Minimal Objectification and Maximal Unnesting in PSOA RuleML

Gen Zou, Harold Boley

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

Abstract. The paper introduces two connected advancements of Positional-Slotted, Object-Applicative RuleML: (1) a model-theoretic semantics, realized transformationally, that directly handles atoms (i.e., predicate applications) without object identifiers (e.g., relationships as in Prolog) and (2) a transformational semantics that handles nested atomic formulas (e.g., nested frames as in Flora-2/F-logic). For (1), the model theory is extended to atoms with optional OIDs, the transformation is developed from static to dynamic objectification, and the correctness of the realization is proved. For (2), the unnesting transformation is defined to decompose nested atomic formulas into equivalent conjunctions.

1 Introduction

The relational and graph (object-centered) modeling paradigms have been widely used for knowledge representation in AI and the Semantic Web. The relational paradigm (e.g., classical logic and logic programming) is built on top of *relationships* with positional arguments, while the object-centered paradigm (e.g., RDF, N3, and F-logic) is built on top of *frames* with a globally unique Object IDentifier (OID), usually typed by a class, and an unordered collection of slotted (attribute-value) arguments. To facilitate interoperation between the two paradigms, e.g. for expressing mappings between frames and relational databases in rule-based data access, combined object-relational paradigms have been studied. F-logic [1,2] and RIF-BLD [3] employ a heterogeneous approach to allow the combined use of relationships and frames. In contrast, the Web rule language PSOA RuleML [4] employs a homogeneous approach by generalizing relationships and frames to **p**ositional-**s**lotted **o**bject-**a**pplicative (psoa) terms[1], which permit the application of a predicate (acting as a relation) to be [in an *oidless/oidful* dimension] without or with an OID – typed by the predicate (acting as a class) – and the predicate's arguments to be [in an orthogonal dimension] *positional, slotted, or combined*.

PSOA RuleML allows the interchangeable use of oidless and oidful *atom*s (i.e., predicate applications) through the *objectification* transformation, which creates an OID for each oidless atom. Earlier, OIDs needed to be statically generated for

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

each oidless psoa term before applying the model-theoretic semantics [4,5]. This is inappropriate for *expressions* (i.e., function applications) since their functions cannot act as classes. It also causes overhead for an atom whose predicate in the clauses of the Knowledge Base (KB) is used only as a Prolog-like relation, in particular does not occur with an OID or slots (the latter also requires an OID for slotribution, explained in Section 2). To address these issues, the model theory is extended so that atoms and expressions can be interpreted without the need for an explicit OID, and an equivalence between an oidless atom and its existential oidful form is guaranteed. A novel static/dynamic objectification transformation – which is minimal by performing as little as possible in a static manner – is introduced to realize this semantics while leaving unchanged as many of the oidless atoms as possible, allowing better use of the underlying Prolog engine in our PSOATransRun [6] implementation since version 1.0, and better interoperation with ground facts in relational databases.

Frames are often nested. PSOA RuleML's presentation syntax (PSOA/PS) generally permits embedded atoms almost everywhere. However, the semantics of embedded atoms has not been clearly defined in the earlier version of the language. In this paper, the unnesting transformation is formally defined to decompose nested atomic formulas into equivalent conjunctions, augmenting our implementation in the PSOATransRun 1.1 release.[2] It recursively extracts oidful atoms from an atomic formula, e.g. another atom, leaving behind their OIDs in the "trimmed" version. Unnesting is maximal by extracting atoms not only from other atoms but also from expressions. This transformational semantics is applied to every atomic formula before the model-theoretic semantics is applied.

The rest of the paper is organized as follows. Section 2 reviews the PSOA RuleML language. Section 3 gives the new semantics with minimal objectification and discusses different objectification transformations. Section 4 discusses the syntax of embedded atoms as well as the unnesting transformation. Section 5 concludes the paper.

2 PSOA RuleML

In this section we introduce the basics of the object-relational Web rule language PSOA RuleML [4], which generalizes RIF-BLD [3] and POSL [7] by introducing psoa terms of the following general forms:

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

$$\textbf{Oidful}: \mathtt{o\#f([t_{1,1} \ldots t_{1,n_1}] \ldots [t_{m,1} \ldots t_{m,n_m}]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)} \tag{2}$$

Both (1) and (2) apply a function or predicate `f`, possibly identified by an OID `o` through a membership `o#f` of `o` in `f` (acting as a class), to a bag of tupled arguments $\mathtt{[t_{i,1} \ldots t_{i,n_i}]}$, $i = 1, \ldots, m$, and to a bag of slotted arguments $\mathtt{p_j\text{->}v_j}$, $j = 1, \ldots, k$, representing attribute-value pairs. For the most often used special case of single-tuple psoa terms (m=1), the square brackets enclosing the tuple may be omitted.

[2] Available from `http://psoa.ruleml.org/transrun/1.1/local/`.

A psoa term can be interpreted as a psoa expression or a psoa atom, depending on whether `f` is a function or predicate. The interpretation as an expression was earlier allowed for both oidless and oidful psoa terms (i.e., (1) and (2) above) but will be restricted to oidless psoa terms here, as explained in Section 4. The interpretation as an atom is permitted for both (1) and (2) on the top-level, while restricted to (2) when embedded, as explained in Section 4.

The OID, tuples, and slots in a psoa atom are all optional. For an oidless psoa atom, without a 'user' OID, *objectification* can introduce a 'system' OID to make it oidful, as detailed in Section 3. Untyped objects are notated by using the root class `Top` as their type `f`. An oidful psoa atom of the form (2) is equivalent to a conjunction

$$\texttt{And(o\#f o\#Top(t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{)}\ \ldots\ \texttt{o\#Top(t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{)}$$
$$\texttt{o\#Top(p}_1\texttt{->v}_1\texttt{)}\ \ldots\ \texttt{o\#Top(p}_k\texttt{->v}_k\texttt{))}$$

of its class membership, its bag of object-centered tuples (*tupribution*), and its bag of object-centered slots (*slotribution*). This *distribution* (tupribution and/or slotribution) could be physically implemented on the Web. A systematics of special kinds of psoa atoms, including (oidless) *relationships* $\texttt{f(t}_1\ \ldots\ \texttt{t}_n\texttt{)}$ and (oidful) *frames* $\texttt{o\#f(p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}$, is elaborated in [5] with many examples.

The alphabet of PSOA RuleML includes a single set `Const` of individual, function, and predicate constants – to prepare functional-logic integration as, e.g., in Relfun, Hilog, and RIF – as well as a set `Var` of variables. Constants include `Top`, denoting the root class, and '`_`'-prefixed *local constants* (e.g., `_a`). Variables are '`?`'-prefixed (e.g., `?X`).

The syntax of PSOA RuleML is built on *terms*. A *simple term* is a constant or a variable. An *atomic formula* is a psoa atom in the form of terms (1) or (2), a subclass term `c1##c2`, an equality term `t1=t2`, or an external term `External(f(...))`. Complex formulas can be constructed using the Horn-like subset of first-order logic (FOL), including conjunctions $\texttt{And(}\tau_1\ \ldots\ \tau_n\texttt{)}$, disjunctions $\texttt{Or(}\tau_1\ \ldots\ \tau_n\texttt{)}$ in the rule body, top-level rule implications τ_1 `:-` τ_2, existential quantification $\texttt{Exists ?X}_1\ \ldots\texttt{?X}_n\ \texttt{(}\tau_1\texttt{)}$, and top-level universal quantification $\texttt{Forall ?X}_1\ \ldots\texttt{?X}_n\ \texttt{(}\tau_1\texttt{)}$. A *group formula* $\texttt{Group(}\tau_1\ \ldots\ \tau_n\texttt{)}$ wraps a set of facts and rules into a KB.

The model-theoretic 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 individual and function interpretations. $\boldsymbol{I}_{\text{C}}$ and $\boldsymbol{I}_{\text{V}}$ interpret constants and variables. $\boldsymbol{I}_{\text{psoa}}$ interprets predicates/functions of psoa terms as semantic functions, which will be shown in Section 3.1. $\boldsymbol{I}_{\text{sub}}$, $\boldsymbol{I}_{=}$, and $\boldsymbol{I}_{\text{external}}$ interpret subclass, equality, and external terms. A generic mapping $\boldsymbol{I}$ from terms to $\boldsymbol{D}$ can be defined using the above interpretation mappings. $\boldsymbol{I}_{\text{truth}}$ maps domain elements to the set of truth values $\boldsymbol{TV} = \{\texttt{t}, \texttt{f}\}$. We will write $\mathcal{I}.\boldsymbol{D}$, $\mathcal{I}.\boldsymbol{I}_{\text{V}}$, etc,. for the components of $\mathcal{I}$ in the rest of the paper.

Truth evaluation for formulas is determined by a recursive evaluation function $TVal_{\mathcal{I}}$ defined in [4]. A semantic structure $\mathcal{I}$ is called a *model* of a KB ϕ if

$TVal_{\mathcal{I}}(\phi) = \mathbf{t}$ and $\mathcal{I}$ conforms to all semantic restrictions (e.g., subclass and slotribution/tupribution restrictions), 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.

We illustrate the syntax and semantics through an example, previewing key concepts, e.g. *non-relational* and *virtual OID*.

Example 1. Consider the following KB:

```
Group (
  Forall ?Pers ?JobTitle ?Comp1 ?Comp2 (
     _transfer(?Pers ?Comp1 ?Comp2) :-
         And(_work(?Pers ?Comp1 ?JobTitle)
             _acquire(_buyer->?Comp2 _seller->?Comp1)))
  _e1#_transfer(_Tony _Rho4biz _Chi4corp _bonus->20000)
  _work(_Kate _Rho4biz "Director")
  _a1#_acquire(_buyer->_Chi4corp _seller->_Rho4biz)
)
```

We will query this KB as follows, without (left) and with (right) an OID:

```
_work(?P ?C ?J)          ?O#_work(?P ?C ?J)
_transfer(?P ?C1 ?C2)    ?O#_transfer(?P ?C1 ?C2)
```

The `_work` fact is relational while the `_acquire` fact has two slots centered on the object `_a1`. The `_transfer` rule uses premises satisfied by these facts. Its `_acquire` premise needs to be objectified to `?1#_acquire(_buyer->?Comp2 _seller->?Comp1)`, so that the generated fresh OID variable `?1` unifies with `_a1`. The `_transfer` fact is non-relational, centered on the graph-node-like object `_e1` – typed by `_transfer` acting as a class – and having one tuple of arguments as well as a slot for an optional `_bonus`. Since `_transfer` acts as a non-relational predicate in this KB atom, it is a non-relational predicate in the entire KB (although it acts as a relation in the rule).

Let us query the KB: The query `_work(?P ?C ?J)` uses the relational KB predicate `_work` in a relationship, while the query `?O#_work(?P ?C ?J)` uses `_work` non-relationally with an OID variable `?O`, which can be bound to a virtual OID created by objectification. The `_transfer(?P ?C1 ?C2)` query uses the non-relational KB predicate `_transfer` without an OID. The `_transfer` rule is invoked, binding `?P` to `_Kate`, `?C1` to `_Rho4biz`, and `?C2` to `_Chi4corp`. The second solution is dependent on query objectification, introducing the existential OID variable `?1`, yielding `Exists ?1 (?1#_transfer(?P ?C1 ?C2))`; the `_transfer` fact is retrieved with the `_bonus` slot ignored, binding `?P` to `_Tony`, `?C1` to `_Rho4biz`, and `?C2` to `_Chi4corp` (the `?1` binding to `_e1` is not shown because of its `Exists` encapsulation). In the query `?O#_transfer(?P ?C1 ?C2)`, `_transfer` acts as a class – i.e., as a non-relational predicate – with an explicit OID variable `?O`. For successful query answering, the `_transfer` rule (e.g., its conclusion) requires objectification while the fact can be used directly. □

More (e.g., geospatial) examples and the open-source implementation of PSOA are available online,[3] for explorations in its operational semantics.

[3] See PSOA's entry to the RuleML website: `http://psoa.ruleml.org`.

3 Minimal Objectification

In PSOA RuleML, each oidless psoa atom σ is understood as having an implicit OID, which allows the interchangeable use of oidless and oidful atoms of the respective term forms (1) and (2) in Section 2. The earlier semantics can only interpret an oidless psoa term after *static objectification*, which generates OIDs for all of the KB's oidless terms. This 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. Moreover, since it will turn out that it is inappropriate to give OIDs to expressions, the earlier semantics cannot deal with expression terms. In this section, we will thus first introduce a modified model-theoretic semantics to allow the direct interpretation of oidless psoa terms. Then we will discuss an objectification systematics for oidless atoms, including undifferentiated and differentiated static objectification transformations, as well as a novel static/dynamic transformation. Static/dynamic objectification is minimal in the sense that it generates as few explicit OIDs as possible, instead constructing virtual OIDs as query variable bindings. The current section assumes that the only allowed embedded complex terms are expressions, which can be achieved by maximal unnesting of embedded atoms – the central theme of Section 4.

3.1 New Semantics for Oidless Psoa Terms

The earlier semantics of PSOA RuleML [4] defined the interpretation of a psoa term as follows, where $\boldsymbol{I}$ is a generic mapping for interpreting any term:

$$\begin{aligned}\boldsymbol{I}(\texttt{o\#f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}) = \\ \boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))(\boldsymbol{I}(\texttt{o}), \\ \{\langle \boldsymbol{I}(\texttt{t}_{1,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_1})\rangle, \ldots, \langle \boldsymbol{I}(\texttt{t}_{m,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_m})\rangle\}, \\ \{\langle \boldsymbol{I}(\texttt{p}_1), \boldsymbol{I}(\texttt{v}_1)\rangle, \ldots, \langle \boldsymbol{I}(\texttt{p}_k), \boldsymbol{I}(\texttt{v}_k)\rangle\})\end{aligned} \quad (3)$$

$\boldsymbol{I}$ first gives a domain element interpretation in $\boldsymbol{D}$ to each component of the psoa term, including: the OID $\texttt{o}$; the predicate/function $\texttt{f}$; each positional argument $\texttt{t}_{i,j}$, $1 \leq \texttt{i} \leq \texttt{m}$, $1 \leq \texttt{j} \leq \texttt{n}_i$; each slot name $\texttt{p}_h$ and filler $\texttt{v}_h$, $1 \leq \texttt{h} \leq \texttt{k}$. Then $\boldsymbol{I}_{\text{psoa}}$ maps $\boldsymbol{I}(\texttt{f}) \in \boldsymbol{D}$ to a semantic function of the general form $\boldsymbol{D}_{\text{ind}} \times \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}}) \times \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}) \rightarrow \boldsymbol{D}$, which takes the following three arguments and returns an element $\texttt{d} \in \boldsymbol{D}$:

- The interpreted OID $\boldsymbol{I}(\texttt{o}) \in \boldsymbol{D}_{\text{ind}}$;
- The bag of interpreted tuples $\{\langle \boldsymbol{I}(\texttt{t}_{i,1}), \ldots, \boldsymbol{I}(\texttt{t}_{i,n_i})\rangle \mid 1 \leq \texttt{i} \leq \texttt{m}\}$ $\in \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}})$, where $\boldsymbol{D}^*_{\text{ind}}$ denotes the set of all finite-length tuples over $\boldsymbol{D}$.
- The bag of interpreted slots $\{\langle \boldsymbol{I}(\texttt{p}_h), \boldsymbol{I}(\texttt{v}_h)\rangle \mid 1 \leq \texttt{h} \leq \texttt{k}\}$ $\in \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}})$.

$\boldsymbol{I}_{\text{psoa}}$ can be applied to $\boldsymbol{I}(\texttt{f})$ no matter $\texttt{f}$ is a predicate or function symbol, where $\boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))$ must be a $\boldsymbol{D}_{\text{ind}}$-valued semantic function if $\texttt{f}$ is a function. Notice that in the above definitions, the mappings $\boldsymbol{I}$ and $\boldsymbol{I}_{\text{psoa}}$ are only defined

for oidful psoa terms and their predicates/functions, hence all psoa terms need to be objectified before applying the semantics. This has several shortcomings:

- Expressions should not have OIDs, which will be explained in Section 4.
- Atoms are required to have OIDs before they can be semantically evaluated, which creates overhead for each atom whose predicate in the KB clauses is used only as a Prolog-like relation.

In order to resolve these problems, we will redefine $\boldsymbol{I}$ and $\boldsymbol{I}_{\text{psoa}}$ to allow a direct interpretation of oidless psoa terms, and also incorporate the objectification virtually into the semantics using a logical equivalence. Specifically, the following changes are introduced:

1. The definition of $\boldsymbol{I}$ for **oidful** psoa terms is changed to

$$\begin{aligned}\boldsymbol{I}(\texttt{o\#f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}) = \\ \boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))(\{\boldsymbol{I}(\texttt{o})\}, \\ \{\langle \boldsymbol{I}(\texttt{t}_{1,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_1})\rangle, \ldots, \langle \boldsymbol{I}(\texttt{t}_{m,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_m})\rangle\}, \\ \{\langle \boldsymbol{I}(\texttt{p}_1), \boldsymbol{I}(\texttt{v}_1)\rangle, \ldots, \langle \boldsymbol{I}(\texttt{p}_k), \boldsymbol{I}(\texttt{v}_k)\rangle\})\end{aligned} \quad (4)$$

   where the first argument of the semantic function $\boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))$ is wrapped into a singleton set $\{\boldsymbol{I}(\texttt{o})\}$. This allows defining $\boldsymbol{I}$ for **oidless** psoa terms separately, by using the empty set $\{\}$ as the first argument:

$$\begin{aligned}\boldsymbol{I}(\texttt{f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}) = \\ \boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))(\{\}, \\ \{\langle \boldsymbol{I}(\texttt{t}_{1,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_1})\rangle, \ldots, \langle \boldsymbol{I}(\texttt{t}_{m,1}), \ldots, \boldsymbol{I}(\texttt{t}_{1,n_m})\rangle\}, \\ \{\langle \boldsymbol{I}(\texttt{p}_1), \boldsymbol{I}(\texttt{v}_1)\rangle, \ldots, \langle \boldsymbol{I}(\texttt{p}_k), \boldsymbol{I}(\texttt{v}_k)\rangle\})\end{aligned} \quad (5)$$

2. The definition of $\boldsymbol{I}_{\text{psoa}}$ is changed to map $\boldsymbol{D}$ to semantic functions of the form $\texttt{SetOfPhiSingletons}(\boldsymbol{D}_{\text{ind}}) \times \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}}) \times \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}) \rightarrow \boldsymbol{D}$, where $\texttt{SetOfPhiSingletons}(\boldsymbol{D}_{\text{ind}})$ is defined as $\{\{\}\} \cup \{\{o\} \mid \texttt{o} \in \boldsymbol{D}_{\text{ind}}\}$, whose elements contains the empty set $\{\}$ and a singleton set $\{\texttt{o}\}$ for each $\texttt{o} \in \boldsymbol{D}_{\text{ind}}$. With this definition, the semantic function $\boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\texttt{f}))$ can be correctly applied to the arguments in equations (4) and (5).
3. Define truth evaluation for oidless psoa terms used as atoms as follows.

$$\begin{aligned}TVal_{\mathcal{I}}(\texttt{f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}) = \\ \boldsymbol{I}_{\text{truth}}(\boldsymbol{I}(\texttt{f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)})\end{aligned} \quad (6)$$

   Here, the following *objectification restriction* is required to capture the logical equivalence between an oidless atom (notice the absence of "`o#`" in front of "`f(...)`") and its existentially objectified form:

$$TVal_{\mathcal{I}}(\texttt{f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{)}) = \mathbf{t}$$

   if and only if

$$TVal_{\mathcal{I}}(\texttt{Exists ?O (?O\#f([t}_{1,1}\ \ldots\ \texttt{t}_{1,n_1}\texttt{]}\ \ldots\ \texttt{[t}_{m,1}\ \ldots\ \texttt{t}_{m,n_m}\texttt{]}\ \texttt{p}_1\texttt{->v}_1\ \ldots\ \texttt{p}_k\texttt{->v}_k\texttt{))}) = \mathbf{t}$$

where ?O is a fresh variable representing the postulated virtual OID of the atom. Notice that the restriction applies only to atoms – after unnesting (cf. Section 4) occurring only the top-level – but not to – embedded – expressions, which do not have a truth value. In the rest of the paper, we will call the semantics with and without objectification restriction *objectification-including* and *objectification-excluding* semantics, respectively.

Next we define the objectification transformation, which realizes the objectification-including semantics by transforming KBs and queries such that entailment can be established under the objectification-excluding semantics.

Definition 1. *(Entailment under objectification-excluding/-including semantics) An interpretation $\mathcal{I}$ is an objectification-excluding (resp., objectification-including) model of ϕ, written as $\mathcal{I} \models_{-o} \phi$ (resp., $\mathcal{I} \models_{+o} \phi$), iff $TVal_{\mathcal{I}}(\phi) = \boldsymbol{t}$ and $TVal_{\mathcal{I}}$ conforms to all other semantic restrictions [4] (e.g., slotribution/tupribution), while it does not (resp., does) guarantee the objectification restriction. A KB ϕ entails a query q under the objectification-excluding (resp. objectification-including) semantics, written as $\phi \models_{-o} q$ (resp., $\phi \models_{+o} q$), iff for every $\mathcal{I}$ such that $\mathcal{I} \models_{-o} \phi$ (resp. $\mathcal{I} \models_{+o} \phi$), $\mathcal{I} \models_{-o} q$ (resp. $\mathcal{I} \models_{+o} q$) holds.*

Since the default semantics is objectification-including, we will use the short notation $\models$ for $\models_{+o}$ in the rest of the paper.

Definition 2. *(Objectification Transformation) An objectification transformation* obj *with respect to a set of KBs Φ and a set of queries Q is a transformation $\Phi \times \Gamma \to \Gamma$, where Φ and Q are subsets of Γ, such that for every KB $\phi \in \Phi$ and query $q \in Q$, $\phi \models q$ iff $\mathsf{obj}(\phi, \phi) \models_{-o} \mathsf{obj}(\phi, q)$. For convenience, we write $\mathsf{obj}(q)$ for $\mathsf{obj}(\phi, q)$ if $\mathsf{obj}(\phi, q)$ is independent of ϕ, and $\mathsf{obj}(\phi)$ for $\mathsf{obj}(\phi, \phi)$.*

The second argument of obj is the input formula to be transformed, while the first argument is the corresponding KB of the formula providing necessary context information. Next, we define a schema for constructing such a transformation obj for KBs and queries from a transformation defined only for atoms.

Definition 3. *Given a set of KBs Φ and a set of queries Q, for any $\phi \in \Phi$, if $\mathsf{obj}(\phi, \tau)$ has been defined, with τ being any atom, then $\mathsf{obj}(\phi, \tau)$ can be extended for τ being a formula other than an atom, where (1) for a subclass or an equality, $\mathsf{obj}(\phi, \tau) = \tau$ is just a projection; (2) for other formulas, $\mathsf{obj}(\phi, \tau)$ is obtained by recursively applying* obj *to each subformula of τ, while keeping the surrounding formula structure unchanged.*

Note that $\mathsf{obj}(\phi, \tau)$ is the same as the formula obtained by replacing every atom ω in τ with $\mathsf{obj}(\phi, \omega)$; for some ω's, this is just a projection $\mathsf{obj}(\phi, \omega) = \omega$. In the rest of the paper we assume every obj is constructed using Definition 3.

In order to prove $\phi \models q$ iff $\mathsf{obj}(\phi) \models_{-o} \mathsf{obj}(\phi, q)$, as required by Definition 2 for any particular obj, one can prove $\phi \models q$ iff $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ and $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ iff $\mathsf{obj}(\phi) \models_{-o} \mathsf{obj}(\phi, q)$. The next lemma gives a sufficient condition for $\phi \models q$ iff $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$, where we write $\mathsf{Var}(\tau)$ for the set of all free

variables in a formula τ, $\mathsf{M}(\mathsf{Var}(\tau), \boldsymbol{D})$ for the set of all mappings from $\mathsf{Var}(\tau)$ to $\boldsymbol{D}$, and $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}})$ for an interpretation that coincides with $\boldsymbol{I}_{\mathrm{V}}$ on all variables it interprets, and with $\mathcal{I}$ on everything else.

We say that a formula has a *positive* occurrence if it is in a fact or a rule conclusion, and a *negative* occurrence if in a query or a rule premise. A semantic structure $\mathcal{I}$ is called a *counter-model* for $\phi \models q$ if $\mathcal{I} \models \phi$ and $\mathcal{I} \not\models q$.

Lemma 1. *Given a KB ϕ, a query q, and a transformation* obj*, $\phi \models q$ iff* $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ *holds if* obj *has the following properties:*

1) For every counter-model $\mathcal{I}$ for $\phi \models q$, there exists $\mathcal{I}'$ s.t. for every KB / query atom ω, $\boldsymbol{I}_V \in \mathsf{M}(\mathsf{Var}(\omega), \mathcal{I}.\boldsymbol{D})$,
1.a) if ω is positive and $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_V) \models \omega$, then $\mathsf{v}(\mathcal{I}', \boldsymbol{I}_V) \models \mathsf{obj}(\phi, \omega)$;
1.b) if ω is negative and $\mathsf{v}(\mathcal{I}', \boldsymbol{I}_V) \models \mathsf{obj}(\phi, \omega)$, then $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_V) \models \omega$.
2) For every counter-model $\mathcal{I}'$ for $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$, there exists $\mathcal{I}$ s.t. for every KB / query atom ω, $\boldsymbol{I}_V \in \mathsf{M}(\mathsf{Var}(\omega), \mathcal{I}'.\boldsymbol{D})$,
2.a) if ω is positive and $\mathsf{v}(\mathcal{I}', \boldsymbol{I}_V) \models \mathsf{obj}(\phi, \omega)$, then $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_V) \models \omega$,
2.b) if ω is negative and $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_V) \models \omega$, then $\mathsf{v}(\mathcal{I}', \boldsymbol{I}_V) \models \mathsf{obj}(\phi, \omega)$.

Proof. (Sketch) Proving $\phi \models q$ iff $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ is equivalent to proving $\phi \not\models q$ iff $\mathsf{obj}(\phi) \not\models \mathsf{obj}(\phi, q)$. We first prove the 'if' part. If $\mathsf{obj}(\phi) \not\models \mathsf{obj}(\phi, q)$, then there exists a counter-model $\mathcal{I}'$ for $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$. Using condition 2.a), we can prove by induction that for every positive formula τ and $\boldsymbol{I}_{\mathrm{V}} \in \mathsf{M}(\mathsf{Var}(\tau), \mathcal{I}.\boldsymbol{D})$, if $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}})}(\mathsf{obj}(\phi, \tau)) = \mathbf{t}$ then $TVal_{\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}})}(\tau) = \mathbf{t}$. Using condition 2.b), we can also prove by induction that for every negative formula τ, if $TVal_{\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}})}(\tau) = \mathbf{t}$ then $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}})}(\mathsf{obj}(\phi, \tau)) = \mathbf{t}$.

Then we can prove that for each top-level formula τ in ϕ, $TVal_{\mathcal{I}}(\mathsf{obj}(\phi, \tau)) = \mathbf{t}$.

- If τ is a ground fact, τ is a positive formula so $TVal_{\mathcal{I}}(\mathsf{obj}(\phi, \tau)) = \mathbf{t}$.
- If τ is a rule of the form `Forall ?X1 ...?Xn (τ1 :- τ2)`, $TVal_{\mathcal{I}'}(\tau) = \mathbf{t}$ means for every $\boldsymbol{I}_{\mathrm{V}}* \in \mathsf{M}(\{\mathtt{?X_1}, \ldots, \mathtt{?X_n}\}, \mathcal{I}.\boldsymbol{D})$, $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}}*)}(\tau_1) = \mathbf{t}$ or $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}}*)}(\tau_2) = \mathbf{f}$ holds. Since τ_1 is positive and τ_2 is negative, either $TVal_{\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}}*)}(\tau_1) = \mathbf{t}$ or $TVal_{\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}}*)}(\tau_2) = \mathbf{f}$ holds for every $\boldsymbol{I}_{\mathrm{V}}*$. So $TVal_{\mathcal{I}}(\tau) = \mathbf{t}$.
- If τ is a non-ground fact, it can be seen as a rule with an empty premise, so that the above proof for general rules still apply.

Hence $\mathcal{I} \models \phi$. Also since the query q is negative, hence $\mathcal{I} \not\models q$. Thus $\phi \not\models q$.

The 'only if' part can be proved using a similar approach based on the existence of a counter-model $\mathcal{I}'$ for $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$. □

Corollary 1. *Given a KB ϕ, a query q, and a transformation* obj*, $\phi \models q$ iff* $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ *holds if for every KB / query atom ω and semantic structure $\mathcal{I}^*$, $\mathcal{I}^* \models \omega$ iff $\mathcal{I}^* \models \mathsf{obj}(\phi, \omega)$ holds.*

Proof. We will show that obj has the properties 1) and 2) in Lemma 1. For property 1), if $\mathcal{I}$ is a counter-model for $\phi \models q$, then we can choose $\mathcal{I}' = \mathcal{I}$. For every $\boldsymbol{I}_{\mathrm{V}} \in \mathsf{M}(\mathsf{Var}(\omega), \mathcal{I}.\boldsymbol{D})$, according to the assumption where $\mathcal{I}^*$ becomes $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}})$, we have $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}}) \models \omega$ iff $\mathsf{v}(\mathcal{I}, \boldsymbol{I}_{\mathrm{V}}) \models \mathsf{obj}(\phi, \omega)$. Hence 1.a) and 1.b) are satisfied and obj has property 1).

That obj has property 2) can be proved similarly by choosing $\mathcal{I} = \mathcal{I}'$. □

Lemma 2 gives a sufficient condition for $\mathsf{obj}(\phi) \models \mathsf{obj}(\phi, q)$ iff $\mathsf{obj}(\phi) \models_{\text{-}o} \mathsf{obj}(\phi, q)$.

Lemma 2. *Given a KB ϕ' and a query q', $\phi' \models q'$ iff $\phi' \models_{\text{-}o} q'$ holds if for each counter-model $\mathcal{I}''$ for $\phi' \models_{\text{-}o} q'$ there exists a counter-model $\mathcal{I}'$ for $\phi' \models q'$.*

Proof. We first prove the 'if' part. For every $\mathcal{I}'$ s.t. $\mathcal{I}' \models \phi'$, $\mathcal{I}' \models_{\text{-}o} \phi'$ since $\models$ more restricted than $\models_{\text{-}o}$. Also since $\phi' \models_{\text{-}o} q'$, $\mathcal{I}' \models_{\text{-}o} q'$. Because $\mathcal{I}' \models \phi'$, $\mathcal{I}'$ guarantees the objectification restriction. Hence $\mathcal{I}' \models q'$ always holds and $\phi' \models q'$ is proved.

Next we prove the 'only if' part, which is equivalent to $\phi' \not\models q'$ if $\phi' \not\models_{\text{-}o} q'$. If $\phi' \not\models_{\text{-}o} q'$, there exists a counter-model $\mathcal{I}''$ for $\phi' \models_{\text{-}o} q'$. According to the assumption of the lemma, there exists a counter-model $\mathcal{I}'$ for $\phi' \models q'$, hence $\phi' \not\models q'$ is proved. □

The next subsections will discuss different objectification transformations.

3.2 Static Objectification Transformations

The static objectification $\mathsf{obj}_s(\alpha)$ of a KB/query α is obtained by replacing each oidless atom σ with its objectified form $\mathsf{obj}_s(\sigma)$ having a generated OID. The generation can adopt either an undifferentiated method $\mathsf{obj}_{s^=}$, which uniformly transforms σ everywhere, or a differentiated method $\mathsf{obj}_{s^\neq}$ [6], which transforms σ differently based on its occurrence.

Definition 4. *(Undifferentiated static objectification) The undifferentiated static objectification* $\mathsf{obj}_{s^=}(\omega)$ *of atom is* **(a)** *ω if ω is oidful;* **(b)** `Exists ?i (?i#f(...))` *if ω is oidless, where* `?i` *is a fresh variable in the clause chosen from* `?1`*,* `?2`*, ...*

$\mathsf{obj}_{s^=}$ *is extended for other formulas according to Definition 3.*

In Example 1, $\mathsf{obj}_{s^=}$ of the KB and the queries replaces all oidless `_transfer`, `_work`, and `_acquire` atoms with their existential forms.

The following theorem shows the correctness of $\mathsf{obj}_{s^=}$.

Theorem 1. $\mathsf{obj}_{s^=}$ *is an objectification transformation for any KB and query.*

Proof. We first show that for every KB / query atom ω and semantic structure $\mathcal{I}^*$, $\mathcal{I}^* \models \omega$ iff $\mathcal{I}^* \models \mathsf{obj}_{s^=}(\omega)$ holds. If ω is oidful, then $\mathsf{obj}_{s^=}(\omega) = \omega$, and the statement holds trivially. Otherwise ω is oidless. If $\mathcal{I}^* \models \omega$, then $\mathcal{I}^*$ guarantees the objectification restriction. So $TVal_{\mathcal{I}^*}(\mathsf{obj}_{s^=}(\omega)) = TVal_{\mathcal{I}^*}(\omega) = \mathbf{t}$, and $\mathcal{I}^* \models \mathsf{obj}_{s^=}(\omega)$ holds. Similarly, if $\mathcal{I}^* \models \mathsf{obj}_{s^=}(\omega)$, $\mathcal{I}^* \models \omega$ also holds. Hence $\mathcal{I}^* \models \omega$ iff $\mathcal{I}^* \models \mathsf{obj}_{s^=}(\omega)$ holds also for oidless atoms. Thus the condition of Corollary 1 is fulfilled and $\phi \models q$ iff $\mathsf{obj}_{s^=}(\phi) \models \mathsf{obj}_{s^=}(q)$.

Next we will show that for each $\mathsf{obj}_{s^=}(\phi)$ and $\mathsf{obj}_{s^=}(q)$, the condition of Lemma 2 is fulfilled. If there exists a counter-model $\mathcal{I}''$ for $\phi \models q$, then it can be made into a model $\mathcal{I}'$ that guarantees the objectification restriction, by redefining $TVal_{\mathcal{I}'}(\omega)$ for every atom ω that is oidless to be the same as $TVal_{\mathcal{I}''}(\mathsf{obj}_{s^=}(\omega))$. Since the change only affect oidless atoms, which neither exist in $\mathsf{obj}_{s^=}(\phi)$ nor

in $\mathsf{obj}_{s^=}(q)$, $TVal_{\mathcal{I}'}(\phi) = TVal_{\mathcal{I}''}(\phi) = \mathbf{t}$ and $TVal_{\mathcal{I}'}(q) = TVal_{\mathcal{I}''}(q) = \mathbf{f}$. Hence $\mathcal{I}'$ is a counter-model for $\phi \models q$ and the condition of Lemma 2 is fulfilled. Thus $\mathsf{obj}_{s^=}(\phi) \models \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$.

So $\phi \models q$ iff $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ and the theorem is proved. □

Definition 5. *(Differentiated static objectification) For an atom ω, the differentiated static objectification $\mathsf{obj}_{s^\neq}(\omega)$ is defined as ω if ω is oidful, or as follows if ω is oidless:*

Case 1: If ω is a ground fact, $\mathsf{obj}_{s^\neq}(\omega) =$ `_i#f(...)`*, where* `_i` *is a fresh local constant symbol chosen from* `_1`*,* `_2`*, ..., which neither occurs elsewhere in the KB nor is used for the objectification of other atoms.*

Case 2: If ω is a non-ground fact, a rule conclusion atom, or a query atom, $\mathsf{obj}_{s^\neq}(\omega) =$ `Exists ?j (?j#f(...))`*.*

Case 3: If ω is a rule premise atom, $\mathsf{obj}_{s^\neq}(\omega) =$ `?j#f(...)`*, where* `?j` *is a fresh variable scoped universally by the enclosing rule.*

$\mathsf{obj}_{s^\neq}$ is extended for other formulas according to Definition 3.

In Example 1, differentiated static objectification generates a fresh OID constant for the `_work` ground fact according to Case 1, replaces the oidless `_work` and `_transfer` atoms in the rule conclusion and in the queries with existentials according to Case 2, and generates fresh universal OID variables for the `_work` and `_acquire` atoms in the rule premise according to Case 3.

It is easy to prove the correctness of $\mathsf{obj}_{s^\neq}$ by showing that it leads to the same set of entailed queries as $\mathsf{obj}_{s^=}$.

Lemma 3. *Given a KB ϕ and a query q, if q does not use a generated OID constant symbol* `_1`*,* `_2`*, ..., then $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^\neq}(q)$.*

Proof. According to their definitions, both methods transform oidless query atoms into the same existential form, hence $\mathsf{obj}_{s^=}(q) = \mathsf{obj}_{s^\neq}(q)$ for any query q. Thus $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^\neq}(q)$.

For KB atoms, the two methods are identical in Case 2 and differ in Case 1 and 3. The transformation of a rule premise atom in Case 3 using a universal OID variable on the top-level is equivalent to using an embedded existentially quantified formula with an existential OID variable in the rule premise. For a ground fact ω handled by Case 1, $\mathsf{obj}_{s^\neq}(\omega)$ can be seen as a Skolemized version of $\mathsf{obj}_{s^=}(\omega)$ using Skolem constants `_1`, `_2`, ..., hence they entail the same set of queries as long as these queries do not use the Skolem constants, thus avoiding a clash of constant symbols. Hence we have $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$. So $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$, and the lemma holds. □

Theorem 2. *$\mathsf{obj}_{s^=}$ is an objectification transformation with respect to a set of KBs Φ and to a set of queries Q that do not use constants* `_1`*,* `_2`*, ...*

Proof. For any $\phi \in \Phi$ and $q \in Q$, by Theorem 1 we have $\phi \models q$ iff $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$. By Lemma 3, $\mathsf{obj}_{s^=}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^=}(q)$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^\neq}(q)$. Hence $\phi \models q$ iff $\mathsf{obj}_{s^\neq}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^\neq}(q)$. So the statement holds by Definition 2. □

3.3 Static/Dynamic Objectification Transformation

For KBs in which most or all of the predicates are Prolog-like relations, it is often not necessary to generate OIDs for their oidless atoms explicitly. In this subsection, a novel static/dynamic objectification approach is introduced to keep unchanged as many of the KB's oidless atoms as possible, instead constructing virtual OIDs at query time when bindings for OID variables are being queried.

In order to apply static/dynamic objectification to a KB ϕ and its queries, the set of KB predicates $\mathsf{PredKB}(\phi)$ will be partitioned into two disjoint subsets. $\mathsf{PredKB}(\phi)$ is defined as $\{\mathsf{Pred}(\lambda) \mid \lambda \text{ is an atom in } \phi\}$, where $\mathsf{Pred}(\lambda)$ denotes the predicate symbol of λ. The partitioning of $\mathsf{PredKB}(\phi)$ is defined next.

Definition 6. *(Non-relational and relational predicates) Given a KB ϕ, a predicate* `f` $\in \mathsf{PredKB}(\phi)$ *is* **non-relational** *in ϕ if* `f` *occurs at least once in a multi-tuple, oidful, or slotted atom of ϕ, or in a subclass formula of ϕ. Conversely,* `f` *is* **relational** *in ϕ if it has no such occurrence. The sets of non-relational and relational predicates of ϕ are written as $\mathsf{PredKB}_{NR}(\phi)$ and $\mathsf{PredKB}_{R}(\phi)$, respectively.*

For atoms using a relational predicate in $\mathsf{PredKB}_{R}(\phi)$, their OIDs can be virtualized by dynamic objectification:

Definition 7. *(Dynamic objectification) The dynamic objectification $\mathsf{obj}_d(\phi, \omega)$ of an atom ω with respect to a KB ϕ, where $\mathsf{Pred}(\omega) \in \mathsf{PredKB}_{R}(\phi)$, is defined as ω if ω is a KB atom in ϕ, or as the following rewriting if ω is a query atom:*

Case 1: If ω is a relationship, $\mathsf{obj}_d(\phi, \omega) = \omega$.
Case 2: If ω has a non-variable (e.g., constant or expression) OID or a slot, $\mathsf{obj}_d(\phi, \omega =$ `Or()`*, where* `Or()` *is an encoding of explicit falsity.*
Case 3: If ω has an OID variable and $m > 0$ tuples, being of the form `?O#f([`$\mathtt{t}_{1,1} \ldots \mathtt{t}_{1,n_1}$`] ... [`$\mathtt{t}_{m,1} \ldots \mathtt{t}_{m,n_m}$`])`*, equivalent (via 'class-reproducing' tupribution) to a conjunction separately applying* `?O#f` *to all tuples,*

$$\mathtt{And(?O\#f([}\mathtt{t}_{1,1} \ldots \mathtt{t}_{1,n_1}\mathtt{])} \ldots \mathtt{?O\#f([}\mathtt{t}_{m,1} \ldots \mathtt{t}_{m,n_m}\mathtt{]))},$$

abbreviated to

$$\mathtt{And(?O\#f(}\mathtt{t}_{1,1} \ldots \mathtt{t}_{1,n_1}\mathtt{)} \ldots \mathtt{?O\#f(}\mathtt{t}_{m,1} \ldots \mathtt{t}_{m,n_m}\mathtt{))},$$

then $\mathsf{obj}_d(\phi, \omega)$ is a relational conjunction querying the oidless versions of these applications while using explicit equalities between the OID variable `?O` *and an OID-constructor function* `_oidcons` *applied to the predicate and the elements of each tuple (where the* `?O` *equalities also enforce tuple unification, so that the m single-tuple* `f` *relationships can be satisfied by a single KB clause, thus realizing special 'virtual multi-tuple' psoa atoms as queries):*

$$\begin{aligned}&\mathtt{And(f(}\mathtt{t}_{1,1} \ldots \mathtt{t}_{1,n_1}\mathtt{)\ ?O = \_oidcons(f\ t}_{1,1} \ldots \mathtt{t}_{1,n_1}\mathtt{)}\\ &\quad \ldots\\ &\quad \mathtt{f(}\mathtt{t}_{m,1} \ldots \mathtt{t}_{m,n_m}\mathtt{)\ ?O = \_oidcons(f\ t}_{m,1} \ldots \mathtt{t}_{m,n_m}\mathtt{))}\end{aligned}$$

In the special case of $\texttt{m = 1}$*, the query atom* $\texttt{?O\#f(t}_1 \ldots \texttt{t}_n\texttt{)}$ *becomes*

$$\texttt{And(f(t}_1 \ldots \texttt{t}_n\texttt{) ?O = \_oidcons(f t}_1 \ldots \texttt{t}_n\texttt{))}$$

The function `_oidcons` *is employed to universally construct OIDs for all relationships so that it needs to take the predicate symbol as the first argument to distinguish between relationships with different predicates.*

Case 4: If ω *is a membership of the form* `?O#f`*,* $\mathsf{obj}_d(\phi, \omega)$ *is a disjunction of* k *formulas* $\mathsf{obj}_d(\texttt{?O\#f(?X}_1 \ldots \texttt{?X}_{n_i}\texttt{)})$*, where* $\texttt{n}_1, \ldots, \texttt{n}_k$ *are the* k *different arities of* f *in the KB:*

$$\texttt{Or(}\mathsf{obj}_d\texttt{(?O\#f(?X}_1 \ldots \texttt{?X}_{n_1}\texttt{))} \ldots \mathsf{obj}_d\texttt{(?O\#f(?X}_1 \ldots \texttt{?X}_{n_k}\texttt{))}\texttt{)}$$

Case 5: If ω *of the form* `f(...)` *has no OID but* m *tuples,* $m > 1$*,* $\mathsf{obj}_d(\phi, \omega) =$ `Exists ?O` $\mathsf{obj}_d\texttt{(?O\#f(...))}$*, where* `?O` *is a fresh variable in the query.*

Definition 8. *(Static/dynamic objectification) Let* obj_s *be a transformation chosen from* $\{\mathsf{obj}_{s^{\neq}}, \mathsf{obj}_{s^{=}}\}$ *and* ϕ *be a KB. The static/dynamic objectification* $\mathsf{obj}_{s+d}(\phi, \omega)$ *of an atom is* $\mathsf{obj}_s(\phi, \omega)$ *if* $\mathsf{Pred}(\omega) \in \mathsf{PredKB}_{NR}(\phi)$*, or* $\mathsf{obj}_d(\phi, \omega)$ *if* $\mathsf{Pred}(\omega) \in \mathsf{PredKB}_R(\phi)$*.* obj_{s+d} *is extended for the KB* ϕ *and other formulas, using Definition 3, if the following conditions are satisfied:*

(i) For every KB clause, universal variables occurring in its conclusion must also occur in its premise (e.g., prohibiting non-ground facts).
(ii) There does not exist a predicate variable or an untyped (`Top`-typed) atom with positional arguments in the premise of a rule or in the top-level query.

In Example 1, conditions (i) and (ii) are satisfied so that static/dynamic objectification can be applied. In the KB, the predicate `_work` is relational while the others are non-relational. Thus, all oidless `_work` atoms are kept unchanged while for the other oidless atoms OIDs are introduced via one of the two static methods. The query `?O#_work(?P ?C ?J)` is rewritten into the conjunction `And(_work(?P ?C ?J) ?O=_oidcons(_work ?P ?C ?J))` according to Case 3 of Definition 7.

In the following we will show the correctness of $\mathsf{obj}_{s^{=}+d}$ and $\mathsf{obj}_{s^{\neq}+d}$, which correspond to the two obj_{s+d} versions using $\mathsf{obj}_{s^{=}}$ and $\mathsf{obj}_{s^{\neq}}$ for the static part.

Lemma 4. *Let* ϕ *be a KB and* q *be a query without* `_oidcons` *and let* ϕ *and* q *comply to the above conditions (i) and (ii). Then* $\mathsf{obj}_{s^{=}+d}$ *has the properties (1) and (2) of Lemma 1 with respect to* ϕ *and* q*.*

Proof. (Sketch) If $\mathcal{I}$ is a counter-model for $\phi \models q$, we can define $\mathcal{I}'$ to be modified from $\mathcal{I}$ by adding interpretation of `_oidcons` so that for every relationship ω where $\mathsf{Pred}(\omega) \in \mathsf{PredKB}_R(\phi)$, being of the form $\texttt{f(t}_1 \ldots \texttt{t}_n\texttt{)}$, $TVal_{\mathcal{I}'}(\texttt{\_oidcons(f t}_1 \ldots \texttt{t}_n\texttt{)\#f(t}_1 \ldots \texttt{t}_n\texttt{)}) = \mathbf{t}$. This is done by making $\boldsymbol{I}(\texttt{\_oidcons(f t}_1 \ldots \texttt{t}_n\texttt{)})$ the domain element that represents the OID of ω. We can prove that $\mathcal{I}'$ fulfills the conditions 1.a) and 1.b) for every KB / query atom ω and $\boldsymbol{I}_{\mathrm{V}} \in \mathsf{M}(\mathsf{Var}(\omega), \mathcal{I}.\boldsymbol{D})$ performing a case-by-case analysis for ω. We can also prove that $\mathsf{obj}_{s^{=}+d}$ satisfies conditions 2) by choosing $\mathcal{I} = \mathcal{I}'$ and doing a case-by-case analysis. □

Lemma 5. *Let ϕ be a KB and q be a query without* `_oidcons`*, let ϕ and q comply to the above conditions (i) and (ii), and let $\phi' = \mathsf{obj}_{s^{=}+d}(\phi)$ and $q' = \mathsf{obj}_{s^{=}+d}(\phi, q)$. Then ϕ' and q' fulfill the conditions of Lemma 2.*

Proof. (Sketch) If $\mathcal{I}''$ is a counter-model for $\phi' \models_{\text{-}o} q'$, then $\mathcal{I}'' \models_{\text{-}o} \phi'$ and $\mathcal{I}'' \not\models_{\text{-}o} q'$. We modify $\mathcal{I}''$ into a semantic structure $\mathcal{I}'$ that guarantees the objectification restriction using the following redefinitions:

a) For every oidful atom ω' in ϕ' and q', we redefine $TVal_{\mathcal{I}'}(\omega)$ for its oidless form ω to be the same as $TVal_{\mathcal{I}''}(\omega')$.
b) For every oidless atom ω' in ϕ' and q' of the form `f(`$\mathtt{t_1} \ldots \mathtt{t_n}$`)`, let $\mathsf{o}(\omega')$ be the virtual OID `_oidcons(f` $\mathtt{t_1} \ldots \mathtt{t_n}$`)`; we redefine $TVal_{\mathcal{I}'}(\mathsf{o}(\omega')$`#f(`$\mathtt{t_1} \ldots \mathtt{t_n}$`))`, $TVal_{\mathcal{I}'}(\mathsf{o}(\omega')$`#f`$)$, and $TVal_{\mathcal{I}'}(\mathsf{o}(\omega')$`#Top(`$\mathtt{t_1} \ldots \mathtt{t_n}$`))` to be the same as $TVal_{\mathcal{I}''}(\omega')$. All other atoms that have a component $\mathsf{o}(\omega')$ are required to evaluate to false.

If $\mathsf{Pred}(\omega') \in \mathsf{PredKB}_{NR}(\phi')$, definition a) applies, and the redefinition would not affect the truth evaluation for formulas in ϕ' and q' since ω does not occur in ϕ'.

If $\mathsf{Pred}(\omega') \in \mathsf{PredKB}_{R}(\phi')$, definition b) applies, and the objectification restriction is guaranteed. We will show that the redefinition would not affect the truth evaluation for top-level formulas in ϕ' and q'. Since $\mathtt{f} \in \mathsf{PredKB}_{R}(\phi')$, it does not occur in an oidful or multi-tuple atom. So $\mathcal{I}'$ preserves the truth evaluation of all atoms with the predicate `f`. We discuss different top-level formulas τ in ϕ':

- For τ being a ground fact, if $\mathsf{Pred}(\tau) \in \mathsf{PredKB}_{R}(\phi')$ then $TVal_{\mathcal{I}'}(\tau) = TVal_{\mathcal{I}''}(\tau) = \mathbf{t}$. Otherwise, its components cannot be interpreted to any $\mathsf{o}(\omega')$ so $TVal_{\mathcal{I}'}(\tau) = TVal_{\mathcal{I}''}(\tau) = \mathbf{t}$ still holds.
- τ being a non-ground fact is disallowed by condition (i).
- For τ being a rule `Forall` $\mathtt{?X_1} \ldots \mathtt{?X_n}$ `(`τ_1 `:-` τ_2`)`, for every $\boldsymbol{I}_{\mathrm{V}} \in \mathsf{M}(\{\mathtt{?X_1}, \ldots, \mathtt{?X_n}\}, \mathcal{I}''.\boldsymbol{D})$, if for some i, $\boldsymbol{I}_{\mathrm{V}}(\mathtt{?X_i})$ interprets to any $\mathsf{o}(\omega')$, then it can be shown that $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}})}(\tau_2) = \mathbf{f}$ always holds based on Conditions (i) and (ii). So $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}})}(\tau_1 \texttt{ :- } \tau_2) = \mathbf{t}$. Otherwise, $TVal_{\mathsf{v}(\mathcal{I}', \boldsymbol{I}_{\mathrm{V}})}(\tau_1 \texttt{ :- } \tau_2) = TVal_{\mathsf{v}(\mathcal{I}'', \boldsymbol{I}_{\mathrm{V}})}(\tau_1 \texttt{ :- } \tau_2) = \mathbf{t}$. Hence $TVal_{\mathcal{I}'}(\tau) = \mathbf{t}$.

$\mathcal{I}'$ can also be verified to conform to the subclass and slotribution/tupribution restrictions. So $\mathcal{I}'$ is a counter-model for $\phi' \models q'$, and the lemma holds. □

Theorem 3. $\mathsf{obj}_{s^{=}+d}$ *is an objectification transformation with respect to a set of KBs Φ and a set of queries Q that do not use* `_oidcons` *and satisfy conditions (i) and (ii).*

Proof. By Lemmas 4, $\phi \models q$ iff $\mathsf{obj}_{s^{=}+d}(\phi) \models \mathsf{obj}_{s^{=}+d}(\phi, q)$. By Lemmas 5 and 5, $\mathsf{obj}_{s^{=}+d}(\phi) \models \mathsf{obj}_{s^{=}+d}(\phi, q)$ iff $\mathsf{obj}_{s^{=}+d}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^{=}+d}(\phi, q)$. Hence $\phi \models q$ iff $\mathsf{obj}_{s^{=}+d}(\phi) \models_{\text{-}o} \mathsf{obj}_{s^{=}+d}(\phi, q)$ and the theorem is proved. □

Theorem 4. $\mathsf{obj}_{s^{\neq}+d}$ *is an objectification transformation with respect to KBs Φ and queries Q that do not use constants* `_oidcons`*,* `_1`*,* `_2`*, ... and satisfy conditions (i) and (ii).*

Proof. (Sketch) This can be proved similarly to Theorem 2 by first proving a corresponding version of Lemma 3 for $\mathsf{obj}_{s^{\neq}+d}$ and $\mathsf{obj}_{s^{=}+d}$. □

4 Maximal Unnesting

In this section we first discuss the syntax and semantics of embedded psoa terms and then formally define the unnesting transformation of nested atomic formulas. Unnesting is maximal in the sense that it can extract atoms not only from other atoms but also from expressions, which may themselves be embedded at any level.

4.1 Syntax and Semantics of Embedded Psoa Terms

In PSOA RuleML, embedded psoa terms can be semantically classified into expressions or atoms, and syntactically classified into oidful or oidless terms. We will discuss different combinations of the two dimensions in the following.

Expressions A psoa expression is a psoa term interpreted as a function application. Like in classical logic and logic programming languages, we require a (psoa) expression to be oidless, having the form `f(...)`, with `f` acting as a function, hence leading to an arbitrary value. The expression cannot be given an OID and have the form `o#f(...)`, because this would make `f` act the class of `o`, hence lead to a truth value.

The semantics for expressions uses the $\boldsymbol{I}$ and $\boldsymbol{I}_{\text{psoa}}$ mappings defined in Section 3.1. However, because these expressions must be oidless, $\boldsymbol{I}_{\text{psoa}}$ interprets $\boldsymbol{I}(\texttt{f})$ into a semantic function that takes the empty set for oidless psoa terms, besides a bag of tuples and a bag of slots, and returns an arbitrary domain element. The lack of an OID for an expression also prevents its slotribution/tupribution, thus avoiding that its arbitrary value gets 'distributed over' the resulting conjunction of single-slot/tuple expressions or 'absorbed by' its truth value.[4]

Atoms A psoa atom is a psoa term interpreted as a predicate application. Embedded atoms have been widely used in object-centered languages such as RDF, N3 [8], and Flora-2/F-logic [9] as a shorthand notation. An atomic formula containing an embedded atom can be unnested into a conjunction of trimmed formulas, which will be illustrated in Section 4.2.

In RDF and N3, embedded atoms are oidless and are 'objectified-while-embedded' using blank nodes, while in Flora-2/F-logic they are oidful. Because in PSOA/PS the set `Const` of constants contains both functions and predicates (cf. Section 2), an oidless embedded atom cannot be distinguished from an expression through the alphabet, hence we use the syntactic distinction between an embedded oidless term for an expression and an embedded oidful term for an atom. The distinction applies only to embedded psoa terms but not to top-level terms, where both oidful and oidless forms are interpreted as atoms. To indicate an RDF/N3-like embedded blank node, the atom can use an explicit OID,

[4] Relfun's *valued conjunctions* of functional-logic expressions only retain the value of the right-most, '`&`'-prefixed conjunct: `http://www.relfun.org`.

thus: (1) `_#f(...)` in ground facts, where '`_`' is the fresh-constant generator; (2) `Exists ?O (... ?O#f(...) ...)` in non-ground facts and rule conclusions; and (3) `?#f(...)` in queries / rule premises, where '`?`' is the anonymous variable.

4.2 Unnesting Transformation for Embedded Psoa Atoms

In this section we will define the unnesting transformation $\mathsf{Unnest}(\alpha)$ for a given atomic formula α, which is extended from the definition in [10]. Before performing $\mathsf{Unnest}(\alpha)$, anonymous OID constants and variables in α need to be eliminated, because they cannot be used as co-references for the same constant/variable in two separate formulas. The elimination can be done by replacing '`_`' with a fresh constant and '`?`' with a fresh variable.

In the following we give the definition of $\mathsf{Unnest}(\alpha)$ based on the recursive Atoms. Here, $\mathsf{Oid}(t)$ denotes the OID of an oidful term t. Also, $\mathsf{Parts}(t)$ denotes the set of top-level components of an atomic formula or a term t, including, optionally, $\mathsf{Oid}(t)$, its positional arguments, slot names, slot fillers, as well as its predicate/function.

$$\mathsf{Unnest}(\alpha) ::= \texttt{And}(\sigma_1 \ldots \sigma_n) \quad \text{s.t. } \{\sigma_1, \ldots, \sigma_n\} = \cup_{t \in \mathsf{Parts}(\alpha)} \mathsf{Atoms}(t) \bigcup \{\mathsf{Trim}(\alpha)\}$$

$$\mathsf{Atoms}(t) ::= \begin{cases} \emptyset & t \text{ is a simple term} \\ \cup_{s \in \mathsf{Parts}(t)} \mathsf{Atoms}(s) & t \text{ is oidless (expression)} \\ \cup_{s \in \mathsf{Parts}(t)} \mathsf{Atoms}(s) \bigcup \{\mathsf{Trim}(t)\} & t \text{ is oidful (atom)} \end{cases}$$

$$\mathsf{Trim}(t) ::= \text{Term/Formula obtained by replacing every } s \in \mathsf{Parts}(t) \text{ in } t \text{ with } \mathsf{Retain}(s)$$

$$\mathsf{Retain}(t) ::= \begin{cases} t & t \text{ is a simple term} \\ \mathsf{Trim}(t) & t \text{ is oidless (expression)} \\ \mathsf{Retain}(\mathsf{Oid}(t)) & t \text{ is oidful (atom)} \end{cases}$$

$\mathsf{Unnest}(\alpha)$ is a conjunction of formulas σ_i without embedded atoms. Each σ_i is a trimmed version of the top-level formula α or of some embedded psoa atom. The set $\mathsf{Atoms}(t)$ contains each σ_i trimmed from an atom embedded in t or t itself. It is constructed by recursively traversing through each component $s \in \mathsf{Parts}(t)$, collecting $\mathsf{Atoms}(s)$ into $\mathsf{Atoms}(t)$, and then adding $\mathsf{Trim}(t)$ to $\mathsf{Atoms}(t)$ if t is oidful, which indicates that t is an atom. The transformation $\mathsf{Trim}(t)$ splits off all embedded atoms from t and leaves behind its 'ultimate' OID for each of them. It is constructed by replacing each $s \in \mathsf{Parts}(t)$ with $\mathsf{Retain}(s)$, which defines the left-behind term for each embedded term s.

In the following we use an example to explain the unnesting transformation. Let the input formula α be `o1#c(p->f(o2#c#d))`. Note that '`#`' is left-associative, hence the embedded atom `o2#c#d` is interpreted to have the OID `o2#c` and the class `d`. The conjuncts of $\mathsf{Unnest}(\alpha)$ are constructed as follows:

$$\begin{aligned} \{\sigma_1, \ldots, \sigma_n\} &= \cup_{t \in \mathsf{Parts}(\alpha)} \mathsf{Atoms}(t) \bigcup \{\mathsf{Trim}(\alpha)\} \\ &= (\mathsf{Atoms}(\texttt{o1}) \cup \mathsf{Atoms}(\texttt{c}) \cup \mathsf{Atoms}(\texttt{p}) \cup \mathsf{Atoms}(\texttt{f(o2\#c\#d)})) \bigcup \{\mathsf{Trim}(\alpha)\} \\ &= \mathsf{Atoms}(\texttt{f(o2\#c\#d)}) \bigcup \{\mathsf{Trim}(\alpha)\} \\ &= (\mathsf{Atoms}(\texttt{f}) \cup \mathsf{Atoms}(\texttt{o2\#c\#d})) \bigcup \{\mathsf{Trim}(\alpha)\} \end{aligned}$$

$$
\begin{aligned}
&= \mathsf{Atoms}(\texttt{o2\#c\#d}) \bigcup \{\mathsf{Trim}(\alpha)\} \\
&= (\mathsf{Atoms}(\texttt{o2\#c}) \cup \mathsf{Atoms}(\texttt{d}) \cup \{\mathsf{Trim}(\texttt{o2\#c\#d})\}) \bigcup \{\mathsf{Trim}(\alpha)\} \\
&= \mathsf{Atoms}(\texttt{o2\#c}) \bigcup \{\mathsf{Trim}(\texttt{o2\#c\#d})\} \bigcup \{\mathsf{Trim}(\alpha)\} \\
&= (\mathsf{Atoms}(\texttt{o2}) \cup \mathsf{Atoms}(\texttt{c}) \cup \{\mathsf{Trim}(\texttt{o2\#c})\}) \bigcup \{\mathsf{Trim}(\texttt{o2\#c\#d})\} \bigcup \{\mathsf{Trim}(\alpha)\} \\
&= \{\mathsf{Trim}(\texttt{o2\#c}), \mathsf{Trim}(\texttt{o2\#c\#d}), \mathsf{Trim}(\alpha)\}
\end{aligned}
$$

The Trim transformations work as follows, using the recursive Retain transformation:

$$
\begin{aligned}
\mathsf{Trim}(\texttt{o2\#c}) &= \mathsf{Retain}(\texttt{o2})\#\mathsf{Retain}(\texttt{c}) = \texttt{o2\#c} \\
\mathsf{Trim}(\texttt{o2\#c\#d}) &= \mathsf{Retain}(\texttt{o2\#c})\#\mathsf{Retain}(\texttt{d}) \\
&= \mathsf{Retain}(\mathsf{Oid}(\texttt{o2\#c}))\#\texttt{d} = \mathsf{Retain}(\texttt{o2})\#\texttt{d} = \texttt{o2\#d} \\
\mathsf{Trim}(\alpha) &= \mathsf{Trim}(\texttt{o1\#c(p->f(o2\#c\#d))}) \\
&= \mathsf{Retain}(\texttt{o1})\#\mathsf{Retain}(\texttt{c})(\mathsf{Retain}(\texttt{p})\texttt{->}\mathsf{Retain}(\texttt{f(o2\#c\#d)})) \\
&= \texttt{o1\#c(p->}\mathsf{Retain}(\texttt{f})(\mathsf{Retain}(\texttt{o2\#c\#d}))) \\
&= \texttt{o1\#c(p->f(}\mathsf{Retain}(\mathsf{Oid}(\texttt{o2\#c\#d})))) \\
&= \texttt{o1\#c(p->f(}\mathsf{Retain}(\texttt{o2\#c}))) \\
&= \texttt{o1\#d(p->f(o2))}
\end{aligned}
$$

Hence, the unnesting $\mathsf{Unnest}(\alpha)$ results in `And(o2#c o2#d o1#c(p->f(o2)))`.

The unnesting transformation has been implemented in the latest release of PSOATransRun 1.1 using a separate ANTLR tree walker.

5 Conclusions

This paper discusses advanced objectification and unnesting transformations for the PSOA RuleML language as well as a novel model-theoretic semantics.

The refined semantics is introduced to allow a direct interpretation of oidless psoa terms. It includes the objectification restriction to establish the equivalence between an oidless atom and its existentially objectified form. Based on the new semantics, a systematics of three objectification transformations is defined, whose correctness is proved. A novel static/dynamic objectification approach for oidless atoms is introduced, which is minimal in that it generates as few explicit OIDs as possible, instead constructing virtual OIDs as query variable bindings. This approach provides better efficiency for the PSOATransRun implementation by allowing direct use of the underlying Prolog engine.

The unnesting transformation is formalized to decompose nested atomic formulas into equivalent conjunctions before applying the model-theoretic semantics. Unnesting is maximal in that it can recursively extract oidful atoms – leaving behind their OIDs – not only from other atoms but also from expressions, which may themselves be embedded at any level. Since embedded oidless terms are interpreted as expressions (usable, e.g., as 'passive' data constructors), rather than as atoms to be 'objectified-while-embedded', for embedded

atoms OIDs need to be explicitly provided to enable unnesting. The unnesting transformation has been implemented in PSOATransRun 1.1.

Future work includes exploring further optimizations for objectification and complementing the unnesting transformation with a flattening transformation for extracting 'active' expressions from both atoms and expressions. For this, PSOATransRun's flattening of expressions calling built-in functions can be easily transferred to expressions with equality-defined functions.

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. Boley, H.: PSOA RuleML: Integrated Object-Relational Data and Rules. In Faber, W., Paschke, A., eds.: Reasoning Web. Web Logic Rules (RuleML 2015) - 11th Int'l Summer School 2015, Berlin, Germany, July 31- August 4, 2015, Tutorial Lectures. Volume 9203 of LNCS., Springer (2015)
6. Zou, G., Boley, H.: PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog. In: Proc. 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany. Lecture Notes in Computer Science, Springer (August 2015)
7. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. Journal of Emerging Technologies in Web Intelligence **4**(2) (November 2010) 343–353
8. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A Logical Framework For the World Wide Web. Theory and Practice of Logic Programming (TPLP) **8**(3) (May 2008)
9. Kifer, M., Yang, G., Wan, H., Zhao, C.: $\mathcal{E}$RGO^{Lite} (a.k.a. $\mathcal{F}$lora-2): User's Manual, v1.1 (2015) `http://flora.sourceforge.net/docs/floraManual.pdf`.
10. Boley, H., Kifer, M.: RIF Basic Logic Dialect (Working Draft) (October 2007) W3C Working Draft, `https://www.w3.org/TR/2007/WD-rif-bld-20071030/`.