# PSOA RuleML: Integrated Object-Relational Data and Rules

(Long version:
http://www.cs.unb.ca/~boley/talks/PSOAObjRelDataRules-talk.pdf
– Search: "PSOA RuleML"; click: Contents "References"; get: 7. . . . Slides)

Harold Boley

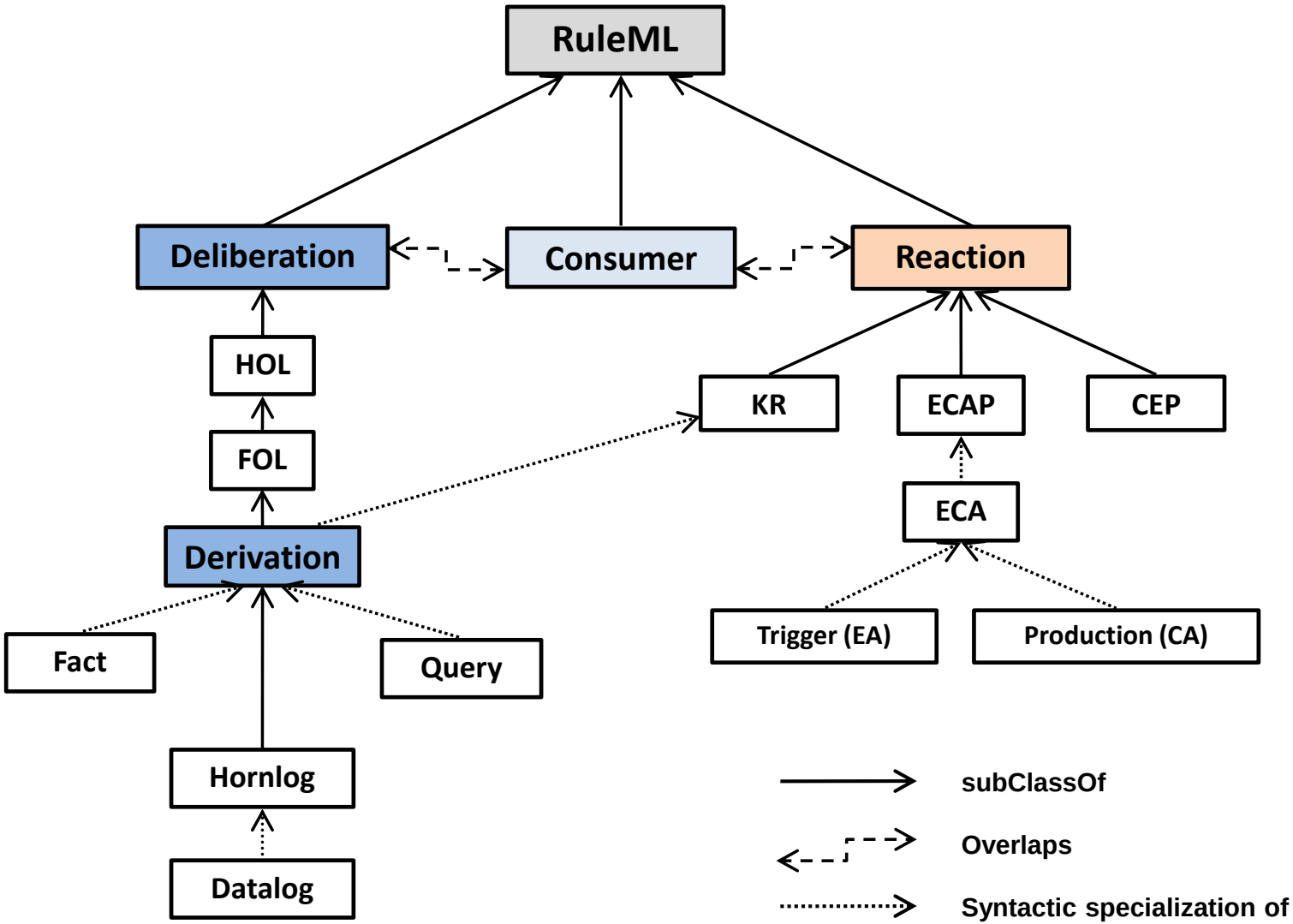
Faculty of Computer Science, University of New Brunswick, Canada

*Computational Logic Seminar – MUGS, Stanford Logic Group, 27 May 2015*
*AIC Seminar Series, SRI, 9 June 2015*
*11th Reasoning Web Summer School (RW2015), FU Berlin, 31 July – 4 Aug. 2015*

# Foundation: RuleML Spec

- RuleML is a knowledge representation architecture
  - specified for the interchange of the major kinds of Web rules
  - in an XML format that is uniform across rule logics and platforms
- RuleML 1.02 encompasses
  - updated versions of the existing Deliberation and Reaction families
  - and their initial integration via the new Consumer RuleML family

# Foundation: RuleML 1.02 Families



# Foundation: PSOA RuleML

- PSOA RuleML is a novel object-relational integration that started with Datalog and Hornlog
- PSOA principles extend to all of Deliberation RuleML as well as to Consumer and Reaction RuleML

# Introduction: Data Divide

- Data obtained status of “raw and processed material for all endeavors”
- In analogy to distinctions for materials, both external and internal distinctions can be made in realm of (complex) data
- External distinctions: “proprietary vs. *open*” and, orthogonally, “siloed vs. *linked*”
- Internal distinctions: data paradigms of ***relations***, in SQL-queried Deep Web, vs. ***graphs***, in SPARQL-queried Semantic Web
- Divide also led to separate relational and graph rule paradigms for processing data (e.g., for inferencing/reasoning with them)

# Introduction: From Amalgamation to Integration

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

- Heterogeneous combination (amalgamation) allows, as in F-logic and RIF, atomic formulas in separated relational and graph language paradigms for data as well as rules, also mixed within same rule
- Homogeneous combination (integration) of Positional-Slotted Object-Applicative (PSOA) RuleML blends atomic relational and graph formulas themselves into uniform kind of atom, for language-internal ("intra-language") transformation of data as well as rules

# Introduction: PSOA RuleML Data and Rules

Basically, data [= ground (variable-less) **facts**] include:

- (Table-row-like) **relational** atoms without Object IDentifier (OID) and with positional arguments
- (Graph-node-like) **graph** atoms with – globally unique – OID, e.g. a URI/IRI on the Web, and slotted arguments (for node's outgoing labeled edges)

- What we call 'slots' is often variously called 'attributes', 'properties', or 'roles'
- Each slot can have one or more values

Beyond data, knowledge consists of implications [= logic **rules**] (and special case of subclassing):

- Implications can use non-ground (variable-containing) versions of **all of above** atoms anywhere in **conditions (bodies)** and **conclusions (heads)**

# Introduction: Two Orthogonal Dimensions

Generally, relational vs. graph distinction based on two orthogonal dimensions, creating system of four quadrants

Object-relational integration achieved by permitting atom to be

- ***predicate-centered*** (without OID) or ***object-centered*** (with OID) – every OID being typed by predicate as its class –

and, orthogonally,

- predicate's arguments to be ***positional*** (sequence), ***slotted*** (bag of pairs), or ***both*** (positional-plus-slotted combination)

# Introduction: Psoa Table

Atoms resulting from orthogonal system are

**p**ositional-**s**lotted, **o**bject-**a**pplicative (**psoa**)

Can be used in six ways, as shown in *psoa table* (quadrants 1. to 4. expanded by combined options 5., 6.):

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

Of six options, positional data widely used under names like ‘tuples’, ‘vectors’, ‘lists’, and (1D) ‘arrays’ (mostly 1.)

Likewise, slotted data include ‘objects’, ‘records’, ‘maps’, and ‘property lists’ (usually 4.)

# Introduction: Six Family Atom Variations

Illustrated with variations of family-example atoms:

1. **Predicate**-centered, **positional** atoms (relationships), without OID and with - ordered - argument sequence, e.g. *Husb* $\times$ *Wife* relationship `family(Joe Sue)`
2. Object-centered, positional atoms (shelves), with OID and arg. sequence, e.g. `inst1#family(Joe Sue)` with `family`-typed OID `inst1`, where "#" is read "$\in$"
3. Predicate-centered, slotted atoms (pairships), without OID and with - unordered - bag (multi-set) of slots (pairs of slot name and slot filler), e.g. `family(husb->Joe wife->Sue)` or `family(wife->Sue husb->Joe)`
4. **Object**-centered, **slotted** atoms (frames), with OID and with multi-set of slots, e.g. `inst1#family(husb->Joe wife->Sue)` or commuted (as in 3.)

# Introduction: Six Family Atom Variations (Cont'd)

5. Predicate-centered, positional+slotted atoms (relpairships), without OID and with both sequence of arguments and multi-set of slots, e.g. 3-slot, 2-argument atom `family(child->Pete dog->Fido dog->Toby Joe Sue)`
6. Object-centered, positional+slotted atoms (shelframes), with OID and with argument sequence and slot multi-set, e.g. `inst1`-identified atom (otherwise as in 5.) `inst1#family(child->Pete dog->Fido dog->Toby Joe Sue)`

# Introduction: Family Rule

Backward implication "*frame* if *conjunction*" illustrated by

- conjunction of two predicate-centered, positional atoms – relational join –

deriving, 'forward speaking',

- object-centered, slotted atom (4.) -cf. F-logic frames- with application of fresh function name, **`famid`**, to **`?Hu`** and **`?Wi`**, denoting OID dependent on them but not on **`?Ch`** – *oid*#*class* typing *oid* with *class* –

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

# Introduction: PSOA RuleML in Context

PSOA RuleML is (head-existential-)extended Horn-logic language (with equality) that systematizes variety of RIF-BLD terms 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 for syntax and adapting RIF-FLD-specified NAF dialects for semantics

Conversely, PSOA RuleML is being developed as module that is **pluggable** into larger (RuleML) logic languages, thus making them likewise object-relational

# Introduction: PSOA RuleML Slides and Paper

Slides and RW 2015 paper give 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, tutorial-style exposition:

- **visualizes** all psoa terms in Grailog, where (n-ary) directed hyperarcs – of directed hypergraphs – are used for positional terms, and (binary) directed arcs – of directed 'graphs' in narrow sense – are used for slotted terms;
- uses ('functional') ***terms*** `p(...)` with predicate symbol `p`, taking them as atomic ***formulas*** as in Relfun, HiLog, and RIF, which – along with equality – is basis for universal functional-logic programming as in Curry;
- is about **instance** frames (frame atoms) and other psoa atoms employed as **queries** and **facts**, and 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;

# Introduction: PSOA RuleML Slides and Paper (Cont'd)

- **uses ordinary constants as Object IDentifiers, which logically connect (distributed) frames and other psoa atoms describing same OID, e.g. after disassembling (slotributing) 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 'backbone' of (typed) frames etc., where **lack** of ***oid*** is compensated by system (e.g. as Skolem constant or existential variable) and **absence** of ***class*** typing is expressed by `Top` class, specifying root of 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 latter operations can again use OIDs to refer to frames (cf. N3);
- focuses on **SQL-SPARQL interoperation use case** about (sub)addresses, while other use cases are about **clinical intelligence**, **music albums**, and **geospatial rules**

# Introduction: Outline (of Slides and RW 2015 Paper)

1. So far: Introduced object-relational combinations with focus on PSOA RuleML integration. Next:
2. Develop PSOA data model with systematically varied example in presentation syntaxes derived from RuleML/POSL and RIF-BLD, and as Grailog visualizations
3. Assert such ground atoms as ground facts and query them by ground or non-ground atoms, followed by non-ground OID-existential PSOA fact and its querying
4. With similar facts, explore PSOA rules and their querying
5. Bidirectional SQL-PSOA-SPARQL transformation use case
6. Define objectification as well as the presentation and serialization syntaxes of PSOA RuleML
7. Formalize model-theoretic semantics, blending (OID-over-) slot distribution with integrated psoa terms
8. Survey PSOATransRun implementation, translating PSOA RuleML knowledge bases and queries to TPTP or Prolog

# Data Model: Grailog Visualization

PSOA RuleML data model systematics intuitively explained through subset of Grailog extended with ***branch lines***, for multiple, 'split-out' (hyper)arcs, and using novel ***Scratch Grailog*** visualization, which stresses connecting lines (rather than surrounding boxes)

In logical languages, data conceived as ground (variable-free) facts often given in (symbolic) presentation syntax. Will use Grailog "skewer figures" as corresponding (graphical) **visualization syntax for PSOA facts integrating relations and objects**

(Scratch) Grailog figures will visualize connectivity within set of ***labelnodes***, where color coding will show correspondence to symbolic facts

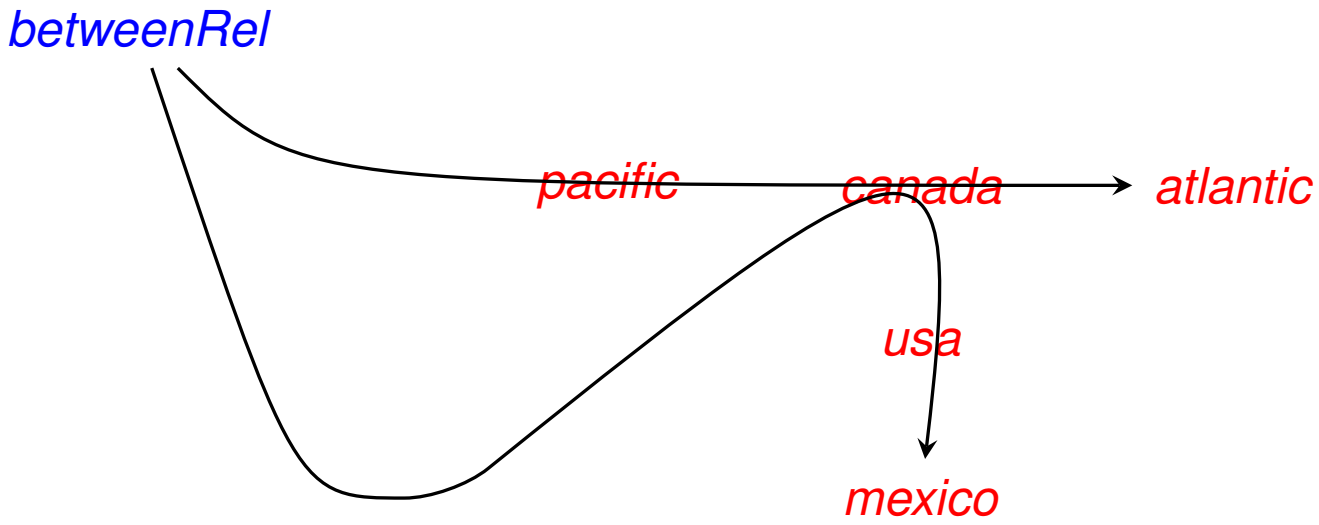
Throughout, will vary running 'betweenness' example for illustration

# Data Model: Relationships – Predicate-Centered, Positional Atoms

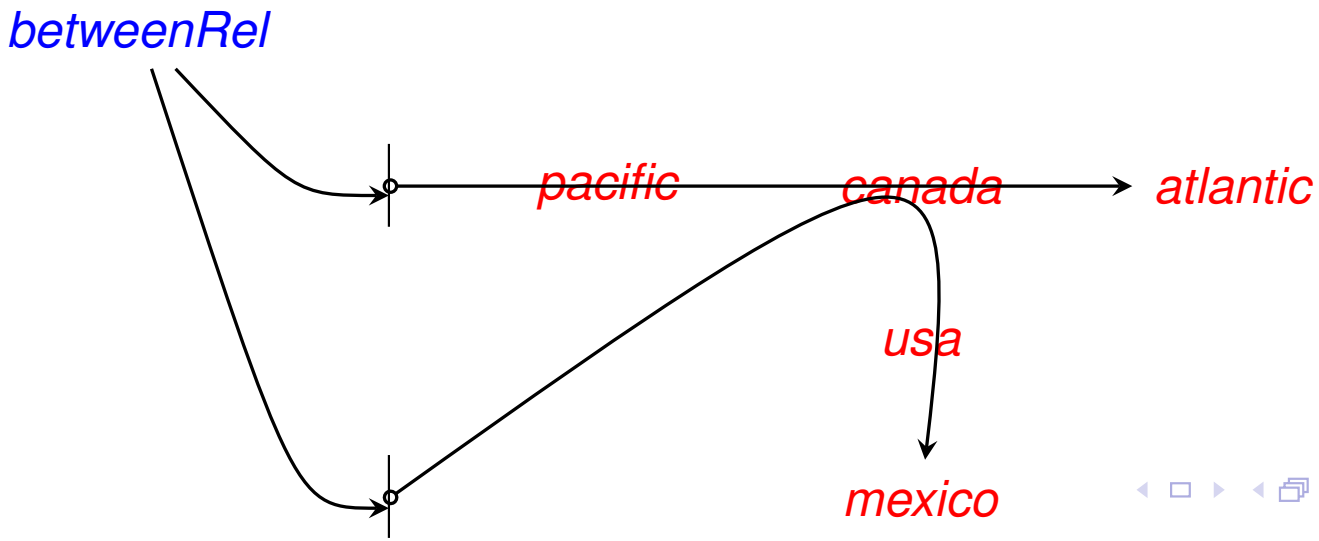
- ***Relationships*** represent n-ary positional information (n$\geq$0)
- In Grailog, relationship becomes ***directed hyperarc***, depicted as: arrow shaft starting at labelnode for relation name or at branch line, cutting through labelnodes for n-1 initial arguments in order they occur, ending with arrow head at labelnode for $n^{th}$ argument
- Sample Grailog figures visualize 3-ary relational betweenness with hyperarcs for two relationships applying relation name betweenRel to three argument individuals

# Data Model: Relationships (Cont'd)

**Grailog-style visualization syntax (without branch lines):**




**Grailog-style visualization syntax (with branch lines):**



# Data Model: Relationships (Cont'd)

- Corresponding relational PSOA facts in POSL- and RIF-like presentation syntax employ **traditional parenthesized relation applications**
- Here, POSL-vs.-RIF difference only in use of separator (comma vs. white-space) and terminator (period vs. newline) symbols

# Data Model: Relationships (Cont'd)

**POSL-like presentation syntax:**

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

**RIF-like presentation syntax:**

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

# Data Model: Relationships (Cont'd)

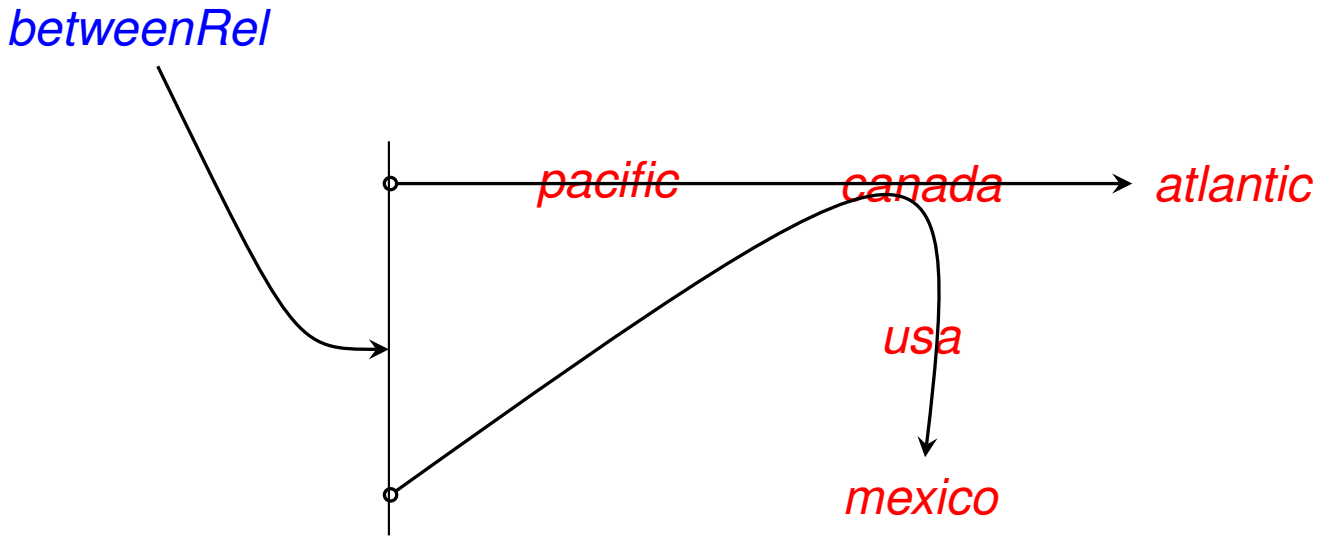
Notice that relation name *betweenRel* and argument *canada* are shared by hyperarcs but become copied in facts

Alternative sample Grailog figure extends above vertical branch lines such that they meet, obtaining single branch line, and uses single unary *betweenRel* hyperarc pointing to it

As for relational tables (e.g., in SQL), multiple copies of relation name can be avoided in PSOA RuleML facts. Corresponding relational psoa term replaces two separate facts for same relation with single multi-tuple (specifically, double-tuple) fact

# Data Model: Relationships (Cont'd)

**Grailog-style visualization syntax (with branch line):**




**POSL-like presentation syntax:**

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

**RIF-like presentation syntax:**

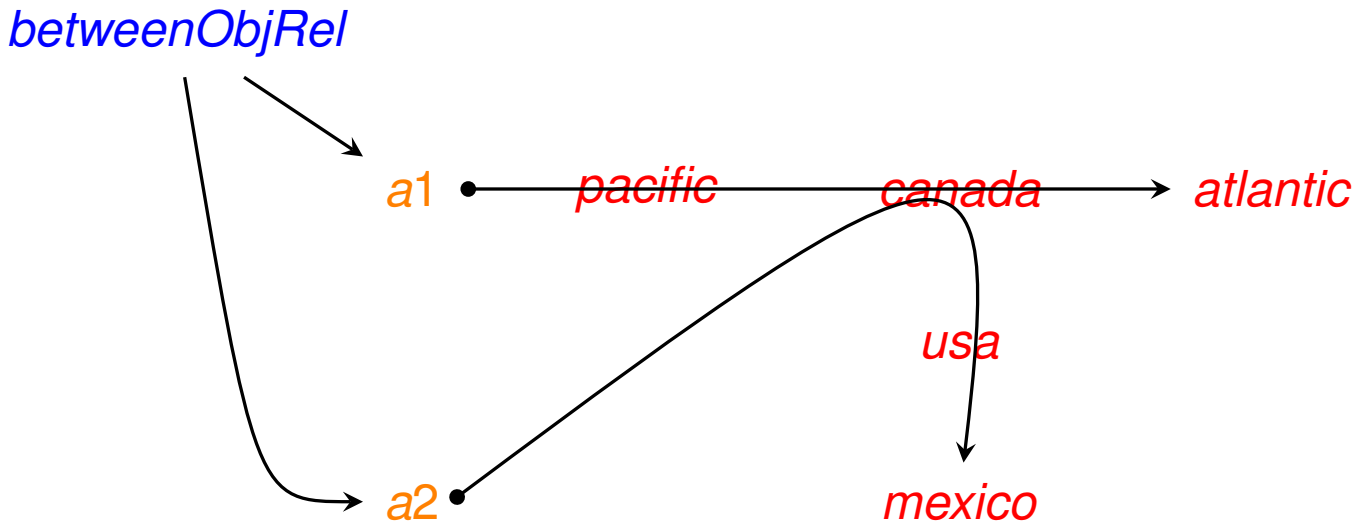
betweenRel([pacific canada atlantic] [canada usa mexico])

# Data Model: Shelves – Object-Centered, Positional Atoms

- ***Shelves*** describe an OID with n positional arguments ($n \geq 0$)
- A shelf thus endows n-tuple with OID, typed by relation/class, keeping positional representation of n-ary relationships
- Sample Grailog figure visualizes two OIDs, *a1* and *a2*, typed by relation/class name *betweenObjRel*, and two 3-tuples with three individuals as arguments

# Data Model: Shelves (Cont'd)

**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)

# Data Model: Shelves (Cont'd)

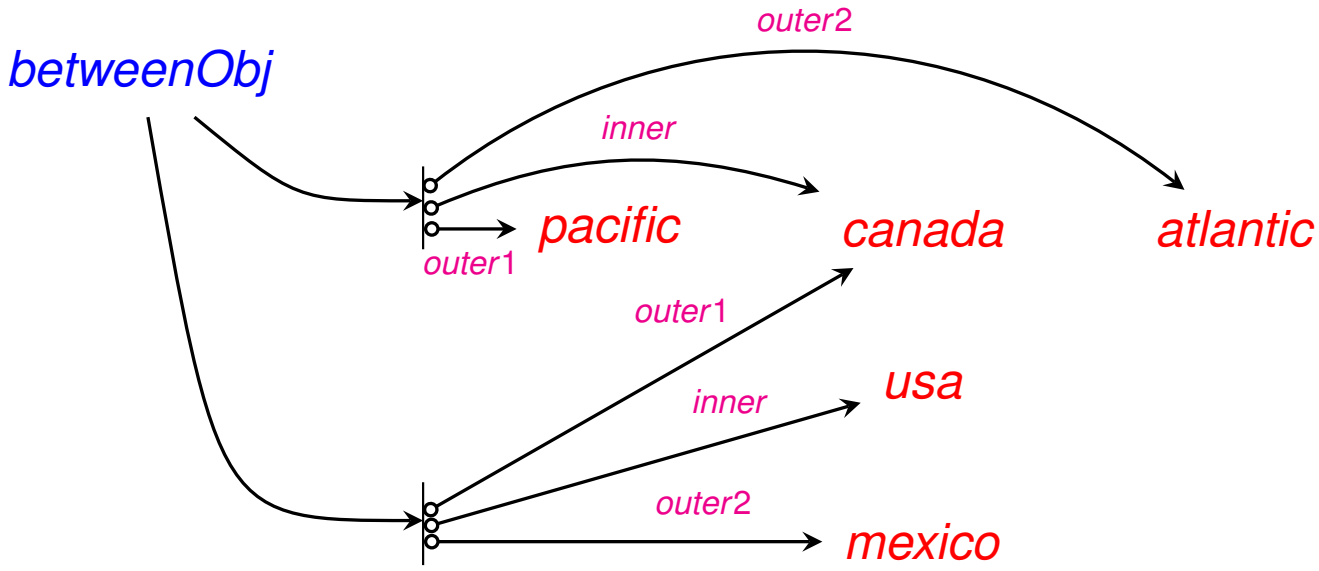
- Corresponding psoa term facts, shown earlier, employ **syntaxes adding OIDs to the parenthesized relation-application syntaxes for the three positional arguments from relationship**:
  - POSL-like version specifies OID at beginning of argument sequence, where hat/circumflex ("^") – think of it as 'property/slot insertion' character – is used as infix separating OID from slots
  - RIF-like version specifies OID along with its typing relation/class, where hash ("#") – think of it as 'set/class membership' character ("∈") – is used as infix separating OID from relation/class

# Data Model: Pairships – Predicate-Centered, Slotted Atoms

- ***Pairships*** apply a relation/class to n ***slots***: non-positional attribute-value pairs ($n \geq 0$)
- In Grailog, **pairship is depicted as relation/class node pointing, with unary hyperarc, to branch line having n outgoing circle-shaft slot arrows, using: label for attribute, target node for value**
- Sample Grailog figure visualizes 3-slot betweenness of two pairships that apply relation name *betweenObj* to branch line for three slots, with labels *outer*1, *inner*, and *outer2*, targeting three individuals as values

# Data Model: Pairships (Cont'd)

**Grailog-style visualization syntax:**




**POSL-like presentation syntax:**

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

**RIF-like presentation syntax:**

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

# Data Model: Pairships (Cont'd)

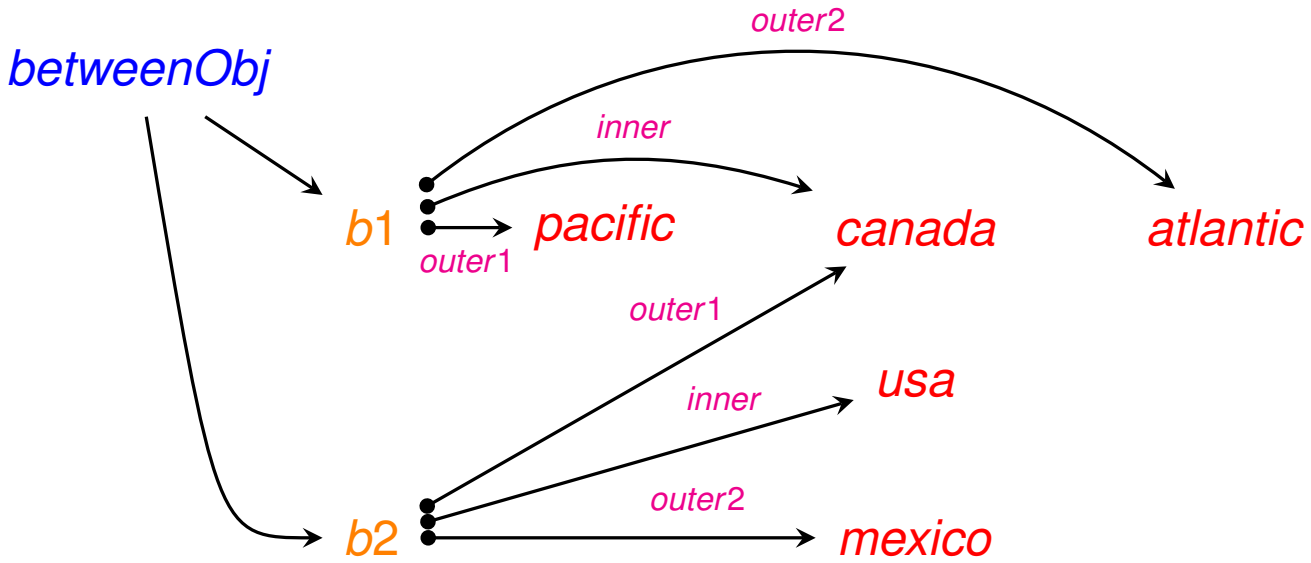
- Corresponding pairship facts, shown earlier, employ syntaxes modifying relationship syntax
- In both the POSL- and RIF-like versions, **'dash-greater' right-arrow ("$->$") sign – think of it as 'has value/filler' character ("$\rightarrow$") – is used as infix separating slot attribute (name) and value (filler)**
- As in Grailog, order in which slots occur in atom is immaterial

# Data Model: Frames – Object-Centered, Slotted Atoms

- ***Frames*** describe OID using n non-positional attribute-value pairs or ***slots*** (n≥0), with kind of object becoming OID-typing class name
- In Grailog, frame is depicted as **typing relation/class node pointing, with unary hyperarc, to central OID node having n outgoing bullet-shaft slot arrows, using: label for attribute, target node for value**
- Sample Grailog figure visualizes object-centered 3-slot betweenness with central nodes, *b*1 and *b*2, for OIDs of two frames typed by relation name *betweenObj*, and three slots, with labels *outer*1, *inner*, and *outer*2, targeting three individuals as values

# Data Model: Frames (Cont'd)

**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)

# Fact Querying: From Look-Up to Look-In

**‘Look-in’ querying generalizes ‘look-up’ querying by ‘looking’ for psoa query terms ‘in’ asserted psoa fact terms**

Define ‘equal to’ and ‘part of’ for psoa fact and query terms in terms of their graph counterparts: For graph equality and parthood, **attachment order of hyperarcs and slot arrows is immaterial**

Corresponds to slotribution and tupribution in model-theoretic semantics and in transformational semantics

Proof-theoretic check that query term is ‘part of’ fact term becomes model-theoretic/transformational reformulation of query term into conjunction of a membership term and single-slot plus single-tuple terms against likewise reformulated fact term

# Fact Querying: From Look-Up to Look-In (Cont'd)

Two elementary binary relations between arbitrary psoa terms:

- 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 re-ordering 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 $t_2$.

# Fact Querying: From Look-Up to Look-In (Cont'd)

A set of PSOA ground atoms, e.g. as visualized in Grailog, 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

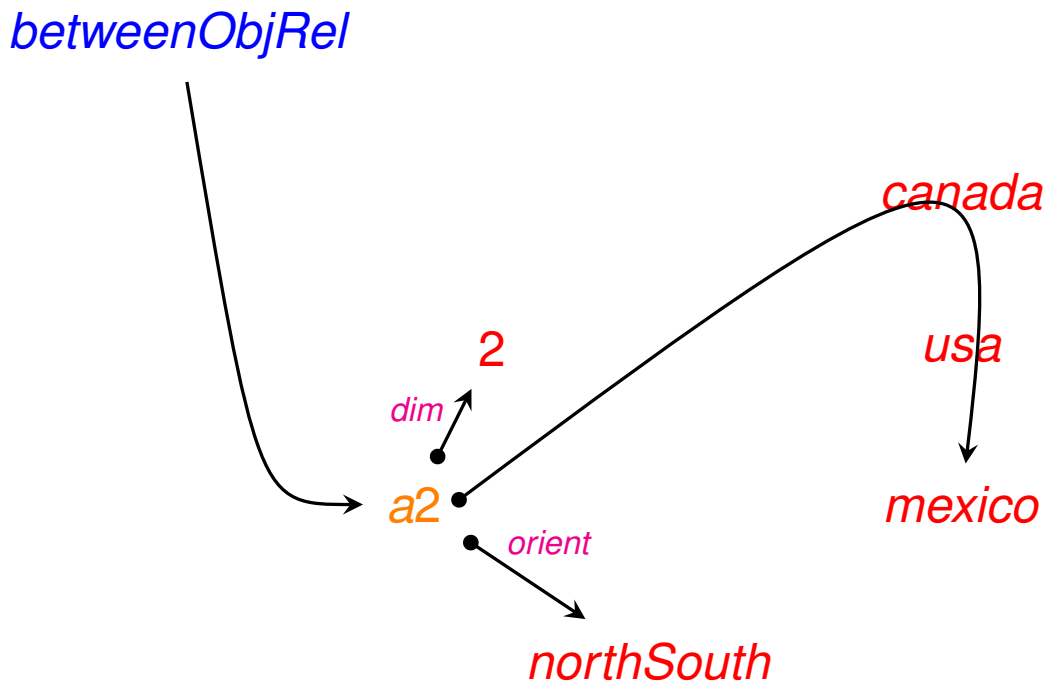
This exemplifies a basic notion of proof-theoretic semantics with positive (success) and negative (fail) entailment tests, KB $\vdash$ q and KB $\nvdash$ q, respectively. Look-in ground and non-ground querying defined below, where the former is a special case of the latter.

**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

# Fact Querying: Shelframe KB

Shelframe atoms are positional+slotted, including both positional and slotted atoms; hence we focus on them. For example, consider ground shelframe atom below, asserted in single-fact sample KB

**Grailog-style visualization syntax:**




**RIF-like presentation syntax:**

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

# Fact Querying: Shelframe KB (Cont'd)

This ground atom can be retrieved by issuing identical ground atom as ground query (special case of 'equal to'), yielding success message

Cannot be retrieved by issuing ground query that is not part of ground fact, e.g. one that expects `alaska` in place of `canada`, yielding failure message

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 positional arguments or adding/deleting some of them within a tuple, or when inserting slots and/or tuples, or when using different OID

# Fact Querying: Ground/Ground Matching

```
a2#betweenObjRel(dim->2 orient->northSouth[canada usa mexico])
success   % Positional arguments use square brackets on 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)
```

# Fact Querying: Ground/Ground Matching (Cont'd)

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

# Fact Querying: Ground/Ground Matching (Cont'd)

```
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
```

# Fact Querying: Non-Ground-Lifted Look-In

Ground atom can also be retrieved by issuing non-ground queries, using non-ground/ground matching, by first 'grounding' query by consistently substituting all query variables with corresponding ground subterms (recording variable bindings for answer) and then doing retrieval with ground query as for earlier look-in ground querying

**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

# Fact Querying: Non-Ground/Ground Matching

```
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 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
```

# Fact Querying: Non-Ground/Ground Matching (Cont'd)

```
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

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
```

# Fact Querying: Non-Ground-Lifted KB

Consider single-fact sample KB containing non-ground-lifted modification of earlier atom, asserted as *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."

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

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

# Fact Querying: 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

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

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

# Fact Querying: 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 argument substitution possible

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

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

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

# Inferential Querying: Look-In Resolution

PSOA RuleML fact querying can be done interactively by user, as presented earlier, but fact – and rule – querying can also take place in conditions of rules, as will be discussed now

Using rules, user's interactive querying becomes inferential. **Rule querying is realized by resolution, which employs unification for consistent instantiation – ultimately, grounding – of (possibly non-ground) query and (possibly non-ground) rule conclusion**

**In PSOA RuleML, after grounding, query must be checked to be 'part of' rule conclusion** in the sense defined earlier. For PSOA sublanguage using only single-tuple psoa terms, this is similar to POSL's unification involving queries that have anonymous rest slots ("?"), as implemented in OO jDREW

# Inferential Querying: Safe Commutativity

**Rule** definition and querying **to selectively specify** derivable properties such as **commutativity (symmetry) of** certain arguments, e.g. **two outer arguments of** **`betweenObjRel`**.

**Derived symmetric tuples are identified by OID `symm(?O)`, depending, via function `symm`, on original tuples, identified by OID `?O`**

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

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

# Inferential Querying: Safe Commutativity (Cont'd)

In order to prevent recursive rules, e.g. for commutativity, from repeatedly undoing/redoing their derivation results, **flag-like slots such as `orientation` slot** come in handy

**On backward reasoning for query answering with our above rule, slot filler is 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 `northSouth` rules and facts will be applicable, e.g. with our fact terminating rule derivation after one step

This (non-ground) commutativity rule and our ground fact

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

can thus be used for safe derivation of ground and non-ground queries

# Inferential Querying: Safe Commutativity (Cont'd)

```
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
```

# Inferential Querying: Safe Commutativity (Cont'd)

```
?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 argument substitution possible

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

# Inferential Querying: Frames on Demand

Another rule can be used to derive **new frames**, specifically `GeoUnit` frames

"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 orientation slot north-to-south – of `?Out1`, `?In`, and `?Out2` holds."

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

# Inferential Querying: Frames on Demand (Cont'd)

This (non-ground) frame-deriving rule and our recurring ground fact

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

play together for derivation of frames on demand through ground and non-ground queries –

inner positional fact argument (`usa`) reused as rule-conclusion OID of new frame:

# Inferential Querying: Frames on Demand (Cont'd)

```
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  % Which GeoUnit ?I has Canada as its northern neighbor?
```

# Inferential Querying: Existential Psoa OIDs

*OID-(head-)existential rule*, which can be used to derive **new psoa terms**, specifically `compassRose` psoa terms

"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 orientation slot north-to-south – of `?Out1`, `?In`, and `?Out2` holds, and an `?O2`-identified `betweenObjRel` relationship – with orientation slot west-to-east – of `?Out3`, `?In`, and `?Out4` holds."

```
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))
                                 )
```

# Inferential Querying: Existential Psoa OIDs (Cont'd)

Conclusion uses four slots representing the cardinal compass directions, `?Out1` through `?Out4`, and single 'positional' argument, `?In`, representing rose center.
Condition uses explicit `And` conjunction for `?In`-intersecting `northSouth`- and `westEast`-oriented `betweenObjRel` relationship queries.
Entire conclusion (head) wrapped into existential (`Exists`) scope for OID variable `?O`

Rule and two facts

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

can be used for derivation of ground and non-ground queries:

# Inferential Querying: Existential Psoa OIDs (Cont'd)

```
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
```

# SQL-PSOA-SPARQL: Interop Use Case

- Suppose you are working on project using **SQL queries over relational data** and then proceeding to **SPARQL queries over graph data** to be used as metadata repository
- Or, vice versa, on project complementing SPARQL with SQL for querying evolving mass-data store
- Or, on project using SQL and SPARQL from the beginning
- In all of these projects, **object-relational interoperability issues** may arise
- Hence use case on **bidirectional SQL-PSOA-SPARQL transformation (schema/ontology mapping)** for interoperability
- **Core transformation between relational and object-centered paradigms is expressed in language-internal manner within PSOA RuleML itself**

# SQL-PSOA-SPARQL: Interop Use Case (Cont'd)

- Use case represents **addresses as (flat) relational facts and as – subaddress-containing – (nested) object-centered facts**, as shown for Seminaris address below
  - Earlier (flat and nested) positional versions have been used to explain XML-to-XML transformation
  - Later, similar use case was employed to demonstrate SPINMap for RDF-to-RDF transformation
- **OID-conclusion direction of implication from relational to object-centered (frame) paradigm** is given as first rule below
- **OID-condition direction from object-centered (frame) to relational paradigm** is given as second rule

# SQL-PSOA-SPARQL: Facts and Rules

```
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))
                                  )
```

# SQL-PSOA-SPARQL: Relational to Object-Centered Queries

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

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

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

?S = "Takustr. 39"
```

# SQL-PSOA-SPARQL: Object-Centered to Relational Queries

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

```
?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"
```

# SQL-PSOA-SPARQL: SQL to SPARQL

- If object-centered PSOA RuleML fact is replaced by corresponding RDF triple facts, **OID-condition PSOA RuleML rule can also be used for language-internal transformation of SQL-like queries to SPARQL-like queries** as shown shortly
  - '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
- **Paradigm-crossing translation step is done by OID-condition rule completely within PSOA RuleML**, starting at SQL queries "lifted" to PSOA and ending at SPARQL queries "dropped" from PSOA

# SQL-PSOA-SPARQL: SQL to SPARQL (Cont'd)

```
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
```

# SQL-PSOA-SPARQL: SQL to SPARQL (Cont'd)

```
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
```

# SQL-PSOA-SPARQL: SQL to SPARQL (Cont'd)

```
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"
```

# SQL-PSOA-SPARQL: SPARQL to SQL

- If relational PSOA RuleML fact is replaced by corresponding SQL table row, **OID-conclusion PSOA RuleML rule can be used for language-internal transformation of SPARQL-like queries to SQL-like queries** as shown shortly
- **Paradigm-crossing translation step is done by OID-conclusion rule completely within PSOA RuleML**

# SQL-PSOA-SPARQL: SPARQL to SQL (Cont'd)

```
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
```

# SQL-PSOA-SPARQL: SPARQL to SQL (Cont'd)

```
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
```

# SQL-PSOA-SPARQL: SPARQL to SQL (Cont'd)

```
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"
```

# Syntax: Object Identifier Assumption

- **Atomic formula** (predicate application) **without OID** is **assumed** to be shorthand for **this formula with implicit OID**
  - Made syntactically explicit by objectification
  - Then atomic formula is endowed with semantics
- Since RIF does not make OID assumption, it has to separately specify semantics of its OID-less subset, mainly for "named-argument terms" (pairships)

# Syntax: Objectification Algorithm

Basically, while **ground fact can be given fixed OID** (that user neglected to provide), **non-ground fact or rule conclusion needs OID for each grounding**

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

# Syntax: Objectification Example

Earlier relational fact
`betweenRel(pacific canada atlantic)`
is objectified to shelf like earlier *a1*-identified shelf fact,
`_#betweenRel(pacific canada atlantic)`,
and – if `_1` is first new constant from `_1`, `_2`, ... – to
`_1#betweenRel(pacific canada atlantic)`

Query
`betweenRel(?X canada ?Z)`
is syntactically transformed to query
`?#betweenRel(?X canada ?Z)`,
i.e. – if `?1` is first new variable in `?1`, `?2`, ... – to
`?1#betweenRel(?X canada ?Z)`

Posed against fact, it succeeds, with variable bindings
`?1 = _1`, `?X = pacific`, and `?Z = atlantic`

# Syntax: Presentation Variants

*Psoa terms* permit **k slots and m tuples** ($k \geq 0$, $m \geq 0$), with tuple $i$ having length $n_i$ ($1 \leq i \leq m$, $n_i \geq 0$), in left-slot and right-slot normal forms (after objectification):

left-slot $o\#f(p_1\texttt{->}v_1 \ldots p_k\texttt{->}v_k\ [t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}])$

right-slot $o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1\texttt{->}v_1 \ldots p_k\texttt{->}v_k)$

We distinguish three cases (explained for 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 here, square brackets can be omitted (see below: `Positional+Slotted` and `Positional`)
- $m = 0$ For ***tuple-less psoa terms***, frames arise (see `Slotted` and `Member` below)

# Syntax: Presentation Variants (Cont'd)

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

```
Positional+Slotted:   o#f(p1->v1 ... pk->vk  [t1 ... tn])
Positional:           o#f(                   [t1 ... tn])
Slotted:              o#f(p1->v1 ... pk->vk)
Member:               o#f()
```

# Syntax: XML Serialization

- PSOA RuleML is integrated with existing RuleML family
- **Pure PSOA** version of multi-tuple psoa atom augments content of RuleML `<Atom>` node element
- Earlier left-slot normal form results in following XML serialization, where primed meta-variables $p'_i$, $v'_i$, and $t'_{i,j}$ indicate recursive XML serializations of their above presentation-syntax versions (`style` attribute uses the value `"distribution"` to specify built-in slotribution and tupribution):

# Syntax: XML Serialization (Cont'd)

<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>

# Semantics: Overview

- Earlier traces for fact and rule querying as well as use case: exemplify PSOA RuleML's proof-theoretic semantics using **backward reasoning directly for PSOA sources** (queries, facts, and rules)
- PSOA RuleML's model-theoretic semantics also involves **transformations on sources: as preparatory step (objectification) or as restrictions on truth valuation (slotribution and tupribution)**
- Key parts of semantics definitions from RuleML 2011 presented for objectified multi-tuple psoa terms in right-slot normal form

# Semantics: Truth Values and Valuation

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

  This indirectness allows HiLog-like generality

# Semantics: Semantic Structures

## Definition (Semantic structure)

A ***semantic structure***, $\mathcal{I}$, is a tuple of the form
$<\boldsymbol{TV}, \boldsymbol{DTS}, \boldsymbol{D}, \boldsymbol{D}_{\text{ind}}, \boldsymbol{D}_{\text{func}}, \boldsymbol{I}_{\text{C}}, \boldsymbol{I}_{\text{V}}, \boldsymbol{I}_{\text{psoa}}, \boldsymbol{I}_{\text{sub}}, \boldsymbol{I}_{=}, \boldsymbol{I}_{\text{external}}, \boldsymbol{I}_{\text{truth}}>$

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

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

$\boldsymbol{D}_{\text{ind}}$ is used to interpret elements of `Const` acting as individuals
$\boldsymbol{D}_{\text{func}}$ is used to interpret constants acting as function symbols

# Semantics: Semantic Structures (Cont'd)

## Definition (Semantic structure, Cont'd)

3. $\boldsymbol{I}_{\text{psoa}}$ maps $\boldsymbol{D}$ to total functions $\boldsymbol{D}_{\text{ind}} \times \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}}) \times \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}) \rightarrow \boldsymbol{D}$ . Interprets psoa terms, combining positional, slotted, and frame terms, as well as class memberships. Argument $\texttt{d} \in \boldsymbol{D}$ of $\boldsymbol{I}_{\text{psoa}}$ represents function or predicate symbol of positional terms and slotted terms, and object class of frame terms, as well as class of memberships. Element $\texttt{o} \in \boldsymbol{D}_{\text{ind}}$ represents object of class $\texttt{d}$, which is described with two bags.
   - Finite bag of finite tuples $\{<\texttt{t}_{1,1}, ..., \texttt{t}_{1,n_1}>, ..., <\texttt{t}_{m,1}, ..., \texttt{t}_{m,n_m}>\} \in \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}})$ , possibly empty, represents positional information. $\boldsymbol{D}^*_{\text{ind}}$ is set of all finite tuples over the domain $\boldsymbol{D}_{\text{ind}}$. Bags are used since order of tuples in a psoa term is immaterial and tuples may repeat
   - Finite bag of attribute-value pairs $\{<\texttt{a1},\texttt{v1}>, ..., <\texttt{ak},\texttt{vk}>\} \in \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}})$ , possibly empty, represents slotted information. Bags, since order of attribute-value pairs in a psoa term is immaterial and pairs may repeat

# Semantics: Semantic Structures (Cont'd)

## Definition (Semantic structure, Cont'd)

Generic recursive mapping $\boldsymbol{I}$ defined from terms to subterms and ultimately to $\boldsymbol{D}$, for the case of psoa terms using $\boldsymbol{I}_{\text{psoa}}$:

- $\boldsymbol{I}(\ \mathtt{o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ a_1\text{->}v_1 \ldots a_k\text{->}v_k)}\ )$
  $= \boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\mathtt{f}))(\boldsymbol{I}(\mathtt{o}), \{<\boldsymbol{I}(\mathtt{t_{1,1}}), \ldots, \boldsymbol{I}(\mathtt{t_{1,n_1}})>, \ldots, <\boldsymbol{I}(\mathtt{t_{m,1}}), \ldots, \boldsymbol{I}(\mathtt{t_{m,n_m}})>\},$
  $\{<\boldsymbol{I}(\mathtt{a_1}),\boldsymbol{I}(\mathtt{v_1})>, \ldots, <\boldsymbol{I}(\mathtt{a_k}),\boldsymbol{I}(\mathtt{v_k})>\}\ )$
  Again {...} denote *bags* of tuples and attribute-value pairs.

# Semantics: Meaning Attached to Classs, Not OIDs

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

# Semantics: Interpretation of Formulas

## Definition (Truth valuation)

***Truth valuation*** for well-formed formulas in PSOA RuleML determined using function $TVal_{\mathcal{I}}$ (via $\boldsymbol{I}$ and $\boldsymbol{I}_{\text{truth}} : \boldsymbol{D} \rightarrow \boldsymbol{TV}$):

3. *Psoa formula*:
   $TVal_{\mathcal{I}}(\texttt{o\#f([t}_{1,1}\texttt{...t}_{1,n_1}\texttt{]...[t}_{m,1}\texttt{...t}_{m,n_m}\texttt{] a}_1\texttt{->v}_1\texttt{...a}_k\texttt{->v}_k\texttt{)}) =$
   $\boldsymbol{I}_{\text{truth}}(\boldsymbol{I}(\texttt{o\#f([t}_{1,1}\texttt{...t}_{1,n_1}\texttt{]...[t}_{m,1}\texttt{...t}_{m,n_m}\texttt{] a}_1\texttt{->v}_1\texttt{...a}_k\texttt{->v}_k\texttt{)}))$.
   The formula consists of an object-typing membership, a bag of tuples representing a conjunction of all the object-centered tuples (*tupribution*), and a bag of slots representing a conjunction of all the object-centered slots (*slotribution*). Hence use restriction, where $\texttt{m}\geq\texttt{0}$ and $\texttt{k}\geq\texttt{0}$:
   - $TVal_{\mathcal{I}}(\texttt{o\#f([t}_{1,1}\texttt{...t}_{1,n_1}\texttt{]...[t}_{m,1}\texttt{...t}_{m,n_m}\texttt{] a}_1\texttt{->v}_1\texttt{...a}_k\texttt{->v}_k\texttt{)}) = \mathbf{t}$ if and only if
     $TVal_{\mathcal{I}}(\texttt{o \# f}) =$
     $TVal_{\mathcal{I}}(\texttt{o\#Top([t}_{1,1}\texttt{...t}_{1,n_1}\texttt{])}) = \ldots = TVal_{\mathcal{I}}(\texttt{o\#Top([t}_{m,1}\texttt{...t}_{m,n_m}\texttt{])}) =$
     $TVal_{\mathcal{I}}(\texttt{o\#Top(a}_1\texttt{->v}_1\texttt{)}) = \ldots = TVal_{\mathcal{I}}(\texttt{o\#Top(a}_k\texttt{->v}_k\texttt{)}) =$
     $\mathbf{t}$

# Semantics: Interpretation of Formulas (Cont'd)

## Definition (Truth valuation, Cont'd)

3. 
   - Observe that on right-hand side of "if and only if" there are 1+m+k subformulas splitting left-hand side into an object membership, m object-centered positional formulas, each associating the object with a tuple, and k object-centered slotted formulas, i.e. 'triples', each associating object with attribute-value pair. All parts on both sides of "if and only if" are centered on object o, which connects subformulas on right-hand side (first subformula providing o-member class f, remaining m+k ones using root class Top)

   For root class, Top, and all $o \in \boldsymbol{D}$, $\mathit{TVal}_{\mathcal{I}}(o \,\#\, \mathtt{Top}) = \mathbf{t}$.

# Semantics: Interpretation of Formulas (Cont'd)

## Definition (Truth valuation, Cont'd)

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

# Semantics: Distribution vs. Centralization

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

# Semantics: Distribution vs. Centralization (Cont'd)

```
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)
                                )
```

slotribution/tupribution

$\rightleftarrows$

centralization

```
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))
                                )
```

# Implementation: PSOATransRun Framework

- 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 (http://wiki.ruleml.org/index.php/PSOA_RuleML#Implementation)
- The ***translation*** subsystem is chain of translators mapping KB and queries from PSOA to intermediate language
- The ***runtime*** subsystem executes KB queries in intermediate language and extracts results
- Our focus has been on translators, reusing targeted runtime systems as 'black boxes'
- For intermediate languages we have chosen first-order subset of **TPTP** and Horn-logic subset of **ISO Prolog**
- Since these are also standard languages, their translation subsystems of PSOATransRun serve both for PSOA RuleML **implementation and interoperation**

# Implementation: PSOATransRun Framework (Cont'd)

- Targeting TPTP requires fewer translation steps since TPTP systems, being first-order-logic provers, directly accommodate extra expressivity of PSOA (e.g., head existentials introduced by objectification)
- Targeting ISO Prolog requires more since ISO Prolog has lower expressivity of Horn logic (e.g., requiring head-existential translation to Skolem function applications)
- Both **translator chains** start with parsing **PSOA RuleML**'s presentation syntax **into** Abstract Syntax Trees (**ASTs**)
- They then perform transformation steps on AST representations of PSOA sources, via **slotribution/tupribution**-introduced '**primitive' PSOA RuleML constructs**, namely **membership terms**, **slot terms**, and **tuple terms**
- Finally, they map finished **AST** representations **to TPTP or ISO Prolog** presentation syntax as intermediate languages – over distinguished predicates **`memterm`**, **`sloterm`**, and **`tupterm`** defined by TPTP or Prolog clauses – to be executed by respective runtime systems

# Implementation: With PSOA2TPTP to VampirePrime

- PSOATransRun[PSOA2TPTP,VampirePrime] instantiation, realized by Gen Zou and Reuben Peter-Paul with guidance from the author and Alexandre Riazanov, combines PSOA2TPTP translator and VampirePrime runtime system, coded in C++ and accessed through Java
- PSOA2TPTP performs **objectification** and **slotribution**/**tupribution**; then maps transformation result to TPTP
- VampirePrime is **open-source first-order reasoner**. KBs and queries in 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** for interactive exploration, documented on RuleML Wiki. Sample PSOA KB textbox, pre-filled by the system, shows easy transcription of our (RIF-like) presentation-syntax examples into executable PSOA RuleML: PSOA syntax is completed by `Document/Group` wrapper for KBs and "_" prefix for local constants

# Implementation: With PSOA2Prolog to XSB Prolog

- PSOATransRun[PSOA2Prolog,XSBProlog] instantiation, realized by Gen Zou with guidance from the author, combines PSOA2Prolog translator and XSB Prolog runtime system, coded in C++ and accessed via Java API
- PSOA2Prolog augments translation chain of PSOA2TPTP and performs different target mapping. Composed of source-to-source normalizer and mapper to pure Prolog (Horn logic) subset of ISO subset of XSB Prolog. **Normalizer** is composed of five transformation layers, namely **objectification**, **Skolemization**, **slotribution**/ **tupribution**, **flattening**, and **rule splitting**. Each layer is self-contained component reusable for processing PSOA KBs in other applications. **Mapper** performs recursive transformation from normalization result to **Prolog clauses**
- XSB Prolog is **fast Prolog engine**, which is targeted because it enables tabling, supporting both termination and efficiency. XSB executes queries over intermediate-Prolog KBs, and PSOATransRun performs answer extraction

# Conclusions: Results

- **Integrated object-relational data and rules of PSOA RuleML enable novel semantic modeling and analysis based on positional-slotted, object-applicative terms**
- PSOA RuleML's data model visualized in Grailog provides logical foundation and visual intuition via 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**
- Also serves as intermediate language for bidirectional **query transformation, e.g. between SQL and SPARQL**
- **Syntax** & **semantics** capture object-relational integration
- Two **implemented open-source PSOATransRun instantiations** allow rapid PSOA RuleML prototyping
- Use cases include MusicAlbumKB and **GeospatialRules**, which started with Gen Zou's Datalog$^+$ rulebase for Region Connection Calculus and is being expanded into Hornlog$^+$ rulebase for geolocation

# Conclusions: Plans

- Further enrich GeospatialRules use case for collaborations with **WSL** (e.g., ΔForest case study) and other partners
- Envision complementary PSOA RuleML applications, in various collaborations, for data querying and interchange in domains of **biomedicine**, **finance**, and **social media**
- Develop PSOA RuleML from (head-existential-)extended Horn logic (PSOA Hornlog[∃] RuleML) to **first-order logic** (PSOA FOL RuleML), facilitated by PSOA-extension focus on atomic formulas (invariant w.r.t. Hornlog[∃] vs. FOL)
- Explore options for PSOATransRun[PSOA2TPTP,SNARK] instantiation using **SRI's SNARK reasoner** for efficient treatment of `memterm`s (sorts) and equality (paramodulation) – with upgrade to PSOA FOL RuleML
- Use PSOA RuleML in knowledge interchange for systems like **Quadri NLIDB** (over clinical databases)
- Transfer Inductive Logic Programming (ILP) techniques to object-relational data **discovering object-relational rules**