F-Logic#: Loosely Coupling F-Logic Rules and Ontologies[⋆]

Stijn Heymans[1], Roman Korf[2], Michael Erdmann[2], Jörg Pührer[1], and Thomas Eiter[1]

[1] Knowledge-Based Systems Group, Institute of Information Systems
Vienna University of Technology
Favoritenstrasse 9-11, A-1040 Vienna, Austria
{heymans,puehrer,eiter}@kr.tuwien.ac.at
[2] ontoprise GmbH
An der RaumFabrik 29, D-76227 Karlsruhe, Germany
{korf,erdmann}@ontoprise.de

Abstract. Although F-Logic rules received considerable attention in the development of the W3C's Rule Interchange Format (RIF), they have not been studied in the context of integrating rules with Description Logic ontologies. This paper makes steps in mending this by defining F-Logic# knowledge bases, a framework that provides a loosely coupled approach to integrating F-Logic rules and ontologies by allowing rules to query the ontology using *external atoms*.
We investigate the semantical properties of this framework and define a stratified fragment that allows for fast reasoning – a necessity on a Web with large amounts of data. We define the corresponding RIF dialect for F-Logic#, thus setting it firmly in a Web context. Finally, we show how to extend the F-Logic rule engine OntoBroker towards reasoning with F-Logic#, enabling as such a first commercial implementation for loosely coupled ontologies and rules.

1 Introduction

How to combine ontologies and rules is by now well-investigated theoretically with tightly-coupled approaches such as *Description Logic Programs* [12, 18][3], *DL-safe rules* [19], *r-hybrid knowledge bases* [26], $\mathcal{DL}+log$ [25], and *Description Logic Rules* [16], as well as loosely coupled approaches such as *dl-programs* [9]. Such approaches usually consider the integration of some Description Logic (DL) [1] (as the ontology language) and some Logic Programming (LP) paradigm (as the rule language). The main technical difficulties with such an integration are the different premises on which those two knowledge representation formalisms are built: an open domain assumption and a classical semantics for Description Logics, and a closed domain assumption and a minimal model semantics for Logic Programming.

In the Semantic Web context, two standards are arising for ontology languages and rule languages: the web ontology language *OWL 2* [24] and the *Rule Interchange Format* RIF [4]. From a Semantic Web practice vantage point, the integration of ontologies

[⋆] This work is partially supported by the European Commission under the project OntoRule (IST-2009-231875).

[3] Note that even though the approaches of [12] and [18] carry the same name, they are different.

and rules is thus concerned with the integration of OWL 2 ontologies and rules in some RIF dialect. For example, [7] investigates the compatibility of the RIF Basic Logic Dialect [3]. This dialect, however, does not allow for *negation as failure*, a construct common in most LP paradigms and responsible for its expressive non-monotonic features.

F-Logic Programming is an expressive rule-based formalism based on F-Logic [15] that allows for object-oriented constructs and higher-order features, as well negation as failure. Although F-Logic Programming takes a prominent place in the development of RIF – RIF supports frame terms for example – and non-commercial F-Logic engines such as FLORA-2 [27] and commercial engines such as OntoBroker [20] are available, little effort has been directed to an integration of F-Logic rules with DL ontologies.

In this paper, we take a first step in mending this by defining F-Logic# knowledge bases, a loosely coupled approach for integrating ontologies with F-Logic rules. Similarly to *dl-programs*, with which F-Logic# shares its basic mechanism of loose coupling, we are not only able to query the ontology but to also to send results from the program up to the ontology for further deductions, effectively enabling a bidirectional flow of information. In practice, such a loose coupling is often enough as witnessed by several use cases in the OntoRule project[4]. Moreover, it allows for significant re-use of existing implementations on the rule and on the ontology side, as well as for easier management of ontologies and rules as they can be both maintained independently.
The main contributions of the paper can be summarized as follows.

- In Section 3 we introduce F-Logic# knowledge bases, discuss semantical properties, and provide computational results for the basic reasoning tasks. We define what it means for an F-Logic# knowledge bases to be stratified in order to have an efficient fragment with an intuitive reasoning procedure that can be integrated with existing rule reasoner implementations. For the semantical definition of F-Logic# knowledge bases we adhere to the principles of dl-programs, but we address several F-Logic related issues, e.g., frame terms, treating them as first-order citizens of the language.
- We define F-Logic# as a RIF dialect in Section 4, by a specification of the RIF Framework for Logical Dialects [4], thus ensuring compatibility with upcoming rule standards and paving the way for different implementation of F-Logic# based on a common syntax.
- We describe the implementation of F-Logic# knowledge bases in Section 5 using OntoBroker [20] – the first recent operationalization of a loosely coupled approach for integration ontologies and rules in a commercial rule engine. The flexibility of the framework is apparent, as this implementation extends the theory by using SPARQL [22] queries instead of the atomic queries that we define in F-Logic#.

2 Preliminaries: F-Logic and F-Logic Programs

F-Logic F-Logic [15] has constructs to specify methods and generalization/specialization and instantiation relationships. The syntax has some seemingly higher-order features,

[4] http://ontorule-project.eu/

namely, the same identifier can be used for a class, an instance, and a method. However, the semantics of F-Logic is strictly first-order. To simplify matters, we do not consider parametrized methods, functional (single-valued) methods, inheritable methods, and compound molecules.

We take the signature of an F-language $\mathcal{L}$ to be of the form $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ with $\mathcal{F}$ constant symbols, disjoint from $\mathcal{P}$, the predicate symbols, each with an associated arity $n \geqslant 0$. Let $\mathcal{V}$ be a set of variable symbols. Terms and atomic formulas are then as usual: $t \in \mathcal{V} \cup \mathcal{F}$ is a term and $\top, \bot, p(t_1, \ldots, t_n)$, $t_1 = t_2$ are atomic formulas, with $p \in \mathcal{P}$ an n-ary predicate symbol, and $t_1, \ldots, t_n$ terms.

A molecule in F-Logic is one of the following: (i) an *is-a* assertion of the form $t_1 : t_2$, which states that an individual t_1 is of the type t_2, or (ii) a *data molecule* of the form $t_1[t_2 \twoheadrightarrow t_3]$, with t_1, t_2, t_3 terms, which states that an individual t_1 has an attribute t_2 with the value t_3. An F-Logic atom or molecule is *ground* if it does not contain variables. As in this paper we only need atoms and molecules, we refer the reader for the usual definition of formulas in an F-language to [15].

The semantics is given by *F-structures* which are tuples $\mathbf{I} = \langle U, \in_U, \mathbf{I}_F, \mathbf{I}_{\twoheadrightarrow}, \mathbf{I}_P \rangle$, where U is a non-empty set and $\in_U$ is a binary relation over U. A constant symbol $c \in \mathcal{F}$ is interpreted as an element in the domain U: $\mathbf{I}_F(c) \in U$. An n-ary predicate symbol $p \in \mathcal{P}$ is interpreted as a relation over the domain U: $\mathbf{I}_P(p) \subseteq U^n$. $\mathbf{I}_{\twoheadrightarrow}$ associates a binary relation over U with each $k \in U$: $\mathbf{I}_{\twoheadrightarrow}(k) \subseteq U \times U$. Variable assignments are defined in the usual way.

A Herbrand F-structure over a signature Σ is an F-structure $\mathbf{I} = \langle U, \in_U, \mathbf{I}_F, \mathbf{I}_{\twoheadrightarrow}, \mathbf{I}_P \rangle$ such that $U = \mathcal{F}$ for $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ and such that for any constant c, $\mathbf{I}_F(c) = c$. As an abuse of notation, for Herbrand structures we use $\mathbf{I}$ to denote both the structure and the set of ground molecules and atomic formulas satisfied in the structure.

Given an F-structure $\mathbf{I}$, a variable assignment B, and a term t of $\mathcal{L}$, $t^{\mathbf{I},B}$ is defined as: $X^{\mathbf{I},B} = X^B$ for a variable symbol X and $c^{\mathbf{I},B} = \mathbf{I}_F(c)$ for a constant symbol c. Satisfaction of atomic formulas and molecules ϕ in $\mathbf{I}$, given the variable assignment B, denoted $(\mathbf{I}, B) \models_{\mathsf{f}} \phi$, is defined such that $(\mathbf{I}, B) \models_{\mathsf{f}} \top$,$(\mathbf{I}, B) \not\models_{\mathsf{f}} \bot$, $(\mathbf{I}, B) \models_{\mathsf{f}} p(t_1, \ldots, t_n)$ iff $(t_1^{\mathbf{I},B}, \ldots, t_n^{\mathbf{I},B}) \in \mathbf{I}_P(p)$, $(\mathbf{I}, B) \models_{\mathsf{f}} t_1 = t_2$ iff $t_1^{\mathbf{I},B} = t_2^{\mathbf{I},B}$, $(\mathbf{I}, B) \models_{\mathsf{f}} t_1 : t_2$ iff $t_1^{\mathbf{I},B} \in_U t_2^{\mathbf{I},B}$, and $(\mathbf{I}, B) \models_{\mathsf{f}} t_1[t_2 \twoheadrightarrow t_3]$ iff $(t_1^{\mathbf{I},B}, t_3^{\mathbf{I},B}) \in \mathbf{I}_{\twoheadrightarrow}(t_2^{\mathbf{I},B})$. Note that for ground atomic formulas (atoms) and molecules ϕ, we will usually write $\mathbf{I} \models_{\mathsf{f}} \phi$.

F-Logic Programs We follow the definitions of [5]. *Rules* r are of the form

$$h \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n \tag{1}$$

with equality-free atoms or molecules $h, b_1, \ldots, b_m, c_1, \ldots, c_n$; $H(r) = h$ is the *head* of r, $B^+(r) = \{b_1, \ldots, b_m\}$ is the *positive body* of r and $B^-(r) = \{c_1, \ldots, c_n\}$ is the *negative body* of r. If $B^-(r) = \emptyset$ we call the rule *positive*. A *(F-Logic) Program* is a set of rules of the form (1). It is positive if all its rules are positive. Rules and programs are ground if they do not contain variables. A ground rule with an empty body is called a *fact*. A Herbrand F-structure $\mathbf{I}$ *models* a ground rule r, denoted $\mathbf{I} \models_{\mathsf{f}} r$, if whenever $B^+(r) \subseteq \mathbf{I}$ and $B^-(r) \cap \mathbf{I} = \emptyset$ then $h \in \mathbf{I}$. It is a *model* of a ground program P, denoted $\mathbf{I} \models_{\mathsf{f}} P$, if it models all rules $r \in P$.

The signature of a program P is an F-Logic signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ that contains the constants and predicate symbols from P. We denote with $gr_{\mathcal{F}}(P)$, the grounding of P

with constants $\mathcal{F}$, i.e., the ground rules originating from rules in P by replacing, per rule, each variable by each possible combination of constants in $\mathcal{F}$. If $\mathcal{F}$ is clear from the context, we will usually write $gr(P)$. For a non-ground program P with signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ and a Herbrand F-structure $\mathbf{I}$ over Σ, we say that $\mathbf{I}$ is a model of P, denoted $\mathbf{I} \models_{\mathsf{f}} P$, if $\mathbf{I}$ is a model of $gr_{\mathcal{F}}(P)$. It is minimal if there is no Herbrand F-structure $\mathbf{J} \subset \mathbf{I}$ over Σ that is a model of P. Note that, as in usual Logic Programming approaches, if P is positive, then there is a unique minimal model of P.[5]

We call a Herbrand F-structure $\mathbf{I}$ a *stable model* of P iff $\mathbf{I}$ is the minimal model of $gr(P)^{\mathbf{I}}$, where $gr(P)^{\mathbf{I}}$ is the GL-reduct [11]: remove all rules from $gr(P)$ that contain a $c \in B^-(r) \cap \mathbf{I}$ and subsequently remove all $not\ c$ from the remaining rules.

Example 1. Consider the following program $P = P_1 \cup P_2$ with P_1 the facts

$$\begin{array}{lcr} f1 : flight & f1[from \twoheadrightarrow paris] & f1[to \twoheadrightarrow vienna] \\ f2 : flight & f2[from \twoheadrightarrow vienna] & f2[to \twoheadrightarrow frankfurt] \end{array}$$

and P_2 the rules

$$\begin{array}{rl} directFlight(X, Y) \leftarrow & Z : flight, Z[from \twoheadrightarrow X], Z[to \twoheadrightarrow Y] \\ route(X, Y) \leftarrow & directFlight(X, Y) \\ route(X, Y) \leftarrow & directFlight(X, Z), route(Z, Y) \\ flightWithStops(X, Y) \leftarrow & route(X, Y), not\ directFlight(X, Y) \end{array}$$

The program P_1 indicates that there is a flight $f1$ from Paris to Vienna and $f2$ from Vienna to Frankfurt. The first rule in P_2 indicates that there is a direct flight from X to Y if there is a flight Z that goes from X to Y. A route is defined as the transitive closure of $directFlight$ and there is a flight with stops if there is a route but no direct flight.

The signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ of P is such that $\mathcal{F} = \{f1, flight, from, to, paris, vienna, f2, frankfurt\}$ and $\mathcal{P} = \{directFlight, route, flightWithStops\}$. A stable model $\mathbf{I}$ of this program contains the ground molecules P_1 since those are facts in P. It contains the atoms $directFlight(paris, vienna)$ and $directFlight(vienna, frankfurt)$ by satisfaction of the ground versions of the first rule. The *route* rules force $\mathbf{I}$ to include $route(paris, vienna)$, $route(vienna, frankfurt)$, and $route(paris, frankfurt)$ and by minimality of stable models, no more $route$ atoms are included such that finally only $flightWithStops(paris, frankfurt)$ is further included using the $flightWithStops$ rule.

Current F-Logic systems such as FLORA-2 [27] and OntoBroker [20] are dealing for efficiency reasons with rules under a well-founded semantics [10] instead of a stable model semantics. We focus for the expansion of the theory on the stable model semantics as (1) we are initially mainly interested, for efficiency reasons, in the stratified case, and in that case the well-founded and stable model semantics coincide (2) the well-founded semantics can be easily defined on top of the stable model semantics, such that

[5] Note that w.l.o.g we can include so-called *subclass molecules* $t_1 :: t_2$ in rules by axiomatizing them as in [5]:

$$\begin{array}{l} X :: Z \leftarrow X :: Y, Y :: Z \\ X : Z \leftarrow X : Y, Y :: Z \\ X :: X \leftarrow \end{array}$$

i.e., transitivity, inheritance of class membership, and reflexivity. Whenever we have subclass molecules we assume these 3 rules are implicitly present as well.

we choose to treat external atoms from the perspective of the more expressive stable model semantics. Indeed, once the concept of a GL-reduct is defined, one can, using the γ operator [2], easily define the well-founded semantics: for a Herbrand structure $\mathbf{I}$ and a program P, let $\gamma_P(\mathbf{I})$ be the least model of $gr(P)^{\mathbf{I}}$ and $\gamma_P^2(\mathbf{I}) = \gamma_P(\gamma_P(\mathbf{I}))$, i.e., applying the γ operator twice. Then the set of *well-founded* atoms of P, denoted WFS_P, is exactly the least fixed point of γ_P^2. Note that the implementation described in Section 5 works for stratified F-Logic# knowledge bases, and thus for both the well-founded as well as the stable model semantics.

3 F-Logic# Programming: F-Logic Programs with External Atoms

In this section, we extend F-Logic programs with so-called *external atoms*, in order to, from the rules, query ontologies as well as feed conclusions to the ontology.

In the following, we assume a very general concept of an ontology language[6]: an ontology language $\mathcal{L}$ is defined such that a theory (i.e., a set of formulas) Φ in $\mathcal{L}$ has a signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$. Furthermore, $\mathcal{L}$ should define an entailment relation $\models_{\mathcal{L}}$ such that for a theory Φ and a formula ϕ in the language $\mathcal{L}$, $\Phi \models_{\mathcal{L}} \phi$ is well-defined, and is a Boolean decidable function. Moreover, we assume atoms a and negated atoms $\neg a$ over the signature Σ are part of the language. For example, if $\mathcal{L}$ is the language of first-order logic, $\models_{\mathcal{L}}$ is the usual first-order entailment $\models$. We impose that the signature contains only 0-ary function symbols (constants) – the same restriction as for F-Logic programs.

Syntax Let $\mathcal{L}$ be the ontology language and $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ the signature under consideration. An *external atom* over Σ is of the form

$$\#[S_1 op_1 P_1, \ldots, S_m op_m P_m;\ Q](\boldsymbol{t}) \tag{2}$$

where for $1 \leqslant i \leqslant m$, $op_i \in \{\uplus, \cap\!\!\!\!\!\cdot\}$, $Q \in \mathcal{P}$ is an n-ary predicate, $\boldsymbol{t}$ are n terms over Σ, $S_i \in \mathcal{P}$, and either[7]

- $P_i \in \mathcal{P}$, such that both S_i and P_i are k-ary predicate symbols,
- $P_i = \twoheadrightarrow p_i$ for $p_i \in \mathcal{F}$ and S_i is a binary predicate, or
- $P_i = {:}p_i$ for $p_i \in \mathcal{F}$ and S_i is a unary predicate.

Intuitively, $op_i = \uplus$ increases S_i by the extension of P_i, while $op_i = \cap\!\!\!\!\!\cdot$ constrains S_i to P_i. The symbols $\twoheadrightarrow p$ and ${:}p$ indicate that p should be interpreted as an attribute name or a type name, as in F-Logic. We call the P_i, $1 \leqslant i \leqslant m$, the *input predicates* of the external atom. Note that the external atoms do not necessarily need to have input predicates, i.e., we allow for external atoms of the form $\#[;Q](\boldsymbol{t})$.

A *ground* external atom is an external atom (2) such that none of the terms $\boldsymbol{t}$ are variables. An *F-Logic#-rule* r w.r.t. the signature Σ is of the form (1) where the body atoms can additionally be external atoms over Σ.

[6] The set of F-Logic programs could be considered an ontology language as well.

[7] Strictly speaking, we do not need a different notation $\twoheadrightarrow p_i$ and ${:}p_i$ as the function symbols and predicate symbols are disjoint, however, for clarity we thought it to be useful.

A set of such F-Logic#-rules is called an *F-Logic#-program*. An F-Logic#-rule is ground if all its components (atoms, molecules, and external atoms) are ground and an F-Logic#-program is ground if all its rules are ground.

We denote with $gr_{\mathcal{F}}(P)$ the grounding of an F-Logic#-program P with the constants $\mathcal{F}$ of the signature Σ, i.e., every rule is replaced by its ground instantiations with $\mathcal{F}$, defined as usual. We will write $gr(P)$ if $\mathcal{F}$ is clear from the context.

An *F-Logic# knowledge base (KB)* w.r.t. Σ is a tuple $\mathcal{KB} = \langle \Phi, P \rangle$ where Φ is a theory in $\mathcal{L}$ with signature Σ and P is an F-Logic#-program w.r.t. Σ.[8]

Example 2. Take an F-Logic# KB $\mathcal{KB} = \langle \Phi, P \rangle$ with a Description Logic (DL) KB Φ:

$$\begin{array}{rl} \mathit{localFlight}(\mathit{bolzano}, \mathit{vienna}) & \qquad \mathit{localFlight}(\mathit{vienna}, \mathit{bolzano}) \\ \mathit{linkedtoHub} \equiv \exists \mathit{localFlight}.\mathit{hub} & \end{array}$$

that defines two local flights and a concept *linkedtoHub* that is equivalent with all objects that are connected via *localFlight* to a *hub*. Take the F-Logic#-program $P = P_1' \cup P_3$ where P_1' is P_1 from Example 1 extended with $\mathit{frankfurt} :\mathit{hubCity}$ and P_3 is

$$\begin{array}{rl} \mathit{directFlight}(X, Y) \leftarrow & Z :\mathit{flight}, Z[\mathit{from} \twoheadrightarrow X], Z[\mathit{to} \twoheadrightarrow Y] \\ \mathit{route}(X, Y) \leftarrow & \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Y) \\ \mathit{route}(X, Y) \leftarrow & \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Z), \mathit{route}(Z, Y) \\ \mathit{flightWithStops}(X, Y) \leftarrow & \mathit{route}(X, Y), \\ & \mathit{not}\ \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Y) \\ X :\mathit{hubReachable} \leftarrow & \#[\mathit{localFlight} \uplus \mathit{route}, \mathit{hub} \uplus :\mathit{hubCity}; \mathit{linkedtoHub}](X) \end{array}$$

Intuitively, the external atom $\#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Y)$ extends local flights in the ontology Φ with direct flights from P, subsequently queries what the entailed local flights under this extension are, and further uses this to calculate the routes and the flights that have stops.

The last rule defines the cities that can reach a hub in one or more steps. We thus query the ontology for all objects that are part of *linkedtoHub* by extending the local flights in the ontology with the routes and the concept *hub* with the objects that are of type *hubCity*. Note the ':' in front of *hubCity* to indicate that it is an F-Logic type. While the *route* rules in P are using a mapping from *directFlight*s to *localFlight*s (i.e., the modeler assumes these are actually the same roles/predicates), in this last rule, one extends the meaning of local flights to also mean routes, enabling a flexible interpretation of ontologies by rules, while still relying on the ontology definitions (of *linkedtoHub*).

Semantics Let $\mathcal{KB} = \langle \Phi, P \rangle$ be an F-Logic# KB with signature Σ. We assume in the following that $\twoheadrightarrow p(a, b)$ and $:p(a)$ is shorthand for $a[p \twoheadrightarrow b]$ and $a :p$, respectively.

Let $\mathbf{I}$ be a Herbrand F-structure over $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$, then, $\mathbf{I}$ models a ground F-Logic#-atom $a = \#[S_1 op_1 P_1, \ldots, S_m op_m P_m; Q](\boldsymbol{u})$ under Φ, denoted $\mathbf{I} \models_{\mathsf{f},\Phi} a$, iff $\Phi \cup_{i=1}^{m} A_i(\mathbf{I}) \models_{\mathcal{L}} Q(\boldsymbol{u})$ where

[8] When the combined signature is clear from the context we will usually omit it. Further note that we do not make a distinction between the ontology's signature and the rule component's signature.

– $A_i(\mathbf{I}) = \{S_i(\boldsymbol{e}) \mid \mathbf{I} \models_{\mathsf{f}} P_i(\boldsymbol{e})\}$, for $op_i = \uplus$, and
– $A_i(\mathbf{I}) = \{\neg S_i(\boldsymbol{e}) \mid \mathbf{I} \not\models_{\mathsf{f}} P_i(\boldsymbol{e})\}$, for $op_i = \Cap$,

where $\boldsymbol{e}$ is a vector of constants from $\mathcal{F}$ with length conforming to the type of the predicate S_i, $1 \leqslant i \leqslant m$, at hand. For ground atomic formulas and molecules a, we define satisfaction under Φ, denoted $\mathbf{I} \models_{\mathsf{f},\Phi} a$, as satisfaction $\mathbf{I} \models_{\mathsf{f}} a$.

We say that $\mathbf{I}$ is a *model* under Φ of a ground F-Logic#-rule r of the form (1), denoted $\mathbf{I} \models_{\mathsf{f},\Phi} r$, if $\mathbf{I} \models_{\mathsf{f},\Phi} h$ whenever $\mathbf{I} \models_{\mathsf{f},\Phi} b_j$ for all $1 \leqslant j \leqslant m$ and $\mathbf{I} \not\models_{\mathsf{f},\Phi} c_k$ for all $1 \leqslant k \leqslant n$. Finally, $\mathbf{I}$ is a model of an F-Logic# KB $\mathcal{KB} = \langle \Phi, P \rangle$, denoted $\mathbf{I} \models_{\mathsf{f}} \mathcal{KB}$, if $\mathbf{I} \models_{\mathsf{f},\Phi} r$ for each $r \in gr(P)$. The *projection* of an F-Logic# KB $\mathcal{KB} = \langle \Phi, P \rangle$ where P is ground, with respect to a Herbrand F-structure $\mathbf{I}$, denoted $\Pi(\mathcal{KB}, \mathbf{I})$, is an F-Logic#-program obtained from P as follows. For every rule r in P,

– if there is an external atom $a \in B^+(r)$ such that $\mathbf{I} \not\models_{\mathsf{f},\Phi} a$, or an external atom $a \in B^-(r)$ such that $\mathbf{I} \models_{\mathsf{f},\Phi} a$, then remove r,
– otherwise, delete all external atoms from r.

Intuitively, the projection "evaluates" the set of rules with respect to $\mathbf{I}$ by removing (evaluating) rules and external atoms consistently with $\mathbf{I}$ and Φ.

Definition 1. *Let $\mathcal{KB} = \langle \Phi, P \rangle$ be an F-Logic# KB over Σ. Then, a Herbrand F-structure $\mathbf{I}$ over Σ is an* NM-model *of $\mathcal{KB}$, denoted $\mathbf{I} \models_s \mathcal{KB}$, if $\mathbf{I}$ is a stable model of $\Pi(\langle \Phi, gr(P) \rangle, \mathbf{I})$.*

In other words, the semantics of an F-Logic# KB is given by first grounding the program part, then projecting away the external atoms w.r.t. some guessed Herbrand F-structure, in correspondence with the ontology part, and finally, calculating the stable model of the resulting F-Logic program (i.e., containing no external atoms) and verifying that it corresponds to the initial Herbrand F-structure (hence the *stability*).

Example 3. Consider the combined KB $\mathcal{KB} = \langle \Phi, P \rangle$ from Example 2 with the signature consisting of exactly the constants and predicates appearing in $\mathcal{KB}$. Any stable model $\mathbf{I}$ will contain P_1' from Example 2 and using the first rule regarding direct flights, $\mathbf{I}$ contains the atoms $directFlight(paris, vienna)$ and $directFlight(vienna, frankfurt)$.

Further, $\mathbf{I} \models_{\mathsf{f},\Phi} \#[localFlight \uplus directFlight; localFlight](u_1, u_2)$ for (u_1, u_2) that are $(paris, vienna)$, $(vienna, frankfurt)$, $(vienna, bolzano)$, and $(bolzano, vienna)$, as $\Phi \cup A(\mathbf{I}) \models_{\mathcal{L}} localFlight(u_1, u_2)$ where

$$A(\mathbf{I}) = \{localFlight(paris, vienna), localFlight(vienna, frankfurt).\}$$

With the first *route* rule, we have that $route(paris, vienna)$, $route(vienna, frankfurt)$, $route(vienna, bolzano)$ and $route(bolzano, vienna)$ are in $\mathbf{I}$. From the second *route* rule, we have $route(paris, frankfurt)$, $route(paris, bolzano)$, $route(vienna, vienna)$, $route(bolzano, bolzano)$, and $route(bolzano, frankfurt)$ are in $\mathbf{I}$. The flights with stops are then all routes except the $route(u_1, u_2)$s as these are direct flights.

Finally, $\mathbf{I} \models_{\mathsf{f},\Phi} \#[localFlight \uplus route, hub \uplus :hubCity; linkedtoHub](v)$ for v that is $vienna$, $paris$, or $bolzano$ as $\Phi \cup A_1(\mathbf{I}) \cup A_2(\mathbf{I}) \models_{\mathcal{L}} linkedtoHub(v)$ where $A_1(\mathbf{I})$ is the *localFlights* that are already in Φ and the *localFlights* that were matched with the *routes* from $\mathbf{I}$ and $A_2(\mathbf{I})$ consists of $hub(frankfurt)$ since $\mathbf{I} \models_{\mathsf{f}} frankfurt :hubCity$. Thus, $vienna :hubReachable$, $paris :hubReachable$, and $bolzano :hubReachable$ are in $\mathbf{I}$.

Note that NM-models are not necessarily minimal.

Example 4. Take an F-Logic# KB $\mathcal{KB} = \langle \Phi, P \rangle$, with $\Phi = \emptyset$ and P the rule $p(X) \leftarrow \#[p \uplus p; p](X)$. With a signature Σ where $\mathcal{F} = \{a\}$ and $\mathcal{P} = \{p\}$, one has that both $\{p(a)\}$ and $\emptyset$ are NM-models of $\mathcal{KB}$.

This example additionally shows that one can give a model-theoretical meaning to predicates appearing in rules, i.e., you have the choice (possibly constrained by other rules or the ontology) to include $p(a)$ in the NM-model or not.

If checking $\models_{\mathcal{L}}$ is in a complexity class $\mathcal{C}$, one can calculate the NM-model in nondeterministic exponential time w.r.t. the size of the KB, using an oracle in $\mathcal{C}$.

Theorem 1. *Let $\mathcal{KB} = \langle \Phi, P \rangle$ be an F-Logic# KB where checking $\Phi \models_{\mathcal{L}} \phi$ is in $\mathcal{C}$. Then, the NM-model of $\mathcal{KB}$ can be computed by a nondeterministic Turing machine in exponential time in the size of $\mathcal{KB}$, using an oracle in $\mathcal{C}$.*

Proof. (Sketch) Intuitively, one can guess an interpretation $\mathbf{I}$ for $\mathcal{KB} = \langle \Phi, P \rangle$ in polynomial time in the size of $\mathcal{KB}$. The grounding of P has exponential size w.r.t. the size of P. Calculating the projection of $gr(P)$ can then be done in exponential time in the size of P (by running through $gr(P)$) and an oracle in $\mathcal{C}$. Finally, calculating the GL-reduct is again polynomial in $gr(P)$ and calculating the minimal model of a positive F-Logic program is polynomial in the size of $gr(P)$ as well. $\Box$

Reasoning with Stratified F-Logic# Knowledge Bases We are interested in more efficient fragments of F-Logic# KBs, i.e., fragments for which one can calculate the NM-model faster than in nondeterministic exponential time with an oracle in $\mathcal{C}$. To this purpose, we introduce *stratified* F-Logic# KBs.

For an F-Logic# rule r, let $\#(r)$ be the external atoms in r; for an F-Logic#-program P, let $\#(P) = \cup\{\#(r) \mid r \in P\}$ be the external atoms in P.

Let $\mathcal{KB} = \langle \Phi, P \rangle$ be an F-Logic# KB over a signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$. We define a *stratified* F-Logic# KB by means of a *stratification function* $\lambda : \mathcal{P} \cup \{:p, \twoheadrightarrow p \mid p \in \mathcal{F}\} \cup \{:, \twoheadrightarrow\} \rightarrow \{0, \ldots, k\}$, $k \geqslant 0$. We extend the definition of this stratification function towards (external) atoms and molecules as follows:

- for atoms, $\lambda(p(\boldsymbol{t})) = \lambda(p)$ for $p \in \mathcal{P}$,
- for is-a assertions,
$$\lambda(t_1 : t_2) = \begin{cases} \lambda(:t_2) & \text{if } t_2 \in \mathcal{F} \\ \lambda(:) & \text{otherwise} \end{cases}$$
- for data molecules,
$$\lambda(t_1[t_2 \twoheadrightarrow t_3]) = \begin{cases} \lambda(\twoheadrightarrow t_2) & \text{if } t_2 \in \mathcal{F} \\ \lambda(\twoheadrightarrow) & \text{otherwise} \end{cases}$$
- for external atoms, $\lambda(\#[S_1 op_1 P_1, \ldots, S_m op_m P_m; Q](\boldsymbol{t})) = \max_{1 \leqslant i \leqslant m}\{\lambda(P_i)\}$.

Intuitively, the *level* of an atom is the same as the level of its predicate. The level of a molecule is the same as the level of its type name or attribute name in case it is ground and has the level of : and $\twoheadrightarrow$ otherwise. External atoms have a level that is the maximum level of the P_i that are used to send input the ontology.

Definition 2. *Let $\mathcal{KB} = \langle \Phi, P \rangle$ be an F-Logic# KB over a signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$. Then $\mathcal{KB}$ is* stratified *if there exists a stratification function λ such that, for each rule $r \in P$ of the form (1):*

- $\lambda(h) \geqslant b_i$, *for each* $1 \leqslant i \leqslant m$ *such that* $b_i \notin \#(r)$,
- $\lambda(h) > b_i$, *for each* $1 \leqslant i \leqslant m$ *such that* $b_i \in \#(r)$,
- $\lambda(h) > c_i$, *for each* $1 \leqslant i \leqslant n$.

Thus the level of the head has to be greater or equal than the body atoms/molecules that are not external, and strictly greater than the level of the external atoms, and, as usual, strictly greater than the negative atoms/molecules.[9]

We can then write a stratified KB $\mathcal{KB} = \langle \Phi, P \rangle$ as $\langle \Phi, (P_0, \ldots, P_k) \rangle$ where, for $0 \leqslant i \leqslant k$, $P_i = \{r \in P \mid \lambda(H(r)) = i\}$. The P_i are called the *strata* of $\mathcal{KB}$.

Example 5. Take the F-Logic# KB from Example 2. This is a stratified KB, that has, for example, the following strata:

$$\begin{array}{rl}
\mathbf{Q_2}: & \\
X \mathbin{:} \mathit{hubReachable} \leftarrow & \#[\mathit{localFlight} \uplus \mathit{route}, \mathit{hub} \uplus \mathbin{:}\mathit{hubCity}; \mathit{linkedtoHub}](X) \\
\hline
\mathbf{Q_1}: & \\
\mathit{route}(X, Y) \leftarrow & \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Y) \\
\mathit{route}(X, Y) \leftarrow & \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Z), \mathit{route}(Z, Y) \\
\mathit{flightWithStops}(X, Y) \leftarrow & \mathit{route}(X, Y), \\
 & \mathit{not}\ \#[\mathit{localFlight} \uplus \mathit{directFlight}; \mathit{localFlight}](X, Y) \\
\hline
\mathbf{Q_0}: & \\
 & \mathbf{P_1'}
\end{array}$$

We define the *iterative least model* of a ground stratified $\mathcal{KB} = \langle \Phi, (P_0, \ldots, P_k) \rangle$:

- Let $\mathbf{I}_0$ be the minimal model of $\Pi(\langle \phi, P_0 \rangle, \emptyset)$. Note that P_0 is positive and does not contain external atoms with input predicates since $\mathcal{KB}$ is stratified such that $\Pi(\langle \phi, P_0 \rangle, \emptyset)$ has a unique such $\mathbf{I}_0$.
- For $i > 0$, define $P_i(\mathbf{I}_{i-1})$ as the projection $\Pi(\langle \Phi, P_i \rangle, \mathbf{I}_{i-1})$ and let $\mathbf{I}_i$ be the minimal model of the GL-reduct $P_i(\mathbf{I}_{i-1})^{\mathbf{I}_{i-1}} \cup \{a \leftarrow \mid a \in \mathbf{I}_{i-1}\}$ (which is again positive and does not contain external atoms). Intuitively, in P_i we remove the external atoms using $\mathbf{I}_{i-1}$, followed by removing the negative atoms again w.r.t. $\mathbf{I}_{i-1}$ and subsequently taking $\mathbf{I}_{i-1}$ into account as facts.

The iterative least model of $\mathcal{KB} = \langle \Phi, (P_0, \ldots, P_k) \rangle$ is then, by definition, $\mathbf{I}_k$. By construction, the iterative least model of a ground stratified KB exists and is unique. For a stratified F-Logic# KB $\mathcal{KB} = \langle \Phi, (P_0, \ldots, P_k) \rangle$ that is not ground, we define the iterative least model as the iterative least model of the grounding $\langle \Phi, (gr(P_0), \ldots, gr(P_1)) \rangle$.

[9] In the presence of subclass molecules $t_1 \mathbin{::} t_2$ in the program P, we extend λ to be a function $\mathcal{P} \cup \{\mathbin{:}p, \twoheadrightarrow p, \mathbin{::}p \mid p \in \mathcal{F}\} \cup \{\mathbin{:}, \twoheadrightarrow, \mathbin{::}\} \to \{0, \ldots, k\}$ and its extension on subclass molecules as follows:

$$\lambda(t_1 \mathbin{::} t_2) = \begin{cases} \lambda(\mathbin{::}t_2) & \text{if } t_2 \in \mathcal{F} \\ \lambda(\mathbin{::}) & \text{otherwise} \end{cases}$$

Note that this is a similar definition as for *is-a* assertions. Due to the implicit presence of the 3 axiomatizing rules when subclass molecules are present, one can see that $\lambda(\mathbin{:}) \geqslant \lambda(\mathbin{::})$ for all stratification functions, due to the inheritance of class membership ($X \mathbin{:} Z \leftarrow X \mathbin{:} Y, Y \mathbin{::} Z$).

Theorem 2. *Let $\mathcal{KB}$ be a stratified F-Logic# KB over $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$. Then, $\mathbf{I}$ is the iterative least model of $\mathcal{KB}$ iff $\mathbf{I}$ is the unique NM-model of $\mathcal{KB}$.*

Corollary 1. *Let $\mathcal{KB} = \langle \Phi, P \rangle$ be a stratified F-Logic# KB over $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$. Then, it has a unique NM-model which corresponds to the iterative least model of $\mathcal{KB}$.*

One can thus use the procedure to construct the iterative least model of a stratified KB to calculate the unique NM-model of that KB. Note that Example 4 is not stratified as for any stratification function λ, $\lambda(p(X)) = \lambda(p) = \max\{\lambda(p)\} = \lambda(\#[p \uplus p; p](X))$, hence the two different NM-models.

The iterative least model calculation procedure gives us a direct means to analyze the complexity of calculating NM-models of stratified KBs. As in the non-stratified case, let $\mathcal{C}$ be the complexity of the ontology language, i.e., checking $\models_{\mathcal{L}}$ is in $\mathcal{C}$, and let $\mathcal{KB} = \langle \Phi, (P_0, \dots, P_k) \rangle$ be ground. Then,

- we can calculate $\Pi(\langle \phi, P_0 \rangle, \emptyset)$ with a linear number of calls (in the size of the P_0) to an oracle in $\mathcal{C}$ in order to remove the external atoms. The minimal model $\mathbf{I}_0$ of a positive program can be then calculated as usual in polynomial time, using an immediate consequence operator, see [6]. Thus, we can calculate $\mathbf{I}_0$ in polynomial time in the size of P_0, using an oracle in $\mathcal{C}$;
- for $i > 0$, we can calculate $P_i(\mathbf{I}_{i-1})$ again in polynomial time in the size of P_i, using an oracle in $\mathcal{C}$. The GL-reduct $P_i(\mathbf{I}_{i-1})^{\mathbf{I}_{i-1}}$ can be calculated in polynomial time in the size of P_i such that the minimal model of $P_i(\mathbf{I}_{i-1})^{\mathbf{I}_{i-1}} \cup \{a \leftarrow\mid a \in \mathbf{I}_{i-1}\}$ can again be calculated in polynomial time. Thus, we can calculate $\mathbf{I}_i$, $i > 0$, in polynomial time in the size of P_i, using an oracle in $\mathcal{C}$.

Combining the above, we have that the unique NM-model of $\mathcal{KB} = \langle \Phi, (P_0, \dots, P_k) \rangle$ can be calculated in polynomial time in the size of $\mathcal{KB}$, using an oracle in $\mathcal{C}$.

In the non-ground case, we have that, in general, grounding the stratified F-Logic#-program, requires exponential time. Thus, the unique NM-model of an unground $\mathcal{KB} = \langle \Phi, (P_0, \dots, P_k) \rangle$ can be calculated in exponential time using an oracle in $\mathcal{C}$.

Theorem 3. *Let $\mathcal{KB} = \langle \Phi, P \rangle$ be a stratified F-Logic# KB where checking $\Phi \models_{\mathcal{L}} \phi$ is in $\mathcal{C}$. Then, the NM-model of $\mathcal{KB}$ can be computed by a deterministic Turing machine in exponential time in the size of $\mathcal{KB}$, using an oracle in $\mathcal{C}$.*

Comparing this result to the non-stratified case in Theorem 1, one can see that in case of a stratified KB, computation is deterministic while for a non-stratified KB it is nondeterministic in exponential time, using an oracle in $\mathcal{C}$.

4 F-Logic# as a RIF Logical Dialect: RIF-F-Logic#

We relate F-Logic# KBs to the *RIF Framework for Logic Dialects (FLD)* and refer the reader to [4] for the definition of the RIF FLD.

For an F-Logic# KB $\mathcal{KB} = \langle \Phi, P \rangle$, we only define the language of P, i.e., F-Logic#-programs, as a RIF fragment, since Φ is a theory in some ontology language (which could be compatible with a RIF dialect, but not a priori). For the exchange of F-Logic# KBs, one would exchange 2 components: the component in the RIF fragment and the component in the ontology language (e.g., an OWL document [8]).

Syntax of RIF-F-Logic# We specialize the different parameters of the RIF Framework for Logic Dialects to obtain the RIF-F-Logic# dialect. We remove all *extension points* and we keep the RIF FLD *alphabet* specification, but leave out argument names `ArgNames`, connective symbols `Or` and `Neg`, the quantifier `Exists`, the symbols for representing lists `List` and `OpenList`, and the aggregate symbols.

The *signature set* of RIF-F-Logic# is defined as the signature set of the RIF Basic Logic Dialect (RIF BLD, [3, Section 6.1]) with removal of the signature of lists, functions, and external functions, and arrow expressions for predicates with named arguments from the predicate signature.

The RIF-F-Logic# dialect allows for the following *terms* (as in [4, 2.4]).

– Simple terms $t \in \texttt{Const}$ or $t \in \texttt{Var}$.
– Positional terms $t(t_1, \ldots, t_n)$ where t_i, $1 \leqslant i \leqslant n$, are simple terms and $t \in \texttt{Const}$, a term of predicate signature `p` (see [3, Section 6.1.3.c] for the definition of a predicate signature in the RIF Framework for Logic Dialects).
– Classification terms (membership and subclass terms, corresponding to what we called is-a assertions and subclass molecules resp.).
– Frame terms, corresponding to what we called data molecules.
– Externally defined terms $\texttt{External}(Q_{[S_1 op_1 P_1, \ldots, S_m op_m P_m]}(\boldsymbol{t})\ loc)$ corresponding to the external atoms defined in Section 3 and where $Q_{[S_1 op_1 P_1, \ldots, S_m op_m P_m]}$ is a predicate (and thus with the signature of predicates). loc indicates the location of the external ontology to be queried; we will usually omit it if the location is not relevant or if it is clear which ontology is being referred to. For each external term $\texttt{External}(Q_{[S_1 op_1 P_1, \ldots, S_m op_m P_m]}(\boldsymbol{t})\ loc)$, we define the corresponding external schema ([4, Section 2.5]), $(?X_1 \ldots ?X_n; Q_{[S_1 op_1 P_1, \ldots, S_m op_m P_m]}(?X_1 \ldots ?X_n); loc)$ where n is the number of terms in $\boldsymbol{t}$.
– As in RIF-BLD, no aggregate terms, module terms, or formula terms are allowed.

The embedding of external atoms in RIF is not fully satisfactory; one would like to avoid encoding information from the F-Logic# program as non-interpretable syntax such as a string of characters[10]. Indeed, the term t in the general $External(t\ loc)$ term definition of RIF-FLD [4, Section 2.4.8] is required to be a constant, positional term, a term with named arguments, an equality, a classification, or a frame term. However, one defines signatures for these terms that are fixed. In F-Logic# we indeed need that a frame term $t_1[t_2 \twoheadrightarrow t_3]$ is such that t_i, $1 \leqslant i \leqslant 3$, is an individual (a constant or a variable). However, if we would like to use frame terms in RIF-F-Logic# external terms – to query an external source – we would need to be able to use (input) predicates where the signature only allows for individuals. We see at the moment little alternatives on how to treat external atoms more elegantly, given the current framework of RIF-FLD.

We include the *symbol spaces* from RIF-BLD. Note that this extends the language F-Logic# compared to how we originally defined it, however, the RIF FLD requires at least the symbol spaces from RIF-BLD to be included in any dialect.

We support the following types of *formulas* (see RIF-BLD, [3, Section 6.1.4], for the definitions):

[10] We sticked to symbols, but one could define a machine-readable, albeit not semantical, variant.

– a *RIF-F-Logic# condition* is an atomic formula, default negation, or a conjunction of atomic formulas, default negations, and/or external atomic formulas,
– a *RIF-F-Logic# rule* is a universally quantified RIF-FLD rule where the conclusion is an atomic formula (that is not externally defined), the premise of the rule is a RIF-F-Logic# condition, and all free (non-quantified) variables in the rule must be quantified with `Forall` outside the rule,
– a *RIF-F-Logic# group* is a RIF-FLD group that contains only RIF-F-Logic# rules; a group thus corresponds to an F-Logic# program,
– a *RIF-F-Logic# document* is defined as a RIF-BLD document but referring to RIF-F-Logic# group formulas instead of RIF-BLD group formulas.

Semantics of RIF-F-Logic# The semantics of RIF-F-Logic# is the semantics of F-Logic#-programs as defined in Section 3. Note that the syntax of RIF-F-Logic# includes the datatypes of RIF-FLD, but neither the semantics nor the syntax defines F-Logic# datatypes. We leave this as future work, and will update the RIF embedding of F-Logic# accordingly when datatypes have been formally investigated in F-Logic#. Other constructs and definitions typical for RIF, such as the `Import` directive and the set of truth values are semantically defined as in RIF-BLD. Logical entailment is defined based on the intended semantics structures of F-Logic#, NM-models (see Definition 1).

5 Implementation of F-Logic# using OntoBroker

In this section, we demonstrate an implementation of F-Logic# for the ontology language OWL-DL [13] based on the OntoBroker [20] inference engine from ontoprise. OntoBroker consists of two reasoners: (i) OntoBroker F-logic, a very sophisticated and fast (cf. [17]) rule engine, and (ii) OntoBroker-OWL, an OWL-DL reasoner (the successor of KAON2[11], [14]) for a subset of OWL 2 [24].

Those two reasoners use the same API thus simplifying the implementation of the interfacing mechanisms as defined in F-Logic# significantly. Another advantage of using OntoBroker is the availability of *queryable* and *updatable* knowledge bases. Since we decided to use SPARQL [22] as a query language and currently there is no standardized update language available (SPARUL [23] is not yet widely accepted) we could benefit from the availability of an update API within OntoBroker-OWL. Note that updates to the ontology in F-Logic# are actually *temporary* updates, only valid within the scope of *one* query. Once the query has finished the updates must be *rolled back*, and parallel queries to the same knowledge base must also be shielded from those temporary facts: OntoBroker-OWL provides such a *session* mechanism.

We have introduced the #-operator to make ontological (OWL-DL) reasoning available from within logic programs with rules: $\#[S_1 op_1 P_1, \ldots, S_m op_m P_m; Q](\boldsymbol{t})$. In the OntoBroker implementation, we realize this operator using the new built-in predicate `dlaccess/6`:

```
dlaccess(
  mappings, facts, ontology-iri, query, query-result, server)
```

[11] http://kaon2.semanticweb.org/

and use some additional predicates from OntoBroker's standard library of predicates. The arguments of the predicate are shortly described here and afterward explained in more detail in the context of a concrete application of the predicate:

– `<mappings>`: A list of mappings from DL facts to F-logic facts. This argument represents the implementation of the list of operations of the form $S_i op_i P_i$ from the formal specification. Currently the implementation supports the $\uplus$ operator for F-logic methods, concepts and binary predicates. Other operators can be emulated easily under a stratified semantics.
– `<facts>`: A list of lists that populate the `<mappings>`, i.e. for each defined mapping in the `mappings` argument `facts` contains one list with all values that will be used to populate the "mapping target" before posing the query.
– `<ontology-iri>`: Since the OntoBroker reasoning server can store and query many different ontologies at the same time we must uniquely identify the ontology which we want to query.
– `<query>`: This is a SPARQL query which retrieves result tuples from the reasoner. The query results will then be used in the LP rules. This is different to the definition from the theoretical part, where the query Q consists of exactly one predicate, and it indicates the flexibility of the approach.
– `<query-result>`: The bindings for the SPARQL query results. Rules can access the results of `dlaccess` for further processing via unification.
– `<server>`: We must identify the exact location of the reasoning server in order to access it. We use a URL for locating the server.

In order to better explain how this predicate can be utilized to connect rule-based logic programs with DL reasoners we show a typical rule using the `dlaccess/6` predicate – for readability, we leave out the ontology ID and the server location:

```
FORALL FROM, TO, DIRECT_FLIGHTS
  route(FROM, TO) AND FROM:#city AND TO:#city
<-
  dlaccess(
    [addProperty( "http://example.org/Flights#localFlight",
     #directFlights)],
    [DIRECT_FLIGHTS],
    "PREFIX <http://example.org/Flights#>
      SELECT ?X ?Y
      WHERE { ?X :localFlight ?Y }",
    F(FROM,TO)) AND
  allDirectFlights(DIRECT_FLIGHTS).
```

The body of the rule merely consists of the `dlaccess/6` predicate and another predicate called `allDirectFlights/1`. The first two arguments of `dlaccess` specify that the `directFlights` instances (computed via `allDirectFlights`) are used to populate the `localFlight` object property in the DL ontology. After asserting this to the DL knowledge base, we execute the SPARQL-query, which retrieves (X, Y)-tuples. These tuples are then bound to `(FROM, TO)`[12].

[12] Since F-logic has no functional terms without a functor we assume a standard function symbol F.

In the head of the rule we then assert that FROM and TO are members of the class city and that they are connected via route/2. We can thus introduce new facts to the F-logic KB which have been mainly derived from executing a SPARQL query on the (extended) DL knowledge base.

The second predicate of the rule body is defined via a normal F-logic rule using a special built-in predicate, which collects a number of values into a single list term. In this case the allDirectFlights literal binds the DIRECT_FLIGHTS variable to a list of the following form: $[F(From_1, To_1), F(From_2, To_2), ...F(From_n, To_n)]$ which corresponds to the instances of the directFlights method.

In order to collect multiple values (the results of a sub-query) into a single list we exploit a special built-in predicate xlist/4 – a so-called aggregator.

The implementation in OntoBroker F-logic for each use of the F-Logic# operator always uses two or more rules. The first rule calls out to dlaccess feeding it some new facts from the F-logic KB and (typically) using its results in the rule head. The other rules aggregate the needed input for dlaccess, i.e. compute the instantiations of certain properties, classes or predicates and stores them in an appropriate list format. If there are multiple mappings in the dlaccess rule, there are also multiple *aggregator rules*.

6 Outlook

Even though a loosely coupled integration is easier to maintain then a tightly coupled integration, one still needs to keep track of changes in the ontology from the rule perspective. Currently, we are implementing a dependency management plug-in for the ontology engineering platform OntoStudio [21] within the EU funded FP7 project OntoRule. The component notifies the knowledge engineer about changes in the OWL ontology that are relevant for external atoms within the F-logic model. Certain changes or refactorings on the OWL side may even automatically propagate to the rules side to keep the models synchronized.

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. C. Baral and V. S. Subrahmanian. Dualities between alternative semantics for logic programming and nonmonotonic reasoning. *J. Autom. Reasoning*, 10(3):399–420, 1993.
3. Harold Boley and Michael Kifer. *RIF Basic Logic Dialect*. W3C, 2009. Candidate Recommendation October 2009.
4. Harold Boley and Michael Kifer. *RIF Framework for Logic Dialects*. W3C, 2009. Working draft July 2009.
5. Jos de Bruijn and Stijn Heymans. A semantic framework for language layering in WSML. In *Proceedings of the 1st International Conference on Web Reasoning and Rule Systems (RR2007)*, pages 103–117, Innsbruck, Austria, June 7–8 2007. Springer.
6. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425, September 2001.

7. J. de Bruijn. *RIF RDF and OWL Compatibility*. W3C, 2009. http://www.w3.org/2005/rules/wiki/SWC.
8. Mike Dean and Guus Schreiber. *OWL Web Ontology Language Reference*. W3C, 2004. W3C Recommendation 10 February 2004.
9. T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. *Artificial Intelligence*, 172(12-13):1495–1539, 2008.
10. Allen Van Gelder, Kenneth Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
11. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
12. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *Proc. of the World Wide Web Conference (WWW)*, pages 48–57. ACM, 2003.
13. Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida, 2003.
14. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing shiq descrption logic to disjunctive datalog programs. In *Proc. of the 9th Int. Conf. on Knowledge Representation and Reasoning (KR2004)*, pages 152–162, Whistler, Canada, Juni 2004. AAAI Press.
15. Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.
16. M. Krötzsch, S. Rudolph, and P. Hitzler. Description logic rules. In *Proc. 18th European Conf. on Artificial Intelligence(ECAI-08)*, pages 80–84. IOS Press, 2008.
17. Senlin Liang, Paul Fodor, Hui Wan, and Michael Kifer. OpenRuleBench: an analysis of the performance of rule engines. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 601–610, New York, NY, USA, 2009. ACM.
18. T. Lukasiewicz. A novel combination of answer set programming with description logics for the semantic web. In *Proc. of ESWC 2007*, pages 348–398, 2007.
19. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, July 2005.
20. ontoprise GmbH, An der RaumFabrik 29, 76137 Karlsruhe, Germany. *OntoBroker Enterprise - Version 5.3*, 2009. Available at http://www.ontoprise.de/help/index.jsp.
21. ontoprise GmbH, An der RaumFabrik 29, 76137 Karlsruhe, Germany. *OntoStudio Manual - Version 2.3.0*, 2009. Available at http://www.ontoprise.de/help/index.jsp.
22. W3C OWL Working Group. *SPARQL Query Language for RDF*. W3C Recommendation, 15 January 2008. Available at http://www.w3.org/TR/rdf-sparql-query/.
23. W3C OWL Working Group. *SPARQL Update*. W3C Member Submission, 15 July 2008. Available at http://www.w3.org/Submission/SPARQL-Update/.
24. W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009. Available at http://www.w3.org/TR/owl2-overview/.
25. R. Rosati. DL+log: Tight integration of description logics and disjunctive datalog. In *Proc. of the Int. Conf. on Principles of Knowledge Representation and Reasoning (KR)*, pages 68–78, 2006.
26. Riccardo Rosati. On the decidability and complexity of integrating ontologies and rules. *Journal of Web Semantics*, 3(1):61–73, 2005.
27. Guizhen Yang and Michael Kifer. Flora-2: User's manual, 2001. http://flora.sourceforge.net/.