Translators for Interoperating and Porting Object-Relational Knowledge

RuleML Webinar
April 27, 2018

Gen Zou

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

Outline

1. Background & Related Work
2. Revising PSOA RuleML for Version 1.0
3. Interoperating and Porting PSOA RuleML
4. Use Cases
5. Evaluation
6. Conclusions and Future Work

Outline

Rule Languages

- Provide a foundation for data and knowledge representation as well as problem solving in AI, Semantic Web, and IT at large
- Used to express
  - Knowledge for semantic data access
  - Associations among data
  - Privacy/security/trust policies
  - Business logics
  - Legal norms
  - Biomedical concept definitions
  - ...
- Paradigms of modeling entity connections
  - Relational
  - Object-centered
  - Combined
- Since systems have been developed on top of languages with different paradigms, it is often necessary to translate, integrate, and reuse Knowledge Bases (KBs) expressed in different languages and/or different paradigms

Relational Rule Languages (1)

- Widely used for representing, e.g., First-Order Logic (FOL) and Logic Programming (LP)
- Model dependency among *n* entities as an **atom**, here as an *n*-ary predicate applied to a **tuple**, which is a sequence of *n* positional arguments

Example of Fact and Rules (in an abstract syntax): Symmetry and Projection

$$betweenRel(canada, usa, mexico)$$

$$\forall Outer1, Inner, Outer2 :$$
$$betweenRel(Outer2, Inner, Outer1) \Leftarrow betweenRel(Outer1, Inner, Outer2)$$

$$\forall Outer1, Inner, Outer2 :$$
$$neighborRel(Outer1, Inner) \Leftarrow betweenRel(Outer1, Inner, Outer2)$$

Relational Rule Languages (2)

- Based on predicate logic, especially FOL and its variants/subsets
- TPTP-FOF
  - Dialect of TPTP (Thousands of Problems for Theorem Provers), a widely used language for interoperating KBs between automated theorem provers
  - Can express the First-Order Formulas (FOF) of FOL
- Prolog
  - Widely used LP language, with an ISO standard
  - Pure Prolog can also be seen as a subset of FOL

Object-Centered Rule Languages (1)

- Receive increasing attention because of expanding research and development in linked data on the Web, graph / knowledge stores, and big data in NoSQL DBs
- An object is represented by a unique Object IDentifier (**OID**) typed by zero or more classes and described by an unordered collection of **slots**, each being a pair of a name and a filler
- An OID-describing slotted atom in AI is called a **frame**

Example of Fact and Rule: Slot Introduction

Syntax: "#" denotes membership; "→" connects the slot name and filler

$b1\#betweenObj(outer1 \rightarrow canada; inner \rightarrow usa; outer2 \rightarrow mexico)$

$\forall B, Out1, In:$

$$Out1\#space(neighborSlot \rightarrow In) \Leftarrow B\#betweenObj(outer1 \rightarrow Out1; inner \rightarrow In)$$

Object-Centered Rule Languages (2)

- Notation 3 (N3)
  - Initially defined by Tim Berners-Lee
  - Extends RDF, a W3C language representing information in the Web, with rules

Object-Relational Rule Languages

- Combine the object-centered and relational paradigms, either in a heterogeneous or a homogeneous way
- Heterogeneous
  1. Allow atoms in both object-centered and relational forms, even mixed in the same rule
  2. Flora-2/F-logic and RIF
- Homogeneous
  1. In addition to item Heterogeneous, sub-item 1, integrate object-centered and relational atoms to a unified form
  2. PSOA RuleML

Example of Fact and Rule: Slot Introduction

$$b1\#betweenObjRel(canada, usa, mexico; dim \rightarrow 2; orient \rightarrow northSouth)$$

$$\forall B, Out1, In, Out2 :$$

$$Out1\#space(neighborSlot \rightarrow In) \Leftarrow B\#betweenObjRel(Out1, In, Out2)$$

Related Work on Rule Interoperation

- Standardized languages for interoperation: RuleML, W3C RIF, ISO Common Logic, OMG OntoIOp
- Translations between different languages/logics
  - RuleML-facilitated translation between N3 and Prolog
  - Relational data in DBs exposed as RDF
  - FOL subsets to Answer Set Programs (ASPs)
  - Object-centered language Knowledge Machine to ASPs

Open Problems

- There were few studies on the syntax and semantics of languages constituting homogeneous object-relational combinations
- The interoperation between these, as exemplified by PSOA, on one hand and purely relational or purely object-centered rule languages on the other hand was not investigated.
  In particular, it was an open question whether the translations required for interoperation are semantics-preserving
- There was no reasoning system available for answering queries posed to KBs in PSOA RuleML. It was open whether translator-based implementation/portation is appropriate not only for reusability and maintainability but also
  for use cases and applications

Objectives (1)

- Overall architecture
  - Design a translator-based architecture for interoperating and porting object-relational knowledge
- Translations and translator-based reasoning systems
  - Study the interoperation from PSOA RuleML to the purely relational languages TPTP and Prolog as well as from the purely object-centered N3 to PSOA RuleML
  - Characterize sublanguages of PSOA RuleML for which the translations are semantics-preserving, i.e. sound and complete
  - Focusing on semantically compatible sublanguages, realize translators based on the proposed translations
  - Using the translators, on top of multiple runtime engines, realize prototype reasoning systems for PSOA RuleML query answering

Objectives (2)

- Evaluation through use cases and test cases
  - Apply the PSOA RuleML language and its translator-based reasoning systems to use cases
  - Develop test cases and evaluate the realized reasoning systems for PSOA
- Revision of PSOA RuleML
  - Revise the syntax and semantics of PSOA RuleML based on findings in the development and the evaluation. Update the translators accordingly

Outline

PSOA RuleML Prior to Version 1.0

- Positional-Slotted Object-Applicative (PSOA) RuleML permits atom to apply predicate – possibly identified by OID typed by predicate – to bag of tupled descriptors (tuples) and to bag of slotted descriptors (slots)

  General case (multi-tuple):

  Oidless: **f**$([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)$

  Oidful: **o#f**$([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)$

  Special cases (single-tuple brackets and membership parentheses are optional):

  Relationship: $f([t_1 \ldots t_n])$
  Frame: $o\#f(\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)$
  Combined: $o\#f([t_1 \ldots t_n]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)$
  Membership: $o\#f()$

- The predicate `f` can be `Top`, denoting the root predicate

New Kinds of Descriptors in Psoa Atoms

- Orthogonally to the tupled vs. slotted distinction, a descriptor in an atom can be independent or dependent on the atom's predicate:
  `http://ruleml.org/talks/PSOAPerspectivalKnowledge-talk.pdf`
- Descriptors dependent on predicate are sensitive to predicate scope, and can only be queried with the same predicate
- Descriptors independent from predicate are not sensitive to predicate scope, and can be queried with a different predicate
- Oidless-vs.-oidful and tupled-vs.-slotted-vs.-tupled+slotted dimensions of atoms are augmented by 3rd dimension of perspectivity:
  - Perspeneutral: having one or more independent descriptors
  - Perspectival: having one or more dependent descriptors
  - Perspeneutral+perspectival: combining one or more independent plus one or more dependent descriptors

Presentation Syntax of General Psoa Terms (Atoms and Expressions)

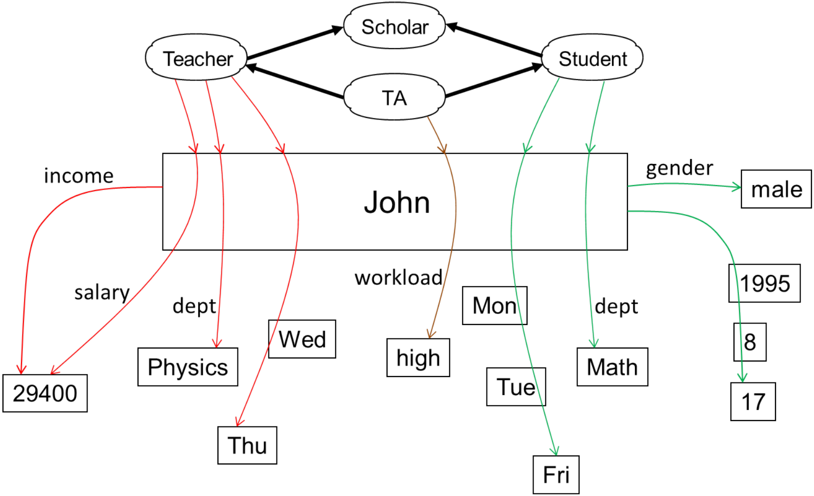
Four "…"-subsequences for four kinds of descriptors, where superscripts indicate subterms that are part of dependent ($^+$) vs. independent ($^-$) descriptors ($\mathtt{m}^+, \mathtt{m}^-, \mathtt{k}^+, \mathtt{k}^- \geq 0$, $\mathtt{n}^+_{\mathtt{i}^+}, \mathtt{n}^-_{\mathtt{i}^-} \geq 0$ for any $\mathtt{i}^+$ and $\mathtt{i}^-$ such that $1 \leq \mathtt{i}^+ \leq \mathtt{m}^+$ and $1 \leq \mathtt{i}^- \leq \mathtt{m}^-$):

$$
\begin{array}{l}
\circ\#\mathtt{f}(+[\mathtt{t}^+_{1,1} \ldots \mathtt{t}^+_{1,\mathtt{n}^+_1}] \ldots +[\mathtt{t}^+_{\mathtt{m}^+,1} \ldots \mathtt{t}^+_{\mathtt{m}^+,\mathtt{n}^+_{\mathtt{m}^+}}] \\
\quad -[\mathtt{t}^-_{1,1} \ldots \mathtt{t}^-_{1,\mathtt{n}^-_1}] \ldots -[\mathtt{t}^-_{\mathtt{m}^-,1} \ldots \mathtt{t}^-_{\mathtt{m}^-,\mathtt{n}^-_{\mathtt{m}^-}}] \\
\quad \mathtt{p}^+_1 {+}{>} \mathtt{v}^+_1 \ldots \mathtt{p}^+_{\mathtt{k}^+} {+}{>} \mathtt{v}^+_{\mathtt{k}^+} \\
\quad \mathtt{p}^-_1 {-}{>} \mathtt{v}^-_1 \ldots \mathtt{p}^-_{\mathtt{k}^-} {-}{>} \mathtt{v}^-_{\mathtt{k}^-})
\end{array}
$$

Relationships now have the form $\mathtt{f}(+[\mathtt{t}^+_{1,1} \ldots \mathtt{t}^+_{1,\mathtt{n}^+_1}])$, where brackets can be omitted for $\mathtt{n}^+_1 \geq 1$

Rich TA Example (Facts Only): Graphical View

Individual OID John described independently and under the perspectives of predicates Teacher, TA, Student



Rich TA Example (Rule Added): Presentation Syntax

```
Document (
  ...
  Group (
    _Teacher##_Scholar                                    % Taxonomy
    _Student##_Scholar
    _TA##_Teacher
    _TA##_Student
    _John#_Teacher(+[_Wed _Thu]                           % Data
                   _coursehours+>12 _dept+>_Physics
                   _salary+>29400 _income->29400)
    _John#_Student(+[_Mon _Tue _Fri] -[1995 8 17]
                   _coursehours+>20 _dept+>_Math _gender->_male)

    Forall ?o ?ht ?hs (                     % Rule
      ?o#_TA(_workload+>_high) :-           % ":-" stands for "⇐"
        And(?o#_Teacher(_coursehours+>?ht)
            math:greaterThan(?ht 10)        % ?ht>10
            ?o#_Student(_coursehours+>?hs)
            math:greaterThan(?hs 18))       % ?hs>18
    )
  )
)
```

Revision of Semantics: Old Semantics

- Cannot formalize perspectival knowledge, e.g. for Rich TA Example
- Can only interpret an oidless psoa term after applying static objectification
- Cannot deal with an expression term – which returns an arbitrary value – since giving it an OID would make the function act as the class of the OID and lead to a truth value
- Causes reasoning overhead for an atom whose predicate in the KB clauses is used only as a Prolog-like relation, e.g. does not occur with an OID or slots

Revision of Semantics: New Semantics

- Allow direct interpretation and truth evaluation of oidless psoa terms
- Add **objectification restriction**

$$TVal_{\mathcal{I}}(\texttt{f(...)}) = \mathbf{t}$$

  if and only if

$$TVal_{\mathcal{I}}(\texttt{Exists ?O (?O\#f(...))}) = \mathbf{t}$$

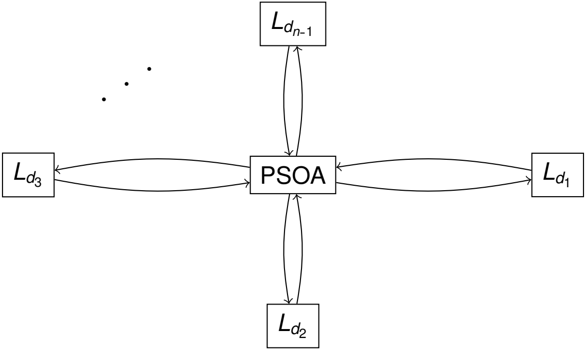
- Incorporate semantics of independent and dependent descriptors for psoa terms
- Update describution restriction for independent and dependent descriptors (cf. later)

Outline

Outline

PSOA-Centered Interoperation Framework

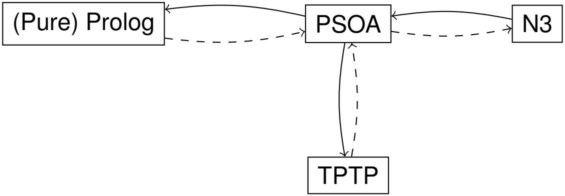
Employs PSOA as the canonical language and implementations of bidirectional translations between PSOA and any designated language



L_{d_i}-to-L_{d_j} translation can be composed from
L_{d_i}-to-PSOA translation and PSOA-to-L_{d_j} translation

Specialization of Interoperation Framework

Specialization of interoperation framework for N3, Prolog, and TPTP



Implemented translations indicated by solid arrows:

- Translations composed of a normalization within source language and a conversion from normalized source to target language
- Normalization within PSOA composed of modularized steps

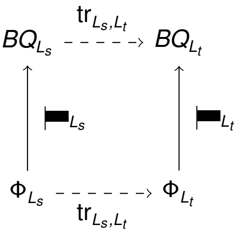
PSOATransRun Portation Framework

- PSOATransRun[PSOA2L_t,*runtime*] framework provides implementations of PSOA RuleML query answering (through porting PSOA KBs and queries) by
  - Translating input KB and query into already implemented L_t, using the translator component PSOA2L_t
  - Executing translated query against translated KB in *runtime* reasoning engine to get the answers
  - Translating the answers in L_t back to PSOA, using a (partial) translator from L_t to PSOA that acts only on base terms but not necessarily KBs and formulas. This component is dependent on the translator PSOA2L_t and can be implemented as a supplement to PSOA2L_t, hence is omitted from the bracketed notation
- Instantiations of PSOATransRun (project repository of sources: `https://github.com/RuleML/PSOATransRunComponents`)
  - PSOATransRun[PSOA2TPTP,VampirePrime]: $L_t = \mathrm{TPTP}$, combining PSOA2TPTP and VampirePrime engine
  - PSOATransRun[PSOA2Prolog,XSBProlog]: $L_t = \mathrm{Prolog}$, combining PSOA2Prolog and XSB Prolog engine

Semantics-Preserving Translation

For a translation tr_{L_s,L_t} from the source language L_s to the target language L_t:

- **Sound**: all entailments that hold after translation to L_t already hold in L_s
- **Complete**: all entailments in L_s still hold after translation to L_t
- **Semantics preserving** = sound + complete
- Semantics-preserving translation required for porting KBs and queries in L_s to L_t:



Outline

Unnesting

- Embedded psoa atoms
  - Widely used in object-centered languages such as RDF, N3, and Flora-2/F-logic as a shorthand notation
  - PSOA RuleML supports the use of embedded oidful atoms, e.g. `o1#c(p->f(o2#d))`
- Unnesting transformation decomposes nested atomic formulas into equivalent conjunctions

$$\text{Unnest}(\texttt{o1\#c(p->f(o2\#d))})$$
$$= \texttt{And(o2\#d o1\#c(p->f(o2)))}$$

Objectification: Systematics

- Objectification transformation realizes the objectification restriction by transforming KBs and queries such that entailments can be established under a relaxed semantics in which the restriction is no longer required
- Systematics of objectification transformation of KBs/queries
  - **Static**: generate explicit OIDs for **all** of the KB's oidless atoms
    - **Undifferentiated**: uniformly transforms oidless atoms everywhere using explicit existentials
    - **Differentiated**: transforms oidless atoms based on their occurrences using Skolem-like constants etc.
  - **Static/Dynamic** (novel enhancement): avoid generating explicit OIDs for relational predicates, instead constructing virtual OIDs as query variable bindings

Static vs. Dynamic Objectification of Atoms

KB: `_work(_Kate _Rho4biz "Director")`

Query: `?O#_work(?P ?C ?J)`

(Users can pose oidless/oidful queries regardless of whether the underlying KB clauses have OIDs or not)

Static: Generate explicit OID
(transform above **KB** ground atom, use **query** unchanged):

- Undifferentiated (using existential OID variable):
  `Exists ?1 (?1#_work(_Kate _Rho4biz "Director"))`
- Differentiated (using fresh OID constant):
  `_1#_work(_Kate _Rho4biz "Director")`

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

```
And(_work(?P ?C ?J)
    ?O=_oidcons(_work ?P ?C ?J))
```

Describution Transformation

Realizes **describution restriction** of semantics by replacing every oidful psoa atom having general form

$$
\begin{aligned}
\texttt{o\#f(}&\texttt{+[}t^+_{1,1}\ \texttt{...}\ t^+_{1,n^+_1}\texttt{]}\ \texttt{...}\ \texttt{+[}t^+_{m^+,1}\ \texttt{...}\ t^+_{m^+,n^+_{m^+}}\texttt{]}\\
&\texttt{-[}t^-_{1,1}\ \texttt{...}\ t^-_{1,n^-_1}\texttt{]}\ \texttt{...}\ \texttt{-[}t^-_{m^-,1}\ \texttt{...}\ t^-_{m^-,n^-_{m^-}}\texttt{]}\\
&p^+_1\texttt{+>}v^+_1\ \texttt{...}\ p^+_{k^+}\texttt{+>}v^+_{k^+}\\
&p^-_1\texttt{->}v^-_1\ \texttt{...}\ p^-_{k^-}\texttt{->}v^-_{k^-}\texttt{)}
\end{aligned}
$$

with the conjunction

$$
\begin{aligned}
&\texttt{And(o\#f}\\
&\quad \texttt{o\#f(+[}t^+_{1,1}\ \texttt{...}\ t^+_{1,n^+_1}\texttt{])}\ \texttt{...}\ \texttt{o\#f(+[}t^+_{m^+,1}\ \texttt{...}\ t^+_{m^+,n^+_{m^+}}\texttt{])}\\
&\quad \texttt{o\#Top(-[}t^-_{m^-,1}\ \texttt{...}\ t^-_{m^-,n^-_{m^-}}\texttt{])}\ \texttt{...}\ \texttt{o\#Top(-[}t^-_{m^-,1}\ \texttt{...}\ t^-_{m^-,n^-_{m^-}}\texttt{])}\\
&\quad \texttt{o\#f(}p^+_1\texttt{+>}v^+_1\texttt{)}\ \texttt{...}\ \texttt{o\#f(}p^+_{k^+}\texttt{+>}v^+_{k^+}\texttt{)}\\
&\quad \texttt{o\#Top(}p^-_1\texttt{->}v^-_1\texttt{)}\ \texttt{...}\ \texttt{o\#Top(}p^-_{k^-}\texttt{->}v^-_{k^-}\texttt{))}
\end{aligned}
$$

Other Transformation Steps

- **Skolemization**: Eliminates existentially quantified formulas in rule conclusions by replacing existential variables with Skolem function applications
- Subclass transformation
  - Subclass axiomatization: Adds axiomatization rules to each KB
  - **Subclass rewriting** (employed for implementation): Replaces each subclass formula with a rule
- **Flattening external expressions**: Extracts each embedded external expression as a separate equality
- **Conjunctive conclusion splitting**: Splits each rule with conjunctive conclusion into multiple rules

Outline

FOL-Targeting Normalization and Conversion

- FOL-targeting normalization: Sequential composition of unnesting, subclass rewriting, objectification, and describution
- Conversion from FOL-Normalized PSOA to TPTP:

PSOA/PS Formulas	TPTP Formulas
`o#Top(-[`$t_1 \ldots t_n$`])`	`tupterm(`$\zeta'(o),\zeta'(t_1), \ldots ,\zeta'(t_n)$`)`
`o#f(+[`$t_1 \ldots t_n$`])`	`prdtupterm(`$\zeta'(o),\zeta'(f),\zeta'(t_1), \ldots ,\zeta'(t_n)$`)`
`o#Top(p->v)`	`sloterm(`$\zeta'(o), \zeta'(p), \zeta'(v)$`)`
`o#f(p+>v)`	`prdsloterm(`$\zeta'(o), \zeta'(f), \zeta'(p), \zeta'(v)$`)`
`o#c`	`memterm(`$\zeta'(o),\zeta'(c)$`)`
`f(+[`$t_1 \ldots t_n$`])`	$\begin{cases} \zeta'(f) & n = 0 \\ \zeta'(f)(\zeta'(t_1), \ldots ,\zeta'(t_n)) & n > 0 \end{cases}$
`And(`$\tau_1 \ldots \tau_n$`)`	$(\zeta'(\tau_1)$ `&` ... `&` $\zeta'(\tau_n))$
`Or(`$\tau_1 \ldots \tau_n$`)`	$(\zeta'(\tau_1)$ `|` ... `|` $\zeta'(\tau_n))$
`Exists` $?X_1 \ldots ?X_n$ (τ)	`?[`$\zeta'(X_1), \ldots ,\zeta'(X_n)$`]:` $\zeta'(\tau)$
`Forall` $?X_1 \ldots ?X_n$ (τ)	`![`$\zeta'(X_1), \ldots ,\zeta'(X_n)$`]:` $\zeta'(\tau)$
τ_1 `:-` τ_2	$\zeta'(\tau_1)$ `<=` $\zeta'(\tau_2)$
$\tau_1 = \tau_2$	$\zeta'(\tau_1) = \zeta'(\tau_2)$

Outline

LP-Targeting Normalization and Conversion

- LP-targeting normalization: FOL-targeting normalization followed by Skolemization, external flattening, and conjunctive conclusion splitting
- Conversion from LP-Normalized PSOA to Prolog:

PSOA/PS Formulas	Prolog Formulas
$\texttt{o\#Top(-[t}_1\texttt{...t}_n\texttt{])}$	$\texttt{tupterm}(\rho'(\texttt{o}),\rho'(\texttt{t}_1),\ \ldots,\rho'(\texttt{t}_n))$
$\texttt{o\#f(+[t}_1\texttt{...t}_n\texttt{])}$	$\texttt{prdtupterm}(\rho'(\texttt{o}),\rho'(\texttt{f}),\rho'(\texttt{t}_1),\ \ldots,\rho'(\texttt{t}_n))$
$\texttt{o\#Top(p->v)}$	$\texttt{sloterm}(\rho'(\texttt{o}),\ \rho'(\texttt{p}),\ \rho'(\texttt{v}))$
$\texttt{o\#f(p+>v)}$	$\texttt{prdsloterm}(\rho'(\texttt{o}),\ \rho'(\texttt{f}),\ \rho'(\texttt{p}),\ \rho'(\texttt{v}))$
$\texttt{o\#c}$	$\texttt{memterm}(\rho'(\texttt{o}),\rho'(\texttt{c}))$
$\texttt{f(+[t}_1\texttt{...t}_n\texttt{])}$	$\begin{cases} \rho'(\texttt{f}) & \texttt{n} = 0 \\ \rho'(\texttt{f})(\rho'(\texttt{t}_1),\ldots,\rho'(\texttt{t}_n)) & \texttt{n} > 0 \end{cases}$
$\tau_1 = \tau_2$	$\begin{cases} \texttt{is}(\rho'(\tau_1),\rho'(\tau_2)) & \text{if } \tau_2 \text{ is } \texttt{External(...)} \\ \texttt{'='}(\rho'(\tau_1),\rho'(\tau_2)) & \text{otherwise} \end{cases}$
$\texttt{And}(\tau_1\ \ldots\ \tau_n)$	$(\rho'(\tau_1),\ \ldots,\rho'(\tau_n))$
$\texttt{Or}(\tau_1\ \ldots\ \tau_n)$	$(\rho'(\tau_1);\ \ldots;\rho'(\tau_n))$
$\texttt{Exists ?X}_1\ldots\texttt{?X}_n\ (\tau)$	$\rho'(\tau)$
$\texttt{Forall ?X}_1\ldots\texttt{?X}_n\ (\tau)$	$\rho'(\tau)$
$\texttt{External}(\tau)$	$\rho'(\tau)$
$\tau_1\ \texttt{:-}\ \tau_2$	$\rho'(\tau_1)\ \texttt{:-}\ \rho'(\tau_2)$

Outline

Proving Semantics Preservation of Translations

- A sufficient condition for proving semantics preservation of PSOA transformations is given
- For each transformation step, the appropriate PSOA sublanguage is defined and semantics-preservation theorems are proved
- Based on that, semantics preservation of their compositions is proved for FOL- and LP-targeting normalizations with respect to appropriate sublanguages
- Finally, semantics preservation is proved
  (1) for the PSOA2TPTP translation with respect to
  the FOL semantics and
  (2) for the PSOA2Prolog translation with respect to
  the declarative semantics of logic programs

Outline

Outline

Decision Management Challenges

- Decision Management (DM) Community has been running Challenges about decision modeling problems since 2014
- The DM Challenge of March 2016 consisted of creating decision models from the structured text of English Port Clearance Rules, available online
- Independently given use case

Port Clearance Rules

Jacob Feldman *pointed us to this* *DM Challenge on The Game of Rules / Port Clearance Rules*

- Decide whether **a ship can enter** a Dutch port on a certain date
- Ten English rules inspired by the international Ship and Port Facility Security Code, originally developed by **Silvie Spreeuwenberg et al.** for "The Game of Rules"
- The English of each one of these independently given rules is **moderately controlled**, some having a structured 'if' part
- We **formalized** the rules in PSOA RuleML, added facts (data) directly in PSOA, **queried** result in PSOATransRun, and propose **generalized** decision models

Examples of Port Clearance Facts

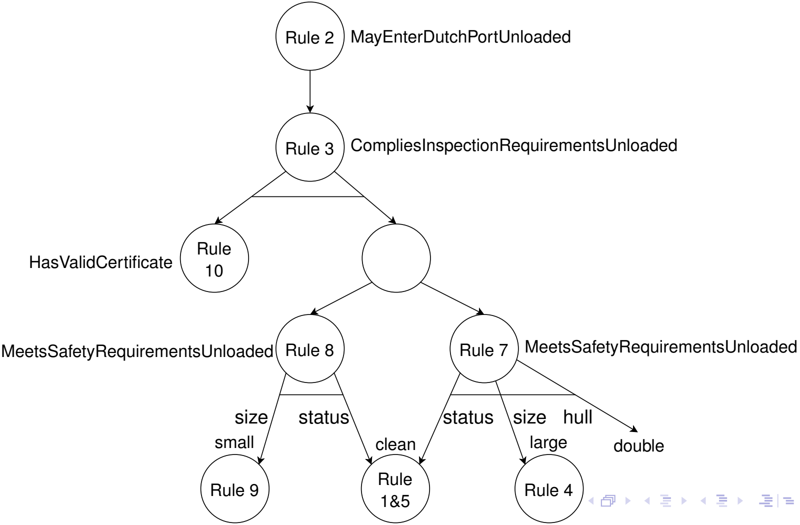
- Since the DM Challenge has introduced only ship rules, we have developed ship facts for systematic testing of rules using PSOATransRun[PSOA2Prolog,XSBProlog]
- Examples of ship facts

```
% Ship 1 - No, registry has expired
:ship1#:Ship(:registryExpirationDate->phys:date(2017 5 1)
             :totalLength->20
             :hold->:h1#:ShipHold(:residualCargoMeasurement->0.2
                                  :hull->:single))

% Ship 7 - Yes, hold clean and double-hulled
:ship7#:Ship(:registryExpirationDate->phys:date(2020 1 1)
             :totalLength->90
             :hold->:h7#:ShipHold(:residualCargoMeasurement->0.4
                                  :hull->:double))
```

Visualization of PSOA's Formal Decision Model (1)

- An object-relational And-Or DAG with rule names as nodes and conclusion predicates as side labels of nodes



Visualization of PSOA's Formal Decision Model (2)

- For the not side-labeled nodes, the root-class predicate `Top` is understood, while slot names are shown as labels of incoming arcs and top labels of the rule nodes (for the slot name `:hull` the filler `:double` does not require any further rule)
- The blank, unlabeled node represents the only 'Or' branch in this model, where Rules 8 and 7 are – operationally speaking – 'pre-invoked' via the conclusion predicate `:MeetsSafetyRequirementsUnloaded`, having conditions with a first conjunct immediately determining whether the slot `:size` is `:small` or `:large`, so that only either Rule 8 or Rule 7, respectively, can be 'fully invoked', causing *near-deterministic* behavior
- The model is object-relational in that the upper part running to the conclusions of Rules 8 and 7 involves unary relations applied to ships while the lower part involves frames with ship OIDs described by slots

Examples of Port Clearance Rules

8. A ship only meets the safety requirements for small unloaded ships if the ship complies with all of the following: a) the ship is categorized as small;
b) the hold of the ship is clean.
7. A ship only meets the safety requirements for large unloaded ships if the ship complies with all of the following: a) the ship is categorized as large;
b) the hold of the ship is clean; c) the hold of the ship is double hulled.

```
% Object-relational size-switched safety rules check status (small)
                                             or status and hull (large)

% Rule 8 (includes disjunct of original Rule 6)
Forall ?s ?h (
  :MeetsSafetyRequirementsUnloaded(?s) :-
    ?s#:Ship(:size->:small
             :hold->?h#:ShipHold(:status->:clean))
)

% Rule 7 (includes disjunct of original Rule 6)
Forall ?s ?h (
  :MeetsSafetyRequirementsUnloaded(?s) :-
    ?s#:Ship(:size->:large
             :hold->?h#:ShipHold(:status->:clean
                                 :hull->:double))
)
```

Queries that Answer DM Challenge Questions

- Queries for Port Clearance questions are ground, using top-level predicate `:MayEnterDutchPortUnloaded` applied to specific ship instances

```
:MayEnterDutchPortUnloaded(:ship1)
No
```

```
:MayEnterDutchPortUnloaded(:ship7)
Yes
```

- Generalized non-ground query can also be posed

```
:MayEnterDutchPortUnloaded(?w)
?w=<http://psoa.ruleml.org/usecases/PortClearance#ship14>
?w=<http://psoa.ruleml.org/usecases/PortClearance#ship2>
?w=<http://psoa.ruleml.org/usecases/PortClearance#ship12>
?w=<http://psoa.ruleml.org/usecases/PortClearance#ship7>
?w=<http://psoa.ruleml.org/usecases/PortClearance#ship4>
```

Outline

Overview of OfficeProspector

- Aims to help companies find office suites for their businesses
- Enriches office-suite data with public (e.g., geospatial) data sets in different modeling paradigms in order to enable user queries, e.g. for finding office suites based on information about building surroundings

Outline

Outline

Comparing PSOA RuleML 1.0 with Flora-2/F-logic and RIF-BLD

- PSOA RuleML allows more kinds of atoms and more flexibility in knowledge representation, e.g. dependent-slotted atoms
- PSOA RuleML supports objectification
- Flora-2/F-logic supports more schema-level formulas, e.g. for signature declarations. In PSOA and RIF-BLD, their usage as top-level KB formulas can be expressed as rules
- Expressivity
  - PSOA RuleML 1.0: Hornlog with existentials and equality
  - RIF-BLD: Hornlog with equality
  - Flora-2/F-logic: Hornlog with equality extended by various kinds of negations and extra-logicals

Outline

First Experiment

- A unit-test suite of test cases with a total of 54 KBs and 302 queries, covering all PSOA features that we have implemented
- Each test case consists of one KB, multiple queries, and user-provided expected answers to each query
- Answers to each query are obtained from PSOATransRun instantiations and compared to expected answers automatically
- Prolog instantiation passed all test cases, while the TPTP instantiation passed all test cases except the 11 tests that contain external built-ins, which cannot be expressed in TPTP-FOF

Second Experiment (1)

- Employs (rule-)Chain test cases for exploring performance differences between differently modeled KBs
- Four groups of test cases, each using one of the four major kinds of atoms: dependent-tuple, independent-tuple, dependent-slot, and independent-slot
- Each group has test cases distinguished by the number *k* of KB rules and each test case includes one KB and one query of the same dependency kind
- In the dependent-tuple group, each generated KB consists of the fact `_r0(_a1 _a2 _a3)` (short for `_r0(+[_a1 _a2 _a3])`) and *k* rules of the form ($i = 1, \ldots, k, i' = i - 1$):

```
Forall ?X1 ?X2 ?X3 (
  _ri(?X1 ?X2 ?X3) :- _ri'(?X1 ?X2 ?X3)
)
```

- The query is `_rk(?X ?Y ?Z)`, which has one answer `?X=_a1 ?Y=_a2 ?Z=_a3`

Second Experiment (2)

- Starting with $k = 0$ rules and incrementing in steps of 50 rules until reaching $k = 500$ rules, we generated 11 test cases for each group and measured the average query execution time
- For the dependent-tuple group, we also compared query execution time between static vs. static/dynamic objectification setups
- Results

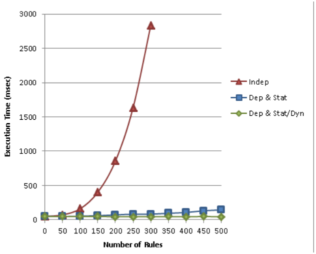


Figure: Execution time of 11 Tupled Chain test cases for Prolog instantiation.

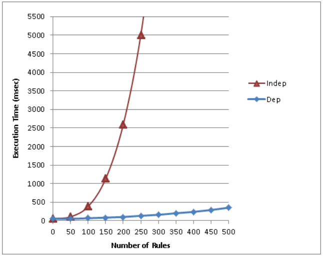


Figure: Execution time of 11 Slotted Chain test cases for Prolog instantiation.

Second Experiment (3)

Results show that

- For descriptors that need to be defined across different predicates via different rules, dependent modeling is more efficient
- For arguments that often go together (e.g., `_a1 _a2 _a3` in the Chain tests), tupled modeling is more efficient than slotted modeling
- For dependent-tuple group, static/dynamic objectification fully retains the efficiency of relational rules, hence is faster than static objectification
- For simple test cases, TPTP instantiation is faster because of the communication overhead in the Prolog instantiation
- For non-simple test cases, Prolog instantiation is faster

Third Experiment

- Employs NDChain test cases, which extend each tupled test case in the second experiment with one fact and k Non-Deterministic Chain rules
- Results are similar to the second experiment except that TPTP instantiation is faster

Outline

Major Contributions

- Revised the PSOA RuleML language, achieving Version 1.0
- Created an architecture for both interoperating and porting integrated object-relational knowledge
- Formalized and implemented translations, as well as proved their semantics preservation
- Combined the PSOA2TPTP and PSOA2Prolog translators with runtime engines into two PSOATransRun instantiations implementing PSOA
- Realized use cases and performed evaluation on test cases

Major Contributions – Expanded (1)

- Revised the PSOA RuleML language, achieving Version 1.0
  - Introduced independent vs. dependent distinction for descriptors and defined perspectivity dimension of atoms
  - Revised EBNF syntax and model-theoretic semantics
- Created an architecture for both interoperating and porting integrated object-relational knowledge
- Formalized and implemented translations, as well as proved their semantics preservation
  - Formalized and implemented PSOA transformation steps that can be reused for further PSOA-sourced translations
  - Following up on these, formalized and implemented the translations from PSOA sublanguages to TPTP and to Prolog
  - Formalized the translation from an N3 sublanguage, N3Basic, to PSOA and implemented the translation for N3 facts
  - Proved semantics preservation of transformation steps, conversions to TPTP and Prolog, as well as their compositions for appropriate sublanguages

Major Contributions – Expanded (2)

- Combined the PSOA2TPTP and PSOA2Prolog translators with runtime engines into two PSOATransRun instantiations implementing PSOA
  - Download: `http://wiki.ruleml.org/index.php/PSOA_RuleML#PSOATransRun`
- Realized use cases and performed evaluation on test cases
  - Applied PSOA and PSOATransRun to realistic use cases, Port Clearance Rules and OfficeProspector
  - Developed test cases and evaluated the PSOA language as well as PSOATransRun instantiations through three experiments

Future Work

- PSOA RuleML language can be orthogonally expanded to include relevant features (e.g., NAF) from other rule languages
- Extend PSOA2TPTP and PSOA2Prolog translations for PSOA with conclusion equalities
- Study and implement (inverse) translators from Prolog and TPTP to PSOA, and from PSOA to N3
- Further optimize the transformation steps, e.g. dynamic objectification
- Finalize and release schema specification of the PSOA RuleML 1.0/XML serialization syntax in Relax NG
- Extend translators for XML serialization of PSOA RuleML
- Proof-explanation facility could be added to PSOATransRun, providing visualization, presentation, and serialization formats for queries derived from facts

Backup Slides

Trade-offs for Realized Architecture

- Canonical-language-centered interoperation framework
  - Advantages
    - Fewer translators needed compared with all-to-all mappings
    - Allow reuse of modules among translators
- Translator-based portation framework
  - Advantage
    - Rapid prototyping and easier to maintain
  - Disadvantage
    - Harder to incorporate specific reasoning optimizations
- Modularized translator implementation
  - Advantage
    - Easier to test, maintain, and reuse
  - Disadvantage
    - Translators could be less efficient

N3Basic and its Translation to PSOA

- We defined a sublanguage of N3, N3Basic, that corresponds to a rule language extending RDF with (head-)existential rules
- Normalization of N3 transforms the input so that it consists of only triples or rules built on top of triples
- Conversion of normalized N3 to PSOA
  - Blank nodes are converted to local constants or existential variables according to their contexts
  - Each triple is converted to a single-slot frame or a membership in PSOA
  - N3 rules are converted to PSOA rules
- Translation currently implemented for facts (corresponding to the RDF/Turtle language)

N3Basic Exemplified

Example of Fact and Rule: Slot Introduction

```
:b1  :outer1 :canada;
     :inner  :usa;
    :outer2 :mexico.

@forAll B,Out,In.
{
  B a :betweenObj;
    :outer1 Out;
    :inner  In.
} => { Out :neighborSlot In. }
```

Revision of EBNF for Presentation Syntax

- Extends `CLAUSE`, `Implies`, and `HEAD` productions for closure under objectification and description transformations (explained later)
- Reflects use of
  - oidless and oidful psoa terms as `Atom`s in/as `FORMULA`s
  - oidful `Atom`s (for unnesting, leaving behind the OID term) as `TERM`s in `Atom`s and `Expr`essions
  - oidless psoa terms as `Expr`essions
- Refines all descriptors for distinction of Dependent vs. Independent tuples (`TUPLEDI`) and slots (`SLOTDI`)

Revised EBNF Grammar: Rule Language

Rule Language:

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

Revised EBNF Grammar: Condition Language

Condition Language:

```
FORMULA ::= 'And' '(' FORMULA* ')' |
            'Or' '(' FORMULA* ')' |
            'Exists' Var+ '(' FORMULA ')' |
            ATOMIC |
            'External' '(' Atom ')'
ATOMIC ::= Atom | Equal | Subclass
Atom ::= ATOMOIDLESS | ATOMOIDFUL
ATOMOIDLESS ::= PSOAOIDLESS
ATOMOIDFUL ::= PSOAOIDFUL
Equal ::= TERM '=' TERM
Subclass ::= TERM '##' TERM
PSOA ::= PSOAOIDLESS | PSOAOIDFUL
PSOAOIDLESS ::= TERM '(' (TERM* | TUPLEDI*) SLOTDI* ')'
PSOAOIDFUL ::= TERM '#' PSOAOIDLESS
TUPLEDI ::= ('+' | '-') '[' TERM* ']'
SLOTDI ::= TERM ('+>' | '->') TERM
TERM ::= Const | Var | ATOMOIDFUL | Expr | 'External' '(' Expr ')'
Expr ::= PSOAOIDLESS
...
```

OfficeProspector Partonomy-Taxonomy

```
Space
  Neighborhood
  |_  Building
      |_  Suite
          |_  Office
                ClosedOffice
                Cubicle
                OpenOffice
          |_  MeetingSpace
                ClosedMeetingSpace
                OpenMeetingSpace
          |_  Kitchen
                ClosedKitchen
                OpenKitchen
          |_  Reception
          |_  Room
                ClosedOffice
                ClosedMeetingSpace
                ClosedKitchen
          |_  OpenSpace
                ...
```

OfficeProspector Data Sets

- Internal data set: Generated data of offices and buildings
- External data sets
  - Relational data set containing information of Toronto's 140 neighborboods
  - Relational data set containing coordinates of addresses in WGS84 geodetic longitude-latitude spatial reference system used by the Global Positioning System
  - An object-centered data set in N3 syntax containing information of amenities in Toronto, extracted from LinkedGeoData

OfficeProspector Rules

- Object(-relational) integration rules using knowledge enrichment
- Object-centered vocabulary-extension rules
- Other object(-relational) rules, e.g. for matching constraints in queries and for converting measures

OfficeProspector Example Query

Find any suite ?s that satisfies the following:
1) the monthly rent of ?s is at most 5000 CAD;
2) the HVAC system of ?s has a rating of at least basic;
3) ?s has Internet.
4) ?s is a part of a building ?b that satisfies:
4.1) the distance to the nearest public transport of ?b is at most 1000 meters;
4.2) the completion year of ?b is at least 1985

```
And(
  ?s#:Suite(
    :constrain(:monthlyRent)->:atmost(:measure(5000 :cad))
    :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)
    :utility->:internet
    :partOf->?b
  )
  ?b#:Building(
    :constrain(:publicTransAccessDistance)->
                                    :atmost(:measure(1000 :m))
    :constrain(:yearBuilt)->:atleast(1985)
  )
)
```