Versatile Semantic Modeling of Frame Logic Programs under Answer Set Semantics

Mario Alviano, Giovambattista Ianni, Marco Marano, and Alessandra Martello

Dipartimento di Matematica, Università della Calabria, I-87036 Rende (CS), Italy.
lastname@mat.unical.it

Abstract. This work introduces the framework of Frame Answer Set programs (FAS). FAS programs are a frame logic-like language working under answer set semantics augmented with higher order constructs.
The syntax of the language includes the possibility to manipulate nested molecules, class hierarchies, basic method signatures and contexts (called *framespaces*). Semantics is defined in terms of a corresponding stable model semantics, paving the way to model object ontologies and their semantics under this well known paradigm.
The language is purposely designed so that inheritance behavior and other features of the language can be easily customized by the introduction of specialized axiomatic modules, which can be modeled on purpose by advanced developers of ontology languages. Also, contexts allow to model hybrid systems integrating multiple data sources working under different entailment regimes. Properties and relationship with original F-logic semantics of some of the presented axiomatizations are given. A system prototype has been implemented and is available for evaluation.

1 Introduction

Frame Logic (F-logic) [17, 33] is a knowledge representation and ontology modeling language which combines the declarative semantics and expressiveness of deductive database languages with the rich data modeling capabilities supported by the object oriented data model.
As such, F-logic constitutes both an important methodology and a tool for modeling ontologies in the context of Semantic Web. This is witnessed by projects which focussed in F-logic as representation language, such as WSMO [9, 27]. Also, F-logic features play a crucial role in the ongoing activity of the RIF Working group [3, 2][1]. F-logic was originally defined under first-order semantics [17], while a well-founded semantics, satisfactorily dealing with nonmonotonic inheritance can be found in [33].
The stable model semantics (nowadays better known as Answer Set Programming – ASP), has some attractive feature which make interesting to consider the possibility of defining a frame-based language under this setting. ASP is nowadays a mature field, offering languages and systems[2], based on a strongly assessed model-theoretic semantics [15]. ASP allows to model declaratively non-determinism and gives the possibility to specify, in a declarative way, search spaces, preferences, strong and soft constraints [6], and more). ASP shares with

[1] http://www.w3.org/2005/rules/wiki/RIF_Working_Group.
[2] among the variety of such systems we recall here DLV [19] and smodels [29].

F-logic under well-founded semantics the possibility to reason about ontologies using nonmonotonic constructs, included nonmonotonic inheritance, as it is done in some ASP extensions conceived for modeling ontologies [26].

This paper aims at closing the gap between F-logic based languages and Answer Set Programming, in both directions: on one hand, Answer Set Programming misses the useful F-logic syntax, its higher order reasoning capabilities, and the possibility to focus knowledge representation on objects, more than on predicates. On the other hand, manipulating F-logic ontologies under stable model semantics opens a variety of modeling possibilities, given the higher expressiveness of the latter with respect to well-founded semantics.

Our approach is set in between a pure model theoretic semantics (proper of F-logic and many of its extensions [17, 33]), and a pure "rewriting" semantics, in which inheritance is specified by means of an ad-hoc translation to logic programming [16].

In the former case, semantics is given in a clean and sound manner: however, the way inheritance (and in general, the semantics of the language) is modeled is hardwired within the logic language at hand, and cannot be easy subject of modifications. In the latter case, semantics is enforced by describing a rewriting algorithm from theories to appropriate logic programs. In such a setting the semantics of the overall language can be better tuned by changing the rewriting strategy. It is however necessary to have knowledge of internal details about how the language is mapped to logic programming, making the process of designing semantics cumbersome and virtually reserved to the authors of the language only.

In this work, we define a basic stable model semantics for FAS programs which does not purposely fix a special meaning for the traditional operators of F-logic, such as class membership ":" and subclass containment "::". Indeed, FAS programs are conceived as a test-bed on which an advanced ontology designer is allowed to choose the behavior of available operators from a predefined library, or to design her own semantics from scratch. The ability to customize the semantics of the language is crucial especially in presence of inheritance constructs. In fact, when one has to model a particular problem, a specific semantics for inheritance may be more suitable than another, and it is often necessary to manipulate and/or combine the predefined behaviors of the language.

The contributions of our paper are highlighted next:

1. We present the family of Frame Answer Set Programs (FAS programs), allowing usage of frame-like constructs, and of higher order atoms. Interestingly, positively *nested frames* may appear both in the head and in the body of rules. The language allows to reason in multiple *contexts* which are called *framespaces*.
2. We provide the model-theoretic semantics of FAS programs in terms of their *answer sets*.
3. We show how semantics features can be introduced on top of the basic semantics of the language by adding an appropriate axiomatization. Structural, behavioral, and arbitrary semantic for inheritance can be easily designed and coupled with user ontologies. In some cases, we show how these axiomatizations relate with F-logic under first order semantics.

4. We illustrate in which terms contexts can be exploited for manipulating hybrid knowledge bases having many data sources working under different entailment regime;
5. The language has been implemented within the DLT system, a front-end for answer set solvers. Besides the fragment of language herein presented, DLT allows negated nested molecules, in the spirit of [20], and re-usable template programs. If coupled with a proper answer set solver, the same front-end allows usage of complex terms (e.g. functions, lists, sets), and external predicates [12]).

The remainder of the paper is structured as follows: Section 2 introduces the syntax of the language FAS (Frame Answer Set). Section 3 contains a formalization of the semantics of FAS programs, while Section 4 describes how to use the language for modeling and axiomatizing knowledge, and proves some properties of the axiomatic modules presented. The system supporting FAS programs is described in Section 6; related works are discussed in Section 7 and conclusions are then drawn.

2 Syntax

We present here the syntax of FAS programs. Informally, the language allows disjunctive rules with negation as failure in the body; with respect to ordinary Ans-Prolog (the basic language of Answer Set Programming), there are three crucial differences. First, besides traditional atoms and predicates, the language supports *frame molecules* in both the body and the head of rules, following the style of F-logic [17]. When representing knowledge, frame molecules allow to focus on objects, more than on predicates. An object can belong to *classes*, and have a number of *property* (attribute) values. As an example, the following is a frame molecule:

$$\begin{aligned} brown : employee\,[\;& surname \rightarrow \text{“Mr. Brown”}, \\ & skill \twoheadrightarrow \{java, asp\}, \\ & salary \rightarrow 800, \\ & gender \rightarrow male, \\ & married \rightarrow pink\,] \end{aligned}$$

The above molecule defines membership of the *subject* of the molecule (*brown*) to the *employee* class and asserts some values corresponding to the *properties* (which we will call also *attributes*) bound to this object. This frame molecule states that *brown* is *male* (as expressed by the value of the attribute *gender*), and is *married* to another employee identified by the subject *pink*. *brown* knows *java* and *asp* languages, as the values of the *skill* property suggest, while he has a *salary* equal to *800*. Intuitively, one can see a class membership statement in form $x : c$ as similar to a unary predicate $c(x)$. Accordingly, $x[m \rightarrow v]$ can be seen has a binary predicate $m(x, v)$.
As a second important difference, higher order reasoning is a first class citizen in the language: in other words, it is allowed quantification over predicate, class and property names. For instance, $C(brown)$ is meant to have the variable C ranging over the Herbrand universe, thus having $employee(brown)$ as possible ground instance.

Finally, our language allows the use of *framespaces* to place atoms and molecules in different contexts. For example, suppose there are two *Mr. Brown*, one working for *Sun* and the other for *Ibm*. We can use two different assertions, related to two different framespaces to distinguish them, e.g. $brown : employee@sun$ and $brown : employee@ibm$.
We formally define the syntax of the language next.
Let $\mathcal{C}$ be an infinite and countable set of distinguished constant and predicate symbols. Let $\mathcal{X}$ be a set of variables. We conventionally denote variables with uppercase first letter (e.g. X, $Project$), while constants will be denoted with lowercase first letter (e.g. x, $brown$, $nonWantedSkill$). A *term* is either a constant or a variable.
Atoms can be either *standard atoms* or *frame atoms*. A standard atom is in the form $t_0(t_1, \ldots, t_n)@f$, where $t_0, \ldots, t_n, f$ are *terms*, t_0 represents the *predicate name* of the atom and f the *context* (or *framespace*) in which the atom is defined. A *frame atom*, or *molecule*, can be in one of the following three forms:

- $s[v_1, \ldots, v_n]@f$
- $s \diamond c@f$
- $s \diamond c[v_1, \ldots, v_n]@f$

where s, c and f are terms, and $v_1, \ldots, v_n$ is a list of *attribute expressions*. Here and in the following, the allowed values for the meta-symbol $\diamond$ are ":" (*instance operator*), or "::" (*subclass operator*). Moreover, s is called the *subject* of the frame, while f represents the *context* (or *framespace*).
To simplify the notation, whenever the context term f is omitted, we will assume $f = d$, for $d \in \mathcal{C}$ a special symbol denoting the *default* context.
An attribute expression is in the form p, $p \rightharpoonup v_1$ or $p \rightharpoonup\!\!\!\rightharpoonup \{v_1, \ldots, v_n\}$, where p (*the property/attribute name*) is a term, and $v_1, \ldots, v_n$ (*the attribute values*) are either terms or frame molecules. Here and in the following, the meta-symbols $\rightharpoonup$ and $\rightharpoonup\!\!\!\rightharpoonup$ are intended to range respectively over $\{\rightarrow, \bullet\!\!\rightarrow\}$ and $\{\Rightarrow, \twoheadrightarrow, \Rightarrow\!\!\!\!\Rightarrow, \bullet\!\!\twoheadrightarrow\}$. Note that, according to this definition, when used within attribute expressions, the symbols in the set $\{\Rightarrow, \twoheadrightarrow, \Rightarrow\!\!\!\!\Rightarrow, \bullet\!\!\twoheadrightarrow\}$ allow sets of attribute values on their right hand side, while $\rightarrow$ and $\bullet\!\!\rightarrow$ allow single values.
A *literal* is either an atom p (positive literal), or an expression of the form $\neg p$ (strongly negated literal or, simply, negated literal), where p is an atom. A *naf-literal* (negation as failure literal) is either of the form b (positive naf-literal), or of the form $not\, b$ (negative naf-literal), where b is a literal.
A *formula* is either a naf-literal, a conjunction of formulas or a disjunction of formulas.
A *simple atom* is either a standard atom, or a frame atom in the forms $s \diamond c@f$, $s[p \rightharpoonup v]@f$ or $s[p \rightharpoonup\!\!\!\rightharpoonup \{v\}]@f$, for s, c, p, v and f terms of the language. The notion of simple literal and of simple naf-literal are defined accordingly on top of the notion of simple atom.
A Frame Answer Set *program* (FAS program) is a set of *rules*, of the form

$$a_1 \vee, \ldots, \vee a_n \leftarrow b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m.$$

where $a_1, \ldots, a_n$ and $b_1, \ldots, b_k$ are literals, $not\, b_{k+1}, \ldots, not\, b_m$ are naf-literals, and $n \geq 0$, $m \geq k \geq 0$. The disjunction $a_1 \vee \cdots \vee a_n$ is the head of r, denoted by

$H(r)$, while the conjunction $b_1 \wedge \cdots \wedge b_k \wedge not\, b_{k+1} \wedge \ldots, \wedge not\, b_m$ is the body of r, denoted by $B(r)$. A rule with empty body will be called *fact*, while a rule with empty head is a *constraint*.
A *plain higher order* FAS program contains only standard atoms, while a *plain* FAS program contains only standard atoms with a constant predicate name. A *positive* FAS program do not contain negation as failure and strongly negated atoms. In the following, we will assume to deal with *safe* FAS programs, that is, programs in which each variable appearing in a rule r appears in at least one positive naf-literal in $B(r)$.

Example 1. The following one rule program is a valid FAS program. Intuitively, it represents the fact that each person is *male* or *female*.

$$P[gender \rightarrow \text{“male”}] \vee P[gender \rightarrow \text{“female”}] \;\leftarrow\; P : person.$$

3 Semantics

Semantics of FAS programs is defined by adapting the traditional Gelfond-Lifschitz reduct, originally given for a ground disjunctive logic program with strong and default negation [15], to the case of FAS programs.
Given a FAS program P, its ground version $grnd(P)$ is given by grounding rules of P by all the possible substitutions of variables that can be obtained using consistently elements of $\mathcal{C}$[3]. A ground rule thus contains only ground atoms; the set of all possible simple ground literals that can be constructed combining predicates and terms occurring in the program is usually referred to as *Herbrand base* (B_P). We remark that the grounding process substitutes also nonground predicates names with symbols from $\mathcal{C}$ (e.g., a valid ground instance of the atom $H(brown, X)$ is $married(brown, pink)$, while a valid ground instance of $brown[H \rightarrow yellow]$ is $brown[color \rightarrow yellow]$).
An *interpretation* for P is a set of simple ground literals, that is, an interpretation is a subset $I \subseteq B_P$. I is said to be *consistent* if $\forall a \in I$ we have that $\neg a \notin I$.
We define the following entailment notion with respect to an interpretation I. For a a ground atom:

($E1$) If a is simple, then $I \models a$ iff $a \in I$;
($E2$) $I \models not\, a$ iff $I \not\models a$.

For $l_1, \ldots, l_n$ ground literals:

($E3$) $I \models l_1 \wedge \cdots \wedge l_n$ iff $I \models l_i$, for each $1 \leq i \leq n$;
($E4$) $I \models l_1 \vee \cdots \vee l_n$ iff $I \models l_i$ for some $1 \leq i \leq n$.

For s, p, f ground terms, and $m_1, \ldots, m_n$ ground frame molecules:

($E5$) $I \models s[p \rightharpoonup \{m_1, \ldots m_n\}]@f$ iff $I \models s[p \rightharpoonup \{m_i\}]@f$, for each $1 \leq i \leq n$.

For s, s', c, p, f, f' ground terms, and $\overline{v} = \{v_1, \ldots, v_n\}$ a set of ground attribute value expressions:

[3] As shown next, our semantics implicitly assumes that elements of $\mathcal{C}$ are mapped to themselves in any interpretation, thus embracing the unique name assumption.

$(E6)$ $I \models s[\,v_1,\ldots,v_n\,]@f$ iff $I \models s[\,v_1\,]@f \wedge \cdots \wedge s[v_n]@f$;
$(E7)$ $I \models s \diamond c[\,\overline{v}\,]@f$ iff $I \models s \diamond c\,@f \wedge s[\,\overline{v}\,]@f$;

$(E8)$ $I \models s[\,p \rightharpoonup s'[\,\overline{v}\,]\,]@f$ iff $I \models s[\,p \rightharpoonup s']@f \wedge s'[\,\overline{v}\,]@f$;
$(E9)$ $I \models s[\,p \rightharpoonup\!\!\rightharpoonup \{s'[\,\overline{v}\,]\}\,]@f$ iff $I \models s[\,p \rightharpoonup\!\!\rightharpoonup \{s'\}]@f \wedge s'[\,\overline{v}\,]@f$;

$(E10)$ $I \models s[\,p \rightharpoonup s'[\,\overline{v}\,]@f'\,]@f$ iff $I \models s[\,p \rightharpoonup s']@f \wedge s'[\,\overline{v}\,]@f'$;
$(E11)$ $I \models s[\,p \rightharpoonup\!\!\rightharpoonup \{s'[\,\overline{v}\,]@f'\}\,]@f$ iff $I \models s[\,p \rightharpoonup\!\!\rightharpoonup \{s'\}]@f \wedge s'[\,\overline{v}\,]@f'$.

Note that rules $(E8)$ and $(E9)$ force $s'[\,\overline{v}\,]$, which does not have an explicit framespace, to belong to the context f of the molecule containing it. On the contrary, $s'[\,\overline{v}\,]@f'$ in $(E10)$ and $(E11)$ has a proper framespace f', and the entailment rules take care of this fact. Then, rules $(E6)$ to $(E11)$ define the context of a frame molecule as the *nearest* framespace explicitly specified.
For a rule r :

$(E12)$ $I \models r$ iff $I \models H(r)$ or $I \not\models B(r)$;

A *model* for P is an interpretation M for P such that $M \models r$ for every rule $r \in grnd(P)$. A model M for P is *minimal* if no model N for P exists such that N is a proper subset of M. The set of all minimal models for P is denoted by $\mathrm{MM}(P)$.

Given a program P and an interpretation I, the *Gelfond-Lifschitz (GL) transformation* of P w.r.t. I, denoted P^I, is the set of positive rules of the form $\{a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k\,\}$ such that $\{a_1 \vee \cdots \vee a_n \leftarrow b_1, \cdots, b_k, \mathit{not}\ b_{k+1}, \cdots, \mathit{not}\ \ b_m\}$ is in grnd(P) and $I \models \mathit{not}\, b_{k+1} \wedge \cdots \wedge\ \mathit{not}\ b_m$. An interpretation I for a program P is an *answer set* for P if $I \in \mathrm{MM}(P^I)$ (i.e., I is a minimal model for the positive program P^I) *[24, 15]*. The set of all answer sets for P is denoted by $ans(P)$. We say that $P \models a$ for an atom a, if $M \models a$ for all $M \in ans(P)$. P is *consistent* if $ans(P)$ is non-empty.
For a positive program P allowing only the term d in context position, we define the F-logic first-order semantics in terms of its *F-models*. A *F-model* M_f is a model of P subject to the conditions

$(F1)$ "::" encodes a partial order in M_f;
$(F2)$ if $a\!:\!b \in M_f$ and $b\!::\!c \in M_f$ then $a\!:\!c \in M_f$;
$(F3)$ if $a[m \rightharpoonup v] \in M_f$ and $a[m \rightharpoonup w] \in M_f$ then $v = w$, for $\rightharpoonup \in \{\rightarrow, \bullet\!\!\rightarrow\}$;
$(F4)$ if $a[m \approx\!> v] \in M_f$ and $b\!::\!a$ then $b[m \approx\!> v] \in M_f$, for $\approx\!> \in \{\Rightarrow, \Rightarrow\!\!\!\!\Rightarrow\}$;
$(F5)$ if $c[m \Rightarrow v]$, $a\!:\!c$ and $a[m \rightarrow w] \in M_f$ then $w\!:\!v \in M_f$;
$(F6)$ if $c[m \Rightarrow\!\!\!\!\Rightarrow v]$, $a\!:\!c$ and $a[m \twoheadrightarrow w] \in M_f$ then $w\!:\!v \in M_f$;

We say that $P \models_f a$ for an atom a if $M_f \models a$ for all F-models of P.

Example 2. The program in Example 1 together with the fact $brown\!:\!person$. has two answer sets, $M_1 = \{\,brown\!:\!person,\ brown[\,gender \rightarrow \text{“male”}\,]\,\}$ and $M_2 = \{\,brown\!:\!person,\ brown[gender \rightarrow \text{“female”}]\,\}$. Both M_1 and M_2 are F-models. Note that $M_3 = \{\,brown : person,\ brown[\,gender \rightarrow \text{“female”}\,],\ brown[\,gender \rightarrow \text{“male”}\,]\,\}$ is neither an F-model nor an answer set for different reasons: it is not an F-model because of condition $(F3)$ given above, while it is not an answer set because it is not minimal. Note also that disjunctive rules trigger in general the existence of multiple answer sets, while the presence of constraints may eliminate some or all constraints: for instance, the same program enriched with the constraints $\leftarrow brown[gender \rightarrow \text{“male”}]$ and $\leftarrow brown[gender \rightarrow \text{“female”}]$ has no answer set[4].

4 Modeling semantics and inheritance

Given the basic semantics for a FAS program P, it is then possible to enforce a specific behavior for operators of the language by adding to P specific "axiomatic modules". An *axiomatic module* A is in general a FAS program. Given a union of axiomatic modules $S = A_1 \cup \cdots \cup A_n$, we will say that P entails a formula ϕ under the axiomatization S ($P \models_S \phi$) if $P \cup S \models \phi$. The answer sets of P under axiomatization S are defined as $ans_S(P) = ans(P \cup S)$.
We illustrate next some basic axiomatic modules.

Basic class taxonomies. The axiomatic module $\mathcal{C}$, shown next, associates to "$:$" and "$::$" the usual meaning of monotonic class membership and subclass operator.

$c_1: \; A::B \leftarrow A::C, \; C::B.$
$c_2: \; A::A \leftarrow X:A.$
$c_3: \; \leftarrow A::C, \; C::A, \; A \neq C.$
$c_4: \; X:C \leftarrow X:D, \; D::C.$

Rules c_1 and c_2 enforce transitivity and reflexivity of the subclass operator, respectively. Rule c_3 prohibits cycles in the class taxonomy, while c_4 implements the class inheritance for individuals by connecting the "$::$" operator to the "$:$" operator. The acyclicity constraint can be relaxed if desired: we define in this case $\mathcal{C}'$ as $\mathcal{C} \setminus c_3$[5].

Single valued attributes. Under standard F-logic, the operators $\rightarrow$ and $\bullet\!\!\rightarrow$ are associated to families of single valued functions: indeed, in a F-model M it can not hold both $a[m \rightharpoonup v]$ and $a[m \rightharpoonup w]$, unless $v = w$. Under unique names assumption, we can state the above condition by the set $\mathcal{F}$ of constraints:

$f_5: \; \leftarrow A[M \rightarrow V], A[M \rightarrow W], V \neq W$
$f_6: \; \leftarrow A[M \bullet\!\!\rightarrow V], A[M \bullet\!\!\rightarrow W], V \neq W$

Structural and behavioral inheritance. We show here how to model some peculiar types of inheritance, such as structural and behavioral inheritance.
Structural inheritance is usually associated to the operator $\Rightarrow$. Let P_1 be the following example program:

$webDesigner::javaProgrammer. \; javaProgrammer::programmer.$
$webDesigner::htmlProgrammer. \; javaProgrammer[salary \Rightarrow medium].$
$htmlProgrammer[salary \Rightarrow low].$

For short, we denote in the following *webDesigner* as *wd*, *javaProgrammer* as *jp* and *htmlProgrammer* as *hp*.
Under structural inheritance, as defined in [17], property values of superclasses are "monotonically" added to subclasses. Thus, since c_1 is subclass of c_2 and

4 A constraint $\leftarrow c$ can be seen as a rule $f \leftarrow c, not\, f$, for which there is no model containing c.
5 Note that the atom $A \neq C$ amounts to *syntactic* inequality between A and C.

c_4, one expects that $P_1 \models_{\mathcal{C}\cup\mathcal{S}} webDesigner[salary \Rightarrow \{low, medium\}]$ for some axiomatic module $\mathcal{S}$.
The axiomatic module $\mathcal{S}$ shown next, associates this behavior to the operators $\Rightarrow$ and $\Rrightarrow$.

$$
\begin{array}{l}
s_7: \; D[A \Rightarrow T] \leftarrow D::C,\; C[A \Rightarrow T].\\
s_8: \; D[A \Rrightarrow T] \leftarrow D::C,\; C[A \Rrightarrow T].
\end{array}
$$

Note that s_5 (resp. s_6) do not enforce any relationship between "$\Rightarrow$" and "$\rightarrow$" (resp. "$\Rrightarrow$" and "$\twoheadrightarrow$") as in [17]. We will discuss this issue later in the section.
Behavioral inheritance [33], allows instead nonmonotonic overriding of property values. Overriding is a common feature in object-oriented programming languages like Java and C++: when a more specific definition (value, in our case) is introduced for a method (a property, in our case), the more general one is overridden. In case different information about an attribute value can be derived from several inheritance paths, inheritance is *blocked*. Let us assume to add to P_1 the assertions $jp[income \bullet\!\!\rightarrow 1000]$ and $hp[income \bullet\!\!\rightarrow 1200]$.
Under behavioral inheritance regime [33][6], the assertions $jp[income \bullet\!\!\rightarrow 1000]$ and $hp[income \bullet\!\!\rightarrow 1200]$ would be considered in conflict when inherited from wd. Indeed, both $wd[income \bullet\!\!\rightarrow 1000]$ and $wd[income \bullet\!\!\rightarrow 1200]$ under the three-valued semantics of [33] are left *undefined*. Under FAS semantics it is then expected to have some axiomatic module $\mathcal{B}$ where neither $P_1 \models_{\mathcal{B}\cup\mathcal{F}\cup\mathcal{C}} wd[income \bullet\!\!\rightarrow 1000]$ nor $P_1 \models_{\cap\mathcal{B}\cup\mathcal{F}\cup\mathcal{C}} wd[income \bullet\!\!\rightarrow 1200]$ hold.
The above behavior can be enforced by defining $\mathcal{B}$ as follows

$$
\begin{array}{lrl}
b_9: & overridden(D, M, C) & \leftarrow E[M \bullet\!\!\rightarrow V],\; C::E,\; E::D,\; C \neq E,\; E \neq D.\\
b_{10}: & inheritable(C, M, D) & \leftarrow C::D,\; D[M \bullet\!\!\rightarrow V],\; not\, overridden(D, M, C).\\
b_{11}: & C[M \bullet\!\!\rightarrow V] \vee C[M \bullet\!\!\rightarrow V]@false & \leftarrow inheritable(C, M, D),\; D[M \bullet\!\!\rightarrow V].\\
b_{12}: & exists(C, M) & \leftarrow C[M \bullet\!\!\rightarrow V].\\
b_{13}: & & \leftarrow inheritable(C, M, D),\; not\, exists(C, M).\\
b_{14}: & existsSubclass(A, C) & \leftarrow A:C, A:D, D::C, C \neq D.\\
b_{15}: & A[M \rightarrow V]@candidate & \leftarrow A:C, C[M \bullet\!\!\rightarrow V], not\, existsSubclass(A, C).\\
b_{16}: & A[M \rightarrow V] \vee A[M \rightarrow V]@false & \leftarrow A[M \rightarrow V]@candidate.\\
b_{17}: & exists'(A, M) & \leftarrow A[M \rightarrow V].\\
b_{18}: & & \leftarrow inheritable(C, M, C), A:C, not\, exists'(A, M).
\end{array}
$$

The above module makes usage of stable model semantics for modeling multiple inheritance conflicts. By means of rule b_{11} and b_{16} it is triggered the existence of multiple answer set in the presence of inheritance conflicts, one for each possible way to solve the conflict itself.
Note that $ans_{\mathcal{B}\cup\mathcal{F}\cup\mathcal{C}}(P_1)$ contains two different answer sets M_1 and M_2 which respectively are such that $M_1 \models wd[income \bullet\!\!\rightarrow 1200]$ and $M_2 \models wd[income \bullet\!\!\rightarrow 1000]$. However, both assertions do not hold in all the possible answer sets. Thus, similarly to "well-founded optimism" semantics, we obtain that $P_1 \not\models_{\mathcal{C}\cup\mathcal{B}} wp[income \bullet\!\!\rightarrow X]$ for any X.

Constructive vs well-typed semantics. The operator $\Rightarrow$ is traditionally associated to $\rightarrow$. For instance if both $jp[keyboard \Rightarrow americanLayout]$ and $jim : jp[keyboard \rightarrow ibm1050]$ hold, one might expect that $ibm1050 : americanLayout$.

[6] Note that in [33] the above semantics is conventionally associated to the $\rightarrow$ operator, while we will use $\bullet\!\!\rightarrow$

However, one might wonder whether to implement the above required behavior under a *constructive* or a *well-typed* semantics.
The two type of semantics differ in the way incomplete information is dealt with. In a "well-typed" flavored semantics, most axioms are seen as hard constraints, which, if not fulfilled, make the theory at hand inconsistent.
In the first case, it may be desirable to use the "⇒" operator for defining strong desiderata about range and domain of properties, while the "→" could be used to denote actual instance values such as in the following program P_2:

$programmer[salary \Rightarrow integer].$
$g : programmer[salary \rightarrow aSalary].$
$\leftarrow X : programmer[salary \rightarrow Y], not\, Y : integer$[7]

Note that $ans(P_2)$ is empty, unless it is not *explicitly* asserted (well-typed) the fact $aSalary : integer$.
On the other hand one may want to interpret *constructively* desiderata about domain and range of properties, as it is typical, e.g. of RDFS[31]. Consider the program P_3:

$programmer[salary \Rightarrow integer].$
$g : programmer[salary \rightarrow aSalary]$
$Y : integer \leftarrow X : programmer[salary \rightarrow Y]$

Here P_3 has a single answer set containing the fact $aSalary : integer$.
The two types of semantics stem from profound philosophical differences: well-typedness is commonly (but not necessarily) associated to modeling languages inspired from database systems, living under a single model semantics and Closed World Assumption. To a large extent one can instead claim that first order logics (and descendant formalisms, such as descriptions logics and RDFS), is much more prone to deal constructively with incomplete information.
It is however worth noting that despite their conceptual difference, constructive and well-typed semantics are often needed together. As a matter of example, modeling in Java (as well as C++ and F-logic) needs both flavors. Constructiveness comes into play in inheritance within class taxonomies (e.g., if $A{::}B$ and $B{::}C$ hold, the information $A{::}C$ does not need to be well-typed and is inferred automatically), but well-typedness is required in several other contexts, (e.g. strong type-checking prescribes that a function having a given signature can not be invoked using actual parameters which are not *explicitly known* to fulfil the function signature).
Whenever required, FAS programs can be coupled with axiomatic modules encoding both well-typed and constructive axioms.
The following axiomatic module $\mathcal{CO}$ encodes constructively how the operators ⇒ and → can be related each other:

$co_{15} : V{:}T \leftarrow C[A \Rightarrow T], I{:}C, I[A \rightarrow V].$

while $\mathcal{W}$, shown next, encodes the same relation under a well-typed semantics.

$w_{16} : \leftarrow C[A \Rightarrow T], I{:}C, I[A \rightarrow V], not\, V : T.$

5 Properties of FAS programs

FAS programs have some property of interest. First, F-logic entailment can me modeled on top of FAS programs by means of the axiomatic modules $\mathcal{C}, \mathcal{S}, \mathcal{F}$, and $\mathcal{CO}$. Let $\mathcal{A} = \mathcal{C} \cup \mathcal{S} \cup \mathcal{F} \cup \mathcal{CO}$.

Theorem 1. *Given a positive, non-disjunctive,* FAS *program* P *with default contexts only, and a formula* ϕ*, then* $P \models_{\mathcal{A}} \phi$ *iff* $P \models_f \phi$*.*

Proof. (Sketch). ($\Rightarrow$) Assume $P \cup \mathcal{A}$ is inconsistent. Given that P is a positive program, then inconsistency amounts to the violation of some instance of constraints c_3, f_5 or f_6. We can show that, accordingly, there is no F-model for P. On the other hand, if $P \cup \mathcal{A}$ is consistent, one can show that the unique answer set of P is the least F-model of P.
($\Leftarrow$) It can be shown that if P has no F-model, then $P \cup \mathcal{A}$ is inconsistent. Viceversa, if P has some F-model its least model corresponds to the unique answer set of $P \cup \mathcal{A}$. □

One might wonder at the significance of $\models_{\mathcal{A}}$-entailment for disjunctive programs with negation. This entailment regime diverges quickly from the behavior of monotonic logic as soon as negation as failure and disjunction is considered, and is thus incomparable with first order F-logic. It is matter of future research to investigate on the relationship between FAS programs and F-logic under well-founded semantics.
As a second important property, we show that contexts can be exploited for modeling hybrid environments in which more than one semantics has to be taken in account. For instance one might desire a context s in which only $\mathcal{C} \cup \mathcal{S}$ hold as axiomatic modules (this is typical e.g. of RDFS reasoning restricted to ρ-DF [22]), while in a context b we would like to have a different entailment regime, taking in account e.g. $\mathcal{B}$ and $\mathcal{F}$.
We will say that an axiomatic module (resp. a program, a formula) $\mathcal{A}$ is defined at context c if for each rule $r \in \mathcal{A}$, each atom $c \in r$ has context c. If an axiomatic module (resp. a program, or a formula) $\mathcal{A}$ is defined at the default context d, then the axiomatic module $\mathcal{A}@c$, defined at context c, is obtained by replacing each atom a appearing in $\mathcal{A}$ with $a@c$.

Example 3. Consider the program P_4 defined as follows. P_4 has two contexts, *rdf* and *inh*. P_4 contains knowledge coming from an RDF triplestore defined in term of the facts $t(gb, rdf{:}type, hp)@rdf$, $t(gb, name, \text{“Gibbi”})@rdf$, etc. Also P_4 contains the rules $X{:}C@rdf \leftarrow t(X, rdf{:}type, C)@rdf$, $X[M \rightarrow V]@rdf \leftarrow t(X, M, V)@rdf$, $C{::}D@rdf \leftarrow t(C, rdfs{:}subClassOf, D)@rdf$. Then, we add to P_4 the program $P_1@inh$ where P_1 is taken from Section 4, plus the rule $X : C@inh \leftarrow X : C@rdf$. We want that $\mathcal{C}$ and $\mathcal{S}$ hold under the *rdf* context, while $\mathcal{C}$ and $\mathcal{B}$ hold under the *inh* context. This can be obtained by defining $\mathcal{A} = (\mathcal{C} \cup \mathcal{S})@rdf \cup (\mathcal{C} \cup \mathcal{B})@inh$ and evaluating P_4 under $\models_{\mathcal{A}}$-entailment.
For instance, $P_4 \models_{\mathcal{A}} gb{:}[income \bullet\!\!\rightarrow 1000]@inh$.

We clarify next how contexts interact each other. First, we consider programs in which contexts are strictly separated: that is, each rule in a program contains only atoms either with context a or only atoms with context b. This way, a program can be seen as composed by two separate modules, one defining a and the other defining b. The following proposition shows that programs defined in separated context behave separately under their axiomatic regime.

Proposition 1. *It is given a program* $P = P'@a \cup P''@b$*, and axiomatic modules* $A@a$ *and* $B@b$*. Then, for formulas* $\phi@a$ *and* $\psi@b$*, we have that, if* $P \cup A@a \cup B@b$ *is consistent,*

$$P \models_{A@a \cup B@b} \phi@a \wedge \psi@b \Leftrightarrow P' \models_A \phi \wedge P'' \models_B \psi$$

Contexts can be seen in some sense as separate knowledge sources, each of which having its own semantics for its data. In such a setting, it is however important to consider cases in which knowledge flows bidirectionally from a context to another and viceversa.

This situation is typical of languages implementing hybrid semantics schemes. For instance, $\mathcal{DL}+log$ [28] is a rule language where each knowledge base combines a description logic base D (living under first order semantics), with a rule program P (living under answer set semantics). D and P can mutually exchange knowledge: in the case of $\mathcal{DL}+log$, predicates of D can appear in P, allowing flow of information from D to P.

Similarly, we are assuming to have a program P, two contexts a and b, each of which coupled with axiomatic modules $A@a$ and $B@b$. The program P freely combines atoms with context a with atoms with context b, possibly in the same rule.

For simplicity, the following theorem is given for programs containing simple naf-literals only.

Given an interpretation I we define I_a as the subset of I containing only atoms with context a. The *extended reduct* P^{*I_a} of a ground program P is given by modifying each rule $r \in P$ in the following way:

- if $l@a \in H(r)$ and $l@a \notin I_a$ then delete $l@a$ from r;
- if $l@a \in H(r)$ and $l@a \in I_a$ then delete r;
- if $l@a \in B(r)$ and $l@a \in I_a$ then delete $l@a$ from r;
- if $l@a \in B(r)$ and $l@a \notin I_a$ then delete r;
- if $not\, l@a \in B(r)$ and $l@a \notin I_a$ then delete $not\, l@a$ from r;
- if $not\, l@a \in B(r)$ and $l@a \in I_a$ then delete r;

Theorem 2. *Let P be a program containing only atoms with context a and b, and $A@a$ and $B@b$ be two axiomatic modules.*
Then,

$$M \in ans_{A@a \cup B@b}(P) \Leftrightarrow M_a \in ans_{A@a}(P^{*M_b}) \wedge M_b \in ans_{B@b}(P^{*M_a})$$

Roughly speaking, the above theorem states that from the point of view of context a one can see atoms from context b as external facts, and viceversa. An answer set M of the overall program is found when, assuming M_a as the set of true facts for a, we obtain that M_b is the answer set of $P^{*M_a} \cup B@b$, i.e. an answer set of the program obtained by assuming facts in M_a true. Viceversa, if one assumes M_b as the set of true facts for context b, one should obtain M_a as the answer set of $P^{*M_b} \cup A@a$.

Proof. (Sketch). ($\Rightarrow$) Assume $M \in ans(P \cup A@a \cup B@b)$, it is easy, yet tedious, to construct M_a and M_b and verify that $M_a \in ans(P^{*M_b} \cup A@a)$ and $M_b \in ans(P^{*M_a} \cup B@b)$. Given $P_a = P^{*M_b} \cup A@a$ and $P_b = P^{*M_a} \cup B@b$, the proof is conducted by showing that M_a (resp. M_b) is a minimal model of $P_a^{M_a}$ (resp. $P_b^{M_b}$).
($\Leftarrow$) Given M_a and M_b such that $M_a \in ans(P^{*M_b} \cup A@a)$ and $M_b \in ans(P^{*M_a} \cup B@b)$, the proof is carried out by showing that $M = M_a \cup M_b$ is a minimal model of $P \cup A@a \cup A@b^M$. $\square$

6 System Overview

FAS programs have been implemented within the DLT environment [8]. The current version of the system is freely available on the DLT Web page[8], together with examples, a tutorial, and the axiomatic modules herein presented.
DLT works as a front-end for an answer set solver of choice S. Programs are rewritten in the syntax of S and then processed. Resulting answer sets in the format of S are then processed back and output in DLT format. DLT is compatible with most of the languages of the DLV family such as DLV [19], dlvhex [13] and the recent DLV-complex[9]. The native features of the solver of choice are made available to the DLT programmer: this way features such as soft constraints, aggregates (DLV), external predicates (dlvhex), and function, list and set terms (DLV-complex) are accessible. Limited support is given also for other ASP solvers.
DLT allows the syntax presented in this paper and implements the presented semantics. Atoms without context specification are assumed to have the default context d. In order to avoid typing, the default implicit context can be switched by using a directive in the form *@name.*, which sets the implicit context to *name* for the rules following the directive.
We overview next some of the other features of DLT, which, for space reasons, can not be focused in the present work.

Complex nested expression. DLT allows the usage of negated attribute expressions. From the operational point of view, if a frame literal in the body of a rule r has subject o and a negative attribute $not\ m$, our prototype removes $not\ m$ from the attributes of o, adds $not\ a$ to the body of r, where a is a fresh auxiliary atom, and adds a new rule $a \leftarrow o[m]$. to the program. This procedure can be iterated until no negated attribute appears in the program. Then, the answer sets of the original program are the answer sets of the rewritten program without auxiliary atoms. Since negated attributes can appear in negative literals and can be nested, they behave like the nested expressions of [20], allowing in many case to represent information in a more succinct way. The model-theoretical semantics of this aspect of the language is not focused in this paper and is matter of future work.

Example 4. The following rule states that a programmer P is suitable for project p_3 if P know *c++* and *perl*, but is not married to another programmer knowing *c++* and *perl*.

$$\begin{aligned} P[suitable \twoheadrightarrow p_3] \leftarrow\ & X : programmer, \\ & P : programmer[skills \twoheadrightarrow \{\text{“c++”}, \text{“perl”}\}, \\ & \quad not\ married \rightarrow X[skills \twoheadrightarrow \{\text{“c++”}, \text{“perl”}\}]. \end{aligned}$$

Template definitions. A DLT program may contain *template atoms*, that allow to define intensional predicates by means of a subprogram, where the subprogram is generic and reusable. This feature provides a succinct and elegant way for quickly introducing new constructs using the DLT language, such as predefined search spaces, custom aggregates, etc. Differently from higher order constructs, which can be used for the same purpose, templates are based on the notion of

[8] http://dlt.gibbi.com.
[9] http://www.mat.unical.it/dlv-complex.

generalized quantifier, and allow more versatile usage. Syntax and semantics of template atoms are described in [8].

7 Related Work and Conclusions

Stable vs well-founded semantics. FAS programs have some peculiar differences with respect to the original F-logic. Importantly, while well-founded semantics [14] is at the basis of the nonmonotonic semantics of F-logic, FAS programs live under stable model semantics. The two semantics are complementary in several respects. The well-founded semantics is preferable in terms of computational costs: at the same time, this limits expressiveness with respect to the stable model semantics, which for disjunctive programs can express any query in the computational class Σ_2^p.
On the other hand, the well-founded semantics is three-valued. Having a third truth value as first class citizen of the language is an advantage in several scenarios, such as just in the case of object inheritance. Indeed, the undefined value is exploited in F-Logic when inheritance conflicts can not be solved with a clear truth value. Note, however, that the stable model semantics gives finer grained details in situations in which the well-founded semantics leaves truth values undefined. The reader can find a thorough comparison of the two semantics in [14].
FAS answer sets should not be confused with the notion of *stable object model* given in [33].

Semantic Web languages. Since F-logic features a natural way for manipulating ontologies and web data, it has been investigated for a long as suitable basis for representing and reasoning on data on the web. The two main F-Logic systems Flora and Florid ([32, 21]) share with FAS programs the ability to work both on the level of concepts and attributes and on instances.
Several Semantic Web initiatives point to F-logic as rule-based language core, like SWSL ([1]) and WSML ([11]) which in its more powerful variants is based on F-logic layered on top of Description Logic [10].
F-logic has been investigated as a logical way to provide reasoning capability on top of RDF in the system TRIPLE ([30]) that has native support for contexts (called *models*), URIs and namespaces. It is possible also to personalize semantics either via rule axiomatization (e.g. one can simulate RDFS reasoning by means of TRIPLE rules) or by means of interfacing external reasoners. The semantics of the full TRIPLE language has not been clearly formalized: its positive, non-higher order fragment coincides with Horn logic.
The possibility to define custom rule set for specifying the semantics which best fits the concrete application context is also allowed in OWLIM ([18]).

Answer Set Programming Several works share some point in common with this paper in the field of Answer Set Programming. An inspiring first definition of F-logic under stable model semantics can be found in [10]. The fragment considered focuses on first order F-logic with class hierarchies, and do not explicitly axiomatize structural inheritance with constructive semantics and single valued attributes. Higher order reasoning is present in dlvhex [12]. Contexts were investigated under stable model semantics also in [23]. In this setting, context atoms are exploited to give meaning to a form of scoped negation, useful in Semantic

Web applications where data sources with complete knowledge need to be integrated with sources expected to work under Open World Assumption. Similarly to our work, multi-context systems of [4] are used in order to define hybrid system with a logic of choice. Contexts can transfer knowledge each other by means of *bridge rules*, while in our setting it is not necessary a clear distinction between knowledge bases and bridge rules.
Nested attribute expressions behave like nested expressions as in [20], although we do not allow the use of negation in the head of rules. A different approach to nonmonotonic inheritance in the context of stable model semantics was proposed in [5], in which modules (which can be overridden each other) are associated with each object, and objects are partially sorted by an *isa* relation. The idea of defining an object-oriented modeling language under stable model semantics has been also subject of research in [26] and [25].
As a matter of future research, the authors plan to investigate thoroughly about the relationship between F-logic under well-founded semantics and similar formalizations of non-monotonic inheritance under stable model semantics. Also, the usage of arbitrarily nested molecules, including negation as failure, deserve further investigation.

References

1. Battle S. et al. Semantic Web Services Language. http://www.w3.org/Submission/SWSF-SWSL/.
2. Harold Boley and Michael Kifer. Rif core design. *W3C Editor's Draft*, 2007.
3. Harold Boley, Michael Kifer, Paula-Lavinia Pătrânjan, and Axel Polleres. Rule interchange on the web. In *Reasoning Web 2007*. LNCS 4636, pages 269–309.
4. Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *AAAI*, pages 385–390, 2007.
5. Francesco Buccafurri, Wolfgang Faber, and Nicola Leone. Disjunctive Logic Programs with Inheritance. *TPLP*, 2(3), May 2002.
6. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE TKDE*, 12(5):845–860, 2000.
7. Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: Theory and implementation. *Unpublished*, 2008.
8. Francesco Calimeri and Giovambattista Ianni. Template programs for disjunctive logic programming: An operational semantics. *AI Communications*, 19(3):193–206, 2006.
9. Jos de Bruijn et al. WSMO Final Draft, 2005. http://www.wsmo.org/TR/d2/v1.2/.
10. Jos de Bruijn and Stijn Heymans. Translating ontologies from predicate-based to frame-based languages. In *RuleML*, pages 7–16, 2006.
11. Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language wsml: An overview. In *ESWC*, pages 590–604, 2006.
12. Thomas Eiter, Giovambattista Ianni, Hans Tompits, and Roman Schindlauer. A uniform integration of higher-order reasoning and external evaluations in answer set programming. In *IJCAI*, pages 90–96, 2005.
13. Thomas Eiter, Giovambattista Ianni, Hans Tompits, and Roman Schindlauer. Effective Integration of Declarative Rules with External Evaluations for Semantic Web Reasoning. In *ESWC*, pages 273–287, 2006.
14. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, 1991.

15. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
16. Hasan M. Jamil. Implementing abstract objects with inheritance in datalog$^{\neg}$. In *VLDB*, pages 56–65, 1997.
17. Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.
18. Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. Owlim - a pragmatic semantic repository for OWL. In *WISE Workshops*, pages 182–192, 2005.
19. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM TOCL*, 7(3):499–562, 2006.
20. Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested Expressions in Logic Programs. *AMAI*, 25(3–4):369–389, 1999.
21. Bertram Ludäscher et al. Managing semistructured data with florid: A deductive object-oriented perspective. *Inf. Syst.*, 23(8):589–613, 1998.
22. Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal deductive systems for rdf. In *ESWC*, pages 53–67, 2007.
23. Axel Polleres, Cristina Feier, and Andreas Harth. Rules with contextually scoped negation. In *ESWC*, pages 332–347, 2006.
24. Teodor C. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.
25. Francesco Ricca et al. OntoDLV: an ASP-based System for Enterprise Ontologies. *Journal of Logic and Computation*, 2008. Forthcoming.
26. Francesco Ricca and Nicola Leone. Disjunctive logic programming with types and objects: The dlv^{+} system. *J. Applied Logic*, 5(3):545–573, 2007.
27. Dumitru Roman et al. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.
28. Riccardo Rosati. Dl+log: Tight integration of description logics and disjunctive datalog. In *KR*, pages 68–78, 2006.
29. Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002.
30. Michael Sintek and Stefan Decker. TRIPLE - an RDF query, inference, and transformation language. In: ISWC, pages 364–378 , 2002.
31. RDF Core Working Group. The Resource Description Framework., 2006. http://www.w3.org/RDF/.
32. G. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *CoopIS/DOA/ODBASE*, pages 671–688, 2003.
33. Guizhen Yang and Michael Kifer. Inheritance in Rule-Based Frame Systems: Semantics and Inference. *Journal on Data Semantics*, 7:79–135, 2006.