Perspectival Knowledge in PSOA RuleML: Representation, Model Theory, and Translation

Harold Boley, Gen Zou

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

Abstract. In Positional-Slotted Object-Applicative (PSOA) RuleML, a predicate application (atom) can have an Object IDentifier (OID) and descriptors that may be positional arguments (tuples) or attribute-value pairs (slots). PSOA RuleML 1.0 extends earlier versions by explicitly specifying for each descriptor whether it is to be interpreted under the perspective of the predicate in whose scope it occurs. This perspectivity dimension refines the design space between oidless, positional atoms (relationships) and oidful, slotted atoms (frames): While relationships use only a predicate-scope-sensitive (predicate-dependent) tuple and frames use only predicate-scope-insensitive (predicate-independent) slots, PSOA RuleML 1.0 uses a systematics of orthogonal constructs also permitting atoms with (predicate-)independent tuples and atoms with (predicate-)dependent slots. This supports advanced data and knowledge representation where, e.g., a slot attribute can have different values depending on the predicate. PSOA thus extends classical object-oriented multi-membership and multiple inheritance. Based on objectification, PSOA laws are explicated: Besides unscoping and centralization, the semantic restriction and implemented transformation of describution permits the rescoping of one atom's independent descriptors to another atom with the same OID but a different predicate. For inheritance, default descriptors are realized by rules. On top of a basic metamodel and a new Grailog visualization, PSOA's use of the atom systematics for facts, queries, and rules is explained. The presentation and (XML-)serialization syntaxes of PSOA RuleML 1.0 are introduced. Its model-theoretic semantics is formalized by extending the interpretation functions to accommodate dependent descriptors. The open PSOATransRun system since Version 1.3 realizes PSOA RuleML 1.0 by a translator to runtime predicates, including for dependent tuples (prdtupterm) and slots (prdsloterm). Our tests show efficiency advantages of dependent and tupled modeling.

1 Introduction

In advanced Artificial Intelligence (AI) Knowledge Bases (KBs), the related notions of "context" and "perspective" are both called for. While a *context* mechanism [1] allows to partition the clauses of a KB, *perspective*, as introduced here, allows to describe the same Object IDentifier (OID) differently with multiple clause conclusions – e.g., predicate applications (atoms) used as facts – having different predicates (cf. Fig. 1's OID `John` with predicates `Teacher`/`TA`/`Student`).

A form of *contextualized* KBs has been available in Positional-Slotted Object-Applicative RuleML (PSOA RuleML) [2–5][1] as realized by PSOATransRun since Version 1.2, allowing (1) constants that are local to each KB and (2) a merging `Import` statement that will rename apart local constants from multiple KBs. Reciprocally, the current paper focuses on the topics of representation, model theory, and translation for *perspectival* KBs (facts and rules) and queries as now explicitly available in PSOA RuleML 1.0 realized since PSOATransRun 1.3.

Let us begin with a discussion of our notion of perspective by employing a novel systematics for **p**ositional-**s**lotted **o**bject-**a**pplicative (**psoa**)[2] atoms, which constitutes the basic PSOA RuleML metamodel of Fig. 5 in Appendix A. Besides their use as data facts and – often with variables – as queries, psoa atoms occur in rule conclusions and conditions (because of the wide use of these formulas, when the intent is obvious we will frequently shorten "psoa atom" to "atom").

PSOA RuleML permits an *atom* to apply a predicate – possibly identified by an OID typed by the predicate – to a bag (multiset) of tupled descriptors, each representing an argument sequence, and to a bag of slotted descriptors, each representing an attribute-value pair. Further extending these descriptor and OID dimensions by the perspectivity dimension, PSOA RuleML 1.0 atoms will be visualized and explained with an oidful example (having one shared OID, shown as a large box).

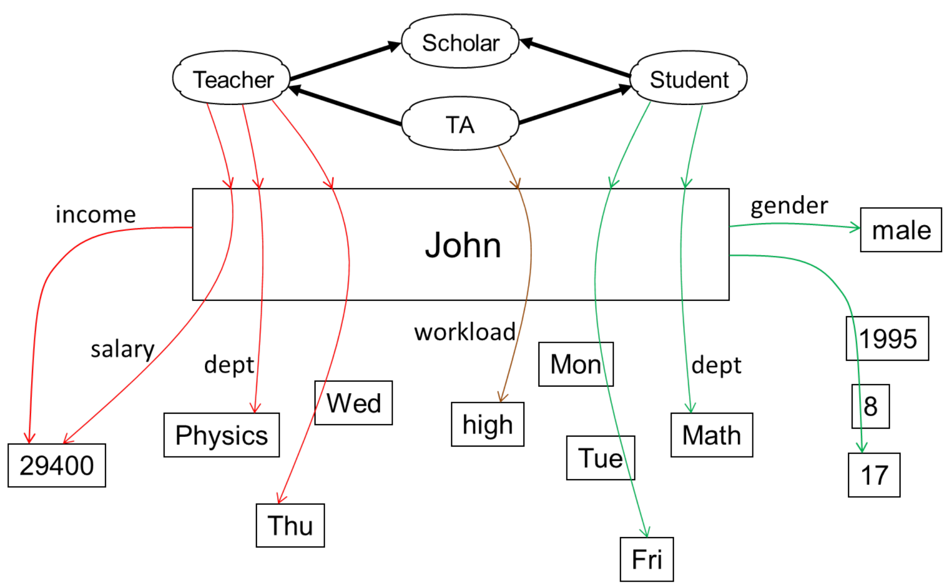

Fig. 1. Rich TA example of perspeneutral-perspectival facts in Grailog: OID John described independently and dependent on predicates Teacher, TA, Student.

[1] `http://wiki.ruleml.org/index.php/PSOA_RuleML`

[2] We use the upper-cased "PSOA" as a qualifier for the language and the lower-cased "psoa" for its terms, i.e. atoms or expressions, and parts of its terms, e.g. descriptors.

We introduce "Rich TA" as a running example of taxonomy-augmented data. Fig. 1 enriches the classical Teaching Assistant (TA) example for multiple inheritance [6] and multi-membership in object-oriented programming languages and databases. Our enriched AI-KB version will illustrate where PSOA 1.0's expressivity increases compared to classical related work [6, 7]:

- Permission of perspectives without diminishing the uniform notion of "class" by an additional notion of (an individual playing a) "role" in the sense of [7][3], which would hinder uniform knowledge representation for, e.g., sorted logic, Description Logic [8], F-logic [9], N3Logic [10], as well as ConceptBase's [11] and PSOA's logics, also suffering from the lack of a clear "class"/"role" boundary when descending a taxonomy like `Top`, ..., `Person` ("class"), `Scholar` ("class" that could be a "role" with a sibling `Vegetarian` "role"), `Teacher` or `Student` (sibling "roles" in [7]), and `TA`.
- Explicit distinction of perspeneutral and perspectival atoms – based on, respectively, predicate-independent and predicate-dependent descriptors – so that the same OID, here `John`, via some psoa atoms – e.g. used as facts – can be seen under no perspective (equivalently, under the vacuous `Top`, i.e. *root*, perspective) and via others under one or more (non-`Top`) perspectives, here `Teacher`, `TA`, and `Student`.
- Clausal extensibility of factual data by rule (*conclusion* `:-` *condition*) knowledge – possibly, as in Section 4.1, Fig. 2, centered on an OID variable – for inferencing, such as to integrity-check existing data or to derive new data from it (e.g., rather than storing a `workload` fact for `John`, as in Fig. 1: deriving it, again perspectivally for any `TA`, based on other facts, as in Fig. 2).

Constituting a portion of what can be regarded as an individual's (`John`'s) "Perspectival Knowledge Graph", Fig. 1 generalizes earlier Grailog [12] visualizations of PSOA KBs [3][4] for accommodating the perspectivity dimension.

In the upper part, it shows a diamond-shaped taxonomy of four oval-like predicates[5] – `Scholar`, `Teacher`, `Student`, and `TA` – connected by heavy arcs understood to be implicitly labeled with `subpredicate`, where `TA` – connecting to both `Teacher` and `Student` – exemplifies multiple inheritance.

In the lower part, showing directed-(hyper)graph-visualized data, three of these predicates – all except `Scholar` – are used to spawn dependent descriptors for perspectival representation. For this, it uses hyperarcs starting with a predicate labelnode, e.g. `Teacher`, pointing to – with an intermediate arrow head –

[3] To be distinguished from the notion of "role" in the sense of "property" as used in Description Logic (standardized as OWL 2: `http://www.w3.org/TR/owl-overview`), "object-holes" in Object-Role Modeling (`http://www.orm.net/pdf/ormwhitepaper.pdf`), and "association ends" in the Object Constraint Language (`https://st.inf.tu-dresden.de/files/general/OCLByExampleLecture.pdf`).

[4] `http://wiki.ruleml.org/index.php/Grailog#Family_Example`

[5] PSOA's taxonomies represent predicate subsumptions much like class subsumptions, where the root predicate `Top` is always understood to subsume all other predicates. Thus, subpredicate arcs and facts linking from non-`Top` subtaxonomy roots to `Top` are not normally shown in, respectively, taxonomy DAGs such as Fig. 1 (with subtaxonomy root `Scholar`) and their symbolic forms such as `(KB1)-(KB3)` of Section 2.1.

and cutting through an optional OID node, e.g. `John`, and cutting through any further nodes in sequence before pointing to the last node, with all nodes being rectangular. Optional labels on these descriptor hyperarcs, e.g. `dept`, are slot names, thus distinguishing slot hyperarcs from tuple hyperarcs. E.g., the `Teacher` hyperarcs indicate, from right to left, that – under the perspective of being a `Teacher` – `John` is characterized by (a length-2 tuple, in standard chronological order, for) `Wed` followed by `Thu`, is in the `dep`(artmen)`t` of `Physics`, and has a `salary` of `29400`. On the far left, a labeled arc, starting directly at the OID, records `John`'s (total) `income` (also) as `29400` – independently of, e.g., the `Teacher`, `TA`, and `Student` perspectives.

Besides various other groupings, two complementary **methods of creating atoms** from these descriptor (hyper)arcs are for **(i)** *single-descriptor atoms*, each made of one (hyper)arc, and for **(ii)** *perspectival-concentrated atoms*, each made of all the hyperarcs starting with a common predicate and continuing with a common OID, as well as of zero or more (hyper)arcs starting only at this OID. Using the descriptor (hyper)arcs discussed so far, the unique, atom-size-minimizing method (i) creates four atoms, where the red color is immaterial, while the non-unique, atom-count-minimizing method (ii) creates one atom, where the red color serves for large-atom chunking of these descriptors, chosen to include the predicate-independent `income` slot (but no other independent descriptors).[6]

The remaining (hyper)arcs are similar except that in the – green-grouped – atom – under the perspective of the start labelnode `Student` – `John` is characterized by, e.g., (a length-3 tuple for) `Mon` followed by `Tue` and `Fri`, and that – independently of predicates (thus applicable to `John` in an 'absolute' manner) – he is characterized by (a length-3 tuple for) `1995` followed by `8` and `17`.[7] Generally, method (i) ignores any – here, three – colors while method (ii) uses them to indicate grouping of descriptors into atoms.

Since `John` is represented as an OID node pointed to and cut through by hyperarcs starting with three different predicates – `Teacher`, `TA`, and `Student` – he is involved under these different perspectives. The "pointing to" also entails a multi-membership of `John` in three predicates, here acting as classes. Abbreviating "under the perspective of" to "as a" or "asa", we can generally say that "asa entails isa", where the "isa" of classical Semantic Nets is often called "is member of" in Semantic Technologies. Notice that for perspectival (data and) knowledge, multi-membership cannot be reduced to multiple inheritance with a newly introduced common subpredicate such as `TA` underneath `Teacher` and `Student`: The very notion of perspective requires that an individual such as `John` stays member of the predicates under whose perspectives it is represented.

[6] The single-descriptor atoms according to method (i) can be obtained from arbitrary atoms by the describution **law/transformation** of Sections 3.1/6 based on a semantic **restriction** in Section 5. The zero-or-more-descriptor atoms according to method (ii) correspond to those obtained by centralization (cf. Section 3.1).

[7] A tuple (hyperarc) can be seen as a shortcut for a tuple-valued slot (hyperarc) having the `Top`-predicate-complementing implicit 'vacuous' name (label) `prop`(erty), which could be specialized here to slot names like `dop` – for the (dependent) days-of-presence of a scholar – and `dob` – for the (independent) date-of-birth of a person. A multi-tuple psoa term can expand its tuples to (in/dependent) multi-(tuple-)valued `prop` slots.

This section introduced the novel perspectivity dimension as part of a systematics with other dimensions for atoms in PSOA RuleML 1.0, illustrated by a visualized three-perspective example. The subsequent Section 2 will continue with symbolic perspectival fact and rule representation and reasoning (through querying) in the abridged syntax of PSOA RuleML 1.0. This will be followed, in Section 3, by equivalence laws for PSOA knowledge, with a subsection on default descriptors realized via default rules. The paper will then proceed, in Section 4, to the appropriately augmented unabridged presentation syntax and the serialization syntax of PSOA RuleML 1.0. Next, in Section 5, it will revise the parts of the model-theoretic semantics that are key to incorporating in/dependent descriptors. Sections 4 and 5 establish PSOA as a logic. Then, in Section 6, the paper will discuss the PSOATransRun implementation of in/dependent descriptors, translating PSOA RuleML knowledge bases and queries to TPTP (PSOA2TPTP) or Prolog (PSOA2Prolog); test results will be shown. Finally, Section 7 will give conclusions and indicate directions of future work. Appendix A will expound on the metamodel, applying it to the TA example. The examples of this paper bridge between theory and practice: They have been tested in the PSOATransRun instantiation targeting XSB Prolog, and readers are encouraged to try and vary some of them, starting with the README[8].

2 Foundations of PSOA Knowledge Representation

In this section we discuss the foundations of knowledge representation in PSOA RuleML 1.0, introducing a concrete syntax to formalize KBs according to the metamodel of Appendix A, illustrated by Fig. 1 of Section 1. We further give positive and negative query examples that provide informal proof-theoretic semantics in preparation for the formal model-theoretic semantics in Section 5. We also discuss modeling approaches to reduce dependent to independent slots.

2.1 Formal Facts and Queries

To formalize the notions of Section 1, we complement the *visualization syntax* used there by a *presentation syntax*, developing the one in [3]. This abridges the unabridged syntax used by the PSOATransRun system (cf. Section 4.1), omitting the `RuleML` and `Assert` wrappers from KBs as well as the "`_`" prefix from local constants such as `_John`, optional except for (objectification) integers.[9]

[8] `http://psoa.ruleml.org/transrun/1.3.1/local/`

[9] For example, any of the three Fig. 1 hyperarcs starting with the predicate `Teacher` and pointing – via an intermediate arrow head – to the OID `John` can be symbolically represented as a *membership* of `John` in (indicated by "`#`") `Teacher` by the oidful empty atom `John#Teacher()`, often shortened to `John#Teacher`, e.g. as a fact, query, or in a (conclusion or condition of a) rule. The corresponding oidless empty atom `Teacher()` will be *objectified* [2, 3, 5] by PSOATransRun, e.g. when used as a fact yielding `_`j`#Teacher()`, $j \geq 1$, where "`_`j" is generated as the fresh local positive-integer OID (employed as a Skolem constant, similarly to an RDF blank node) having the minimal j. Objectification works the same for non-empty atoms, since it does not involve their (dependent or independent) descriptors. The "`_`" of "`_`j" can never be omitted. For any integer j, `_`$j \neq j$. No other numbers are used with a "`_`" prefix.

The **first symbolic representation** of the entire iconic Fig. 1 is shown as a PSOA RuleML KB of atomic ground (variableless) facts below such that the ground-atom colors correspond to the (hyper)arc colors in Fig. 1, except that gray for Top is new:

```
                                                     % (KB1)
Teacher##Scholar                                       % Taxonomy
Student##Scholar
TA##Teacher
TA##Student
John#TA(workload+>high)                                % Data (i) Fact 1
John#Teacher(+[Wed Thu])                               %          Fact 2
John#Teacher(dept+>Physics)                            %          Fact 3
John#Teacher(salary+>29400)                            %          Fact 4
John#Student(+[Mon Tue Fri])                           %          Fact 5
John#Student(dept+>Math)                               %          Fact 6
John#Top(-[1995 8 17])   % Top abstracted from Student            Fact 7
John#Top(gender->male)   % Top abstracted from Student            Fact 8
John#Top(income->29400)  % Top abstracted from Teacher            Fact 9
```

In (KB1)'s upper four facts, representing the TA-diamond taxonomy part of Fig. 1, the "##" infix indicates the binary subpredicate relation.

In Data (i) Facts 1 to 9, the data part constituting the rest of Fig. 1 is represented according to Section 1's method (i) such that there are only single-descriptor atoms (i.e., according to a simplified version of Section 3.1's descributed normal form (KB3')). Particularly, in Data (i) Facts 7 to 9, the unique root predicate Top is employed, which keeps this symbolic form of the method (i) representation unique (by avoiding to choose from the non-Top predicates). These Top-typed atoms can also be regarded as untyped atoms, as often used in F-logic and RIF.

The dual "+" vs. "-" marks are uniformly used for, respectively, dependent vs. independent descriptors, leading to four kinds of descriptors (exemplified with the descriptors of some of (KB1)'s Student and Top atoms):

– For tuples, "+" vs. "-" are used as prefixes for the square brackets, yielding the syntaxes +[...] vs. -[...], e.g. +[Mon Tue Fri] vs. -[1995 8 17].[10]
– For slots, "+" vs. "-" are used as shafts of the infix arrows, yielding the syntaxes ...+>... vs. ...->..., e.g. dept+>Math vs. gender->male.[11]

Specifically, in each atom, a predicate, e.g. Student – possibly identified by an OID, e.g. John, typed by the predicate – may be "(...)"-applied to one such dependent or independent descriptor (tuple or slot), e.g. Student to dept+>Math.

According to the metamodel of Appendix A, (KB1)'s Data (i) Facts 1 to 6 – all with a dependent descriptor – are perspectival atoms, while its Data (i) Facts 7 to 9 – all with an independent descriptor – are perspeneutral atoms.

[10] In earlier PSOA versions, no prefix was used on any (square-)bracketed tuple, and for atoms with an explicitly bracketed or a non-bracketed tuple dependency was decided on the basis of their predicate being "relational" [5]. In PSOA 1.0, a prefix is used on every bracketed tuple, and a non-bracketed tuple is interpreted as dependent.

[11] In earlier PSOA versions, the "-" shaft was used for each arrow and the arrow-infixed slot was always interpreted as independent.

The **second symbolic representation** of Fig. 1 changes (only) the data part according to Section 1's method (ii) such that there are a single- and two multiple-descriptor atoms complying to Fig. 1's colors:

```
                                                          % (KB2)
. . .                                                       % Taxonomy
John#TA(workload+>high)                                     % Data (ii)
John#Teacher(+[Wed Thu] dept+>Physics salary+>29400 income->29400)
John#Student(+[Mon Tue Fri] -[1995 8 17] dept+>Math gender->male)
```

These lower three ground facts represent the data as perspectival-concentrated atoms in the – logically immaterial – color order "brown-red-green". Such atoms can arbitrarily distribute independent descriptors, e.g. moving one to the `TA` fact.

Generally, in each atom, a predicate, e.g. `Student`, possibly identified by an OID, e.g. `John`, typed by the predicate (acting as a class of the OID), can be applied to zero or more dependent and independent descriptors (tuples and slots). Here, `Student` is applied to four descriptors of all four kinds (in/dependent tuples/slots).

According to the metamodel of Appendix A, both of `(KB2)`'s last two facts are perspeneutral+perspectival psoa atoms. Another case is perspeneutral psoa atoms, having only independent descriptors, e.g. `John#Student(-[1995 8 17] gender->male)`. These, then, further specialize to psoa frames, perspeneutral atoms having only independent slots, e.g. `John#Student(gender->male)`. Such a frame atom corresponds to an F-logic-like typed frame, which is often – e.g. in W3C RIF [13] – rewritten to a conjunction of a membership and an untyped frame, e.g., in PSOA RuleML's presentation syntax, `And(John#Student John#Top(gender->male))`. Similarly, such a typed frame that happens to have just one independent slot in RDF corresponds to a KB of a typing triple and one slot triple; e.g., the above frame in simplified N-Triples syntax becomes

```
John rdf:type Student.
John gender male.
```

An issue with triples and untyped frames is that, by detaching the predicate[12] acting as a class from the OID, they cannot easily accommodate (predicate-) dependent slots, as provided, for example, by the special case of perspectival psoa atoms that have only dependent slots, e.g. `John#Student(dept+>Math)`.

`(KB2)`'s data part distributes the independent descriptors across the `Teacher` and `Student` atoms, in one of several possible ways according to Section 1's non-unique method (ii), where, e.g., the `TA` atom could also receive one, two, or all three independent descriptors. In the unique **method (iii)** all independent descriptors are extracted from form (ii) and collected in one perspeneutral atom (using the unique root predicate `Top`),[13] obtaining the following unique *perspectivity-concentrated form* of the data, where the colors are like in `(KB1)`:

[12] PSOA's notion of 'predicate' can be regarded as a generalization of, e.g., RDF's notion of 'class'. However, RDF's notion of (binary/dyadic) 'predicate' corresponds to RIF's and PSOA's notion of 'slot name'.

[13] If a non-`Top` predicate such as `Teacher` were used for collecting all independents, the meaning would not change (all descriptors are independent of any predicate) but uniqueness would be lost. Additionally, for the uniqueness of such symbolic forms, a canonical order of the bags of descriptors (tuples before slots, dependent before independent) and lexicographic order of the slots are normally used.

```
                                                  % (KB3)
. . .                                               % Taxonomy
John#TA(workload+>high)                             % Data (iii)
John#Teacher(+[Wed Thu] dept+>Physics salary+>29400)
John#Student(+[Mon Tue Fri] dept+>Math)
John#Top(-[1995 8 17] gender->male income->29400)
```

Generally, a psoa atom with one or more dependent descriptors and no independent descriptor is called a *perspectival atom*. If Π is the predicate of a perspectival atom (on which its descriptors are dependent), it is also called a Π*-perspectival atom* (in Section 2.3 this notion will be lifted to facts and rules). Complementarily, a psoa atom with one or more independent descriptors and no dependent descriptor is a *perspeneutral atom*. E.g., the first three atoms in (KB3) are TA-, Teacher-, and Student-perspectival; the last atom is perspeneutral.

Posing ground queries to the ground atoms of (KB3) exemplifies a *prerequisite for psoa-term unification*, which generalizes oidless-positional-term unification [14] (this prerequisite applies also to non-ground atoms in queries and KB clauses): To unify, two psoa terms must "pair up" [15] descriptors of the same dependency kind – either both independent or both dependent – after Top-dependent descriptors have been "reversed to" (cf. Footnote 17) independent descriptors. The following examples systematically vary the dependency kind for slots and tuples in the KB and the query without using any Top-dependent descriptors (queries will be indicated by a "> " prompt):

```
                                        % Slots
John#Student(... gender->male)            % Fragment of (KB2)
> John#Student(gender->male)
success
> John#Student(gender+>male)
fail

John#Student(... dept+>Math ...)          % Fragment of (KB2)
> John#Student(dept->Math)
fail
> John#Student(dept+>Math)
success
                                        % Tuples
John#Student(... -[1995 8 17] ...)        % Fragment of (KB2)
> John#Student(-[1995 8 17])
success
> John#Student(+[1995 8 17])
fail

John#Student(+[Mon Tue Fri] ...)          % Fragment of (KB2)
> John#Student(-[Mon Tue Fri])
fail
> John#Student(+[Mon Tue Fri])
success
```

Here are examples with Top-dependent descriptors in the KB, the query, or both, hence performing "KB reversion" (lifting the "reversed to" notion from descriptors to their atoms and from atoms to their KBs):

```
                                  % Slots
John#Top(gender+>male)              % (KB*)
John#Top(gender->male)              % Reversed (KB*)
> John#Student(gender->male)
success
```

```
John#Student(... gender->male)            % Fragment of (KB2)
> John#Top(gender+>male)
   John#Top(gender->male)
success

John#Top(gender+>male)                    % (KB*)
John#Top(gender->male)                    % Reversed (KB*)
> John#Top(gender+>male)
   John#Top(gender->male)
success
```

For determining the above `success` outcomes – besides the same-dependency-kind prerequisite – psoa unification, hence resolution, could be realized (e.g., by generalizing OO jDREW's POSL interpreters [15] to PSOA), complementing PSOATransRun's PSOA translators in Section 6. For example, given the KB `John#Student(+[Mon Tue Fri] ...)`, a dependency-agreeing non-ground (variable-containing) query `John#Student(+[Mon ?y ?z])` could apply unification to succeed with `?y`=`Tue` and `?z`=`Fri`. However, the prerequisite for psoa-term unification allows fast-failure decisions, e.g. as in the above `fail` outcomes. Thus, the perspectivity dimension can support both expressivity and efficiency.

The next examples demonstrate fixed- and variable-perspective querying:

```
> John#Teacher(dept+>?unit)  % Under the perspective of John as a Teacher
?unit = Physics              % his department is Physics

> John#Student(dept+>?unit)  % Under the perspective of John as a Student
?unit = Math                 % his department is Math

> John#?Persp(dept+>?unit)   % Under the perspective of John as a ...
?Persp=Teacher ?unit=Physics % ... Teacher his department is Physics
?Persp=Student ?unit=Math    % ... Student his department is Math
```

The predicate variable `?Persp` is bound non-deterministically by PSOATransRun.

2.2 Possible Dependence-to-Independence Reductions

We now discuss possible reductions that translate dependent descriptors to independent descriptors, mainly by encoding the former as the latter.

Referring to the metamodel of Appendix A, reductions of kinds of psoa atoms to other kinds have already been done before the introduction of the perspectivity dimension D_3 such as, in the descriptor-kind dimension D_2, of a tuple to slots ("positional-to-slotted", with slot names like `arg1`, ..., `argN`) and vice versa ("slotted-to-positional") of slots to a tuple [16]. Both of these should now be done in a *dependency-preserving* manner, so that an independent (resp., dependent) tuple reduces to a bag of independent (resp., dependent) slots, and a bag of independent (resp., dependent) slots reduces to an independent (resp., dependent) tuple. Other reductions are likewise possible such as, in the OID dimension D_1, of oidful to oidless atoms (moving the OID to a new, 'zeroth', argument position [15], similarly as on the runtime level by PSOATransRun's TPTP/Prolog primitives, cf. Section 6) and vice versa (PSOA's objectification, cf. [2, 3, 5]).

The current subsection augments these to considerations of reductions, in D_3, of dependent to independent descriptors, which could be complemented by reductions of independent to dependent descriptors (again, as done on the runtime level by PSOATransRun). However, of all these reductions, only (static or dynamic [5]) objectification of oidless to oidful atoms, in D_1, is required by PSOATransRun (as will be indicated), while the reductions in every dimension contribute to maximum expressivity for PSOA-centered interoperation.

For dependent descriptors that are dependent slots a simple encoding is as follows. For a pair of a predicate p and a slot name s, a new slot name s@p is introduced, where "@" is assumed to be a reserved infix character indicating that the slot name is used 'at' the predicate. For example, the perspectival atom `John#Student(dept+>Math)` would become the perspeneutral atom `John#Student(dept@Student->Math)`, while `John#Teacher(dept+>Physics)` would become `John#Teacher(dept@Teacher->Physics)`, etc. A disadvantage of this encoding is that, as one new name, s@p is indivisible, hence s1@p1 (e.g., `dept@Student`) and s1@p2 (e.g., `dept@Teacher`) appear as different as, say, s1@p1 and s2 (e.g., `income`). A further problem is lack of scalability: The combinatorics of concatenating[14] a slot name with ("@" and) predicates to form new slot names leads to multiplicative growth in the number of slot names, which creates issues for KB interchange. In particular, for real-world applications, the slot name vocabulary (e.g., a subPropertyOf taxonomy) may well become unmanageable.

Another encoding would make use of slots as (syntactically) 'higher-order' functions. For a pair of a predicate p and a slot name s, a new complex slot name s(p) is introduced, where the slot name s becomes a function taking the predicate p (hence 'higher-order') as the only argument. For example, the perspectival atom `John#Student(dept+>Math)` would become the perspeneutral atom `John#Student(dept(Student)->Math)`. This encoding would not have the vocabulary scalability problem since no new symbols are needed. A problem is the encoding-caused transition from function-free (Datalog-like) PSOA RuleML languages to function-using (Hornlog-like) ones, which are even 'higher-order'.

A third conceivable, quite different, translation, basically employing a context for each perspective, will be discussed in Section 7.

An obvious disadvantage of all these translations is the issue of unique inverse translation for reserved symbols such as "@" and for encoding constructs such as complex terms like `dept(Student)`.

For dependent descriptors that are dependent tuples, the situation is yet different. One possibility would be reducing dependent tuples to dependent slots, as indicated in Footnote 7, and then applying one of the above encodings (with their mentioned drawbacks).

Overall, since there is no uniformly 'best' translation and since dependence is the usual case for tuples, such as in relationships, and for efficiency (cf. Section 6),

[14] Since we use the abridged PSOA syntax, e.g. omitting indicators for local constants, we just need to concatenate the slot names. In the internal unabridged PSOA syntax, a slightly more involved combination of slot names would be needed.

we prefer to allow the direct modeling of dependent descriptors in the PSOA RuleML subfamily of languages (which still contains PSOA languages that do not make use of dependence but – for modeling predicate-dependent knowledge – would require some of the discussed dependence-reducing translations).

2.3 Formal Rules and Queries

Let us now proceed to rules (implications): they can use non-ground versions of all four of the psoa descriptors anywhere in their conclusion (head) and condition (body) atoms. We will focus on the unusual cases of dependent slots, `...+>...`, in `(R1)`, and independent tuples, `-[...]`, in `(R2)`.

`(KB1)`'s-`(KB3)`'s `John`-specific `TA`-perspectival fact `John#TA(workload+>high)` can be replaced by the following more versatile (conclusion-)perspectival rule over dependent slots (and built-ins), where the fact's overall color, brown, is refined with a new color, orange, for the `John`-generalizing variable `?o`:

```
Forall ?o ?ht ?hs (              % (R1)
  ?o#TA(workload+>high) :-
    And(?o#Teacher(coursehours+>?ht)
        External(pred:numeric-greater-than(?ht 10))    % ?ht > 10
        ?o#Student(coursehours+>?hs)
        External(pred:numeric-greater-than(?hs 18)))   % ?hs > 18
)
```

The rule conclusion deduces – for arbitrary OIDs `?o` that are member of `TA` – a `TA`-dependent slot `workload+>high` from a condition performing arithmetic threshold comparisons for a `Teacher`-dependent slot `coursehours+>?ht` and a `Student`-dependent slot `coursehours+>?hs`. The three `?o` occurrences refer to the same individual, but under different perspectives. The rule thus augments each condition-satisfying OID `?o` with the dependent qualitative `workload` slot.

Assuming that `(KB1)`'s-`(KB3)`'s `Teacher`/`Student` descriptors for `John` are augmented by corresponding dependent quantitative `coursehours` slots,

```
John#Teacher(... coursehours+>12 ...)
John#Student(... coursehours+>20 ...)
```

the combined changes for, e.g., `(KB2)` lead to what is called `(KB2#)` in Section 4.1, and adding the rule `(R1)` we arrive at a sample KB that is called `(KB)` in Fig. 2. The rule successfully answers the following dependent-slot ground query:

```
> John#TA(workload+>high)                       % (Q+1)
```

For this, the query is first unified with the conclusion, without need for any bindings. Then, in the condition, the first/third conjunct performs a "look-in"-retrieval [3] of the `_Teacher`/`_Student`-dependent `_coursehours` `12`/`20` slot "in" (i.e., as part of) the corresponding fact; the second/fourth conjunct "`>`"-compares the `_coursehours` filler with its threshold `10`/`18`. The (RIF-like) `External` wrapper is employed here, as usually, for built-in calls.

Similarly, the rule makes the dependent-slot non-ground query

```
> ?who#TA(workload+>?level)                     % (Q+1?)
```

succeed, with bindings ?who = John and ?level = high.[15]

A (conclusion-)perspeneutral rule mapping from a (ValidDated) independent tuple to independent slots can be used to test whether the three elements of the tuple form a valid date and putting such elements into the filler positions of appropriately named slots:

```
Forall ?o ?y ?m ?d (                         % (R2)
  ?o#Person(year->?y month->?m day->?d) :-
    And(?o#Person(-[?y ?m ?d]) ValidDate(?y ?m ?d))
)
```

The rule thus enriches an OID ?o of predicate Person that is described with such a tuple by the three slots year, month, and day.

We assume that (KB1)-(KB3) are augmented by (R2) as well as the following subpredicate fact and ValidDate-checking rule[16]:

```
Scholar##Person              % Extend TA diamond by a new subtaxonomy root
Forall ?y ?m ?d ( ValidDate(?y ?m ?d) :- And(...) )  % Ensure date triples
```

Now, rule (R2) will successfully answer the independent-slot ground query

```
> John#Person(year->1995 month->8 day->17)                    % (Q-2)
```

and succeed for the independent-slot non-ground query

```
> John#Person(year->?ye month->?mo day->?da)                  % (Q-2?)
```

with bindings ?ye = 1995, ?mo = 8, and ?da = 17.

3 Equivalence Laws for PSOA Knowledge

In this section we continue the discussion of Section 2 about knowledge representation in PSOA 1.0 by explaining laws used for its knowledge transformation, namely unscoping, describution and centralization, rescoping, as well as default expansion. The laws are formalized as meta-level equivalences ("≡") usable left (top) to right (bottom) and right to left. As equivalence laws, they define (semantics-preserving) equivalence classes of formulas, thus further preparing the model-theoretic semantics in Section 5. Some of these equivalences will also be taken up – used in one direction, for non-empty atoms – for the translation-based implementation in Section 6. In the following subsections we assume oidful atoms (oidless atoms require prior objectification), except for Section 3.3, where oidless facts are expanded into oidful rules (and vice versa).

[15] Besides the TA-dependent workload being defined here via a double threshold of Teacher- and Student-dependent coursehours, rules for Teacher- and Student-dependent workloads could also be defined, e.g.: `?o#Teacher(workload+>high) :- And(?o#Teacher(coursehours+>?ht) External(pred:numeric-greater-than(?ht 16)))`. Since John's 12 Teacher-dependent coursehours are not greater than this rule's threshold of 16, a Teacher-dependent query John#Teacher(workload+>high) would fail, unlike the TA-dependent queries.

[16] The "..." conjuncts stand for subrule queries ensuring, e.g., 28 days for February, except 29 in leap years. Finite subsets of triples from the infinite virtual date table, including ValidDate(1995 8 17), could also be materialized as facts.

3.1 From Unscoping to Describution and Centralization

In this subsection we discuss unscoping and describution as well as centralization as the inverse [3] of describution. For this, recall that independent descriptors are not sensitive to any specific (non-`Top`) predicate in whose scope they occur within an atom.

Unscoping of the independent descriptors in a perspeneutral atom with a non-`Top` predicate extracts the atom's membership, leaving behind an atom in which the non-`Top` predicate is replaced by `Top`.

Unscoping has the following general form, where Ω, Π, and Δ^{-}_{i} are meta-variables for, respectively, arbitrary OIDs, predicates, and independent ($^{-}$) descriptors ($s \geq 0$, where for $s = 0$ empty atoms ensue):

$$\Omega\texttt{\#}\Pi\texttt{(}\Delta^{-}_{1} \ldots \Delta^{-}_{s}\texttt{)}$$
$$\equiv$$
$$\texttt{And(}\Omega\texttt{\#}\Pi\ \Omega\texttt{\#Top(}\Delta^{-}_{1} \ldots \Delta^{-}_{s}\texttt{))}$$

Describution, which has also been called tupribution/slotribution and will be further characterized in the second half of this subsection, is similar to unscoping but decomposes a given zero-or-more-descriptor atom into a conjunction of single-descriptor atoms, where the given atom's OID is 'distributed' over the conjoined atoms with their single descriptors (tuples or slots).

Next, we develop examples for unscoping and describution as applied to queries, facts, and rules.

For instance, complementing the ground-query dependent-slot atom `(Q+1)` and the non-ground-query dependent-slot atom `(Q+1?)` of Section 2.3, their dual ground and non-ground independent-slot queries are (with "`+>`" reversed to "`->`"):

```
> John#TA(workload->high)                           % (Q-1)
> ?who#TA(workload->?level)                         % (Q-1?)
```

Being (predicate-)independent, the slots of these two atoms can be unscoped – from the predicate `TA` to the vacuous predicate `Top`, yielding untyped atoms – by extracting the memberships `John#TA` and `?who#TA` into separate conjuncts. By leaving behind `John#Top` and `?who#Top`, `Top` occurrences are introduced for unscoping, thus transforming the above atoms (here, the queries `(Q-1)` and `(Q-1?)`) to these equivalent conjunctions (here, conjunctive queries):

```
And(John#TA John#Top(workload->high))               % (C-1)
And(?who#TA ?who#Top(workload->?level))             % (C-1?)
```

Since `(Q-1)` and `(Q-1?)` already have single descriptors, `(C-1)` and `(C-1?)` are also their describution results.

While (`?who` =) `John` is a `TA` and as a `TA` was deduced, in Section 2.3, by the rule `(R1)` to have (`?level` =) `high workload`, generally, as a member of `Top`, which is made explicit by unscoping, `John` cannot be deduced by `(R1)` to have any description, because `(R1)`'s conclusion retains the corresponding OID variable `?o` as a member of `TA`. This difference is due to the descriptor being independent in the query (leading to `Top`) while being dependent in the rule conclusion (retaining the non-`Top` predicate), so that the prerequisite for psoa-term unification of Section 2.1 is not fulfilled. Therefore, the `(C-1)` and `(C-1?)` conjunctions (here, queries) fail.

As another example, refining the `Person` predicate of the ground independent-slot query `(Q-2)` and the non-ground independent-slot query `(Q-2?)` of Section 2.3, their `TA`-predicate versions are:

```
> John#TA(year->1995 month->8   day->17)                 % (Q-3)
> John#TA(year->?ye  month->?mo day->?da)                % (Q-3?)
```

On one hand, unscoping of the atoms of queries `(Q-3)` and `(Q-3?)` creates conjunctions where the membership `John#TA` is extracted and the atoms' predicate `TA` is evacuated to `Top`:

```
And(John#TA John#Top(year->1995 month->8   day->17))
And(John#TA John#Top(year->?ye  month->?mo day->?da))
```

On the other hand, describution (here pure slotribution rather than pure tupribution or combined tupribution+slotribution) of the same atoms creates conjunctions where the membership `John#TA` is extracted and all (here, three) slots are used for `Top`-typed single-slot atoms:

```
And(John#TA John#Top(year->1995) John#Top(month->8)   John#Top(day->17))
And(John#TA John#Top(year->?ye)  John#Top(month->?mo) John#Top(day->?da))
```

Again, `John#TA` can be shown by fact retrieval; the conclusion of the rule `(R2)` of Section 2.3 is also transformed by slotribution, so that the entire conjunctions, hence `(Q-3)` and `(Q-3?)`, can be successfully deduced with the same answers as for `(Q-2)` and `(Q-2?)`.

The meta-level equivalence for *describution* (when used left to right) and *centralization* (when used right to left) has the following general form, where Δ_i^{+} and Δ_j^{-} are names for, respectively, arbitrary dependent ($^{+}$) and independent ($^{-}$) descriptors ($r \geq 0$, $s \geq 0$):[17]

$$\Omega\texttt{\#}\Pi\texttt{(}\Delta_1^{+} \ldots \Delta_r^{+}\ \Delta_1^{-} \ldots \Delta_s^{-}\texttt{)}$$
$$\equiv$$
$$\texttt{And(}\Omega\texttt{\#}\Pi\ \ \Omega\texttt{\#}\Pi\texttt{(}\Delta_1^{+}\texttt{)} \ldots \Omega\texttt{\#}\Pi\texttt{(}\Delta_r^{+}\texttt{)}\ \Omega\texttt{\#Top(}\Delta_1^{-}\texttt{)} \ldots \Omega\texttt{\#Top(}\Delta_s^{-}\texttt{))}$$

The general case of describution will be further explained on the concrete-descriptor level in Section 6. It corresponds to Section 1's method (i), transforming a zero-or-more-descriptor atom into a conjunction of one membership

[17] Note that the "+"/"-" superscripts – like the subscripts – are part of the metavariable names. A unary prefix operator $\pm$ can be used to reverse a dependent to an independent descriptor and vice versa, keeping its content unchanged. It is defined with four equations on the concrete-descriptor level: $\pm(\texttt{+[}t_1 \ldots t_n\texttt{]}) = \texttt{-[}t_1 \ldots t_n\texttt{]}$, $\pm(\texttt{-[}t_1 \ldots t_n\texttt{]}) = \texttt{+[}t_1 \ldots t_n\texttt{]}$, $\pm$(p+>v) = p->v, $\pm$(p->v) = p+>v. For any descriptor Δ, $\pm(\pm(\Delta)) = \Delta$. The prefix $\pm$ can be applied (omitting the parentheses) on the right-hand side of a meta-level equivalence between atoms with r descriptors that are marked as dependent on `Top`, and their reversions, which are marked as independent: $\Omega\texttt{\#Top(}\Delta_1^{+} \ldots \Delta_r^{+}\ \Delta_1^{-} \ldots \Delta_s^{-}\texttt{)} \equiv \Omega\texttt{\#Top(}\pm\Delta_1^{+} \ldots \pm\Delta_r^{+}\ \Delta_1^{-} \ldots \Delta_s^{-}\texttt{)}$.

and zero or more single-descriptor atoms, where each independent descriptor's non-`Top` predicate is evacuated to `Top`, as in unscoping, while each dependent descriptor is kept within the scope of the original predicate.

Describution is applicable to each atom of a query or a KB. For example, the three facts of `(KB2)` can be transformed to this *descributed normal form* (pretty-printed so that the same kinds of descriptors are on the same line):

```
                                                         % (KB2')
. . .                                                      % Taxonomy
And(John#TA John#TA(workload+>high))                       % Data (ii')
And(
    John#Teacher
    John#Teacher(+[Wed Thu])
    John#Teacher(dept+>Physics) John#Teacher(salary+>29400)
    John#Top(income->29400)
)
And(
    John#Student
    John#Student(+[Mon Tue Fri])
    John#Top(-[1995 8 17])
    John#Student(dept+>Math)
    John#Top(gender->male)
)
```

The conjuncts can be regrouped to collect all independent descriptors into a separate conjunction (pretty-printed as above), which is also the descributed normal form of `(KB3)`:

```
                                                         % (KB3')
. . .                                                      % Taxonomy
And(John#TA John#TA(workload+>high))                       % Data (iii')
And(
    John#Teacher
    John#Teacher(+[Wed Thu])
    John#Teacher(dept+>Physics) John#Teacher(salary+>29400)
)
And(
    John#Student
    John#Student(+[Mon Tue Fri])
    John#Student(dept+>Math)
)
And(
    John#Top(-[1995 8 17])
    John#Top(gender->male) John#Top(income->29400)
)
```

This shows the logical equivalence between `(KB2)` and its perspectivity-concentrated form `(KB3)`.

The conjuncts can also be directly used in the (implicit) top-level conjunction of (the `Assert` of) a PSOA RuleML KB (cf. Section 4.1).

3.2 Rescoping as Describution and Centralization

Building on Section 3.1, we now explain rescoping for oidful atoms. *Rescoping* removes an independent descriptor of an atom that has some OID and predicate and adds this independent descriptor to an atom that has the same OID but in the non-trivial case has a different predicate. In the taxonomy, the two predicates may (a) be on the same taxonomic level – i.e., have an equal shortest distance to the root predicate `Top` – ("horizontal" rescoping), (b) be on the same taxonomic inheritance line – i.e., be on the same path to `Top` – ("vertical" rescoping), or (c) be taxonomically unrelated – i.e., neither (a) nor (b) applies – ("diagonal" rescoping). Rescoping first does unscoping (for a single-descriptor atom) or, generally, describution (for a one-or-more-descriptor atom); it then does centralization, targeting the (scope of the) other predicate.

Rescoping has the following general form ($r \geq 0$ and $r' \geq 0$ because there need not be any dependent descriptor, $s \geq 0$ because there need not be any independent descriptor in the rescoping target, and $s' \geq 1$ because there must be at least the independent descriptor $\Delta'^{-}_{i'}$ in the rescoping source):[18]

$$\begin{array}{l}
\texttt{And(}\Omega\texttt{\#}\Pi\ (\Delta^{+}_{1} \ldots \Delta^{+}_{r}\ \ \Delta^{-}_{1} \ldots \Delta^{-}_{i} \ldots \Delta^{-}_{s}\texttt{)} \\
\qquad \Omega\texttt{\#}\Pi'\texttt{(}\Delta'^{+}_{1} \ldots \Delta'^{+}_{r'}\ \Delta'^{-}_{1} \ldots \Delta'^{-}_{i'\text{-}1}\Delta'^{-}_{i'} \ldots \Delta'^{-}_{s'}\texttt{))} \\
\qquad\qquad\qquad\qquad \equiv \\
\texttt{And(}\Omega\texttt{\#}\Pi\ (\Delta^{+}_{1} \ldots \Delta^{+}_{r}\ \ \Delta^{-}_{1} \ldots \Delta^{-}_{i}\Delta'^{-}_{i'} \ldots \Delta^{-}_{s}\texttt{)} \\
\qquad \Omega\texttt{\#}\Pi'\texttt{(}\Delta'^{+}_{1} \ldots \Delta'^{+}_{r'}\ \Delta'^{-}_{1} \ldots \Delta'^{-}_{i'\text{-}1} \ldots \Delta'^{-}_{s'}\texttt{))}
\end{array}$$

For example, assuming the ground facts of the example of Fig. 1 are asserted as in Section 2.1, `(KB2)`, the below simple, "horizontal" rescoping in a conjunctive ground query uses unscoping of a `Student`-independent slot from the `Student` scope followed by centralization targeting the `Teacher` scope (intermediate derivation steps are traced using indentation):[19]

```
> And(John#Teacher() John#Student(income->29400))
   And(John#Teacher() And(John#Student() John#Top(income->29400)))
   And(John#Teacher() John#Student() John#Top(income->29400))
   And(John#Teacher() John#Top(income->29400) John#Student())
   And(And(John#Teacher() John#Top(income->29400)) John#Student()))
   And(John#Teacher(income->29400) John#Student())
success
```

Similarly, using the same `(KB2)`, the below crosswise, "horizontal" rescopings of a conjunctive ground query use 'parallel' unscopings of a `Teacher`-independent tuple from the `Teacher` scope and a `Student`-independent slot from the `Student` scope followed by two 'parallel' centralizations targeting, respectively, the `Student` and the `Teacher` scope:

---

[18] An enriched form of rescoping could be introduced, where one or more independent descriptors are moved to the rescoping target together. However, such multi-descriptor rescoping can be reduced to repeated single-descriptor rescopings.

[19] To emphasize the target of the rescoped descriptor, we retain here the empty parentheses of `John#Teacher()` instead of using the shortened `John#Teacher`.

```
> And(John#Teacher(-[1995 8 17]) John#Student(income->29400))

   And(And(John#Teacher John#Top(-[1995 8 17])) And(John#Student John#Top(income->29400)))

   And(John#Teacher John#Top(-[1995 8 17]) John#Student John#Top(income->29400))
   And(John#Teacher John#Top(income->29400) John#Student John#Top(-[1995 8 17]))

   And(And(John#Teacher John#Top(income->29400)) And(John#Student John#Top(-[1995 8 17])))

   And(John#Teacher(income->29400) John#Student(-[1995 8 17]))
success
```

Again based on (KB2)'s Teacher-independent John#Teacher descriptors and Student-independent John#Student descriptors, the below multiway, "vertical" rescopings of a conjunctive ground query containing an atom with two TA-independent John#TA descriptors use tupribution/slotribution-combining describution – where the non-Top predicate TA is evacuated to Top, as in unscoping – followed by two 'parallel' centralizations:

```
> And(John#Teacher John#TA(-[1995 8 17] income->29400) John#Student)

   And(John#Teacher And(John#TA John#Top(-[1995 8 17]) John#Top(income->29400)) John#Student)

   And(John#Teacher John#TA John#Top(-[1995 8 17]) John#Top(income->29400) John#Student)

   And(John#TA John#Teacher John#Top(income->29400) John#Student John#Top(-[1995 8 17]))

   And(John#TA And(John#Teacher John#Top(income->29400)) And(John#Student John#Top(-[1995 8 17])))

   And(John#TA John#Teacher(income->29400) John#Student(-[1995 8 17]))
success
```

Note that, although TA is a subpredicate of both Teacher and Student, the derivation does not require this taxonomic information, but instead directly uses the multi-memberships of John in the three predicates (John#TA, John#Teacher, and John#Student).

In contrast to an independent descriptor, the scope of a dependent descriptor is limited to the predicate of its enclosing atom, and no rescoping is allowed. The below query-answer pairs exemplify, also based on (KB2):

```
> And(John#Teacher(+[Mon Tue Fri]) John#Student)
fail    % Query tuple dependent on Teacher, rescoping for (KB2) impossible

> And(John#Teacher(+[Wed Thu]) John#Student)
success % Query tuple dependent on Teacher, rescoping for (KB2) unnecessary

> And(John#Teacher John#Student(dept+>Physics))
fail    % Query slot dependent on Student, rescoping for (KB2) impossible

> And(John#Teacher John#Student(dept+>Math))
success % Query slot dependent on Student, rescoping for (KB2) unnecessary
```

Note that the unscoping of a single-independent-descriptor atom is equivalent to its describution, which – when the atom with OID o is equivalently extended to a conjunction by a trivially true same-OID empty atom with predicate `Top` of the form o`#Top()` – is a special case of rescoping the descriptor to o`#Top()`. For example, revisiting Section 3.1, `(Q-1)` and `(Q-1?)` are equivalent to

```
> And(John#TA(workload->high) John#Top())                % (Q-1')
> And(?who#TA(workload->?level) ?who#Top())              % (Q-1?')
```

where the descriptors `workload->high` and `workload->?level` can be rescoped from the original atoms to the empty atoms, obtaining Section 3.1's `(C-1)` and `(C-1?)`, again usable as queries.

3.3 Default Descriptors and Their Inheritance

In AI knowledge representation, so-called "default values" (in PSOA: *default fillers*) permit *default slots* (names with fillers) to be inherited from a class to all of its instances. PSOA RuleML 1.0 allows a monotonic version of such inheritance also for *default tuples*, arriving at the generalized notion of *default descriptors*.

For realizing default-descriptor inheritance, monotonic *default rules* are used, which are rules whose conclusion derives descriptors for a universally quantified OID from a condition that proves an OID-predicate membership, where the OID represents all of the predicate's instances. This proof may directly retrieve a membership ('base case') or proceed through one or more subpredicate facts to chain to a less general predicate ('recursive case').

Following the orthogonality principle, the initial predicate is just the *seed* of the descriptors that are inherited to all of its instances – the descriptors need not be dependent but can be independent from their seed predicate. While dependent-descriptor default rules use a non-`Top` conclusion predicate, namely the same predicate as in the condition, independent-descriptor default rules use the `Top` conclusion predicate.

For example, `(KB1)-(KB3)` can be augmented by two independent-descriptor default rules as follows (cf. Int'l Standard Classification of Occupations[20]):

```
Forall ?o (
  ?o#Top(-[2 3]  % ISCO major (2: Professionals) and sub-major (3: Teaching)
         offer->service
         aptitude->explanation) :-
    ?o#Teacher
)
Forall ?o (
  ?o#Top(acquire->KSAs  % Knowledge, Skills and Abilities
         aptitude->comprehension) :-
    ?o#Student
)
```

Here are query examples inheriting default tuples and slots, where the default slot `aptitude` becomes multi-valued with non-conflicting fillers `explanation` and `comprehension` (for a conflicting example, see Section 7):

[20] `http://www.ilo.org/public/english/bureau/stat/isco`

```
> John#Teacher(-[2 3] offer->service)
success
> John#Student(acquire->KSAs aptitude->?w)
?w=explanation    % Independently from Teacher
?w=comprehension  % Independently from Student
```

These same answers are still obtained after removing the memberships `John#Teacher` and `John#Student` from `(KB1)-(KB3)`, since the remaining membership `John#TA` is reached from both rule conditions by one step of subpredicate chaining.

Moreover, if `(KB1)-(KB3)` are further augmented by the ground fact `John#TA(aptitude->illustration)`, a query like `John#TA(aptitude->?w)` exemplifies that PSOA uses – to keep the semantics simple – non-overriding, monotonic fillers, cumulatively binding `?w` to multiple values, `illustration`, `explanation`, and `comprehension`.[21]

Default rules can be abbreviated to *default facts*, a new kind of atomic formulas having the general form $\Pi\{\Delta^{+}_1 \ldots \Delta^{+}_r\ \Delta^{-}_1 \ldots \Delta^{-}_s\}$, where curley braces are used instead of parentheses. Each default fact retains an `?o`-free version of a rule's conclusion that acquires the condition predicate while omitting the condition itself. For the special case where $r{=}0$ and each Δ^{-}_j is an (independent) slotted descriptor, they correspond to "class frame formulas" of Flora-2 [17] – when used with the compiler directive `inheritance=monotonic` – of the form Π`[|`$\Delta^{-}_1 \ldots \Delta^{-}_s$`|]`, with Π acting as the class.

For our example, the following two succinct default facts are obtained:

```
Teacher{-[2 3]  % ISCO major (2: Professionals) and sub-major (3: Teaching)
        offer->service
        aptitude->explanation}

Student{acquire->KSAs  % Knowledge, Skills and Abilities
        aptitude->comprehension}
```

PSOA's *default expansion* from default facts to rules is formalized by a meta-level equivalence used left to right, where Π is an arbitrary predicate and Π' stands for Π [if $r \geq 1$, i.e. there is at least one dependent descriptor, Π must be kept for its scope] or `Top` [if $r = 0$, i.e. there are no dependent descriptors, Π can be evacuated to `Top`] ($s \geq 0$):

$$\Pi\{\Delta^{+}_1 \ldots \Delta^{+}_r\ \Delta^{-}_1 \ldots \Delta^{-}_s\}$$

$$\equiv$$

$$\texttt{Forall ?o (?o\#}\Pi'\texttt{(}\Delta^{+}_1 \ldots \Delta^{+}_r\ \Delta^{-}_1 \ldots \Delta^{-}_s\texttt{) :- ?o\#}\Pi\texttt{)}$$

[21] For cases where it is preferable to regard `illustration` as an 'exception' overriding the other two default fillers, a non-monotonic semantics – as, e.g., in Flora-2 [17] – could be orthogonally added to PSOA, both for independent and dependent descriptors, and selectively for certain slot names (e.g., `policy` but not `aptitude`), predicates, or slot-name-(dependent-on-)predicate combinations – rather than for an entire KB.

After default expansion, the inheritance querying exemplified above is realized by the PSOATransRun system as part of its normal subpredicate-to-rule transformation and rule processing.

4 Augmented PSOA RuleML Syntaxes

Extending the abridged informal syntax introduced in Section 2, the unabridged formal presentation syntax and the serialization (XML) syntax of PSOA RuleML 1.0 are dealt with in this section (assuming Section 3.3's default facts have already been expanded to rules). First, the presentation syntax is introduced in a step-wise manner, highlighting the incorporation of the dependency dimension into earlier syntaxes. Derived from this, the PSOA RuleML serialization syntax is developed, focusing on how it extends atoms of Hornlog RuleML/XML.

4.1 PSOA RuleML Presentation Syntax

We revise the syntax of [2,3,5] to indicate the dependent/independent distinction just where it is needed. This is done such that the original syntax is reused as much as possible.

In particular, for the *dependent-tuple, independent-slot special case of psoa terms*, oidless or oidful, m dependent tuples and k independent slots are permitted (m ≥ 0, k ≥ 0), with tuple i having length n_i ($1 \leq i \leq$ m, $n_i \geq 0$), where a right-slot (i.e., left-tuple) normal form is assumed:

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

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

We distinguish three subcases:

m ≥ 2 For *psoa terms with multiple dependent tuples*, "+"-prefixed square brackets are necessary (see above).

m = 1 For *psoa terms with a single dependent tuple*, "+"-prefixed square brackets can be omitted (see `1Tupled+kSlotted` and `1Tupled` below).

m = 0 For *tupleless psoa terms*, frames arise (see `kSlotted` below), which with k = 0 can additionally be specialized to *slotless psoa terms*, arriving at *empty psoa terms*, for which round parentheses can be omitted in the oidful case (see `Membership` below).

Starting with the below oidful psoa terms, color-coding shows syntactic variants for the subcases m = 1 and m = k = 0 (single-dependent-tuple brackets for $n_1 \geq 1$ are optional, as are zero-argument parentheses):

$\mathtt{1Tupled\text{+}kSlotted:}\ \mathtt{o\#f(+[t_{1,1} \ldots t_{1,n_1}]\ p_1\text{->}v_1 \ldots p_k\text{->}v_k)}$
$\mathtt{1Tupled:}\ \mathtt{o\#f(+[t_{1,1} \ldots t_{1,n_1}])}$
$\mathtt{kSlotted:}\ \mathtt{o\#f(p_1\text{->}v_1 \ldots p_k\text{->}v_k)}$
$\mathtt{Membership:}\ \mathtt{o\#f()}$

Moving on to the *dependent/independent-tuple, dependent/independent-slot general case of oidful psoa terms*, below we obtain four subsequences for the four bags of descriptor and dependency kinds (in the pretty-print arranged as four separate lines). Here, the superscripts indicate subterms that are part of dependent ($^+$) vs. independent ($^-$) descriptors. Refining earlier PSOA versions, a *right-slot, right-independent* (i.e., left-tuple, left-dependent) normal form is assumed. As suggested by the order of the italicized qualifiers, this normal form primarily distinguishes the descriptor kind and secondarily the dependency kind ($m^+ \geq 0$, $m^- \geq 0$, $k^+ \geq 0$, $k^- \geq 0$, and for $1 \leq i^{+/-} \leq m^{+/-}$, $n^+_{i^+} \geq 0$, $n^-_{i^-} \geq 0$):

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

For formulating the laws in Section 3 using the abstract-descriptor-level pattern $\Delta^+_1 \ldots \Delta^+_r \; \Delta^-_1 \ldots \Delta^-_s$, an equivalent *right-independent* (i.e., left-dependent) form was assumed for convenience, which could be refined to an equivalent *right-independent, right-slot* (i.e., left-dependent, left-tuple) form:

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

The below EBNF grammar for the presentation syntax of PSOA RuleML 1.0 uses a right-slot form (all slots are to the right of all tuples) but not any dependency form (the order between dependent vs. independent descriptors is not prescribed). It advances the grammar of the earlier PSOA RuleML [2] as follows:

- Employs the document root `RuleML`, rather than the earlier `Document`, as well as `Assert`, rather than the earlier `Group`, complementing it with `Query`, which was absent earlier.
- Refines both kinds of descriptors for the ("`DI`"-)distinction of Dependent vs. Independent tuples (`TUPLEDI`) and slots (`SLOTDI`).
- Reflects the use of (a) oidless and oidful psoa terms as `Atom`s in/as `FORMULA`s, (b) oidful `Atom`s (for unnesting, leaving behind the OID term) as `TERM`s in `Atom`s and `Expr`essions, as well as (c) oidless psoa terms as `Expr`essions.
- Revises the `CLAUSE`, `Implies`, and `HEAD` productions to make the PSOA RuleML language closed under objectification and describution.

On the top-level, the EBNF grammar is divided into two parts: Basically, while the *Rule Language* provides "wrapper" declarations around rules and the upper levels of the rules themselves, the *Condition Language* provides the formula specification for the rule conditions and for queries, and also defines psoa terms.

Rule Language:

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

Condition Language:[22]

```
FORMULA ::= 'And' '(' FORMULA* ')' |    % Main start symbol: formulas
            'Or' '(' FORMULA* ')' |
            'Exists' Var+ '(' FORMULA ')' |
            ATOMIC |
            'External' '(' Atom ')'
ATOMIC ::= Atom | Equal | Subclass
Atom ::= ATOMOIDLESS | ATOMOIDFUL  % Atoms can be oidless or oidful
ATOMOIDLESS ::= PSOAOIDLESS        % Oidless atoms are oidless psoa terms
ATOMOIDFUL ::= PSOAOIDFUL          % Oidful atoms are oidful psoa terms
Equal ::= TERM '=' TERM
Subclass ::= TERM '##' TERM  % Subclass is pars pro toto for Subpredicate
PSOA ::= PSOAOIDLESS | PSOAOIDFUL       % Extra start symbol: psoa terms
PSOAOIDLESS ::= TERM '(' (TERM* | TUPLEDI*) SLOTDI* ')'
PSOAOIDFUL ::= TERM '#' PSOAOIDLESS
TUPLEDI ::= ('+' | '-') '[' TERM* ']'
SLOTDI ::= TERM ('+>' | '->') TERM
TERM ::= Const | Var | ATOMOIDFUL | Expr | 'External' '(' Expr ')'
Expr ::= PSOAOIDLESS               % Exprs are oidless psoa terms
Const ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
Var ::= '?' PN_LOCAL?
SYMSPACE ::= ANGLEBRACKIRI | CURIE
CONSTSHORT ::= ANGLEBRACKIRI | CURIE | '"' UNICODESTRING '"'
                                        | NumericLiteral | PN_LOCAL
```

Examples for KBs according to a wrapperless Rule Language and the Condition Language were given in Sections 2 and 3.

[22] PSOA 1.0's constant and variable names use a simplified SPARQL 1.1 production [169] for `PN_LOCAL` (`https://www.w3.org/TR/sparql11-query/#sparqlGrammar`), with sources on GitHub (`https://github.com/RuleML/PSOATransRunComponents/blob/master/PSOACore/src/main/antlr3/org/ruleml/psoa/parser/PSOAPS.g`). In this ANTLR grammar, the `CONSTSHORT`-produced (rather than `Var`-produced) occurrence of `PN_LOCAL` is further differentiated – by an embedded Java action – into an "_"-prefixed or unprefixed local name (a stand-alone "_" works separately).

Fig. 2 shows a sample KB, called (KB), for the RuleML/Assert-wrapped Rule Language adding (R1) to the Condition Language clauses of the correspondingly modified (KB2), called (KB2#). Note that the RuleML wrapper contains a Prefix statement for defining, CURIE-like, pred: to access W3C RIF built-in predicates [18] from within the Assert-wrapped PSOA KB. Fig. 2 also includes the optional, hence again (blue-)colored, "_" prefix for local constants, and can be copied & pasted in any "_" form into a *.psoa KB file for PSOATransRun.

```
RuleML (
  Prefix(pred: <http://www.w3.org/2007/rif-builtin-predicate#>)

  Assert (                      % (KB)
                                   % (KB2#)
    _Teacher##_Scholar               % Taxonomy
    _Student##_Scholar
    _TA##_Teacher
    _TA##_Student
                                     % Data (ii)
    _John#_Teacher(+[_Wed _Thu]
                   _coursehours+>12 _dept+>_Physics _salary+>29400
                                                    _income->29400)
    _John#_Student(+[_Mon _Tue _Fri] -[1995 8 17]
                   _coursehours+>20 _dept+>_Math _gender->_male)

    Forall ?o ?ht ?hs (            % (R1)
      ?o#_TA(_workload+>_high) :-
        And(?o#_Teacher(_coursehours+>?ht)
            External(pred:numeric-greater-than(?ht 10))    % ?ht > 10
            ?o#_Student(_coursehours+>?hs)
            External(pred:numeric-greater-than(?hs 18)))   % ?hs > 18
    )
  )
)
```

Fig. 2. Sample (KB) of taxonomy plus data, constituting (KB2#), and rule (R1) in unabridged presentation syntax for PSOATransRun translation & execution.

4.2 PSOA RuleML Serialization Syntax

The PSOA RuleML 1.0/XML serialization syntax extends the one of Hornlog RuleML 1.02/XML. The XML serialization syntax of PSOA RuleML 1.0 can be derived from the presentation syntax. Besides obvious differences due to its use of XML markup, the serialization syntax mainly differs from the presentation syntax in being "striped", alternating between – (Java-method-style) all-lower-cased – *edges* (absent from the presentation syntax) and – (Java-class-style) first-letter-upper-cased – *Nodes* (having counterparts in the presentation syntax). While edges and Nodes are non-terminals that are 'visible' in the (parsed or generated) serialization syntax, there are also traditional – all-upper-cased – non-terminals that are 'invisible'.[23]

For the core (dependent and independent) descriptor-defining EBNF-grammar productions of the presentation syntax in Section 4.1 (reproduced – slightly modified – with a "P(resentation):" label), we give below corresponding EBNF-like productions for the serialization syntax (introduced with an "X(ML):" label).

Condition Language Descriptors (Presentation to Serialization):

```
P:  TUPLEDI ::= '+' '[' TERM* ']' | '-' '[' TERM* ']'
X:  TUPLEDI ::= tupdep | tup   % Different edges
X:  tupdep ::= Tuple           % lead into same
X:  tup ::= Tuple              % Tuple Node

P:  SLOTDI ::= TERM '+>' TERM | TERM '->' TERM
X:  SLOTDI ::= slotdep | slot  % Different edges
X:  slotdep ::= TERM TERM      % lead into same
X:  slot ::= TERM TERM         % pair of TERM Nodes
```

Entire atoms with such in/dependent-tuple, in/dependent-slot descriptors in serialization syntax can be similarly derived from the presentation syntax of Section 4.1. This can be used to parse or generate XML-serialized atoms as follows:

<Atom>
<oid><Ind>o</Ind></oid><op><Rel>f</Rel></op>
<tupdep><Tuple>$t^+_{1,1}$ ... t^+_{1,n^+_1}</Tuple></tupdep> ...
<tupdep><Tuple>$t^+_{m^+,1}$ ... $t^+_{m^+,n^+_{m^+}}$</Tuple></tupdep>
<tup><Tuple>$t^-_{1,1}$ ... t^-_{1,n^-_1}</Tuple></tup> ...
<tup><Tuple>$t^-_{m^-,1}$ ... $t^-_{m^-,n^-_{m^-}}$</Tuple></tup>
<slotdep>p^+_1 v^+_1</slotdep> ... <slotdep>$p^+_{k^+}$ $v^+_{k^+}$</slotdep>
<slot>p^-_1 v^-_1</slot> ... <slot>$p^-_{k^-}$ $v^-_{k^-}$</slot>
</Atom>

Here, the meta-variables o and f as well as the decorated meta-variables t, p, and v indicate, respectively, recursively XML-serialized OIDs and predicates as well as terms, properties, i.e. slot names, and values, i.e. slot fillers, of their presentation-syntax versions in Section 4.1.

The three psoa-atom ground facts of Section 2.1's Rich TA example (KB2) in presentation syntax result in the following serialization syntax (color-coded as in Fig. 1 and (KB2)):

```
<Atom>
  <oid><Ind>John</Ind></oid><op><Rel>TA</Rel></op>
  <slotdep><Ind>workload</Ind><Ind>high</Ind></slotdep>
</Atom>
```

[23] We employ RELAX NG as the main language to define schemas for XML, where 'visible' non-terminals correspond to element patterns and 'invisible' non-terminals correspond to *named patterns* (http://relaxng.org/compact-tutorial-20030326.html#id2814516).

```
<Atom>
  <oid><Ind>John</Ind></oid><op><Rel>Teacher</Rel></op>
  <tupdep><Tuple><Ind>Wed</Ind><Ind>Thu</Ind></Tuple></tupdep>
  <slotdep><Ind>dept</Ind><Ind>Physics</Ind></slotdep>
  <slotdep><Ind>salary</Ind><Ind>29400</Ind></slotdep>
  <slot><Ind>income</Ind><Ind>29400</Ind></slot>
</Atom>

<Atom>
  <oid><Ind>John</Ind></oid><op><Rel>Student</Rel></op>
  <tupdep>
    <Tuple><Ind>Mon</Ind><Ind>Tue</Ind><Ind>Fri</Ind></Tuple>
  </tupdep>
  <tup>
    <Tuple><Ind>1995</Ind><Ind>8</Ind><Ind>17</Ind></Tuple>
  </tup>
  <slotdep><Ind>dept</Ind><Ind>Math</Ind></slotdep>
  <slot><Ind>gender</Ind><Ind>male</Ind></slot>
</Atom>
```

The serialization for the rest of the Condition Language and for the Rule Language can be derived analogously.

A Relax NG schema for PSOA RuleML 1.0 is being developed, first in a monolithic manner[24] and then using Relax NG's modularization capability[25].

5 Model-Theoretic PSOA RuleML Semantics

In this section, key parts of the formal model-theoretic semantics definitions from [2] and [5] are revised for PSOA RuleML 1.0's object-virtualizing, in/dependent-tuple, in/dependent-slot psoa terms in right-slot, right-independent normal form (cf. Section 4.1).

Truth valuation of PSOA RuleML formulas is defined as a mapping $TVal_{\mathcal{I}}$ in two steps: 1. A mapping $\boldsymbol{I}$ generically bundles various mappings from a ***semantic structure***, $\mathcal{I}$; $\boldsymbol{I}$ maps a formula to an element of the domain $\boldsymbol{D}$. 2. A mapping $\boldsymbol{I}_{\text{truth}}$ takes such a $\boldsymbol{D}$ element to a set of ***truth values***, $\boldsymbol{TV}$. For the interpretation of individuals, $\boldsymbol{D}_{\text{ind}}$, a non-empty subset of $\boldsymbol{D}$, is used.

As a central part of $\mathcal{I}$, Definition 4, case 3, of [2] introduced the total mapping $\boldsymbol{I}_{\text{psoa}}$ without yet specifying object virtualization, of [5], nor dependency: $\boldsymbol{I}_{\text{psoa}}$ mapped $\boldsymbol{D}$ to total functions that have the general 3-ary form $\boldsymbol{D}_{\text{ind}} \times \texttt{SetOfFiniteBags}(\boldsymbol{D}^*_{\text{ind}}) \times \texttt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}) \rightarrow \boldsymbol{D}$. An argument $\texttt{d} \in \boldsymbol{D}$ of $\boldsymbol{I}_{\text{psoa}}$ uniformly represents the function or predicate symbol `f` of psoa terms `o#f(...)`. An element $\texttt{c} \in \boldsymbol{D}_{\text{ind}}$ in the first Cartesian argument of the resulting total functions represents an object as the interpretation of `o` from `o#f`, where `c` is described with two bags in the second and third Cartesian arguments (enclosed by "{. . . }", but allowing repeated elements):

[24] http://wiki.ruleml.org/index.php/PSOA_RuleML#Monolithic_Syntax

[25] http://wiki.ruleml.org/index.php/PSOA_RuleML#Modular_Syntax

- A finite bag of finite tuples $\{<\mathtt{t_{1,1}}, ..., \mathtt{t_{1,n_1}}>, \ldots, <\mathtt{t_{m,1}}, ..., \mathtt{t_{m,n_m}}>\} \in$ $\mathtt{SetOfFiniteBags}(\boldsymbol{D^*}_{\text{ind}})$, possibly empty, represents positional information. Here, $\boldsymbol{D^*}_{\text{ind}}$ is the set of all finite tuples over the domain $\boldsymbol{D}_{\text{ind}}$.
- A finite bag of attribute-value pairs $\{<\mathtt{a1,v1}>, ..., <\mathtt{ak,vk}>\} \in$ $\mathtt{SetOfFiniteBags}(\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}})$, possibly empty, for slotted information.

For PSOA RuleML 1.0, the definition of $\boldsymbol{I}_{\text{psoa}}$ is revised to map $\boldsymbol{D}$ to total functions of the general 5-ary form

$$\begin{array}{l} \quad \mathtt{SetOfPhiSingletons(}\boldsymbol{D}_{\text{ind}}\mathtt{)} \\ \times\, \mathtt{SetOfFiniteBags(}\boldsymbol{D^*}_{\text{ind}}\mathtt{)} \times \mathtt{SetOfFiniteBags(}\boldsymbol{D^*}_{\text{ind}}\mathtt{)} \\ \times\, \mathtt{SetOfFiniteBags(}\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}\mathtt{)} \times \mathtt{SetOfFiniteBags(}\boldsymbol{D}_{\text{ind}} \times \boldsymbol{D}_{\text{ind}}\mathtt{)} \\ \hfill \to \boldsymbol{D} \end{array}$$

where the argument in the first line interprets the possibly virtualized object, the two arguments of the same type in the second line interpret dependent and independent tuples, and the two arguments of the same type in the third line interpret dependent and independent slots (thus the earlier two bags are refined to four). Also, $\mathtt{SetOfPhiSingletons(}\boldsymbol{D}_{\text{ind}}\mathtt{)}$, from [5], is defined as $\{\{\}\} \cup \{\{\mathtt{c}\} \mid \mathtt{c} \in \boldsymbol{D}_{\text{ind}}\}$, whose elements are the empty set $\{\}$ and a singleton set $\{\mathtt{c}\}$ for each $\mathtt{c} \in \boldsymbol{D}_{\text{ind}}$. With this definition, the total function resulting from $\boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\mathtt{f}))$ can be appropriately applied to its arguments in Equations (1) and (2) below.

The generic recursive mapping $\boldsymbol{I}$ is defined from terms to their subterms and ultimately to $\boldsymbol{D}$. In [2], Definition 4 – before the differentiation of in/dependent descriptors – the mapping of psoa terms was as follows:

$$\begin{array}{l} \boldsymbol{I}(\mathtt{o\#f([t_{1,1}}\ ...\ \mathtt{t_{1,n_1}]\ ...\ [t_{m,1}}\ ...\ \mathtt{t_{m,n_m}]\ a_1\text{->}v_1\ ...\ a_k\text{->}v_k))} = \\ \boldsymbol{I}_{\text{psoa}}(\boldsymbol{I}(\mathtt{f}))(\boldsymbol{I}(\mathtt{o}), \\ \qquad\qquad \{<\boldsymbol{I}(\mathtt{t_{1,1}}), ..., \boldsymbol{I}(\mathtt{t_{1,n_1}})>, ..., <\boldsymbol{I}(\mathtt{t_{m,1}}), ..., \boldsymbol{I}(\mathtt{t_{m,n_m}})>\}, \\ \qquad\qquad \{<\boldsymbol{I}(\mathtt{a_1}),\boldsymbol{I}(\mathtt{v_1})>, ..., <\boldsymbol{I}(\mathtt{a_k}),\boldsymbol{I}(\mathtt{v_k})>\}) \end{array}$$

In PSOA RuleML 1.0, for **oidful** psoa terms, the definition of $\boldsymbol{I}$ becomes:

$$\boldsymbol{I}\left(\begin{array}{l} \mathtt{o\#f(+[t^+_{1,1}\ \ldots\ t^+_{1,n^+_1}]\ \ldots\ +[t^+_{m^+,1}\ \ldots\ t^+_{m^+,n^+_{m^+}}]} \\ \qquad \mathtt{-[t^-_{1,1}\ \ldots\ t^-_{1,n^-_1}]\ \ldots\ -[t^-_{m^-,1}\ \ldots\ t^-_{m^-,n^-_{m^-}}]} \\ \qquad \mathtt{p^+_1\text{+>}v^+_1\ \ldots\ p^+_{k^+}\text{+>}v^+_{k^+}} \\ \qquad \mathtt{p^-_1\text{->}v^-_1\ \ldots\ p^-_{k^-}\text{->}v^-_{k^-})} \end{array}\right) =$$

$$\begin{array}{l} \boldsymbol{I}_{\text{psoa}}\left(\boldsymbol{I}\left(\mathtt{f}\right)\right)\left(\{\boldsymbol{I}\left(\mathtt{o}\right)\},\right. \\ \qquad\qquad \{\langle \boldsymbol{I}\left(\mathtt{t^+_{1,1}}\right), \ldots, \boldsymbol{I}\left(\mathtt{t^+_{1,n^+_1}}\right)\rangle, \ldots, \langle \boldsymbol{I}\left(\mathtt{t^+_{m^+,1}}\right), \ldots, \boldsymbol{I}\left(\mathtt{t^+_{m^+,n^+_{m^+}}}\right)\rangle\}, \\ \qquad\qquad \{\langle \boldsymbol{I}\left(\mathtt{t^-_{1,1}}\right), \ldots, \boldsymbol{I}\left(\mathtt{t^-_{1,n^-_1}}\right)\rangle, \ldots, \langle \boldsymbol{I}\left(\mathtt{t^-_{m^-,1}}\right), \ldots, \boldsymbol{I}\left(\mathtt{t^-_{m^-,n^-_{m^-}}}\right)\rangle\}, \\ \qquad\qquad \{\langle \boldsymbol{I}\left(\mathtt{p^+_1}\right), \boldsymbol{I}\left(\mathtt{v^+_1}\right)\rangle, \ldots, \langle \boldsymbol{I}\left(\mathtt{p^+_{k^+}}\right), \boldsymbol{I}\left(\mathtt{v^+_{k^+}}\right)\rangle\} \\ \qquad\qquad \left.\{\langle \boldsymbol{I}\left(\mathtt{p^-_1}\right), \boldsymbol{I}\left(\mathtt{v^-_1}\right)\rangle, \ldots, \langle \boldsymbol{I}\left(\mathtt{p^-_{k^-}}\right), \boldsymbol{I}\left(\mathtt{v^-_{k^-}}\right)\rangle\}\right) \end{array} \tag{1}$$

Here, the first argument of the semantic function $\boldsymbol{I}_{\text{psoa}}\left(\boldsymbol{I}\left(\mathtt{f}\right)\right)$ is wrapped into a singleton set $\{\boldsymbol{I}\left(\mathtt{o}\right)\}$ [5], the second and third arguments are interpretations of, respectively, dependent and independent tuples, and the fourth and fifth arguments are interpretations of, respectively, dependent and independent slots. The first-argument wrapping method in Equation (1) specializes to using the empty set {} as the first argument to separately define $\boldsymbol{I}$ for **oidless** psoa terms:

$$\boldsymbol{I}\begin{pmatrix} \texttt{f(+[t}^{+}_{1,1}\ \ldots\ \texttt{t}^{+}_{1,\texttt{n}^{+}_{1}}\texttt{]}\ \ldots\ \texttt{+[t}^{+}_{\texttt{m}^{+},1}\ \ldots\ \texttt{t}^{+}_{\texttt{m}^{+},\texttt{n}^{+}_{\texttt{m}^{+}}}\texttt{]} \\ \quad \texttt{-[t}^{-}_{1,1}\ \ldots\ \texttt{t}^{-}_{1,\texttt{n}^{-}_{1}}\texttt{]}\ \ldots\ \texttt{-[t}^{-}_{\texttt{m}^{-},1}\ \ldots\ \texttt{t}^{-}_{\texttt{m}^{-},\texttt{n}^{-}_{\texttt{m}^{-}}}\texttt{]} \\ \quad \texttt{p}^{+}_{1}\texttt{+>v}^{+}_{1}\ \ldots\ \texttt{p}^{+}_{\texttt{k}^{+}}\texttt{+>v}^{+}_{\texttt{k}^{+}} \\ \quad \texttt{p}^{-}_{1}\texttt{->v}^{-}_{1}\ \ldots\ \texttt{p}^{-}_{\texttt{k}^{-}}\texttt{->v}^{-}_{\texttt{k}^{-}}\texttt{)} \end{pmatrix} =$$

$$\begin{aligned} &\boldsymbol{I}_{\text{psoa}}\left(\boldsymbol{I}\left(\texttt{f}\right)\right)(\{\}, \\ &\qquad \{\langle \boldsymbol{I}\left(\texttt{t}^{+}_{1,1}\right),\ldots,\boldsymbol{I}\left(\texttt{t}^{+}_{1,\texttt{n}^{+}_{1}}\right)\rangle,\ldots,\langle \boldsymbol{I}\left(\texttt{t}^{+}_{\texttt{m}^{+},1}\right),\ldots,\boldsymbol{I}\left(\texttt{t}^{+}_{\texttt{m}^{+},\texttt{n}^{+}_{\texttt{m}^{+}}}\right)\rangle\}, \\ &\qquad \{\langle \boldsymbol{I}\left(\texttt{t}^{-}_{1,1}\right),\ldots,\boldsymbol{I}\left(\texttt{t}^{-}_{1,\texttt{n}^{-}_{1}}\right)\rangle,\ldots,\langle \boldsymbol{I}\left(\texttt{t}^{-}_{\texttt{m}^{-},1}\right),\ldots,\boldsymbol{I}\left(\texttt{t}^{-}_{\texttt{m}^{-},\texttt{n}^{-}_{\texttt{m}^{-}}}\right)\rangle\}, \\ &\qquad \{\langle \boldsymbol{I}\left(\texttt{p}^{+}_{1}\right),\boldsymbol{I}\left(\texttt{v}^{+}_{1}\right)\rangle,\ldots,\langle \boldsymbol{I}\left(\texttt{p}^{+}_{\texttt{k}^{+}}\right),\boldsymbol{I}\left(\texttt{v}^{+}_{\texttt{k}^{+}}\right)\rangle\} \\ &\qquad \{\langle \boldsymbol{I}\left(\texttt{p}^{-}_{1}\right),\boldsymbol{I}\left(\texttt{v}^{-}_{1}\right)\rangle,\ldots,\langle \boldsymbol{I}\left(\texttt{p}^{-}_{\texttt{k}^{-}}\right),\boldsymbol{I}\left(\texttt{v}^{-}_{\texttt{k}^{-}}\right)\rangle\}) \end{aligned} \tag{2}$$

When, as in the below Definition 5, case 3, $\boldsymbol{I}$ is applied to a psoa term, its total function is obtained from $\boldsymbol{I}_{\text{psoa}}$ applied to the recursively interpreted predicate argument `f`. The application of the resulting total function to the recursively interpreted other parts of a psoa term denotes the term's interpretation in $\boldsymbol{D}$. Because PSOA RuleML's model theory has incorporated oidless psoa terms since [5], as reflected by the above Equation (2), it could not uniformly use the (interpreted) OID `o` as the $\boldsymbol{I}_{\text{psoa}}$ argument. Instead, already since [2], it has uniformly used the (interpreted) predicate `f`, which is justified by the predicate `f` always being present and user-controlled for psoa terms, with increasing precision when descending the taxonomy from the 'catch-all' total function obtained from $\boldsymbol{I}_{\text{psoa}}$ applied to the interpretation $\top$ of the root predicate `Top`. On the other hand, the OID `o` – which in RIF-BLD is used for the $\boldsymbol{I}_{\text{frame}}$ argument – need not be user-controlled in PSOA but can be system-generated via objectification, e.g. as an existential variable or a (Skolem) constant, so is not suited to obtain a meaningful total function for a psoa term. When applied to the same predicate used in different psoa terms, $\boldsymbol{I}_{\text{psoa}}$ obtains the same total function, which when itself applied to different psoa terms can return the same or different values.

PSOA RuleML 1.0 revises [2], Definition 5, case 3, by recursively defining truth valuation $TVal_{\mathcal{I}}$ for psoa formulas, based on the above-revised $\boldsymbol{I}$ and on the mapping $\boldsymbol{I}_{\text{truth}}$ from $\boldsymbol{D}$ to $\boldsymbol{TV}$ (the complementary case 8, for rule implications, is also given, unchanged):

Case 3. Psoa formulas:

$$TVal_{\mathcal{I}}\left(\texttt{f(...)}\right) = \boldsymbol{I}_{\text{truth}}\left(\boldsymbol{I}\left(\texttt{f(...)}\right)\right)$$
$$TVal_{\mathcal{I}}\left(\texttt{o\#f(...)}\right) = \boldsymbol{I}_{\text{truth}}\left(\boldsymbol{I}\left(\texttt{o\#f(...)}\right)\right)$$

For the oidful formula, consisting of an object-typing membership, two bags of tuples representing a conjunction of all the object-centered tuples, and two bags of slots representing a conjunction of all the object-centered slots, the following *describution* restriction is used, where $\texttt{m}^{+},\texttt{m}^{-},\texttt{k}^{+},\texttt{k}^{-} \geq 0$:

$$TVal_{\mathcal{I}}\left(\begin{array}{l}\texttt{o\#f(+[t}^{+}_{\texttt{1,1}}\ \ldots\ \texttt{t}^{+}_{\texttt{1,n}^{+}_{\texttt{1}}}\texttt{]}\ \ldots\ \texttt{+[t}^{+}_{\texttt{m}^{+}\texttt{,1}}\ \ldots\ \texttt{t}^{+}_{\texttt{m}^{+}\texttt{,n}^{+}_{\texttt{m}^{+}}}\texttt{]}\\ \qquad\texttt{-[t}^{-}_{\texttt{1,1}}\ \ldots\ \texttt{t}^{-}_{\texttt{1,n}^{-}_{\texttt{1}}}\texttt{]}\ \ldots\ \texttt{-[t}^{-}_{\texttt{m}^{-}\texttt{,1}}\ \ldots\ \texttt{t}^{-}_{\texttt{m}^{-}\texttt{,n}^{-}_{\texttt{m}^{-}}}\texttt{]}\\ \qquad\texttt{p}^{+}_{\texttt{1}}\texttt{+>v}^{+}_{\texttt{1}}\ \ldots\ \texttt{p}^{+}_{\texttt{k}^{+}}\texttt{+>v}^{+}_{\texttt{k}^{+}}\\ \qquad\texttt{p}^{-}_{\texttt{1}}\texttt{->v}^{-}_{\texttt{1}}\ \ldots\ \texttt{p}^{-}_{\texttt{k}^{-}}\texttt{->v}^{-}_{\texttt{k}^{-}}\texttt{)}\end{array}\right) = \mathbf{t}$$

if and only if

$$\begin{array}{ll} & TVal_{\mathcal{I}}\,(\texttt{o\#f})\\ = & TVal_{\mathcal{I}}\left(\texttt{o\#f(+[t}^{+}_{\texttt{1,1}}\ \ldots\ \texttt{t}^{+}_{\texttt{1,n}^{+}_{\texttt{1}}}\texttt{])}\right) = \ldots = TVal_{\mathcal{I}}\left(\texttt{o\#f(+[t}^{+}_{\texttt{m}^{+}\texttt{,1}}\ \ldots\ \texttt{t}^{+}_{\texttt{m}^{+}\texttt{,n}^{+}_{\texttt{m}^{+}}}\texttt{])}\right)\\ = & TVal_{\mathcal{I}}\left(\texttt{o\#Top(-[t}^{-}_{\texttt{1,1}}\ \ldots\ \texttt{t}^{-}_{\texttt{1,n}^{-}_{\texttt{1}}}\texttt{])}\right) = \ldots = TVal_{\mathcal{I}}\left(\texttt{o\#Top(-[t}^{-}_{\texttt{m}^{-}\texttt{,1}}\ \ldots\ \texttt{t}^{-}_{\texttt{m}^{-}\texttt{,n}^{-}_{\texttt{m}^{-}}}\texttt{])}\right)\\ = & TVal_{\mathcal{I}}\,(\texttt{o\#f(p}^{+}_{\texttt{1}}\texttt{+>v}^{+}_{\texttt{1}}\texttt{))} = \ldots = TVal_{\mathcal{I}}\,(\texttt{o\#f(p}^{+}_{\texttt{k}^{+}}\texttt{+>v}^{+}_{\texttt{k}^{+}}\texttt{))}\\ = & TVal_{\mathcal{I}}\,(\texttt{o\#Top(p}^{-}_{\texttt{1}}\texttt{->v}^{-}_{\texttt{1}}\texttt{))} = \ldots = TVal_{\mathcal{I}}\,(\texttt{o\#Top(p}^{-}_{\texttt{k}^{-}}\texttt{->v}^{-}_{\texttt{k}^{-}}\texttt{))}\\ = & \mathbf{t}\end{array}$$

On the right-hand side of the "if and only if" there are $1+\texttt{m}^{+}+\texttt{m}^{-}+\texttt{k}^{+}+\texttt{k}^{-}$ subformulas splitting the left-hand side into: (1) an object membership; (2) $\texttt{m}^{+}$ object-centered tupled formulas, each associating the object and the predicate with a tuple; (3) $\texttt{m}^{-}$ object-centered tupled formulas, each associating the object with a tuple using the root predicate `Top`; (4) $\texttt{k}^{+}$ object-centered slotted formulas, each associating the object and the predicate with an attribute-value pair; and (5) $\texttt{k}^{-}$ object-centered slotted formulas, each associating the object with an attribute-value pair using the root predicate `Top`.

To ensure that all members of a subpredicate are also members of its superpredicates, i.e. `o # f` and `f ## g` imply `o # g`, the following *subpredicate-membership* restriction is imposed:

– If $TVal_{\mathcal{I}}(\texttt{o \# f}) = TVal_{\mathcal{I}}(\texttt{f \#\# g}) = \mathbf{t}$ then $TVal_{\mathcal{I}}(\texttt{o \# g}) = \mathbf{t}$.

Case 8. Rule implication:

– $TVal_{\mathcal{I}}(\mathit{conclusion}$:- $\mathit{condition}) = \mathbf{t}$ if either $TVal_{\mathcal{I}}(\mathit{conclusion}) = \mathbf{t}$ or $TVal_{\mathcal{I}}(\mathit{condition}) = \mathbf{f}$.
– $TVal_{\mathcal{I}}(\mathit{conclusion}$:- $\mathit{condition}) = \mathbf{f}$ otherwise.

6 PSOA RuleML Translation by PSOATransRun

To achieve a reference implementation for deduction in PSOA RuleML, we have realized the PSOATransRun prototype as an open-source framework system, generally referred to as PSOATransRun[*translator*,*runtime*], with a pair of components 'plugged in' as parameters to create instantiations [2, 4, 19][26]. The *translator* component maps KBs and queries from PSOA RuleML to an intermediate language. The *runtime* component executes queries against a KB, both in the intermediate language, and extracts the results. Our focus is on translators, reusing the targeted runtime systems as 'black boxes'. Each translator is composed of a chain of transformers, which implement internal translation

[26] http://wiki.ruleml.org/index.php/PSOA_RuleML#Implementation

steps within PSOA RuleML, as well as a converter, which implements an external translation step to the intermediate language. For our current two instantiations, we have chosen two intermediate languages: the first-order subset, TPTP-FOF, of TPTP [20][27] and the Horn-logic subset of ISO Prolog [21]. Since these are also standard languages, their translator components in PSOATransRun serve both for PSOA RuleML implementation and interoperation [4].

The chain targeting TPTP requires four PSOA-internal translation steps – unnesting, subclass[28] rewriting, objectification, and describution – while the chain into ISO Prolog requires three subsequent translation steps – Skolemization, conjunctive-conclusion splitting, and flattening – since ISO Prolog has lower expressivity (e.g., requiring head existentials to be eliminated via Skolemization).

To incorporate the perspectival knowledge of the PSOA RuleML 1.0 language into, initially, the PSOATransRun 1.3 system, the describution transformation step is revised to replace every oidful psoa atom having the general form

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

– reflecting the describution restriction of Section 5 – with the conjunction

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

Examples of the transformation have already been given in Section 3.1.

The describution-yielded conjuncts are converted to Prolog and TPTP, which share the same syntax for atoms. This conversion uses the reserved runtime predicates `memterm`, `tupterm`, `prdtupterm`, `sloterm`, and `prdsloterm` for, respectively, membership, independent-tuple, dependent-tuple, independent-slot, and dependent-slot terms, as shown in the following table, where ρ denotes the recursive mapping from PSOA to Prolog or TPTP. The predicates `memterm`, `tupterm`, and `sloterm` have been used since our previous work [4] while `prdtupterm` and `prdsloterm` are newly introduced to translate dependent descriptors.

Psoa Atoms	Prolog and TPTP Atoms
`o#f`	`memterm(`ρ`(o),`ρ`(f))`
`o#Top(-[`$\mathtt{t_1 \ldots t_n}$`])`	`tupterm(`ρ`(o),`ρ`(`$\mathtt{t_1}$`), ... ,`ρ`(`$\mathtt{t_n}$`))`
`o#f(+[`$\mathtt{t_1 \ldots t_n}$`])`	`prdtupterm(`ρ`(o),`ρ`(f),`ρ`(`$\mathtt{t_1}$`), ... ,`ρ`(`$\mathtt{t_n}$`))`
`o#Top(p->v)`	`sloterm(`ρ`(o),` ρ`(p),` ρ`(v))`
`o#f(p+>v)`	`prdsloterm(`ρ`(o),` ρ`(f),` ρ`(p),` ρ`(v))`

[27] TPTP-FOF is also targeted by `http://wiki.ruleml.org/index.php/TPTP_RuleML`.
[28] In PSOATransRun software/papers, “subclass” is pars pro toto for “subpredicate”.

All these 'machine' predicates are oidless while taking the ρ-mapped OID `o` as their first argument. Also, `memterm`, `prdtupterm`, and `prdsloterm` take the mapped predicate `f` as the second argument. Moreover, `tupterm` and `prdtupterm` take the `n` mapped components of the tuple as the remaining arguments. Finally, `sloterm` and `prdsloterm` take the mapped slot name and the mapped slot filler as the last two arguments. The 'perspectival' `prdtupterm` and `prdsloterm` can be seen as extensions of, respectively, the 'perspeneutral' `tupterm` and `sloterm` with an extra predicate argument ρ(`f`). The extension of the three earlier runtime predicates by the two new ones has not incurred an overhead when not used and – as demonstrated below – can speed up execution when used.

In [5], we introduced static/dynamic objectification as an alternative to static objectification in [2]. The static/dynamic objectification tries to avoid generating explicit static OIDs for Prolog-like relations, instead constructing dynamic virtual OIDs at query time if and when bindings for OID variables are requested. The dynamic part of static/dynamic objectification, i.e. *dynamic objectification*, applies to atoms having a *relational predicate* in a given KB, which was defined as a predicate that has no occurrence in an oidful, multi-tuple, or slotted atom. Equivalently, a relational predicate was to occur only in oidless atoms that are empty or have one tuple. With the new perspectival/perspeneutral atoms and empty atoms now differing from atoms having one dependent empty tuple, a *relational predicate* is further restricted to a predicate with no occurrence in an oidful, empty, independent-tuple-ful, multi-tuple, or slotted atom. Equivalently, it occurs only in oidless atoms that have one dependent tuple. For an atom having an independent tuple the tuple is intended to become separated from the predicate via the atom's describution. Since dynamic objectification is designed to keep the predicate together with the tuple, it is not suitable for such an atom.

To explore performance trade-offs for differently modeled KBs, in a series of experiments we measured the runtime of tupled vs. slotted and perspectival (dependent) vs. perspeneutral (independent) variations of rule-chaining test cases, Chain, in PSOATransRun 1.3.1's Prolog instantiation, which – via the InterProlog API[29] – employs XSB Prolog as the underlying engine. The experiments were conducted with a standard XSB 3.7 installation on Ubuntu 11 running on a VirtualBox 4.3.16 virtual machine with 4GB memory over a Windows 7 host on an Intel Core i7-2670QM 2.20GHz CPU. Since PSOA RuleML's main area of differentiation is in offering novel kinds of atoms, as systematized in the metamodel of Appendix A, we focus the discussion on test-querying single (rather than conjunctions/joins of) atoms through rule chains of increasing lengths (while various other test cases – some with conjunctive queries – are provided online[30]).

We used Python-based generators to create four groups of Chain test cases,[31] each probing one of the four major kinds of atoms: dependent-tuple, independent-tuple, dependent-slot, and independent-slot. Each group has test cases distinguished by the number k of KB rules, which is a parameter of the group's

[29] `http://interprolog.com`

[30] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Test_Cases`

[31] The programs and tests are online at `http://psoa.ruleml.org/testcases/chain/`.

generator (detailed below). Each generated test case includes one KB and one query of the same dependency kind (enabling successful query answering). For each test case, the KB consists of one fact and $k \geq 0$ rules.

In the dependent-tuple group, each generated KB consists of the fact `_r0(_a1 _a2 _a3)` (an abbreviation of `_r0(+[_a1 _a2 _a3])`) and k rules of the following form ($i = 1, \ldots, k, i' = i$ - 1):

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

The dependent-tuple query of the form `_rk(?X1 ?X2 ?X3)`, posed to this k-rule KB, has one answer, `?X1=_a1,?X2=_a2,?X3=_a3`.

In the dependent-slot group, each KB consists of one fact `_r0(_p1+>_a1 _p2+>_a2 _p3+>_a3)` and k rules of the following form ($i = 1, \ldots, k, i' = i$ - 1):

```
Forall ?X1 ?X2 ?X3 (
  _ri(_p1+>?X1 _p2+>?X2 _p3+>?X3) :- _ri'(_p1+>?X1 _p2+>?X2 _p3+>?X3)
)
```

The dependent-slot query `_rk(_p1+>?X1 _p2+>?X2 _p3+>?X3)`, posed to this k-rule KB, has the same answer, `?X1=_a1,?X2=_a2,?X3=_a3`.

The dependent-slot group can be seen as a dependency-preserving, positional-to-slotted-reduced version of the dependent-tuple group (cf. Section 2.2).

The independent-tuple and independent-slot groups are constructed by reversing the two dependent groups (cf. Footnote 17).

Starting with k=0 rules and increasing in steps of 50 rules until reaching k=500 rules, we generated eleven test cases for each group and measured their query execution time. For the dependent-tuple group, we also compared the query execution time using a switch in PSOATransRun between the above-discussed static vs. static/dynamic objectification. For the other three groups, where none of the predicates can be relational, static/dynamic objectification degenerates to static objectification, hence we did not compare the two settings.

The results for the tupled groups are shown in Table 1 and Figure 3 while the results for the slotted groups are shown in Table 2 and Figure 4. In the tables, the shortcut "*query-err*" means that the query execution ran out of memory in XSB Prolog.

From the tables and figures, we can see that the slotted test cases are slower than their tupled counterparts. This is because each slotted atom has three slots while each tupled atom has one tuple: hence, after describution, each slotted atom becomes a 4-ary conjunction while each tupled atom becomes a 2-ary conjunction, leading to more branches for the slotted versions during reasoning.

Also from the tables and figures, the test cases using independent descriptors are slower than their dependent counterparts. This is because the k rules in the Chain test cases differ only in their predicates, yet for independent descriptors, describution separates the predicate from the descriptors, leaving behind `Top`-typed, single-descriptor atoms in rule conclusions and conditions that can be unified with each other, leading to a significant increase in reasoning time.

Table 1. Execution time of eleven Tupled Chain test cases.

		Dependency & Objectification Choices		
		Indep	Dep	
			Stat	Stat/Dyn
Number of Rules	0	47	51	49
	50	72	47	47
	100	161	52	46
	150	403	59	48
	200	858	71	44
	250	1636	81	44
	300	2834	82	45
	350	*query-err*	95	46
	400		106	44
	450		131	47
	500		143	44

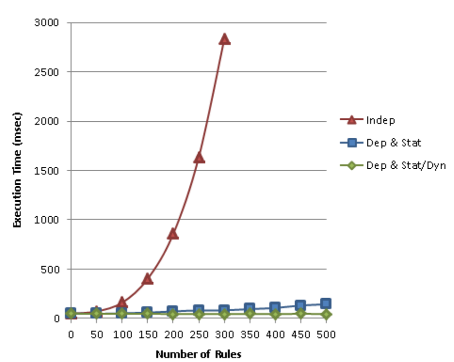

Fig. 3. Execution time of eleven Tupled Chain test cases.

Table 2. Execution time of eleven Slotted Chain test cases.

		Dependency Choices	
		Indep	Dep
Number of Rules	0	55	54
	50	106	52
	100	384	67
	150	1134	83
	200	2595	101
	250	5012	132
	300	8613	160
	350	*query-err*	202
	400		239
	450		289
	500		352

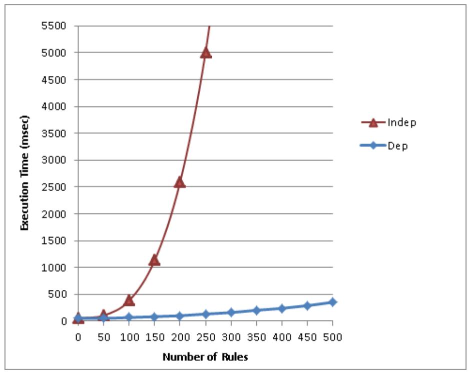

Fig. 4. Execution time of eleven Slotted Chain test cases.

For the above and similar dependent-tuple tests, static/dynamic objectification is faster than static objectification since the former keeps the PSOA relationships in Chain, converting them directly to Prolog relationships, while the latter introduces explicit (Skolem-function-nesting) OIDs for the relationships, descributes them into conjunctions, and translates them via reserved predicates.

These experiments indicate that: (1) for rules whose conclusions and conditions contain atoms with different predicates but unifiable descriptors, dependent modeling of those descriptors is more efficient than their independent modeling; (2) for argument collections that occur jointly in many atoms (e.g., arguments `?X1 ?X2 ?X3` in Chain), tupled modeling is more efficient than slotted modeling.

7 Conclusions

PSOA RuleML 1.0, featuring perspectival knowledge, constitutes a succinct yet expressive language mostly due to its orthogonal overall design according to dimensions D_1-D_3 of our metamodel for atoms in Fig. 5. Perspectivity, the novel perspeneutral/perspectival dimension D_3, is defined via independent/dependent descriptors, which can be tuples or slots. Perspectival knowledge is illustrated by the Rich TA example, visualized in Grailog in Fig. 1, describing the same OID individual differently under different perspectives using atoms having different predicates. Each descriptor of an atom can be independent or dependent from the predicate, which, respectively, allows or disallows the rescoping of the descriptor to a different predicate for a given OID. Enabling descriptor inheritance, default descriptors are realized by default rules and facts.

To incorporate perspectival knowledge, the presentation and serialization syntaxes of PSOA RuleML are appropriately augmented and the model-theoretic semantics is revised in 1.0. The novel D_3 was first realized in PSOATransRun 1.3, whose translator component revises (a) the multiple PSOA-internal translation steps, focused on the tupribution/slotribution – i.e., describution – step and (b) the conversion to Prolog or TPTP. The conversion uses new reserved runtime predicates `prdtupterm` and `prdsloterm` for dependent descriptors.

Future work on PSOA RuleML is partly driven by the structured agenda for PSOATransRun[32], with open-source development organized via GitHub.

This may involve the metamodel's dimension D_0. Its two bags of descriptors could be combined to one bag with cardinality $m + k$. Conversely, as in the semantics, D_0's two bags could be refined to four bags, also distinguishing perspeneutral vs. perspectival atoms, enabling another mapping: $D_0 \rightarrow D_3$. The current 2-bag D_0 can be reconstructed from the 4-bag D_0 by $m^+ + m^-$ and $k^+ + k^-$ (also, the 1-bag D_0 directly from the 4-bag D_0 by $m^+ + m^- + k^+ + k^-$).

The complete Grailog visualization systematics for psoa atoms of [3], introduced before the perspeneutral/perspectival distinction, can be extended to the perspectivity dimension according to Fig. 1, where an oidless atom can – instead of a rectangular OID box – use a 'box' degenerated to a "branch line" [3], e.g. as a starting point for descriptor (hyper)arcs.

The schema specification of the PSOA RuleML 1.0/XML serialization syntax in Relax NG[33] (allowing automatic translation to XSD) should be finalized. The preliminary RuleML/JSON syntax[34] could be formally defined to accommodate PSOA RuleML 1.0.

Use cases employing PSOA RuleML have been conducted, e.g. for data querying and mapping in the geospatial [22] and biomedical domains [23]. Further PSOA RuleML 1.0 KBs are being developed, e.g. with (legal/regulatory) knowledge about ships (cf. the Port Clearance use case [24]) and cars, where, e.g., amphibious vehicles are calling for perspectival knowledge. This and other use cases have been collected on the PSOA RuleML wiki page[35].

Expanding on Section 3.3, the well-known "Nixon Diamond" problem [25] can also be modeled as perspectival knowledge in a PSOA KB: (1) `Quaker{policy+>pacifist}`, i.e. under the perspective of being a `Quaker`, one's `policy` is `pacifist`. (2) `Republican{policy+>nonpacifist}`, i.e. as a `Republican`, one's `policy` is `nonpacifist`. (3) Nixon is both a `Quaker` and a `Republican`, leading to a conflict. In this modeling, querying the `policy` slot (fillers: `pacifist` or `nonpacifist`) without specifying the perspective (predicates: `Quaker` or `Republican`) would fail. Thus, the semantics is similar to the "skeptical" approach, where no conflicted conclusions can be drawn. In contrast, using perspeneutral modeling (`Quaker{policy->pacifist}` and `Republican{policy->nonpacifist}`), the `policy` slot would be predicate-independent, and the semantics similar to the "credulous" approach, where querying the `policy` slot would give conflicted (`pacifist` and `nonpacifist`) conclusions.

Moreover, the connections between contextual and perspectival knowledge can be further elaborated, including (mutual) reductions: Besides the reductions discussed in Section 2.2, perspectival knowledge could be emulated via contextual knowledge by contextualizing the clauses describing the same global (IRI) OID[36] – in a very fine-grained manner – w.r.t. their different predicates, permitting

[32] `http://wiki.ruleml.org/index.php/PSOATransRun_Development_Agenda`
[33] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Syntax`
[34] `http://wiki.ruleml.org/index.php/RuleML_in_JSON`
[35] `http://wiki.ruleml.org/index.php/PSOA_RuleML#Use_Cases`
[36] Copies of the same local ("`_`"-prefixed) OID in different (perspectival) contexts would be renamed apart on merging, which would usually be unintended.

dependent descriptors to become independent. In particular, for the OID `John` a context for each of the three predicates `Teacher`, `TA`, and `Student` could be created (an OID's multi-membership becomes a 'multi-contextship'), where, e.g., the `Teacher` context would permit independent descriptors like `dept->Physics`. Conversely, for a context-partitioned KB of oidless ground facts, OIDs could be introduced to represent context names (similar to, e.g., "named graphs" [26]), where the OID-typing predicates could provide cross-contextual perspectives.

The PSOA RuleML 1.0 reference implementation PSOATransRun 1.3.1 should be further developed as part of the PSOATransRun framework, whose current instantiations target both Prolog and TPTP. The performance of the Prolog instantiation – e.g., based on Tables 1 and 2, and on PSOA user feedback – should be further increased as part of the next release. Besides accepting the presentation syntax, it should also accept the serialization syntax, and permit translations between the two (for the serialization-to-presentation direction using the PSOA RuleML API[37]). Since, as shown in this paper, much of the expressivity of perspectival knowledge representation can already be realized on the function-free level of Datalog (rather than requiring Horn logic), a new instantiation of PSOATransRun should be done for Datalog PSOA (specializing the current Hornlog PSOA) by targeting an (object-relational) database engine, whose "views" implement rules. Conversely, PSOA's atoms could be `<repo>`-tuple-extended [3] and carried up to Naf Hornlog, FOL, and all other levels of Deliberation RuleML and, in the PSOA Prova project, to Reaction RuleML etc.

Finally, some or all of the dimensions of the PSOA metamodel could be transferred to other object-centered logics and deductive database systems.

8 Acknowledgements

The RuleML Wiki lists results by the PSOA Group. Feedback on PSOA RuleML 1.0 was obtained in the Standards Session of RuleML+RR 2017[38]. We thank Jon Hael Simon Brenas, Alexandre Riazanov, and Tara Athan for helpful comments on earlier versions of this paper. Thanks go to Sofia Almpani for a use case on medical devices, to Theodoros Mitsikas for an SWI Prolog adaptation of PSOATransRun, and to Miguel Calejo for an SWI InterProlog upgrade. NSERC is thanked for its Discovery Grants.

References

1. Gratton, M.J.: A Strongly-Local Contextual Logic. In Rossi, F., ed.: Proc. 23rd Int'l Joint Conf. on Artificial Intelligence (IJCAI 2013). (August 2013) 919–925
2. 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
3. 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)

[37] http://wiki.ruleml.org/index.php/PSOA_RuleML_API
[38] http://2017.ruleml-rr.org/schedule/

4. 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)
5. Zou, G., Boley, H.: Minimal Objectification and Maximal Unnesting in PSOA RuleML. In Alferes, J.J., Bertossi, L.E., Governatori, G., Fodor, P., Roman, D., eds.: Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings. Volume 9718 of Lecture Notes in Computer Science., Springer (2016) 130–147
6. Taivalsaari, A.: On the Notion of Inheritance. ACM Computing Surveys **28**(3) (September 1996) 438–479
7. Pernici, B.: Objects with Roles. In: Proc. ACM/IEEE Conference of Office Information Systems. (1990) 205
8. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: AL-LOG: Integrating Datalog and Description Logic. J. Intelligent Information Systems **10**(3) (1998) 227–252
9. 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
10. 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)
11. Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M., Eherer, S.: ConceptBase – A Deductive Object Base for Meta Data Management. J. Intelligent Information Systems **3**(2) (1995) 167–194
12. Boley, H.: Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. In Morgenstern, L., Stefaneas, P.S., Lévy, F., Wyner, A., Paschke, A., eds.: Proc. 7th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2013), Seattle, Washington, USA. Volume 8035 of Lecture Notes in Computer Science., Springer (July 2013) 52–67
13. Boley, H., Kifer, M.: RIF Basic Logic Dialect (Second Edition) (February 2013) W3C Recommendation, `http://www.w3.org/TR/rif-bld`.
14. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)
15. Boley, H.: Integrating Positional and Slotted Knowledge on the Semantic Web. J. Emerging Technologies in Web Intelligence **4**(2) (November 2010) 343–353
16. Boley, H., Shafiq, O., Smith, D., Osmun, T.: The Social Semantic Subweb of Virtual Patient Support Groups. In Baker, C.J.O., Chen, H., Bagheri, E., Du, W., eds.: Proceedings of the 3rd Canadian Semantic Web Symposium (CSWS2011), Vancouver, British Columbia, Canada, CEUR (August 2011) 1–18
17. Kifer, M., Yang, G., Wan, H., Zhao, C.: $\mathcal{E}$RGO^{Lite} (a.k.a. $\mathcal{F}$lora-2): User's Manual, v1.2 (2017) `http://flora.sourceforge.net/docs/floraManual.pdf`.
18. 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/`.
19. 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
20. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. J. Autom. Reasoning **43**(4) (2009) 337–362
21. Deransart, P., Ed-Dbali, A., Cervoni, L.: Prolog: The Standard – Reference Manual. Springer (1996)

22. Zou, G.: PSOA RuleML Integration of Relational and Object-Centered Geospatial Data. In Bassiliades, N., Fodor, P., Giurca, A., Gottlob, G., Kliegr, T., Nalepa, G.J., Palmirani, M., Paschke, A., Proctor, M., Roman, D., Sadri, F., Stojanovic, N., eds.: Challenge+Rule-based Recommender System+Industry Track+DC@RuleML 2015. Volume 1417 of CEUR Workshop Proceedings., CEUR-WS.org (2015)
23. Al-Manir, M.S., Riazanov, A., Boley, H., Klein, A., Baker, C.J.: Automated Generation of SADI Semantic Web Services for Clinical Intelligence. In: Proceedings of the International Workshop on Semantic Big Data. SBD '16, New York, NY, USA, ACM (2016) 6:1–6:6
24. Zou, G., Boley, H., Wood, D., Lea, K.: Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic. In: Proceedings of the RuleML+RR 2017 Challenge. Volume 1875., CEUR (July 2017)
25. Horty, J.F.: Reasons as Defaults. Oxford University Press (2012)
26. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 Concepts and Abstract Syntax (February 2014) W3C Recommendation, `https://www.w3.org/TR/rdf11-concepts/`.
27. Athan, T., Boley, H.: The MYNG 1.01 Suite for Deliberation RuleML 1.01: Taming the Language Lattice. In Patkos, T., Wyner, A., Giurca, A., eds.: Proceedings of the RuleML 2014 Challenge, at the 8th International Web Rule Symposium. Volume 1211., CEUR (August 2014)

A The Metamodel for Psoa Atoms of PSOA RuleML 1.0

This appendix introduces the metamodel for PSOA RuleML 1.0's psoa atoms. Psoa atoms can be initially characterized using a **quantitative** dimension D_0. This zeroth dimension classifies an atom via its zero-or-more ($\geq$0) descriptors partitioned into bags of m ($\geq$0) tuples and k ($\geq$0) slots, where both bags can have, e.g., zero (=0), zero-or-more ($\geq$0), single (=1), one-or-more ($\geq$1), or multiple ($\geq$2) descriptors (for details and the concrete syntax see Section 4.1). E.g., empty atoms are characterized by D_0(m=0,k=0), i.e. have neither tuples (m=0) nor slots (k=0), but like all atoms must have a predicate (which can be the *root predicate* `Top`) and may have an OID. Empty atoms constitute a category of their own since being *descriptorless* (tupleless and slotless) will make the tupled/slotted[39] and perspeneutral/perspectival distinctions non-applicable to them (not even expressible in the syntax of Section 4.1). For non-empty atoms, D_0 is used to define D_2, as shown by the $D_0 \rightarrow D_2$ mapping in Fig. 5.

There are three orthogonal **qualitative** dimensions D_1-D_3 for non-empty atoms generating eighteen subcubes. Refining the earlier six quadrants of the "psoa table" [3], the PSOA RuleML 1.0 subcubes will be labeled according to three layers of six, for non-empty atoms that are **p**erspe**n**eutral, **p**erspecti**v**al, and **p**erspeneutral+**p**erspectival, as well as suffixed with digits 1 – 6 in each layer. Besides receiving systematic labels/digits, some of the subcubes also have common names such as pn4, i.e. oidful, one-or-more-slotted, perspeneutral atoms, being referred to as *frames*; other subcubes are further specialized by D_0 for defining subsets, such as pv1, which is D_0(m=1,k=0)-specialized to oidless, single-tupled, perspectival atoms, having the common name *relationships*.[40]

[39] To emphasize the option of multiple tuples, the earlier terms "positional/slotted" have been replaced by "tupled/slotted" in most cases.

[40] The subcubes partition the set of PSOA RuleML languages into languages of, e.g., relationship and frame facts and/or rules, some of which could have anchor names, as introduced for RuleML/XML [27].

In each layer, atoms are characterized using the same two distinctions. The first dimension D_1 distinguishes atoms that are oidless-vs.-oidful predicate applications. The second dimension D_2 distinguishes atoms having as descriptors one or more tuples vs. one or more slots vs. combining one or more tuples plus one or more slots. The two main quadrants of the earlier psoa table are also accommodated by these dimensions via the above-mentioned pv1 subcube (specializing to relationships) and pn4 subcube (constituting frames). Intuitively speaking, because a tuple contains zero or more elements, a relationship affords only a single (m=1) descriptor; because a plain slot pairs a name with only a plain filler,[41] a frame affords one or more (k≥1) descriptors. Similarly, because of the only tuple's (non-association with an OID but) dependence on a predicate, relationships are perspectival; because of the one or more slots' (association with an OID but) independence from a predicate, frames are perspeneutral.

In this paper, dimensions D_0-D_2 are augmented by the dimension D_3 of atoms being *perspeneutral*, i.e. having only one or more predicate-independent descriptors, vs. *perspectival*, i.e. having only one or more predicate-dependent descriptors, vs. *perspeneutral+perspectival*, i.e. combining one or more predicate-independent plus one or more predicate-dependent descriptors.[42] In the systematics of Fig. 5, the zeroth dimension is indicated by (m,k) in/equality pairs, the first and second dimensions are constituted by the columns and rows of each layer, and the third dimension is unraveled layer-wise. As mentioned above, relationships belong to the perspectival layer, while frames belong to the perspeneutral layer. Conversely, there are also relationship-like oidless, tupled, perspeneutral atoms (pn1) and frame-like oidful, slotted, perspectival atoms (pv4). Moreover, oidful, tupled+slotted, perspeneutral atoms (pn6) D_0(m=1,k≥1)-specialize to *shelframes*; likewise, atoms in (pv5) D_0(m=1,k≥1)-specialize to *relpairships*. Analogously to the third rows for the tupled+slotted combination in D_2, the third layer is introduced for the perspeneutral+perspectival combination in D_3, accommodating atoms having at least one independent and at least one dependent descriptor.

Collections of atoms broader than one subcube (which represents a basic category) can be specified by just omitting constraints for some dimensions. For example, omitting all constraints but one, *single-tuple atoms* specify all oidless or oidful, m=1 tuple, k≥0 slot, perspeneutral or perspectival atoms. Further non-basic categories of atoms can be constructed as the union of basic or non-basic categories. In particular, *oidless, m=1 tuple, k≥0 slot, perspectival atoms* can be constructed as the union of relationship and relpairship atoms.

The dimensions of the metamodel allow the following categorization of the three color-grouped psoa atoms in Section 1, Fig. 1, all of which fixing D_1: oidful.

- The (red) `Teacher` atom fixes D_0(m=1,k=3); D_2: tupled+slotted; D_3: perspeneutral+perspectival
- The (green) `Student` atom fixes D_0(m=2,k=2); D_2: tupled+slotted; D_3: perspeneutral+perspectival
- The (brown) `TA` atom fixes D_0(m=0,k=1), a single-descriptor case, i.e. Section 1's methods (i) and (ii) coincide; D_2: slotted; D_3: perspectival

Such categorizations exemplify PSOA 1.0's novel distinction of (predicate-)independent vs. (predicate-)dependent descriptors (tuples and slots) as dimension D_3 within the larger design space generated by dimensions D_0-D_3.

[41] Here, "plain" refers to an ordinary slot with a non-tuple-valued filler; cf. Footnote 7.

[42] The perspectivity (perspeneutral/perspectival) dimension D_3 for an atom is thus based on the dependency (independent/dependent or independence/dependence) distinction for its descriptors. With the dependency superscripts ($^-/^+$) of Section 4.1: perspeneutral iff $m^- + k^- \geq 1$ and $m^+ + k^+ = 0$; perspectival iff $m^+ + k^+ \geq 1$ and $m^- + k^- = 0$; perspeneutral+perspectival iff $m^- + k^- \geq 1$ and $m^+ + k^+ \geq 1$.

Origin:

$\mathrm{D}_0(\mathrm{m}{=}0,\mathrm{k}{=}0) \quad \}$ descriptorless

Mapping:

$$\left.\begin{array}{l}\mathrm{D}_0(\mathrm{m}{\geq}1,\mathrm{k}{=}0) \longrightarrow \mathrm{D}_2\text{: tupled}\\ \mathrm{D}_0(\mathrm{m}{=}0,\mathrm{k}{\geq}1) \longrightarrow \mathrm{D}_2\text{: slotted}\\ \mathrm{D}_0(\mathrm{m}{\geq}1,\mathrm{k}{\geq}1) \longrightarrow \mathrm{D}_2\text{: tupled+slotted}\end{array}\right\}\text{ descriptorful}$$

Empty atoms are descriptorless, either oidless or (for memberships) oidful.
Non-empty atoms are constituted as descriptorful by the three layers below.

D_3: perspeneutral	D_1: oidless	D_1: oidful
D_2: tupled	pn1	pn2. $\mathrm{D}_0(\mathrm{m}{=}1,\mathrm{k}{=}0)$: shelves
D_2: slotted	pn3	pn4: ***frames***
D_2: tupled+slotted	pn5	pn6. $\mathrm{D}_0(\mathrm{m}{=}1,\mathrm{k}{\geq}1)$: shelframes

D_3: perspectival	D_1: oidless	D_1: oidful
D_2: tupled	pv1. $\mathrm{D}_0(\mathrm{m}{=}1,\mathrm{k}{=}0)$: ***relationships***	pv2
D_2: slotted	pv3: pairships	pv4
D_2: tupled+slotted	pv5. $\mathrm{D}_0(\mathrm{m}{=}1,\mathrm{k}{\geq}1)$: relpairships	pv6

D_3: perspeneutral+perspectival	D_1: oidless	D_1: oidful
D_2: tupled	pp1	pp2
D_2: slotted	pp3	pp4
D_2: tupled+slotted	pp5	pp6

Fig. 5. Basic metamodel of PSOA RuleML 1.0: Multi-dimensional psoa atoms.