STUDIA UNIV. "BABEŞ–BOLYAI", INFORMATICA, Volume XLII, Number 2, 1997

REASONING WITH FRAME-BASED AND OBJECT-ORIENTED KNOWLEDGE

D. TATAR AND A. DUMITRESCU

Abstract. In this study, some possibilities to deal with incomplete information are investigated, using the connection between frame-based systems and object-oriented systems, made via F-logic formalisme developed in [10].

We present here a proof theory that is used for obtaining, as a deductive process, new information starting with a minimal, incomplete information.

1. Logical object-oriented language

1.1. **Introduction.** The frame based-systems deal with well-known or well-defined hierarchies of frames. The aim of this approach is to look at the possibility of reasoning about the hierarchical structure of a frame based system, while dynamically building it from incomplete bits of information. We will assume that at the beginning the hierarchy (or the set of frames) is incompletely known or, in the happiest case, a minimal structure is given.

It is knowed that "the object-oriented" approach of a database programming language is a term not entirely formalised from mathematical point of view. In [10] the authors give a novel formalism, F-logic, which "stands in the same relationship to the object-oriented paradigm, as classical predicate calculus stands to relational programming". This formalism has an important application in the area of frame-based languages in AI, where the notion as frames, slots and facets are viewed as objects and attributes. In first section we present F-logic formalisme developed in [10] and we present a proof theory that is used for obtaining a frame from information about it as a deductive process in this theory [1].

The second section presents the frame-based knowledge representation, stressing the types of incomplete information a frame structure can deal with [6]. Some sets of structural relations between the elements of a domain of expertise will be

Received by the editors: October 1, 1997.

1991 *Mathematics Subject Classification.* 68T30, 68T35.

1991 *CR Categories and Descriptors.* H.2.3 [**Database Management**]: Languages – *Data description languages*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving – *Deduction (e.g., natural, rule-based)*.

used as initial formulas in the proof theory in first section. Each relation represents a different, limited view of the domain to be represented. Through a process of relations' composition, new or hidden knowledge about the domain can be revealed and an equivalent frame structure can be built, as a new theorem obtained in F-logic.

In the third section we present an example of this idea.

1.2. **Basic concepts in F-logic.** As object-oriented approach seeks to group data around objects, in F-logic is used a concept that is called logical object identity (**loid**). Also, F-logic allows grouping of the data around properties, as in relational database languages. Thus, F-logic is a multiparadigm language which proves that relational and object-oriented paradigm can be reconciliated [11], [4], [8], [13]. The alphabet A of a logic language consists of

$$A = F \cup V \cup Aux \cup Op$$

where:

- F is the set of function symbols (object constructors). As usually, each function symbols has a fixed arity. The symbols of arity 0 are called **constants**.
- V is the set of variables.
- Aux is the set of auxiliary symbols, such as:$(,), [,], \rightarrow, \rightarrow\!\!\rightarrow, \Rightarrow, \Rightarrow\!\!\Rightarrow, ::, :$.
- Op is the set of usual logical conectives and quantifiers :$\wedge, \vee, \neg, \longrightarrow, \forall, \exists$.

A **term** (or id-term) is a usual first-order term composed of function symbols and variables.

A **ground** (variable-free) id-term is denoted **logical object id** or **loid**.

In the bellow definition we made some simplification from the original [10], by considering Q an id-term, and not a method with arguments (or, equivalentely, we will consider only method with 0 arguments). For our aims, this simplification is not restrictive.

Definition 1.1. *A* **molecular** *formula is one of the following :*

- *an* **is-a** *assertion of the form:*

$$O :: C,$$

with the meaning: the id-term (that denotes an object) O is a nonstrict subclass of class represented by the class C, or

$$O : S$$

with the meaning: the id-term O is a member of the class S.

- O[data expression, $\cdots$,data expression], *where* data expression *is:*
  a noninheritable scalar expression:

$$Q \rightarrow T$$

that means that the id-term or the attribute Q takes the value T, where T is a **loid** *or a molecular formula.*

- *a noninheritable set-valued expression:*

$$Q \twoheadrightarrow \{S_1, \cdots, S_n\}$$

that means that the id-term or the attribute Q takes the value from the set of **loid**'*s or molecular formulas $S_1, \cdots, S_n$.*

- *an inheritable scalar expression:*

$$Q\odot \rightarrow T$$

that means that the id-term or the attribute Q takes the value T, as above, and this value can be inherited by the subclass and individual members of the object Q, when it plays the role of a class.

- *an inheritable set-valued expression:*

$$Q\odot \twoheadrightarrow \{S_1, \cdots, S_n\}$$

that means that the id-term or the attribute Q takes the value from the set of **loid**'*s or molecular formulas $S_1, \cdots, S_n$. and this values can be inherited by the subclass and individual members of the object Q, when it plays the role of a class.*

- O[signature expression, $\cdots$,signature expression], *where* signature expression *is:*
  - *a scalar signature expression: $Q \Rightarrow (A_1, \cdots, A_r)$, that means that the id-term or the attribute Q must be of types that simultaneously belong to classes $A_1, \cdots, A_r$.*
  - *a set-valued signature expression: $Q \Rightarrow\!\!\!\Rightarrow (B_1, \cdots, B_n)$ that means that each element of the set of id-terms or the attributes Q must be of types that simultaneously belong to classes $B_1, \cdots, B_n$.*

Informaly, a molecular F-formula asserts that the object denoted by O has properties specified by the data expression inside the brackets "[" and "]" or has type constraints specified by the signature expression inside the brackets "[" and "]".

Now we can define the formulas in F-logic, the F-formulas, which are buit by means of logical conectives and quantifiers as in definition:

Definition 1.2.
A molecular formula is a F-formula.
If φ and ψ are F-formulas, then $\varphi \wedge \psi, \varphi \vee \psi, \varphi \longrightarrow \psi, \neg\varphi$ are F-formulas.
If X, Y are variables, then $(\forall X)\varphi, (\exists X)\varphi$ are formulas.

1.3. **A proof system.** This section describes a proof theory for the F-formulas. The deductive rules are separated in two classes: the general and the local deductive rules. The general rules expres some properties of the "operations" $\rightarrow, \twoheadrightarrow$

$,\Rightarrow,\Rightarrow\Rightarrow,::,:$. The local deductive rules correspond to some structural relation possible between the objects (frames, slots, facets) of a frame-based system. Some of these relations will be presented in the next section [6]. The large number of inference rules in our proof theory stems from the rich semantics of any logical system that attempts capture the object-oriented paradigm.

In F-logic much of the theory of unifiers carries over from the clasical predicate calculus [10]. Thus, a **substitution** is a mapping $\sigma : V \longrightarrow T$, where T is the set of id-terms of a language L. As in classical logic, substitutions extend to mappings $L \longrightarrow L$ as follows:

$$\sigma(f(t_1,\cdots,t_n)) \doteq f(\sigma(t_1),\cdots,\sigma(t_n)).$$

A substitution can be further extended to a mapping from F-formulas to F-formulas by distributing σ through formulas components:

$$\sigma(Q:P)=\sigma(P):\sigma(Q),\sigma(Q::P)=\sigma(P)::\sigma(Q)$$
$$\sigma(O[Q\rightarrow T])=\sigma(O)[\sigma(Q)\rightarrow\sigma(T)])$$

Definition 1.3. *Let T_1 and T_2 be a pair of id-terms or of* **is-a** *F-formulas. A substitution σ is a unifier of T_1 and T_2 if $\sigma(T_1)=\sigma(T_2)$.*

In [10] is defined also the unification for tuples of id-terms.

Definition 1.4. *The tuples $< P_1,\cdots,P_n >$ and $< Q_1,\cdots,Q_n >$ are unifiable if there is a substitution σ such that $\sigma(P_i)=\sigma(Q_i)$, for $i=1,\cdots;n$.*

The *mgu* unifier is defined as usually.

The following rules are obtained from the properties of F-formulas and are accordingly with the deductive rule for resolution in [10].

The first rules capture the semantics of the subclass-relationship and its interaction with class membership and we will call them **is-a** rules.

I1: $P::Q,Q'::P'\vdash\sigma(P=Q)$, if $\sigma=mgu(< P,Q >,< P',Q' >)$.
I2: $P::Q,Q'::R\vdash\sigma(P::R)$, if $\sigma=mgu(Q,Q')$.
I3: $P:Q,Q'::R\vdash\sigma(P:R)$ if $\sigma=mgu(Q,Q')$.

Type-inference rules

T1: $O[Q\Rightarrow T],O1::O2\vdash\sigma(O1[Q\Rightarrow T])$, if $\sigma=mgu(O,O2)$. (the type inheritance)
T2: $O[Q\Rightarrow\{S_1,\cdots,S_n\}],O1::O2\vdash\sigma(O1[Q\Rightarrow\{S_1,\cdots,\S_n\}])$, if $\sigma=mgu(O,O2)$. (similarly to 1 for set valued methods)
T3: $O[Q\Rightarrow T],Q1::Q2\vdash\sigma(O[Q1\Rightarrow T])$ if $\sigma=mgu(Q,Q2)$. (input restriction)
T4: $O[Q\Rightarrow\{S_1,\cdots,S_n\}],Q1::Q2\vdash\sigma(O[Q1\Rightarrow\{S_1,\cdots,S_n\}])$, if $\sigma=mgu(Q,Q2)$. (similarly to 3 for set valued methods)
T5: $a.O[Q\Rightarrow T],T1::T2\vdash\sigma(O[Q\Rightarrow T2])$, if $\sigma=mgu(T,T1)$.

$b.O[Q \Rightarrow T], O1[Q1 \Rightarrow T1] \vdash \sigma(O[Q \Rightarrow O1[Q1 \Rightarrow T1]])$ if $\sigma = mgu(T, O1)$. (output restrictions)

Union rules

R1: $O[Q \Rightarrow T_1; Q_1 \Rightarrow T], O[Q \Rightarrow T_2] \vdash O[Q \Rightarrow \{T_1, T_2\}; Q_1 \Rightarrow T]$
R2: $O[Q \Rightarrow T], O1[Q_1 \Rightarrow T_1], O :: O1 \vdash O[Q \Rightarrow T; Q_1 \Rightarrow T_1]$ This rules are doubled for the $\rightarrow$ replacing $\Rightarrow$.

Modus-ponens rule (as in first-order logic) $MP\ U, U \longrightarrow V \vdash V$
Theorems in proposition and predicata calculus

Definition 1.5. (**Deduction from a set of F-formuas**). *Given a set of* **S** *of F-formulas, a* **deduction** *of a F-formula* U *from* **S** *is a finite sequence of F-formulas* $U_1, \cdots, U_n$ *such that* $U_n = U$, *and for* $i = 1, \cdots, n$,U_i *is either:*
-a member of S.
-is a theorem as above.
-is derived from some U_k *and* U_j, $k, j < i$, *using one of the* **is-a** *rule, type inference rule, union rules or modus-ponens.*

Remark 1.6. *The only restriction in a deduction is that firstly are applied the rules for inheritable data and scalar expressions, and then the rules for noninheritable data and scalar expressions.*

The process of signature accumulation by an object can be now simulated by a deduction from a set of F-formulas. Let us illustrate the example from [10].

Example 1.7. *The set* S *of F-formulas are:*

$$\begin{gathered}
\text{empl} :: \text{person}, \text{assistant} :: \text{student}, \text{assistant} :: \text{empl} \\
\text{person}[\text{name} \Rightarrow \text{string}] \\
\text{student}[\text{drinks} \Rightarrow \text{beer}; \text{drives} \Rightarrow \text{bargain}] \\
\text{empl}[\text{salary} \Rightarrow \text{integer}; \text{drives} \Rightarrow \text{car}] \\
\text{assistant}[\text{drives} \Rightarrow \text{oldThing}]
\end{gathered}$$

The signature accumulated by *assistant* from all sourses will be obtained by a deduction from S as follows:

$$U_1 : empl :: person$$
$$U_2 : person[name \Rightarrow string]$$
$$U_3 : empl[name \Rightarrow string]; U_1, U_2 \vdash_{T_1} U_3$$
$$U_4 : assistant :: empl$$
$$U_5 : assistant[name \Rightarrow string]; U_3, U_4 \vdash_{T_1} U_5$$
$$U_6 : assistant[drives \Rightarrow oldThing]$$
$$U_7 : student[drinks \Rightarrow beer; drives \Rightarrow bargain]$$
$$U_8 : assistant :: student$$
$$U_9 : assistant[drinks \Rightarrow beer; drives \Rightarrow bargain],, U_7, U_8 \vdash_{T_1} U_9$$
$$U_{10} : assistant[drinks \Rightarrow beer; drives \Rightarrow (bargain, oldThing)],, U_6, U_9 \vdash_{R1} U_{10}$$
$$U_{11} : empl[salary \Rightarrow integer; drives \Rightarrow car]$$
$$U_{12} : assistant[drinks \Rightarrow beer;$$
$$drives \Rightarrow (bargain, oldThing, car)], salary \Rightarrow integer], U_{10}, U_{11} \vdash_{R2} U_{12}$$

The formula U_{12} represents the final signature accumulated by the object *assistant*.

2. Knowledge representation by frames

2.1. Introduction. Frames are structures that represent a chunk of knowledge about a small domain of the world. They are rather like a stereotype of a situation or thing. Frames therefore set up the expected items in a given situation although these may easily be modified.

A frame consists of attributes (slots) each of them describing a s pecific aspect of the concept it represents. The attributes have to reflect the important features of the represented concept, according to the chosen point of view. They can be shared by different frames or they can refer to other frames, thus giving the possibility to represent different visions of the same object. Each attribute is described by facets that specify the nature of the attribute and the behaviour associated to it.

The descriptions in a frame, also called slots, generally consist of two parts: a slot-name, which specifies an attribute, and a slot-filler, which qualifies that attribute.

Therefore the structure of a frame is

```
<frame: (slot 1 (facet 1  value 1)
                           ...
                (facet n  value n))
                ...
        (slot q (facet 1  value 1)
                           ...
                (facet m  value m))>
```

where each attribute is described by facets that specify the nature of the attribute and the behaviour associated to it.

The classical facets values. The values can be simple (the integer, real, Boolean and string values) or complex. The latter ones usually represent pointers towards other frames. They give the type of attribute and their value represent the name of another concept defined in the frame hierarchy. The type facet allows the system to verify the coherence of a complex value that is to be assigned to an attribute with the attribute itself. The value assigned to an attribute has to be of a type that has as father a concept hierarchically equal or inferior to the attribute's type.

2.2. **Reasoning with frames.** Frames form a hierarchy such that each level of frames is more specialised than the previous level of frames. They follow an inheritance hierarchy such that default values of a class frame are propagated across the class/subclass and class/me mber hierarchy. Obviously this is very suited to object oriented programming. In the frame representation the prototype of a situation is memorised as a unit of knowledge. The particular situations related to that prototype are memo rised in form of knowledge units linked to it. This latter units contain only the information which is different from the one contained in the prototype. The reasoning is based on re-memorisation: being given a network of units representing a domain of application, it is requested to find the unit that can be used as prototype to describe the current situation.

So the basic idea is to seek (according to some heuristics specific to each domain) the objects that fit the current situation and then, if needed, to classify the found objects according to their degree of appropriateness with the current situation.

There are two main styles of reasoning with frames: first, as described above, the matcher must decide in a given situation which of the many frames it is the best match. This requires the matcher to be able to receive information about the existing si tuation and perform a best match type search on all the frames it has in its knowledge base.

Problems obviously occur when a default value has been overridden in a frame, because the matcher will have problems in recognising that, although the default value was overridden, the given frame fits in the closest way to the same frame as if the de fault value wouldn't have been overridden. So the reasoner needs to go up the frame hierarchy checking each slot starting from the slots directly associated with that frame, once these are exhausted it goes up the tree getting values from its parents' slots. This is the inheritance mechanism which is the second main reasoning style.

An instance of a frame possesses only its particular properties, the general properties being inherited in a dynamic way. Ignoring the exact value of one or more of these properties doesn't prevent the definition of an instance that can be completed later. All the modification at a certain level of the hierarchy are propagated without any other treatment to all the descendants of that object.

An informant can state:

i) an object belongs to a class/ this object inherits from these classes

ii) an object does not belong to a certain class / this object does not inherit from those classes

iii) an object might belong/inherit from one of these classes

iv) nothing is known about the class to which an object belongs

It is possible to have an incomplete mechanism for the control of reasoning i.e. the order of the events could be incompletely specified. One may know that there is a certain order given by the "pre required" and the "post required" attributes, but some of these attributes might miss and/or the order might not be precise.

In conclusion, frames store information in much larger chunks than other methods of representation and tend to focus more on the relevant issues. Having the idea of default-values is a very powerful one as it saves the same information having to be continually entered and enables the system to make assumptions if no specific data is given. This can be totally valid in many scenarios and any deviations from the norm can be easily fitted into the model by changing the value from the default to the actual. Another important idea introduced is that of having procedures attached to frames. These allow the system to deal with situations as they arise in a far more flexible fashion than a simply 'static' information based system such as semantic nets. Also with frames, the hierarchy is generally quite simple for humans to follow and as such it is a less onerous task for the designer to enter all the initial frames. It is quite possible for experts in various domains to present their knowledge to a frame based system without a great deal of difficulty. To resume, the frame representation allows i) to consider the same event following different perspectives ii) to describe an object in different ways, according to the inheritance that is highlighted iii) to consider only a part of the information that represents a particular vision upon an object and use that part in reasoning iv) to describe the objects by comparing them with the prototypes (parents, ancestors) allowing some information to miss in the description of a certain object

2.3. **Structural relations.** To build a system of frames might be, sometimes, a complex task, requiring a high degree of expertise. An alternative to directly building the system is to describe the domain of expertise by relations between its elements which give, individually, a very partial knowle dge, but whose assembling might lead to the desired frame system.

In what follows, by a feature of a frame a slot or a facet is meant. Mainly, the following relations will concern the slots, the declarative features and their values.

Let S be a frame system representing a certain domain. S can be either a hierarchy of frames or a set of frames.

Let $St(F)$ denote the structure of a frame F, that is the set of its slots, with the relations between them. $St(F) = Pu(F) + Pr(F)$, where $Pr(F)$ is the set of slots private to the frame F, that is the slots which cannot be inherited by other frames and $Pu(F)$ is the set of public slots of F, that is the set of characteristics which F can transmit to other frames. The structure of a slot will be referred in a similar way: $sl(slot) = pu(slot) + pr(slot)$.

Most of the relations defined here can be used not only to describe the properties of the slots of a frame, but also to describe the properties of the facets and, also, slots' and facets' properties related to a certain value of anot her slot or facet. It means that these relations can be used at 3 levels: (1) values of slots or of facets, (2) facets and (3) slots.

$$1.1 alike(F1.f1, ..., Fn.fn)$$

Two features f and g of two frames Fi and Fj are called alike features iff they are entirely the same, with the single possible exception of their names. This property will be denoted by the relation $Fi.f$ alike $Fj.g$ or by $alike(Fi.f, Fj.g)$.

Example 2.1. *Let us take the slots angle and length of two frames called triangle and segment.*

Both the slots length and angle can have the following identical structure value: type: real default: 0 domain: 0, infinite property: additivity

Such a representation of the two slots is sufficient for many purposes. The identity of structure can be expressed as the relation

triangle.angle alike segment.length. The relation *alike* introduces in a set of formulas the following implication:

$$U_{10}; alike(U.X, V.Y) \longrightarrow (U[S \Rightarrow X.Z] \longrightarrow V[S' \Rightarrow Y.Z])$$

$$1.2. similar(F1.f1, ..., Fn.fn)$$

Two features $f1$ and $f2$ of two frames, $F1$ and $F2$ are called similar if they have the same public facets. This can be written as $pu(F1.f1) = pu(F2.f2)$ and the relation will be expressed as follows: $F1.f1$ *similar* $F2.f2$ or *similar*$(F1.f1, F2.f2)$.

In the upper example, let the slot length to have the same structure and change the structure of angle to angle value

```
type: real
default : 0
domain: (0, infinite)
property: additivity
```

$$1.3. unlike(F1.f1, ..., Fn.fn)(F1.f unlike F2.g)$$

The feature $F1.f$ of a frame $F1$ is unlike the feature $F2.g$ of a frame $F2$ iff none of their characteristics are the same.

This relation can be expressed as $sl(F1.f)$ intersected with $sl(F2.g)$ is the empty set.

Example. The slot cartesian belonging to a frame coordinate and defined as coordinate cpoint is a location cangle is an angle cartesian type Boolean default true has no common facet with and it is entirely different of the slot angle given in the upper example. We write this as coordinate.cartesian unlike triangle.angle or unlike(coordinate.cartesian, triangle.angle).

$$1.4.rare : rare(f, F1, ..., Fn, max)$$

A rare feature of a set of frame is a feature that does almost never occur in the frames of that set.

For example, in the above hierarchy, the slot area is a rare feature in the set of frames that form the hierarchy, since it appears only in the frame circle. This can be written $rare(area, segment, line, angle, circle)$

$$1.5.common(f, F1, ..., Fn, min)$$

A feature f is called common to a set of frames is it appears in the structure of the most of the frames of that set.

For instance, the slot sort-of, which appears in most of the frames of the upper hierarchy can be regarded as a common feature. Like in the case of the rare feature, a numeric variable can state the minimum number of occurrences of the feature in the frames of the set: there is an $m >= min$ such that there is a subset $i1, ..., im$ of $1, ..., n$ for which f belongs to $St(Fij)$ for every $j = 1, ..., m$.

$$1.6.must(f1, ..., fm, F1, ..., Fn)$$

The occurence of a public feature common to a set of frames can be stated by the must relation: $must(feature, setofframes)$. The relation $must(f, F1, ..., Fn)$ can be expressed as for every i from $1, ..., n$ f belongs to $Pu(Fi)$.

$$1.7.should(f1, ..., fm, F1, ..., Fn)$$

The relation should states that the occurrence of a feature in the frame's structure is sufficient to characterize that frame, meaning that the feature is a private characteristic of the frames belonging to that set. That is: for every I from $1, ..., n$ f belongs to $Pu(Fi)$.

3. Example

Let us consider the following example of frame given by some informations.

```
concept type(segment.angle,slot)
alike(segment.length,?.angle)
facet(lenght,name,L)
facet(lenght,type,real)
facet(lenght,default value,0)
facet(length,property,aditivity)
```

Altough incomplete,this single frame provides a big amount of information about the described concept.The final frame can be represented as the following:

```
segment
    length
        name L
        type real
        default value 0
        property additivity
    angle
        type real
        default value 0
        property additivity
```

As a molecular F-formula, this frame can be described as:

$$segment[slot1 \Rightarrow length[name \rightarrow L, type \rightarrow real, property \rightarrow additivity, defaultvalue \rightarrow 0],$$

$$slot2 \Rightarrow angle[type \rightarrow real, property \rightarrow additivity, defaultvalue \rightarrow 0]]$$

This F-formula can be obtained by a deduction from the set S of formulas corresponding to above informations:

$$U_1 : segment[slot1 \Rightarrow angle]$$

$$U_2 : slot2 : segment$$

$$U_3 : segment[slot2 \Rightarrow lenght]$$

$$U_4 : alike(segment.lenght, X.angle)$$

$$U_5 : lenght[type \rightarrow real$$

$$U_6 : lenght[defaultvalue \rightarrow 0$$

$$U_7 : lenght[property \rightarrow additivity]$$

The new formulas obtained by deduction are:

$$U_8 : segment[slot2 \Rightarrow lenght[type \rightarrow real]] U_3, U_5 \vdash_{T_5 b} U_8$$

Finaly,

$$U_9 : segment[slot2 \Rightarrow lenght[type \rightarrow real, defaultvalue \rightarrow 0, property \rightarrow additivity]]$$

$$U_{10}; alike(U.X, V.Y) \longrightarrow (U[S \Rightarrow X.Z] \longrightarrow V[S' \Rightarrow Y.Z])$$

(from semantics of relation "*alike*")

$$U_{11} : segment[S \Rightarrow lenght.Z] \longrightarrow V[S' \Rightarrow angle.Z]; U_4, U_{10} \vdash_{MP} U_{11}$$

$$U_{12} : segment[slot1 \Rightarrow angle[type \rightarrow real, defaultvalue \rightarrow 0, property \rightarrow additivity]]; U_9, U_{11} \vdash_{MP} U_{12}$$

$$U_{13} : segment[slot2 \Rightarrow length[type \rightarrow real, property \rightarrow additivity, defaultvalue \rightarrow 0],$$

$$slot1 \Rightarrow angle[type \to real, property \to additivity, defaultvalue \to 0]]$$

where $U_9, U_{12} \vdash_{R2} U_{13}$

In the previous deduction we assumed that all the data expressions and signature expressions were inheritable,without noticing that by $\odot \to$ or $\odot \Rightarrow$,to simplify the notation. As we remarked, this rules are always applied firstly. If we suppose that,in the set S exists the F-formula:

$$U_{14} length[name \to L]$$

, were $\to$ means now that this value of attribute *name* of object *length* is noninheritable, then from U_{14} and U_{13} we obtain the final frame:

$$U_{15} : segment[slot2 \Rightarrow length[name \to L, type \to real, property \to additivity, defaultvalue \to 0],$$

$$slot1 \Rightarrow angle[type \to real, property \to additivity, defaultvalue \to 0]]$$

References

[1] K.H. Blasius, H.J. Burkert, *Deduction systems in Artificial Intelligence*, Ellis Horwood Ltd., 1989.

[2] S. Ceri, G. Gottlob, L. Tanca, *Logic Programming and Databases*, Spriger-Verlag, 1990.

[3] K.L. Clark, *Predicate Logic as a computational formalism*, Res. Mon. 79/59 TOC, Imperial College, London, 1979.

[4] J. Cohen, *Constraint logic programming languages*, Comm. of the ACM, 33 (1990), pp. 52–68.

[5] P. Deransart, J. Maluszynski, *Relating logic programs and attribute grammars*, J. of Logic Programming, 2 (1985), pp. 119–155.

[6] A. Dumitrescu, *Incomplete information in frame-based systems*, Report LIA Universite de Savoie, October, 1996. (adviser L. Siklossy)

[7] M. Fitting, *First-Order Logic and Automated theorems proving*, Springer-Verlag, 1990.

[8] J.Jaffar,M.J.Maher:"Constraint logic programming:a survey" *J.Logic programming* 1994:19,20:pp503-581.

[9] 0 K.L.Kwast:"The incomplete database",*Proceedings of IJCAI-91* ,pp 897-902. *J.of Automated Reasoning*, vol5, 1989, pp.167–205.

[10] M.Kifer,G.Lausen,J.Wu:"Logical Foundations of Object-Oriented and Frame-Based Languages",Journal of ACM,vol.42,no.4,July 1995,pp741-843.

[11] J.Minker: "Perspective in deductive databases" *J.of Logic Programming*, vol.5, 1988, pp.33–61.

[12] D.Tatar:"Logic grammars as formal languages",Studia Universitas"Babes-Bolyai", 1994,nr.3.

[13] A.Thayse(ed):"From standard logic to logic programming"1988,John Wiley ,Sons.

[14] M.H.van Emden, R.A.Kowalski: "The semantics of predicate logic", *Journal of ACM*, oct.1976, pp.733–742.

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogălniceanu 1, Romania

E-mail address: `dtatar@cs.ubbcluj.ro`

Laboratoire d'Intelligence Artificielle, Universite de Savoie, France

E-mail address: `adina@univ-sovoie.fr`