

# Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases<sup>\*</sup>

Guizhen Yang   Michael Kifer

Department of Computer Science  
Stony Brook University  
Stony Brook, NY 11794, U.S.A.  
{guizyang,kifer}@CS.StonyBrook.EDU

**Abstract.** F-logic is a popular formalism for knowledge-intensive applications and, especially, for ontology management in Semantic Web. However, the original F-logic's semantics for inheritance suffers from a number of anomalies when inheritance and deduction closely interact. This work rectifies this problem and develops a natural model-theoretic semantics for inheritance in frame-based knowledge bases, which supports inference by inheritance as well as inference via rules. Inference by inheritance supports a multitude of features, such as overriding and nonmonotonic multiple inheritance, meta programming, and dynamic inheritance hierarchies — the features that are fundamental to advanced knowledge management. This semantics has been effectively implemented in the Flora-2 system which is extensively used in a number of projects. To the best of our knowledge, this work is the only model-theoretic semantics for nonmonotonic multiple inheritance that applies to general, unrestricted frame-based knowledge bases and has several independent characterizations, which testifies to its naturalness and robustness. The problems discussed in this paper are inherent in any logic-based system that supports inheritance and deductive rules and our techniques apply to such systems. In particular, they apply to DAML+OIL extended with rules and inheritance.

## 1 Introduction

F-logic (Frame Logic) [21, 22] is a formalism that unifies the deductive and object-oriented programming paradigms. Since then several implementations have been made available, such as FLORID<sup>1</sup>, SILRI<sup>2</sup>, FLORA<sup>3</sup>, and TFL<sup>4</sup>. Some of these systems have served as critical components in projects ranging from data integration in neuroscience [16], to processing semistructured and semantic information on the Web [11, 9], to information mediation [15], to commercial and research prototypes of Web information management systems [8, 10, 30].

---

<sup>\*</sup> This work was supported in part by NSF grants INT-9809945 and IIS-0072927.

<sup>1</sup> <http://www.informatik.uni-freiburg.de/~dbis/florid/>

<sup>2</sup> <http://ontobroker.semanticweb.org/silri/>

<sup>3</sup> <http://flora.sourceforge.net/>

<sup>4</sup> <http://www.dsic.upv.es/~pcarsi/tfl/>

The use of F-logic for ontology management was pioneered by the Ontobroker project [30, 10] and since then it has become one of the primary contenders to provide formalization and inference mechanism for the various aspects of Semantic Web. In [9], F-logic was advocated as an inference service for RDF and later an F-logic based language, TRIPLE, was developed specifically for this purpose [29]. The ability to handle RDF also exists in FLORA-2 [33, 34], a powerful system for programming knowledge-based applications, which is built around the semantics of F-logic, HiLog [6], and Transaction Logic [2].

Inheritance is one of the key aspects in frame-based knowledge representation and it has been extensively studied both in the AI and database literature (see [31, 3, 5, 1, 4, 25, 22, 19, 20] just to name a few). Unfortunately, as pointed out in [22], integration of inference by inheritance into rule-based deductive systems presents serious semantic and computational difficulties. Although F-logic, as described in [22], resolved many semantic and proof-theoretic issues in frame-based knowledge systems, nonmonotonic multiple inheritance was handled in an ad hoc, nonlogical way. In fact, the semantics proposed in [22] was known to yield questionable results in many cases. Subsequent works by a number of authors [24, 4, 25, 1, 19, 20] tried to either “justify” this flawed (in our opinion) semantics or propose new ones, which worked by extensively restricting the syntax of F-logic and eliminating some of the desired features (like meta-programming, virtual class definitions, etc.).

In this paper, we solve all these problems and develop a natural model-theoretic semantics for nonmonotonic multiple inheritance in F-logic. We also describe its implementation in Flora-2 [33, 34], which is based on the top-down tabling inference engine of XSB<sup>5</sup>. While the original fixpoint procedure [22] for computing inheritance in F-logic was inherently bottom-up, the new semantics can be implemented either bottom-up, using an extended alternating fixpoint, or top-down, using XSB’s realization of the SLG resolution [7].

We should note that the problems discussed in this paper are *inherent* in logic-based systems that support inheritance and deductive rules. Our techniques apply to such systems and, in particular, to DAML+OIL [17] when it is extended with rules and inheritance.

We now briefly survey the literature on inheritance. To make our comparison concrete, we first list the main features of inheritance that, in our opinion, must be supported by a frame-based knowledge base system:

- implicit inference by inheritance, as well as explicit inference via rules
- overriding by intermediate superclasses
- dynamic class hierarchies, *i.e.*, the ability to define both ISA membership and subclass relationship via rules
- nonmonotonic inheritance from multiple superclasses that are incomparable with respect to the subclass relationship
- meta-programming, by which variables can range over class and method names

---

<sup>5</sup> <http://xsb.sourceforge.net>

There is a large body of work based on Touretzky’s framework of Inheritance Nets [31]. On one hand, the overriding mechanism in this framework is more sophisticated than what is typically considered in the knowledge base context. On the other hand, this framework supports neither deductive inference via rules nor dynamic class hierarchies, which makes it too weak for many applications of knowledge bases. We will not discuss this framework any further.

Abiteboul et al. [1] propose a framework for implementing inheritance that is based on program rewriting using Datalog with negation. In spirit, this implementation is close to our implementation in Flora-2. However, [1] is not based on a formal, model-theoretic formalization. On the practical side, this framework excludes nonmonotonic multiple inheritance and makes strong assumptions, such as that programs must have a total (two-valued) well-founded model. This property is undecidable without far-reaching syntactic restrictions.

Bugliesi and Jamil [4] propose a model-theoretic semantics for inheritance with overriding which bears close resemblance to two-valued stable models [14]. However, their semantics applies only to negation-free programs (a severe limitation in practice) and does not handle multiple inheritance conflicts properly. More importantly, [4] is not backed by an algorithm.

May et al. [25] apply the ideas behind the well-founded semantics to F-logic. However, inheritance is still dealt with in the same way as in the original F-logic. Apart from being ad hoc, this semantics is known to produce counter-intuitive results when dynamic class hierarchies interact with overriding and multiple inheritance (cf. Section 2).

In [19, 20], Jamil introduces a series of techniques to tackle the inheritance problem. Among these, the ideas of *locality* and *context*, which were proposed to resolve code inheritance and encapsulation in the language Datalog<sup>++</sup>, have influenced our approach the most. However, these works do not define a model-theoretic inheritance semantics and support neither dynamic class hierarchies nor meta-programming. In [19] the inheritance semantics is defined by program rewriting while in [20] the approach is proof-theoretic.

In contrast to all these previous works, we propose a comprehensive framework and develop a natural model theory for nonmonotonic multiple inheritance in knowledge base systems. We adopt the well-founded semantics [13, 12] and extend it with the ideas of locality and context [19]. In order to capture the common intuition behind overriding and conflict resolution in multiple inheritance, we formalize locality and context in the setting of *three-valued* models and introduce the concept of *inheritance candidacy*. We then formally define the *inheritance postulates*, which embody the common intuition and the main principles behind nonmonotonic multiple inheritance. To the best of our knowledge, this intuition has not been formalized before. We treat the inheritance postulates as the minimum requirements for an *object model* of a program. To further tighten the semantics, we apply the principle of *well-founded optimism* and develop the notion of a *unique* canonical model, called *optimistic object model*, which exists for any program. “Optimism” here means that we apply the closed world assumption with a bias towards undefinedness rather than falsehood.

The new optimistic object model semantics satisfies all the aforementioned desiderata for inheritance, produces intuitively satisfactory results in all known “benchmark” cases, does not impose syntactic restrictions on the programs (beyond requiring them to be rule-based), and has been effectively implemented in the Flora-2 system. The proposed semantics is *robust* in the sense that it can be characterized in at least three different ways: as the least fixpoint of an extended alternating fixpoint operator, as a minimal object model (with respect to truth ordering), and as the intersection of all *three-valued stable* object models.

The paper is organized as follows. Section 2 introduces the basic F-logic syntax, defines its three-valued semantics, and illustrates the main challenges in dealing with nonmonotonic multiple inheritance in the presence of deductive rules and dynamic class hierarchies. Section 3 describes our model theory for nonmonotonic multiple inheritance and Section 4 provides an alternating fixpoint computation for the optimistic object model of any F-logic program. The properties of optimistic object models are discussed in Section 5. Section 6 describes an implementation of our semantics by translation to general logic programs, which can be executed by any deductive engine that supports the well-founded semantics for negation, and formally proves that this implementation is correct. The data complexity of computing optimistic object models is discussed in Section 7. Section 8 concludes the paper.

Limitation of space does not allow us to present the proofs and expand on the relationship between optimistic object models and three-valued stable object models. However, it can be shown that, similarly to [27], three-valued stable object models that obey all inheritance postulates can be defined and the optimistic object model is the least among these models.

All proofs and a detailed account of the theoretical development of the optimistic object model semantics can be found in a technical report available at <http://www.cs.sunysb.edu/~guizyang/papers/inheritance.ps>

## 2 Preliminaries

### 2.1 Basic Syntax

To simplify the exposition, we focus on a subset of F-logic, which includes only two kinds of atoms: those that represent the subclass relationships and those that represent inheritable multivalued method specifications. An atom of the form  $s::c$  says that  $s$  is a subclass of  $c$ , while  $s[m \twoheadrightarrow v]$ <sup>6</sup> specifies that  $s$  has an inheritable multivalued method,  $m$ , whose return value is a set, and  $v$  is one of the members in that set. The symbols  $s$ ,  $c$ ,  $m$ , and  $v$  in the above atomic formulas are first-order terms, which represent the ID of an object, a class, a method, and a value of the method, respectively. Moreover, the terms that represent these entities in a program can contain variables and, thus, they can represent multiple

<sup>6</sup> To reduce clutter, we slightly depart from the syntax of F-logic and Flora-2 and use  $\twoheadrightarrow$  instead of  $\star\rightarrow$  to represent inheritable multivalued methods.

objects, one per variable instantiation. This design makes meta-programming in F-logic as natural as querying.

Let  $A$  be any atom. A literal of the form  $A$  is called a *positive* literal while a literal of the form  $\neg A$  is called a *negative* literal. An F-logic program is a finite set of rules where all variables are universally quantified. An F-logic rule has the following form:  $\forall(H \leftarrow L_1 \wedge \dots \wedge L_n)$  where  $n \geq 0$ ,  $H$  is a positive literal, and  $L_i$  ( $1 \leq i \leq n$ ) is either a positive or a negative literal.  $H$  is called the *head* of the rule, and the *conjunction* of  $L_i$ 's is called the *body* of the rule. The symbol  $\forall$  indicates that all variables are universally quantified. Following the standard convention, we will omit universal quantifiers in the rules and simply write  $H \leftarrow L_1, \dots, L_n$ . We will also use uppercase names to denote variables and lowercase names to denote constants. A rule with an empty body is called a *fact*. When writing down the facts, we will omit the implication symbol and simply show the head.

## 2.2 Motivating Examples

We now illustrate some of the main issues that arise from the interaction among deduction, inheritance, and dynamic class hierarchies. These issues were first explored in [22].

In the following examples, a solid arrow from a node  $c_1$  to another node  $c_2$  means that  $c_1$  is a subclass of  $c_2$ . All examples in this section are discussed informally. The formal treatment will be given in Sections 3 and 4.

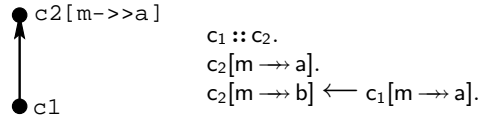
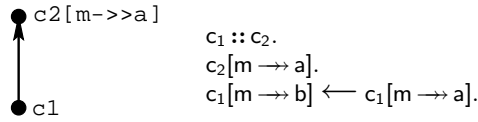


Fig. 1. Inheritance through Context

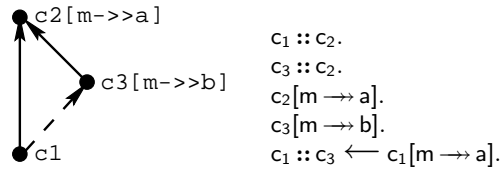
Consider the program in Figure 1. Without inheritance, this program has a unique model, which consists of the first two facts. According to the common intuition behind inheritance,  $c_1$  ought to inherit  $m \rightarrow a$  from  $c_2$ . However, just adding the fact  $c_1[m \rightarrow a]$  will not make the resulting set a model, since the last rule is no longer satisfied: The least model that contains the inherited fact should also include  $c_2[m \rightarrow b]$ . However, this begs the question as to whether  $c_1$  should inherit  $m \rightarrow b$  from  $c_2$  as well. The intuition suggests that the intended model should be “stable” with respect to not only deduction but inheritance as well. Therefore  $c_1[m \rightarrow b]$  also should be in that model. This problem was recognized in [22], but the proposed solution was not stable in the above sense, because it was not based on semantic principles but rather on an ad hoc definition of a plausible fixpoint computation.

Now consider the program in Figure 2, which is the same as the program in Figure 1 except for the head of the last rule. Again, the intuition suggests that  $c_1[m \rightarrow a]$  ought to be inherited, and  $c_1[m \rightarrow b]$  be derived to make the resulting set of facts into a model in the conventional sense. This, however, leads to the



**Fig. 2.** Interaction between Derived and Inherited Facts

following observation. The method  $m$  of  $c_1$  now has one value,  $a$ , which is inherited, and another value,  $b$ , which is derived via a rule. Although the traditional frameworks for inheritance were developed without deduction in mind, it is clear that derived facts like  $c_1[m \rightarrow b]$  are akin to “local” method definitions and so should be treated similarly. In particular, local definitions always override inheritance. The conclusion is that although derivation is done “after” inheritance, its existence undermines the original reason for inheritance. This is similar to the known phenomenon where a reasoner rejects an assumption when it leads to the derivation of a contradiction. Again, the framework presented in this paper, which is based on semantic principles, differs from the ad hoc computation in [22] (which keeps both derived and inherited facts).



**Fig. 3.** Interaction between Inheritance and Derived Intervening Superclass

The example in Figure 3 shows a case where inheritance changes the class hierarchy, which creates conditions that undermine the original reason for inheritance. Initially,  $c_3$  is not known to be a superclass of  $c_1$ . So, it seems that  $c_1$  can inherit  $m \rightarrow a$  from  $c_2$ . However, this makes the fact  $c_1[m \rightarrow a]$  true, which in turn causes  $c_1 \text{ :: } c_3$  to be derived by the last rule of the program. Since this makes  $c_3$  a more specific superclass of  $c_1$  than  $c_2$ , it appears that  $c_1$  ought to inherit  $m \rightarrow b$  from  $c_3$  rather than  $m \rightarrow a$  from  $c_2$ . However, this would make the fact  $c_1 \text{ :: } c_3$  unsupported. Either way, the deductive inference enabled by the original inheritance undermines the support for the inheritance itself. Unlike [22], a logically correct solution in this case would be to leave both  $c_1 \text{ :: } c_3$  and  $c_1[m \rightarrow a]$  undefined. The dashed arrow from  $c_1$  to  $c_3$  represents the undefinedness of  $c_1 \text{ :: } c_3$ .

### 2.3 Three-Valued Semantics

The above examples show that inheritance candidacy can be invalidated by a subsequent derivation, which suggests the use of the stable model semantics [14] or the well-founded semantics [13]. In this paper we adopt the latter. Since well-founded models are three-valued and the original F-logic models were two-valued [22], we define a suitable three-valued semantics for F-logic programs first.

The Herbrand universe  $\mathcal{HU}_P$  of an F-logic program  $P$  consists of all the *ground* (i.e., variable-free) terms constructed using the function symbols and constants found in the program. The *Herbrand instantiation* of an F-logic program  $P$ , denoted  $ground(P)$ , is the set of rules obtained by consistently substituting all the terms in  $\mathcal{HU}_P$  for all variables in every rule of  $P$ . Although the program  $P$  is finite, its Herbrand instantiation may well be infinite. In this paper, we consider only the Herbrand instantiation of a program.

The Herbrand base  $\mathcal{HB}_P$  of an F-logic program  $P$  consists of two sorts of atoms:  $s::c$  and  $s[m \rightarrow v]/c$ , where  $s$ ,  $m$ ,  $v$ , and  $c$  are terms from  $\mathcal{HU}_P$ . A *three-valued* interpretation  $\mathcal{I}$  of an F-logic program  $P$  is a pair  $\langle T; U \rangle$ , where  $T$  and  $U$  are *disjoint* subsets of the Herbrand base  $\mathcal{HB}_P$  of  $P$ . The set  $T$  contains all atoms that are *true* in  $\mathcal{I}$  and  $U$  contains all atoms that are *undefined* in  $\mathcal{I}$ . The set  $F$  of all atoms that are *false* in  $\mathcal{I}$  is defined as  $F = \mathcal{HB}_P - (T \cup U)$ . A three-valued interpretation  $\mathcal{I} = \langle T; U \rangle$  is called *two-valued* if  $U = \emptyset$ .

Following [26], we define the valuation functions for atoms, literals, and rules. The atoms in  $\mathcal{HB}_P$  can take one of the three values:  $\mathbf{t}$ ,  $\mathbf{f}$ , and  $\mathbf{u}$ . The ordering among truth values is as follows:  $\mathbf{f} < \mathbf{u} < \mathbf{t}$ . Given an interpretation  $\mathcal{I} = \langle T; U \rangle$  of an F-logic program  $P$ , for any atom  $A$  from  $\mathcal{HB}_P$  we can define a truth valuation function  $\mathcal{I}$  as follows: (i)  $\mathcal{I}(A) = \mathbf{t}$ , if  $A \in T$ ; (ii)  $\mathcal{I}(A) = \mathbf{u}$ , if  $A \in U$ ; (iii) Otherwise,  $\mathcal{I}(A) = \mathbf{f}$ . Moreover, for any  $A_i \in \mathcal{HB}_P$ ,  $1 \leq i \leq n$ :  $\mathcal{I}(A_1 \wedge \dots \wedge A_n) = \min\{\mathcal{I}(A_i) \mid 1 \leq i \leq n\}$ .

We can extend the truth valuation function  $\mathcal{I}$  to all rules in the Herbrand instantiation of  $P$ ,  $ground(P)$ . The intuitive reading of a rule is as follows: the head of the rule functions as a *local definition* while the body of the rule functions as a *query*. In particular, if  $s[m \rightarrow v]$  is in the head of a rule and the body of the rule is satisfied, it means that  $m \rightarrow v$  is *locally defined* for  $s$ . If  $s[m \rightarrow v]$  appears in the body of a rule, it is a query that tests whether  $s$  has a local definition of  $m \rightarrow v$ , or  $s$  inherits  $m \rightarrow v$  from some superclass. In an interpretation of an F-logic program, atoms of the form  $s[m \rightarrow v]/s$  capture the idea that  $m \rightarrow v$  is locally defined at  $s$ , and atoms of the form  $s[m \rightarrow v]/c$ , where  $s \neq c$ , capture the idea that  $s$  inherits  $m \rightarrow v$  from  $c$ . Therefore, the truth valuation of an F-logic literal may be different depending on whether it appears in a rule head or in a rule body. The following definitions make the above discussion precise.

**Definition 1.** *Given an interpretation  $\mathcal{I}$  of an F-logic program  $P$ , the truth valuation functions  $\mathcal{V}_{\mathcal{I}}^h$  and  $\mathcal{V}_{\mathcal{I}}^b$  ( $\mathbf{h}$  stands for head and  $\mathbf{b}$  for body) on ground F-logic literals are defined as follows:*

$$\begin{aligned} \mathcal{V}_{\mathcal{I}}^h(s::c) &= \mathcal{I}(s::c) & \mathcal{V}_{\mathcal{I}}^h(c[m \rightarrow v]) &= \mathcal{I}(c[m \rightarrow v]/c) \\ \mathcal{V}_{\mathcal{I}}^b(s::c) &= \mathcal{I}(s::c) & \mathcal{V}_{\mathcal{I}}^b(s[m \rightarrow v]) &= \max\{\mathcal{I}(s[m \rightarrow v]/c) \mid c \in \mathcal{HU}_P\} \end{aligned}$$

Let  $L$  and  $L_i$  ( $1 \leq i \leq n$ ) be variable-free literals. Then

$$\mathcal{V}_{\mathcal{I}}^b(\neg L) = \neg \mathcal{V}_{\mathcal{I}}^b(L) \quad \mathcal{V}_{\mathcal{I}}^b(L_1 \wedge \dots \wedge L_n) = \min\{\mathcal{V}_{\mathcal{I}}^b(L_i) \mid 1 \leq i \leq n\}$$

where  $\neg \mathbf{f} = \mathbf{t}$ ,  $\neg \mathbf{u} = \mathbf{u}$ , and  $\neg \mathbf{t} = \mathbf{f}$ .

**Definition 2.** Given an interpretation  $\mathcal{I}$  of an F-logic program  $P$ , the truth valuation function  $\mathcal{I}$  on a rule,  $H \leftarrow B$ , in  $\text{ground}(P)$ , is defined as follows:

$$\mathcal{I}(H \leftarrow B) = \begin{cases} \mathbf{t}, & \text{if } \mathcal{V}_{\mathcal{I}}^{\mathbf{h}}(H) \geq \mathcal{V}_{\mathcal{I}}^{\mathbf{b}}(B); \\ \mathbf{f}, & \text{otherwise.} \end{cases}$$

The truth valuation function  $\mathcal{I}$  on a fact,  $H \in \text{ground}(P)$ , is defined as follows:

$$\mathcal{I}(H) = \begin{cases} \mathbf{t}, & \text{if } \mathcal{V}_{\mathcal{I}}^{\mathbf{h}}(H) = \mathbf{t}; \\ \mathbf{f}, & \text{otherwise.} \end{cases}$$

**Definition 3 (Program Satisfaction).** A three-valued interpretation  $\mathcal{I}$  satisfies an F-logic program  $P$ , if for every rule  $R$  in  $\text{ground}(P)$ ,  $\mathcal{I}(R) = \mathbf{t}$ .

### 3 Nonmonotonic Multiple Inheritance

Program satisfaction, as in Definition 3, does not necessarily mean that an interpretation  $\mathcal{I}$  is an intended *object model* of an F-logic program  $P$ , because  $\mathcal{I}$  must also include facts that are derived by inheritance. An F-logic program specifies only the class hierarchy and method definitions — what needs to be inherited is not explicitly stated. In fact, as we saw in Section 2.2, defining exactly what should be inherited is a subtle issue. In our framework, it is the job of the *inheritance postulates*, which embody the common intuition behind nonmonotonic multiple inheritance. The purpose of this section is to develop these postulates and the associated notion of an object model.

Note that although the intuition behind these postulates is commonly accepted, these postulates have never been formalized before in a general framework where inheritance and deduction coexist. Our contribution is in showing that these postulates can form a foundation for a semantics that is both mathematically robust and intuitively satisfactory.

Intuitively,  $c[m \rightarrow v]$  is an *inheritance context* for  $s$ , which has no local definition for the method  $m$ , if  $c$  is a superclass (but not necessarily an immediate superclass) of  $s$ , and  $m \rightarrow v$  is locally defined at  $c$ , *i.e.*,  $c[m \rightarrow v]$  is defined as a fact or derived via a program rule. Inheritance context is necessary for inheritance to take place, but is not sufficient. Indeed, inheritance of  $m \rightarrow v$  from  $c$  might be overridden by a more specific inheritance context that sits below  $c$  along the inheritance path. If an inheritance context is not overridden by any other inheritance context, then we call it an *inheritance candidate*. Intuitively, inheritance candidates represent potential sources for inheritance. But there must be exactly one inheritance candidate for inheritance to take place — having more just leads to a multiple inheritance conflict.

The various concepts defined below come in with two flavors: *strong* or *weak*. The “strong” flavor of a concept requires that all relevant facts must be positively established; the “weak” flavor allows some or all facts to be undefined.

**Definition 4 (Local Context).** Given an interpretation  $\mathcal{I}$ ,  $s[m \rightarrow v]$  is a strong local context for  $s$  in  $\mathcal{I}$ , if  $\mathcal{I}(s[m \rightarrow v]_s) = \mathbf{t}$ . Similarly,  $s[m \rightarrow v]$  is a weak local context for  $s$  in  $\mathcal{I}$  if  $\mathcal{I}(s[m \rightarrow v]_s) = \mathbf{u}$ .

**Definition 5 (Inheritance Context).** Given an interpretation  $\mathcal{I}$ ,  $c[m \rightarrow v]$  is a strong inheritance context for  $s$  in  $\mathcal{I}$ , if  $c \neq s$ <sup>7</sup>,  $\mathcal{I}(s :: c \wedge c[m \rightarrow v]/c) = \mathbf{t}$ , and  $\mathcal{I}(s[m \rightarrow x]/s) = \mathbf{f}$  for all  $x$ . (i.e.,  $s$  is a proper subclass of  $c$ ,  $m \rightarrow v$  is locally defined at  $c$ , and  $s$  has neither strong nor weak local context on the method  $m$ ). Similarly,  $c[m \rightarrow v]$  is a weak inheritance context for  $s$  in  $\mathcal{I}$  if  $c \neq s$ ,  $\mathcal{I}(s :: c \wedge c[m \rightarrow v]/c) = \mathbf{u}$ , and  $\mathcal{I}(s[m \rightarrow x]/s) \neq \mathbf{t}$  for all  $x$  (i.e.,  $s$  has no strong local context on the method  $m$ ).

**Definition 6 (Overriding).** Given an interpretation  $\mathcal{I}$ , the class  $o$  strongly overrides  $c[m \rightarrow v]$  for  $s$  in  $\mathcal{I}$ , if  $o \neq c$ ,  $\mathcal{I}(o :: c) = \mathbf{t}$ , and there is  $x$  such that  $o[m \rightarrow x]$  is a strong inheritance context for  $s$ .

The class  $o$  weakly overrides  $c[m \rightarrow v]$  for  $s$  in  $\mathcal{I}$  if the above conditions are relaxed by allowing  $o :: c$  to be undefined and/or allowing  $o[m \rightarrow x]$  to be a weak context. Formally this means that either

- (1)  $\mathcal{I}(o :: c) = \mathbf{t}$  and there is  $x$  such that  $o[m \rightarrow x]$  is a weak inheritance context for  $s$ ; or
- (2)  $\mathcal{I}(o :: c) = \mathbf{u}$  and there is  $x$  such that  $o[m \rightarrow x]$  is either a weak or a strong inheritance context for  $s$ .

**Definition 7 (Inheritance Candidate).** Given an interpretation  $\mathcal{I}$ ,  $c[m \rightarrow v]$  is a strong inheritance candidate for  $s$  in  $\mathcal{I}$ , denoted  $c[m \rightarrow v] \overset{s}{\rightsquigarrow}_{\mathcal{I}} s$ , if  $c[m \rightarrow v]$  is a strong inheritance context for  $s$ , and there is no  $o$  that strongly or weakly overrides  $c[m \rightarrow v]$  for  $s$ .

$c[m \rightarrow v]$  is a weak inheritance candidate for  $s$  in  $\mathcal{I}$ , denoted  $c[m \rightarrow v] \overset{w}{\rightsquigarrow}_{\mathcal{I}} s$ , if the above conditions are relaxed by allowing  $c[m \rightarrow v]$  to be a weak inheritance context and/or allowing weak overriding. Formally, this means that there is no  $o$  that strongly overrides  $c[m \rightarrow v]$  for  $s$ , and either

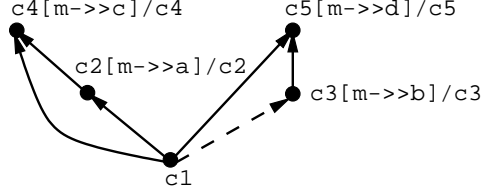
- (1)  $c[m \rightarrow v]$  is a weak inheritance context for  $s$ ; or
- (2)  $c[m \rightarrow v]$  is a strong inheritance context for  $s$  and there is  $o$  that weakly overrides  $c[m \rightarrow v]$  for  $s$ .

As an example, consider an interpretation  $\mathcal{I} = \langle T; U \rangle$ , where  $T = \{c_1 :: c_2, c_1 :: c_4, c_1 :: c_5, c_2 :: c_4, c_3 :: c_5, c_2[m \rightarrow a]/c_2, c_3[m \rightarrow b]/c_3, c_4[m \rightarrow c]/c_4, c_5[m \rightarrow d]/c_5\}$  and  $U = \{c_1 :: c_3\}$ .  $\mathcal{I}$  is shown in Figure 4, where solid and dashed arrows represent true and undefined subclass relationships, respectively.

In  $\mathcal{I}$ ,  $c_2[m \rightarrow a]$ ,  $c_4[m \rightarrow c]$ , and  $c_5[m \rightarrow d]$  are strong inheritance contexts for  $c_1$ . On the other hand,  $c_3[m \rightarrow b]$  is a weak inheritance context for  $c_1$ . The class  $c_2$  strongly overrides  $c_4[m \rightarrow c]$ , while  $c_3$  weakly overrides  $c_5[m \rightarrow d]$ . The context  $c_2[m \rightarrow a]$  is a strong inheritance candidate for  $c_1$ , while  $c_3[m \rightarrow b]$  and  $c_5[m \rightarrow d]$  are both weak inheritance candidates for  $c_1$ , and  $c_4[m \rightarrow c]$  is neither a strong nor a weak inheritance candidate for  $c_1$ .

For convenience, we use  $c[m \rightarrow v] \rightsquigarrow_{\mathcal{I}} s$  in situations where it does not matter whether  $c[m \rightarrow v]$  is a strong or a weak inheritance candidate. Now we are ready to introduce our postulates for nonmonotonic multiple inheritance.

<sup>7</sup>  $c \neq s$  means that  $c$  and  $s$  are distinct terms.



**Fig. 4.** Inheritance Context, Overriding, and Inheritance Candidate

**Definition 8 (Positive ISA Transitivity).** An interpretation  $\mathcal{I}$  satisfies the positive ISA transitivity constraint if the positive part of the class hierarchy is transitively closed, formally, for all  $s, c$ : if there is  $o$  such that  $\mathcal{I}(s::o) = \mathbf{t} \wedge \mathcal{I}(o::c) = \mathbf{t}$ , then  $\mathcal{I}(s::c) = \mathbf{t}$ .

**Definition 9 (Context Consistency).** An interpretation  $\mathcal{I}$  satisfies the context consistency constraint, if the following two conditions hold:

- (1) for all  $c, m, v$ : if  $\mathcal{I}(c[m \rightarrow v]/c) = \mathbf{f}$ , then  $\mathcal{I}(s[m \rightarrow v]/c) = \mathbf{f}$  for all  $s$ ;
- (2) for all  $s, m$ : if there is  $v$  such that  $\mathcal{I}(s[m \rightarrow v]/s) = \mathbf{t}$ , then  $\mathcal{I}(s[m \rightarrow x]/c) = \mathbf{f}$  for all  $x, c$ , such that  $c \neq s$ ,

The context consistency constraint captures the implications of locality and specificity. The first condition states that if  $m \rightarrow v$  is not locally defined for  $c$ , then no class should inherit  $m \rightarrow v$  from  $c$ . The second condition states that if  $m \rightarrow v$  is locally defined for  $s$ , then this definition should prevent  $s$  from inheriting any information about  $m$  from other classes.

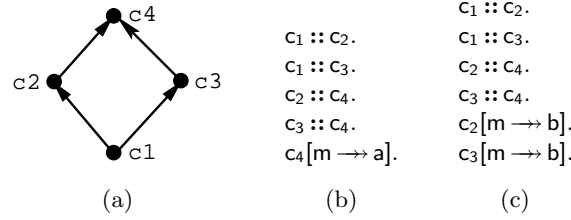
The following constraint captures the meaning of nonmonotonic multiple inheritance. Intuitively, we want our semantics for inheritance to have the property that if inheritance is allowed, then it should take place from a *unique* source.

**Definition 10 (Unique Source Inheritance).** An interpretation  $\mathcal{I}$  satisfies the unique source inheritance constraint, if the following two conditions hold:

- (1) for all  $c, s, m$ , and  $v$ : if  $c[m \rightarrow v] \xrightarrow{s} \mathcal{I} s$ , then  $\mathcal{I}(s[m \rightarrow x]/o) = \mathbf{f}$  for all  $x, o$ , such that  $o \neq c$ ;
- (2) for all  $m, v, c$ , and  $s$ , such that  $s \neq c$ :  $\mathcal{I}(s[m \rightarrow v]/c) = \mathbf{t}$  iff
  - (i)  $c[m \rightarrow v] \xrightarrow{s} \mathcal{I} s$ ; and
  - (ii) there are no  $o, y$ , such that  $o \neq c$  and  $o[m \rightarrow y] \xrightarrow{\mathcal{I}} s$ .

The first condition above states that if a strong inheritance candidate,  $c$ , exists then inheritance cannot take place from any other source (because this would be a multiple inheritance conflict). However, inheritance might take place from  $c$ , if there are no other inheritance candidates. The second condition specifies when “definite” inheritance takes place:  $s$  *must* inherit  $m \rightarrow v$  from  $c$  if and only if: (i)  $c[m \rightarrow v]$  is a strong inheritance candidate for  $s$ ; and (ii) there are no other inheritance candidates — weak or strong — from which  $s$  could inherit the method  $m$ .

**Definition 11 (Object Model).** An interpretation  $\mathcal{I}$  of an  $F$ -logic program  $P$  is called an object model of  $P$  if  $\mathcal{I}$  satisfies  $P$  plus the three postulates defined in this section: the positive ISA transitivity constraint, the context consistency constraint, and the unique source inheritance constraint.



**Fig. 5.** Unique Source Inheritance

Consider the two programs in Figure 5(b) and Figure 5(c) which share the same class hierarchy as shown in Figure 5(a). Let  $C = \{c_1 :: c_2, c_1 :: c_3, c_2 :: c_4, c_3 :: c_4\}$  and  $\mathcal{I}_1 = \langle T_1; U_1 \rangle$  be an interpretation for the program in Figure 5(b) where  $T_1 = C \cup \{c_4[m \rightarrow a]/c_4, c_2[m \rightarrow a]/c_4, c_3[m \rightarrow a]/c_4, c_1[m \rightarrow a]/c_4\}$  and  $U_1 = \emptyset$ . One can verify that  $\mathcal{I}_1$  is an object model for the program in Figure 5(b). From  $\mathcal{I}_1$  we can see that  $c_4[m \rightarrow a]$  is the *unique* strong inheritance candidate for  $c_2, c_3,$  and  $c_4$ , which all inherit  $m \rightarrow a$  from  $c_4$ .

Let  $C$  be the same set of ISA atoms as before and consider the interpretation  $\mathcal{I}_2 = \langle T_2; U_2 \rangle$  for the program in Figure 5(c) where  $T_2 = C \cup \{c_2[m \rightarrow b]/c_2, c_3[m \rightarrow b]/c_3, c_1[m \rightarrow b]/c_2, c_1[m \rightarrow b]/c_3\}$  and  $U_2 = \emptyset$ . Clearly,  $\mathcal{I}_2$  satisfies the program in Figure 5(c). But it is not an object model — the presence of each one of  $c_1[m \rightarrow b]/c_2$  and  $c_1[m \rightarrow b]/c_3$  in  $\mathcal{I}_2$  violates the first condition of the unique source inheritance constraint, because both  $c_2[m \rightarrow b]$  and  $c_3[m \rightarrow b]$  are strong inheritance candidates for  $c_1$  and  $c_1 \neq c_2 \neq c_3$ .

It is worth pointing out the difference between *source-based* and *value-based* approaches to multiple inheritance. Suppose  $c_2[m \rightarrow x]$  and  $c_3[m \rightarrow y]$  are both strong inheritance candidates for  $c_1$ , where  $c_1 \neq c_2 \neq c_3$ . In the source-based approach  $c_1$  has a multiple inheritance conflict on the method  $m$  regardless of the values of  $x$  and  $y$ . On the contrary, in the value-based approach, no conflict would occur if  $m$  returns the *same* set of values in both classes  $c_2$  and  $c_3$ . For instance, the above interpretation  $\mathcal{I}_2$  for the program in Figure 5(c) would be an object model under the value-based approach, since  $m$  returns the same set of values,  $\{b\}$ , in  $c_2$  and  $c_3$ . However, value-based nonmonotonic multiple inheritance requires higher-order reasoning and is expensive to compute. In this paper we consider only source-based inheritance.

The constraints introduced so far capture the intuition behind the “definite” part of an object model, *i.e.*, the true and the false components. We view them as *core postulates* that any reasonable object model must obey. However, we still need to assign a meaning to the undefined part of an object model. Since “unde-

“fined” means possibly true or possibly false, we intuitively want the conclusions drawn from undefined facts to remain undefined. In other words, the semantics should be “closed” with regard to undefined facts. Thus, although it might seem tempting to “jump” to negative conclusions from undefined facts in some cases (e.g., if there are multiple weak inheritance candidates), our semantics is biased towards undefined conclusions, which is why we call it “optimistic.”

**Definition 12 (Optimistic ISA Transitivity).** *An interpretation  $\mathcal{I}$  satisfies the optimistic ISA transitivity constraint if the undefined part of the class hierarchy is transitively closed, formally, for all  $s, c$ : if there is  $o$  such that  $\mathcal{I}(s::o \wedge o::c) = \mathbf{u}$  and  $\mathcal{I}(s::c) \neq \mathbf{t}$ , then  $\mathcal{I}(s::c) = \mathbf{u}$ .*

**Definition 13 (Optimistic Inheritance).** *An interpretation  $\mathcal{I}$  satisfies the optimistic inheritance constraint if the following condition is true: for all  $m, v, c$ , and  $s$ , such that  $s \neq c$ :  $\mathcal{I}(s[m \rightarrow v]/c) = \mathbf{u}$  iff*

- (i)  $c[m \rightarrow v] \rightsquigarrow_{\mathcal{I}} s$ ; and
- (ii) there are no  $o, y$ , such that  $o \neq c$  and  $o[m \rightarrow y] \overset{s}{\rightsquigarrow}_{\mathcal{I}} s$ ; and
- (iii)  $\mathcal{I}(s[m \rightarrow v]/c) \neq \mathbf{t}$ .

The optimistic inheritance constraint captures the intuition behind multiple inheritance based on undefined knowledge. Indeed,  $s$  *optimistically* inherits  $m \rightarrow v$  from  $c$  if and only if: (i)  $c[m \rightarrow v]$  is either a strong or a weak inheritance candidate for  $s$ ; (ii) there are no other strong inheritance candidates that can *invalidate* inheritance from  $c$  (by the unique source inheritance constraint); and (iii)  $s$  cannot *positively* inherit  $m \rightarrow v$  from  $c$ .

## 4 Optimistic Object Model

In this section we define a particular object model, called *optimistic object model*, which exists for *any* F-logic program and has certain desirable properties. First, it satisfies all the postulates of the previous section, including the two “optimistic” postulates at the end. Second, it has several independent characterizations. This section presents a procedural characterization, which can be used to compute such models bottom-up.

First we need to extend the definition of an interpretation in Section 2 to include book-keeping information used by the computation. The book-keeping information is projected out when the final model is produced. The *extended Herbrand base* of an F-logic program  $P$ , denoted  $\widehat{\mathcal{HB}}_P$ , consists of atoms from  $\mathcal{HB}_P$  and *auxiliary* atoms of the form  $c[m \rightarrow v] \rightsquigarrow s$ , where  $c, m, v$ , and  $s$  are terms from  $\mathcal{HU}_P$ . During the computation, we use the auxiliary atoms to approximate the inheritance candidates. An *extended atom set* is a subset of  $\widehat{\mathcal{HB}}_P$ . In the sequel, we will use symbols with a hat (i.e.,  $\widehat{\mathcal{I}}$ ) to denote extended atom sets. The *projection* of an extended atom set  $\widehat{\mathcal{I}}$ , denoted  $\pi(\widehat{\mathcal{I}})$ , is  $\widehat{\mathcal{I}}$  with the auxiliary atoms removed.

It is easy to extend the definitions of the valuation functions in Section 2 to extended atom sets, since the auxiliary atoms do not occur in F-logic programs. Formally, given an extended atom set  $\widehat{\mathcal{I}}$ , let  $\mathcal{I} = \langle \pi(\widehat{\mathcal{I}}); \emptyset \rangle$ . We define:

(i)  $val_{\hat{I}}^h(H) \stackrel{\text{def}}{=} \mathcal{V}_{\mathcal{I}}^h(H)$ , for a ground rule head  $H$ ; (ii)  $val_{\hat{I}}^b(B) \stackrel{\text{def}}{=} \mathcal{V}_{\mathcal{I}}^b(B)$ , for a ground rule body  $B$ ; and (iii)  $val_{\hat{I}}(R) \stackrel{\text{def}}{=} \mathcal{I}(R)$ , for a ground rule  $R$ .

The computation process for the optimistic object model, defined below, extends the alternating fixpoint computation in [12]. The new element here is the book-keeping mechanism for recording the inheritance information.

**Definition 14.** *Given a ground literal  $L$  of an  $F$ -logic program  $P$  and an atom  $A \in \mathcal{HB}_P$ , we say that  $L$  matches  $A$ , if either  $L = s::c$  and  $A = s::c$ ; or  $L = s[m \rightarrow v]/_s$  and  $A = s[m \rightarrow v]/_s$ .*

**Definition 15 (Rule Consequence).** *The rule consequence operator,  $\mathbf{RC}_{P, \hat{I}}$ , is defined for an  $F$ -logic program  $P$  and an extended atom set  $\hat{I}$ . It takes as input an extended atom set,  $\hat{J}$ , and generates a new extended atom set,  $\mathbf{RC}_{P, \hat{I}}(\hat{J})$ , as follows:*

$$\left\{ A \left| \begin{array}{l} \text{There is } H \leftarrow L_1, \dots, L_n \text{ in } \text{ground}(P), \text{ such that } H \text{ matches } A, \\ \text{and for every literal } L_i \text{ (} 1 \leq i \leq n \text{): (i) if } L_i \text{ is positive, then} \\ \text{val}_{\hat{J}}^b(L_i) = \mathbf{t}; \text{ and (ii) if } L_i \text{ is negative, then } \text{val}_{\hat{J}}^b(L_i) = \mathbf{t}. \end{array} \right. \right\}$$

This operator is adopted from the usual alternating fixpoint computation; it derives new facts from the rules in the program.

**Definition 16 (Inheritance Blocking).** *The inheritance blocking operator,  $\mathbf{IB}$ , takes as input an extended atom set,  $\hat{I}$ , and generates the set,  $\mathbf{IB}(\hat{I})$ , which is the union of the following three sets of atoms:*

$$\begin{aligned} & \left\{ lc(s, m) \mid \exists v, \text{ such that } s[m \rightarrow v]/_s \in \hat{I} \right\} \\ & \left\{ mc(c, s, m) \mid \exists o, v, \text{ such that } o \neq c \text{ and } o[m \rightarrow v] \rightsquigarrow s \in \hat{I} \right\} \\ & \left\{ ov(c, s, m) \mid \exists o, v, \text{ such that } o \neq c, o \neq s, \text{ and } s::o, o::c, o[m \rightarrow v]/_o \in \hat{I} \right\} \end{aligned}$$

This is an auxiliary operator used in defining the inheritance consequence operator below. It returns the book-keeping information that is needed in deciding which facts can be inherited and which ones are the inheritance candidates. Intuitively,  $lc(s, m)$  means that the method  $m$  is locally defined at  $s$ ;  $mc(c, s, m)$  means that inheritance of method  $m$  from  $c$  to  $s$  is not possible due to a multiple inheritance conflict (as manifested by the existence of an inheritance candidate at  $o$ );  $ov(c, s, m)$  means that any inheritance of the method  $m$  from  $c$  to  $s$  would be overridden by another class ( $o$ ) that lies between  $s$  and  $c$  in the class hierarchy.

**Definition 17 (Inheritance Consequence).** *The inheritance consequence operator,  $\mathbf{IC}_{\hat{I}}$ , where  $\hat{I}$  is an extended atom set, takes as input an extended atom set,  $\hat{J}$ , and generates a new extended atom set as follows:*

$$\mathbf{IC}_{\hat{I}}(\hat{J}) \stackrel{\text{def}}{=} \mathbf{IC}^s(\hat{J}) \cup \mathbf{IC}_{\hat{I}}^c(\hat{J}) \cup \mathbf{IC}_{\hat{I}}^i(\hat{J})$$

$$\begin{aligned}
\mathbf{IC}^s(\widehat{\mathcal{J}}) &= \left\{ s :: c \mid \exists o, \text{ such that } s :: o, o :: c \in \widehat{\mathcal{J}} \right\} \\
\mathbf{IC}_{\widehat{\mathcal{I}}}^c(\widehat{\mathcal{J}}) &= \left\{ c[m \rightarrow v] \rightsquigarrow s \mid \begin{array}{l} s :: c \in \widehat{\mathcal{J}}, c \neq s, c[m \rightarrow v]/c \in \widehat{\mathcal{J}}, \\ lc(s, m) \notin \mathbf{IB}(\widehat{\mathcal{I}}), \text{ and } ov(c, s, m) \notin \mathbf{IB}(\widehat{\mathcal{I}}) \end{array} \right\} \\
\mathbf{IC}_{\widehat{\mathcal{I}}}^i(\widehat{\mathcal{J}}) &= \left\{ s[m \rightarrow v]/c \mid c[m \rightarrow v] \rightsquigarrow s \in \widehat{\mathcal{J}}, \text{ and } mc(c, s, m) \notin \mathbf{IB}(\widehat{\mathcal{I}}) \right\}
\end{aligned}$$

At each step this operator derives newly inherited facts as well as inheritance candidates.

**Definition 18 (Program Completion).** *The program completion operator,  $\mathbf{T}_{P, \widehat{\mathcal{I}}}$ , where  $P$  is an F-logic program and  $\widehat{\mathcal{I}}$  an extended atom set, takes as input an extended atom set,  $\widehat{\mathcal{J}}$ , and generates a new extended atom set as follows:  $\mathbf{T}_{P, \widehat{\mathcal{I}}}(\widehat{\mathcal{J}}) \stackrel{\text{def}}{=} \mathbf{RC}_{P, \widehat{\mathcal{I}}}(\widehat{\mathcal{J}}) \cup \mathbf{IC}_{\widehat{\mathcal{I}}}(\widehat{\mathcal{J}})$ .*

In other words,  $\mathbf{T}_{P, \widehat{\mathcal{I}}}$  derives new “local” facts (via the rules), inherited facts, plus inheritance candidacy information that is used to decide which facts to inherit in the future.

**Lemma 1.**  *$\mathbf{RC}_{P, \widehat{\mathcal{I}}}$  is monotonic when  $P$  and  $\widehat{\mathcal{I}}$  are fixed.  $\mathbf{IB}$  is monotonic.  $\mathbf{IC}^s$  is monotonic.  $\mathbf{IC}_{\widehat{\mathcal{I}}}^c$ ,  $\mathbf{IC}_{\widehat{\mathcal{I}}}^i$ , and  $\mathbf{IC}_{\widehat{\mathcal{I}}}$  are monotonic when  $\widehat{\mathcal{I}}$  is fixed.  $\mathbf{T}_{P, \widehat{\mathcal{I}}}$  is monotonic when  $P$  and  $\widehat{\mathcal{I}}$  are fixed.*

Let  $P$  be an F-logic program. The set of all subsets of the extended Herbrand base  $\widehat{\mathcal{HB}}_P$  constitutes a complete lattice where the partial ordering is defined by set inclusion. Therefore, any monotonic operator,  $\Phi$ , defined on this lattice has a unique least fixpoint  $\text{lfp}(\Phi)$  [23].

**Definition 19 (Alternating Fixpoint).** *The alternating fixpoint operator,  $\Psi_P$ , for an F-logic program  $P$  takes as input an extended atom set,  $\widehat{\mathcal{I}}$ , and generates a new extended atom set as follows:  $\Psi_P(\widehat{\mathcal{I}}) \stackrel{\text{def}}{=} \text{lfp}(\mathbf{T}_{P, \widehat{\mathcal{I}}})$ .*

**Definition 20 (F-logic Fixpoint).** *The F-logic fixpoint operator,  $\mathbf{F}_P$ , where  $P$  is an F-logic program, takes as input an extended atom set,  $\widehat{\mathcal{I}}$ , and generates a new extended atom set as follows:  $\mathbf{F}_P(\widehat{\mathcal{I}}) \stackrel{\text{def}}{=} \Psi_P(\Psi_P(\widehat{\mathcal{I}}))$ .*

**Lemma 2.** *When  $P$  is fixed,  $\Psi_P$  is antimonotonic and  $\mathbf{F}_P$  is monotonic.*

**Definition 21 (Optimistic Object Model).** *The optimistic object model,  $\mathcal{M}$ , of an F-logic program  $P$  is defined as follows:  $\mathcal{M} = \langle \mathbf{T}; \mathbf{U} \rangle$ , where  $\mathbf{T} = \pi(\text{lfp}(\mathbf{F}_P))$  and  $\mathbf{U} = \pi(\Psi_P(\text{lfp}(\mathbf{F}_P))) - \pi(\text{lfp}(\mathbf{F}_P))$ . Here  $\pi$  is the projection operator defined earlier. It removes the auxiliary atoms of the form  $c[m \rightarrow v] \rightsquigarrow s$ , which are used for book-keeping during computation.*

We illustrate the computation of optimistic object models using the following example. Consider the F-logic program  $P$  in Figure 3. Let  $\mathbf{T} = \{c_1 :: c_2, c_3 :: c_2, c_2[m \rightarrow a]/c_2, c_3[m \rightarrow b]/c_3\}$  and  $\mathbf{U} = \{c_1 :: c_3, c_1[m \rightarrow a]/c_2, c_1[m \rightarrow b]/c_3\}$ . Then we have  $\text{lfp}(\mathbf{F}_P) = \mathbf{T}$  and  $\Psi_P(\text{lfp}(\mathbf{F}_P)) = \mathbf{T} \cup \mathbf{U} \cup \{c_2[m \rightarrow a] \rightsquigarrow c_1, c_3[m \rightarrow b] \rightsquigarrow c_1\}$ . So  $\langle \mathbf{T}; \mathbf{U} \rangle$  is the optimistic object model for the program in Figure 3.

**Theorem 1.** *The optimistic object model  $\mathcal{M}$  of an F-logic program  $P$  is an object model of  $P$ . In addition, this model satisfies the optimistic ISA transitivity constraint, and the optimistic inheritance constraint.*

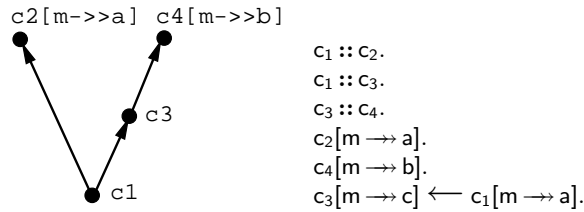
## 5 Minimal Object Model

Since the introduction of the Closed World Assumption [28], comparing different models of a program based on the amount of “truth” contained in those models has become a common technique. Typically, the true component of a model is minimized and the false component is maximized. However, in F-logic we also deal with inheritance, which complicates the matters somewhat, because the truth value of a fact may depend on inheritance. This can create object models that look similar but actually are incomparable. This issue is illustrated by an example that follows the definition of minimality below. The solution is to minimize not only the set of true atoms of an object model, but also the amount of positive inheritance information implied by the object model.

**Definition 22.** *Given two object models,  $\mathcal{I}_1 = \langle P_1; Q_1 \rangle$  and  $\mathcal{I}_2 = \langle P_2; Q_2 \rangle$ , of an F-logic program  $P$ , we write  $\mathcal{I}_1 \leq \mathcal{I}_2$  if and only if  $P_1 \subseteq P_2$ ,  $P_1 \cup Q_1 \subseteq P_2 \cup Q_2$ , and  $c[m \rightarrow v] \overset{s}{\sim}_{\mathcal{I}_1} s$  implies  $c[m \rightarrow v] \overset{s}{\sim}_{\mathcal{I}_2} s$ , for all  $c, m, v, s$ .*

**Definition 23 (Minimal Object Model).** *An object model  $\mathcal{I}$  is minimal iff there exists no object model  $\mathcal{I}'$  such that  $\mathcal{I}' \leq \mathcal{I}$  and  $\mathcal{I}'$  is different from  $\mathcal{I}$ .*

The above definitions minimize the number of strong inheritance candidates implied by an object model *in addition to* the usual minimization of truth and maximization of falsehood. This is needed because increasing the number of false facts might inflate the number of strong inheritance candidates, which in turn might unjustifiably inflate the number of facts that are derived by inheritance.



**Fig. 6.** Minimal Object Model

Consider the program in Figure 6 and the following two object models of the program:  $\mathcal{I}_1 = \langle P_1; Q_1 \rangle$ , where  $P_1 = \{c_1 :: c_2, c_1 :: c_3, c_3 :: c_4, c_2[m \rightarrow a]/c_2, c_4[m \rightarrow b]/c_4\}$ ,  $Q_1 = \emptyset$ , and  $\mathcal{I}_2 = \langle P_2; Q_2 \rangle$ , where  $P_2 = P_1$ ,  $Q_2 = \{c_1[m \rightarrow a]/c_2, c_3[m \rightarrow c]/c_3\}$ .  $\mathcal{I}_1$  and  $\mathcal{I}_2$  both agree on the atoms that are true. But in  $\mathcal{I}_1$  both  $c_1[m \rightarrow a]/c_2$  and  $c_3[m \rightarrow c]/c_3$  are false, whereas in  $\mathcal{I}_2$  they are both undefined.

Clearly,  $\mathcal{I}_1$  carries more false atoms than  $\mathcal{I}_2$  and so with the usual notion of minimality we would say  $\mathcal{I}_1 \leq \mathcal{I}_2$ . However,  $\mathcal{I}_1$  is not as “tight” as it appears, because the additional false atoms in  $\mathcal{I}_1$  are not automatically implied by the program under our optimistic object model semantics. Indeed, although  $c_4[m \rightarrow b]$  is a strong inheritance candidate for  $c_1$  in  $\mathcal{I}_1$ , it is only a weak inheritance candidate in  $\mathcal{I}_2$ . We can see that it is due to this extra positive information about inheritance candidates that  $\mathcal{I}_1$  is able to increase the number of false atoms while keeping the true atoms intact. This anomaly is eliminated by the inheritance minimization built into Definition 22, which renders the two models incomparable, *i.e.*,  $\mathcal{I}_1 \not\leq \mathcal{I}_2$ .

**Theorem 2.** *The optimistic object model  $\mathcal{M}$  of an F-logic program  $P$  is minimal among the object models of  $P$  that satisfy the optimistic ISA transitivity constraint and the optimistic inheritance constraint.*

## 6 Implementation

It turns out that the optimistic object model of an F-logic program  $P$  can be computed as the well-founded model [13, 12] of a certain general logic program with negation. Given an F-logic program  $P$ , we first rewrite  $P$  into a general logic program  $P^{wf}$ . Then we show that the well-founded model of  $P^{wf}$  is isomorphic to the optimistic object model of  $P$ .

**Definition 24.** *Given an F-logic program  $P$ , let  $L$  be a literal in  $P$ . The functions  $\rho^h$  and  $\rho^b$  for rewriting head and body literals in  $P$ , respectively, are defined as follows:*

$$\rho^h(L) = \begin{cases} \text{sub}(s, c), & \text{if } L = s :: c \\ \text{locmvd}(s, m, v), & \text{if } L = s[m \rightarrow v] \end{cases}$$

$$\rho^b(L) = \begin{cases} \text{sub}(s, c), & \text{if } L = s :: c \\ \text{mvd}(s, m, v), & \text{if } L = s[m \rightarrow v] \\ \neg(\rho^b(G)), & \text{if } L = \neg G \end{cases}$$

Let  $R \equiv H \leftarrow L_1, \dots, L_n$ ,  $n \geq 0$ , be a rule in  $P$ . The function  $\rho$  for rewriting the rules in  $P$  is defined as follows:  $\rho(R) = \rho^h(H) \leftarrow \rho^b(L_1), \dots, \rho^b(L_n)$ .

**Definition 25 (Well-Founded Rewriting).** *The well-founded rewriting of an F-logic program  $P$ , denoted  $P^{wf}$ , is a general logic program constructed by the following steps:*

- (1) For every rule  $R$  in  $P$ , add its rewriting  $\rho(R)$  into  $P^{wf}$ .
- (2) Include the trailer shown in Figure 7 to  $P^{wf}$  (note that uppercase letters denote variables in the trailer rules).

**Definition 26.** *Let  $P^{wf}$  be the well-founded rewriting of an F-logic program  $P$ ,  $\mathcal{HB}_{P^{wf}}$  be the Herbrand base of  $P^{wf}$ ,  $\mathcal{HB}_P$  be the Herbrand base of  $P$ ,  $I^{wf}$  be a subset of  $\mathcal{HB}_{P^{wf}}$ , and  $I$  be a subset of  $\mathcal{HB}_P$ , we say that  $I^{wf}$  is isomorphic to  $I$ , if all of the following conditions are true:*

$mvd(S, M, V) \leftarrow locmvd(S, M, V).$
$mvd(S, M, V) \leftarrow inhmvd(S, M, V, C).$
$sub(S, C) \leftarrow sub(S, O), sub(O, C).$
$inhmvd(S, M, V, C) \leftarrow candidate(C, M, V, S), \neg multiple(C, S, M).$
$candidate(C, M, V, S) \leftarrow sub(S, C), locmvd(C, M, V), C \neq S,$
$\quad \neg local(S, M), \neg override(C, S, M).$
$local(S, M) \leftarrow locmvd(S, M, V).$
$multiple(C, S, M) \leftarrow candidate(O, M, V, S), O \neq C.$
$override(C, S, M) \leftarrow sub(O, C), sub(S, O), locmvd(O, M, V), O \neq C, O \neq S.$

Fig. 7. Trailer Rules for Well-Founded Rewriting

- (1) for all  $s$  and  $c$ :  $sub(s, c) \in I^{wf}$  iff  $s :: c \in I$
- (2) for all  $s, m$ , and  $v$ :  $locmvd(s, m, v) \in I^{wf}$  iff  $s[m \rightarrow v]_s \in I$
- (3) for all  $s, m, v$ , and  $c$ , such that  $s \neq c$ :  $inhmvd(s, m, v, c) \in I^{wf}$  iff  $s[m \rightarrow v]_c \in I$

Let  $T^{wf}$  and  $U^{wf}$  be disjoint sets of true and undefined atoms in  $\mathcal{HB}_{P^{wf}}$ , respectively,  $\mathcal{M}^{wf} = \langle T^{wf}; U^{wf} \rangle$  be the well-founded model of  $P^{wf}$ , and  $\mathcal{M} = \langle T; U \rangle$  be the optimistic object model of  $P$ . We say that  $\mathcal{M}^{wf}$  is isomorphic to  $\mathcal{M}$ , if  $T^{wf}$  is isomorphic to  $T$  and  $U^{wf}$  is isomorphic to  $U$ .

**Theorem 3.** Given the well-founded rewriting  $P^{wf}$  of an F-logic program  $P$ , the well-founded model of  $P^{wf}$  is isomorphic to the optimistic object model of  $P$ .

## 7 Data Complexity

In general, the optimistic object model of an F-logic program is not necessarily recursively enumerable. However, for function-free F-logic programs, the Herbrand universe is finite and thus the optimistic object model can be effectively constructed. In this section we discuss data complexity for such programs.

Since in this paper we only consider a subset of F-logic, which contains only two kinds of atoms,  $s :: c$  and  $s[m \rightarrow v]$ , any ground atomic query must have one of these two forms, where  $s, c, m, v$  are constants.

As for Datalog programs, we can divide any F-logic program,  $P$ , into two disjoint parts: an *intensional* database (IDB),  $P_R$ , which consists of all rules in  $P$  and no facts, and an *extensional* database (EDB),  $P_F$ , which contains only the facts in  $P$ . We can think of  $P_R$  as a function that maps any EDB,  $P_F$ , to the optimistic object model of the combined F-logic program  $P_R \cup P_F$ . Following [32], we have the following definition of data complexity.

**Definition 27 (Data Complexity).** Given an IDB  $P_R$  and an EDB  $P_F$ , the data complexity of  $P_R$  is defined as the computational complexity of deciding the

*truth value of any ground atomic query in the optimistic object model of  $P_R \cup P_F$ , as a function of the size of  $P_F$ .*

Given an F-logic program  $P = P_R \cup P_F$  and its well-founded rewriting  $P^{wf}$ , let  $P_R^{wf}$  be the IDB of  $P^{wf}$ , and  $P_F^{wf}$  be the EDB of  $P^{wf}$ . By Definitions 24 and 25,  $P_R^{wf}$  consists of the trailer shown in Figure 7 plus the rewritings of all rules in  $P_R$ . The EDB  $P_F^{wf}$  consists of the rewritings of all facts in  $P_F$ . Because the rewriting of an F-logic rule is linear and the size of the trailer is a constant, the size of  $P_R^{wf}$  is linear in the size of  $P_R$  and the size of  $P_F^{wf}$  is also linear in the size of  $P_F$ .

By Theorem 3, the well-founded model of  $P^{wf}$  is isomorphic to the optimistic object model of  $P$ . Therefore, the data complexity of the optimistic object model semantics reduces to the data complexity of the well-founded semantics.

Because the rewriting does not introduce new function symbols, the rewriting of a function-free F-logic program is a function-free Datalog program. Since data complexity of the well-founded semantics for function-free programs is polynomial time [13], we have the following corollary.

**Corollary 1.** *The data complexity of the optimistic object model semantics for function-free F-logic programs is polynomial time.*

## 8 Conclusion and Future Work

We developed a new model theory for nonmonotonic multiple inheritance in frame-based knowledge bases. Unlike previous attempts, this new semantics is rooted in well-defined postulates, which formalize the commonly accepted intuition behind nonmonotonic multiple inheritance in frame-based languages, and extend this intuition to handle the interaction between deduction and inheritance. Although we chose F-logic as a framework for presenting our results, the problem described here is *inherent* in any logic-based system that supports both inheritance and rules. Our solution applies to such systems and, in particular, to extensions of DAML+OIL [17] which permit deductive rules and inheritance.

We should note that the kind of inheritance considered here is known as *value inheritance*, which is commonly used in AI systems. However, it has been shown that value inheritance can also simulate *code inheritance* that is traditionally used in object-oriented programming [22]. Therefore, our results apply to code inheritance as well.

In addition to the inheritance semantics, our formalization of the concepts of locality, context, inheritance candidacy, and the inheritance postulates sets the stage for a framework in which *programmable* inheritance policies can be defined formally. It has been argued in the past that along with the “default” semantics for inheritance, tools for programming inheritance-like deduction in an ad hoc way can benefit certain applications. For instance, discretionary access control and trust management can benefit from a variety of inheritance-based strategies, such as most-specific-definition-based overriding [18] (similar to the semantics

developed here), path-based overriding [18], inflating inheritance [20], and null inheritance [20]. We are currently working on extensions of our framework to allow users to specify their inheritance strategies *declaratively*. Such a system can be part of an infrastructure for advanced knowledge base systems.

## References

1. S. Abiteboul, G. Lausen, H. Uphoff, and E. Waller. Methods and rules. In *ACM SIGMOD Conference on Management of Data*, pages 32–41, 1993.
2. A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133:205–265, October 1994.
3. G. Brewka. The logic of inheritance in frame systems. In *International Joint Conference on Artificial Intelligence*, pages 483–488, San Francisco, CA, 1987. Morgan Kaufmann.
4. M. Bugliesi and H. M. Jamil. A stable model semantics for behavioral inheritance in deductive object oriented languages. In *International Conference on Database Theory*, pages 222–237, 1995.
5. L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2):138–164, February 1988.
6. W. Chen, M. Kifer, and D. S. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.
7. W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of ACM*, 43(1):20–74, 1996.
8. H. Davulcu, G. Yang, M. Kifer, and I.V. Ramakrishnan. Design and implementation of the physical layer in webbases: The XRouter experience. In *First International Conference on Computational Logic, DOOD'2000 Stream*, July 2000.
9. S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL'98 - The Query Languages Workshop*, December 1998.
10. D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: Or how to enable intelligent access to the WWW. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1998.
11. J. Frohn, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–613, 1998.
12. A. Van Gelder. The alternating fixpoint of logic programs with negation. In *ACM Symposium on Principles of Database Systems*, pages 1–10, 1989.
13. A. Van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, July 1991.
14. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080. The MIT Press, 1988.
15. C. H. Goh, S. Bressan, S. E. Madnick, and M. D. Siegel. Context interchange: Representing and reasoning about data semantics in heterogeneous systems. Technical report, MIT, School of Management, 1996.
16. A. Gupta, B. Ludäscher, and M. E. Martone. Knowledge-based integration of neuroscience data sources. In *12th International Conference on Scientific and Statistical Database Management (SSDBM)*, Berlin, Germany, July 2000. IEEE.
17. I. Horrocks. DAML+OIL: A description logic for the Semantic Web. *IEEE Bulletin of the Technical Committee on Data Engineering*, 25(1), March 2002.

18. S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems*, 26(2):214–260, June 2001.
19. H. M. Jamil. Implementing abstract objects with inheritance in Datalog<sup>neg</sup>. In *International Conference on Very Large Data Bases*, pages 56–65, 1997.
20. H. M. Jamil. A logic-based language for parametric inheritance. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 611–622, San Francisco, 2000. Morgan Kaufmann.
21. M. Kifer and G. Lausen. F-Logic: A higher-order language for reasoning about objects, inheritance and schema. In *ACM SIGMOD Conference on Management of Data*, pages 134–146, New York, 1989. ACM.
22. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.
23. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1984.
24. W. May and P. Kandzia. Nonmonotonic inheritance in object-oriented deductive database languages. *Journal of Logic and Computation*, 11(4), 2001.
25. W. May, B. Ludäscher, and G. Lausen. Well-founded semantics for deductive object-oriented database languages. In *International Conference on Deductive and Object-Oriented Databases*, pages 320–336. Springer Verlag LNCS, 1997.
26. T. C. Przymusiński. Every logic program has a natural stratification and an iterated least fixed point model. In *ACM Symposium on Principles of Database Systems*, pages 11–21, 1989.
27. T. C. Przymusiński. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464, 1990.
28. R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.
29. M. Sintek and S. Decker. TRIPLE – An RDF query, inference, and transformation language. In *Deductive Databases and Knowledge Management (DDL’2001)*, October 2001.
30. S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. AI for the Web — Ontology-based community web portals. In *9-th International World Wide Web Conference (WWW9)*, Amsterdam, The Netherlands, May 2000.
31. D. S. Touretzky. *The Mathematics of Inheritance*. Morgan-Kaufmann, Los Altos, CA, 1986.
32. M. Vardi. The complexity of relational query languages. In *ACM Symposium on Theory of Computing*, pages 137–145, 1982.
33. G. Yang and M. Kifer. Implementing an efficient DOOD system using a tabling logic engine. In *First International Conference on Computational Logic, DOOD’2000 Stream*, July 2000.
34. G. Yang and M. Kifer. Flora-2: User’s manual. <http://flora.sourceforge.net/>, June 2002.