PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog

Gen Zou, Harold Boley

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
{gen.zou, harold.boley}[AT]unb.ca

Abstract. PSOA2Prolog consists of a multi-step source-to-source normalizer followed by a mapper to a pure (Horn) subset of ISO Prolog. We show the semantics preservation of the steps. Composing PSOA2Prolog and XSB Prolog, a fast Prolog engine, we achieved a novel instantiation of our PSOATransRun framework. We evaluated this interoperation and implementation technique with a suite of 30 test cases using 90 queries, and found a considerable speed-up compared to our earlier instantiation.

1 Introduction

In the Semantic Web and AI, the relational and object-centered modeling paradigms have been widely used for representing knowledge. The relational paradigm (e.g., classical logic and relational databases) models entity relationships using predicates applied to positional arguments, while the object-centered paradigm (e.g., RDF and N3) uses *frames* to model each entity using a globally unique Object IDentifier (OID) typed by a class and described by an unordered collection of slotted (attribute-value) arguments. To facilitate interoperation between the two paradigms, e.g. for expressing the mapping between frames and relational database schemas in rule-based data access, combined object-relational paradigms have been studied. F-logic [1, 2] and RIF-BLD [3] employ a heterogeneous approach which allows the mixed use of both relations and frames. In contrast, the Web rule language PSOA RuleML [4] employs a homogeneous approach by generalizing relations and frames into **p**ositional-**s**lotted **o**bject-**a**pplicative terms, which permit a relation application to have an OID – typed by the relation – and, orthogonally, to have positional or slotted arguments.

In order to reuse knowledge bases (KBs) and implementations of different rule languages, translators among them have been developed, including one from an F-logic-based language to Prolog [5, 6], and one from the object-centered language KM into answer set programs [7]. To create a major interoperation path for the homogeneous object-relational PSOA RuleML, we developed a translator PSOA2Prolog from PSOA RuleML to a subset of the relational ISO Prolog [8,9], a logic programming standard with subsets supported by many fast engines. The translator supports KBs and queries employing all major PSOA features but restricting the use of equality to external-function evaluation. PSOA2Prolog

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

By composing PSOA2Prolog and XSB Prolog, a fast Prolog engine for an ISO Prolog superset, we realized the PSOATransRun[PSOA2Prolog,XSBProlog][1] instantiation of our PSOATransRun framework. The implementation performs query answering in PSOA RuleML by translating a user-provided PSOA presentation syntax (PSOA/PS) KB and queries into Prolog, executing the queries in the Prolog engine, and translating the results back to PSOA/PS. Within its realm of Horn logic, the new instantiation supports more PSOA features than our earlier instantiation PSOATransRun[PSOA2TPTP,VampirePrime] [10], and our empirical evaluation shows a considerable speed-up on large test cases.

The rest of the paper is organized as follows. Section 2 reviews the basics of PSOA RuleML and ISO Prolog. Sections 3 and 4 explain the techniques employed by the normalizer and the mapper components of PSOA2Prolog. Section 5 explains the realization of PSOATransRun[PSOA2Prolog,XSBProlog], and compares it with PSOATransRun[PSOA2TPTP,VampirePrime] through test cases. Section 6 concludes the paper and discusses future work.

2 Background on the Source and Target Languages

In this section we introduce the basics of the source language PSOA RuleML and the target language ISO Prolog, of the PSOA2Prolog translator.

2.1 PSOA RuleML

PSOA RuleML [4] is an object-relational Web rule language that integrates relations and frames into positional-slotted, object-applicative (psoa)[2] terms, which have the general form

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

Here, an object is identified by an Object IDentifier (OID) `o` and described by (1) a class membership `o#f`, (2) a set of tupled arguments $\mathtt{[t_{i,1}\ \ldots\ t_{i,n_i}]}$, $i = 1, \ldots, m$, each being a sequence of terms, and (3) a set of slotted arguments $\mathtt{p_j\text{->}v_j}$, $j = 1, \ldots, k$ representing attribute-value pairs. The OID as well as tuples and, orthogonally, slots in a psoa term are all optional. A psoa term can express untyped objects by treating them as being typed by the root class `f=Top`. For an *anonymous psoa term*, without a 'user' OID, *objectification* will introduce a 'system' OID as explained in Section 3.1. For the most often used special case of

[1] `http://psoa.ruleml.org/transrun/0.7/local/`

[2] We use the upper-cased "PSOA" as a qualifier for the language and the lower-cased "psoa" for its terms.

single-tuple psoa terms (m=1), the square brackets enclosing the tuple can be omitted.

PSOA RuleML constants have the form `"literal"^^symspace`, where `literal` is a sequence of Unicode characters and `symspace` is a symbol space identifier. Six kinds of shortcuts for constants are defined in [11], including numbers, strings, Internationalized Resource Identifiers (IRIs), and '_'-prefixed *local constants* (e.g., `_a`) whose `symspace` is specialized to `rif:local`. `Top` is a shortcut for the root class. PSOA RuleML variables are '`?`'-prefixed sequences.

A *base term* can be a constant, a variable, an anonymous psoa term, or an external term of the form `External(t)`, where `t` is an anonymous psoa term. An atomic formula is a psoa term with or without an OID, a subclass term `c1 ## c2`, an equality term `t1=t2`, or an external term. Complex formulas are constructed using the Horn-like subset of first-order logic (FOL), e.g. conjunctions and rule implications.

The semantics of PSOA RuleML is defined through semantic structures [4]. A semantic structure $\mathcal{I}$ is a tuple $<\boldsymbol{TV}, \boldsymbol{DTS}, \boldsymbol{D}, \boldsymbol{D}_{\text{ind}}, \boldsymbol{D}_{\text{func}}, \boldsymbol{I}_{\text{C}}, \boldsymbol{I}_{\text{V}}, \boldsymbol{I}_{\text{psoa}}, \boldsymbol{I}_{\text{sub}}, \boldsymbol{I}_{=}, \boldsymbol{I}_{\text{external}}, \boldsymbol{I}_{\text{truth}}>$. Here $\boldsymbol{D}$ is a non-empty set called the domain of $\mathcal{I}$. $\boldsymbol{D}_{\text{ind}}$ and $\boldsymbol{D}_{\text{func}}$ are subsets of $\boldsymbol{D}$ for interpreting individuals and functions, respectively. $\boldsymbol{I}_{\text{C}}$, $\boldsymbol{I}_{\text{V}}$, $\boldsymbol{I}_{\text{psoa}}$ interprets constants, variables, and psoa terms. $\boldsymbol{I}_{\text{sub}}$, $\boldsymbol{I}_{=}$, $\boldsymbol{I}_{\text{external}}$ interprets subclass, equality and external terms. $\boldsymbol{TV} = \{\mathbf{t}, \mathbf{f}\}$ is the set of truth values. $\boldsymbol{I}_{\text{truth}}$ maps domain elements to $\boldsymbol{TV}$, allowing HiLog-like generality. Truth evaluation for well-formed formulas is determined by an evaluation function $TVal_{\mathcal{I}}$. A semantic structure $\mathcal{I}$ is called a *model* of a KB ϕ if $TVal_{\mathcal{I}}(\phi) = \mathbf{t}$, denoted by $\mathcal{I} \models \phi$. A PSOA KB ϕ is said to *entail* a formula ψ, denoted by $\phi \models \psi$, if for every model $\mathcal{I}$ of ϕ, $\mathcal{I} \models \psi$ holds.

A more detailed introduction, with many examples leading to the semantics, can be found in [12].

Startup Example The KB in Fig. 1 demonstrates key features of PSOA RuleML via a startup company scenario, serving as the paper's running example.

The example contains four facts, a rule, and a subclass ('`##`') formula. The rule derives an anonymous psoa term (in [12] called a "relpairship") with class `_startup` from: (1) a `_cofounders` relationship between the CEO and the CTO; (2) a `_hire` relationship between the CEO and an employee; (3) two `_equity` relationships describing the equity shares of the CEO and the CTO; (4) external calls ensuring that the sum of the equity percentages of the CEO and the CTO is not greater than 100. The `_startup` term has one tuple for the `_ceo` and the `_cto`, as well as one slot for one or more `_employee`s (PSOA slots can be multi-valued). The '`##`' formula states that `_startup` is a subclass of `_company`.

Leaving the topic of normalization to Section 3, the rule can be applied to the four facts, deriving the anonymous psoa term `_startup(_Ernie _Tony _employee->_Kate)`. Combined with the subclass formula, a `_company` psoa term `_company(_Ernie _Tony _employee->_Kate)` with the same tuple and slot can be derived. This entails all psoa terms omitting the tuple or the slot, namely `_company(_employee->_Kate)` and `_company(_Ernie _Tony)`.

```
Document (
  Group (
    Forall ?X ?Y ?Z ?EX ?EY (
      _startup(?X ?Y _employee->?Z) :-
         And(_cofounders(?X ?Y) _hire(?X ?Z)
             _equity(?X ?EX) _equity(?Y ?EY)
             External(
               pred:numeric-less-than-or-equal(
                 External(func:numeric-add(?EX ?EY)) 100)))
    )

    _cofounders(_Ernie _Tony)  _hire(_Ernie _Kate)
    _equity(_Ernie 50)         _equity(_Tony 30)
    _startup##_company
  )
)
```

Fig. 1. KB using psoa-term rule with implicit OID typed as startup company.

Amongst the many possible queries over the KB in Fig. 1, our running query will be

```
_company(?X ?Y)
```

It asks for the positional arguments `?X` and `?Y` of the `_company` psoa term, omitting the `_employee` slot. The desired answer is `?X=_Ernie ?Y=_Tony`.

2.2 ISO Prolog

Prolog is a widely used logic programming language. ISO Prolog [8] is the international standard of Prolog. In this paper we focus on the Horn subset of ISO Prolog, which excludes procedural features like negation-as-failure. The syntax of ISO Prolog is built on top of *terms*: A term can be a *variable*, a *number*, an *atosym* (atomic symbol)[3], or a *compound term* of the form $p(t_1, \ldots, t_n)$, where p is an atosym and $t_1, ..., t_n$ are terms. A variable is basically a sequence of letters or digits starting with an upper-case letter, e.g. `X1`. An atosym, for a predicate or a function (including a nullary function, i.e. an individual), starts with a lower-case letter, e.g. `a1`, or is a single-quoted sequence of arbitrary characters, e.g. `'http://abc'`. An ISO Prolog *predication* is an atosym (for a nullary predicate) or a compound term. A *clause* is either a *fact* in the form of a predication or a *rule* of the form `Head :- Body`. The `Head` is a predication, while the `Body` can be a predication, a conjunction of bodies `(Body , ... , Body)`, or a disjunction or bodies `(Body ; ... ; Body)`. All variables in a clause are considered to be in the scope of universal quantifiers preceding the entire clause. A *logic program* consists of a set of clauses.

[3] To avoid confusion with logical atoms, we use "atosym" to refer to symbols that ISO Prolog calls "atoms".

3 Normalization of the PSOA Source in Five Steps

The first translation phase is to transform the input PSOA *KB* and *queries* into a normalized form such that all clauses are objectified, existential-free, and contain only elementary formulas which cannot be split into equivalent subformulas. For PSOA KBs, there are five steps applied in sequential order: (1) Objectification; (2) Skolemization; (3) slotribution and tupribution; (4) flattening of external function applications; (5) splitting of rules with conjunctive heads. For PSOA queries, only steps (1), (3), and (4) are needed. The semantics preservation of the normalization steps will be indicated in English for the easier steps (1), (3), and (4), while detailed correctness proofs will be given for the more involved steps (2) and (5).

3.1 Objectification

In PSOA RuleML, an atomic psoa formula without an OID is regarded as having an implicit OID, which can be made explicit through *objectification*. We employ a modified version of the objectification technique introduced in [4]. For an OID-less psoa formula `p(...)` in a KB clause, we define three cases:

- If it is a ground fact, it is objectified into `_`*`i`*`#p(...)`, where `_`*`i`* is a newly generated local constant name in the KB.
- If it is a non-ground fact or an atomic formula in a rule conclusion, it is objectified into `Exists` *`?j`* `(`*`?j`*`#p(...))`.
- If it is an atomic formula in a rule premise, it is objectified into *`?j`*`#p(...)`, where *`?j`* is a new variable in the universal scope of the enclosing rule. In [4], an anonymous variable '`?`' is used as the OID in this case. However, the next Skolemization step, which will be explained in Section 3.2, requires all universally quantified variables in an existential rule to be named, since they will become arguments of Skolem functions. Thus, we further parse the stand-alone '?'s into explicitly named variables *`?j`* quantified in the top-level universal scope in our objectification step.

We define the objectification of an OID-less psoa formula in a (conjunctive) query as `Exists` *`?j`* `(`*`?j`*`#p(...))`. Here, the new variable *`?j`* is not a free query variable but encapsulated in an existential scope, so that the bindings of *`?j`* will not be returned in contrast to those of user-provided free query variables.

3.2 Skolemization

After objectification, existentially quantified formulas may occur in rule conclusions. Since such conclusion existentials are not allowed in logic programming languages such as Prolog, we employ a specialized FOL Skolemization [13] to eliminate them. Our approach is adapted for PSOA RuleML clauses, whose universals are already in prenex form and whose existentials are confined to (conjunctive) conclusions, hence does not require preprocessing of formulas. We replace each formula `Exists ?X (`σ`)` in a clause (specifically, in a rule conclusion or a fact) with σ[`?X/_skolem`k`(?`$\mathtt{v}_1$ `... ?`$\mathtt{v}_m$`)`], 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. For each existentially quantified variable in the KB, a fresh Skolem function name `_skolem`k is chosen from the first name in the sequence `_skolem1, _skolem2, ...` that has not yet been used.

Skolemization Example The following PSOA rule

```
Forall ?v ( Exists ?1 (?1#_c1(?v)) :- _o#_c2(?v) )
```

has an existentially quantified formula in the rule conclusion. Skolemization removes the existential quantifier and replaces the variable ?1 with `_skolem1(?v)`, which is a unary Skolem function `_skolem1` applied to the variable `?v` in the clause's quantifier prefix, yielding

```
Forall ?v ( _skolem1(?v)#_c1(?v) :- _o#_c2(?v) )
```

Next we prove the correctness of our Skolemization. In the proof, we use $\mathcal{I}$[`?v`$_1$ ... `?v`$_m$] to denote the set of semantic structures that coincide with $\mathcal{I}$ on everything but the variables `?v`$_1$ ... `?v`$_m$, $t^{\mathcal{I}}$ to denote the interpretation of a base term t in $\mathcal{I}$, and $SK(\psi)$ to denote the Skolem form of a formula ψ.

Lemma 1 *Let ϕ be a formula and $\mathcal{I}$,$\mathcal{I}'$ be semantic structures. Then (1) if $\mathcal{I}' \models SK(\phi)$ then $\mathcal{I}' \models \phi$; (2) if $\mathcal{I} \models \phi$, there exists $\mathcal{I}'$ such that $\mathcal{I}' \models SK(\phi)$ and $\mathcal{I}'$ coincides with $\mathcal{I}$ on everything but the interpretation of Skolem functions.*

Proof. If ϕ does not have an existential (not necessarily proper) subformula then $SK(\phi) = \phi$ and the lemma holds trivially. Next we prove (1) and (2) by induction over every subformula σ of ϕ that contain an existential subformula. Proof for (1):

- $\sigma =$ `Exists ?X (`σ_1`)`
  If $TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}$, there exists a semantic structure $\mathcal{I}^*$ in $\mathcal{I}'$[`?X`] which interprets `?X` as `_skolem`k`(?v`$_1$ ... `?v`$_m$`)`$^{\mathcal{I}'}$, and

  $$TVal_{\mathcal{I}^*}(\sigma_1) = TVal_{\mathcal{I}'}(\sigma_1[\texttt{?X}/\texttt{\_skolem}k\texttt{(?v}_1\ \ldots\ \texttt{?v}_m\texttt{)}]) = TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}.$$

  Hence $TVal_{\mathcal{I}'}(\sigma) = \mathbf{t}$.
- $\sigma =$ `And(`σ_1 ... σ_n`)`
  Since $SK(\sigma) =$ `And(`$SK(\sigma_1)$ ... $SK(\sigma_n)$`)`, if $TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}$, then $TVal_{\mathcal{I}'}(SK(\sigma_i)) = \mathbf{t}$, $i = 1, \ldots, n$. If σ_i has an existential subformula, then by induction hypothesis $TVal_{\mathcal{I}'}(\sigma_i) = \mathbf{t}$. Otherwise, $TVal_{\mathcal{I}'}(\sigma_i) = TVal_{\mathcal{I}'}(SK(\sigma_i)) = \mathbf{t}$. Thus, $TVal_{\mathcal{I}'}(\sigma_i) = \mathbf{t}, i = 1, \ldots, n$, and $TVal_{\mathcal{I}'}(\sigma) = \mathbf{t}$.
- $\sigma = \sigma_1$ `:-` σ_2
  Since existentials occur only in rule conclusions, $SK(\sigma) = SK(\sigma_1)$ `:-` σ_2. So

  $$\begin{aligned} TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t} &\Rightarrow TVal_{\mathcal{I}'}(SK(\sigma_1)) = \mathbf{t} \text{ or } TVal_{\mathcal{I}'}(\sigma_2) = \mathbf{f} \\ &\Rightarrow TVal_{\mathcal{I}'}(\sigma_1) = \mathbf{t} \text{ or } TVal_{\mathcal{I}'}(\sigma_2) = \mathbf{f} \\ &\Rightarrow TVal_{\mathcal{I}'}(\sigma) = \mathbf{t} \end{aligned}$$

- $\sigma =$ `Forall ?v1 ... ?vm (`σ_1`)`
  Since $SK(\sigma) =$ `Forall ?v1 ... ?vm (`$SK(\sigma_1)$`)`, we have

$$\begin{aligned} TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t} &\Rightarrow \text{for all } \mathcal{I}^* \in \mathcal{I}'[\texttt{?v}_1 \ldots \texttt{?v}_m], TVal_{\mathcal{I}^*}(SK(\sigma_1)) = \mathbf{t} \\ &\Rightarrow \text{for all } \mathcal{I}^* \in \mathcal{I}'[\texttt{?v}_1 \ldots \texttt{?v}_m], TVal_{\mathcal{I}^*}(\sigma_1) = \mathbf{t} \\ &\Rightarrow TVal_{\mathcal{I}'}(\sigma) = \mathbf{t} \end{aligned}$$

- $\sigma =$ `Group(`σ_1 `...` σ_n`)`
  The semantics of a group formula is the same as a conjunction. So the same proof can be applied.

Proof for (2): For each semantic structure $\mathcal{I}$, we construct $\mathcal{I}'$ by adding the interpretation for all Skolem functions in ϕ. Assume `Exists ?X (`σ_1`)` is a subformula of ϕ, and `?X` is replaced with `_skolem`k`(?v1 ... ?vm)` during Skolemization. For elements $x_1, \ldots, x_n$ in the domain of $\mathcal{I}$, if there exists one $\mathcal{I}^*$ such that $TVal_{\mathcal{I}^*}(\sigma_1) = \mathbf{t}$ and $\texttt{?v}_i^{\mathcal{I}'} = x_i, i = 1, \ldots, n$, then we define $\texttt{\_skolem}k^{\mathcal{I}'}(x_1, \ldots, x_n)$ to be $\texttt{?X}^{\mathcal{I}^*}$. Next we prove if $TVal_{\mathcal{I}}(\sigma) = \mathbf{t}$, then $TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}$ by induction over all σ that contain an existential subformula.

- $\sigma =$ `Exists ?X (`σ_1`)`
  In this case $TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}$ follows by the definition of $\mathcal{I}'$.
- $\sigma =$ `And(`σ_1 `...` σ_n`)`
  If $TVal_{\mathcal{I}}(\sigma) = \mathbf{t}$, then $TVal_{\mathcal{I}}(\sigma_i) = \mathbf{t}$ for all $i = 1, \ldots, n$. If σ_i has an existential subformula, then by induction hypothesis $TVal_{\mathcal{I}'}(SK(\sigma_i)) = \mathbf{t}$. Otherwise $SK(\sigma_i) = \sigma_i$, and $TVal_{\mathcal{I}'}(SK(\sigma_i)) = TVal_{\mathcal{I}}(\sigma_i) = \mathbf{t}$. Hence $TVal_{\mathcal{I}'}(SK(\sigma_i)) = \mathbf{t}$ holds for all $i = 1, \ldots, n$, and $TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t}$.
- $\sigma = \sigma_1$ `:-` σ_2

$$\begin{aligned} TVal_{\mathcal{I}}(\sigma) = \mathbf{t} &\Rightarrow TVal_{\mathcal{I}}(\sigma_1) = \mathbf{t} \text{ or } TVal_{\mathcal{I}}(\sigma_2) = \mathbf{f} \\ &\Rightarrow TVal_{\mathcal{I}'}(SK(\sigma_1)) = \mathbf{t} \text{ or } TVal_{\mathcal{I}'}(\sigma_2) = \mathbf{f} \\ &\Rightarrow TVal_{\mathcal{I}'}(SK(\sigma)) = \mathbf{t} \end{aligned}$$

- $\sigma =$ `Forall ?v1 ... ?vm (`σ_1`)`

$$\begin{aligned} TVal_{\mathcal{I}}(\sigma) = \mathbf{t} &\Rightarrow \text{for all } \mathcal{J} \in \mathcal{I}[\texttt{?v}_1 \ldots \texttt{?v}_m], TVal_{\mathcal{J}}(\sigma_1) = \mathbf{t} \\ &\Rightarrow \text{for all } \mathcal{J}' \in \mathcal{I}'[\texttt{?v}_1 \ldots \texttt{?v}_m], TVal_{\mathcal{J}'}(SK(\sigma_1)) = \mathbf{t} \\ &\Rightarrow TVal_{\mathcal{I}'}(\sigma) = \mathbf{t} \end{aligned}$$

- $\sigma =$ `Group(`ψ_1 `...` ψ_n`)`
  The proof for conjunction formulas can be reused.

Theorem 1 *Let Γ be a PSOA KB, and τ be a clause that does not contain Skolem functions used in $SK(\Gamma)$. Then $\Gamma \models \tau$ if and only if $SK(\Gamma) \models \tau$.*

Proof. (Only if) If $\Gamma \models \tau$, then by part (1) of Lemma 1, for every model $\mathcal{I}'$ of $SK(\Gamma)$, $TVal_{\mathcal{I}'}(SK(\Gamma)) = TVal_{\mathcal{I}}(\Gamma) = TVal_{\mathcal{I}}(\tau) = \mathbf{t}$. So $SK(\Gamma) \models \tau$ holds.

(If) We prove the contrapositive statement, i.e. if $\Gamma \not\models \tau$ then $SK(\Gamma) \not\models \tau$. If $\Gamma \not\models \tau$, there exists a semantic structure $\mathcal{I}$ such that $TVal_{\mathcal{I}}(\Gamma) = \mathbf{t}$ and $TVal_{\mathcal{I}}(\tau) = \mathbf{f}$. By part (2) of Lemma 1, there exists $\mathcal{I}'$ such that $TVal_{\mathcal{I}'}(SK(\Gamma)) = \mathbf{t}$ and $\mathcal{I}'$ coincides with $\mathcal{I}'$ on everything but the Skolem functions, which do not occur in τ. Hence, $TVal_{\mathcal{I}'}(\tau) = TVal_{\mathcal{I}}(\tau) = \mathbf{f}$, making $\mathcal{I}'$ a counter-model of $SK(\Gamma) \models \tau$. Thus $SK(\Gamma) \not\models \tau$ and the statement holds.

3.3 Slotribution/Tupribution

The truth value of a psoa formula is defined via slotribution and tupribution [4]:

- $TVal_{\mathcal{I}}$(o#f([$t_{1,1}$ ... t_{1,n_1}] ... [$t_{m,1}$ ... t_{m,n_m}] p_1->v_1 ...p_k->v_k)) = **t** if and only if
  $TVal_{\mathcal{I}}$(o#f) =
  $TVal_{\mathcal{I}}$(o#Top([$t_{1,1}$ ... t_{1,n_1}])) = ... = $TVal_{\mathcal{I}}$(o#Top([$t_{m,1}$ ... t_{m,n_m}])) =
  $TVal_{\mathcal{I}}$(o#Top(p_1->v_1)) = ... = $TVal_{\mathcal{I}}$(o#Top(p_k->v_k)) = **t**.

According to the semantics of psoa formulas, we can rewrite each psoa formula o#f([$t_{1,1}$ ... t_{1,n_1}] ... [$t_{m,1}$ ... t_{m,n_m}] p_1->v_1 ... p_k->v_k), containing m tuples and k slots, into an equivalent conjunction of $1 + m + k$ subformulas, including 1 class membership formula, m single-tuple formulas and k single-slot formulas:

```
And(o#f
    o#Top(t1,1 ... t1,n1) ... o#Top(tm,1 ... tm,nm)
    o#Top(p1->v1) ... o#Top(pk->vk))
```

For the special case where f=Top but $m + k > 0$, we omit the tautological Top-typed class membership of the form o#Top from the conjunction for efficiency.[4] The correctness of this step follows directly from the definition. This step is central to both PSOA2TPTP [14] and PSOA2Prolog.

3.4 Flattening Nested External Function Applications

A PSOA function application can use a constructor function with no definition, a user-defined function specified by equalities in the KB, or an externally defined function such as an arithmetic built-in. The latter two types of applications evaluate an 'interpreted' function to a returned value. Flattening is employed to create a conjunction extracting an embedded interpreted function application as a separate equality. This version of PSOA2Prolog supports only the use of equalities of the form ?X=External(f(...)), equating a variable and an external function application for the evaluation of a function call, in preparation of the mapping to the Prolog is-primitive in Section 4.2.

In the flattening step, 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. If ψ is in a KB rule, then the variable ?*i* becomes a universal variable of the rule; otherwise ?*i* is encapsulated

[4] For $m+k = 0$, the description-less o#Top does not undergo slotribution/tupribution.

in a top-level existential scope in a query. This step is repeated for every clause in the KB until there are no more nested function applications. The correctness of this step follows from back-substitution of the embedded application for the variable `?i`.

3.5 Splitting Rules with Conjunctive Conclusions

A rule with a conjunction in the conclusion

$$\texttt{Forall ?v}_1 \texttt{ ... ?v}_\texttt{m} \texttt{ (And(}\varphi_1 \texttt{ ... } \varphi_n\texttt{) :- } \varphi'\texttt{)}$$

can be split into n rules

$$\begin{array}{c}\texttt{Forall ?v}_1 \texttt{ ... ?v}_\texttt{m} \texttt{ (}\varphi_1 \texttt{ :- } \varphi'\texttt{),} \\ \texttt{...} \\ \texttt{Forall ?v}_1 \texttt{ ... ?v}_\texttt{m} \texttt{ (}\varphi_n \texttt{ :- } \varphi'\texttt{)}\end{array}$$

with each conjunct becoming the conclusion of one rule, and with the premise and the quantification copied unchanged.

Lemma 2 *Let ϕ be the given rule, $\phi_1, ..., \phi_n$ be the split rules, and $\mathcal{I}$ be a semantic structure, then $\mathcal{I} \models \phi$ if and only if $\mathcal{I} \models \phi_i, i = 1, \ldots, n$.*

Proof. Let ϕ be of the form $\texttt{Forall ?v}_1 \texttt{ ... ?v}_\texttt{m} \texttt{ (And(}\varphi_1 \texttt{ ... } \varphi_n\texttt{) :- } \varphi'\texttt{)}$. If $\mathcal{I} \models \phi$, then for all $\mathcal{I}^* \in \mathcal{I}[\texttt{?v}_1 \ldots \texttt{?v}_\texttt{m}]$, we have

$$\begin{array}{ll} & TVal_{\mathcal{I}^*}(\texttt{And(}\varphi_1 \texttt{ ... } \varphi_n\texttt{) :- } \varphi'\texttt{)} = \mathbf{t} \\ \iff & TVal_{\mathcal{I}^*}(\texttt{And(}\varphi_1 \texttt{ ... } \varphi_n\texttt{)}) = \mathbf{t} \text{ or } TVal_{\mathcal{I}^*}(\varphi') = \mathbf{f} \\ \iff & (TVal_{\mathcal{I}^*}(\varphi_1) = \ldots = TVal_{\mathcal{I}^*}(\varphi_n) = \mathbf{t}) \text{ or } TVal_{\mathcal{I}^*}(\varphi') = \mathbf{f} \\ \iff & (TVal_{\mathcal{I}^*}(\varphi_1) = \mathbf{t} \text{ or } TVal_{\mathcal{I}^*}(\varphi') = \mathbf{f}) \text{ and } \ldots \text{ and} \\ & (TVal_{\mathcal{I}^*}(\varphi_n) = \mathbf{t} \text{ or } TVal_{\mathcal{I}^*}(\varphi') = \mathbf{f}) \\ \iff & TVal_{\mathcal{I}^*}(\varphi_1 \texttt{ :- } \varphi') = \ldots = TVal_{\mathcal{I}^*}(\varphi_n \texttt{ :- } \varphi') = \mathbf{t} \end{array}$$

Hence $\mathcal{I} \models \phi_i, i = 1, \ldots, n$.

Theorem 2 *If Γ is a PSOA KB and Γ' is the KB after performing rule splitting in Γ. Then for any formula τ, $\Gamma \models \tau$ if and only if $\Gamma' \models \tau$.*

Proof. By extending Lemma 2 for KBs, we have $TVal_{\mathcal{I}}(\Gamma) = TVal_{\mathcal{I}}(\Gamma')$. If $\Gamma' \models \tau$, then for all $\mathcal{I}$ such that $\mathcal{I} \models \Gamma$, $TVal_{\mathcal{I}}(\Gamma') = TVal_{\mathcal{I}}(\tau) = \mathbf{t}$, so $\Gamma \models \tau$. The "only if" part can be proved similarly.

3.6 Normalizing the Startup Example

In this subsection we demonstrate the normalization steps with the Startup Example given in Section 2.1.

The objectification step introduces OIDs `_1,...,_4` for the ground facts. It also introduces an existentially quantified variable `?1` for the anonymous psoa term in the rule conclusion while introducing four universally quantified variables `?2, ..., ?5` for the OID-less relations in the rule premise. Moreover, the query is objectified using a variable `?1`, which is encapsulated in an existential scope to avoid being treated as a free query variable.

Objectified KB:

```
Document (
  Group (
    Forall ?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5 (
      Exists ?1 (
        ?1#_startup(?X ?Y _employee->?Z)) :-
          And(?2#_cofounders(?X ?Y) ?3#_hire(?X ?Z)
              ?4#_equity(?X ?EX) ?5#_equity(?Y ?EY)
              External(
                pred:numeric-less-than-or-equal(
                  External(func:numeric-add(?EX ?EY)) 100)))
    )
    _1#_cofounders(_Ernie _Tony)  _2#_hire(_Ernie _Kate)
    _3#_equity(_Ernie 50)         _4#_equity(_Tony 30)
    _startup##_company
  )
)
```

Objectified Query:

```
Exists ?1 (?1#_company(?X ?Y))
```

After objectification, the Skolemization step eliminates the existential in the rule conclusion and replaces the variable `?1` with a Skolem function application, giving `_skolem1(?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5)#_startup(?X ?Y _employee->?Z)`.[5]

Next, slotribution and tupribution transform each psoa formula into a conjunction (possibly merged into a surrounding `Group`/`And`). The result, which is shown on the next page, contains a rule with a conjunctive conclusion.

The application of the external predicate `pred:numeric-less-than-or-equal` is then flattened with the argument `External(func:numeric-add(...))` being replaced by a fresh universal variable `?6`, obtaining the conjunction

```
And(?6=External(func:numeric-add(?EX ?EY))
    External(pred:numeric-less-than-or-equal(?6 100))
```

Finally, the conjunction in the rule conclusion is split to obtain the normalized KB as shown on the next page. The four output rules have the same premise and the same argument list for the Skolem function. The normalized query after rule splitting is the same as the slotributed/tupributed version shown above.

The example can be easily refined, e.g. with additional conclusion slots such as `_founderEquity`, which would store the already computed sum of the equities held by the CEO and the CTO.

[5] Objectification generates an existential OID for each derived `_startup`. Skolemization then replaces that OID by a system function application dependent on all universal variables, including `?Z`, thus denoting different OIDs for startups with the same CEO/CTO pair and different employees. If a user wants all startups with the same CEO and CTO to have the same OID, he/she can specify the OID of the original rule conclusion explicitly via an application, `_startupid(?X ?Y)`, of a fresh function name, `_startupid`, to the CEO, `?X`, and the CTO, `?Y`, denoting the OID dependent on them but not on any employee, `?Z`.

Slotributed/Tupributed KB:

```
Document(
  Group (
    Forall ?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5 (
         And(_skolem1(?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5)#_startup
             _skolem1(...)#Top(?X ?Y)
             _skolem1(...)#Top(_employee->?Z)) :-
         And(?2#_cofounders ?2#Top(?X ?Y)  ?3#_hire ?3#Top(?X ?Z)
             ?4#_equity  ?4#Top(?X ?EX) ?5#_equity ?5#Top(?Y ?EY)
             External(
               pred:numeric-less-than-or-equal(
                 External(func:numeric-add(?EX ?EY)) 100)))
    )

    _1#_cofounders _1#Top(_Ernie _Tony) _2#_hire   _2#Top(_Ernie _Kate)
    _3#_equity     _3#Top(_Ernie 50)    _4#_equity _4#Top(_Tony 30)
    _startup##_company
  )
)
```

Slotributed/Tupributed Query:

```
Exists ?1 (And(?1#_company ?1#Top(?X ?Y)))
```

Rule-split KB:

```
Document(
  Group (
    Forall ?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5 ?6 (
      _skolem1(?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5)#_startup :-
        And(?2#_cofounders ?2#Top(?X ?Y)  ?3#_hire ?3#Top(?X ?Z)
            ?4#_equity  ?4#Top(?X ?EX) ?5#_equity ?5#Top(?Y ?EY)
            And(?6=External(func:numeric-add(?EX ?EY))
                External(pred:numeric-less-than-or-equal(?6 100)))
    )

    Forall ?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5 ?6 (
      _skolem1(...)#Top(?X ?Y) :- And(...)
    )

    Forall ?X ?Y ?Z ?EX ?EY ?2 ?3 ?4 ?5 ?6 (
      _skolem1(...)#Top(_employee->?Z) :- And(...)
    )

    _1#_cofounders _1#Top(_Ernie _Tony) _2#_hire   _2#Top(_Ernie _Kate)
    _3#_equity     _3#Top(_Ernie 50)    _4#_equity _4#Top(_Tony 30)
    _startup##_company
  )
)
```

4 Mapping the Normalized PSOA Source to Prolog

In this section, we explain the mapping from normalized PSOA constructs to Prolog constructs. The mapping function is denoted by ρ_{psoa}, which is applied recursively in a top-down manner. The translation introduces three distinguished predicates shown in Table 2, `memterm`, `sloterm`, and `tupterm`, defined by Prolog clauses, to represent the three central constructs: membership terms, slot terms, and tuple terms from PSOA RuleML. Section 4.1 discusses the translation of constants and variables and Section 4.2 discusses the translation of central PSOA constructs.

4.1 Constants and Variables

The translation $\rho_{psoa}(c)$ of a constant c is determined as follows:

- If c is a number, $\rho_{psoa}(c)$ is the corresponding Prolog number.
- If c is an arithmetic built-in adopted from RIF [11], $\rho_{psoa}(c)$ is the corresponding Prolog built-in, as listed in Table 1.

Table 1. Mapping of PSOA built-ins to Prolog

PSOA/PS Built-in	Prolog Built-in
`func:numeric-add`	`'+'`
`func:numeric-subtract`	`'-'`
`func:numeric-multiply`	`'*'`
`func:numeric-divide`	`'/'`
`func:numeric-integer-divide`	`'//'`
`func:numeric-mod`	`mod`
`pred:numeric-equal`	`'=:='`
`pred:numeric-less-than`	`'<'`
`pred:numeric-less-than-or-equal`	`'=<'`
`pred:numeric-greater-than`	`'>'`
`pred:numeric-greater-than-or-equal`	`'>='`
`pred:numeric-not-equal`	`'=\='`

- Otherwise, $\rho_{psoa}(c)$ is the single-quoted version of c.

The translation $\rho_{psoa}(v)$ of a '`?`'-prefixed variable v replaces '`?`' with the upper-case letter '`Q`' (`Q`uestion mark) to make it a valid Prolog variable. For example, a PSOA variable `?x` is mapped to a Prolog variable `Qx`.

4.2 Central PSOA Constructs

Table 2 gives mappings of all central PSOA constructs. The translation of tuple terms of the form `o#Top(`$\mathtt{t}_1 \ldots \mathtt{t}_k$`)` and slot terms of the form `o#Top(p->v)` is adopted from PSOA2TPTP [14], using distinguished predicates `tupterm` and `sloterm` respectively.

Table 2. Mapping from PSOA/PS constructs to Prolog constructs

PSOA/PS Constructs	Prolog Constructs
o#Top(t_1...t_k)	tupterm(ρ_{psoa}(o), ρ_{psoa}(t_1) ... ρ_{psoa}(t_k))
o#Top(p->v)	sloterm(ρ_{psoa}(o), ρ_{psoa}(p), ρ_{psoa}(v))
o#c()	memterm(ρ_{psoa}(o), ρ_{psoa}(c))
f(t_1...t_k)	ρ_{psoa}(f)(ρ_{psoa}(t_1), ... ,ρ_{psoa}(t_k))
And(f_1 ... f_n)	(ρ_{psoa}(f_1) , ... , ρ_{psoa}(f_n))
Or(f_1 ... f_n)	(ρ_{psoa}(f_1) ; ... ; ρ_{psoa}(f_n))
Exists ?v_1 ... ?v_m (φ)	$\rho_{psoa}(\varphi)$
Forall ?v_1 ... ?v_m (φ)	$\rho_{psoa}(\varphi)$
φ :- ψ	$\rho_{psoa}(\varphi)$:- $\rho_{psoa}(\psi)$.
?v=External(f(t_1...t_k))	is(ρ_{psoa}(?v), ρ_{psoa}(f)(ρ_{psoa}(t_1), ... , ρ_{psoa}(t_k)))
c1##c2	memterm(X,ρ_{psoa}(c2)) :- memterm(X,ρ_{psoa}(c1)).

The translation of a membership term o#c() is a binary term using the predicate memterm.[6]

The translation of a constructor function application f(t_1...t_k) results from recursively mapping the function name and all arguments into Prolog. The translation of a conjunction And(...) is a parenthesized comma-separated Prolog formula, which can be nested inside an outer formula. Similarly, the translation of a disjunction Or(...) is a semicolon-separated Prolog formula.

The translation of an existential quantification, which can only occur in queries after the normalization phase, is the translation of the quantified formula. In the translation of queries, all existentially quantified variables become free Prolog variables in the translated query, and the bindings for them are discarded in the post-processing of query answers from the Prolog engine.

The translation of an universal quantification, which can only occur in KBs, is the translation of the quantified formula, since all free variables in a Prolog clause are treated as implicitly universally quantified.

The translation of an equality formula ?V=External(f(t_1...t_k)) that equates a variable and an external function application, which evaluates the function application, results in a binary is-primitive invocation in Prolog. The first argument of is is the translated variable, which will be bound to the evaluation result of the second argument, the translated external function application in Prolog syntax, omitting the keyword External. The mapping of PSOA built-ins to Prolog built-ins is listed in Table 1.

The translation of a subclass formula c1##c2 is the same as the translation of the equivalent PSOA rule Forall ?X (?X#c2 :- ?X#c1), resulting in memterm(X,ρ_{psoa}(c2)) :- memterm(X,ρ_{psoa}(c1)).

For each Prolog clause in the translated KB, a period '.' is added to its end.

[6] In our earlier translation from PSOA to the first-order TPTP, the predicate name member was used. We changed the name in both PSOA2Prolog and PSOA2TPTP, because member is a built-in predicate in some Prolog engines such as SWI Prolog.

4.3 Mapping the Startup Example

The Prolog KB and query mapped from the normalized Startup Example in Section 3.6, as well as the query answer are shown in the following. The first four rules share the same premise so we only expand the premise for the first rule due to space limitation.

Translated KB:

```
memterm('_skolem1'(QX,QY,QZ,QEX,QEY,Q2,Q3,Q4,Q5),'_startup') :-
 (((memterm(Q2,'_cofounders'),tupterm(Q2,QX,QY)),
   (memterm(Q3,'_hire'),tupterm(Q3,QX,QZ)),
   (memterm(Q4,'_equity'),tupterm(Q4,QX,QEX)),
   (memterm(Q5,'_equity'),tupterm(Q5,QY,QEY)),
   (is(Q0,'+'(QEX,QEY)),'=<'(Q0,100)))).
tupterm('_skolem1'(...),QX,QY) :- (...).
sloterm('_skolem1'(...),'_employee',QX) :- (...).
memterm('_1','_cofounders').   tupterm('_1','_Ernie','_Tony').
memterm('_2','_hire').         tupterm('_2','_Ernie','_Kate').
memterm('_3','_equity').       tupterm('_3','_Ernie',50).
memterm('_4','_equity').       tupterm('_4','_Tony',30).
memterm(X,'_company') :- memterm(X,'_startup').
```

Translated Query:

```
(memterm(Q1,'_company'),tupterm(Q1,QX,QY)).
```

Query Answer in Prolog:

```
Q1='_skolem1'(...), QX='_Ernie', QY='_Tony'
```

Since the variable `?1` is existentially quantified in the normalized query, the binding for its Prolog translation `Q1` will be discarded, and the bindings for `QX` and `QY` are translated back to PSOA, resulting `?X=_Ernie ?Y=_Tony`.

5 Realization and Evaluation

We realized the PSOA2Prolog translator in Java,[7] based on the ANTLR v3 software.[8] It is composed of a lexer, a parser, and multiple tree walkers, generated from ANTLR using the grammars we developed. The lexer and parser read the input PSOA/PS KB or query and constructs an ANTLR abstract syntax tree (AST), which is a condensed and structured internal representation of the input. The AST is then processed by six tree walkers. Five of them implement the five normalization steps explained in Section 3 by rewriting the AST. The other one implements the mapping step in Section 4 by traversing the normalized AST and generating the translated Prolog KB/query.

[7] We chose Java for better reusability of components of the implementation in other applications.

[8] A language framework for constructing recognizers, interpreters, compilers and translators from grammatical descriptions. `http://www.antlr3.org/`

Composing PSOA2Prolog and XSB Prolog,[9] a fast Prolog engine for an ISO Prolog superset, we achieved the instantiation PSOATransRun[PSOA2Prolog, XSBProlog][10] of our PSOATransRun framework [10]. XSB Prolog does tabling of subgoals and their answers, which ensures termination and optimal efficiency for queries to a large class of programs. This new PSOATransRun instantiation provides query answering in PSOA RuleML by translating the input PSOA KB and query into Prolog, executing them in XSB Prolog and obtaining all answers, discarding bindings of existentially quantified variables in the original query, and translating the results back to PSOA/PS. In our implementation, the InterProlog Java API[11] is employed for accessing XSB Prolog from Java.

We evaluated this Prolog instantiation of PSOATransRun by a comparison to our previous TPTP instantiation PSOATransRun[PSOA2TPTP,VampirePrime]. The experiments were executed on a virtual machine with Intel Core i5-2410M 2.30GHz CPU and 4GB memory running Ubuntu 11. The first evaluation was performed on a test suite[12] of 30 test cases and 90 queries, which covers all PSOA features that we have implemented. Each test case consists of one KB, multiple queries and one user-provided answer to each query. The Prolog instantiation passed all 30 test cases, while the TPTP instantiation failed on 10 test cases which contain features that it currently does not support: external functions, subclass formulas, and IRI constants. For the 20 test cases on which both instantiations succeeded, the Prolog instantiation takes 78.6ms on average for each query while the TPTP instantiation takes 12.6ms. Here, the Prolog instantiation is slower largely due to the communication overhead between the InterProlog API and the engine, which takes constant time.

To compare the performance of the two instantiations on larger KBs, we started developing size-parameterized test-case generators including `chain(k)` in Python. Each call of `chain(k)` generates a KB consisting of one fact `_r0(_a1 _a2 _a3)` and k rules of the form

`Forall ?X ?Y ?Z (_r`i`(?X ?Y ?Z) :- _r`i'`(?X ?Y ?Z))`, $i = 1, \ldots, k, i' = i - 1$.

The query is `_r`k`(?X ?Y ?Z)`, which has one answer `?X=_a1 ?Y=_a2 ?Z=_a3`. We measured the average query execution time of the two instantiations on ten test cases, starting with $k = 20$ and increasing in steps of 20 rules until reaching $k = 200$. The results are shown in Fig. 2.

As seen in the chart, the Prolog instantiation is slower than the TPTP instantiation when $k = 20$ and $k = 40$, due to communication overhead, breaks even at $k = 60$, and becomes faster as k further increases. At $k = 200$, the Prolog instantiation takes 38% of time used by the TPTP instantiation.

Besides being more efficient on larger test cases, the Prolog instantiation is also more efficient when similar queries are posed to the engine, because XSB Prolog can reuse the tabled solutions to subgoals of a query for future queries

[9] `http://xsb.sourceforge.net/`
[10] `http://psoa.ruleml.org/transrun/0.7/local/`
[11] `http://interprolog.com/`
[12] `http://psoa2tptp.googlecode.com/svn/trunk/PSOATransRun/test/`

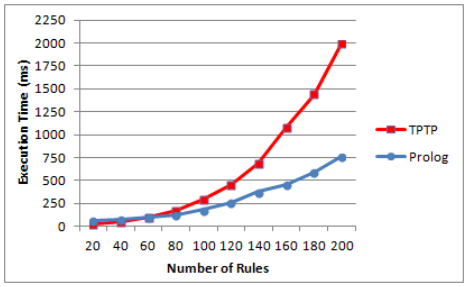

Fig. 2. Performance of PSOATransRun instantiations on `chain(k)` test cases

that use the same subgoals. We tested a special case on the `chain(200)` KB by posing the same query repeatedly. The initial query takes around 760ms while later ones take only around 70ms, which comprise mostly communication overhead.

6 Conclusions and Future Work

In this paper, we discussed the realization of PSOA2Prolog, a Java- and ANTLR-based translator from a subset of the homogeneous object-relational Web rule language PSOA RuleML (with restricted use of equality) to the relational ISO Prolog, for the interoperation and implementation of PSOA rules. PSOA2Prolog is composed of a multi-step source-to-source normalizer followed by a mapper to a pure (Horn) subset of ISO Prolog. The normalizer transforms the KB in five steps: Objectification, Skolemization, slotribution and tupribution, flattening, and rule splitting. We showed the semantics preservation of the steps. The subsequent mapper transforms the normalized PSOA KB into Prolog syntax. The PSOATransRun[PSOA2Prolog, XSBProlog] composition provides efficient query answering for PSOA RuleML. It outperforms our earlier PSOATransRun instantiation targeting TPTP on large test cases. The new interoperation and implementation platform also supports more PSOA features than the TPTP instantiation.

Future work includes versions of PSOA2Prolog that implement equality for user-defined functions and translate to an expanded set of built-ins. Further, detailed comparisons regarding the functionality and performance of comparable subsets of PSOA RuleML and (1) 'native' (not mapped) ISO Prolog and (2) F-logic, whose Flora-2 compiler also maps to XSB Prolog. Moreover, (1) exploring further relational translation targets besides TPTP and Prolog; (2) realizing translations between PSOA RuleML and object-centered languages such as N3; (3) studying subsets with interesting properties, e.g. function-free PSOA RuleML, and its connection to Datalog$^{\pm}$ [15].

References

1. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42**(4) (July 1995) 741–843

2. Yang, G., Kifer, M.: Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. In Spaccapietra, S., March, S.T., Aberer, K., eds.: J. Data Semantics I. Volume 2800 of Lecture Notes in Computer Science., Springer (2003) 69–97
3. Boley, H., Kifer, M.: RIF Basic Logic Dialect (Second Edition) (February 2013) W3C Recommendation, `http://www.w3.org/TR/rif-bld`.
4. Boley, H.: A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules. In: Proc. 5th International Symposium on Rules: Research Based and Industry Focused (RuleML-2011 Europe), Barcelona, Spain. Lecture Notes in Computer Science, Springer (July 2011) 194–211
5. Yang, G., Kifer, M.: FLORA: Implementing an efficient DOOD system using a tabling logic engine. In Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palamidessi, C., Pereira, L., Sagiv, Y., Stuckey, P.J., eds.: Computational Logic – CL 2000. Volume 1861 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2000) 1078–1093
6. Kifer, M., Yang, G., Wan, H., Zhao, C.: Flora-2: User Manual `http://flora.sourceforge.net/`.
7. Baral, C., Liang, S.: From knowledge represented in frame-based languages to declarative representation and reasoning via ASP. In Brewka, G., Eiter, T., McIlraith, S.A., eds.: KR, AAAI Press (2012)
8. ISO/IEC 13211-1: Prolog – part 1: General core (1995)
9. Deransart, P., Ed-Dbali, A., Cervoni, L.: Prolog: The Standard. Springer (1996)
10. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: PSOATransRun: Translating and Running PSOA RuleML via the TPTP Interchange Language for Theorem Provers. In Ait-Kaci, H., Hu, Y.J., Nalepa, G.J., Palmirani, M., Roman, D., eds.: Proceedings of the RuleML2012@ECAI Challenge, at the 6th International Symposium on Rules, CEUR-874 (August 2012)
11. Polleres, A., Boley, H., Kifer, M.: RIF Datatypes and Built-ins 1.0 (Second Edition) (February 2013) W3C Recommendation, `http://www.w3.org/TR/2013/REC-rif-dtb-20130205/`.
12. Boley, H.: PSOA RuleML: Integrated object-relational data and rules. In: Reasoning Web. Springer (2015)
13. C.L. Chang, R.C.T. Lee: Symbolic Logic and mechanical Theorem Proving. Academic Press (1973)
14. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners. In Bikakis, A., Giurca, A., eds.: Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France. Volume 7438 of Lecture Notes in Computer Science., Springer (August 2012) 264–279
15. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics **14** (July 2012) 57–83