PSOA RuleML: Integrated Object-Relational Data and Rules

Harold Boley

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
`harold[DT]boley[AT]unb[DT]ca`

Abstract. Object-relational combinations are reviewed with a focus on the integrated Positional-Slotted, Object-Applicative (PSOA) RuleML. PSOA RuleML permits a predicate application (atom) to be without or with an Object IDentifier (OID) – typed by the predicate as its class – and, orthogonally, the predicate's arguments to be positional, slotted, or combined. This enables six uses of atoms, which are systematically developed employing examples in presentation syntaxes derived from RuleML/POSL and RIF-BLD, and visualized in Scratch Grailog. These atoms, asserted as facts, are retrieved by object-relational look-in queries. On top of such facts, PSOA rules and their inferential querying are explored, e.g. permitting F-logic-like frames derived from relational joins. A use case of bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping) is shown. Objectification and the presentation plus (XML-)serialization syntaxes of PSOA RuleML are described. The first-order model-theoretic semantics is formalized, blending (OID-over-)slot distribution, as in RIF, with integrated psoa terms, as in RuleML. The PSOATransRun implementation is surveyed, translating PSOA RuleML to TPTP (PSOA2TPTP) or Prolog (PSOA2Prolog).

1 Introduction

Data has recently obtained the status of what might be called "raw and processed material for all endeavors". In analogy to the many distinctions for materials (e.g., concerning, 'externally', their cost and logistics, and, 'internally', their plasticity and reactivity), both external and internal distinctions can also be made in the realm of (complex) data. External distinctions for data include "proprietary vs. *open*" (e.g., on an intranet vs. on the Internet, particularly the Web) and, orthogonally, "siloed vs. *linked*", with two popular choices in italics.[1] Internal distinctions include a couple that is often described by the contrasting data paradigms of *relations* (below: "predicate-centered, positional" data), e.g. in the SQL-queried Deep Web, vs. *graphs* (below: "object-centered, slotted" data), e.g. in the SPARQL-queried Semantic Web.

This divide has also led to separate relational and graph rule paradigms that capture knowledge for processing the data (e.g., for inferencing/reasoning with them). Projects involving both relations and graphs are thus impeded by the paradigm boundaries, from modeling to implementation. These boundaries can

[1] `http://en.wikipedia.org/wiki/Linked_open_data`.

be bridged or even dissolved by languages combining the relational and graph paradigms for data as well as rules:

– A heterogeneous combination (an amalgamation), as in F-logic [1] and RIF [2], allows atomic formulas in the separated relational and graph language paradigms for data as well as rules, possibly mixed within the same rule.
– The homogeneous combination (an integration) Positional-Slotted, Object-Applicative (PSOA) RuleML [3][2] blends the atomic relational and graph formulas themselves into a uniform kind of atom, allowing language-internal transformation of data as well as rules.

In PSOA RuleML, data, i.e. ground (variable-less) facts, include (table-row-like) relational atoms without an Object IDentifier (OID) and with positional arguments vs. (graph-node-like) graph atoms with an OID and slotted arguments (for the node's outgoing labeled edges). What we call 'slots' is often called 'attributes', 'properties', or 'roles'. Each PSOA slot can have one or more values. Rules (implications) can use non-ground (variable-containing) versions of all of the above atoms anywhere in their conditions (bodies) and conclusions (heads).

Generally, the relational vs. graph distinction can be based on two orthogonal dimensions, creating a system of four quadrants. Expanding one of the dimensions, the object-relational integration in PSOA RuleML is achieved by permitting an atom to be predicate-centered (without an OID) or object-centered[3] (with an OID) – every OID being typed by the predicate as its class – and, orthogonally, the predicate's arguments to be positional (a sequence), slotted (a bag of pairs), or both (a positional-plus-slotted combination). The resulting **p**ositional-**s**lotted **o**bject-**a**pplicative (**psoa**)[4] atoms can be used in six ways, as in this *psoa table* (quadrants 1. to 4. expanded by combined options 5. and 6.):

	predicate-centered	object-centered
positional	1. relationships	2. shelves
slotted	3. pairships	4. frames
positional+slotted	5. relpairships	6. shelframes

Of the six options, positional data are widely used under names like 'tuples', 'vectors', 'lists', and (1-dimensional) 'arrays' (mostly 1.). Likewise, slotted data include 'objects', 'records', 'maps', and 'property lists' (usually 4.). All six are illustrated with variations of the family-example atoms from [3, 4][5]:

[2] `http://wiki.ruleml.org/index.php/PSOA_RuleML`.
[3] With 'object-*centered*' rather than 'object-*oriented*' atoms we refer to atoms that have a typed OID described by slots and/or positional arguments. Object-Oriented Programming (OOP) usually only employs descriptive slots but not positional arguments; on the other hand, OOP allows the *re-assignment* of slot fillers (instance variables) while object-centered modeling – as its declarative core – only allows the *refinement* of non-ground slot fillers and – in PSOA RuleML – positional arguments.
[4] We use the upper-cased "PSOA" as a qualifier for the language and the lower-cased "psoa" for its terms.
[5] `http://wiki.ruleml.org/index.php/Grailog#Family_Example`.

1. Predicate-centered, positional atoms (relationships), without an OID and with an – ordered – sequence of arguments, e.g. a $Husb \times Wife$ relationship `family(Joe Sue)`
2. Object-centered, positional atoms (shelves), with an OID and with a sequence of arguments, e.g. `inst1#family(Joe Sue)` with `family`-typed OID `inst1`
3. Predicate-centered, slotted atoms (pairships), without an OID and with an – unordered – multi-set of slots (each a pair of a slot name and a slot filler), e.g. `family(husb->Joe wife->Sue)` or `family(wife->Sue husb->Joe)`
4. Object-centered, slotted atoms (frames), with an OID and with a multi-set of slots, e.g. `inst1#family(husb->Joe wife->Sue)` or commuted (as in 3.)
5. Predicate-centered, positional+slotted atoms (relpairships), without an OID and with both a sequence of arguments and a multi-set of slots, e.g. a 3-slot, 2-argument atom `family(child->Pete dog->Fido dog->Toby Joe Sue)`
6. Object-centered, positional+slotted atoms (shelframes), with an OID and with both an argument sequence and a slot multi-set, e.g. an `inst1`-identified atom (cf. 5.) `inst1#family(child->Pete dog->Fido dog->Toby Joe Sue)`

The original family-example rule from [3] illustrates one combination of these six uses of psoa atoms in conditions and conclusions: Its predicate-centered, positional atoms in the condition (1.) derive a predicate-centered, slotted atom in the conclusion (3.). The following family-rule variant illustrates a conjunction of two predicate-centered, positional atoms – a relational join – deriving an object-centered, slotted atom (4.) – an F-logic-like frame – with the application of a fresh function name, `famid`, to `?Hu` and `?Wi` denoting the OID dependent on them but not on `?Ch` (in this preview, free variables are assumed to be universal):

```
famid(?Hu ?Wi)#family(husb->?Hu wife->?Wi child->?Ch) :-
                                          And(married(?Hu ?Wi) kid(?Wi ?Ch))
```

With its OID function, this rule crosses from Datalog to Horn-logic expressivity.

PSOA RuleML is a (head-existential-)extended Horn-logic language (with equality) that systematizes the variety of RIF-BLD terms[6] by generalizing its positional and slotted ("named-argument") terms as well as its frame and membership terms. It can be extended in various ways, e.g. with Negation As Failure (NAF), augmenting RuleML's MYNG configurator [5] for the syntax and adapting the RIF-FLD-specified NAF dialects for the semantics. Conversely, PSOA RuleML is being developed as a module that is pluggable into larger (RuleML) logic languages, thus making them likewise object-relational (cf. Section 6).

This paper gives a tutorial-style overview of PSOA RuleML, spanning from conceptual foundation, to data model, to fact and rule querying, to use case, to syntax, to semantics, to implementation. Specifically, the paper:

– visualizes all psoa terms in Grailog, where (n-ary) directed hyperarcs [4] – of directed hypergraphs – are used for positional terms, and (binary) directed arcs – of directed 'graphs' in the narrow sense – are used for slotted terms;
– uses ('functional') *terms* `p(...)` with a predicate symbol `p`, taking them as atomic *formulas* as in Relfun, HiLog, and RIF, which – along with equality – is a basis for universal functional-logic programming as in Curry [6];

[6] `http://www.w3.org/TR/rif-bld/#Terms`.

– is about instance frames (frame atoms) and other psoa atoms employed as queries and facts, as well as about rules having frames etc. as their conditions and/or conclusions; it is not about (signature) declarations, as e.g. for frames in F-logic; however, integrity rules can be defined over arbitrary psoa terms, as e.g. for relationships in Dexter [7];
– uses ordinary constants as Object IDentifiers, which can logically connect (distributed) frames and other psoa atoms describing the same OID, e.g. after disassembling (slotributing) a frame into its smallest (RDF-triple-like) single-slot parts at compile- or interpretation/run-time;
– uses class membership $oid \in class$ (written RIF-like: *oid*#*class*) as the 'backbone' of (typed) frames etc., where a missing *oid* is provided by the system (e.g. as a Skolem constant or existential variable) and the absence of *class* typing is expressed by the `Top` class, specifying the root of the class hierarchy;
– is only about (monotonically) deriving new frames etc., and does not go into negation (as failure) or into frame retraction or updating, although the latter operations can again use OIDs to refer to frames (cf. N3 [8]);
– focuses on an SQL-SPARQL interoperation use case about (sub)addresses (Section 5), while other use cases are about clinical intelligence [9], music albums[7], and geospatial rules [10][8].

This section introduced object-relational combinations, focused on the PSOA RuleML integration. Next, the paper develops the PSOA data model with a systematically varied example in presentation syntaxes derived from RuleML/POSL and RIF-BLD, and in a neat Grailog visualization syntax. Subsequently, such ground atoms are asserted as ground facts and queried by ground or non-ground atoms, followed by a non-ground OID-existential PSOA fact and its querying. Based on similar facts, PSOA rules and their querying are being explored. The paper then shows a use case of bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping). It continues with defining objectification as well as the presentation and serialization syntaxes of PSOA RuleML. Next, it formalizes the model-theoretic semantics, blending (OID-over-)slot distribution, as in RIF, with integrated psoa terms, as in RuleML. Finally, the paper surveys the PSOATransRun implementation, translating PSOA RuleML knowledge bases and queries to TPTP (PSOA2TPTP) or Prolog (PSOA2Prolog).

2 Grailog-Visualized Data Model of PSOA RuleML

The data model of PSOA RuleML, based on a long tradition of similar distinctions in the space of data, is structured by the two main (orthogonal) dimensions of "predicate-centered vs. object-centered" and "positional vs. slotted". These dimensions permit a more precise terminology than is possible, e.g., with JSON's array vs. object distinction [11][9], which (in spite of JSON's "object" notion) corresponds only to our "positional vs. slotted" dimension.

[7] `http://www.cs.unb.ca/~boley/papers/MusicAlbumKB.txt`.
[8] `http://wiki.ruleml.org/index.php/Geospatial_Rules`.
[9] `http://wiki.ruleml.org/index.php/RuleML_in_JSON`.

Our data model will be intuitively explained through a corresponding PSOA RuleML subset of Grailog [4][10] extended with *branch lines*, for multiple, 'split-out' (hyper)arcs, and using the novel *Scratch Grailog* visualization, which emphasizes connecting lines (rather than surrounding boxes). In logical languages, data are conceived as ground (variable-free) facts often given in a (symbolic) presentation syntax. We will use Grailog "skewer figures"[11] as a corresponding (graphical) visualization syntax for PSOA facts integrating relations and objects. Our (Scratch) Grailog figures will visualize the connectivity within a set of *labelnodes*,[12] where color coding will show the correspondence to the symbolic facts. The following subsections will visualize the PSOA RuleML systematics in Section 1 of six uses of facts from n-ary relationships, to frames, to integrated object-relational atoms. While the focus will be on single-tuple atoms, the generalization to multi-tuple atoms will be exemplified in Sections 2.1 and 2.2. Throughout, we will vary a running 'betweenness' example for illustration.

2.1 Predicate-Centered, Positional Atoms (Relationships)

Predicate-centered, positional atoms (often called *relationships*) represent n-ary positional information (n$\geq$0), i.e. the left-to-right-ordered connection of n arguments into a tuple, where the kind of tuple is represented by a relation name applied to the arguments. In Grailog, each relationship becomes a *directed hyperarc* (directed hyperedge),[13] which is depicted as an arrow shaft starting at the labelnode for the relation name or at a branch line, cutting through the labelnodes for the n-1 initial arguments in the order they occur, and ending with an arrow head at the labelnode for the nth argument. Labelnodes for relation names as well as for arguments can be shared by several hyperarcs.

The sample Grailog figures, right below, visualize 3-ary relational betweenness with hyperarcs (connecting four labelnodes) for two relationships applying the relation name betweenRel, in blue, to three individuals (geographic entities) as arguments, in red. The variant without branch lines can be seen as a shortcut for the variant with branch lines, shown here in preparation for extensions.[14]

The corresponding relational PSOA facts in POSL-like [12][15] and RIF-like presentation syntax, further below, employ traditional parenthesized relation applications. Here, the POSL-vs.-RIF difference is only in the use of separator

[10] `http://wiki.ruleml.org/index.php/Grailog`.

[11] The usual "stick figures" for directed graphs – connecting pairs of nodes with arrows – are generalized to "skewer figures" for directed hypergraphs – each (bendable) skewer holding arbitrarily many nodes together in a totally ordered fashion.

[12] A labelnode can be used as a label (relation) or as a node (argument).

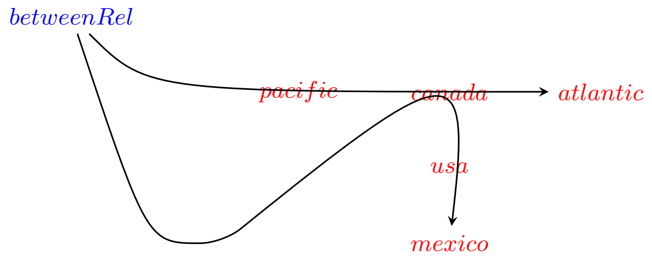
[13] In the following, "hyperarc" will be used as an abbreviation for "directed hyperarc".

[14] Branch lines permit multiple attachment points for visualizing multiple tuples [3], exemplified below, as well as (multiple) slots, to be introduced in Section 2.3.

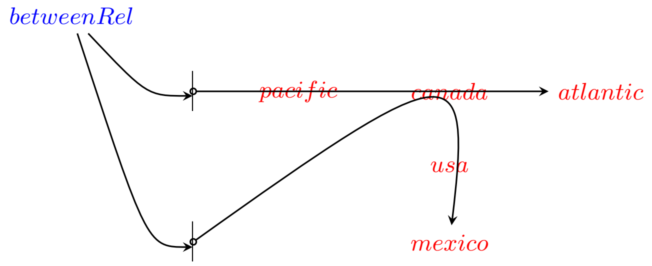
[15] The POsitional-SLotted language started integrating positional and slotted syntaxes: `http://ruleml.org/submission/ruleml-shortation.html`.

(comma vs. white-space) and terminator (period vs. newline) symbols.[16]

Grailog-style visualization syntax (without branch lines):



Grailog-style visualization syntax (with branch lines):



POSL-like presentation syntax:

betweenRel(pacific, canada, atlantic).
betweenRel(canada, usa, mexico).

RIF-like presentation syntax:

betweenRel(pacific canada atlantic)
betweenRel(canada usa mexico)

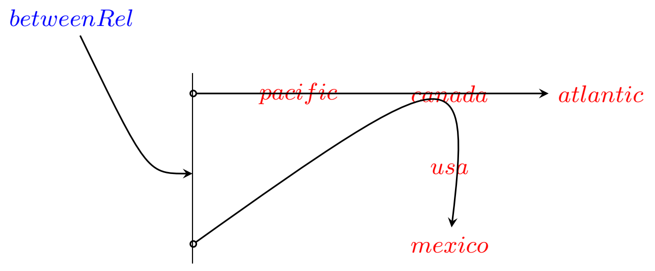
Notice that the relation name *betweenRel* as well as the argument *canada* are shared by the two hyperarcs but become copied in the two facts.

The alternative sample Grailog figure, right below, is a visualization that extends the two above vertical branch lines such that they meet, obtaining a single branch line, and uses a single unary *betweenRel* hyperarc pointing to it.

Likewise, also as for relational tables (e.g., in SQL), the multiple copies of the relation name can be avoided in the PSOA RuleML facts. The corresponding relational psoa term, further below, replaces the two separate facts for the same relation with a single multi-tuple (specifically, double-tuple) fact.

[16] This is partly due to the RIF-like presentation syntax used here being somewhat simplified w.r.t. the one used by PSOA RuleML tools: in particular, the "_" prefix is omitted from local constants, except for system-generated ones.

Grailog-style visualization syntax (with branch line):



POSL-like presentation syntax:

betweenRel(pacific, canada, atlantic; canada, usa, mexico).

RIF-like presentation syntax:

betweenRel([pacific canada atlantic] [canada usa mexico])

For the frequent case of n-ary relations with of n=1, needed in the next subsection, hyperarcs (connecting two labelnodes) point from the relation labelnode or branch line directly to the only argument labelnode.[17]

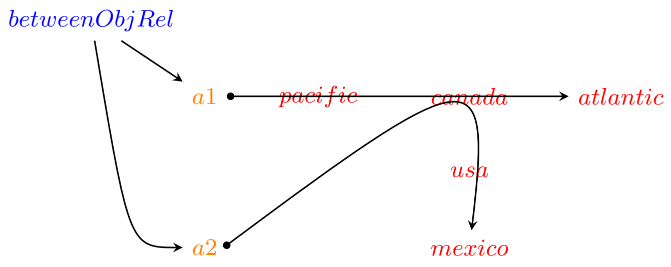
2.2 Object-Centered, Positional Atoms (Shelves)

Object-centered, positional atoms (here called *shelves*) describe an OID with n positional arguments (n≥0). A shelf thus endows an n-tuple with an OID, typed by the relation/class, keeping the positional representation of n-ary relationships in Section 2.1.

The sample Grailog figure, right below (objectifying the variant with branch lines in Section 2.1), visualizes two OIDs, *a*1 and *a*2, in orange, typed by the relation/class name *betweenObjRel*, in blue, and two 3-tuples with the three individuals as arguments, in red. The corresponding psoa term facts, further below, employ syntaxes augmenting with OIDs the parenthesized relation-application syntaxes for the three positional arguments from the relationship: The POSL-like version specifies the OID at the beginning of the argument sequence, where a hat/caret/circumflex ("^") sign – think of it as a 'property/slot insertion' character – is used as an infix separating the OID from the slots. The RIF-like version specifies the OID along with its typing relation/class, where a hash ("#") sign – think of it as a 'set/class membership' character ("∈") – is used as an infix separating the OID from the relation/class.

[17] For the infrequent case of n=0, not needed in this paper, hyperarcs ('connecting' one labelnode) degenerate to an outgoing arrow head attached to the relation labelnode or branch line.

Grailog-style visualization syntax:



POSL-like presentation syntax:

```
betweenObjRel(a1^pacific,  canada, atlantic).
betweenObjRel(a2^canada, usa,      mexico).
```

RIF-like presentation syntax:

```
a1#betweenObjRel(pacific  canada  atlantic)
a2#betweenObjRel(canada  usa      mexico)
```

This shelf version is like the relational version in the previous section (2.1) in that it keeps the three positional arguments for both hyperarcs. It is like the frame version in the section after the next (2.4) in that it introduces two relation/class-typed OIDs.

Notice that the use of the same OID for multiple facts/hyperarcs is allowed, e.g. replacing the two above OIDs, *a*1 and *a*2, with a single OID, *a*0.

POSL-like presentation syntax:

```
betweenObjRel(a0^pacific,  canada, atlantic).
betweenObjRel(a0^canada, usa,      mexico).
```

RIF-like presentation syntax:

```
a0#betweenObjRel(pacific  canada  atlantic)
a0#betweenObjRel(canada  usa      mexico)
```

Similarly as in Section 2.1, these can be merged into a single multi-tuple (specifically, double-tuple) fact.[18]

[18] Such merging of tuples – and (later) slots – centered on the same OID is called 'centralization'. It constructs one object-identified psoa term from a given set of equally identified psoa terms. Centralization will be assumed when illustrating the proof-theoretic semantics in Sections 3 and 4. It is the inverse of tupribution – and slotribution – to be introduced in Section 7. Harvesting the set of all psoa terms with a fixed OID from a distributed network – e.g. published on the Web – can use techniques analogous to finding all RDF triples having a fixed resource as their subject (cf. `http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/SemanticWebSearchEngines`). This is a non-trivial task, since such OIDs and resources normally are not dereferenceable locators themselves but occur within documents at other locators (although, ideally, those documents have filename extensions like `.ruleml` and `.rdf`, respectively).

POSL-like presentation syntax:

betweenObjRel(a0^pacific, canada, atlantic; canada, usa, mexico).

RIF-like presentation syntax:

a0#betweenObjRel([pacific canada atlantic] [canada usa mexico])

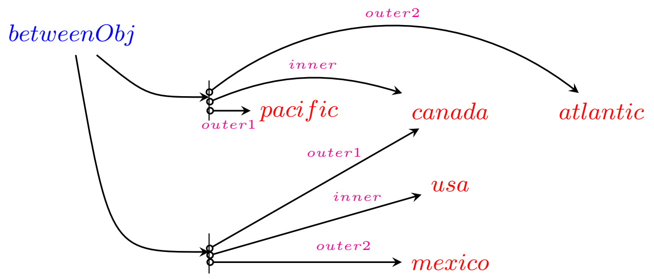
The corresponding Grailog-style visualization syntax could likewise keep the two *a*0 nodes separate, e.g. for layout purposes, or merge them into a single *a*0 node with a single unary *betweenObjRel* hyperarc pointing to it.

2.3 Predicate-Centered, Slotted Atoms (Pairships)

Predicate-centered, slotted atoms (here called *pairships*)[19] apply a relation/class to n non-positional attribute-value pairs (often called *slots*) (n≥0). In Grailog, a pairship is depicted as a relation/class node pointing, with a unary hyperarc, to a branch line having n outgoing circle-shaft slot arrows, each using a label for the attribute and a target node for the value. The order in which slot arrows emanate from a branch line is immaterial (like for arrows emanating from a node, as in Section 2.4).

The sample Grailog figure, right below (re-representing the same information as the version in Section 2.1), visualizes 3-slot betweenness of two pairships that apply the relation name *betweenObj*, in blue, to a branch line for three slots, with labels *outer*1, *inner*, and *outer*2, in magenta, targeting three individuals as values, in red. The corresponding pairship facts, further below, employ syntaxes modifying the relationship syntax of Section 2.1: In both the POSL- and RIF-like versions, a 'dash-greater' right-arrow ("->") sign – think of it as a 'has value/filler' character ("→") – is used as an infix separating a slot attribute (name) and value (filler). As in Grailog, the order in which slots occur in an atom is immaterial. In *lexicographic normal form*, slots are ordered alphabetically according to, primarily, their names and, secondarily, their fillers.

Grailog-style visualization syntax:



POSL-like presentation syntax:

betweenObj(outer1->pacific; inner->canada; outer2->atlantic).
betweenObj(outer1->canada; inner->usa;   outer2->mexico).

[19] In RIF called "named-argument terms" [2].

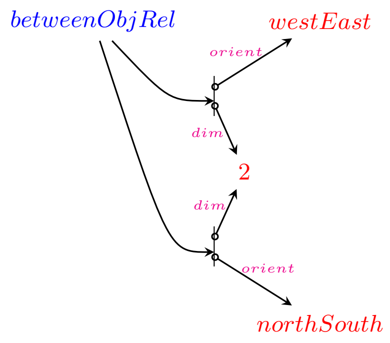
RIF-like presentation syntax:

```
betweenObj(outer1->pacific  inner->canada  outer2->atlantic)
betweenObj(outer1->canada  inner->usa      outer2->mexico)
```

The following correspondences lead from the relational version in Section 2.1 to the current version: The three positional arguments become the values (fillers) of three non-positional slots with attributes (names) outer1, inner, and outer2. The shared betweenRel becomes the shared betweenObj. The shared argument canada becomes a shared value (filler).

As a second sample, the Grailog figure, right below, visualizes two pairships that apply the relation name, *betweenObjRel*, in blue, to two slots, with labels *orient*ation and *dim*ension[20], targeting three new individuals, as values, in red. The corresponding pairship facts, further below, employ the pairship syntaxes from above.

Grailog-style visualization syntax:



POSL-like presentation syntax:

```
betweenObjRel(dim->2; orient->westEast).
betweenObjRel(dim->2; orient->northSouth).
```

RIF-like presentation syntax:

```
betweenObjRel(dim->2  orient->westEast)
betweenObjRel(dim->2  orient->northSouth)
```

This version is like the pairship version above but represents new information.

2.4 Object-Centered, Slotted Atoms (Frames)

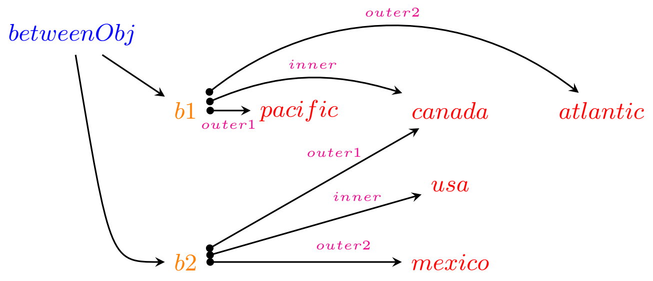
Object-centered, slotted atoms (often called *frames*) describe an OID with n non-positional attribute-value pairs (often called *slots*) (n≥0), where the kind of object is represented by a class name typing the OID. In Grailog, a frame is depicted as a typing relation/class node pointing, with a unary hyperarc, to

[20] The figure's representation of dimension = 2 indicates that this betweenness is relative to a 2D plane (rather than, say, to a 3D sphere).

a central OID node having n outgoing bullet-shaft (OID-marking) slot arrows, each using a label for the attribute and a target node for the value. A frame can thus be seen as a pairship (as in Section 2.3) enriched by an OID that results from expanding the branch line to an entire OID box, and from filling the (empty) circles of outgoing arrow shafts so they become (solid) bullets.

The sample Grailog figure, right below (OID-enriching the first figure in Section 2.3), visualizes object-centered 3-slot betweenness with central nodes, *b*1 and *b*2, in orange, for the OIDs of two frames typed by the relation name *betweenObj*, in blue, and three slots, with labels *outer*1, *inner*, and *outer*2, in magenta, targeting three individuals as values, in red. The corresponding psoa frame facts, further below, employ syntaxes enriching the pairship syntaxes.

Grailog-style visualization syntax:



POSL-like presentation syntax:

```
betweenObj(b1^outer1->pacific; inner->canada; outer2->atlantic).
betweenObj(b2^outer1->canada; inner->usa;     outer2->mexico).
```

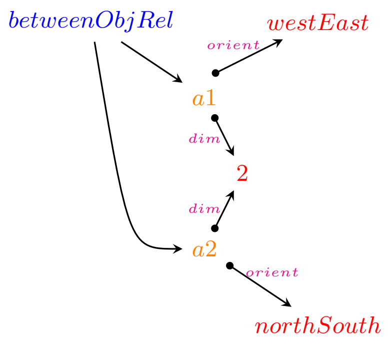
RIF-like presentation syntax:

```
b1#betweenObj(outer1->pacific  inner->canada  outer2->atlantic)
b2#betweenObj(outer1->canada   inner->usa     outer2->mexico)
```

The following correspondences lead from the relational version in Section 2.1 to the current version, complementing both of their characteristics: (1) The anonymous relationships become frames with OIDs b1 and b2. (2) The three positional arguments become the values (fillers) of three slots with attributes (names) outer1, inner, and outer2. Moreover, the shared betweenRel becomes the shared betweenObj. The shared argument canada becomes a shared value (filler).

As a second sample, the Grailog figure, right below (OID-enriching the second pairship figure of Section 2.3), again visualizes the two OIDs of Section 2.2, *a*1 and *a*2, in orange, typed by their relation/class name, *betweenObjRel*, in blue, but now described by two slots, with labels *orient* and *dim*, targeting three individuals, as values, in red. The corresponding frame facts, further below, employ the frame syntaxes from above.

Grailog-style visualization syntax:



POSL-like presentation syntax:

betweenObjRel(a1^dim->2; orient->westEast).
betweenObjRel(a2^dim->2; orient->northSouth).

RIF-like presentation syntax:

a1#betweenObjRel(dim->2 orient->westEast)
a2#betweenObjRel(dim->2 orient->northSouth)

This version is like the frame version above but introduces new information describing its OIDs.

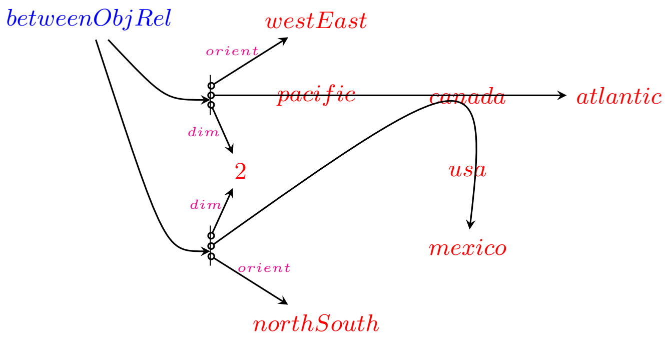
2.5 Predicate-Centered, Positional+Slotted Atoms (Relpairships)

Predicate-centered, positional+slotted atoms (here called *relpairships*) blend the relationships of Section 2.1 and pairships of Section 2.3 as follows. A branch line typed by the relation/class is shared for a relationship and a pairship. The branch line has an outgoing circle-shaft hyperarc arrow for the relationship part's tuple and outgoing circle-shaft slot arrows for the pairship part's slots. The order between the hyperarc arrow and the slot arrows emanating from a branch line is immaterial (like for arrows emanating from a node, as in Section 2.6).

The sample Grailog figure, right below, visualizes two relpairships, each composed of a relationship (as the variant with branch lines in Section 2.1) and a pairship (as in Section 2.3). The relationship parts use two branch lines, typed by the relation/class name *betweenObjRel*, in blue, and two 3-tuples with the three individuals as arguments, in red. The pairship parts use the same two branch lines, additionally having two slots each, with labels *orient* and *dim*, and targeting three further individuals, as values, in red. The corresponding relpairship facts, further below, employ an integrated syntax blending the three positional arguments from the relationship and the two slots from the pairship. As in Grailog, the order between the tuple of positional arguments and the multi-set of slots occurring in an atom is immaterial. In *left-slot normal form*, the multi-set of slots precedes the tuple. In *right-slot normal form*, it follows the tuple. These normal forms can be combined with the lexicographic normal form of Section 2.3

to, respectively, *lexicographic left-slot normal form* and *lexicographic right-slot normal form*. Here we use left-slot normal form, because it directly corresponds to the Grailog figures, while earlier papers have used right-slot normal form.

Grailog-style visualization syntax:



POSL-like presentation syntax:

```
betweenObjRel(dim->2; orient->westEast;    pacific,  canada, atlantic).
betweenObjRel(dim->2; orient->northSouth; canada, usa,       mexico).
```

RIF-like presentation syntax:

```
betweenObjRel(dim->2  orient->westEast     pacific   canada  atlantic)
betweenObjRel(dim->2  orient->northSouth  canada  usa        mexico)
```

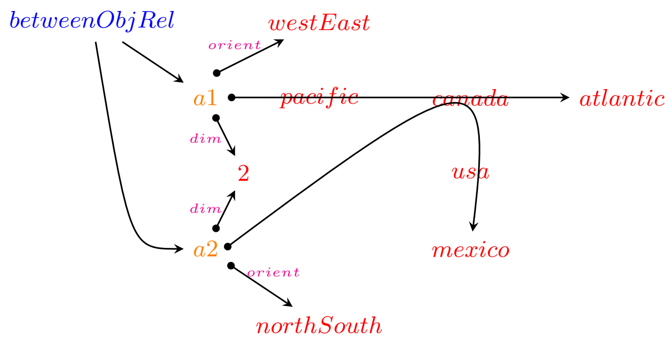
This version is a 'disjoint union' of the second pairship version in Section 2.3 and the relationship version in Section 2.1, 'plugging together' their information over the same branch lines. The graphical overlay thus becomes a logical conjunction, as expected.

2.6 Object-Centered, Positional+Slotted Atoms (Shelframes)

Object-centered, positional+slotted atoms (here called *shelframes*) blend the shelves of Section 2.2 and frames of Section 2.4 as follows. An OID typed by the relation/class is shared for a shelf and a frame. The OID is described with both the shelf's tuple and the frame's slots. Equivalently, a shelframe can be seen as a relpairship (as in Section 2.5) enriched by an OID.

The sample Grailog figure, right below, visualizes two shelframes, each composed of a shelf (as in Section 2.2) and a frame (as the second version in Section 2.4). They can also be seen as OID enrichments of the relpairships in Section 2.5. The shelf parts center on two OIDs, $a1$ and $a2$, in orange, typed by the relation/-class name *betweenObjRel*, in blue, and two 3-tuples with the three individuals as arguments, in red. The frame parts center on the same two OIDs, additionally describing each with two slots, having labels *orient* and *dim*, and targeting three further individuals, as values, in red. The corresponding shelframe facts, further below, employ an integrated syntax blending the three positional shelf arguments and the two frame slots.

Grailog-style visualization syntax:



POSL-like presentation syntax:

```
betweenObjRel(a1^dim->2; orient->westEast;    pacific,  canada, atlantic).
betweenObjRel(a2^dim->2; orient->northSouth; canada, usa,      mexico).
```

RIF-like presentation syntax:

```
a1#betweenObjRel(dim->2  orient->westEast     pacific   canada  atlantic)
a2#betweenObjRel(dim->2  orient->northSouth  canada  usa       mexico)
```

This version is a 'disjoint union' of the second frame version in Section 2.4 and the shelf version in Section 2.6, 'plugging together' their information over the same OIDs. The graphical overlay thus becomes a logical conjunction, as expected.

3 PSOA Facts for Look-In Querying

The Grailog data model of PSOA RuleML in Section 2 will serve as the foundation for PSOA RuleML fact querying discussed in the current section, which – along with rule querying in the next section – will illustrate PSOA RuleML's proof-theoretic semantics in preparation for the model-theoretic semantics discussed in Section 7.

We will introduce the notion of 'look-in' querying, which generalizes look-up querying by 'looking' for psoa query terms 'in' asserted psoa fact terms. The below definitions of 'equal to' and 'part of' for psoa fact and query terms can be understood in terms of their graph counterparts: For graph equality and parthood, the attachment order of hyperarcs and slot arrows is immaterial; this can be easily gleaned from the relevant sample visualizations in Section 2. These definitions correspond to slotribution and tupribution in the model-theoretic semantics of Section 7, Definition 5, "Psoa formula", and in the transformational semantics of Section 8: The proof-theoretic check that a query term is 'part of' a fact term becomes the model-theoretic/transformational reformulation of the query term into a conjunction of a membership term and single-slot plus single-tuple terms against the likewise reformulated fact term.

Two elementary binary relations between arbitrary psoa terms are defined:

- A psoa term t_1 is *equal to* a psoa term t_2 if t_1 and t_2 can be made (syntactically) identical by renaming any (universally or existentially) bound variables, omitting any duplicate slots (entire pairs) and argument tuples (entire sequences), and reordering any slots and argument tuples in t_1 and in t_2.
- A psoa term t_1 is *part of* a psoa term t_2 if t_1 is equal to a version of t_2 that omits zero or more slots and/or entire argument tuples from t_2. A psoa term t_1 is *proper part of* a psoa term t_2 if t_1 is part of t_2 and t_1 is not equal to t_2.

A set of PSOA ground atoms, e.g. as visualized in Section 2, can be asserted as ground facts in a Knowledge Base (KB) and then be queried by ground or non-ground atoms, some of which will succeed while others will fail. This exemplifies a basic notion of the proof-theoretic semantics with positive (success) and negative (fail) entailment tests, KB $\vdash$ q and KB $\nvdash$ q, respectively. Look-in ground and non-ground querying will be defined below, where the former is a special case of the latter.

Look-in ground querying: Consider a ground KB k and a ground query q. k $\vdash$ q (resp., k $\nvdash$ q) iff there exists (resp., does not exist) a ground fact g in k such that q is part of g.

Since positional+slotted atoms include both positional and slotted atoms, we will focus on them in the following, i.e. on Sections 2.5 and 2.6. Also, we will use the (RIF-like) symbolic presentation syntax only, hence color will be omitted.

For example, consider the following ground atom from Section 2.6, asserted as a double-slot, single-tuple ground fact in a single-fact sample KB:

```
a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)
```

This ground atom can be retrieved by issuing an identical ground atom as a ground query (special case of 'equal to'), yielding a success message; it cannot be retrieved by issuing a ground query that is not part of the ground fact, e.g. one that expects `alaska` in place of `canada`, yielding a failure message; it can again be retrieved when commuting ('equal to' as 'non-proper part of') or omitting slots and/or tuples ('proper part of'), but not when commuting the positional arguments or adding/deleting some of them within a tuple, or when inserting slots and/or tuples, or when using a different OID:

```
a2#betweenObjRel(dim->2 orient->northSouth [canada usa mexico])
success   % Desugared positional arguments with square brackets for tuple

a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)
success   % Syntactic-sugar version is identical to fact

a2#betweenObjRel(dim->2 orient->northSouth alaska usa mexico)
fail      % Different constant in same position of tuple

a2#betweenObjRel(orient->northSouth dim->2 canada usa mexico)
success   % Commuted two slots
```

```
a2#betweenObjRel(canada usa mexico dim->2 orient->northSouth)
success   % Swapped both slots with entire tuple

a2#betweenObjRel(dim->2 canada usa mexico orient->northSouth)
success   % Swapped one slot with entire tuple

a2#betweenObjRel(orient->northSouth canada usa mexico)
success   % Omitted one slot (query is proper part of fact)

a2#betweenObjRel(canada usa mexico orient->northSouth)
success   % Omitted one slot and swapped other slot with entire tuple

a2#betweenObjRel(canada usa mexico)
successs  % Omitted both slots

a2#betweenObjRel(dim->2 orient->northSouth)
success   % Omitted entire tuple

a2#betweenObjRel(orient->northSouth)
success   % Omitted entire tuple and one slot

a2#betweenObjRel()
success   % Omitted entire tuple and both slots

a2#betweenObjRel(dim->2 orient->northSouth usa canada mexico)
fail      % Commuted positional arguments of tuple

a2#betweenObjRel(dim->2 orient->northSouth alaska canada usa mexico)
fail      % Added element to tuple

a2#betweenObjRel(dim->3 orient->northSouth canada usa mexico)
fail      % Different filler for one slot

a2#betweenObjRel(dim->2 orient->northSouth start->1867 canada usa mexico)
fail      % Inserted slot

a2#betweenObjRel(dim->2 orient->northSouth
                 [canada usa mexico] [estonia latvia lithuania])
fail      % Inserted tuple

a2#betweenObjRel(dim->2 orient->northSouth usa mexico)
fail      % Deleted positional argument of tuple

a2#betweenObjRel(usa mexico)
fail      % Deleted positional argument of tuple

a1#betweenObjRel(dim->2 orient->northSouth canada usa mexico)
fail      % Different OID
```

```
betweenObjRel(dim->2 orient->northSouth canada usa mexico)
success   % Omitted OID
```

Apart from the OID-sensitive final two ground queries, all of the above ground queries work the same when omitting the OID specification, "`a2#`", both from the ground fact and from these queries; hence they also cover Section 2.5. The success of the final ground query, without an OID, against the ground fact with an OID is due to objectification to be further discussed in Section 6.

The ground atom can also be retrieved by issuing a non-ground query,[21] using non-ground/ground matching, by first 'grounding' the query by consistently substituting all query variables with corresponding ground subterms (remembering these variable bindings for the answer) and then doing retrieval with the ground query as for look-in ground querying above.

Look-in non-ground querying: Consider a ground KB k and a non-ground query q. k $\vdash$ q (resp., k $\nvdash$ q) iff there exist (resp., do not exist) a ground fact g in k and a substitution s such that s applied to q gives q' and q' is part of g.

For example, given the above sample KB, the following non-ground queries succeed with the variable bindings shown (for which `success` is understood, where variables are marked by a "`?`" prefix) or fail without a variable binding:

```
a2#betweenObjRel(dim->2 orient->northSouth ?X usa mexico)
?X = canada

a2#betweenObjRel(dim->2 orient->northSouth ?X usa ?Z)
?X = canada
?Z = mexico

a2#betweenObjRel(dim->2 orient->northSouth ?X usa ?X)
fail      % No consistent positional-argument substitution possible

a2#betweenObjRel(dim->2 orient->?V canada usa mexico)
?V = northSouth

a2#betweenObjRel(dim->?U orient->?V canada usa mexico)
?U = 2
?V = northSouth

a2#betweenObjRel(dim->?U orient->?U canada usa mexico)
fail      % No consistent slot-filler substitution possible

a2#betweenObjRel(orient->?V canada usa mexico)
?V = northSouth

a2#betweenObjRel(?S->2 orient->northSouth canada usa mexico)
?S = dim  % Slot-name variable bound to slot name
```

[21] As usual in Logic Programming, a non-ground query is understood to have existential quantification for all free variables. For basic LP terminology and notions see [13].

```
a2#betweenObjRel(?S->2 ?T->northSouth canada usa mexico)
?S = dim
?T = orient

a2#betweenObjRel(?S->2 ?S->northSouth canada usa mexico)
fail      % No consistent slot-name substitution possible

?I#betweenObjRel(dim->2 orient->northSouth canada usa mexico)
?I = a2   % OID variable bound to OID

?I#betweenObjRel(canada usa mexico)
?I = a2
```

KBs containing non-ground facts are implicit in KBs of rules (specifically, with empty bodies) to be discussed in Section 4. As an example, consider a single-fact sample KB containing the following non-ground atom modifying the one from Section 2.6, asserted as an *OID-existential fact* stating:
"Every ?M is in an ?O-identified betweenObjRel relationship – with dimension = 2 and orientation = north-to-south – of the North Pole, ?M, and the South Pole."[22]

```
Forall ?M (
  Exists ?O ( ?O#betweenObjRel(dim->2 orient->northSouth
                               northPole ?M southPole)   )
          )
```

While the earlier ground fact has an OID constant, a2, the current non-ground fact has an OID variable, ?O, that is existentially quantified in the scope of a universal variable, ?M: For each ?M binding, there is a dependent ?O binding.

This non-ground fact can be retrieved by ground queries as follows, using ground/non-ground matching:

```
a2#betweenObjRel(dim->2 orient->northSouth northPole usa southPole)
fail      % Existential fact does not assert specific OID

a1#betweenObjRel(dim->2 orient->northSouth northPole usa southPole)
fail      % Existential fact does not assert specific OID

?#betweenObjRel(dim->2 orient->northSouth northPole usa southPole)
success

?#betweenObjRel(dim->2 orient->northSouth northPole eu southPole)
success

?#betweenObjRel(dim->2 orient->northSouth northPole usa eu southPole)
fail      % Too many elements in query tuple
```

[22] Betweenness with dimension = 2 for geographical entities assumes some projection of the globe to a 2D coordinate system.

```
betweenObjRel(dim->2 orient->northSouth northPole usa southPole)
success
```

The first and second queries, employing respective constants, `a2` and `a1`, in the OID position, fail since the corresponding `Exists` variable `?O` of the fact does not need to denote them nor any named constant. In the third and fourth queries, the anonymous OID variable "`?`" causes binding-free `success` because it unifies with `?O` but prevents the creation of a (named-)variable binding.

The non-ground fact can also be retrieved by non-ground queries as follows, using non-ground/non-ground unification:

```
a2#betweenObjRel(dim->2 orient->northSouth ?X usa southPole)
?X = northPole

a2#betweenObjRel(dim->2 orient->northSouth ?X usa ?Z)
?X = northPole
?Z = southPole

a2#betweenObjRel(dim->2 orient->northSouth ?X usa ?X)
fail      % No consistent positional-argument substitution possible

?I#betweenObjRel(dim->2 orient->northSouth northPole usa southPole)
?I = skolem1(usa)

?I#betweenObjRel(northPole usa southPole)
?I = skolem2(usa)
```

In the fourth query, the OID query variable `?I` is successfully bound to a Skolem function application, `skolem1(usa)`, generated from the `Exists` by the PSOATransRun system. Similarly, in the fifth query.

4 PSOA Rules for Inferential Querying

PSOA RuleML fact querying can be done interactively by the user, as presented in Section 3, but fact – and rule – querying can also take place in the conditions of PSOA RuleML rules, as will be discussed in the current section. Using rules, the user's interactive querying becomes inferential. Rule querying is realized by resolution [13], which employs unification for consistent instantiation – ultimately, grounding – of a (possibly non-ground) query and the (possibly non-ground) conclusion of the rule to be applied to it. In PSOA RuleML, after grounding, the query must be checked to be 'part of' the rule conclusion in the sense defined in Section 3. For the PSOA sublanguage using only single-tuple psoa terms, this is similar to POSL's [12] unification involving queries that have anonymous rest slots ("`!?`"), as implemented in OO jDREW [14][23].

As we have seen, commuting of positional arguments is not supported by fact querying – and it would not make sense for arbitrary pairs of such arguments of relations like `betweenObjRel`. However, it is possible to use rule

[23] `http://www.jdrew.org/oojdrew/`.

definition and querying to selectively specify derivable properties such as the commutativity (symmetry) of certain arguments, e.g. the two outer arguments of `betweenObjRel`. We define a rule whose derived symmetric tuples are identified by the OID `symm(?O)`, depending, via a function `symm`, on the original tuples, identified by the OID `?O`. Following our fact reproduced from Sections 2.6 and 3, the rule states:

"For every `?Out1`, `?In`, `?Out2`, and `?O`, a `symm(?O)`-identified `betweenObjRel` relationship – with an extra orientation slot south-to-north – of `?Out2`, `?In`, and `?Out1` holds if an `?O`-identified `betweenObjRel` relationship – with an extra orientation slot north-to-south – of `?Out1`, `?In`, and `?Out2` holds."

```
a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)

Forall ?Out1 ?In ?Out2 ?O (
  symm(?O)#betweenObjRel(orient->southNorth ?Out2 ?In ?Out1) :-
    ?O#betweenObjRel(orient->northSouth ?Out1 ?In ?Out2)
                          )
```

The 'colon-dash' ("`:-`") sign – think of it as an 'if' symbol (a kind of "$\leftarrow$") – separates the conclusion from the condition of a rule.

In order to prevent recursive rules, e.g. for commutativity, from repeatedly undoing and redoing their own derivation results, flag-like slots such as the `orient`ation slot come in handy. On backward reasoning for query answering with our above rule, the slot filler will be switched from `southNorth` in the conclusion to `northSouth` in the condition, preventing recursive rule application. When this condition is posed as a new query, only our fact will be applicable, terminating rule derivation after one step.

This (non-ground) rule and ground fact can be used for the derivation of ground and non-ground queries as follows (intermediate derivation steps are traced using indentation):

```
symm(a2)#betweenObjRel(dim->2 orient->southNorth mexico usa canada)
fail      % Query is not part of grounded rule conclusion

symm(a2)#betweenObjRel(orient->southNorth mexico usa canada)
  a2#betweenObjRel(orient->northSouth canada usa mexico)
success   % Query is identical to grounded rule conclusion

symm(a2)#betweenObjRel(orient->southNorth mexico usa ?X)
  a2#betweenObjRel(orient->northSouth ?X usa mexico)
?X = canada

?I#betweenObjRel(orient->southNorth mexico usa ?X)
  a2#betweenObjRel(orient->northSouth ?X usa mexico)
?I = symm(a2)
?X = canada

?I#betweenObjRel(orient->southNorth ?Z usa ?X)
  a2#betweenObjRel(orient->northSouth ?X usa ?Z)
?I = symm(a2)
```

```
?X = canada
?Z = mexico

?I#betweenObjRel(orient->southNorth ?X usa ?X)
  a2#betweenObjRel(orient->northSouth ?X usa ?X)
fail      % No consistent positional-argument substitution possible

symm(?J)#betweenObjRel(orient->?V mexico usa canada)
  a2#betweenObjRel(orient->northSouth canada usa mexico)
?J = a2
?V = southNorth
```

Another rule can be used to derive new frames, specifically `GeoUnit` frames. Following our recurring fact, the rule states:

"For every `?Out1`, `?In`, `?Out2`, and `?O`, an `?In`-identified `GeoUnit` frame – with northern neighbor `?Out1` and southern neighbor `?Out2` – holds if an `?O`-identified `betweenObjRel` relationship – with an extra orientation slot north-to-south – of `?Out1`, `?In`, and `?Out2` holds."

```
a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)

Forall ?Out1 ?In ?Out2 ?O (
  ?In#GeoUnit(neighborNorth->?Out1 neighborSouth->?Out2) :-
    ?O#betweenObjRel(orient->northSouth ?Out1 ?In ?Out2)
                          )
```

This rule and fact can be used for the derivation of ground and non-ground queries as follows (where the final query asks: "Which GeoUnit `?I` has Canada as its northern neighbor?"):

```
usa#GeoUnit(neighborNorth->canada neighborSouth->mexico)
  ?O#betweenObjRel(orient->northSouth canada usa mexico)
success   % Query is identical to grounded rule conclusion

usa#GeoUnit(neighborSouth->mexico neighborNorth->canada)
  ?O#betweenObjRel(orient->northSouth canada usa mexico)
success   % Query is equal to grounded rule conclusion

usa#GeoUnit(neighborNorth->canada neighborSouth->?OutX)
  ?O#betweenObjRel(orient->northSouth canada usa ?OutX)
?OutX = mexico

usa#GeoUnit(neighborNorth->canada)
  ?O#betweenObjRel(orient->northSouth canada usa ?Out2)
success   % Query is proper part of grounded rule conclusion

mexico#GeoUnit(neighborNorth->canada)
  ?O#betweenObjRel(orient->northSouth canada mexico ?Out2)
fail      % OID cannot be proved to be in inner position

?I#GeoUnit(neighborNorth->canada)
```

```
  ?O#betweenObjRel(orient->northSouth canada ?I ?Out2)
?I = usa
```

While the first three queries specify both the `neighborNorth` and `neighborSouth` slots (albeit the third uses a free filler variable, `?OutX`), the remaining queries specify only the `neighborNorth` slot. In the subsequent derivation, the omission of the `neighborSouth` slot, hence of a fixed value for its filler, amounts to keeping that filler open as a free variable, `?Out2`.

Let us finally proceed to an *OID-(head-)existential rule*, which can be used to derive new psoa terms, specifically `compassRose` psoa terms. Following our recurring fact and an analogous one, the rule states:
"For every `?Out1`, `?Out2`, `?Out3`, `?Out4`, `?In`, `?O1`, and `?O2`, an existentially quantified `?O`-identified `compassRose` psoa term – with western, northern, eastern, and southern values `?Out1`, `?Out2`, `?Out3`, and `?Out4`, respectively – holds of `?In` if a conjunction of an `?O1`-identified `betweenObjRel` relationship – with an extra orientation slot north-to-south – of `?Out1`, `?In`, and `?Out2` holds, and an `?O2`-identified `betweenObjRel` relationship – with an extra orientation slot west-to-east – of `?Out3`, `?In`, and `?Out4` holds."

```
a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)
a3#betweenObjRel(dim->2 orient->westEast pacific usa atlantic)

Forall ?Out1 ?Out2 ?Out3 ?Out4 ?In ?O1 ?O2 (
  Exists ?O (
    ?O#compassRose(west->?Out3 north->?Out1 east->?Out4 south->?Out2 ?In)
            )                                                        :-
      And(?O1#betweenObjRel(orient->northSouth ?Out1 ?In ?Out2)
          ?O2#betweenObjRel(orient->westEast   ?Out3 ?In ?Out4))
                                            )
```

The rule conclusion uses four slots representing the cardinal compass directions, `?Out1` through `?Out4`, and a single 'positional' argument, `?In`, representing the rose center. The rule condition uses an explicit `And` conjunction for `?In`-intersecting `northSouth`- and `westEast`-oriented `betweenObjRel` relationship queries. While the first rule of this section employs a user-provided (Skolem-like) function, `symm`, for the conclusion OID, the current rule wraps the entire conclusion (head) into an existential (`Exists`) scope for the OID variable `?O`. However, our PSOATransRun implementation of this (head-)existential rule transforms the `Exists` into a system-provided Skolem function, depending on the enclosing universal variables including the condition OIDs `?O1` and `?O2`.

This rule and the two facts can be used for the derivation of ground and non-ground queries as follows:

```
a4#compassRose(west->pacific north->canada east->atlantic south->mexico usa)
fail      % Existential rule does not assert specific OID

?#compassRose(west->pacific north->canada east->atlantic south->mexico usa)
  And(a2#betweenObjRel(orient->northSouth canada  usa mexico)
      a3#betweenObjRel(orient->westEast   pacific usa atlantic))
```

```
success   % Left-slot normal query is identical to grounded rule conclusion

?#compassRose(usa west->pacific north->canada east->atlantic south->mexico)
  And(a2#betweenObjRel(orient->northSouth canada  usa mexico)
      a3#betweenObjRel(orient->westEast   pacific usa atlantic))
success   % Right-slot normal query is equal to grounded rule conclusion

?#compassRose(south->mexico west->pacific)
  And(a2#betweenObjRel(orient->northSouth ?Out1   ?In mexico)
      a3#betweenObjRel(orient->westEast   pacific ?In ?Out4))
success   % Query is proper part of grounded rule conclusion

?I#compassRose(west->pacific north->canada east->atlantic south->mexico usa)
  And(a2#betweenObjRel(orient->northSouth canada  usa mexico)
      a3#betweenObjRel(orient->westEast   pacific usa atlantic))
?I = skolem3(canada mexico pacific atlantic usa a2 a3)

?I#compassRose(west->?W north->?N east->?E south->?S ?C)
  And(a2#betweenObjRel(orient->northSouth ?N ?C ?S)
      a3#betweenObjRel(orient->westEast   ?W ?C ?E))
?I = skolem4(canada mexico pacific atlantic usa a2 a3)
?W = pacific
?N = canada
?E = atlantic
?S = mexico
?C = usa
```

The first query, while employing a new constant, `a4`, in the OID position, fails since the corresponding `Exists` variable `?O` of the rule does not need to denote it nor any named constant. In the second to fourth queries, the anonymous OID variable "`?`" causes binding-free `success` because it unifies with `?O` but prevents the creation of a (named-)variable binding. In the fifth query, the OID query variable `?I` is successfully bound to a Skolem function application, `skolem3(... a2 a3)`, generated from the `Exists` by the PSOATransRun system. Similarly, in the sixth query, which is maximally non-ground, except for its fixed `compassRose` relation/class.

5 SQL-PSOA-SPARQL Interoperation Use Case

Suppose you are working on a project using SQL queries over relational data and then proceeding to SPARQL queries over graph data to be used as a metadata repository. Or, vice versa, on a project complementing SPARQL with SQL for querying an evolving mass-data store. Or, on a project using SQL and SPARQL from the beginning. In all of these projects, object-relational interoperability issues may arise.

This section explains a use case on bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping) for interoperability. The pivotal transformation between the relational and object-centered paradigms is expressed in a language-internal manner within PSOA RuleML itself.

The use case represents addresses as (flat) relational facts and as – subaddress-containing – (nested) object-centered facts, as shown for the Seminaris address below.[24] The OID-conclusion direction of implication from the relational to the object-centered (frame) paradigm is given as the first rule below; the OID-condition direction from the object-centered (frame) to the relational paradigm is given as the second rule:

```
addressRel("Seminaris" "Takustr. 39" "14195 Berlin") % relational fact

r1#addressObj(name->"Seminaris"                      % object-centered fact
              place->r2#placeObj(street->"Takustr. 39"
                                 town->"14195 Berlin"))

Forall ?Name ?Street ?Town (                         % OID-conclusion rule
  Exists ?O1 ?O2 ( ?O1#addressObj(name->?Name
                                  place->?O2#placeObj(street->?Street
                                                      town->?Town)) )  :-
    addressRel(?Name ?Street ?Town)
                            )

Forall ?Name ?Street ?Town ?O1 ?O2 (                 % OID-condition rule
  addressRel(?Name ?Street ?Town)    :-
    ?O1#addressObj(name->?Name
                   place->?O2#placeObj(street->?Street
                                       town->?Town))
                                   )
```

While these rules define the most cross-paradigmatic cases, versions for the intermediate psoa terms could also be defined, e.g. for shelves and pairships.

Besides directly retrieving the relational fact, the OID-condition rule and the object-centered fact can be used for the derivation of relational queries as follows (corresponding to the RDF-to-RDB data mapping direction [16]):

[24] Earlier (flat and nested) positional versions have been used to explain XML-to-XML transformation (`http://www.cs.unb.ca/~boley/cs6795swt/cs6795swt-XML.pdf`). Later, a similar use case was employed to demonstrate SPINMap for RDF-to-RDF transformation [15].

```
addressRel("Seminaris" ?S "14195 Berlin")

  ?O1#addressObj(name->"Seminaris"
                 place->?O2#placeObj(street->?S
                                     town->"14195 Berlin"))

?S = "Takustr. 39"
```

Besides directly retrieving the object-centered fact, the OID-conclusion rule and the relational fact can be used for the derivation of object-centered queries as follows (corresponding to the RDB-to-RDF data mapping direction [17]):

```
?O1#addressObj(name->"Seminaris"
               place->?O2#placeObj(street->?S
                                   town->"14195 Berlin"))

  addressRel("Seminaris" ?S "14195 Berlin")

?O1 = skolem5("Seminaris" "Takustr. 39" "14195 Berlin")
?O2 = skolem6("Seminaris" "Takustr. 39" "14195 Berlin")
?S = "Takustr. 39"
```

If the object-centered PSOA RuleML fact is replaced by corresponding RDF triple facts, the OID-condition PSOA RuleML rule can also be used for the language-internal transformation of SQL-like queries to SPARQL-like queries as follows ('neutral' column headings Col$i$, with $1 \leq i \leq 3$, are used to avoid providing slot-name-like information, thus keeping SQL purely positional):[25]

```
EXISTS                                                            -- SQL
 (SELECT * FROM addressRel
  WHERE Col1='Seminaris' AND Col2='Wikingerufer 7' AND Col3='14195 Berlin')

  addressRel("Seminaris" "Wikingerufer 7" "14195 Berlin")          % PSOA

  ?O1#addressObj(name->"Seminaris"                                 % PSOA
                 place->?O2#placeObj(street->"Wikingerufer 7"
                                     town->"14195 Berlin"))

  ASK {?O1 rdf:type addressObj. ?O1 name "Seminaris".            # SPARQL
                                ?O1 place ?O2.
       ?O2 rdf:type placeObj. ?O2 street "Wikingerufer 7".
                              ?O2 town "14195 Berlin".}

fail      % Wrong street
```

[25] Alternatively, given column headings like `Name`, `Street`, and `Town`, the input conversion for PSOA could skip the relationship `addressRel("Seminaris" "Wikingerufer 7" "14195 Berlin")`, but generate a pairship `addressRel(name->"Seminaris" street->"Wikingerufer 7" town->"14195 Berlin")`, already closer to the level of frames and SPARQL.

```
EXISTS                                                          -- SQL
 (SELECT * FROM addressRel
  WHERE Col1='Seminaris' AND Col2='Takustr. 39' AND Col3='14195 Berlin')

  addressRel("Seminaris" "Takustr. 39" "14195 Berlin")            % PSOA

  ?O1#addressObj(name->"Seminaris"                                % PSOA
                 place->?O2#placeObj(street->"Takustr. 39"
                                     town->"14195 Berlin"))

  ASK {?O1 rdf:type addressObj. ?O1 name "Seminaris".           # SPARQL
                                ?O1 place ?O2.
       ?O2 rdf:type placeObj. ?O2 street "Takustr. 39".
                              ?O2 town "14195 Berlin".}

success


SELECT * FROM addressRel                                        -- SQL
WHERE Col1='Seminaris'

  addressRel("Seminaris" ?S ?T)                                   % PSOA

  ?O1#addressObj(name->"Seminaris"                                % PSOA
                 place->?O2#placeObj(street->?S
                                     town->?T))

  SELECT ?S ?T                                                  # SPARQL
  WHERE {?O1 rdf:type addressObj. ?O1 name "Seminaris".
                                  ?O1 place ?O2.
         ?O2 rdf:type placeObj. ?O2 street ?S.
                                ?O2 town ?T.}

?S = "Takustr. 39"
?T = "14195 Berlin"
```

The paradigm-crossing translation step is thus done by the OID-condition rule completely within PSOA RuleML, starting at SQL queries "lifted" to PSOA and ending at SPARQL queries "dropped" from PSOA.[26] Bridging the paradigm chasm from relations to objects constitutes one direction of PSOA RuleML's interoperation capability.

In the other direction, if the relational PSOA RuleML fact is replaced by a corresponding SQL table row, the OID-conclusion PSOA RuleML rule can be used for the language-internal transformation of SPARQL-like queries to SQL-like queries as follows:

[26] The "lift" and "drop" terminology for conversions at the input and output interfaces has been introduced in `http://yosemiteproject.org`, and is related to the "lifting" and "lowering" terminology of `http://www.w3.org/TR/sawsdl/#schemaMapping`.

```
ASK {?O1 rdf:type addressObj. ?O1 name "Seminaris".               # SPARQL
                              ?O1 place ?O2.
     ?O2 rdf:type placeObj. ?O2 street "Wikingerufer 7".
                            ?O2 town "14195 Berlin".}

  ?O1#addressObj(name->"Seminaris"                                  % PSOA
                 place->?O2#placeObj(street->"Wikingerufer 7"
                                     town->"14195 Berlin"))

  addressRel("Seminaris" "Wikingerufer 7" "14195 Berlin")           % PSOA

EXISTS                                                              -- SQL
 (SELECT * FROM addressRel
  WHERE Col1='Seminaris' AND Col2='Wikingerufer 7' AND Col3='14195 Berlin')

fail      % Wrong street


ASK {?O1 rdf:type addressObj. ?O1 name "Seminaris".               # SPARQL
                              ?O1 place ?O2.
     ?O2 rdf:type placeObj. ?O2 street "Takustr. 39".
                            ?O2 town "14195 Berlin".}

  ?O1#addressObj(name->"Seminaris"                                  % PSOA
                 place->?O2#placeObj(street->"Takustr. 39"
                                     town->"14195 Berlin"))

  addressRel("Seminaris" "Takustr. 39" "14195 Berlin")              % PSOA

EXISTS                                                              -- SQL
 (SELECT * FROM addressRel
  WHERE Col1='Seminaris' AND Col2='Takustr. 39' AND Col3='14195 Berlin')

success


SELECT ?S ?T                                                      # SPARQL
WHERE {?O1 rdf:type addressObj. ?O1 name "Seminaris".
                                ?O1 place ?O2.
       ?O2 rdf:type placeObj. ?O2 street ?S.
                              ?O2 town ?T.}

  ?O1#addressObj(name->"Seminaris"                                  % PSOA
                 place->?O2#placeObj(street->?S
                                     town->?T))

  addressRel("Seminaris" ?S ?T)                                     % PSOA
  SELECT * FROM addressRel                                          -- SQL
  WHERE Col1='Seminaris'
```

```
?S = "Takustr. 39"
?T = "14195 Berlin"
```

The reach of the PSOA-internal transformation can be increased at the PSOA/SPARQL interfaces. First, 'unnested' PSOA intermediaries are introduced as follows, where the `placeObj` frame is extracted into a conjunction, leaving behind a copy of its OID variable `?O2`:

```
And(?O1#addressObj(name->"Seminaris"
                   place->?O2)
    ?O2#placeObj(street->?S
                 town->?T))
```

These are then used to split the above 'unnesting' PSOA-to-SPARQL transformations and 'nesting' SPARQL-to-PSOA transformations into unnesting/nesting PSOA-to-PSOA transformations and clerical PSOA/SPARQL conversions. Besides for interoperation, the transformation into unnested sublanguages can also be used for the implementation of nested PSOA RuleML.

6 PSOA RuleML Syntax

The (RIF-like) presentation and (RuleML/XML) serialization syntaxes of PSOA RuleML will be discussed in this section. First, the objectification of the OID-less subset of (atomic and rule) psoa formulas in presentation syntax is introduced. Second, variants of the OID-containing superset of (atomic) psoa formulas in presentation syntax are illustrated, while details (e.g., on the variety of constants such as document-local vs. Web-IRI-global) are specified in [3]. Third, versions of PSOA RuleML serialization syntax are considered.

Object identifier assumption: An atomic formula (predicate application) without an OID is assumed to be a shorthand for this formula with an implicit OID, which is made syntactically explicit by objectification (see below) before the atomic formula is endowed with semantics (cf. Section 7). Since RIF does not make the OID assumption, it has to separately specify the semantics of its OID-less subset, mainly for "named-argument terms" (pairships).

Objectification algorithm: This may be seen as mapping the three rows of the psoa table in Section 1 to their 2nd ("object-centered") column. Basically, while a ground fact can be given a fixed OID (that the user neglected to provide), a non-ground fact or rule conclusion needs an OID for each grounding.

These formulas, when OID-less, are *objectified* by syntactic transformation: *The OID of a ground fact is a new constant generated by the 'new local constant' (a stand-alone "`_`", corresponding to "`_#`" in [18]), where each occurrence of "`_`" denotes a distinct name, not occurring elsewhere (i.e., a Skolem constant); the OID of a non-ground fact or of an atomic formula in a rule conclusion, `f(...)`, is a new, existentially scoped variable `?i`, resulting in `Exists ?i (?i#f(...))`; the OID of any other atomic formula, including in a rule condition (also usable* *as a query), is a new variable generated by the 'anonymous variable' (a standalone "?").*

In our PSOATransRun implementation (cf. Section 8), the objectification algorithm is realized as an ANTLR tree walker. Objectification transforms the three uses of psoa facts in Sections 2.1, 2.3, and 2.5 to, respectively, the three uses in Sections 2.2, 2.4, and 2.6.

For example, the relational fact `betweenRel(pacific canada atlantic)` in Section 2.1 is objectified to a shelf version like the *a*1-identified shelf fact in Section 2.2, `_#betweenRel(pacific canada atlantic)`, and – if `_1` is the first new constant from `_1`, `_2`, ... – to `_1#betweenRel(pacific canada atlantic)`. The query `betweenRel(?X canada ?Z)` is syntactically transformed to the query `?#betweenRel(?X canada ?Z)`, i.e. – if `?1` is the first new variable in `?1`, `?2`, ... – to `?1#betweenRel(?X canada ?Z)`. Posed against the fact, it succeeds, with variable bindings `?1 = _1`, `?X = pacific`, and `?Z = atlantic`.

For the general case of *(arbitrary) psoa terms* in [3], k slots and m tuples are permitted ($k \geq 0$, $m \geq 0$), with tuple i having length n_i ($1 \leq i \leq m$, $n_i \geq 0$), where we use both the left-slot and right-slot normal forms of Section 2.5 (after objectification):

left-slot $\texttt{o\#f(p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\ \texttt{[t}_{1,1} \ldots \texttt{t}_{1,n_1}\texttt{]} \ldots \texttt{[t}_{m,1} \ldots \texttt{t}_{m,n_m}\texttt{])}$
right-slot $\texttt{o\#f([t}_{1,1} \ldots \texttt{t}_{1,n_1}\texttt{]} \ldots \texttt{[t}_{m,1} \ldots \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\texttt{)}$

We distinguish three cases (explained here for the left-slot normal form):

m > 1 For *multi-tuple psoa terms*, square brackets are necessary (see above).
m = 1 For *single-tuple psoa terms*, focused in this paper, square brackets can be omitted (see `Positional+Slotted` and `Positional` below).
m = 0 For *tuple-less psoa terms*, frames arise (see `Slotted` and `Member` below).

Color coding shows syntactic variants for the cases m = 1 and k = m = 0 (single-tuple brackets and zero-argument parentheses are optional):

`Positional+Slotted:`	$\texttt{o\#f(p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\ \texttt{[t}_1 \ldots \texttt{t}_n\texttt{])}$
`Positional:`	$\texttt{o\#f(}\ \texttt{[t}_1 \ldots \texttt{t}_n\texttt{])}$
`Slotted:`	$\texttt{o\#f(p}_1\texttt{->v}_1 \ldots \texttt{p}_k\texttt{->v}_k\texttt{)}$
`Member:`	`o#f()`

An EBNF Grammar for the (RIF-like) presentation syntax of PSOA RuleML can be found in [3], Section 2.5.

Regarding the XML serialization syntax, PSOA RuleML is integrated with the earlier RuleML family as follows. We start with a pure PSOA version of the Hornlog RuleML sublanguage of the Deliberation RuleML subfamily of RuleML. We then proceed to a PSOA version extended with "positional rests" from Hornlog RuleML. We finally give a Hornlog RuleML version extended with multiple tuples from PSOA RuleML. Other sublanguages of Deliberation RuleML can also be complemented by pure and extended PSOA versions and be given multiple tuples. Similarly, for the Reaction RuleML subfamily of RuleML. The schema specification of PSOA RuleML/XML in MYNG can reflect these versions in a modular fashion.

The **pure PSOA** version of a multi-tuple psoa atom augments the content of the RuleML `<Atom>` node element[27] with `<Tuple>` node elements, different from RuleML's `<Plex>` and RIF's `<List>` elements. The above left-slot normal form results in the following XML serialization, where the primed meta-variables p'_i, v'_i, and $t'_{i,j}$ indicate recursive XML serializations of their above presentation-syntax versions (the `style` attribute uses the value `"distribution"` to specify built-in slotribution and tupribution):

```
<Atom style="distribution">
  <oid><Ind>$o$</Ind></oid><op><Rel>$f$</Rel></op>
  <slot>$p'_1$ $v'_1$</slot> . . . <slot>$p'_k$ $v'_k$</slot>
  <Tuple>$t'_{1,1}$ ... $t'_{1,n_1}$</Tuple> . . . <Tuple>$t'_{m,1}$ ... $t'_{m,n_m}$</Tuple>
</Atom>
```

The **extended PSOA** version refines the above serialization by using an optional `<repo>` edge element from Hornlog RuleML (in POSL-like presentation syntax corresponding to a "`|`" infix) to specify a "positional rest" *within* a tuple (on the level of the entire atom, the `style` attribute still specifies `"distribution"`):

```
<Atom style="distribution">
  <oid><Ind>$o$</Ind></oid><op><Rel>$f$</Rel></op>
  <slot>$p'_1$ $v'_1$</slot> . . . <slot>$p'_k$ $v'_k$</slot>
  <Tuple>$t'_{1,1}$ ... $t'_{1,n_1}$<repo>$rp'_1$</repo></Tuple> . . .
                    <Tuple>$t'_{m,1}$ ... $t'_{m,n_m}$<repo>$rp'_m$</repo></Tuple>
</Atom>
```

Conversely, Hornlog RuleML can be generalized to **multiple tuples** from PSOA RuleML without specifying PSOA's built-in tupribution, instead using a new optional `<retu>` edge to specify the "rest of tuples", i.e. *further* tuples. Combined with Hornlog RuleML's optional `<resl>` edge (in POSL-like presentation syntax corresponding to a "`!`" infix) for the "rest of slots", i.e. for *further* slots – hence without specifying built-in slotribution either – the following XML serialization is obtained (as in Hornlog RuleML, no `style` attribute is required):

```
<Atom>
  <oid><Ind>$o$</Ind></oid><op><Rel>$f$</Rel></op>
  <slot>$p'_1$ $v'_1$</slot> . . . <slot>$p'_k$ $v'_k$</slot>
  <resl>$rs'$</resl>
  <Tuple>$t'_{1,1}$ ... $t'_{1,n_1}$<repo>$rp'_1$</repo></Tuple> . . .
                    <Tuple>$t'_{m,1}$ ... $t'_{m,n_m}$<repo>$rp'_m$</repo></Tuple>
  <retu>$rt'$</retu>
</Atom>
```

When the three above serializations are re-specialized to a single-tuple psoa atom (i.e., for m=1), the `<Tuple>` ... `</Tuple>` wrapper can be just omitted.

[27] For an example-based introduction to the basic tags of Deliberation RuleML see `http://ruleml.org/papers/Primer`.

7 PSOA RuleML Semantics

The traces given in Sections 3, 4, and 5 exemplify PSOA RuleML's proof-theoretic semantics using backward reasoning directly for the PSOA sources (queries, facts, and rules). PSOA RuleML's model-theoretic semantics also involves transformations on the sources, either as a preparatory step (objectification) or as restrictions on truth valuation (slotribution and tupribution).

In the following, key parts of the semantics definitions from [3] are presented for **objectified multi-tuple psoa terms in right-slot normal form**.

Truth valuation of PSOA RuleML formulas is defined as a mapping $TVal_{\mathcal{I}}$ in two steps: 1. A mapping $\boldsymbol{I}$ generically bundles 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 the set of truth values, $\boldsymbol{TV}$.

Definition 4, case 3, as part of a ***semantic structure***, introduces the total mapping $\boldsymbol{I}_{\text{psoa}}$:
$\boldsymbol{I}_{\text{psoa}}$ maps $\boldsymbol{D}$ to total functions that have the general form $\boldsymbol{D}_{\text{ind}} \times$ `SetOfFiniteBags`$(\boldsymbol{D^*}_{\text{ind}}) \times$ `SetOfFiniteBags`$(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}) \rightarrow \boldsymbol{D}$. This mapping interprets psoa terms, uniformly combining positional, slotted, and frame terms, as well as class memberships. An argument `d` $\in \boldsymbol{D}$ of $\boldsymbol{I}_{\text{psoa}}$ 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` $\in \boldsymbol{D}_{\text{ind}}$ of the resulting total functions represents an object of class `d`, which is described with two bags.

- A finite bag of finite tuples $\{<\mathtt{t}_{1,1}, ..., \mathtt{t}_{1,\mathtt{n}_1}>, \ldots, <\mathtt{t}_{\mathtt{m},1}, ..., \mathtt{t}_{\mathtt{m},\mathtt{n}_\mathtt{m}}>\} \in$ `SetOfFiniteBags`$(\boldsymbol{D^*}_{\text{ind}})$, possibly empty, represents positional information. Here $\boldsymbol{D^*}_{\text{ind}}$ is the set of all finite tuples over the domain $\boldsymbol{D}_{\text{ind}}$.
- A finite bag of attribute-value pairs $\{<\mathtt{a1,v1}>, ..., <\mathtt{ak,vk}>\} \in$ `SetOfFiniteBags`$(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}})$, possibly empty, represents slotted information.

The generic recursive mapping $\boldsymbol{I}$ is defined from terms to their subterms and ultimately to $\boldsymbol{D}$, for the case of psoa terms using $\boldsymbol{I}_{\text{psoa}}$:
$\boldsymbol{I}(\mathtt{o\#f([t_{1,1}\ ...\ t_{1,n_1}]\ ...\ [t_{m,1}\ ...\ t_{m,n_m}]\ a_1\text{->}v_1\ ...\ a_k\text{->}v_k)}) =$
$\boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\mathtt{f}))(\boldsymbol{I}(\mathtt{o}),$
$\quad\{<\boldsymbol{I}(\mathtt{t}_{1,1}), ..., \boldsymbol{I}(\mathtt{t}_{1,\mathtt{n}_1})>, ..., <\boldsymbol{I}(\mathtt{t}_{\mathtt{m},1}), ..., \boldsymbol{I}(\mathtt{t}_{\mathtt{m},\mathtt{n}_\mathtt{m}})>\},$
$\quad\{<\boldsymbol{I}(\mathtt{a}_1),\boldsymbol{I}(\mathtt{v}_1)>, ..., <\boldsymbol{I}(\mathtt{a}_\mathtt{k}),\boldsymbol{I}(\mathtt{v}_\mathtt{k})>\})$

When, as in the below Definition 5, case 3, $\boldsymbol{I}$ is applied to a psoa term, its total function is obtained from $\boldsymbol{I}_{\text{psoa}}$ applied to the recursively interpreted class argument `f`. The application of the resulting total function to the recursively interpreted other parts of a psoa term denotes the term's interpretation in $\boldsymbol{D}$. PSOA RuleML's use of the class `f`, rather than the OID `o`, for the $\boldsymbol{I}_{\text{psoa}}$ argument is justified by the class being always user-controlled for psoa terms, even if 'defaulted' to the 'catch-all' total function obtained from $\boldsymbol{I}_{\text{psoa}}$ applied to the interpretation $\top$ of the root class `Top`. On the other hand, the OID `o` – which in RIF-BLD is used for the $\boldsymbol{I}_{\text{frame}}$ argument – need not be user-controlled in PSOA but can be system-generated via objectification, e.g. as an existential variable or a (Skolem) constant, so is not suited to obtain the total function for a psoa term.

Definition 5, cases 3 and 8, recursively define truth valuation $TVal_{\mathcal{I}}$ for psoa formulas and rule implications, based on the above $\boldsymbol{I}$ and on a mapping $\boldsymbol{I}_{\text{truth}}$ from $\boldsymbol{D}$ to $\boldsymbol{TV}$:

Psoa formula:
$TVal_{\mathcal{I}}(\texttt{o\#f([}\texttt{t}_{1,1}\ \ldots\ \texttt{t}_{1,\texttt{n}_1}\texttt{]}\ \ldots\ \texttt{[t}_{\texttt{m},1}\ \ldots\ \texttt{t}_{\texttt{m},\texttt{n}_\texttt{m}}\texttt{]}\ \texttt{a}_1\texttt{->v}_1\ \ldots\ \texttt{a}_\texttt{k}\texttt{->v}_\texttt{k}\texttt{)}) =$
$\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}(\texttt{o\#f([}\texttt{t}_{1,1}\ \ldots\ \texttt{t}_{1,\texttt{n}_1}\texttt{]}\ \ldots\ \texttt{[t}_{\texttt{m},1}\ \ldots\ \texttt{t}_{\texttt{m},\texttt{n}_\texttt{m}}\texttt{]}\ \texttt{a}_1\texttt{->v}_1\ \ldots\ \texttt{a}_\texttt{k}\texttt{->v}_\texttt{k}\texttt{)})).$
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 $\texttt{m} \geq \texttt{0}$ and $\texttt{k} \geq \texttt{0}$:

– $TVal_{\mathcal{I}}(\texttt{o\#f([}\texttt{t}_{1,1}\ \ldots\ \texttt{t}_{1,\texttt{n}_1}\texttt{]}\ \ldots\ \texttt{[t}_{\texttt{m},1}\ \ldots\ \texttt{t}_{\texttt{m},\texttt{n}_\texttt{m}}\texttt{]}\ \texttt{a}_1\texttt{->v}_1\ldots\ \texttt{a}_\texttt{k}\texttt{->v}_\texttt{k}\texttt{)}) = \mathbf{t}$
  if and only if
  $TVal_{\mathcal{I}}(\texttt{o\#f}) =$
  $TVal_{\mathcal{I}}(\texttt{o\#Top([}\texttt{t}_{1,1}\ \ldots\ \texttt{t}_{1,\texttt{n}_1}\texttt{])}) = \ldots = TVal_{\mathcal{I}}(\texttt{o\#Top([}\texttt{t}_{\texttt{m},1}\ \ldots\ \texttt{t}_{\texttt{m},\texttt{n}_\texttt{m}}\texttt{])}) =$
  $TVal_{\mathcal{I}}(\texttt{o\#Top(a}_1\texttt{->v}_1\texttt{)}) = \ldots = TVal_{\mathcal{I}}(\texttt{o\#Top(a}_\texttt{k}\texttt{->v}_\texttt{k}\texttt{)}) =$
  $\mathbf{t}$.
  Observe that on the right-hand side of the "if and only if" there are $1+\texttt{m}+\texttt{k}$ 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 $\texttt{m}+\texttt{k}$ ones using the root class `Top`).

For the root class, `Top`, and all $\texttt{o} \in \boldsymbol{D}$, $TVal_{\mathcal{I}}(\texttt{o\#Top}) = \mathbf{t}$.
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 $\texttt{o}, \texttt{f}, \texttt{g} \in \boldsymbol{D}$, if $TVal_{\mathcal{I}}(\texttt{o\#f}) = TVal_{\mathcal{I}}(\texttt{f\#\#g}) = \mathbf{t}$ then $TVal_{\mathcal{I}}(\texttt{o\#g}) = \mathbf{t}$.

Rule implication:

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

To exemplify the transformations, let us reconsider the `GeoUnit` KB of Section 4, focussing on the rule. Objectification acts as an identity transformation on this input rule, since the psoa atoms in both its condition and conclusion already have OIDs (two different variables, `?O` and `?In`). Slotribution and tupribution, however, transform the rule such that both its condition and conclusion become a conjunction linked by their OID variable. At that point, the psoa atoms have become minimal (three single-slot frames and one single-tuple shelf), so repeated slotribution and tupribution act as identity transformations on that output rule, which – also insensitive to objectification – is a fixpoint for these transformations. Adding a 'centralization' back arrow for the inverse of slotribution/tupribution, we obtain a bidirectional transformation scheme:

```
a2#betweenObjRel(dim->2 orient->northSouth canada usa mexico)

Forall ?Out1 ?In ?Out2 ?O (
  ?In#GeoUnit(neighborNorth->?Out1 neighborSouth->?Out2) :-
    ?O#betweenObjRel(orient->northSouth ?Out1 ?In ?Out2)
                           )
```

$$\underset{\text{centralization}}{\overset{\text{slotribution/tupribution}}{\rightleftarrows}}$$

```
And(a2#betweenObjRel
    a2#Top(dim->2)
    a2#Top(orient->northSouth)
    a2#Top(canada usa mexico))

Forall ?Out1 ?In ?Out2 ?O (
  And(?In#GeoUnit
      ?In#Top(neighborNorth->?Out1)
      ?In#Top(neighborSouth->?Out2)) :-
    And(?O#betweenObjRel
        ?O#Top(orient->northSouth)
        ?O#Top(?Out1 ?In ?Out2))
                           )
```

While slotribution and tupribution of psoa terms is built into the semantics, namely into the above Definition 5, case 3, these transformations can also be performed statically, as pre-processing steps.

8 PSOA RuleML Implementation

In order to support reasoning in PSOA RuleML, we have implemented PSOATransRun as an open-source framework system, generally referred to as PSOATransRun[*translation*,*runtime*], with a pair of subsystems plugged in as parameters [3, 19, 20].[28] The *translation* subsystem is a chain of translators mapping a KB and queries from PSOA RuleML to an intermediate language. The *runtime* subsystem executes KB queries in the intermediate language and extracts the results. Our focus has been on translators, reusing the targeted runtime systems as 'black boxes'. For the intermediate languages we have chosen the first-order subset, TPTP-FOF, of TPTP [21][29] and the Horn-logic subset of ISO Prolog [22]. Since these are also standard languages, their translation subsystems of PSOATransRun serve both for PSOA RuleML implementation and interoperation [20].

[28] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Implementation`.
[29] TPTP-FOF is also targeted by `http://wiki.ruleml.org/index.php/TPTP_RuleML`.

The chain targeting TPTP requires fewer translation steps since TPTP systems, being first-order-logic provers, directly accommodate the extra expressivity of PSOA (particularly, head existentials introduced by objectification). The chain targeting ISO Prolog requires more translation steps since ISO Prolog has the lower expressivity of Horn logic (particularly, requiring head existentials to be translated to Skolem function applications). Both translator chains start with parsing PSOA RuleML's (RIF-like) presentation syntax into Abstract Syntax Trees (ASTs). They then perform their transformation steps on AST representations of the PSOA sources, using slotribution/tupribution-introduced 'primitive' PSOA RuleML constructs, namely membership terms, slot terms, and tuple terms. Finally, they map the finished AST representations to TPTP or ISO Prolog presentation syntax as the intermediate languages – over distinguished predicates `memterm`, `sloterm`, and `tupterm` defined by TPTP or Prolog clauses – to be executed by the respective runtime systems. The translators are written in Java 1.6 and ANTLR v3[30].

The following subsections will survey our two implemented PSOATransRun instantiations, the parameterized PSOATransRun[PSOA2TPTP,VampirePrime] and PSOATransRun[PSOA2Prolog,XSBProlog].

We have also implemented the PSOA RuleML API [23][31], which uses JAXB to parse PSOA RuleML/XML syntax into abstract syntax objects, and translates these into PSOA RuleML's RIF-like presentation syntax.

8.1 With PSOA2TPTP to VampirePrime

The PSOATransRun[PSOA2TPTP,VampirePrime] instantiation [19][32], realized by Gen Zou and Reuben Peter-Paul with guidance from the author and Alexandre Riazanov, combines the PSOA2TPTP translator and the VampirePrime runtime system. The runtime system consists of the C++-implemented VampirePrime, accessed through Java.

PSOA2TPTP performs objectification (cf. Section 6) as well as slotribution and tupribution (cf. Section 7). PSOA2TPTP then maps the transformation result to TPTP.

VampirePrime[33] is an open-source first-order reasoner. KBs and queries in the intermediate TPTP language can also be run on other TPTP systems that allow extracting answers (variable bindings) from successful results.

This PSOA2TPTP instantiation is available online[34] for interactive exploration, with documentation in the above-linked RuleML Wiki page. The sample PSOA KB textbox, pre-filled by the system, shows the easy transcription of our (RIF-like) presentation-syntax examples into executable PSOA RuleML: According to the EBNF referenced in Section 6, the PSOA syntax is completed by a `Document`/`Group` wrapper for KBs and the "`_`" prefix for local constants.

[30] `http://www.antlr.org`.
[31] `https://github.com/sadnanalmanir/PSOARuleML-API`.
[32] `http://psoa2tptp.googlecode.com`.
[33] `http://riazanov.webs.com/software.htm`.
[34] `http://psoa-ruleml.rhcloud.com`.

8.2 With PSOA2Prolog to XSB Prolog

The PSOATransRun[PSOA2Prolog,XSBProlog] instantiation [20][35], realized by Gen Zou with guidance from the author, combines the PSOA2Prolog translator and the XSB Prolog runtime system. The runtime system consists of the C++-implemented XSB Prolog, accessed via a Java API[36].

PSOA2Prolog augments the translation chain of PSOA2TPTP in Section 8.1 and performs a different target mapping. PSOA2Prolog is composed of a source-to-source normalizer followed by a mapper to a pure Prolog (Horn logic) subset of the ISO Prolog subset of XSB Prolog. The normalizer is composed of five transformation layers, namely objectification, Skolemization, slotribution/tupribution, flattening, as well as rule splitting. Each layer is a self-contained component that can be reused for processing PSOA KBs in other applications. The mapper performs a recursive transformation from the normalization result to Prolog clauses.

XSB Prolog[37] is a fast Prolog engine, which we use for processing a pure ISO Prolog subset. While this ISO Prolog subset can also be run on other Prolog engines, XSB Prolog is targeted because it enables tabling, supporting both termination and efficiency. XSB Prolog executes queries over KBs in the intermediate Prolog language, and the PSOATransRun framework system performs answer extraction.

9 Conclusions

The integrated object-relational data and rules of PSOA RuleML enable a novel approach to semantic modeling and analysis based on positional-slotted, object-applicative terms. PSOA RuleML's data model visualized in Grailog provides the logical foundation and visual intuition via the psoa-table systematics of six uses of psoa atoms in queries and facts as well as conditions and conclusions of rules. PSOA RuleML allows direct (look-in and inferential) querying over heterogeneous data sets. Moreover, PSOA RuleML serves as an intermediate language for bidirectional query transformation, e.g. between SQL and SPARQL. The syntax and semantics capture the essence of PSOA RuleML's object-relational integration. Two implemented open-source PSOATransRun instantiations, one also usable online, allow rapid PSOA RuleML prototyping. Besides the PSOA RuleML test cases on the RuleML Wiki, there are PSOA RuleML use cases such as MusicAlbumKB and GeospatialRules. The latter started with a Datalog$^+$ rulebase for the Region Connection Calculus (RCC) [10] and is being expanded into a Hornlog$^+$-like rulebase over psoa-generalized RCC atoms. Future PSOA RuleML applications are envisioned for data querying and interchange in the domains of biomedicine, finance, and social media.

[35] `http://psoa.ruleml.org/transrun/0.7/local/`.
[36] `http://interprolog.com`
[37] `http://xsb.sourceforge.net`

10 Acknowledgements

Many thanks go to Gen Zou for helpful discussions on multiple drafts of this paper and for spearheading the PSOATransRun implementation. I want to thank Tara Athan, Sadnan Al Manir, Alexandre Riazanov, and Robert Kirby for reviewing earlier partial versions. I extend my thanks to Michael Genesereth, Sudhir Agarwal, Abhijeet Mohapatra, and Eric Kao for comments on a PSOA RuleML presentation in the Computational Logic Seminar, and to Michael Genesereth and the entire Stanford Logic Group for hosting my research stay. My thankfulness goes to Richard Waldinger for comments at various occasions, and for hosting my recent SRI visits. The 11th Reasoning Web Summer School (RW 2015) reviewer and organizers are thanked for early feedback and for running this event. NSERC is thanked for its support through Discovery Grants.

References

1. Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-oriented and Frame-based Languages. Journal of the ACM **42**(4) (July 1995) 741–843
2. Boley, H., Kifer, M.: RIF Basic Logic Dialect (Second Edition) (February 2013) W3C Recommendation, `http://www.w3.org/TR/rif-bld`.
3. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: Proc. 5th International Symposium on Rules: Research Based and Industry Focused (RuleML-2011 Europe), Barcelona, Spain. Lecture Notes in Computer Science, Springer (July 2011) 194–211
4. Boley, H.: Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. In Morgenstern, L., Stefaneas, P.S., Lévy, F., Wyner, A., Paschke, A., eds.: Proc. 7th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2013), Seattle, Washington, USA. Volume 8035 of Lecture Notes in Computer Science., Springer (July 2013) 52–67
5. Athan, T., Boley, H.: The MYNG 1.01 Suite for Deliberation RuleML 1.01: Taming the Language Lattice. In Patkos, T., Wyner, A., Giurca, A., eds.: Proceedings of the RuleML 2014 Challenge, at the 8th International Web Rule Symposium. Volume 1211., CEUR (August 2014)
6. Hanus (ed.), M.: Curry: An Integrated Functional Logic Language (Vers. 0.8.3). `http://www-ps.informatik.uni-kiel.de/currywiki/_media/documentation/report.pdf` (February 2014)
7. Agarwal, S., Mohapatra, A., Genesereth, M., Boley, H.: Rule-Based Exploration of Structured Data in the Browser. In: Proc. 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany. Lecture Notes in Computer Science, Springer (August 2015)
8. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A Logical Framework For the World Wide Web. Theory and Practice of Logic Programming (TPLP) **8**(3) (May 2008)
9. Riazanov, A., Rose, G.W., Klein, A., Forster, A.J., Baker, C.J.O., Shaban-Nejad, A., Buckeridge, D.L.: Towards Clinical Intelligence with SADI Semantic Web Services: a Case Study with Hospital-Acquired Infections Data. In: Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences. SWAT4LS '11, New York, NY, USA, ACM (2012) 106–113

10. Zou, G.: GeospatialRules: A Datalog+ RuleML Rulebase for Geospatial Reasoning. In Patkos, T., Wyner, A., Giurca, A., eds.: Challenge+DC@RuleML. Volume 1211 of CEUR Workshop Proceedings., CEUR-WS.org (2014)
11. Crockford, D.: Introducing JSON (May 2009) Format home page, `http://json.org`.
12. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. Journal of Emerging Technologies in Web Intelligence **4**(2) (November 2010) 343–353
13. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
14. Ball, M., Boley, H., Hirtle, D., Mei, J., Spencer, B.: The OO jDREW Reference Implementation of RuleML. In Adi, A., Stoutenburg, S., Tabet, S., eds.: Rules and Rule Markup Languages for the Semantic Web, First International Conference (RuleML 2005), Galway, Ireland, November 10-12, 2005, Proceedings. Volume 3791 of Lecture Notes in Computer Science., Springer (2005) 218–223
15. Knublauch, H.: SPINMap: SPARQL-based Ontology Mapping with a Graphical Notation (April 2011) Composing the Semantic Web: A tool developer's blog on ontology development for the Semantic Web and beyond, `http://composing-the-semantic-web.blogspot.ca/2011/04/spinmap-sparql-based-ontology-mapping.html`.
16. Brunnbauer, M.: RDF2RDB – convert RDF data to relational databases. `http://www.netestate.de/en/software-development/rdf2rdb/` (2012)
17. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. World Wide Web Consortium, Recommendation REC-r2rml-20120927 (September 2012)
18. Yang, G., Kifer, M.: Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. In Spaccapietra, S., March, S.T., Aberer, K., eds.: J. Data Semantics I. Volume 2800 of Lecture Notes in Computer Science., Springer (2003) 69–97
19. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners. In Bikakis, A., Giurca, A., eds.: Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France. Volume 7438 of Lecture Notes in Computer Science., Springer (August 2012) 264–279
20. Zou, G., Boley, H.: PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog. In: Proc. 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany. Lecture Notes in Computer Science, Springer (August 2015)
21. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. J. Autom. Reasoning **43**(4) (2009) 337–362
22. ISO/IEC 13211-1: Prolog – part 1: General core (1995)
23. Al Manir, M.S., Riazanov, A., Boley, H., Baker, C.J.O.: PSOA RuleML API: A Tool for Processing Abstract and Concrete Syntaxes. In Bikakis, A., Giurca, A., eds.: Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France. Volume 7438 of Lecture Notes in Computer Science., Springer (August 2012) 280–288