented principles like well-typing conditions and functionality.

The antisymmetry of is-a relations leads to the necessity of introducing *equations* having the syntactic form  $t_1=t_2$ .

*Well-formed formulas* in DF-logic are defined quite naturally starting from the atom definitions given above, following the classic way (see [Llo,1987]). The same will be true for other notions like literals, clauses, Horn clauses, definite programs, etc. For exemplification, we suggest the reader to see the section 3 ("A Simple Example").

## 2. Semantics:

The declarative semantics of DF-logic is given in the Appendix at the end of this paper. Herbrand semantics of DF-logic is naturally definable and a Herbrand-like theorem is valid in these settings.

Now we will concentrate on presenting the object-oriented inference rules in the DF-logic procedural semantics.

*Definition 1:*

A *conditional term hierarchy* in a DF-logic is a tuple  $H = \langle T, \rho, \Delta, \Gamma \rangle$ , where  $T$  is a set of terms,  $\rho$  is a relation on  $T$ , and  $\Delta$  and  $\Gamma$  are application defined on  $\rho$  and  $\{(A, m) \mid A \in F, m \in \text{methods}(A)\}$  respectively, and taking as value formulas. (So, for each pair  $(a, b)$  in  $\rho$ ,  $\Delta(a, b)$  will denote the condition under which a "is a"  $b$ , i.e., the object  $a$  belongs to the class  $b$ , or:  $a$  is subclass of the class  $b$ .)

If  $(A, B) \in \rho$ , then  $A$  is called a *parent* of  $B$ .  $B$  is a *proper parent* of  $A$  if it is parent of  $A$  but they do not belong to a same cycle in  $H$ . A term  $B \in T$  is an *ancestor* of  $A$  in  $H$  if  $(A, B) \in \text{trans-closure}(\rho)$  in  $H$ , where  $\text{trans-closure}(\rho)$  denotes the reflexive and transitive closure of  $\rho$  in  $H$ . A term  $B \in T$  is a *meta-term* of  $A$  in  $H$  if there is  $A' \in T$ , which is both an ancestor of  $A$  in  $H$  and a parent of  $B$ ,  $A$  is an instance of  $B$ , and for any proper parent of  $B$  is an ancestor of  $A$ .

*Remark:* Because every formula in DF-logic could be equivalently written in the disjunctive normal form, the value taken by  $\Delta$  and  $\Gamma$  could be seen as a set of atom conjunctions, and so will be in the subsequent. The empty set should be interpreted as  $T$ , the formula always true. If

---

\* This is why we will call class constraints as *derivation constraints*.

$\Delta$  and  $\Gamma$  are always  $T$  (true), then  $H$  in the above definition is in fact a un-conditional hierarchy.

*Definition 2:*

Two terms  $t_1$  and  $t_2$  in a DF-logic language unify wrt a conditional given term hierarchy  $H = \langle T, \rho, \Delta, \Gamma \rangle$ , under the condition  $\delta$ ,  $\sigma$  being one of their unifying substitutions, (and we will denote this fact by  $\text{unify}(t_1, t_2, \sigma, \delta)$ ), if they unify as Datalog terms modulo the conditional equality relation induced on  $T$  when considering  $\text{Trans-closure}(\rho)$  as antisymmetric relation,  $\delta$  being the underlying equality condition.<sup>1</sup>

*Note:* If  $H$  does not contain any cycle, then term unification wrt  $H$  is just Datalog term unification, i.e. two constant symbols unify iff they are the same, and any variable unifies with any term, either constant or variable.

If  $H = \langle T, \rho, \Delta \rangle$  is a conditional term hierarchy, and  $a$  and  $b$  are terms, then  $(a, b) \in \text{trans-closure}(\rho)$  in  $H$ , under the condition  $\delta$ , if there are  $a_1, a_2, \dots, a_k$  in  $T$ , with  $a = a_1$ , and  $a_k = b$ , such that  $(a_i, a_{i+1}) \in \rho$ , for  $i = 1, \dots, k-1$ , and  $\delta = \delta_1 \wedge \dots \wedge \delta_{k-1}$ , where  $\delta_i \in \Delta(a_i, a_{i+1})$ , for  $i = 1, \dots, k-1$ .

*Definition 3:*

Two elementary methods

$$\begin{array}{ll} m_1: & \text{name}_1 @ a_{11}, a_{12}, \dots, a_{1n_1} [q\_t_1 [value_1]] \\ m_2: & \text{name}_1 @ a_{21}, a_{22}, \dots, a_{2n_2} [q\_t_2 [value_2]] \end{array}$$

unify wrt the conditional term hierarchy  $H = \langle T, \rho, \Delta \rangle$ , and  $\sigma$  is one of their unifying substitutions, under the condition  $\delta$  (we will denote this fact by  $\chi\text{-unify}(m_1, m_2, \sigma, \delta)$ ), if

- i.  $q\_t_1$  and  $q\_t_2$  are the same (i.e., the two methods are of the same quasi-type), and  $n_1 = n_2$ ;
- ii. if  $m_1$  and  $m_2$  are (they both) functional, set-valuated or predicative methods, then the term arrays

$$\begin{array}{l} \langle \text{name}_1, a_{11}, a_{12}, \dots, a_{1n_1}, [value_1] \rangle \\ \langle \text{name}_2, a_{21}, a_{22}, \dots, a_{2n_2}, [value_2] \rangle \end{array}$$

unify,  $\sigma$  being one of their unifying substitution, and

$\delta$  is the conjunction of the corresponding unifying conditions (on equalities);

- iii. if  $m_1$  and  $m_2$  are (they both) function-typing or set-typing methods, then<sup>2</sup>

$\text{unify}(\text{name}_1, \text{name}_2, \sigma, \delta_0)$ ,

$(a_{1i}\sigma, a_{2i}\sigma)$ , for  $i = 1, \dots, n_1$ , and  $(value_1\sigma, value_2\sigma)$  are instances of tuples in  $\text{trans-closure}(\rho)$  in  $H$ , under the  $\delta_i$ , respectively  $\delta_{n_1+1}$  conditions, and

$$\delta = (\delta_0 \wedge \delta_1 \wedge \dots \wedge \delta_{n_1} \wedge \delta_{n_1+1})\sigma.$$

*Remark:* In the case iii. of the above definition, the unifying order  $m_1, m_2$  is important; it may not be reversible. This is why it is said that  $m_1$  unifies "into"  $m_2$  wrt  $H$ .

*Note:* Method unification wrt  $H$  could be defined by adding  $a_{1i}\sigma = a_{2i}\sigma$ , respectively  $a_{1i}\sigma : a_{2i}\sigma$  to the underlying condition. (So, the "early" binding of variables determined by Definition 3 is replaced by "latter" binding, which is more efficient.) Also, if  $H$  does not contains any cycle,  $a_{1i}\sigma = a_{2i}\sigma$  will be replaced by a corresponding unit substitution in the mgu construction, if one of  $a_{1i}\sigma$  and  $a_{2i}\sigma$  is variable. In these new settings the results which follows in the present paper do maintain their validity.

<sup>1</sup> Here is the point where we substitute the Paramodulation resolution rule in [KLW,1990].

<sup>2</sup> Here is the point where we substitute the Argument Sub-Typing and Range Super-Typing resolution rules in [KLW,1990].

*Definition 4:*

Let  $m_1$  and  $m_2$  be methods, and  $H$  a conditional term hierarchy. The substitution  $\sigma$  is one of the  $m_1$  and  $m_2$  most general unifiers wrt  $H$  if

- i.  $\sigma$  is one of the  $m_1$  and  $m_2$  unifiers wrt  $H$ , and
- ii. for each unifier  $\theta$  of  $m_1$  into  $m_2$  wrt  $H$ , with  $\theta \leq \sigma$ , it follows that  $\sigma \leq \theta$ .

*Definition 5:*

Let  $m_1$  and  $m_2$  be methods, and  $H$  a conditional term hierarchy. A set  $\Omega$  of most general unifiers of  $m_1$  into  $m_2$  wrt  $H$  is said to be complete if for any  $\chi$ -unifier  $\theta$  of  $m_1$  into  $m_2$  wrt  $H$  there is  $\sigma \in \Omega$ , such that  $\sigma \leq \theta$ .

### Method Conditional $\chi$ -Unification Algorithm

Input:  $m_1$  and  $m_2$ , two elementary methods

$m_1$ :        name<sub>1</sub> @ a<sub>11</sub>, a<sub>12</sub>, ..., a<sub>1n<sub>1</sub></sub> [q\_t<sub>1</sub> [value<sub>1</sub>]]  
 $m_2$ :        name<sub>2</sub> @ a<sub>21</sub>, a<sub>22</sub>, ..., a<sub>2n<sub>2</sub></sub> [q\_t<sub>2</sub> [value<sub>2</sub>]]

and  $H = \langle T, \rho, \Delta \rangle$  a conditional term hierarchy.

Output:  $\Omega$ , a complete set of most general unifiers of  $m_1$  into  $m_2$  wrt  $H$ , together with the underlying conditions.

Procedure:

```

if q_t1 = q_t2, n1 = n2 and mgu(name1, name2,  $\theta$ ,  $\delta_0$ )
  if  $m_1$  and  $m_2$  are non-typing methods
     $\Omega = \{ (\sigma, \delta) \mid \text{mgu}(\langle a_{11}, a_{12}, \dots, a_{1n_1}, \text{value}_1 \rangle, \langle a_{21}, a_{22}, \dots, a_{2n_2}, \text{value}_2 \rangle, \sigma, \delta) \}$ ;
  else ( $m_1$  and  $m_2$  are typing methods)
    {
       $\Omega = \emptyset$ ;
      for each application
         $\lambda: \{ a_{11}, a_{12}, \dots, a_{1n_1}, a_{1n_1+1} = \text{value}_1 \} \times \{ a_{21}, a_{22}, \dots, a_{2n_2}, a_{2n_2+1} = \text{value}_2 \} \rightarrow T \times T$ 
        such that  $a$  is an instance of  $\lambda(a)$  for every  $a$  in the domain of  $\lambda$ 
        {
           $\sigma_\lambda = \theta$ , and  $\delta_\lambda = \delta_0$ ;
          for  $i = 1, \dots, n_1 + 1$ 
            if  $(\lambda(a_{1i}), \lambda(a_{2i}))$  belongs to trans-closure( $\rho$ )
               $\sigma_\lambda = \sigma_\lambda \sigma_i$ , with  $\text{mgu}(\langle a_{1i} \sigma_\lambda, a_{2i} \sigma_\lambda \rangle, \langle \lambda(a_{1i}) \sigma_\lambda, \lambda(a_{2i}) \sigma_\lambda \rangle, \sigma_i, \delta_i)$ , and
               $\delta_\lambda = \delta_\lambda \wedge \delta_i$ ;
            else
              jump out the inner loop to select another  $\lambda$ ;
           $\Omega = \Omega \cup \{ \sigma_\lambda \}$ ;
        }
      }
    else  $\Omega = \emptyset$ ;
  return  $\Omega$ .

```

### Theorem 1:

Let  $m_1$  and  $m_2$  be methods, and  $H = \langle T, \rho, \Delta \rangle$  a conditional term hierarchy. The above algorithm returns a complete set of most general unifiers of  $m_1$  into  $m_2$  wrt  $H$  (together with the corresponding underlying conditions).

*Definition 6:*

A conditional DF-atom hierarchy (or, simply: a conditional DF-hierarchy) in a DF-logic language  $L$  is a tuple  $H = \langle F, \rho, \Delta, \Gamma \rangle$ , where  $F$  is a set of DF-atoms in  $L$ ,  $\rho$  is a relation on  $\text{term}(F)$ , and  $\Delta$  and  $\Gamma$  are applications defined on  $\rho$  respectively  $\{(A, m) \mid A \in F, m \in \text{methods}(A)\}$ ,

and taking as value formulas in the disjunctive normal form. ( $\Gamma(A,m)$  will denote the condition under which the method  $m$  belonging to the atom  $A$  could be applied.)

*Notes:*

1. The notions of parent, descendent, ancestor and meta-atom in a DF-hierarchy are immediately definable starting from their counterparts for conditional term hierarchies.

2. A term hierarchy could be easily extended to a Df-atom hierarchy by considering  $t[]$  the ("empty") DF-atom associated to each term  $t$  in the hierarchy). Also, some notions introduced above, like method  $\chi$ -unification extend naturally to DF-hierarchies.

*Definition 7:*

Let  $A$  and  $B$  be DF-atoms, and  $H$  a conditional DF-hierarchy.  $A$  unifies into  $B$  wrt  $H$ , and  $\sigma$  is one of their unifying substitution, under the condition  $\delta$ , (simply said:  $A$   $\chi$ -unifies into  $B$  under  $\delta$ ), and this fact will be denoted by  $\chi$ -unify ( $A, B, \sigma, \delta$ ), if

- i. unify ( $\text{id}(A), \text{id}(B), \sigma, \delta_0$ );
- ii. for each method  $m$  in  $A$ 
  - there is an atom  $B_m$  in  $H$  such that
    - if  $m$  is not-typing method then  $B_m$  is meta-atom of  $B$  (under the condition  $\delta_m$ ) and
    - if  $m$  is typing method then  $B_m$  is ancestor<sup>3</sup> of  $B$  (under the condition  $\delta_m$ ),
  - there is a method  $m'$  in  $B_m$  in  $H$  such that
  - unify ( $\text{id}(B), \text{id}(B_m), \sigma, \delta_m'$ ), and
  - $\chi$ -unify ( $m, m', \sigma, \delta_m''$ );
- iii.  $\delta = (\delta_0 \wedge \bigwedge_{m \in \text{methods}(A)} (\delta_m \wedge \delta_m' \wedge \delta_m'')) \sigma$ .

*Note:* The definition of most general unifier and complete set of most general unifiers for two DF-atoms (wrt a DF-hierarchy  $H$ ) are natural versions of the corresponding definitions given in the case of methods.

The following algorithm gives a constructive way to obtain a complete set of  $\chi$ -mgus for two DF-atoms wrt a given DF-hierarchy. This algorithm will use the following notations:

- i. for a DF-atom  $A$ ,  $\text{methods}(A)$  denotes the set of all methods in  $A$ ;
- ii. for  $A$  in a DF-hierarchy  $H = \langle T, \rho, \Delta, \Gamma \rangle$ ,  $\text{meta-atoms}(A)$  and  $\text{ancestors}(A)$  will denote the set of all meta-atoms of  $A$ , respectively the set of all ancestors of  $A$  in  $H$ .

### **DF-Atom Conditional $\chi$ -Unification Algorithm**

Input:  $A$  and  $B$ , two DF-atoms, and

$H = \langle F, \rho, \Delta, \Gamma \rangle$  a conditional DF-hierarchy.

Output:  $\Omega$ , a complete set of most general unifiers of  $A$  into  $B$  wrt  $H$ , together with the unification underlying conditions.

Procedure:

```

 $\Omega = \emptyset$ ;
if ( $\text{mgu}(\text{id}(A), \text{id}(B), \theta, \delta_0)$ )
  if  $A$  has no methods,
    return  $\Omega = \{(\theta, \delta_0)\}$ .
  else

```

---

<sup>3</sup> Here is the point where we substitute the Type Inheritance resolution rule in [KLW,1990].

for each  $\lambda: \text{methods}(A\theta) \rightarrow \bigcup_{B_m \in \text{meta-atom}(B\theta, H)} \text{methods}(B_m) \cup \bigcup_{B_m \in \text{ancestor}(B\theta)} \text{methods}(B_m)$   
such that  $\lambda(m) \in \text{methods}(B\theta) \cup \bigcup_{B_m \in \text{meta-atom}(B\theta, H)} \text{methods}(B_m)$  for each not-typing method  $m \in \text{methods}(A\theta)$

{  
 $\sigma_\lambda = \varepsilon$  (the empty substitution), and  $\delta_\lambda = \delta_0$ ;  
for each method  $m$  in  $A\theta$   
if  $(\gamma\text{-mgu}(m\sigma_\lambda, \lambda(m)\sigma_\lambda, \sigma_m, \delta_m))$   
 $\sigma_\lambda = \sigma_\lambda\sigma_m$ , and  $\delta_\lambda = \delta_\lambda \wedge \gamma_m \wedge \delta_m \wedge \Gamma(B_m, \lambda(m))$ , with  $\gamma_m$  the condition under which  $B_m$  involved in the range of  $\lambda$  is a meta-ancestor of  $B$  in  $H$ ;  
else  
jump out the inner loop to select another  $\lambda$ ;  
 $\Omega = \Omega \cup \{(\theta\sigma_\lambda, \delta_\lambda\theta\sigma_\lambda)\}$ ;  
}  
return  $\Omega$ .

**Theorem 2:**

Let  $A$  and  $B$  be DF-atoms, and  $H = \langle F, \rho, \Delta, \Gamma \rangle$  a conditional DF-hierarchy.

The above algorithm returns a complete set of most general unifiers of  $A$  into  $B$  wrt  $H$  (together with the corresponding underlying conditions).

Now, speaking about DF-logic program  $P$  (set of Horn clauses), one could easily associate it in a natural way a unique DF-hierarchy:

*Definition 8:*

Let  $P$  be a set of Horn clauses apart standardized in DF-logic language. *The canonical DF-hierarchy* associated to  $P$  is the conditional DF-hierarchy  $H(P) = \langle F, \rho, \Delta, \Gamma \rangle$  defined by starting from (the "base" hierarchy):

$F = \{\tau\} \cup \{A \mid A \text{ is DF-atom occurring in the head of a clause } C \text{ in } P\}$ ,

where  $\tau$  is a special symbol, the "top" element in the hierarchy,

$\rho$  is the minimal relation which satisfy the following two conditions:

- i.  $\rho \supseteq \{(a,b) \mid a:b, \text{ or } a=b, \text{ or } b=a \text{ appears in the head of a clause } C \text{ in } P\}$ ;
- ii.  $\rho \supseteq \{(X,b) \mid \text{there is a clause } C \text{ in } P \text{ such that the variable } X \text{ is } \text{id}(\text{head}(C)) \text{ and } X:b, \text{ with } b \text{ constant symbol, appears in body}(C)\}$ ,  $C$  been written as  $\text{head}(C):\text{-body}(C)$ ,

$\Delta(a,b) = \{\text{body}(C) \mid a:b, \text{ or } a=b, \text{ or } b=a \text{ is head of a clause } C \text{ in } P\}$ , and

$\Gamma(A,m) = \{\text{body}(C) \setminus \{\text{id}(A):b \text{ in body}(C) \mid b \text{ is constant symbol}\}$ , for each method  $m$  in  $A \in F$ , if  $m$  comes into  $A$  (by merging) from the head of the clause  $C$ .

and then closing it through well-typing conditions and the functionality rules (paramodularity being enclosed into unification), i.e.,

- iii.  $\rho$  is extended with  $(v,t)\sigma$  if<sup>4 5</sup>

there are two DF-atoms in  $F$

$o[ \dots ; m@a_1, a_2, \dots a_k \rightarrow v ; \dots ]$  and

$o'[ \dots ; m'@a'_1, a'_2, \dots a'_k \Rightarrow t ; \dots ]$ ,

or

$o[ \dots ; m@a_1, a_2, \dots a_k \rightarrow\!\!\rightarrow v ; \dots ]$  and

$o'[ \dots ; m'@a'_1, a'_2, \dots a'_k \Rightarrow\!\!\rightarrow t ; \dots ]$ , such that

<sup>4</sup> If there is no atom in  $F$  having  $v$ , respectively  $t$ , as its identification part, then  $v\sigma[ \ ]$ , respectively  $t\sigma[ \ ]$  are priorly added to  $F$ . The same remark affects any completion of  $\rho$  in the present definition.

<sup>5</sup> Here is the point where we substitute the Well-Typing inference rule in [KLW, 1990].

$\text{mgu}(\langle o, o' \rangle, \langle \alpha_0, \beta_0 \rangle, \sigma_0)$ , where  $(\alpha_0, \beta_0) \in \text{trans-closure}(\rho)$  under the condition  $\delta_0$ ,  
 $\text{mgu}(\langle a_i, a_i' \rangle, \langle \alpha_i, \beta_i \rangle, \sigma_i)$ , where  $(\alpha_i, \beta_i) \in \text{trans-closure}(\rho)$  under the condition  $\delta_i$ , for  $i = 1, \dots, k$ , and  
 $\sigma = \sigma_0 \sigma_1 \dots \sigma_k$ ; and, in this case  $\Delta$  is extended with  
 $\Delta(v\sigma, t\sigma) \supseteq \{\delta\sigma\}$ , with  $\delta = \delta_0 \wedge \delta_1 \wedge \dots \wedge \delta_k$ , and  
iv.  $\rho \supseteq \{(v, w), (w, v)\}$  if<sup>6</sup>  
there are two DF-atoms in  $F$   
 $o[\dots; m@a_1, a_2, \dots, a_k \rightarrow v; \dots]$  and  
 $o'[\dots; m'@a_1', a_2', \dots, a_k' \rightarrow v'; \dots]$ , such that  
 $\text{mgu}(\langle o, o' \rangle, \langle \alpha_0, \beta_0 \rangle, \sigma_0)$ , where  $\alpha_0$  and  $\beta_0$  belong to a same cycle in  $H(P)$ , or one is meta-atom of the other, under the condition  $\delta_0$ ,  
 $\chi\text{-mgu}(a_i \sigma_{i-1}, a_i' \sigma_{i-1}', \sigma_i, \delta_i)$ , for  $i = 1, \dots, k$ , and  
 $\sigma = \sigma_0 \sigma_1 \dots \sigma_k$ ; and, in this case  $\Delta$  is extended with  
 $\Delta(v\sigma, v'\sigma) \supseteq \{\delta\sigma\}$ , with  $\delta = \delta_0 \wedge \delta_1 \wedge \dots \wedge \delta_k$ .

Finally,  $\rho$  will be extended such that any maximal acyclic chain  $\gamma$  in  $\rho$  ends up in  $T$ .

Optionally, the atoms in  $F$  which have the same identification part (up to a renaming substitution) and the same proper parents could be merged.

Furthermore, we define the  $\chi$ -inference rules, which are intended to be applied (using  $\chi$ -unification) wrt a given DF-hierarchy. In the case of a DF-logic program  $P$ , we will see, it should be the canonical DF-hierarchy  $H(P)$  just defined above, in order to provide a sound and complete resolution in DF-logic.

1.  $\chi$ -Resolution:

If  $\neg A \vee C$  and  $B \vee C'$  are two clauses standardized apart in a DF-logic language, and  $\sigma$  is a  $\chi$ -mgu of  $A$  into  $B$  under the condition  $\delta$ , then derive  $\neg \delta \vee (C \vee C')\sigma$ .

2.  $\chi$ -Factorization:

If  $\neg A \vee \neg B \vee C$  is a clause, and  $\sigma$  is a  $\chi$ -mgu of  $A$  into  $B$  under the condition  $\delta$ , then derive  $\neg \delta \vee (\neg B \vee C)\sigma$ . If  $A \vee B \vee C$  is a clause, and  $\sigma$  is a  $\chi$ -mgu of  $A$  into  $B$  under the condition  $\delta$ , then derive  $\neg \delta \vee (A \vee C)\sigma$ .

3. Merging:

If  $A \vee C$  and  $B \vee C'$  are two clauses standardized apart,  $A$  and  $B$  are DF-atoms, and  $\sigma$  is a mgu of the identification parts of  $A$  and  $B$  under the condition  $\delta$ , then derive  $\neg \delta \vee R(A\sigma, B\sigma) \vee (C \vee C')\sigma$ , were  $R$  is the canonical union of  $A\sigma$  and  $B\sigma$ .

4. Elimination:

From  $\neg A[] \vee C$ , derive  $C$ .

Derive  $\neg \delta \vee \neg \delta' \vee (X:Y)\sigma$ , if there are  $X'$  and  $Y'$  in  $H$  such that  $\text{unify}(\langle X, Y \rangle, \langle X', Y' \rangle, \sigma, \delta)$ , and  $(X', Y') \in \text{trans-closure}(\rho)$  in  $H$  under the condition  $\delta'$ .

Derive  $\neg \delta \vee \neg \delta' \vee (X=Y)\sigma$ , if there are  $X'$  and  $Y'$  in  $H$  such that  $\text{unify}(\langle X, Y \rangle, \langle X', Y' \rangle, \sigma, \delta)$ , and  $X'$  and  $Y'$  belong to a same cycle in  $H$  under the condition  $\delta'$ .

The following two results give the soundness and completeness of  $\chi$ -deduction rules on Horn clauses:

**Theorem 3 (Soundness):**

Let  $S$  be a set of Horn clauses in a DF-logic language, and  $C_1, \dots, C_n$  a finite sequence of clauses such that  $C_n = C$  and, for  $1 \leq k \leq n$ ,  $C_k \in S$  or  $C_k$  is derived from  $C_i$  and (possibly)  $C_j$ , with  $i, j < k$ , by one of the  $\chi$ -deduction rules, wrt the canonical DF-hierarchy associated to  $S$ .  $C$  is a logical consequence of  $S$ .

<sup>6</sup> Here is the point where we substitute the Functionality inference rule in [KLW,1990].

#### **Theorem 4 (Completeness):**

Let  $S$  be an unsatisfiable set of clauses in a DF-logic language. There is a refutation (i.e. a derivation of the empty clause) from  $S$  using the  $\chi$ -deduction rules working wrt the canonical DF-hierarchy associated to  $S$ .

The above theorems are proven in a natural manner by reference to the corresponding results in [KLW,1990].

Finally, we give a top-down inference procedure in DF-logic:

#### **A Top-Down Inference Algorithm in DF-Logic**

Input:  $P$ , a definite program written in a DF-logic language,

$G$ , a goal  $\leftarrow A_1, A_2, \dots, A_k$ ,

$R$ , a selection rule,  $R(G) = A_i$ .

Output:  $\sigma$ , an answer substitution, if there is a substitution  $\theta$  such that  $P \in I_G \theta$ , and 'Fail', otherwise.

Procedure (identified in the following as 'top-down( $P, G, \sigma$ )'):

if  $G = \epsilon$  (the null list)

return  $\sigma = \epsilon$  (the empty substitution);

compute  $H(P) = \langle F, \rho, \Delta, \Gamma \rangle$ , the canonical hierarchy associated to  $P$ ;

eliminate the empty DF-atoms in  $G$ ;

factorize the DF-atoms in  $G$  having the same identification part;

rename the variables in  $G$  into new variables, distinct from those in  $P$ ;

$A_i = R(G)$ ;

if ( $A_i$  is DF-atom)

$B = \{B \in F \mid \text{mgu}(A_i, B, \theta, \delta)$  and, under the condition  $\gamma$ ,  $(\text{id}(B\theta), b) \in \text{trans-closure}(\rho)$ , for any ground term  $b$  such that  $\text{id}(A_i):b$  is is-a atom in  $G\}$ , and

$G' = \emptyset\delta' \cup G$ ;

else

if ( $A_i$  is is-a atom)

$B = \{B_1: B_2 \mid B_1, B_2 \in F \text{ and } \text{unify}(A_i, B_1: B_2, \theta, \delta)\}$ , and

$G' = G \setminus \{A_i\}$ ;

else ( $A_i$  is equality atom)

$B = \{B_1=B_2 \mid B_1 \text{ and } B_2 \text{ belong to a same cycle in } F, \text{ and } \text{unify}(A_i, B_1=B_2, \theta, \delta)\}$ , and

$G' = G \setminus \{A_i\}$ ;

if ( $\exists B \in B, \chi\text{-mgu}(A_i, B, \sigma, \delta)$ )

if (top-down( $P, \neg\delta' \cup G'\theta, \sigma$ ))

return  $\theta\sigma$ ;

else;

else;

return 'Fail'.

### **3. A Simple Example**

We suppose the example we give now illustrates to some extent the ideas exposed in this paper.

Let be the DF-logic program (inspired by [CL,1990]):

$X[\text{happy}] :- X:\text{person}[\text{friend} \rightarrow \_ ]$ .

$\text{person}[\text{friend} \Rightarrow \text{person}]$ .

$X[F \rightarrow Y] :- Y[F:\text{symmetric} \rightarrow X]$ .

$\text{friend}:\text{symmetric}$ .

$\text{albert}:\text{person}[\text{friend} \rightarrow \text{lucy}]$ .

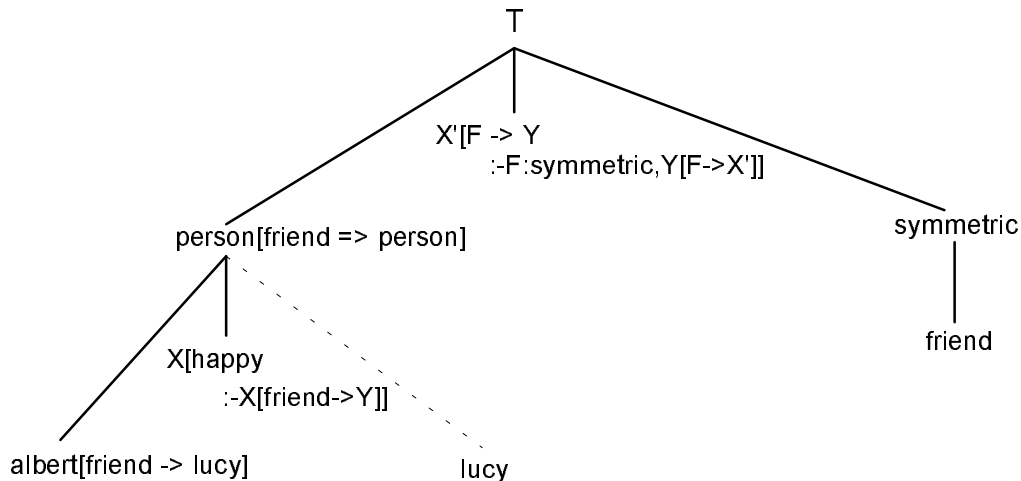
and the goal

?-  $\text{lucy}[\text{happy}]$ .

After (de)applying the easily understandable syntactic conventions used above, the first, the third, and the fifth clauses will be written:

```
X[happy] :- X:person, X[friend -> Z].
X[F -> Y] :- F:symmetric, Y[F -> X].
albert:person.
albert[friend -> lucy].
```

Inspired by the idea of definite feature atoms in [Abr,1990], the canonical DF-hierarchy associated to the above logic program could be visualized as it follows:



Remarks: The above hierarchy is in fact a non-conditional one, due to the simplicity of the considered program. The dotted line corresponds to the is-a relationship added in the second phase of the canonical DF-hierarchy associated to the DF-logic program.

The  $\chi$ -resolution produces the following goal lists:

G0:	lucy[happy].	$\chi$ -unifier: $\sigma_0 = \{X/lucy\}$ ;
G1:	lucy[friend -> Z].	$\chi$ -unifier: $\sigma_1 = \{X'/lucy, F/friend, Z/Y\}$ ;
G2:	friend:symmetric, Y[friend -> lucy].	
G3:	Y[friend -> lucy].	$\chi$ -unifier: $\sigma_3 = \{Y/albert\}$ ;
G4:	□.	

Remarks:

1. One could backtrack after G3, but no more solutions will be found.
2. We could enhance  $\chi$ -unification in DF-logic taking into account fully bounded class variables. Namely, each variable  $X$  in the expressions to be unified is characterized by a set of "bound" constraints  $X:b$ , with  $b$  constant symbol. In these new settings, the above resolution could be done in four steps, instead of five, by eliminating G2.

## Conclusion

We have defined a way for doing object-oriented inferences in DF-logic, a data frame-based logic obtained from F-logic (a frame-based logic introduced in [KLW,1990]) by restricting identification terms to variables and constant symbols. The is-a atoms in DF-logic are treated as class/object derivation relationships. We deal with these constraints in a dynamic way, i.e. we infer new derivation relationships starting from the given ones, through the so-called well-typing conditions, the is-a reflexivity, is-a transitivity, and functionality principles, and using a special unification procedure called  $\chi$ -unification. This computation on feature frames represented by DF-atoms involves type-inheritance, argument sub-typing, and range super-typing wrt a given (conditional) hierarchy of DF-atoms.

## Further Work

This work is the theoretical base for further implementing HPSGs for the Romanian language syntax.

## Bibliography

- [Abr,1990] H.Abramson, *Definite Feature Grammars for Natural Language Processing*, in Logic Programming and Natural Language Understanding, vol. III, 1990.
- [AKN,1986] H.Ait-Kaci, R.Nasr, *LOGIN: A Logic Programming Language with Built-in Inheritance*, J. Logic Programming, 1986:3, pp185-215.
- [Cio,1994] L.-V.Ciortuz, *Logic Programming with Built-in Object-Orientation*, Annals of "Ai.I. Cuza" University of Iasi, Computer Science Section, 1994 (to appear).
- [Hen,1989] P. van Hentenrick, *Constraint Satisfaction in Logic Programming*, MIT Press, 1989.
- [KLW,1990] M.Kifer, G.Lausen, J.Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, Technical Report 90/14, SUNY at Stony Brook, 1990.
- [Llo,1987] J.W.Lloyd, *Foundations of Logic Programming*, Springer-Verlag, second edition, 1987.

## Appendix: Declarative Semantics of DF-Logic

*Definition 1:*

The *semantic structure* (or: interpretation) of an DF-logic language  $L$  is a tuple

$$I = \langle U, \leq, I_C, I_{\rightarrow}, I_{\Rightarrow}, I_{\Rightarrow\Rightarrow}, I_{\Rightarrow\Rightarrow\Rightarrow}, I_P \rangle$$

where:

- $U$  - the universe of interpretation - is a non-empty set;
- $\leq$  is a partial order relation on  $U$ ;
- $I_C$  is the interpretation function for constant symbols, with  $I_C(c)$  being an element  $c'$  in  $U$ , for each constant symbol  $c$  in the language alphabet;
- $I_{\rightarrow}: U \rightarrow \prod_{n \geq 0} \text{Partial}(U^{n+1}, U)$  is the interpretation of functional methods.  
 $\text{Partial}(A, B)$  is the set of all partially defined functions from  $A$  to  $B$ .  
 When  $I_{\rightarrow}(m)(u, u_1, u_2, \dots, u_n)$  is defined, it denotes the value of the method  $m$  applied to the object/class  $u$  in the context of the  $u_1, u_2, \dots, u_n$  objects/classes;
- $I_{\Rightarrow}: U \rightarrow \prod_{n \geq 0} \text{PartialAntimonotone}(U^{n+1}, 2^{\uparrow}U)$  is the interpretation of function-typing methods.  
 Here  $2^{\uparrow}U$  denotes the set of all upward-closed subsets of the  $U$  (i.e.,  $A \subseteq U$  is in  $2^{\uparrow}U$  iff  $y \in A$  for all  $y \geq x$ , if  $x \in A$ ). A function  $f: A \rightarrow B$  is antimonotone if and only if whenever  $f(x)$  is defined, then for all  $y \in A$  with  $y \leq_A x$ ,  $f(y)$  is defined too, and  $f(x) \leq_B f(y)$ . Obviously,  $\leq_A$  and  $\leq_B$  are partial order relations on  $A$  and  $B$ , respectively. (The antimonotone property will be used for type inheritance.)  
 When  $I_{\Rightarrow}^{(n)}(m)(u, u_1, u_2, \dots, u_n)$  is defined, it will say that the value of the functional method  $m$  applied to the object/class  $u$  in the context defined by the  $u_1, u_2, \dots, u_n$  objects/classes belongs to each one of the classes in the set denoted by  $I_{\Rightarrow}^{(n)}(m)(u, u_1, u_2, \dots, u_n)$ ;
- $I_{\Rightarrow\Rightarrow}: U \rightarrow \prod_{n \geq 0} \text{Partial}(U^{n+1}, 2^U)$  is the interpretation of set-valuated methods.  
 $2^U$  denotes the powerset of  $U$ , i.e. the set of all subsets of the  $U$  set.  
 When  $I_{\Rightarrow\Rightarrow}^{(n)}(m)(u, u_1, u_2, \dots, u_n)$  is defined, it denotes that subset of  $U$  which is the value of the set-valuated method  $m$  applied to the object/class  $u$  in the context of the  $u_1, u_2, \dots, u_n$  objects/classes;
- $I_{\Rightarrow\Rightarrow\Rightarrow}: U \rightarrow \prod_{n \geq 0} \text{PartialAntimonotone}(U^{n+1}, 2^{\uparrow}U)$  is the interpretation of set-typing methods.  
 When  $I_{\Rightarrow\Rightarrow\Rightarrow}^{(n)}(m)(u, u_1, u_2, \dots, u_n)$  is defined, it will be used to say that each element in the value of the set-valuated method  $m$  applied to the object/class  $u$  in the context defined by the  $u_1, u_2, \dots, u_n$  objects/classes belongs to every class in the set denoted by  $I_{\Rightarrow\Rightarrow\Rightarrow}^{(n)}(m)(u, u_1, u_2, \dots, u_n)$ ;
- $I_P: U \rightarrow \prod_{n \geq 0} 2^{U^{n+1}}$  is the interpretation of predicative methods.  
 For each  $n \geq 0$ , the  $I_P^{(n)}(m)$  component of  $I_P(m)$  denotes the set of all  $(n+1)$ -uples  $\langle u, u_1, u_2, \dots, u_n \rangle$  belonging to  $m$ , when  $m$  is interpreted as a  $(n+1)$ -ary relation.

Two *well-typing conditions* link the interpretations of method values and types:

- i. if  $I_{\rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$  is defined, then  $I_{\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$  is defined, and if  $I_{\Rightarrow\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$  is defined, then  $I_{\Rightarrow\Rightarrow\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$  is defined too;
- ii. if  $q = I_{\rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$ , then  $q \leq r$  for each  $r \in I_{\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$ , and if  $q \in I_{\Rightarrow\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$ , then  $q \leq r$  for each  $r \in I_{\Rightarrow\Rightarrow\Rightarrow}^{(n)}(m)(a, a_1, a_2, \dots, a_n)$ .

We define a *variable assignment* in the language  $L$  a mapping that applies to each variable  $X$  an element  $u$  in the semantic universe  $U$ . The mapping could be naturally extended to the set of terms in the language  $L$ .

The *truth values* for the two *special predicates* used in the DF-logic settings, namely the *is-a* relation and the *equality* relation are defined (in the semantic structure  $\mathcal{I}$  and the variable assignment  $as$ ) as it follows.

*Definition 2:*

$$\begin{aligned} \mathcal{I} \models_{as} a:b \text{ iff } as(a) \leq as(b) \text{ in } U, \text{ and} \\ \mathcal{I} \models_{as} a=b \text{ is true iff } as(a) = as(b) \text{ in } U. \end{aligned}$$

Here  $\mathcal{I} \models_{as} F$  stands for: the formula  $F$  is true in the semantic structure  $\mathcal{I}$  and the variable assignment  $as$ , and  $\mathcal{I} \models F$  will say that  $F$  is true in the structure  $\mathcal{I}$  (and every assignment  $as$ ).

The truth values for non-compound DF-atoms (i.e. DF-atoms containing only one method) are given by the following equivalence relations.

*Definition 3:*

$$\begin{aligned} \mathcal{I} \models_{as} a[ma_1, a_2, \dots, a_n \rightarrow v] \text{ iff} \\ \quad \mathcal{I}_{\rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \text{ is defined and } \mathcal{I}_{\rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') = v', \text{ where} \\ \quad m', a', a_1', a_2', \dots, a_n', \text{ and } v' \text{ are respectively the } m, a, a_1, a_2, \dots, a_n, \text{ and } v \text{ images} \\ \quad \text{through } as. \\ \mathcal{I} \models_{as} a[ma_1, a_2, \dots, a_n \Rightarrow t_1, t_2, \dots, t_r] \text{ iff} \\ \quad \mathcal{I}_{\Rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \text{ is defined and } \mathcal{I}_{\Rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \supseteq \{t_1, t_2, \dots, t_r\}, \\ \quad \text{where } m', a', a_1', a_2', \dots, a_n', \text{ and } t_1, t_2, \dots, t_r \text{ are respectively the } m, a, a_1, a_2, \dots, a_n, \text{ and} \\ \quad t_1, t_2, \dots, t_r \text{ images through } as. \\ \mathcal{I} \models_{as} a[m@a_1, a_2, \dots, a_n \rightarrow v_1, v_2, \dots, v_m] \text{ iff} \\ \quad \mathcal{I}_{\rightarrow\rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \text{ is defined, and } \mathcal{I}_{\rightarrow\rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \supseteq \{v_1, v_2, \dots, \\ \quad v_m\} \text{ where } m', a', a_1', a_2', \dots, a_n', \text{ and } v' \text{ are respectively the } m, a, a_1, a_2, \dots, a_n, \text{ and} \\ \quad v_1, v_2, \dots, v_m \text{ images through } as. \\ \mathcal{I} \models_{as} a[m@a_1, a_2, \dots, a_n \Rightarrow t_1, t_2, \dots, t_r] \text{ iff} \\ \quad \mathcal{I}_{\Rightarrow\Rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \text{ is defined and } \mathcal{I}_{\Rightarrow\Rightarrow}^{(n)}(m')(a', a_1', a_2', \dots, a_n') \supseteq \{t_1, t_2, \dots, \\ \quad t_r\}, \text{ where } m', a', a_1', a_2', \dots, a_n', \text{ and } t_1, t_2, \dots, t_r \text{ are respectively the } m, a, a_1, a_2, \dots, a_n, \\ \text{ and } t_1, t_2, \dots, t_r \text{ images through } as. \\ \mathcal{I} \models_{as} a[m@a_1, a_2, \dots, a_n] \text{ iff} \\ \quad \mathcal{I}_p^{(n)}(m') \ni (a', a_1', a_2', \dots, a_n'), \text{ where } m', a', a_1', a_2', \dots, a_n' \text{ are respectively the} \\ \quad m, a, a_1, a_2, \dots, a_n \text{ images through } as. \end{aligned}$$

Finally, the truth value of an atom (1.1) is given by

*Definition 4:*

$$\mathcal{I} \models_{as} id[method_1; \dots; method_k] \text{ iff } \mathcal{I} \models_{as} id[method_1] \& \dots \& \mathcal{I} \models_{as} id[method_k]$$

For a non-compound atom

$$a[m@a_1, a_2, \dots, a_n \text{ q\_t } v_1, v_2, \dots, v_m]$$

with  $q\_t$  being  $\rightarrow, \Rightarrow, \rightarrow\rightarrow, \Rightarrow\Rightarrow$ , or  $\Rightarrow\Rightarrow\Rightarrow$ , we consider its *elementary sub-atoms*

$$a[m@a_1, a_2, \dots, a_n \text{ q\_t } v_k] \text{ for } k = 1, \dots, m.$$

Any other atom like  $a[m@a_1, a_2, \dots, a_n \rightarrow v]$  or  $a[m@a_1, a_2, \dots, a_n]$  is its own elementary sub-atom.

The *elementary sub-atom set* of an atom (1.1) is the union of the elementary sub-atom set of all non-compound sub-atoms of the given atom. Using the above truth relations, it follows that an atom (1.1) is true in the semantic structure  $\mathcal{I}$  (and the variable assignment  $as$ ) iff all its elementary sub-atoms are true in  $\mathcal{I}$  (and  $as$ ). So, any atom could be written under elementary form, i.e. it could be equivalently replaced by an atom having only elementary methods.

The other semantic definitions in DF-logic extend in a natural way the corresponding definitions in the first-order predicate calculus.