

Distributed Object-Relational Knowledge  
as  
Positional-Slotted, Object-Applicative Rules  
*Computer Science Seminar Series*  
*Faculty of Computer Science, University of New Brunswick*  
*Fredericton, Canada, 14 September 2011*

Harold Boley

Institute for Information Technology, National Research Council;  
Faculty of Computer Science, University of New Brunswick, Canada

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 slottribution, as in F-logic and RIF (as well as shelf tuptribution) with integrated psoa terms, as in POSL and RuleML

# Background: Linked Object-and-Rule Interchange

- Rules derive relations between, and slots of, objects. Objects are members of classes. A taxonomy (simple ontology) uses subclass relationships – special rules deriving a unary relation (predicate) from another one
- Linking is characteristic for modern object and rule languages on the ((Social) Semantic) Web
- Employ IRIs – Internationalized Resource Identifiers – as OIDs – Object IDentifiers – (which can also occur as slot fillers), slot keys, type identifiers, and document identifiers for object & rule bases (modules)
- IRI-linked objects and rules can thus be regarded as generalized linked data
- Therefore, object & rule language such as RuleML are usable for the exchange of information ranging from linked data sets to linked Horn/Frame logic knowledge bases to linked, object-extended first-order and higher-order logic documents

Knowledge representation & problem solving in

- AI
- the (Semantic) Web
- IT at large  
(databases, specification & programming languages)

can be

- 1 Relational (and logic-based):  
FOL, Horn, LP
- 2 Object-oriented (and frame-based):  
CLOS, RDF, N3

## Combined approaches

(with essence of objects – OIDs, classes, slots – in logic) :

- Description Logics (DLs)
- Object-Oriented Databases (OODBs) /  
Deductive Object-Oriented Databases (DOODs)
- Object-oriented logic languages:  
LIFE and Frame logic (F-logic)
- W3C Rule Interchange Format (RIF):
  - Semantics based on F-logic
  - Serialization syntax based on RuleML

- F-logic and RIF extend first-order model-theoretic semantics for **objects (frames)**
- Added separately from **function and predicate applications to arguments**
- Resulting complexity of object-extended semantics can be reduced by **integrating objects with applications**

- Integration based on **positional-slotted, object-applicative** rules of POSL and RuleML
- **F-logic's model-theoretic semantics in the style of RIF** is also the **starting point** of our **integrated semantics**
- 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 be freely combined**

# Introduction: Psoa Terms and Rules

- RuleML-2011 paper formalizes **positional-slotted, object-applicative** (*psoa*) terms and rules
- Psoa term applies **function or predicate** symbol, possibly **instantiated by object**, to zero or more **positional or slotted (named)** arguments
- For a psqa term as **atomic formula**, predicate symbol is **class (type) of object** as well as **relation between arguments**, which describe object
- Each **argument** of a psqa term can be psqa term applying **function** symbol



# Introduction: Distinctions in Psoa Taxonomy

- 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, is term `married(Joe Sue)` for binary predicate `married` applied to `Joe` and `Sue`, where positional (left-to-right) order can be used to identify husband, as 1<sup>st</sup> argument, and wife, as 2<sup>nd</sup> argument

- Psoa terms that apply a predicate symbol (as a class) to *slotted arguments* correspond to typed attribute-value descriptions
- An example is psoa term  
`family(husb->Joe wife->Sue)` or  
`family(wife->Sue husb->Joe)` for  
**family-typed attribute-value pairs (slots)**  
`{<husb, Joe>, <wife, Sue>}`
  - Easily extended with further slots, e.g. by adding children, as in `family(husb->Joe wife->Sue child->Pete)`

## Introduction: Distinctions in Psoa Taxonomy (Cont'd)

- 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
- E.g., using RIF's membership syntax  $\#$ , OID `inst1` in class `family` is describable by `inst1#family(husb->Joe wife->Sue)`, `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 ∈ family`

# Introduction: Slotted and Positional OID Description

- 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, `family`'s `husb` and `wife` slots can be positionalized as in earlier `married` example: `inst1#family (Joe Sue)` describes `inst1` with tuple `[Joe Sue]`

# Introduction: Positional-Slotted OID Description

- *Combined positional-slotted* psoa terms are allowed, similarly as in XML elements (tuple $\rightsquigarrow$ subelements, slots $\rightsquigarrow$ attributes), e.g. describing an object, as in RDF descriptions (object $\rightsquigarrow$ subject, slots $\rightsquigarrow$ properties)
- For example, `inst1#family (Joe Sue child->Pete)` **describes** `inst1` with two positional and one slotted argument

- On the other hand, 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)
```

- A **frame without explicit class** is semantically treated as **typing its object with root class**  $\top$  (syntactically, `Top`)
- For example, (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)`

# Introduction: Atom Objectification

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



- For example, slotted-fact assertion `family(husb->Joe wife->Sue)` is syntactically objectified to assertion `_#family(husb->Joe wife->Sue)`, and — if `_1` is 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)`

# Introduction: Objectified Atom Querying

- Query `family(husb->Joe)` is syntactically objectified to query `?#family(husb->Joe)`, i.e.
  - if `?1` is first new variable in `?1, ?2, ...` — to `?1#family(husb->Joe)`
- Posed against fact, it succeeds with first slot, unifying `?1` with `_1`
- Slotribution ('slot distribution') avoids POSL's 'rest-slot' variables: frame's OID 'distributes' over 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
- Consider example with rule deriving `family` frames

# Introduction: Psoa Rules Exemplified

## Example (Rule-defined anonymous family frame)

Group is used to collect a rule and two facts. Forall quantifier declares original universal argument variables and generated universal OID variables ?2, ?3, ?4. Infix :- separates conclusion from premises of rule, which derives anonymous/existential family frame from married relation And from kid relation of husband Or wife (the left-hand side is objectified on the right).

```
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, example is modeled by predicate extensions corresponding to following set of ground facts (the subdomain of individuals  $D_{ind}$  is to be defined):

$\{o\#family(husb \rightarrow Joe \ wife \rightarrow Sue \ child \rightarrow Pete)\} \cup$

$\{\_1\#married(Joe \ Sue), \_2\#kid(Sue \ Pete)\},$  where  $o \in D_{ind}$ .

PSOA RuleML is defined here as a language incorporating this integration:

- PSOA RuleML's human-readable presentation syntax
- PSOA RuleML's model-theoretic semantics
- Conclusion and future work

## Definition (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`,  $\dots \in$  `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`,  $\dots \in$  `Var`)
- The connective symbols `And`, `Or`, and `:-`
- The quantifiers `Exists` and `Forall`
- The symbols `=`, `#`, `##`, `->`, `External`, `Import`, `Prefix`, and `Base`
- The symbols `Group` and `Document`

# Presentation Syntax: Alphabet (Cont'd)

## Definition (Alphabet, Cont'd)

Constants have form `"literal"^^symospace`, where `literal` is a sequence of Unicode characters and `symospace` is an identifier for a symbol space. E.g., `"_123"^^rif:local`. Constants use shortcuts defined in RIF-DTB, including underscored `_literal` (e.g., `_123`) for above form with `symospace` specialized to `rif:local`. `Top` is a new shortcut for 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 question mark symbol

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. Symbol `External` indicates that an atomic formula or a function term is defined externally (e.g., a built-in) and symbols `Prefix` and `Base` enable abridged representations of IRIs (Internationalized Resource Identifiers)



# Presentation Syntax: Terms

In this definition, *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. Anonymous term can be *deobjectified* (by omitting `main ?#`) if its re-objectification results in old term (i.e., re-introduces `?#`).

## Definition (Term)

- 1 *Constants and variables.* If  $t \in \text{Const}$  or  $t \in \text{Var}$  then  $t$  is a **simple term**
- 2 *Equality terms.*  $t = s$  is an **equality term** if  $t, s$  are base terms
- 3 *Subclass terms.*  $t\#\#s$  is a **subclass term** if  $t, s$  are base terms
- 4 *Positional-slotted, object-applicative terms.*  
 $o\#f([t_{1,1}\dots t_{1,n_1}] \dots [t_{m,1}\dots t_{m,n_m}] p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$   
is a **positional-slotted, object-applicative (psoa) term** if  $f \in \text{Const}$  and  $o, t_{1,1}, \dots, t_{1,n_1}, \dots, t_{m,1}, \dots, t_{m,n_m}, p_1, \dots, p_k, v_1, \dots, v_k, m \geq 0, k \geq 0$ , are base terms



## Definition (Term, Cont'd)

Psoa terms can be specialized in the following way

- For  $m = 0$  they become **(typed or untyped) frame terms**  
 $\circ\#f(p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ . We consider two overlapping subcases
  - For  $k = 0$  they become **class membership terms**  $\circ\#f()$ , abridged to  $\circ\#f$ , 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  $\circ$  being the anonymous variable  $?$ , they become **anonymous frame terms (slotted terms)**  $?\#f(p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ , deobjectified  $f(p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ , corresponding to *terms with named arguments* in RIF
    - For  $f$  being the root class  $\text{Top}$ , they become **untyped frame terms**  $\circ\#\text{Top}(p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$  corresponding to *frames* in abridged form  $\circ[p_1 \rightarrow v_1 \dots p_k \rightarrow v_k]$  of F-logic and RIF, where square brackets are used instead of round parentheses

## Definition (Term, Cont'd)

- For  $m = 1$  *psoa* terms become **single-tuple *psoa* terms**
  - $\#f([t_{1,1} \dots t_{1,n_1}] p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ , abridged to
  - $\#f(t_{1,1} \dots t_{1,n_1} p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$

These can be further specialized in two ways, which can be orthogonally combined:

- For  $\circ$  being the anonymous variable  $?$ , they become **anonymous single-tuple *psoa* terms**  $? \#f(t_{1,1} \dots t_{1,n_1} p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ , deobjectified  $f(t_{1,1} \dots t_{1,n_1} p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ . These can be further specialized:
  - For  $k = 0$ , they become **positional terms**  $? \#f(t_{1,1} \dots t_{1,n_1})$ , deobjectified  $f(t_{1,1} \dots t_{1,n_1})$ , corresponding to the usual terms and atomic formulas of classical first-order logic
- For  $f$  being the root class  $\text{Top}$ , they become **untyped single-tuple *psoa* terms**  $\circ \#_{\text{Top}}(t_{1,1} \dots t_{1,n_1} p_1 \rightarrow v_1 \dots p_k \rightarrow v_k)$ . These can be further specialized:
  - For  $k = 0$ , they become **untyped single-tuple shelf terms**  $\circ \#_{\text{Top}}(t_{1,1} \dots t_{1,n_1})$  describing object  $\circ$  with positional arguments  $t_{1,1}, \dots, t_{1,n_1}$

## Definition (Term, Cont'd)

- 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 □

# Presentation Syntax: Multiple Tuples

- The notion of psoa term is generalized here from allowing a single tuple, as in POSL, to allowing a bag (multi-set) of tuples
- Together with ‘tupribution’, 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 ( $m = 1$ )
- Special case  $n_1 = \dots = n_m$  used to describe a distributed object with ‘homogeneous’ equal-length tuples of a relation: OID names extension of relation’s tuples

# Presentation Syntax: Argument Names

- Argument names of psoa terms,  $p_1, \dots, p_n$ , are base terms, e.g. constants or variables
- Since psoa terms include anonymous frames (slotted terms), this generalizes RIF, where corresponding named-argument terms must use argument names from set `ArgNames`, to reduce unification complexity
- PSOA RuleML could emulate such special treatment of slotted terms by reserving `ArgNames`-style subset of `Const`
- But, since PSOA RuleML's slotted terms via objectification become frames, they can be queried by slottribution rather than unification

# Presentation Syntax: Formulas

Any psqa term  $f(\dots)$  or  $o\#f(\dots)$ , with  $f$  a predicate symbol and  $o$  a simple term (constant or variable), is an **atomic formula**. Likewise, externally defined term  $\text{External}(\varphi)$ , where  $\varphi$  is atomic formula, equality, and subclass term. Simple terms are *not* formulas

Generalized formulas built from atomic formulas by connectives

## Definition (Formula, Condition Language)

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:

## Definition (Formula, Condition Language, Cont'd)

- 2 • *Conjunction*: If  $\varphi_1, \dots, \varphi_n, n \geq 0$ , are condition formulas then so is  $\text{And}(\varphi_1 \dots \varphi_n)$ , called a **conjunctive** formula. As special case,  $\text{And}()$  is allowed and treated as tautology, i.e., formula that is always true
- *Disjunction*: If  $\varphi_1, \dots, \varphi_n, n \geq 0$ , are condition formulas then so is  $\text{Or}(\varphi_1 \dots \varphi_n)$ , called a **disjunctive** formula. As special case,  $\text{Or}()$  is considered as contradiction, i.e., formula that is always false
- *Existentials*: If  $\varphi$  is a condition formula and  $?V_1, \dots, ?V_n, n > 0$ , are distinct variables then  $\text{Exists } ?V_1 \dots ?V_n(\varphi)$  is an **existential** formula

Condition formulas are intended as premises of rules

## Definition (Formula, Rule Language)

- ③ *Rule implication*:  $\varphi : - \psi$  is a formula, called **rule implication**, if:
- $\varphi$  is a head formula or a *conjunction* of head formulas, where a head formula is an atomic formula or an *existentially* scoped atomic formula,
  - $\psi$  is a condition formula, and
  - none of the atomic formulas in  $\varphi$  is an externally defined term (i.e., term of the form `External(...)`)
- ④ *Universal rule*: If  $\varphi$  is a rule implication and  $?V_1, \dots, ?V_n$ ,  $n > 0$ , distinct variables then `forall ?V1 ... ?Vn ( $\varphi$ )` is a *universal rule* formula. It is required that all *free* variables in  $\varphi$  occur among variables  $?V_1 \dots ?V_n$  in quantification part. Generally, an occurrence of variable  $?v$  is *free* in  $\varphi$  if it is not inside subformula of  $\varphi$  of the form `exists ?v ( $\psi$ )` and  $\psi$  is a formula. Universal rules are also referred to as *PSOA RuleML rules*.



## Definition (Formula, Rule Language, Cont'd)

- 5 **Universal fact:** If  $\varphi$  is an atomic formula and  $?V_1, \dots, ?V_n$ ,  $n > 0$ , distinct variables then  $\text{Forall } ?V_1 \dots ?V_n (\varphi)$  is a *universal fact* formula, provided that all free variables in  $\varphi$  occur among variables  $?V_1 \dots ?V_n$ .  
Universal facts are treated as rules without premises
- 6 **Group:** If  $\varphi_1, \dots, \varphi_n$  are PSOA RuleML rules, universal facts, variable-free rule implications or atomic formulas, or groups then  $\text{Group}(\varphi_1 \dots \varphi_n)$  is a *group formula*.  
Group formulas represent sets of rules and facts
- 7 **Document:** An expression of the form  $\text{Document}(\text{directive}_1 \dots \text{directive}_n \Gamma)$  is a *PSOA RuleML document formula*, if
  - $\Gamma$  is an optional group formula; it is called the group formula *associated* with the document
  - $\text{directive}_1, \dots, \text{directive}_n$  is optional sequence of *directives*.  
Can be *base directive*, *prefix directive* or *import directive* □

- Not all formulas or documents are well-formed in PSOA RuleML
- Well-formedness restriction similar to standard first-order logic: required that no constant appear in more than one **context**
- No constant symbol can occur within same document as individual or (plain or external) function or predicate in different places

# Presentation Syntax: Condition Language

## Example (PSOA RuleML conditions)

This example shows conditions that are composed of psOA terms ("Opticks" is 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)
```

# Presentation Syntax: Rule Language

## Example (PSOA RuleML business rule)

Adapts business rule from POSL logistics use case. Ternary `reciship` conclusion represents *reciprocal shippings*, at total cost (as single positional argument), between `source` and `destination` (as two slotted arguments). First two premises apply 4-ary `shipment` relation that uses anonymous cargo and named cost variables as two positional arguments, as well as `reciship`'s slotted arguments (in both 'directions').

Third premise is `External-wrapped numeric-add RIF-DTB built-in` applied on right-hand side of equality to sum up `shipment` costs for total. 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-built-in-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)
)
```

## Example (PSOA RuleML business rule, Cont'd)

The rule can be objectified as follows (Externals 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 (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)) )  
  )
```

PSOA RuleML semantics in style of RIF-BLD — more general than what would be required:

- Ensure that the semantics stay comparable
- Future RIF logic dialect could be specified to cater for PSOA
  - E.g., via an updated RIF-FLD

- In given document, *new local constant* generator, written as stand-alone  $\_$ , denotes first new local constant  $\_i$ ,  $i \geq 1$ , from the sequence  $\_1, \_2, \dots$  that does not already occur in that document
- For each document we assume OID-less psOA terms have undergone objectification, whose head existentials make PSOA RuleML non-Horn

- Use  $TV$  as set of semantic truth values  $\{\mathbf{t}, \mathbf{f}\}$
- Truth valuation of PSOA RuleML formulas will be defined as mapping  $TVal_{\mathcal{I}}$  in two steps:
  - 1 Mapping  $I$  generically bundles various mappings from semantic structure,  $\mathcal{I}$ ;  
 $I$  maps formula to element of domain  $D$
  - 2 Mapping  $I_{\text{truth}}$  takes such a domain element to  $TV$

This indirectness allows HiLog-like generality



## Definition (Semantic structure)

A **semantic structure**,  $\mathcal{I}$ , is a tuple of the form  
 $\langle TV, DTS, D, D_{ind}, D_{func}, I_C, I_V, I_{psoa}, I_{sub}, I_=, I_{external}, I_{truth} \rangle$

Here  $D$  is a non-empty set of elements called the **domain** of  $\mathcal{I}$ ,  
and  $D_{ind}, D_{func}$  are nonempty subsets of  $D$

The domain must contain at least the root class:  $\top \in D$

$D_{ind}$  is used to interpret elements of `Const` acting as individuals

$D_{func}$  is used to interpret constants acting as function symbols

As before, `Const` denotes set of all constant symbols and  
`Var` set of all variable symbols

**DTS** denotes set of identifiers for primitive datatypes

## Definition (Semantic structure, Cont'd)

The other components of  $\mathcal{I}$  are *total* mappings defined thus:

①  $I_C$  maps  $\text{Const}$  to  $D$ .

This mapping interprets constant symbols.

In addition, it is required that:

- If a constant,  $c \in \text{Const}$ , is an *individual* then it is required that  $I_C(c) \in D_{\text{ind}}$
- If  $c \in \text{Const}$  is a *function symbol* then it is required that  $I_C(c) \in D_{\text{func}}$
- It is required that  $I_C(\text{Top}) = \top$

②  $I_V$  maps  $\text{Var}$  to  $D_{\text{ind}}$ . Mapping interprets variable symbols

## Definition (Semantic structure, Cont'd)

- ③  $I_{\text{psoa}}$  maps  $\mathbf{D}$  to total functions  $\mathbf{D}_{\text{ind}} \times \text{SetOfFiniteBags}(\mathbf{D}^*_{\text{ind}}) \times \text{SetOfFiniteBags}(\mathbf{D}_{\text{ind}} \times \mathbf{D}_{\text{ind}}) \rightarrow \mathbf{D}$ . Interprets psOA terms, combining positional, slotted, and frame terms, as well as class memberships. Argument  $d \in \mathbf{D}$  of  $I_{\text{psoa}}$  represents function or predicate symbol of positional terms and slotted terms, and object class of frame terms, as well as class of memberships. Element  $o \in \mathbf{D}_{\text{ind}}$  represents object of class  $d$ , which is described with two bags.
- Finite bag of finite tuples  $\{\langle t_{1,1}, \dots, t_{1,n_1} \rangle, \dots, \langle t_{m,1}, \dots, t_{m,n_m} \rangle\} \in \text{SetOfFiniteBags}(\mathbf{D}^*_{\text{ind}})$ , possibly empty, represents positional information.  $\mathbf{D}^*_{\text{ind}}$  is set of all finite tuples over the domain  $\mathbf{D}_{\text{ind}}$ . Bags are used since order of tuples in a psOA term is immaterial and tuples may repeat
  - Finite bag of attribute-value pairs  $\{\langle a_1, v_1 \rangle, \dots, \langle a_k, v_k \rangle\} \in \text{SetOfFiniteBags}(\mathbf{D}_{\text{ind}} \times \mathbf{D}_{\text{ind}})$ , possibly empty, represents slotted information. Bags, since order of attribute-value pairs in a psOA term is immaterial and pairs may repeat

## Definition (Semantic structure, Cont'd)

- ③ In addition:
  - If  $d \in \mathbf{D}_{\text{func}}$  then  $I_{\text{psoa}}(d)$  must be a ( $\mathbf{D}_{\text{ind}}$ -valued) function  $\mathbf{D}_{\text{ind}} \times \text{SetOfFiniteBags}(\mathbf{D}_{\text{ind}}^*) \times \text{SetOfFiniteBags}(\mathbf{D}_{\text{ind}} \times \mathbf{D}_{\text{ind}}) \rightarrow \mathbf{D}_{\text{ind}}$
  - Implies that when a function symbol is applied to arguments that are individual objects then result is also individual object
- ④  $I_{\text{sub}}$  gives meaning to the subclass relationship. It is a total mapping of the form  $\mathbf{D}_{\text{func}} \times \mathbf{D}_{\text{func}} \rightarrow \mathbf{D}$
- ⑤  $I_{=}$  is a mapping of the form  $\mathbf{D}_{\text{ind}} \times \mathbf{D}_{\text{ind}} \rightarrow \mathbf{D}$ . Gives meaning to the equality operator
- ⑥  $I_{\text{external}}$  is a mapping to give meaning to `External` terms. Maps external symbols in `Const` to fixed functions
- ⑦  $I_{\text{truth}}$  is a mapping of the form  $\mathbf{D} \rightarrow \mathbf{TV}$ . Used to define truth valuation for formulas

## Definition (Semantic structure, Cont'd)

Generic mapping from terms to  $\mathbf{D}$ , denoted by  $I$

- $I(k) = I_C(k)$ , if  $k$  is a symbol in  $\text{Const}$
- $I(?v) = I_V(?v)$ , if  $?v$  is a variable in  $\text{Var}$
- $I(o \# f ([t_{1,1} \dots t_{1,n_1}] \dots [t_{m,1} \dots t_{m,n_m}] a_1 \rightarrow v_1 \dots a_k \rightarrow v_k))$   
 $= I_{\text{psoa}}(I(f))(I(o), \{<I(t_{1,1}), \dots, I(t_{1,n_1})>, \dots, <I(t_{m,1}), \dots, I(t_{m,n_m})>, \dots, <I(a_1), I(v_1)>, \dots, <I(a_k), I(v_k)>\})$

Again  $\{\dots\}$  denote *bags* of tuples and attribute-value pairs.

- $I(c1 \# \# c2) = I_{\text{sub}}(I(c1), I(c2))$
- $I(x=y) = I_{=} (I(x), I(y))$
- $I(\text{External}(p(s_1 \dots s_n))) = I_{\text{external}}(p)(I(s_1), \dots, I(s_n))$

# Semantics: Method of Formula Interpretation

- Define mapping,  $TVal_{\mathcal{I}}$ , from set of all non-document formulas to  $\mathbf{TV}$
- For atomic formula  $\phi$ ,  $TVal_{\mathcal{I}}(\phi)$  defined essentially as  $I_{\text{truth}}(I(\phi))$
- Recall that  $I(\phi)$  is just an element of domain  $\mathbf{D}$  and  $I_{\text{truth}}$  maps  $\mathbf{D}$  to truth values in  $\mathbf{TV}$
- Might surprise, since normally mapping  $I$  defined only for terms that occur as arguments to predicates, not for atomic formulas. Similarly, truth valuations usually defined via mappings from instantiated formulas to  $\mathbf{TV}$ , not from interpretation domain  $\mathbf{D}$  to  $\mathbf{TV}$
- HiLog-style definition inherited from RIF-FLD and equivalent to a standard one for first-order languages such as RIF-BLD and PSOA RuleML — but enables future higher-order features

## Definition (Truth valuation)

**Truth valuation** for well-formed formulas in PSOA RuleML determined using function  $TVal_I$ :

- 1 **Equality:**  $TVal_I(x = y) = I_{\text{truth}}(I(x = y))$ .
  - Required that  $I_{\text{truth}}(I(x = y)) = \mathbf{t}$  if  $I(x) = I(y)$  and that  $I_{\text{truth}}(I(x = y)) = \mathbf{f}$  otherwise
  - This can also be expressed as  $TVal_I(x = y) = \mathbf{t}$  if and only if  $I(x) = I(y)$

- 2 **Subclass:**  $TVal_I(sc \## cl) = I_{\text{truth}}(I(sc \## cl))$ .

In particular, for root class,  $Top$ , and all  $sc \in \mathbf{D}$ ,  
 $TVal_I(sc \## Top) = \mathbf{t}$ .

To ensure that  $\##$  is transitive, i.e.,  $c1 \## c2$  and  $c2 \## c3$  imply  $c1 \## c3$ , the following is required:

- For all  $c1, c2, c3 \in \mathbf{D}$ , if  $TVal_I(c1 \## c2) = TVal_I(c2 \## c3) = \mathbf{t}$  then  $TVal_I(c1 \## c3) = \mathbf{t}$

## Definition (Truth valuation, Cont'd)

### 3 *Psoa formula:*

$$TVal_{\mathcal{I}}(\circ \# f ([t_{1,1} \dots t_{1,n_1}] \dots [t_{m,1} \dots t_{m,n_m}] a_1 \rightarrow v_1 \dots a_k \rightarrow v_k)) = I_{\text{truth}}(I(\circ \# f ([t_{1,1} \dots t_{1,n_1}] \dots [t_{m,1} \dots t_{m,n_m}] a_1 \rightarrow v_1 \dots a_k \rightarrow v_k))).$$

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*). Hence use restriction, where  $m \geq 0$  and  $k \geq 0$ :

- $TVal_{\mathcal{I}}(\circ \# f ([t_{1,1} \dots t_{1,n_1}] \dots [t_{m,1} \dots t_{m,n_m}] a_1 \rightarrow v_1 \dots a_k \rightarrow v_k)) = \mathbf{t}$   
if and only if  

$$TVal_{\mathcal{I}}(\circ \# f) = TVal_{\mathcal{I}}(\circ \# \text{Top}([t_{1,1} \dots t_{1,n_1}])) = \dots = TVal_{\mathcal{I}}(\circ \# \text{Top}([t_{m,1} \dots t_{m,n_m}])) = TVal_{\mathcal{I}}(\circ \# \text{Top}(a_1 \rightarrow v_1)) = \dots = TVal_{\mathcal{I}}(\circ \# \text{Top}(a_k \rightarrow v_k)) = \mathbf{t}$$



## Definition (Truth valuation, Cont'd)

- 3 • Observe that on right-hand side of “if and only if” there are  $1+m+k$  subformulas splitting 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 object with attribute-value pair. All parts on both sides of “if and only if” are centered on object  $o$ , which connects subformulas on right-hand side (first subformula providing  $o$ -member class  $f$ , remaining  $m+k$  ones using root class  $\text{Top}$ )

For root class,  $\text{Top}$ , and all  $o \in \mathbf{D}$ ,  $TVal_I(o \# \text{Top}) = \mathbf{t}$ .

To ensure that all members of subclass are also members of its superclasses, i.e.,  $o \# f$  and  $f \## g$  imply  $o \# g$ , the following restriction is imposed:

- For all  $o, f, g \in \mathbf{D}$ , if  $TVal_I(o \# f) = TVal_I(f \## g) = \mathbf{t}$  then  $TVal_I(o \# g) = \mathbf{t}$

## Definition (Truth valuation, Cont'd)

- ④ *Externally defined atomic formula:*  
 $TVal_{\mathcal{I}}(\text{External}(t)) = I_{\text{truth}}(I_{\text{external}}(t))$
- ⑤ *Conjunction:*  $TVal_{\mathcal{I}}(\text{And}(c_1 \dots c_n)) = \mathbf{t}$   
if and only if  $TVal_{\mathcal{I}}(c_1) = \dots = TVal_{\mathcal{I}}(c_n) = \mathbf{t}$ .  
Otherwise,  $TVal_{\mathcal{I}}(\text{And}(c_1 \dots c_n)) = \mathbf{f}$ .  
Empty conjunction becomes tautology:  $TVal_{\mathcal{I}}(\text{And}()) = \mathbf{t}$
- ⑥ *Disjunction:*  $TVal_{\mathcal{I}}(\text{Or}(c_1 \dots c_n)) = \mathbf{f}$   
if and only if  $TVal_{\mathcal{I}}(c_1) = \dots = TVal_{\mathcal{I}}(c_n) = \mathbf{f}$ .  
Otherwise,  $TVal_{\mathcal{I}}(\text{Or}(c_1 \dots c_n)) = \mathbf{t}$ .  
Empty disjunction becomes contradiction:  $TVal_{\mathcal{I}}(\text{Or}()) = \mathbf{f}$

## Definition (Truth valuation, Cont'd)

### 7 Quantification:

- $TVal_{\mathcal{I}}(\text{Exists } ?v_1 \dots ?v_n (\varphi)) = \mathbf{t}$   
if and only if for some  $\mathcal{I}^*$ , described below,  $TVal_{\mathcal{I}^*}(\varphi) = \mathbf{t}$
- $TVal_{\mathcal{I}}(\text{Forall } ?v_1 \dots ?v_n (\varphi)) = \mathbf{t}$   
if and only if for every  $\mathcal{I}^*$ , described below,  $TVal_{\mathcal{I}^*}(\varphi) = \mathbf{t}$

Here  $\mathcal{I}^*$  is a semantic structure of the form  $\langle \mathbf{TV}, \mathbf{DTS}, \mathbf{D}, \mathbf{D}_{\text{ind}}, \mathbf{D}_{\text{func}}, \mathbf{I}_C, \mathbf{I}^*_V, \mathbf{I}_{\text{psoa}}, \mathbf{I}_{\text{sub}}, \mathbf{I}_=, \mathbf{I}_{\text{external}}, \mathbf{I}_{\text{truth}} \rangle$ , which is exactly like  $\mathcal{I}$ , except that mapping  $\mathbf{I}^*_V$  is used instead of  $\mathbf{I}_V$ .  $\mathbf{I}^*_V$  is defined to coincide with  $\mathbf{I}_V$  on all variables except, possibly, on  $?v_1, \dots, ?v_n$

## Definition (Truth valuation, Cont'd)

### 8 *Rule implication:*

- $TVal_{\mathcal{I}}(\textit{conclusion} \textit{ :- } \textit{condition}) = \mathbf{t}$ , if either  $TVal_{\mathcal{I}}(\textit{conclusion}) = \mathbf{t}$  or  $TVal_{\mathcal{I}}(\textit{condition}) = \mathbf{f}$
- $TVal_{\mathcal{I}}(\textit{conclusion} \textit{ :- } \textit{condition}) = \mathbf{f}$  otherwise

### 9 *Groups of rules:*

If  $\Gamma$  is a group formula of the form  $\text{Group}(\varphi_1 \dots \varphi_n)$  then

- $TVal_{\mathcal{I}}(\Gamma) = \mathbf{t}$  if and only if  $TVal_{\mathcal{I}}(\varphi_1) = \dots = TVal_{\mathcal{I}}(\varphi_n) = \mathbf{t}$
- $TVal_{\mathcal{I}}(\Gamma) = \mathbf{f}$  otherwise

In other words, rule groups are treated as conjunctions □

# Conclusion: From Semantics to Implementations

- W3C's RIF-BLD has provided a **reference semantics** for extensions, and for continued efforts, as described here
- Flora 2, OO jDREW, and other engines could be adapted for our PSOA RuleML semantics
- A subset of PSOA RuleML with single-tuple psoa terms has already been prototyped in OO jDREW
- Project with Alexandre Riazanov is **implementing** PSOA RuleML in Vampire Prime via TPTP

# Conclusion: Encodings and Alignments

- 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`

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

# Conclusion: Psoa Rules Made Horn

## Example (Rule-extended named family frame)

Horn version of introductory example retrieves `family` frame with named OID variable in premise and uses its binding to extend that frame in conclusion (left: given; right: objectified).

```
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)
)
```

↪ Simpler semantics corresponding to this set of ground facts:  
{*inst4#family(husb->Joe wife->Sue child->Pete)*, *\_1#kid(Sue Pete)*}