A RIF-style semantics for RuleML-integrated positional-slotted, object-applicative rules

Boley, Harold

Publisher’s version / Version de l'éditeur:
https://doi.org/10.1007/978-3-642-22546-8_16
Rule-based Reasoning, Programming, and Applications: 5th International Symposium, RuleML 2011 – Europe, Barcelona, Spain, July 19-21, 2011. Proceedings, Lecture Notes in Computer Science; no. 6826, pp. 194-211, 2011-

A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules

Harold Boley

Institute for Information Technology
National Research Council of Canada
Fredericton, NB, E3B 9W4, Canada

Abstract. In F-logic and RIF, objects (frames) are defined entirely separately from function and predicate applications. In POSL and RuleML, these fundamental notions are integrated by permitting applications with optional object identifiers and, orthogonally, arguments that are positional or slotted. The resulting positional-slotted, object-applicative (psoa) terms are given a novel formalization, reducing the number of RIF terms by generalizing its positional and slotted (named-argument) terms as well as its frame terms and class memberships. Like multi-slot frames accommodate for (Web-)distributed slotted descriptions of the same object identifier (IRI), multi-tuple psoa terms (e.g., shelves) do for positional descriptions. The syntax and semantics of these integrated terms and rules over them are defined as PSOA RuleML in the style of RIF-BLD. The semantics provides a novel first-order model-theoretic foundation, blending frame slotribution, as in F-logic and RIF (as well as shelf tupribution) with integrated psoa terms, as in POSL and RuleML.

1 Introduction

Logic-based (e.g., FOL, Horn, LP) as well as object-oriented (and frame-based) paradigms (e.g., CLOS, RDF, N3) have been employed for knowledge representation and problem solving in AI, the (Semantic) Web, and IT at large. In search for a unified paradigm for AI/(Sem)Web languages, there have been various approaches to combining these paradigms in Description Logics (DLs), Object-Oriented Databases (OODBs) / Deductive Object-Oriented Databases (DOODs), and object-oriented logic languages such as LIFE [AK93] and F-logic [KLW95]. The W3C Rule Interchange Format (RIF) [BK10a] adopted a semantics based on F-logic with a serialization syntax based on RuleML [BPS10].

While F-logic and RIF have accommodated the standard first-order model-theoretic semantics [CL73] for the incorporation of objects (frames), these are added separately from function and predicate applications to arguments. The resulting complexity of the object-extended semantics can be reduced by integrating objects with applications. In this paper we present an integration based on the positional-slotted, object-applicative rules of POSL and RuleML [Bol10]. F-logic's model-theoretic semantics in the style of RIF is also the starting point of our integrated semantics. Our integration permits applications with optional object identifiers and, orthogonally, arguments that are positional or slotted.

Structured by these independent dimensions of defining features, language constructs can then be freely combined.

The integration is based on **p**ositional-**s**lotted, **o**bject-**a**pplicative (***psoa***) terms and rules over them. A psoa term applies a function or predicate symbol, possibly instantiated by an object, to zero or more positional or slotted (named) arguments. In the interpretation of a psoa term as an atomic formula, the predicate symbol is both the class (type) of the object and the relation between the arguments, which describe the object. Each argument of a psoa term can be a psoa term that applies a function symbol.

The intuition behind the fundamental distinctions in the taxonomy of psoa terms is as follows. Psoa terms that apply a predicate symbol (as a relation) to *positional arguments* can be employed to make factual assertions. An example, in simplified RIF (presentation) syntax,[1] is the term `married(Joe Sue)` for the binary predicate `married` applied to `Joe` and `Sue`, where the positional (left-to-right) order can be used to identify the husband, as the first argument, and wife, as the second argument.

Psoa terms that apply a predicate symbol (as a class) to *slotted arguments* correspond to typed attribute-value descriptions. An example is the psoa term `family(husb->Joe wife->Sue)` or `family(wife->Sue husb->Joe)` for the `family`-typed attribute-value pairs (slots) {<`husb,Joe`>, <`wife,Sue`>}. Such a description can be easily extended with further slots, e.g. by adding one or more children, as in `family(husb->Joe wife->Sue child->Pete)`.[2] Usually, slotted terms describe an object symbol, i.e. an object identifier (OID), maintaining object identity even when slots of their descriptions are added or deleted. This leads to (typed) frames in the sense of F-logic. For example, using RIF's membership syntax `#`, the OID `inst1`, as a member of the class `family`, can be described by `inst1#family(husb->Joe wife->Sue)`, by `inst1#family(husb->Joe wife->Sue child->Pete)`, etc. Psoa terms can also specialize to class membership terms, e.g. `inst1#family()`, abridged `inst1#family`, represents `inst1` $\in$ `family`.

While positional and slotted, object-oriented and applicative terms have mostly been treated separately, psoa terms integrate them, allowing for all intermediate forms. Like OID-describing slotted terms constitute a (multi-slot) 'frame', positional terms that describe an object constitute a (single-tuple) 'shelf', similar to a (one-dimensional) array describing its name. Thus, in the `family` example, the `husb` and `wife` slots can be positionalized as in the earlier `married` example: `inst1#family(Joe Sue)` describes `inst1` with the argument tuple `[Joe Sue]`. *Combined positional-slotted* psoa terms are allowed, as in XML elements (tuple$\rightsquigarrow$subelements, slots$\rightsquigarrow$attributes), optionally describing an object, as always required by RDF descriptions (object$\rightsquigarrow$subject, slots$\rightsquigarrow$properties).[3]

[1] In this introduction, we omit RIF's namespace prefixes for simplicity.

[2] As in RDF and RIF, attributes are multi-valued by default, allowing, e.g., `family(... child->Pete child->Jane)`. Duplications of entire slots are also syntactically permitted, e.g., `family(... child->Pete child->Pete)`, but will be semantically treated as duplicate-free, e.g., `family(... child->Pete)`.

[3] See earlier XML/RDF unification: http://www.dfki.uni-kl.de/~boley/xmlrdf.html.

For example, `inst1#family(Joe Sue child->Pete)` describes `inst1` with two positional and one slotted argument.

On the other hand, the positional `married` example could be made slotted, leading to `married(husb->Joe wife->Sue)`, and even be used to describe a (marriage) object: positionally, as in `inst2#married(Joe Sue)`, or slotted, as in `inst2#married(husb->Joe wife->Sue)`.

Summarizing, an object's description or an application's arguments can consist of slots as well as a tuple of values. This includes object-describing atomic formulas playing the role of frames, shelves, or the combination of both.

A frame without an explicit class is semantically treated as typing its object with the root class $\top$ (syntactically, `Top`). For example, the (untyped) frame `inst3[color->red shape->diamond]` in square-bracketed F-logic/RIF syntax is equivalent to our parenthesized `inst3#Top(color->red shape->diamond)`.[4]

An atomic formula without an OID is treated as having an implicit OID. An OID-less application is *objectified* by a syntactic transformation as follows: *The OID of a ground fact is a new constant generated by the 'new local constant' (a stand-alone `_`); the OID of a non-ground fact or of an atomic formula in a rule conclusion, `f(...)`, is a new, existentially scoped variable `?i`, leading to `Exists ?i (?i#f(...))`; the OID of any other atomic formula is a new variable generated by the 'anonymous variable' (a stand-alone `?`).* Objectification allows compatible semantics for an atom constructed as a RIF-like slotted (named-argument) term and a corresponding frame, solving an issue with RIF's named-argument terms.[5]

For example, the slotted-fact assertion `family(husb->Joe wife->Sue)` is syntactically objectified to the assertion `_#family(husb->Joe wife->Sue)`, and – if `_1` is the first new constant from `_1`, `_2`, ... – to `_1#family(husb->Joe wife->Sue)`. This typed frame, then, is semantically *slotributed* to `_1#family(husb->Joe)` and `_1#family(wife->Sue)`. The query `family(husb->Joe)` is syntactically objectified to the query `?#family(husb->Joe)`, i.e. – if `?1` is the first new variable in `?1`, `?2`, ... – to `?1#family(husb->Joe)`. Posed against the fact, it succeeds with the first slot, unifying `?1` with `_1`. Slotribution ('slot distribution') avoids POSL's 'rest-slot' variables [Bol10]: a frame's OID 'distributes' over the slots of a description.

Rules can be defined on top of psoa terms in a natural manner. A rule derives (a conjunction of possibly existentially scoped) conclusion psoa atoms from (a formula of) premise psoa atoms. Let us consider an introductory example with a rule deriving `family` frames; this will be modified in Example 4 of Section 4.

Example 1 (Rule-defined anonymous family frame). A `Group` is used to collect a rule and two facts. The `Forall` quantifier declares the original universal argument variables as well as the generated universal OID variables `?2`, `?3`, `?4`. The `:-` infix separates the conclusion from the premises of a rule, which derives an anonymous/existential `family` frame from a `married` relation `And` from a `kid` relation of the `husb` `Or` `wife` (the left-hand side is objectified on the right).

[4] `Top` will allow us to always use parenthesized typed frames, and to reserve square brackets for enclosing positional tuples.

[5] See Dave Reynolds' point: http://lists.w3.org/Archives/Public/public-rif-wg/2008Jul/0000.html.

```
Group (
 Forall ?Hu ?Wi ?Ch (

  family(husb->?Hu wife->?Wi child->?Ch) :-
    And(married(?Hu ?Wi)
        Or(kid(?Hu ?Ch) kid(?Wi ?Ch))) )
 married(Joe Sue)
 kid(Sue Pete)
      )
```

```
Group (
 Forall ?Hu ?Wi ?Ch ?2 ?3 ?4 (
  Exists ?1 (
   ?1#family(husb->?Hu wife->?Wi child->?Ch)) :-
    And(?2#married(?Hu ?Wi)
        Or(?3#kid(?Hu ?Ch) ?4#kid(?Wi ?Ch))) )
 _1#married(Joe Sue)
 _2#kid(Sue Pete)
      )
```

Semantically, this example is modeled by predicate extensions corresponding to the following set of ground facts (the subdomain of individuals $\boldsymbol{D}_{\mathrm{ind}}$ will be defined in Section 3.1):

$\{o\#family(husb\text{-}{>}Joe\ wife\text{-}{>}Sue\ child\text{-}{>}Pete)\ \} \cup$
$\{\_1\#married(Joe\ Sue),\ \_2\#kid(Sue\ Pete)\}$, where $o \in \boldsymbol{D}_{\mathrm{ind}}$.

A language incorporating this integration, PSOA RuleML, is defined here. The rest of the paper is organized as follows. Section 2 gives the human-readable presentation syntax of PSOA RuleML. Section 3 gives its model-theoretic semantics. Section 4 concludes the paper and discusses future work.

2 The Presentation Syntax

The presentation syntax of PSOA RuleML is built on the one of RIF-BLD and described in "mathematical English". An EBNF syntax is then given, although it cannot fully capture the presentation syntax, as the latter is not context-free.

2.1 Alphabet of PSOA RuleML

Definition 1 (Alphabet). *The **alphabet** of the presentation language of PSOA RuleML consists of the following disjoint sets:*

- *A countably infinite set of **constant symbols** `Const` (including the root class* `Top` $\in$ *`Const` and the positive-integer-enumerated local constants* `_1`, `_2`, $\ldots \in$ *`Const` as well as individual, function, and predicate symbols).*
- *A countably infinite set of **variable symbols** `Var` (including the positive-integer-enumerated variables* `?1`, `?2`, $\ldots \in$ *`Var`).*
- *The connective symbols `And`, `Or`, and `:-`.*
- *The quantifiers `Exists` and `Forall`.*
- *The symbols `=`, `#`, `##`, `->`, `External`, `Import`, `Prefix`, and `Base`.*
- *The symbols `Group` and `Document`.*
- *The auxiliary symbols `(`, `)`, `<`, `>`, `^^`, and `_`.*

Constants have the form `"literal"^^symspace`, where `literal` is a sequence of Unicode characters and `symspace` is an identifier for a symbol space. An example is `"_123"^^rif:local`. *Constants can use shortcuts as defined in [PBK10], including the underscore notation `_literal` (e.g.,* `_123`*) for the above form with `symspace` specialized to* `rif:local`. `Top` *is a new shortcut for the root class constant* `"Top"^^psoa:global` *in PSOA RuleML's global symbol space.*

Anonymous variables are written as a stand-alone question mark symbol (`?`); named variables, as Unicode strings preceded with the question mark symbol.

The symbols `=` *and* `##` *are used in formulas that define equality and subclass relationships. The symbols* `#` *and* `->` *are used in positional-slotted, object-applicative formulas for class memberships and slots, respectively. The symbol* `External` *indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and the symbols* `Prefix` *and* `Base` *enable abridged representations of IRIs (Internationalized Resource Identifiers).*

□

The language of PSOA RuleML is the set of formulas built using the above alphabet according to the construction methods given below.

2.2 Terms

The main parts of rules are called *terms*. PSOA RuleML defines several kinds of terms: *constants* and *variables*, *psoa* terms, *equality*, *subclass*, and *external* terms. Thus "*term*" will be used to refer to any one of these constructs.

Below, the phrase *base term* means a simple term, an anonymous psoa term (i.e., an anonymous frame term, single-tuple psoa term, or multi-tuple psoa term), or a term of the form `External(t)`, where `t` is an anonymous psoa term. An anonymous term can be *deobjectified* (by omitting the main `?#`) if its re-objectification (cf. Section 1) results in the original term (i.e., re-introduces `?#`).

Definition 2 (Term). *PSOA RuleML defines several different types of logic terms. Here we describe the syntax of the most important ones.*

1. *Constants and variables. If* $t \in$ *Const or* $t \in$ *Var then* t *is a* ***simple term****.*
2. *Equality terms.* $t = s$ *is an* ***equality term*** *if* t *and* s *are base terms.*
3. *Subclass terms.* $t\#\#s$ *is a* ***subclass term*** *if* t *and* s *are base terms.*
4. *Positional-slotted, object-applicative terms.* $o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)$ *is a* ***positional-slotted, object-applicative (psoa) term*** *if* $f \in$ *Const and* o, $t_{1,1}, \ldots, t_{1,n_1}, \ldots, t_{m,1}, \ldots, t_{m,n_m}, p_1, \ldots, p_k, v_1, \ldots, v_k$, $m \geq 0$, $k \geq 0$, *are base terms.*
   Psoa terms can be specialized in the following way.[6]
   - *For* $m = 0$ *they become* ***(typed or untyped) frame terms*** $o\#f(p_1\text{->}v_1 \ldots p_k\text{->}v_k)$*. We consider two overlapping subcases.*
     - *For* $k = 0$ *they become* ***class membership terms*** $o\#f()$*, abridged to* $o\#f$*, corresponding to those in F-logic and RIF.*
     - *For* $k \geq 0$ *they can be further specialized in two ways, which can be orthogonally combined.*
       - *For* o *being the anonymous variable* $?$*, they become* ***anonymous frame terms (slotted terms)*** $?\#f(p_1\text{->}v_1 \ldots p_k\text{->}v_k)$*, deobjectified* $f(p_1\text{->}v_1 \ldots p_k\text{->}v_k)$*, corresponding to* terms with named arguments *in RIF.*

[6] Distinctions similar to those for `m = 1`, and further ones, could be made for `m > 1`, i.e. ***multi-tuple psoa terms***, but for space reasons we leave most of them implicit in the general psoa term definition here. We do note that for `m > 1` and `k = 0` multi-tuple psoa terms specialize to ***multi-tuple shelf terms***. Also, for `o` being the anonymous variable `?`, these terms become ***anonymous multi-tuple psoa terms***.

- *
  - *
    - ∗ *For f being the root class `Top`, they become* ***untyped frame terms*** *`o#Top(`p_1`->`v_1 `...` p_k`->`v_k`)` corresponding to* frames *in the abridged form `o[`p_1`->`v_1 `...` p_k`->`v_k`]` of F-logic and RIF, where square brackets are used instead of round parentheses.*
  - *For `m = 1` they become* ***single-tuple psoa terms*** *`o#f([`$t_{1,1}$ ... t_{1,n_1}`]` p_1`->`v_1 `...` p_k`->`v_k`)`, abridged to `o#f(`$t_{1,1}$ ... t_{1,n_1} p_1`->`v_1 `...` p_k`->`v_k`)`. These can be further specialized in two ways, which can be orthogonally combined:*[7]
    - *For `o` being the anonymous variable `?`, they become* ***anonymous single-tuple psoa terms*** *`?#f(`$t_{1,1}$ ... t_{1,n_1} p_1`->`v_1 `...` p_k`->`v_k`)`, deobjectified `f(`$t_{1,1}$ ... t_{1,n_1} p_1`->`v_1 `...` p_k`->`v_k`)`.*
      These can be further specialized:
      - ∗ *For `k = 0`, they become* ***positional terms*** *`?#f(`$t_{1,1}$ ... t_{1,n_1}`)`, deobjectified `f(`$t_{1,1}$ ... t_{1,n_1}`)`, corresponding to the usual terms and atomic formulas of classical first-order logic.*
    - *For `f` being the root class `Top`, they become* ***untyped single-tuple psoa terms*** *`o#Top(`$t_{1,1}$ ... t_{1,n_1} p_1`->`v_1 `...` p_k`->`v_k`)`.*
      These can be further specialized:
      - ∗ *For `k = 0`, they become* ***untyped single-tuple shelf terms*** *`o#Top(`$t_{1,1}$ ... t_{1,n_1}`)` describing the object `o` with the positional arguments $t_{1,1}$, ..., t_{1,n_1}.*

5. *Externally defined terms. If `t` is an anonymous psoa term then `External(t)` is an* ***externally defined term****.*
   External terms represent built-in function or predicate invocations as well as "procedurally attached" function or predicate invocations. Procedural attachments are often provided by rule-based systems, and external terms constitute a way of supporting them in PSOA RuleML. □

The notion of psoa term is generalized here from allowing a single tuple, as in [Bol10], to allowing a bag (multi-set) of tuples. Together with 'tupribution' (cf. definition 5, item 3), this accommodates for distributed positional descriptions of the same OID. For multiple tuples (m>1) each tuple is enclosed by square brackets, which can be omitted for a single tuple (m=1). The special case $\mathtt{n_1} = \ldots = \mathtt{n_m}$ is useful to describe the distributed object with 'homogeneous' equal-length tuples of a relation: the OID names the extension of the relation's tuples.

Observe that the argument names of psoa terms, $\mathtt{p_1}$, ..., $\mathtt{p_n}$, are base terms, hence can be constants or variables. Since psoa terms include anonymous frames (slotted terms), this generalizes RIF, where the corresponding named-argument terms can only use argument names from a separate set `ArgNames`, to reduce the complexity of unification [BK10a]. PSOA RuleML could emulate such a special treatment of slotted terms based on reserving an `ArgNames`-style subset of `Const` for argument names. On the other hand, as shown in Section 1, since PSOA RuleML's slotted terms via objectification are conceived as frames, they can be queried by slotribution rather than unification.

[7] The combination of `o` = `?` and `f` = `Top` leads to ***anonymous, untyped psoa terms***, describing anonymous variables without a class/type, which could be further specialized for `m = 0` and for `k = 0`.

2.3 Formulas

An ***atomic formula*** is any psoa term of the form `f(...)` or `o#f(...)`, with `f` being a predicate symbol and `o` a simple term (constant or variable), or any equality or subclass term, or any externally defined term of the form `External`(φ), where φ is an atomic formula. Simple terms are *not* formulas. More general formulas are built from atomic formulas via logical connectives.

Definition 3 (Formula).
*A **formula** can have one of the following forms:*

1. *Atomic: An atomic formula is also a formula.*
2. *Condition formula: A **condition formula** is either an atomic formula or a formula that has one of the following forms:*
   - *Conjunction: If φ_1, ..., φ_n, $n \geq 0$, are condition formulas then so is `And`(φ_1 ... φ_n), called a **conjunctive** formula. As a special case, `And()` is allowed and is treated as a tautology, i.e., a formula that is always true.*
   - *Disjunction: If φ_1, ..., φ_n, $n \geq 0$, are condition formulas then so is `Or`(φ_1 ... φ_n), called a **disjunctive** formula. As a special case, `Or()` is considered as a contradiction, i.e., a formula that is always false.*
   - *Existentials: If φ is a condition formula and ?V_1, ..., ?V_n, n>0, are distinct variables then `Exists` ?V_1 ... ?V_n(φ) is an **existential** formula.*
3. *Rule implication: φ `:-` ψ is a formula, called **rule implication**, if:*
   - *φ is a head formula or a conjunction of head formulas, where a head formula is an atomic formula or an existentially scoped atomic formula,*
   - *ψ is a condition formula, and*
   - *none of the atomic formulas in φ is an externally defined term (i.e., a term of the form `External(...)`).*
     Note that external terms can occur in the arguments of atomic formulas in the rule conclusion, but they cannot occur as atomic formulas.
4. *Universal rule: If φ is a rule implication and ?V_1, ..., ?V_n, n>0, are distinct variables then `Forall` ?V_1 ... ?V_n(φ) is a universal rule formula. It is required that all the free variables in φ occur among the variables ?V_1 ... ?V_n in the quantification part. Generally, an occurrence of a variable ?v is free in φ if it is not inside a subformula of φ of the form `Exists` ?v (ψ) and ψ is a formula. Universal rules are also referred to as* PSOA RuleML rules.
5. *Universal fact: If φ is an atomic formula and ?V_1, ..., ?V_n, n>0, are distinct variables then `Forall` ?V_1 ... ?V_n(φ) is a universal fact formula, provided that all the free variables in φ occur among the variables ?V_1 ... ?V_n.*
   Universal facts are treated as rules without premises.
6. *Group: If φ_1, ..., φ_n are PSOA RuleML rules, universal facts, variable-free rule implications, variable-free atomic formulas, or group formulas then `Group`(φ_1 ... φ_n) is a group formula.*
   Group formulas are used to represent sets of rules and facts. Note that some of the φ_i's can be group formulas themselves, i.e. groups can be nested.
7. *Document: An expression of the form `Document`(directive$_1$ ... directive$_n$ Γ) is a PSOA RuleML document formula, if*

– *Γ is an optional group formula; it is called the group formula associated with the document.*
– *directive$_1$, ..., directive$_n$ is an optional sequence of directives. A directive can be a base directive, a prefix directive or an import directive. For details see [BK10a].* □

2.4 Well-formed Formulas

Not all formulas or documents are well-formed in PSOA RuleML. The well-formedness restriction is similar to standard first-order logic: it is required that no constant appear in more than one context. Informally, unique context means that no constant symbol can occur within the same document as an individual or a (plain or external) function or predicate symbol in different places. The detailed definitions are as in RIF-BLD, found in [BK10b], Section 2.5.

2.5 EBNF Grammar for the Presentation Syntax of PSOA RuleML

Until now, we have been using mathematical English to specify the syntax of PSOA RuleML. Since tool developers might prefer a more succinct overview of the syntax using familiar grammar notation, our PSOA RuleML specification also supplies an EBNF definition. For instance, a condition formula in mathematical English becomes a `FORMULA` nonterminal in EBNF.

The EBNF grammar for the PSOA RuleML presentation syntax is as follows:

Rule Language:

```
Document ::= 'Document' '(' Base? Prefix* Import* Group? ')'
Base     ::= 'Base' '(' ANGLEBRACKIRI ')'
Prefix   ::= 'Prefix' '(' Name ANGLEBRACKIRI ')'
Import   ::= 'Import' '(' ANGLEBRACKIRI PROFILE? ')'
Group    ::= 'Group' '(' (RULE | Group)* ')'
RULE     ::= ('Forall' Var+ '(' CLAUSE ')') | CLAUSE
CLAUSE   ::= Implies | ATOMIC
Implies  ::= (HEAD | 'And' '(' HEAD* ')') ':-' FORMULA
HEAD     ::= ATOMIC | 'Exists' Var+ '(' ATOMIC ')'
PROFILE  ::= ANGLEBRACKIRI
```

Condition Language:

```
FORMULA  ::= 'And' '(' FORMULA* ')' |
             'Or' '(' FORMULA* ')' |
             'Exists' Var+ '(' FORMULA ')' |
             ATOMIC |
             'External' '(' Atom ')'
ATOMIC   ::= Atom | Equal | Subclass
Atom     ::= PSOA
Equal    ::= TERM '=' TERM
Subclass ::= TERM '##' TERM
PSOA     ::= TERM '#' TERM '(' TUPLE* (TERM '->' TERM)* ')'
```

```
TUPLE    ::= '[' TERM* ']'
TERM     ::= Const | Var | Expr | 'External' '(' Expr ')'
Expr     ::= PSOA
Const    ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Var      ::= '?' UNICODESTRING?
SYMSPACE ::= ANGLEBRACKIRI | CURIE
```

The following subsections explain and illustrate the two parts of the syntax; first the foundational language of positive conditions, then the language of rules.

EBNF for the Condition Language The Condition Language represents formulas that can be used as queries or in the premises of PSOA RuleML rules.

The production for the non-terminal FORMULA represents *PSOA RuleML condition formulas* (cf. definition 3, item 2). The connectives And and Or define conjunctions and disjunctions of conditions, respectively. Exists introduces existentially quantified variables. Here Var+ stands for the list of variables that are free in FORMULA. A PSOA RuleML FORMULA can also be an ATOMIC term, i.e., an Atom, Equal, or Subclass. A TERM can be a constant, variable, Expr, or External Expr.

Example 2 (PSOA RuleML conditions). This example shows conditions that are composed of psoa terms ("Opticks" is a shortcut for "Opticks"^^xs:string).

```
Prefix(bks  <http://eg.com/books#>)
Prefix(auth <http://eg.com/authors#>)
Prefix(cts  <http://eg.com/cities#>)
Prefix(cpt  <http://eg.com/concepts#>)
```
Formula that uses an anonymous psoa (positional term):
```
  ?#cpt:book(auth:Newton "Opticks")
```
Deobjectified version:
```
  cpt:book(auth:Newton "Opticks")
```
Formula that uses an anonymous psoa (slotted term):
```
  ?#cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```
Deobjectified version:
```
  cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```
Formula that uses a named psoa (typed frame):
```
  bks:opt1#cpt:book(cpt:author->auth:Newton cpt:title->"Opticks")
```
Formula that uses a named psoa (untyped frame):
```
  bks:opt1#Top(cpt:author->auth:Newton cpt:title->"Opticks")
```
Deobjectified version of a formula that uses an anonymous psoa (multi-tuple term):
```
  cpt:book([auth:Newton "Opticks"] [cts:London "1704"^^xs:integer])
```
Deobjectified version of a formula that uses an anonymous psoa (positional-slotted term):
```
  cpt:book(auth:Newton "Opticks" cpt:place->cts:London cpt:year->"1704"^^xs:integer)
```

EBNF for the Rule Language The EBNF for PSOA RuleML rules and documents is given in Section 2.5. A PSOA RuleML Document consists of an optional Base directive, followed by any number of Prefixes and then any number of Imports. These may be followed by an optional Group. Base and Prefix are employed by the shortcut mechanisms for IRIs. An Import directive indicates the location of a document to be imported and an optional profile. A PSOA RuleML Group is a collection of any number of RULE elements along with any number of nested Groups.

Rules are generated using CLAUSE elements via two RULE alternatives:

– In the first, a `CLAUSE` is in the scope of the `Forall` quantifier. In that case, all variables mentioned in `CLAUSE` are required to also appear among the variables in the `Var+` sequence.
– In the second alternative, `CLAUSE` appears on its own. In that case, `CLAUSE` cannot have variables.

`Var`, `ATOMIC`, and `FORMULA` were defined as part of the syntax for positive conditions in Section 2.5. In the `CLAUSE` production, `ATOMIC` is what is usually called a *fact*. An `Implies` *rule* can have a `HEAD` element or a conjunction of `HEAD` elements as its conclusion; a `HEAD` is an `ATOMIC` element or an `Exists` of an `ATOMIC` element. The `Implies` has a `FORMULA` as its premise. Note that, by Definition 3, externally defined atoms (i.e., formulas of the form `External(Atom)`) are not allowed in the conclusion part of a rule (`ATOMIC` does not expand to `External`).

Example 3 (PSOA RuleML business rule). This example adapts a business rule from a POSL logistics use case [Bol10]. The ternary `reciship` conclusion represents `reci`procal `ship`pings, at a total cost (as the single positional argument), between a `source` and a `dest`ination (as two slotted arguments). The first two premises apply a 4-ary `shipment` relation that uses an anonymous cargo and named cost variables as two positional arguments, as well as `reciship`'s slotted arguments (in both 'directions'). The third premise is an `External`-wrapped `numeric-add` built-in [PBK10] applied on the right-hand side of an equality to sum up the `shipment` costs for the total cost. With the two facts, `?cost` = `?57.0`.

```
Prefix(cpt  <http://eg.com/concepts#>)
Prefix(mus  <http://eg.com/museums#>)
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
Prefix(xs   <http://www.w3.org/2001/XMLSchema#>)
Group (
  Forall ?cost ?cost1 ?cost2 ?A ?B (
    cpt:reciship(?cost cpt:source->?A cpt:dest->?B) :-
      And(cpt:shipment(? ?cost1 cpt:source->?A cpt:dest->?B)
          cpt:shipment(? ?cost2 cpt:source->?B cpt:dest->?A)
          ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                )
  shipment("PC"^^xs:string "47.5"^^xs:float cpt:source->mus:BostonMoS cpt:dest->mus:LondonSciM)
  shipment("PDA"^^xs:string "9.5"^^xs:float cpt:source->mus:LondonSciM cpt:dest->mus:BostonMoS)
      )
```

The rule can be objectified as follows (`External`s are not being transformed):

```
  Forall ?cost ?cost1 ?cost2 ?A ?B ?2 ?3 (
    Exists ?1 (?1#cpt:reciship(?cost cpt:source->?A cpt:dest->?B)) :-
      And(?2#cpt:shipment(? ?cost1 cpt:source->?A cpt:dest->?B)
          ?3#cpt:shipment(? ?cost2 cpt:source->?B cpt:dest->?A)
          ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                      )
```

Further, it can be tupributed and slotributed thus (actually done by the semantics):

```
  Forall ?cost ?cost1 ?cost2 ?A ?B ?2 ?3 (
    Exists ?1 (And(?1#cpt:reciship(?cost)
                   ?1#cpt:reciship(cpt:source->?A)
                   ?1#cpt:reciship(cpt:dest->?B))) :-
      And(?2#cpt:shipment(? ?cost1)
          ?2#cpt:shipment(cpt:source->?A)
          ?2#cpt:shipment(cpt:dest->?B)
          ?3#cpt:shipment(? ?cost2)
          ?3#cpt:shipment(cpt:source->?B)
          ?3#cpt:shipment(cpt:dest->?A)
          ?cost = External(func:numeric-add(?cost1 ?cost2)) )
                                      )
```

3 Semantics

The formalization of the PSOA RuleML semantics in this section is in the style of RIF-BLD [BK10a], which in some respects is more general than what would be actually required. The reason for this generality is the need to ensure that the semantics stay comparable, and that a future RIF logic dialect could be specified to cater for PSOA (e.g., via an updated RIF-FLD [BK10c]).

For the interpretation of (multiple) PSOA RuleML documents, we refer to the RIF-BLD article [BK10a]. We mention that a *local* constant, marked by an underscore prefix (e.g., `_uvw`), is encapsulated within documents, i.e. it can be interpreted differently in different documents. Based on that, in a given document, the *new local constant* generator, written as a stand-alone `_`, denotes the first new local constant $\_i$, $i \geq 1$, from the sequence `_1`, `_2`, ... that does not already occur in that document (cf. *anonymous ID symbols* in [YK03]). For each document we will assume OID-less psoa terms to be objectified by the transformation of Section 1, whose head existentials make PSOA RuleML non-Horn.

To save space, in describing the semantics we omit lists and datatypes, and simplify the semantics of external functions and predicates, all found in the RIF-BLD specification [BK10b].

3.1 Semantic Structures

The semantics of PSOA RuleML is an extension of the standard semantics for Horn clauses. This semantics is specified using *general models* while the semantics for Horn clauses is usually given via *Herbrand models* [Llo87]. Without head existentials, the two semantics become equivalent. We will use $\boldsymbol{TV}$ to denote $\{\mathbf{t},\mathbf{f}\}$–the set of truth values used in the semantics. $\boldsymbol{TV}$ is used in RIF because it is intended to address (through RIF-FLD [BK10c]) a range of logic languages, including those that are based on multi-valued logics. Since PSOA RuleML is based on the classical two-valued logic, its set $\boldsymbol{TV}$ is particularly simple.

Truth valuation of PSOA RuleML formulas will be defined as a mapping $TVal_{\mathcal{I}}$ in two steps: 1. A mapping $\boldsymbol{I}$ generically bundles the various mappings from the semantic structure, $\mathcal{I}$; $\boldsymbol{I}$ maps a formula to an element of the domain $\boldsymbol{D}$. 2. A mapping $\boldsymbol{I}_{\text{truth}}$ takes such a domain element to $\boldsymbol{TV}$. This indirectness allows HiLog-like generality, as detailed at the beginning of Section 3.2.

The key concept in a model-theoretic semantics for a logic language is the notion of *semantic structures* [End01], which is defined next.

Definition 4 (Semantic structure). *A* ***semantic structure****, $\mathcal{I}$, is a tuple of the form $<\boldsymbol{TV}, \boldsymbol{DTS}, \boldsymbol{D}, \boldsymbol{D}_{ind}, \boldsymbol{D}_{func}, \boldsymbol{I}_C, \boldsymbol{I}_V, \boldsymbol{I}_{psoa}, \boldsymbol{I}_{sub}, \boldsymbol{I}_{=}, \boldsymbol{I}_{external}, \boldsymbol{I}_{truth}>$. Here $\boldsymbol{D}$ is a non-empty set of elements called the* ***domain*** *of $\mathcal{I}$, and $\boldsymbol{D}_{ind}$, $\boldsymbol{D}_{func}$ are nonempty subsets of $\boldsymbol{D}$. The domain must contain at least the root class: $\top \in \boldsymbol{D}$. $\boldsymbol{D}_{ind}$ is used to interpret the elements of `Const` that play the role of individuals. $\boldsymbol{D}_{func}$ is used to interpret the constants that play the role of function symbols. As before, `Const` denotes the set of all constant symbols and `Var` the set of all variable symbols. $\boldsymbol{DTS}$ denotes a set of identifiers for primitive datatypes.*

The remaining components of $\mathcal{I}$ are total mappings defined as follows:

1. $\boldsymbol{I}_C$ *maps* `Const` *to* $\boldsymbol{D}$*. This mapping interprets constant symbols. In addition:*
   - *If a constant,* $c \in$ `Const`*, is an individual then it is required that* $\boldsymbol{I}_C(c) \in \boldsymbol{D}_{ind}$.
   - *If* $c \in$ `Const` *is a function symbol then it is required that* $\boldsymbol{I}_C(c) \in \boldsymbol{D}_{func}$.
   - *It is required that* $\boldsymbol{I}_C(\mathtt{Top}) = \top$.
2. $\boldsymbol{I}_V$ *maps* `Var` *to* $\boldsymbol{D}_{ind}$*. This mapping interprets variable symbols.*
3. $\boldsymbol{I}_{psoa}$ *maps* $\boldsymbol{D}$ *to total functions that have the general form* $\boldsymbol{D}_{ind} \times \mathtt{SetOfFiniteBags}(\boldsymbol{D}^*_{ind}) \times \mathtt{SetOfFiniteBags}(\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind}) \to \boldsymbol{D}$*. This mapping interprets psoa terms, uniformly combining positional, slotted, and frame terms, as well as class memberships. An argument* $d \in \boldsymbol{D}$ *of* $\boldsymbol{I}_{psoa}$ *uniformly represents the function or predicate symbol of positional terms and slotted terms, and the object class of frame terms, as well as the class of memberships. An element* $o \in \boldsymbol{D}_{ind}$ *represents an object of class* d*, which is described with two bags.*
   - *A finite bag of finite tuples* $\{<t_{1,1}, \ldots, t_{1,n_1}>, \ldots, <t_{m,1}, \ldots, t_{m,n_m}>\} \in \mathtt{SetOfFiniteBags}(\boldsymbol{D}^*_{ind})$*, possibly empty, represents positional information. Here* $\boldsymbol{D}^*_{ind}$ *is the set of all finite tuples over the domain* $\boldsymbol{D}_{ind}$*. Bags rather than sets of tuples are used since the order of the tuples in a psoa term is immaterial and tuples may repeat, e.g.,* `o#d([a b c][a b c])`*. Such repetitions arise through variables instantiations as explained below for slots.*
   - *A finite bag of attribute-value pairs* $\{<a1,v1>, \ldots, <ak,vk>\} \in \mathtt{SetOfFiniteBags}(\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind})$*, possibly empty, represents slotted information. Bags are again used since the order of the attribute-value pairs in a psoa term is immaterial and pairs may repeat, e.g.,* `o#d(a->b a->b)`*. Such repetitions arise naturally when variables are instantiated with constants. For instance,* `o#d(?A->?B ?C->?D)` *becomes* `o#d(a->b a->b)` *if variables* `?A` *and* `?C` *are instantiated with the symbol* `a` *and* `?B`*,* `?D` *with* `b`*. (We shall see later that* `o#d(a->b a->b)` *is actually equivalent to* `o#d(a->b)`*.)*

   In addition:
   - *If* $d \in \boldsymbol{D}_{func}$ *then* $\boldsymbol{I}_{psoa}(d)$ *must be a* ($\boldsymbol{D}_{ind}$*-valued) function* $\boldsymbol{D}_{ind} \times \mathtt{SetOfFiniteBags}(\boldsymbol{D}^*_{ind}) \times \mathtt{SetOfFiniteBags}(\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind}) \to \boldsymbol{D}_{ind}$.
   - *This implies that when a function symbol is applied to arguments that are individual objects then the result is also an individual object.*

   We will see shortly how $\boldsymbol{I}_{psoa}$ *is used to determine the truth valuation of psoa terms.*
4. $\boldsymbol{I}_{sub}$ *gives meaning to the subclass relationship. It is a total mapping of the form* $\boldsymbol{D}_{func} \times \boldsymbol{D}_{func} \to \boldsymbol{D}$.

   An additional restriction in Section 3.2 ensures that the operator `##` *is transitive, i.e., that* `c1 ## c2` *and* `c2 ## c3` *imply* `c1 ## c3`*.*
5. $\boldsymbol{I}_=$ *is a mapping of the form* $\boldsymbol{D}_{ind} \times \boldsymbol{D}_{ind} \to \boldsymbol{D}$*. It gives meaning to the equality operator.*

6. *$\boldsymbol{I}_{external}$ is a mapping that is used to give meaning to* `External` *terms. It maps symbols in* `Const` *designated as external to fixed functions of appropriate arity. Typically, external terms are invocations of built-in functions or predicates, and their fixed interpretations are determined by the specification of those built-ins.*
7. *$\boldsymbol{I}_{truth}$ is a mapping of the form $\boldsymbol{D} \rightarrow \boldsymbol{TV}$. It is used to define truth valuation for formulas.*

We also define the following generic mapping from terms to $\boldsymbol{D}$, which we denote by $\boldsymbol{I}$.

- *$\boldsymbol{I}$(k) = $\boldsymbol{I}_C$(k), if* `k` *is a symbol in* `Const`
- *$\boldsymbol{I}$(?v) = $\boldsymbol{I}_V$(?v), if* `?v` *is a variable in* `Var`
- *$\boldsymbol{I}$(o#f([$t_{1,1}$ ... t_{1,n_1}] ... [$t_{m,1}$ ... t_{m,n_m}] a_1->v_1 ... a_k->v_k)) =*
  $\boldsymbol{I}_{psoa}$($\boldsymbol{I}$(f))($\boldsymbol{I}$(o),
  {<$\boldsymbol{I}$($t_{1,1}$), ..., $\boldsymbol{I}$(t_{1,n_1})>, ..., <$\boldsymbol{I}$($t_{m,1}$), ..., $\boldsymbol{I}$(t_{m,n_m})>},
  {<$\boldsymbol{I}$(a_1),$\boldsymbol{I}$(v_1)>, ..., <$\boldsymbol{I}$(a_k),$\boldsymbol{I}$(v_k)>})
  Here {...} again denote bags *of tuples and attribute-value pairs. Section 3.2 will show that duplicate elements in such a bag do not affect the value of $\boldsymbol{I}_{psoa}$($\boldsymbol{I}$(f)). For instance, $\boldsymbol{I}$(o#f(a->b a->b)) = $\boldsymbol{I}$(o#f(a->b)).*
- *$\boldsymbol{I}$(c1##c2) = $\boldsymbol{I}_{sub}$($\boldsymbol{I}$(c1), $\boldsymbol{I}$(c2))*
- *$\boldsymbol{I}$(x=y) = $\boldsymbol{I}_{=}$($\boldsymbol{I}$(x), $\boldsymbol{I}$(y))*
- *$\boldsymbol{I}$(External(p(s_1 ... s_n))) = $\boldsymbol{I}_{external}$(p)($\boldsymbol{I}$($s_1$), ..., $\boldsymbol{I}$(s_n)).*

In addition, PSOA RuleML imposes certain restrictions on datatypes so that they would be interpreted as intended (for instance, that the constants in the symbol space `xs:integer` *are interpreted by integers). Details are found in [BK10b].*
□

3.2 Formula Interpretation

This section establishes how semantic structures determine the truth value of PSOA RuleML formulas other than document formulas. Truth valuation of document formulas is as defined in RIF-BLD [BK10a]. Here we define a mapping, $TVal_{\mathcal{I}}$, from the set of all non-document formulas to $\boldsymbol{TV}$.

Observe that in case of an atomic formula ϕ, $TVal_{\mathcal{I}}(\phi)$ is defined essentially as $\boldsymbol{I}_{\mathrm{truth}}(\boldsymbol{I}(\phi))$. Recall that $\boldsymbol{I}(\phi)$ is just an element of the domain $\boldsymbol{D}$ and $\boldsymbol{I}_{\mathrm{truth}}$ maps $\boldsymbol{D}$ to truth values in $\boldsymbol{TV}$. This might surprise those used to textbook-style definitions, since normally the mapping $\boldsymbol{I}$ is defined only for terms that occur as arguments to predicates, not for atomic formulas. Similarly, truth valuations are usually defined via mappings from instantiated formulas to $\boldsymbol{TV}$, not from the interpretation domain $\boldsymbol{D}$ to $\boldsymbol{TV}$. This HiLog-style definition [CKW93] is inherited from RIF-FLD [BK10c] and is equivalent to a standard one for first-order languages such as RIF-BLD and PSOA RuleML. In RIF-FLD, this style of definition is a provision for enabling future RIF dialects that support higher-order features, such as those of HiLog, Relfun, and FLORA-2 [YKZ03].

Definition 5 (Truth valuation). ***Truth valuation*** *for well-formed formulas in PSOA RuleML is determined using the following function, denoted* $TVal_{\mathcal{I}}$*:*

1. *Equality:* $TVal_{\mathcal{I}}(\mathtt{x = y}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\mathtt{x = y}))$.
   - *To ensure that equality has precisely the expected properties, it is required that:*
     $\boldsymbol{I}_{truth}(\boldsymbol{I}(\mathtt{x = y})) = \boldsymbol{t}$ *if* $\boldsymbol{I}(x) = \boldsymbol{I}(y)$ *and that* $\boldsymbol{I}_{truth}(\boldsymbol{I}(\mathtt{x = y})) = \boldsymbol{f}$ *otherwise.*
   - *This can also be expressed as* $TVal_{\mathcal{I}}(\mathtt{x = y}) = \boldsymbol{t}$ *if and only if* $\boldsymbol{I}(x) = \boldsymbol{I}(y)$.
2. *Subclass:* $TVal_{\mathcal{I}}(\mathtt{sc\ \#\#\ cl}) = \boldsymbol{I}_{truth}(\boldsymbol{I}(\mathtt{sc\ \#\#\ cl}))$.
   In particular, for the root class, `Top`*, and all* $\mathtt{sc} \in \boldsymbol{D}$, $TVal_{\mathcal{I}}(\mathtt{sc\ \#\#\ Top}) = \boldsymbol{t}$.
   To ensure that `##` *is transitive, i.e.,* `c1 ## c2` *and* `c2 ## c3` *imply* `c1 ## c3`*, the following is required:*
   - *For all* `c1`, `c2`, $\mathtt{c3} \in \boldsymbol{D}$, *if* $TVal_{\mathcal{I}}(\mathtt{c1\ \#\#\ c2}) = TVal_{\mathcal{I}}(\mathtt{c2\ \#\#\ c3}) = \boldsymbol{t}$ *then* $TVal_{\mathcal{I}}(\mathtt{c1\ \#\#\ c3}) = \boldsymbol{t}$.
3. *Psoa formula:*
   $TVal_{\mathcal{I}}(\mathtt{o\#f([}t_{1,1} \ldots t_{1,n_1}\mathtt{]} \ \ldots \ \mathtt{[}t_{m,1} \ldots t_{m,n_m}\mathtt{]} \ a_1\mathtt{->}v_1 \ \ldots \ a_k\mathtt{->}v_k\mathtt{))} =$
   $\boldsymbol{I}_{truth}(\boldsymbol{I}(\mathtt{o\#f([}t_{1,1} \ldots t_{1,n_1}\mathtt{]} \ \ldots \ \mathtt{[}t_{m,1} \ldots t_{m,n_m}\mathtt{]} \ a_1\mathtt{->}v_1 \ \ldots \ a_k\mathtt{->}v_k\mathtt{)}))$.
   *Since the formula consists of an object-typing membership, a bag of tuples representing a conjunction of all the object-centered tuples (*tupribution*), and a bag of slots representing a conjunction of all the object-centered slots (*slotribution*), the following restriction is used, where* $\mathtt{m} \geq 0$ *and* $\mathtt{k} \geq 0$*:*
   - $TVal_{\mathcal{I}}(\mathtt{o\#f([}t_{1,1} \ldots t_{1,n_1}\mathtt{]} \ \ldots \ \mathtt{[}t_{m,1} \ldots t_{m,n_m}\mathtt{]} \ a_1\mathtt{->}v_1 \ldots \ a_k\mathtt{->}v_k\mathtt{))} = \boldsymbol{t}$
     if and only if
     $TVal_{\mathcal{I}}(\mathtt{o\ \#\ f}) =$
     $TVal_{\mathcal{I}}(\mathtt{o\#Top([}t_{1,1} \ldots t_{1,n_1}\mathtt{]))} = \ldots = TVal_{\mathcal{I}}(\mathtt{o\#Top([}t_{m,1} \ldots t_{m,n_m}\mathtt{]))} =$
     $TVal_{\mathcal{I}}(\mathtt{o\#Top(}a_1\mathtt{->}v_1\mathtt{))} = \ldots = TVal_{\mathcal{I}}(\mathtt{o\#Top(}a_k\mathtt{->}v_k\mathtt{))} =$
     $\boldsymbol{t}$.
     Observe that on the right-hand side of the "if and only if" there are $1+\mathtt{m}+\mathtt{k}$ *subformulas splitting the left-hand side into an object membership,* `m` *object-centered positional formulas, each associating the object with a tuple, and* `k` *object-centered slotted formulas, i.e. 'triples', each associating the object with an attribute-value pair. All parts on both sides of the "if and only if" are centered on the object* `o`*, which connects the subformulas on the right-hand side (the first subformula providing the* `o`*-member class* `f`*, the remaining* $\mathtt{m}+\mathtt{k}$ *ones using the root class* `Top`*).*

   For the root class, `Top`*, and all* $\mathtt{o} \in \boldsymbol{D}$, $TVal_{\mathcal{I}}(\mathtt{o\ \#\ Top}) = \boldsymbol{t}$.
   To ensure that all members of a subclass are also members of its superclasses, i.e., `o # f` *and* `f ## g` *imply* `o # g`*, the following restriction is imposed:*
   - *For all* `o`, `f`, $\mathtt{g} \in \boldsymbol{D}$, *if* $TVal_{\mathcal{I}}(\mathtt{o\ \#\ f}) = TVal_{\mathcal{I}}(\mathtt{f\ \#\#\ g}) = \boldsymbol{t}$ *then* $TVal_{\mathcal{I}}(\mathtt{o\ \#\ g}) = \boldsymbol{t}$.
4. *Externally defined atomic formula:* $TVal_{\mathcal{I}}(\mathtt{External(t)}) = \boldsymbol{I}_{truth}(\boldsymbol{I}_{external}(\mathtt{t}))$.
5. *Conjunction:* $TVal_{\mathcal{I}}(\mathtt{And(}c_1 \ldots c_n\mathtt{))} = \boldsymbol{t}$ *if and only if* $TVal_{\mathcal{I}}(c_1) = \ldots = TVal_{\mathcal{I}}(c_n) = \boldsymbol{t}$. *Otherwise,* $TVal_{\mathcal{I}}(\mathtt{And(}c_1 \ldots c_n\mathtt{))} = \boldsymbol{f}$. *The empty conjunction is treated as a tautology:* $TVal_{\mathcal{I}}(\mathtt{And()}) = \boldsymbol{t}$.

6. *Disjunction:* $TVal_{\mathcal{I}}$*(*`Or`*(*c_1 ... c_n*))* $= \boldsymbol{f}$ *if and only if* $TVal_{\mathcal{I}}(c_1) = \ldots = TVal_{\mathcal{I}}(c_n) = \boldsymbol{f}$*. Otherwise,* $TVal_{\mathcal{I}}$*(*`Or`*(*c_1 ... c_n*))* $= \boldsymbol{t}$*. The empty disjunction is treated as a contradiction:* $TVal_{\mathcal{I}}$*(*`Or`*())* $= \boldsymbol{f}$*.*
7. *Quantification:*
   - $TVal_{\mathcal{I}}$*(*`Exists` *?*v_1 ... *?*v_n *(*φ*))* $= \boldsymbol{t}$ *if and only if for some* $\mathcal{I}^*$*, described below,* $TVal_{\mathcal{I}^*}(\varphi) = \boldsymbol{t}$*.*
   - $TVal_{\mathcal{I}}$*(*`Forall` *?*v_1 ... *?*v_n *(*φ*))* $= \boldsymbol{t}$ *if and only if for every* $\mathcal{I}^*$*, described below,* $TVal_{\mathcal{I}^*}(\varphi) = \boldsymbol{t}$*.*

   Here $\mathcal{I}^*$ *is a semantic structure of the form* $<\boldsymbol{TV}, \boldsymbol{DTS}, \boldsymbol{D}, \boldsymbol{D}_{ind}, \boldsymbol{D}_{func}, \boldsymbol{I}_C, \boldsymbol{I}^*_V, \boldsymbol{I}_{psoa}, \boldsymbol{I}_{sub}, \boldsymbol{I}_=, \boldsymbol{I}_{external}, \boldsymbol{I}_{truth}>$*, which is exactly like* $\mathcal{I}$*, except that the mapping* $\boldsymbol{I}^*_V$*, is used instead of* $\boldsymbol{I}_V$*.* $\boldsymbol{I}^*_V$ *is defined to coincide with* $\boldsymbol{I}_V$ *on all variables except, possibly, on ?*v_1*,...,?*v_n*.*
8. *Rule implication:*
   - $TVal_{\mathcal{I}}$*(conclusion :- condition)* $= \boldsymbol{t}$*, if either* $TVal_{\mathcal{I}}$*(conclusion)* $= \boldsymbol{t}$ *or* $TVal_{\mathcal{I}}$*(condition)* $= \boldsymbol{f}$*.*
   - $TVal_{\mathcal{I}}$*(conclusion :- condition)* $= \boldsymbol{f}$ *otherwise.*
9. *Groups of rules:*
   If Γ *is a group formula of the form* `Group`*(*φ_1 ... φ_n*) then*
   - $TVal_{\mathcal{I}}(\Gamma) = \boldsymbol{t}$ *if and only if* $TVal_{\mathcal{I}}(\varphi_1) = \ldots = TVal_{\mathcal{I}}(\varphi_n) = \boldsymbol{t}$*.*
   - $TVal_{\mathcal{I}}(\Gamma) = \boldsymbol{f}$ *otherwise.*

   In other words, rule groups are treated as conjunctions. □

The tupribution and slotribution in item 3 render their syntactic counterparts (cf. Example 3) unnecessary.

4 Conclusions

As a W3C Recommendation, RIF-BLD has provided a reference semantics for extensions, e.g. with negations, and for continued efforts, as described here. Implementations of RIF-BLD engines are currently being planned or developed, including as extensions to the F-logic engine Flora 2 and the POSL and RuleML engine OO jDREW. Flora 2, OO jDREW, and other engines could be extended for the PSOA RuleML semantics of this paper. A subset of PSOA RuleML with single-tuple psoa terms has already been prototyped in OO jDREW.

The PSOA RuleML syntax of this paper is built on RIF-BLD's presentation syntax, which in OO jDREW will be complemented with a generalized POSL syntax. A psoa term `o#f([`$t_{1,1}$ ... t_{1,n_1}`] ... [`$t_{m,1}$ ... t_{m,n_m}`]` p_1`->`v_1 ... p_k`->`v_k`)` corresponds to `f(o^`$t_{1,1}$, ..., t_{1,n_1}`; ...;` $t_{m,1}$, ..., t_{m,n_m}`;` p_1`->`v_1`; ...;` p_k`->`v_k`)` in POSL, where the OID moves into the argument list, separated from the other arguments by a hat infix, and tuple brackets are replaced with comma infixes that have precedence over the tuple- and slot-separating semicolon infixes. The generalization here with respect to the POSL publication [Bol10] is multi-tuple psoa terms.[8] Their PSOA RuleML/XML serialization can build on the

[8] For `m = 1` they gracefully degenerate to `f(o^`$t_{1,1}$, ..., t_{1,n_1}`;` p_1`->`v_1`; ...;` p_k`->`v_k`)`.

XML schemas of Hornlog RuleML (with some FOL RuleML) and RIF-BLD (with some RIF-FLD), adding a `<Tuple>` element, different from RuleML's `<Plex>` and RIF's `<List>`. On the other hand, POSL's *explicit* rest-slot variables are avoided through frame slotribution.

Our semantics gives a first-order model-theoretic foundation for a revised POSL and PSOA RuleML, showing how a RIF-style semantics can be adapted for them. By blending *implicit* rest slots from F-logic and RIF with integrated psoa terms from POSL and RuleML, the advantages of both rule approaches have thus been combined. This is a crucial step in RIF-RuleML convergence, which could lead to a RIF-PSOA dialect corresponding to PSOA RuleML and, ultimately, to a joint RIF-PSOA RuleML.

Future work on psoa terms includes encoding (multi-)slots and slotribution as (multi-)tuples and tupribution; conversely, tuples could be encoded as multi-list values of a `tuple` slot. Web ontologies, especially taxonomies, in OWL 2, RDF Schema, etc. could be reused for PSOA RuleML's OID type systems by alignments rooted in their classes `owl:Thing`, `rdfs:Resource`, etc. and in `Top`. While the base terms used as (function-applying) arguments of a psoa term currently are anonymous psoa terms, uses of *named base terms* could be studied. PSOA RuleML could incorporate more features of POSL such as *signature declarations.* Membership of an object, e.g. `atv1`, in multiple classes, e.g. `car` and `ship`, is written as a conjunction of psoa terms, e.g. `And(atv1#car(borne->land drive->wheel) atv1#ship(borne->water drive->propeller))`; instead using DL-style *class intersection*, e.g. `atv1#Intersect(car ship)(... slot union ...)`, may be feasible.

Further efforts concern Horn rules. Notice Example 1 is not Horn in that there is a head existential after objectification. To address this issue, it can be modified as follows.

Example 4 (Rule-extended named family frame). This Horn-rule version of Example 1 retrieves a `family` frame with a named OID variable in the premise and uses its binding to extend that frame in the conclusion (the left-hand side is objectified on the right).

```
Group (
 Forall ?Hu ?Wi ?Ch ?o (
  ?o#family(husb->?Hu wife->?Wi child->?Ch) :-
   And(?o#family(husb->?Hu wife->?Wi)
       Or(kid(?Hu ?Ch) kid(?Wi ?Ch))) )
  inst4#family(husb->Joe wife->Sue)
  kid(Sue Pete)
      )
```

```
Group (
 Forall ?Hu ?Wi ?Ch ?o ?1 ?2 (
  ?o#family(husb->?Hu wife->?Wi child->?Ch) :-
   And(?o#family(husb->?Hu wife->?Wi)
       Or(?1#kid(?Hu ?Ch) ?2#kid(?Wi ?Ch))) )
  inst4#family(husb->Joe wife->Sue)
  _1#kid(Sue Pete)
      )
```

It leads to a simpler semantics corresponding to the following set of ground facts: {*inst4#family(husb->Joe wife->Sue child->Pete)*, *_1#kid(Sue Pete)*}.

Various sublanguages of PSOA RuleML could be defined to reflect Horn rules and other restrictions, both syntactic and semantic. It will be interesting to precisely align these with existing RuleML sublanguages as well as RIF dialects. While the current PSOA RuleML is closest to Hornlog RuleML and RIF-BLD, its integrated psoa terms with implicit rest slots could be 'lifted' to full FOL RuleML and RIF-FLD as well as 'lowered' to Datalog RuleML and RIF-Core, further advancing the unified RIF RuleML effort for Web rule interchange.

5 Acknowledgements

Many thanks go to Michael Kifer and all colleagues in the RuleML Technical Groups and the W3C RIF Working Group for Web rule collaboration. Also thanks to Tara Athan, Jidi Zhao, and Alexandre Riazanov for helpful discussions on drafts of this paper. Further thanks go to the RuleML-2011@IJCAI reviewers and editors. NSERC is thanked for its support through Discovery Grants.

References

[AK93] Hassan Aït-Kaci. An Introduction to LIFE: Programming with Logic, Inheritance, Functions, and Equations. In Dale Miller, editor, *Proceedings of the 1993 International Symposium on Logic Programming*, pages 52–68, Vancouver, B.C., Canada, October 1993. MIT Press.

[BK10a] Harold Boley and Michael Kifer. A Guide to the Basic Logic Dialect for Rule Interchange on the Web. *IEEE Transactions on Knowledge and Data Engineering*, 22(11):1593–1608, November 2010.

[BK10b] Harold Boley and Michael Kifer. RIF Basic Logic Dialect, June 2010. W3C Recommendation, `http://www.w3.org/TR/rif-bld`.

[BK10c] Harold Boley and Michael Kifer. RIF Framework for Logic Dialects, June 2010. W3C Recommendation, `http://www.w3.org/TR/rif-fld`.

[Bol10] Harold Boley. Integrating Positional and Slotted Knowledge on the Semantic Web. *Journal of Emerging Technologies in Web Intelligence*, 4(2):343–353, November 2010. Academy Publisher, Oulu, Finland, http://ojs.academypublisher.com/index.php/jetwi/article/view/0204343353.

[BPS10] Harold Boley, Adrian Paschke, and Omair Shafiq. RuleML 1.0: The Overarching Specification of Web Rules. In *Proc. 4th International Web Rule Symposium: Research Based and Industry Focused (RuleML-2010), Washington, DC, USA, October 2010*, Lecture Notes in Computer Science. Springer, 2010.

[CKW93] W. Chen, M. Kifer, and D.S. Warren. HiLog: A Foundation for Higher-Order Logic Programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[CL73] C.L. Chang and R.C.T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[End01] H.B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2001.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of ACM*, 42:741–843, July 1995.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer-Verlag, 1987.

[PBK10] Axel Polleres, Harold Boley, and Michael Kifer. RIF Datatypes and Built-ins 1.0, June 2010. W3C Recommendation, `http://www.w3.org/TR/rif-dtb`.

[YK03] Guizhen Yang and Michael Kifer. Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. In Stefano Spaccapietra, Salvatore T. March, and Karl Aberer, editors, *J. Data Semantics I*, volume 2800 of *Lecture Notes in Computer Science*, pages 69–97. Springer, 2003.

[YKZ03] G. Yang, M. Kifer, and C. Zhao. FLORA-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003)*, volume 2888 of *Lecture Notes in Computer Science*, pages 671–688. Springer, November 2003.