The PSOATransRun 1.0 System for Object-Relational Reasoning in RuleML

The 6th Atlantic Workshop on Semantics and Services (AWoSS 2015)
December 9, 2015

Gen Zou

Faculty of Computer Science,
University of New Brunswick, Fredericton, Canada

Outline

Outline

1. Introduction
2. Normalization of PSOA Source
3. Mapping the Normalized PSOA Source to Prolog
4. Conclusions

Introduction – PSOA RuleML

- Uses **p**ositional-**s**lotted **o**bject-**a**pplicative (**psoa**) terms, permitting the application of a predicate (acting as a relation) to be [in an *oidless*/*oidful* dimension] without or with an Object IDentifier (OID) – typed by the predicate (acting as a class) – and the predicate's arguments to be [in an orthogonal dimension] *positional*, *slotted*, or *combined*

  General case (multi-tuple), where "#" means "member of":

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

  Special cases (optional single-tuple brackets):

Relationship:	$\texttt{f([t}_1\texttt{ ... t}_n\texttt{])}$
Shelf:	$\texttt{o\#f([t}_1\texttt{ ... t}_n\texttt{])}$
Frame:	$\texttt{o\#f(p}_1\texttt{->v}_1\texttt{ ... p}_k\texttt{->v}_k\texttt{)}$
Shelframe:	$\texttt{o\#f([t}_1\texttt{ ... t}_n\texttt{] p}_1\texttt{->v}_1\texttt{ ... p}_k\texttt{->v}_k\texttt{)}$

- For an **oidless psoa term**, i.e. one without a 'user' OID, *objectification* will introduce a 'system'-generated OID
- Psoa terms without class typing are expressed by `Top`, specifying the root of class hierarchy

PSOA RuleML – Presentation Syntax (PS)

Integrates W3C RIF for relationships and frames (see above)

As in RIF PS:

- "*oid* is member of *class*" written as "*oid*#*class*" (see above)
- "$class_1$ is subclass of $class_2$" written as "$class_1$ ## $class_2$"
- Local constants prefixed by underscore ('_'); variables prefixed by question mark ('?')

Introduction – PSOATransRun 1.0

- **Efficient** reasoning in PSOA RuleML enabled
  - All forms of psoa terms, including relationships, shelves, pairships, and (shel)frames
  - (OID-)head-existential rules
  - Equality in the body, restricted to unification and external-function evaluation
  - Subclass formulas for ‘ABox’ reasoning only
  - Built-in arithmetic functions
- **Released** in Java source form and as an executable jar file. Downloadable from: `http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun`

Introduction – PSOATransRun 1.0

- Includes a composition of translator PSOA2Prolog and well-known XSB Prolog engine
  - PSOA2Prolog translates Knowledge Bases (KBs) and queries in PSOA RuleML presentation syntax into a subset of the logic programming language ISO Prolog
  - XSB Prolog engine provides efficient query answering over translated Prolog KB
- PSOA2Prolog performs a multi-step source-to-source normalization followed by a mapping to the pure (Horn) subset of ISO Prolog

Outline

Normalization

1. Objectification
2. Skolemization
3. Slotribution and Tupribution
4. Flattening
5. Rule Splitting

Static vs. Dynamic Objectification – Preview

KB:

```
_hire(_Ernie _Kate)
```

Query:

```
?O#_hire(?X ?Y)
```

Static: Generate fresh OID constant from '`_1`', '`_2`', ...
(transform above **KB** ground atom, use **query** unchanged):

```
_1#_hire(_Ernie _Kate)
```

Dynamic: Virtualize with '`_oidcons`' function and equality '='
(keep above **KB** unchanged, transform **query** atom):

```
And(_hire(?X ?Y) ?O = _oidcons(_hire ?X ?Y))
```

Objectification – From Static to Static/Dynamic

- Static: generate OIDs for all of the KB's oidless atoms
- Static/Dynamic (novel refinement)
  - Leave unchanged as many of the KB's oidless atoms as possible, instead dynamically constructing virtual OIDs as query variable bindings
  - Partition the set of KB predicates into two disjoint subsets: *non-relational* (at least one occurrence in a multi-tuple, oidful, or slotted atom, or in a subclass formula) and *relational* (no such occurrence)
  - Statically generate OIDs only for the KB's oidless psoa atoms with non-relational predicates

Static/Dynamic Objectification

- Perform OID virtualization for queries with OID variables corresponding to the KB's psoa atoms with relational predicates
    - Query atoms using the KB's relational predicates with OID variables are rewritten via equalities that unify an OID variable with a (Skolem-like) OID-constructor ('`_oidcons`') function application
    - Allow users to pose queries with OIDs regardless of whether the underlying KB clauses have OIDs or not
    - Make maximum use of the underlying Prolog engine for efficient inference on KB clauses with relational predicates

Dynamic Objectification by OID Virtualization

- Leave all **KB atoms** with relational predicates unchanged
- For each **query atom** ψ using a relational KB predicate `f`:
  - If ψ is a relationship, keep it unchanged
  - If ψ has a **non-variable** OID or a slot, rewrite it to explicit falsity, here encoded as `Or()`
  - If ψ has an OID **variable** and *m* tuples, being of the form
    $\texttt{?O\#f([}t_{1,1} \; \ldots \; t_{1,n_1}\texttt{]} \; \ldots \; \texttt{[}t_{m,1} \; \ldots \; t_{m,n_m}\texttt{])}$,
    equivalent to a tupribution-like conjunction, copying `?O#f`,
    $\texttt{And(?O\#f(}t_{1,1} \; \ldots \; t_{1,n_1}\texttt{)} \; \ldots \; \texttt{?O\#f(}t_{m,1} \; \ldots \; t_{m,n_m}\texttt{)}$,
    rewrite it to a relational conjunction using equality

$$\texttt{And(f(}t_{1,1} \; \ldots \; t_{1,n_1}\texttt{) ?O = \_oidcons(f } t_{1,1} \; \ldots \; t_{1,n_1}\texttt{)}$$
$$\ldots$$
$$\texttt{f(}t_{m,1} \; \ldots \; t_{m,n_m}\texttt{) ?O = \_oidcons(f } t_{m,1} \; \ldots \; t_{m,n_m}\texttt{))}$$

Skolemization

- Specialized FOL Skolemization is employed to eliminate existentials in rule conclusions or facts, which are not allowed in Prolog
- Replace each existential formula `Exists ?X` (σ) in a rule conclusion or a fact with σ[`?X/_skolem`*k*`(?v`$_1$ `...` `?v`$_m$`)`], where each occurrence of `?X` in σ becomes a Skolem function `_skolem`*k* applied to all universally quantified variables `?v`$_1$ `...` `?v`$_m$ from the clause's quantifier prefix
- **New skolem function name** `_skolem`*k* ($k = 1, 2, \ldots$) generated for each existential variable

Slotribution and Tupribution

- Rewrite each psoa atom

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

  into a conjunction

  $\texttt{And(o\#f}$
  $\quad \texttt{o\#Top(}t_{1,1}\ \texttt{...}\ t_{1,n_1}\texttt{)}\ \texttt{...}\ \texttt{o\#Top(}t_{m,1}\ \texttt{...}\ t_{m,n_m}\texttt{)}$
  $\quad \texttt{o\#Top(}p_1\texttt{->}v_1\texttt{)}\ \texttt{...}\ \texttt{o\#Top(}p_k\texttt{->}v_k\texttt{))}$

Flattening

- Extract each embedded interpreted function application as a separate equality
- Each atomic formula φ (in a rule premise or a query) that embeds an external function application ψ, which is not on the top level of an equality, is replaced with `And(?`i`=`ψ $\varphi[\psi$`/?`$i]$`)`, where `?`i is the first variable in `?1`, `?2`, ... that does not occur in the enclosing rule

Rule Splitting

- Remove conjunctions in rule conclusions, which are not supported in Prolog
- Each rule with an *n*-ary conjunction in the conclusion

  `Forall ?v`$_1$ `... ?v`$_m$ `(And(`φ_1 `...` φ_n`) :-` φ'`)`

  is split into *n* rules

  `Forall ?v`$_1$ `... ?v`$_m$ `(`φ_1 `:-` φ'`)`

  ...

  `Forall ?v`$_1$ `... ?v`$_m$ `(`φ_n `:-` φ'`)`

Outline

Simple Terms

Use recursive mapping function denoted by ρ_{psoa}

- Constants
  - If c is a number, $\rho_{psoa}(c)$ is the corresponding Prolog number
  - If c is an arithmetic built-in, $\rho_{psoa}(c)$ is the corresponding Prolog built-in
  - Otherwise, $\rho_{psoa}(c)$ is the single-quoted version of c
- Variables
  For a '`?`'-prefixed variable v, $\rho_{psoa}(v)$ replaces '`?`' with the upper-case letter '`Q`' (`Q`uestion mark)

Central PSOA Constructs

Mapping from PSOA/PS constructs to Prolog constructs

(To accommodate the relationships preserved by dynamic objectification, the 4th-row mapping of $\texttt{f(t}_1\texttt{...t}_k\texttt{)}$ has to be extended from functions to predicates $\texttt{f}$)

PSOA/PS Constructs	Prolog Constructs
$\texttt{o\#Top(t}_1\texttt{...t}_k\texttt{)}$	$\texttt{tupterm(}\rho_{psoa}\texttt{(o),}\rho_{psoa}\texttt{(t}_1\texttt{) ...}\rho_{psoa}\texttt{(t}_k\texttt{))}$
$\texttt{o\#Top(p->v)}$	$\texttt{sloterm(}\rho_{psoa}\texttt{(o),}\rho_{psoa}\texttt{(p),}\rho_{psoa}\texttt{(v))}$
$\texttt{o\#c()}$	$\texttt{memterm(}\rho_{psoa}\texttt{(o),}\rho_{psoa}\texttt{(c))}$
$\texttt{f(t}_1\texttt{...t}_k\texttt{)}$	$\rho_{psoa}\texttt{(f)(}\rho_{psoa}\texttt{(t}_1\texttt{), ...,}\rho_{psoa}\texttt{(t}_k\texttt{))}$
$\texttt{And(f}_1\texttt{ ... f}_n\texttt{)}$	$\texttt{(}\rho_{psoa}\texttt{(f}_1\texttt{), ...,}\rho_{psoa}\texttt{(f}_n\texttt{))}$
$\texttt{Or(f}_1\texttt{ ... f}_n\texttt{)}$	$\texttt{(}\rho_{psoa}\texttt{(f}_1\texttt{); ...;}\rho_{psoa}\texttt{(f}_n\texttt{))}$
$\texttt{Exists ?v}_1\texttt{ ... ?v}_m\texttt{ (}\varphi\texttt{)}$	$\rho_{psoa}(\varphi)$
$\texttt{Forall ?v}_1\texttt{ ... ?v}_m\texttt{ (}\varphi\texttt{)}$	$\rho_{psoa}(\varphi)$
$\varphi\ \texttt{:-}\ \psi$	$\rho_{psoa}(\varphi)\ \texttt{:-}\ \rho_{psoa}(\psi)\texttt{.}$
$\texttt{?v=External(f(t}_1\texttt{...t}_k\texttt{))}$	$\texttt{is(}\rho_{psoa}\texttt{(?v),}\ \rho_{psoa}\texttt{(f)(}\rho_{psoa}\texttt{(t}_1\texttt{), ...,}\rho_{psoa}\texttt{(t}_k\texttt{)))}$
$\texttt{c1\#\#c2}$	$\texttt{memterm(X,}\rho_{psoa}\texttt{(c2)) :- memterm(X,}\rho_{psoa}\texttt{(c1)).}$

Outline

Conclusions

- PSOATransRun 1.0 supports reasoning in PSOA RuleML by composing the efficient translator PSOA2Prolog and the efficient run-time engine XSB Prolog
- PSOA2Prolog performs a multi-step source-to-source normalization followed by a mapping to the pure (Horn) subset of ISO Prolog
- Normalization employs a novel static/dynamic approach for the objectification step, which makes optimal use of the underlying Prolog engine for efficient inference on KB clauses with relational predicates
- Future work includes adding support of other PSOA features, e.g. an expanded set of built-ins, and more detailed functionality and performance comparisons with realizations of other rule languages, e.g. Flora-2 and EYE