

# Translators for Interoperating and Porting Object-Relational Knowledge

by

Gen Zou

Master of Computer Science and Technology, Tsinghua University,  
China, 2011

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF

**Doctor of Philosophy**

In the Graduate Academic Unit of Computer Science

Supervisor(s): Harold Boley, Ph.D, Computer Science  
Huajie Zhang, Ph.D, Computer Science  
Examining Board: Christopher J.O. Baker, Ph.D, Computer Science  
Michael W. Fleming, Ph.D, Computer Science  
Monica Wachowicz, Ph.D, Geodesy and  
Geomatics Engineering  
External Examiner: Markus Stumptner, Ph.D, School of Information Technology  
and Mathematical Sciences, University of South Australia

This dissertation is accepted by the  
Dean of Graduate Studies

**THE UNIVERSITY OF NEW BRUNSWICK**

**April, 2018**

©Gen Zou, 2018

# Abstract

With the growth of analytics for knowledge-from-data extraction, Knowledge Bases (KBs) have been increasingly represented using various, expressively overlapping, rule and ontology languages. We will focus on rule knowledge, which includes factual data as a special case. Relational rule KBs, on top of relational data, have been widely represented in languages centered on n-ary relationships. Knowledge – including rules – centered on objects and corresponding languages have been studied in Artificial Intelligence and Semantic Web as graph-based representations (frames). Combining relational and object-centered rules, object-relational rule languages have been proposed for leveraging the advantages of the two other rule paradigms: The heterogeneous approach allows both relationships and frames to be separately used while the homogeneous approach generalizes them to a uniform construct.

This dissertation advances homogeneous object-relational combinations by studying and realizing translators into and out of the Positional-Slotted Object-Applicative (PSOA) RuleML language. Two translators interoperate from PSOA to the relational languages TPTP (PSOA2TPTP) and Prolog (PSOA2Prolog); another one, from the object-centered language N3 to PSOA (N3-to-PSOA). All three translators are used for interoperating between rule

languages in different paradigms. Moreover, the PSOA2TPTP and PSOA2Prolog translators are used for porting PSOA KBs and queries.

An interoperation and portation architecture is proposed. It consists of two frameworks whose instantiations share the translator components. Each translator interoperating from PSOA realizes a composition of (1) a normalization within PSOA followed by (2) a conversion from the normalized PSOA to the target language. The normalization is itself sequentially composed of modular transformation steps within PSOA, which are shared for different targets. The transformation steps are formalized mathematically and realized as Java modules, partly generated from ANTLR grammars.

We characterize the PSOA sublanguages for which the transformation steps, the conversions to TPTP and Prolog, as well as their compositions are semantics-preserving. The proofs of semantics preservation are given.

The PSOA2TPTP and PSOA2Prolog translators are combined with reasoning engines into two instantiations of the PSOATransRun portation framework, which provide two implementations of PSOA RuleML query answering. The practicality of PSOA and PSOATransRun is supported by two use cases and evaluations of several test cases. The efficiency of relational rules is fully retained and the efficiency of certain object-centered and object-relational rules increases through our PSOA RuleML and PSOATransRun technology.

Based on the findings in translator development and PSOATransRun evaluation, the PSOA RuleML language has been revised to achieve Version 1.0.

# Acknowledgements

First and foremost, I want to express my gratitude to my main supervisor Dr. Harold Boley for his valuable guidance and the amount of energy he focused on discussing and improving this dissertation with me. I also want to thank my co-supervisor Dr. Huajie Zhang for his great assistance and support during my PhD study.

I would like to thank the members of my dissertation examining committee – Dr. Christopher J.O. Baker, Dr. Michael Fleming, Dr. Markus Stumptner, and Dr. Monica Wachowicz – for having generously given their time and expertise to improve my work.

Thanks to Dr. Alexandre Riazanov and Reuben Peter-Paul for their feedback on the work and their technical support in the development of an initial version of the TPTP instantiation of the PSOATransRun system. Thanks to Dr. Tara Athan for her support, including the design and creation of the initial structure of the psoa subdomain of the RuleML website. Thanks also to Dylan Wood and Kieran Lea as well as to Dr. Jacob Feldman for their efforts as well as hints related to the Port Clearance Rules use case.

Finally, I would like to give my special thanks to my family and friends for their support and encouragement that have facilitated the completion of the dissertation.

# Table of Contents

<b>Abstract</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iv</b>
<b>Table of Contents</b>	<b>v</b>
<b>List of Tables</b>	<b>x</b>
<b>List of Figures</b>	<b>xii</b>
<b>Abbreviations</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Research Problems . . . . .	5
1.3 Objectives of the Dissertation . . . . .	5
1.4 Dissertation Structure . . . . .	6
<b>2 Background and Related Work</b>	<b>8</b>
2.1 Relational Rule Languages . . . . .	8
2.1.1 First-Order Logic and TPTP-FOF . . . . .	11
2.1.2 Logic Programming and Pure Prolog . . . . .	14
2.2 Object-Centered Rule Languages . . . . .	16

2.2.1	N3 . . . . .	19
2.3	Object-Relational Rule Languages . . . . .	21
2.3.1	Heterogeneous Combination: F-logic and RIF . . . . .	23
2.3.2	Homogeneous Combination: PSOA RuleML . . . . .	24
2.4	Interoperation of Rules . . . . .	37
<b>3</b>	<b>Revising the PSOA RuleML Language for Version 1.0</b>	<b>42</b>
3.1	Syntax and Semantics of Embedded Psoa Terms . . . . .	42
3.2	Representing Perspectival Knowledge . . . . .	44
3.2.1	Dimensions of Psoa Atoms . . . . .	45
3.2.2	Formal Facts and Queries . . . . .	49
3.2.3	Formal Rules and Queries . . . . .	56
3.3	Revision of EBNF Grammar for Syntax . . . . .	60
3.4	Revision of Model-Theoretic Semantics . . . . .	66
<b>4</b>	<b>Interoperation and Portation Architecture</b>	<b>71</b>
4.1	PSOA-Centered Interoperation . . . . .	71
4.2	Translator-Based Portation . . . . .	77
<b>5</b>	<b>PSOA Transformation Modules and their Composition</b>	<b>84</b>
5.1	PSOA Transformations and their Semantics Preservation . . . . .	85
5.2	Unnesting . . . . .	92
5.3	Objectification . . . . .	95
5.3.1	Static Objectification Transformations . . . . .	95
5.3.2	Static/Dynamic Objectification Transformation . . . . .	99
5.4	Skolemization . . . . .	109
5.5	Subclass Transformation . . . . .	112

5.6	Description . . . . .	117
5.7	Flattening Embedded External Expressions . . . . .	119
5.8	Splitting Rules with Conjunctive Conclusions . . . . .	123
5.9	Local-Constant Renaming and KB Merging . . . . .	126
5.10	Composition of PSOA Transformations . . . . .	128
5.11	PSOA Transformations Demonstrated with Startup Example .	130
<b>6</b>	<b>Interoperation from N3 to PSOA</b>	<b>140</b>
6.1	N3Basic: An N3 Sublanguage for Interoperation . . . . .	140
6.2	Translation from N3Basic to PSOA RuleML . . . . .	142
<b>7</b>	<b>Interoperation from PSOA to TPTP (PSOA2TPTP)</b>	<b>149</b>
7.1	FOL-Targeting Normalization of the PSOA Kernel Source . .	149
7.2	Converting Normalized External-Free PSOA Kernel to TPTP	152
7.2.1	Base Terms . . . . .	152
7.2.2	Formulas . . . . .	153
7.3	Proof of Semantics Preservation . . . . .	157
<b>8</b>	<b>Interoperation from PSOA to Prolog (PSOA2Prolog)</b>	<b>161</b>
8.1	LP-Targeting Normalization of the PSOA Kernel Source . . .	161
8.2	Converting Normalized PSOA Kernel to Prolog . . . . .	164
8.2.1	Base Terms . . . . .	164
8.2.2	Formulas . . . . .	166
8.3	Proof of Semantics Preservation . . . . .	168
<b>9</b>	<b>Implementation of PSOA RuleML (PSOATransRun)</b>	<b>171</b>
9.1	ANTLR-based Translator Implementation . . . . .	171

9.2	Combining Translators and Reasoning Engines . . . . .	178
9.2.1	Combining PSOA2TPTP and a TPTP Engine . . . . .	178
9.2.2	Combining PSOA2Prolog and a Prolog Engine . . . . .	180
9.3	Overall Program Structure . . . . .	181
<b>10</b>	<b>Use Cases</b>	<b>184</b>
10.1	Port Clearance Rules . . . . .	185
10.1.1	Background . . . . .	185
10.1.2	Formalizing the Port Clearance Rules in PSOA . . . . .	186
10.1.3	Enrichment by Port Clearance Facts and Queries . . . . .	194
10.2	OfficeProspector . . . . .	199
10.2.1	Knowledge Modeling . . . . .	199
10.2.2	Data Facts . . . . .	202
10.2.3	Rules . . . . .	207
10.2.4	Use Scenarios . . . . .	216
<b>11</b>	<b>Evaluation</b>	<b>221</b>
11.1	Evaluation of PSOA RuleML 1.0 . . . . .	221
11.2	Evaluation of PSOATransRun Instantiations . . . . .	222
<b>12</b>	<b>Conclusions and Future Work</b>	<b>235</b>
12.1	Summary . . . . .	235
12.2	Dissertation Contributions . . . . .	237
12.3	Future Work . . . . .	241
	<b>Bibliography</b>	<b>254</b>
<b>A</b>	<b>Port Clearance KB in PSOA RuleML</b>	<b>255</b>

Glossary

261

Vita

# List of Tables

2.1	TPTP-FOF constructs. . . . .	13
5.1	Constraints on applying transformation steps. . . . .	130
7.1	Conversion from PSOA/PS formulas to TPTP formulas. . . . .	156
8.1	Conversion of PSOA (arithmetic) built-ins to Prolog. . . . .	165
8.2	Conversion from PSOA/PS formulas to Prolog formulas. . . . .	167
9.1	Major classes in each component of PSOATransRun. . . . .	182
10.1	Slots of building objects. . . . .	204
11.1	Execution time of eleven Tupled Chain test cases for Prolog instantiation. . . . .	226
11.2	Execution time of eleven Slotted Chain test cases for Prolog instantiation. . . . .	227
11.3	Execution time of eleven Tupled Chain test cases for TPTP instantiation. . . . .	228
11.4	Execution time of eleven Slotted Chain test cases for TPTP instantiation. . . . .	228
11.5	Execution time of Tupled NDChain test cases for Prolog instantiation. . . . .	232

11.6 Execution time of Tupled NDChain test cases for TPTP instantiation. . . . .	233
--	-----

# List of Figures

3.1	Basic metamodel of PSOA RuleML 1.0: Multi-dimensional psoa atoms. . . . .	48
4.1	Interoperation metaframework. . . . .	73
4.2	PSOA-centered interoperation framework. . . . .	73
4.3	An instantiation of PSOA-centered interoperation framework. . . . .	74
4.4	Expanded view of translation network. . . . .	77
4.5	Translator-based metaframework for portation. . . . .	77
4.6	Semantics-preserving translation from $L_s$ to $L_t$ . . . . .	80
9.1	Schematic ANTLR-based translator implementation. . . . .	172
10.1	Visualization of PSOA RuleML decision model for Port Clearance Rules. . . . .	187
10.2	Partonomy-taxonomy for office suites. . . . .	200
10.3	Example floor plan of Office Suite 101. . . . .	202
10.4	Example of facts describing interior structures of office suites. . . . .	203
10.5	A building fact. . . . .	204
10.6	Integration rule for buildings. . . . .	208
10.7	Integration rule for buildings. . . . .	209

11.1 Execution time of eleven Tupled Chain test cases for Prolog instantiation. . . . .	226
11.2 Execution time of eleven Slotted Chain test cases for Prolog instantiation. . . . .	227

# List of Symbols, Nomenclature or Abbreviations

Symbols:

$\Phi$	A set of knowledge bases
$\phi$	A knowledge base
$BQ$	A set of boolean queries, see Definition 4.1
$Q$	A set of queries
$q$	A query
$\circ$	A composition of translations, see Definition 4.3
$L' \preceq L$	$L'$ is a sublanguage of $L$ , see Definition 4.1
$\text{tr}_{L_s, L_t}$	A translation from language $L_s$ to $L_t$ , see Definition 4.2
$\text{tr}_L$	A translation within language $L$ , also called a transformation
$\dashv\!\!\!\dashv_L$	An entailment in language $L$ defined under either model-theoretic or proof-theoretic semantics
$\vdash_L$	An entailment in language $L$ defined under proof-theoretic semantics
$\models_L$	An entailment in language $L$ defined under model-theoretic semantics
$\models^\bullet, \models^{\bullet\bullet}$	A model-theoretic entailment chosen from a set of entailments
$\models_{-r_1 r_2 \dots r_m}^\bullet$	A model-theoretic entailment under a relaxed PSOA semantics in which the semantic restrictions $r_1, \dots, r_m$ are excluded (notated by a preceding '-' sign which applies to each $r_i$ ) from $\models^\bullet$
$\mathcal{P}_K$	PSOA Kernel, which is the PSOA sublanguage focused in the dissertation, see Definition 5.3
$\mathcal{I}$	A semantic structure
$TVal_{\mathcal{I}}(\tau)$	Truth evaluation of a formula $\tau$ under the semantic structure $\mathcal{I}$
<b>t</b>	True value
<b>f</b>	False value

Acronyms:

AI	Artificial Intelligence
ANTLR	ANother Tool for Language Recognition
AST	Abstract Syntax Tree
DL	Description Logic
EBNF	Extended Backus-Naur Form
F-logic	Frame logic
FOL	First-Order Logic
IRI	Internationalized Resource Identifier
LP	Logic Programming
KB	Knowledge Base
PSOA (RuleML)	Positional-Slotted Object-Applicative RuleML
OID	Object IDentifier
RDF	Resource Description Framework
RIF	Rule Interchange Format
RIF-BLD	RIF Basic Logic Dialect

# Chapter 1

## Introduction

Artificial Intelligence (AI) is a main research area in computer science that aims at designing “intelligent agents” that can mimic some human cognitive functions such as learning, reasoning, and problem solving. Knowledge representation is a subarea of artificial intelligence that studies the representation of human knowledge in machine-readable formats for various reasoning tasks.

In this chapter, we explain the motivation, research problems, research objective, and the structure of the dissertation.

### 1.1 Motivation

Data has been playing an increasingly important role for analytic technology supporting the decision making in the daily business of many companies. Based on such analytics, data can be processed (e.g., by inductive learning) to synthesize knowledge, which itself can then – as explored in this dissertation – be used (e.g., by deductive reasoning) to add more data. In the rest of the dissertation, expanding this notion of “knowledge (in the narrow sense)”, we

will employ “knowledge” to stand for “knowledge (in the broad sense) incorporating data as a special case”. Knowledge Bases (KBs) and knowledge-rich applications have long constituted a major research area in AI. The demand for KBs supporting such systems has been growing along with the rapid increase of the incorporated data sources, often mapped from external (e.g. Web) databases and document archives. However, much of the current Web content is still encoded on the syntactic level using formats such as HTML and its variants, which focus mainly on the ‘shallow’ presentation of text and multimedia to humans rather than the processing of knowledge by machines. Enabling computers to increasingly capture the meaning of text and multimedia through ‘deep’ representations amenable to semantic processing technologies, while difficult, will lead to ever more Web KBs. The Semantic Web is the effort of organizing Web information in logic-based languages with formalized meanings, so that it can be better employed by knowledge-based applications. For example, state-of-the-art search engines not only index Web pages, but also extract entities and their connections from those pages and store them in graph databases,<sup>1</sup> which model information in an object-centered manner, e.g. using W3C’s RDF language or corresponding proprietary formats. This information is then employed to enrich document-retrieving answers to queries with direct answers about classes and properties of the main entities referred to in the queries.

An appropriate representation<sup>2</sup> of knowledge is the key to successful KBs and knowledge-rich applications, and many languages have been developed

---

<sup>1</sup><http://arstechnica.com/information-technology/2012/06/inside-the-architecture-of-googles-knowledge-graph-and-microsofts-satori/>

<sup>2</sup>‘Knowledge representation’ and ‘modeling’ will be used interchangeably in the rest of the dissertation.

for modeling knowledge in various ways. The major (overlapping) categories of knowledge representation languages are ontology and rule languages, both widely used in many systems and applications. Ontology languages are mostly based on description logic [53], where KBs are composed of statements called axioms, which are built on the basis of individuals, concepts, and roles. A concept denotes a set of individuals, which can be represented as a unary relation or as a class in rule languages. A role denotes a binary relation between the individuals, which can be represented as a binary relation or a slot in rule languages. In ontologies, axioms are usually separated into assertional (ABox) and terminological (TBox) axioms as well as, sometimes, into relational (RBox) axioms. ABox axioms describe *instance-level knowledge*, mainly expressing that an individual belongs to a concept or a pair of individuals belongs to a role. TBox and RBox axioms describe *schema-level knowledge*, e.g. expressing a subsumption between, respectively, two concepts and two roles. In contrast, KBs modeled using rule languages are composed of statements called *clauses*, which are normally separated into *facts* and *rules*. Fact clauses generalize ABox axioms, which represent only unary and binary relationships, to *n*-ary relationships, which can additionally have nested function applications and may contain variables. The special case of variable-free facts is used as formal representations of data as well as simple schema information such as subsumptions. Rule clauses are flexibly usable in two directions: A rule can answer a query by reduction to subqueries until reaching facts (top-down) or derive new facts from given facts (bottom-up). Rules can also be used to express schema-level knowledge in an alternative form. In the literature, rule languages have been used to express knowledge-based data access [32, 40],

data associations/dependencies [70], privacy/security/trust policies [28], business logics [39], legal norms [2, 7], and biomedical concept definitions [30, 64].

Traditionally, the relational and object-centered modeling paradigms have been widely used for representing knowledge in the Semantic Web and AI. The relational paradigm (e.g., first-order logic and logic programming) models entity connections using predicates applied to positional arguments, while the object-centered paradigm (e.g., RDF and N3) uses *frames* to model each entity using a globally unique Object Identifier (OID) typed by a class and described by an unordered collection of slotted (attribute-value) arguments. In order to facilitate interoperability between the two paradigms and provide more modeling options, combined object-relational paradigms have been studied. F-logic [49] and RIF-BLD [22] employ a heterogeneous approach which allows the mixed use of both relations and frames. In contrast, the Web rule language PSOA RuleML [16] employs a homogeneous approach by generalizing relations and frames into **positional-slotted object-applicative** terms, which permit a predicate application to have an OID – typed by the predicate – and, orthogonally, to have positional or slotted arguments.

Since different systems have been developed on top of different knowledge representation languages, it is often necessary to translate,<sup>3</sup> integrate, and reuse KBs expressed in different languages that are in the same or different paradigms. To address this need, several rule standards have been proposed, e.g. by the AI (e.g., KIF [37] and Common Logic [69]), Semantic Web (RIF [21]), and Business Rules (e.g., SBVR [1]) communities, as well as

---

<sup>3</sup>In this dissertation, we focus on ‘high-level’ translations, which translate KBs/queries in a source knowledge representation language to KBs/queries in a target language (cf. Section 4.1), rather than to an implementation-level language, e.g. an assembly language.

in a cross-community setting (RuleML [26]<sup>4</sup>).

## 1.2 Research Problems

Although there has been considerable amount of work on knowledge representation and interoperation between rule languages as well as different logic formalisms, there are still several open problems that remain unexplored, or not completely investigated.

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

## 1.3 Objectives of the Dissertation

The overarching objective of this dissertation is to study and realize translations between the object-relational PSOA RuleML and rule languages in other

---

<sup>4</sup><http://ruleml.org>

paradigms for the interoperation and portation of object-relational knowledge. Specifically, the work aims to accomplish the following sub-objectives.

1. Study the interoperation from PSOA RuleML to the purely relational languages TPTP and Prolog as well as from the purely object-centered N3 to PSOA RuleML.
2. Design an architecture for interoperating and porting object-relational knowledge.
3. Characterize sublanguages of PSOA RuleML for which the translations are semantics-preserving, i.e. sound and complete.
4. Focusing on semantically compatible sublanguages, realize translators based on the proposed translations for interoperation.
5. Using the translators, realize prototype reasoning systems for PSOA RuleML query answering.
6. Apply the PSOA RuleML language and its translator-based reasoning systems to use cases.
7. Develop test cases and evaluate the realized reasoning systems for PSOA.
8. Revise the syntax and semantics of PSOA RuleML based on findings in the development and the evaluation. Update the translators accordingly.

## 1.4 Dissertation Structure

The rest of the dissertation is organized as follows. Chapter 2 presents necessary preliminaries of relational, object-centered, and object-relational rule

languages as well as related work of rule interoperation. Chapter 3 revises the syntax and semantics of PSOA RuleML to achieve Version 1.0 based on findings in translator development and the evaluation of translator-based reasoning systems. Chapter 4 gives an architecture for rule interoperation and portation. Chapter 5 explains PSOA transformation modules that can be reused across different translators as well as their semantics preservation. Chapter 6 explores the interoperation from the purely object-centered N3 language to PSOA RuleML. Built on top of the architecture in Chapter 4, Chapters 7 and 8 explore the interoperation from PSOA RuleML to the purely relational TPTP and Prolog languages, respectively. The translations for interoperating to both targets are compositions of a normalization within PSOA RuleML (Sections 7.1 and 8.1) and a conversion from PSOA to the target language (Sections 7.2 and 8.2). The semantics preservation of the translations is also studied. Chapter 9 explains the prototype implementation of PSOA RuleML based on the PSOATranRun framework in Section 4.2. Chapter 10 studies the PortClearanceRules and OfficeProspector use cases for PSOA RuleML and PSOATransRun. Chapter 11 evaluates the PSOA RuleML 1.0 language as well as the two implemented PSOATransRun instantiations. Finally, Chapter 12 concludes the dissertation and envisions future work.

# Chapter 2

## Background and Related Work

Modeling entities and their connections is the main characteristic of knowledge representation, e.g. with rule languages. In this dissertation, we will focus on declarative rule languages, which model entities and connections, where rules are used to deduce information that is implicit in a KB. Despite different syntaxes, most languages share similar paradigms for modeling entities and connections: relational (table-like), object-centered (graph-like), or object-relational (combined). To demonstrate the three modeling paradigms, we will use a running example that represents and deduces geometric connections among geographical units. This chapter introduces the declarative cores of rule languages as representatives for each of these paradigms in Sections 2.1, 2.2, and, 2.3, as well as interoperation of rules in Section 2.4.

### 2.1 Relational Rule Languages

In relational rule languages such as Prolog [35, 44], a relationship among  $n$  entities is modeled as an  $n$ -ary predicate applied to an ordered sequence of  $n$

argument entities ( $n \geq 0$ ).

For example, a ternary *betweenRel* predicate can be applied to  $n = 3$  arguments in an outer1-inner-outer2 order, as in the relational fact *betweenRel(canada, usa, mexico)*. A similar **between** predicate is adopted in the betweenness ontologies in the COLORE repository<sup>5</sup>. The sequence of arguments is called a *tuple*. Each entity in a tuple is a *positional argument* whose meaning is relative to its position in the sequence employed by the definition of the predicate. For example, the *betweenRel* predicate in many natural languages corresponds to a *NLbetweenRel* predicate using entities in an inner-outer1-outer2 order, as in the fact *NLbetweenRel(usa, canada, mexico)*, for the direct representation of sentences like “The USA is between Canada and Mexico.” Such facts can be augmented by rules that allow (top-down) inferential querying and (bottom-up) derivation of new facts. For example, the symmetry of *betweenRel* in the outer1/outer2 arguments can be formalized by the following rule:

$\forall Outer1, Inner, Outer2 :$

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

Here we use predicate-logic syntax, where the head and the tail of ‘ $\Leftarrow$ ’ correspond to the conclusion and the condition of the rule, respectively. Constants and predicates start with lower-case letters, while variables start with upper-case letters.

With this rule, one can successfully query *betweenRel(mexico, usa, canada)* or derive this relationship as a new fact based on the given fact *betweenRel(canada, usa, mexico)*. Hence, the deducible (queryable or deriv-

---

<sup>5</sup><http://st1.mie.utoronto.ca/colore/betweenness/betweenness.xml>

able) relationship does not need to be explicitly stored in the KB. For each separate given fact, e.g.  $betweenRel(norway, sweden, finland)$ , this same rule deduces a symmetric counterpart, e.g.  $betweenRel(finland, sweden, norway)$ . Another rule can be used to deduce binary adjacency relationships from ternary betweenness relationships:

$\forall Outer1, Inner, Outer2 :$

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

This neighbor rule can, e.g., deduce  $neighborRel(mexico, usa)$  from  $betweenRel(mexico, usa, canada)$ , which was itself deduced by the earlier symmetry rule. In general, because of the symmetry rule, no second adjacency rule with the conclusion  $neighborRel(Outer2, Inner)$  is required.

Such rules are analogous to views in relational databases: With only simple terms (constants and variables) as arguments of predicates, they lead to the (declarative) modeling core of Datalog [40]. Datalog can be extended to Horn logic (Hornlog) and first-order logic. Hornlog extends Datalog by allowing compound terms which apply (constructor) functions to terms. For example, the following fact and rule define the infinite natural number sequence  $0, s(0), s(s(0)), \dots$  in successor arithmetic using the unary predicate  $nat$  and the unary function  $s$  representing the successor of a number.

$$nat(0)$$

$$\forall X : nat(s(X)) \Leftarrow nat(X)$$

Starting from the fact  $nat(0)$  and repeatedly applying the rule,

$nat(s(0)), nat(s(s(0))), \dots$  can be derived. For such a Hornlog KB, a query like  $nat(X)$ , which asks for any natural number, leads to a (potentially) infinite enumeration of answers.

### 2.1.1 First-Order Logic and TPTP-FOF

Many relational languages use First-Order Logic (FOL) or its variants (e.g., FOL with equality) or subsets as their logical foundation. The syntax of FOL uses four types of symbols: constants, variables, functions, and predicates. Constant symbols represent entities in the domain of interest (e.g., the above *canada*, *usa*, and *mexico*). Variable symbols range over entities in the domain (e.g., the above *Outer1*, *Inner* and *Outer2*). Function symbols represent mappings from tuples of entities to entities. Predicate symbols denote mappings from tuples of entities to truth values, representing relations among entities in the domain (e.g., the above *betweenRel*).<sup>6</sup> In the rest of this section, we use  $a, b, c$  for constants,  $X, Y, Z$  for variables,  $f, g, h, \dots$  for functions, and  $p, q, r, \dots$  for predicates. A *term* in FOL represents an entity in the domain. It can be simple, i.e. a constant or a variable, or compound, i.e. a function applied to a tuple of (recursively) terms.<sup>7</sup> Compound terms are also called *expressions*. An *atomic formula* or *atom*<sup>8</sup> in FOL is a predicate symbol applied to a tuple of terms,  $p(t_1, \dots, t_n)$ , or an equality between two terms,  $t_1 = t_2$ . Formulas are recursively constructed from atomic formulas using the well-known logical connectives and quantifiers:  $\neg F_1, F_1 \wedge F_2, F_1 \vee F_2, F_1 \Rightarrow F_2, F_1 \Leftrightarrow F_2, \forall x : F_1$

---

<sup>6</sup>For other languages in the dissertation, the notion of constants is generalized to include also predicates and functions.

<sup>7</sup>This sense of ‘term’ is restricted to FOL terms. By default, we use the higher-order sense of ‘term’ in the dissertation, which can also be an atom.

<sup>8</sup>In RIF and PSOA RuleML the notion of “atomic formulas” are different from “atoms”.

and  $\exists x : F_1$ . Each connective determines how the truth value is obtained from the truth values of its subformulas. A first-order theory or KB consists of a set of *clauses* which are first-order formulas without *free variables*, i.e. variables that are not quantified by the existential quantifier  $\exists$  or the universal quantifier  $\forall$ . A *ground atom* is a variable-free atom while a *non-ground atom* is an atom with a variable. The Horn subset of FOL, for which efficient reasoning algorithms exist, consists of formulas that are free of negations, disjunctions, and existentially quantified variables.

The semantics of FOL can be defined using semantic structures [36]. A semantic structure  $\mathcal{I}$  defines a nonempty set  $\mathbf{D}$  called the *domain*. The structure also interprets constants as entities in  $\mathbf{D}$ ,  $n$ -ary functions as mappings  $\mathbf{D}^n \rightarrow \mathbf{D}$ , and  $n$ -ary predicates as mappings from  $\mathbf{D}^n$  to truth values. Given a semantic structure  $\mathcal{I}$ , truth values of predicate applications can be directly obtained. An equality formula  $t_1 = t_2$  is true if  $t_1$  and  $t_2$  are interpreted as the same entity in  $\mathbf{D}$ . Truth values of compound formulas are defined recursively according to the meanings of the various constructs. A structure  $\mathcal{I}$  satisfies a set of formulas  $\phi$  if each formula in  $\phi$  is evaluated to be true in  $\mathcal{I}$ . In such case we call  $\mathcal{I}$  a model of  $\phi$ , written as  $I \models \phi$ . A formula  $\psi$  is entailed by  $\phi$ , written as  $\phi \models \psi$  (overloading the “ $\models$ ” operator), if each model of  $\phi$  is also a model of  $\psi$ .

First-order formulas can be expressed in a machine-readable way using TPTP (Thousands of Problems for Theorem Provers) [67], a widely used language in automated theorem proving systems.<sup>9</sup> A TPTP problem is a list of

---

<sup>9</sup>As its full name suggests, “TPTP” is also used to refer to the library of test problems written in the TPTP language

annotated formulas of the form:

*language(name, role, formula, source, useful info).*

*language* specifies the TPTP dialect, i.e. a sublanguage for a logic, used to write the formula. In this dissertation, we focus on the widely used FOF dialect which is able to express arbitrary first-order formulas. *name* is a name given to the formula. *role* specifies the intended use of the formula. The most important roles are `axiom`, `hypothesis`, and `conjecture`. *formula* is the formula body. *source* and *useful info* are optional and irrelevant for us.

TPTP uses strings starting with an upper-case letter or lower-case letter for variables or constants, respectively. Table 2.1 shows the TPTP-FOF syntax for common FOL symbols.

Table 2.1: TPTP-FOF constructs.

TPTP-FOF	FOL	TPTP-FOF	FOL
~	¬	=>	⇒
&	∧	<=	⇐
	∨	?[X1, X2, ...]:	∃X1, X2, ...:
=	=	![X1, X2, ...]:	∀X1, X2, ...:

The TPTP-FOF forms of the fact and two rules in the preamble of Section 2.1 are:

```
fof(ax1,axiom,betweenRel(canada,usa,mexico)).
fof(ax2,axiom,! [Outer1,Inner,Outer2]:
    betweenRel(Outer2,Inner,Outer1) <= betweenRel(Outer1,Inner,Outer2)).
fof(ax3,axiom,! [Outer1,Inner,Outer2]:
    neighborRel(Outer1,Inner) <= betweenRel(Outer1,Inner,Outer2)).
```

### 2.1.2 Logic Programming and Pure Prolog

Logic Programming (LP) can use logic as a procedural language. A full logic program uses procedural features like mode-restricted (e.g., ground-call-only) built-ins and negation-as-failure, which cannot be directly expressed in FOL. Prolog is the most commonly used language for logic programming. ISO Prolog [44] is the international standard of Prolog. For the relational parts of this dissertation we focus on the Horn subset of Prolog, also called *Pure Prolog*, which excludes procedural features like negation-as-failure and is a subset of FOL.

Constants in Prolog include *atosyms* (atomic symbols)<sup>10</sup> and *numbers*. An atosym denotes a predicate or a function (including a nullary function, i.e. an individual). It starts with a lower-case letter, followed by other characters, e.g. `a1`, or is a single-quoted sequence of arbitrary characters,<sup>11</sup> e.g. `'http://abc'`. Variables in Prolog are basically sequences of letters or digits starting with an upper-case letter. A Prolog term is a constant, a variable, or a *compound term* of the form  $p(t_1, \dots, t_n)$ , where  $p$  is an atosym and  $t_1, \dots, t_n$  are terms.

An ISO Prolog *predication* is an atosym (for a nullary predicate) or a compound term. It corresponds to a logical atom in other languages. A *clause* is either a *fact* in the form of a predication or a *rule* of the form `Head :- Body`, where for Pure Prolog the symbol `:-` has the same meaning as `⇐` in FOL. The `Head` is a predication, while the `Body` can be a predication, a conjunction of bodies (`Body, . . . , Body`), or a disjunction of bodies (`Body; . . . ; Body`). All variables in a clause are considered to be in the scope of **universal** quan-

---

<sup>10</sup>To avoid confusion with logical atoms, we use “atosym” to refer to symbols that ISO Prolog calls “atoms”.

<sup>11</sup>Special characters, such as the single quote itself, need to be escaped.

tifiers preceding the entire clause. A *logic program* consists of a set of clauses.

Similar to an FOL atom, a Prolog fact with no variables is a *ground fact*, while a fact with variables is a *non-ground fact*. A *query* in Prolog can be an atom or a conjunction of atoms. Similar to a fact, a query can be either ground or non-ground depending on whether it has variables. However, variables in a query are treated to be in the scope of **existential** quantifiers preceding the entire query. Entailment between a Prolog KB and query is defined via a proof procedure and was shown in [57] to be a special case of entailment between an FOL theory and a formula. In Prolog systems, an answer to a non-ground query is not only a *yes/no* (i.e., true/false) answer, but also the bindings of the (existential) variables, i.e. the substitution or mapping from variables to terms, that makes the query true.

As an example of a Prolog KB, a logic program consisting of the fact and two rules from the example in the preamble of Section 2.1 can be written as:

```
betweenRel(canada,usa,mexico).  
betweenRel(Outer2,Inner,Outer1) :- betweenRel(Outer1,Inner,Outer2).  
neighborRel(Outer1,Inner) :- betweenRel(Outer1,Inner,Outer2).
```

Most Prolog systems do not employ loop detection for inferential querying, hence would use the `betweenRel` rule in a circular fashion. We thus replace it with the following 2-premise variant, adding a `lexicographic(X,Y)` predication to check whether the names of `X` and `Y` are in lexicographically ascending order (`lexicographic` is definable, restricted to ground calls, via a Prolog `string-less-or-equal` built-in):

```
betweenRel(Outer2,Inner,Outer1) :-  
    lexicographic(Outer1,Outer2), betweenRel(Outer1,Inner,Outer2).
```

For the query `neighborRel(X,Y)` Prolog uses the `neighborRel` rule and the `betweenRel` fact to deduce the first `yes` answer with bindings `X=canada`, `Y=usa`. When the next solution is requested, Prolog also uses the `betweenRel` rule to deduce the second `yes` answer with bindings `X=mexico`, `Y=usa`. When the next solution is again requested, Prolog deduces that there are no (more) answers.

## 2.2 Object-Centered Rule Languages

In contrast to  $n$ -ary relational rule languages, object-centered rule languages distinguish an entity as the object that is shared as the first argument by  $n$  binary relations ( $n \geq 0$ ). An object consists of a central globally unique Object Identifier (OID) typed by zero or more classes and described by an unordered collection (usually, a bag, i.e. a multiset) of  $n$  binary relations called *slots* leading to other objects. Slots correspond to notions in many modeling and programming languages, including *fields* in object-oriented languages like C++, Java, C#, and JSON, *attribute-value pairs* (as used for representing training examples) in machine learning, *features* in feature grammars [65]. Following the AI and Semantic Web terminology of F-logic, RIF, and PSOA RuleML, in this dissertation OID-describing slotted terms will be called *frames*. Object-centered languages constitute a (declarative) modeling core, which can be extended to (procedural) object-oriented languages by adding dynamic updates of slots.

For example, corresponding to the ternary *betweenRel* predicate in the preamble of Section 2.1, a *betweenObj* class can be used to type an OID *b1* described by three slots in any order, as in the following frame fact, where the

positional arguments become slot fillers and the slot names represent the roles of the arguments:

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

The symbol ‘#’ describes the type, *betweenObj*, of object *b1*. Each ‘→’ symbol describes a slot of *b1*, with the slot name and the slot filler on the left-hand and right-hand sides, respectively. The symbols ‘[’ and ‘]’ in the following are used to notate untyped frames.

Such a frame fact is equivalent to a conjunction of a typing fact  $b1\#betweenObj$ , and three separate untyped RDF-triple-like single-slot facts  $b1[outer1 \rightarrow canada]$ ,  $b1[inner \rightarrow usa]$ , and  $b1[outer2 \rightarrow mexico]$ . We call such a transformation of an arbitrary frame to a conjunction of single-slot frames *slottribution* [15]. By commutativity of conjunctions, altering slot orders in a fact does not change its meaning. Moreover, the conjunction of single-slot frames resulting from slottribution can be used to show that a frame with a given set of slots entails all frames having any subset of those slots.

The outer1-outer2 symmetry of *betweenRel* can be captured here even without a rule by replacing the two slots *outer1* and *outer2* by one two-valued slot *outer* (written here as two slots with the same name and different fillers):

$$b1\#betweenObj(outer \rightarrow canada, outer \rightarrow mexico, inner \rightarrow usa).$$

Some frame languages use a set-like shorthand syntax for multi-valued slots,

which would result in the following:

$$b1\#betweenObj(outer \rightarrow \{canada, mexico\}, inner \rightarrow usa).$$

Using *betweenObj*, the neighbor rule introduced in the preamble of Section 2.1 can be rewritten in the following manner:

$\forall B, Out, In :$

$$Out[neighborSlot \rightarrow In] \Leftarrow B\#betweenObj(outer \rightarrow Out, inner \rightarrow In)$$

The binary relation *neighborRel* in the previous examples is now modeled as a slot called *neighborSlot*. This rule can derive *canada*[*neighborSlot*  $\rightarrow$  *usa*] and *mexico*[*neighborSlot*  $\rightarrow$  *usa*] from, respectively,  $b1\#betweenObj(outer \rightarrow canada, inner \rightarrow usa)$  and  $b1\#betweenObj(outer \rightarrow mexico, inner \rightarrow usa)$ , both obtained via partial slottribution of the above three-slot fact.

Besides frames, taxonomies, i.e. hierarchical relationships among classes, constitute important knowledge in object-centered modeling. A class *c1* is a subclass of another class *c2*, written as  $c1 \#\# c2$ , if all instances of *c1* are also instances of *c2*. For example, based on different spatial relations of the outer and inner values, we can introduce three subclasses of *betweenObj* adapted from the region connection calculus [63]: for *betweenDC*, *betweenEC*, and *betweenPO* the inner element is disconnected, externally connected, and partially overlapping with *both* of the outer elements, respectively. The taxonomy

is written as:

*betweenDC ## betweenObj*

*betweenEC ## betweenObj*

*betweenPO ## betweenObj*

This information can be used for inheritance from a class to its subclasses. For example, with the subclass statements, the above *neighborSlot* rule would also succeed based on *betweenDC* facts.

### 2.2.1 N3

Notation 3 (N3) [6, 12, 13] is a rule language which extends the object-centered (graph-like) abstract syntax of the Resource Description Framework (RDF) [33, 43]. N3 models instance-instance, instance-class, as well as class-level relationships as a set of RDF triples  $(s, p, o)$ , which can be seen as 1-slot frames where  $s$  is the object to be described and  $(p, o)$  is its only slot, with  $p$  the slot name and  $o$  the slot filler.<sup>12</sup> For example, the triple

```
:USA :name "United States of America".
```

describes the object `:USA` with a slot `(:name, "United States of America")`. As an extension of the Web-based language RDF, N3 denotes objects by Uniform Resource Identifiers (URIs), which can be either locators (URLs) referring to accessible resources on the Web or names (URNs) of resources. The ‘:’ prefix of URIs in the above example is a shortcut for a long URI prefix defined in

---

<sup>12</sup>In RDF terminology,  $s, p, o$  are called *subject*, *property*, *object*, respectively. Here we use the more common terminology of object-oriented languages, also adopted in F-logic, RIF and PSOA RuleML.

the document containing the triple.

Multiple slots of an object in N3 can also be written in a more compact manner. For example, the three slots of object `:b1` shown in the preamble of Section 2.2 would be initially written as three triples:

```
:b1 :outer :Canada;  
:b1 :inner :USA;  
:b1 :outer :Mexico.
```

These can be grouped around the central object, `:b1`, as in frames:

```
:b1 :outer :Canada;  
    :inner :USA;  
    :outer :Mexico.
```

Type information can be added via a special slot name `a` (from “is a”), which is N3’s shorthand for the RDF property `rdf:type`:

```
:b1 a :betweenObj.
```

N3 adopts class hierarchies directly from RDF Schema (RDFS) via another special property, `rdfs:subClassOf`:

```
:betweenDC rdfs:subClassOf :betweenObj.  
:betweenEC rdfs:subClassOf :betweenObj.  
:betweenPO rdfs:subClassOf :betweenObj.
```

Again building on triples, N3 also allows the use of conjunctions, existential quantifiers, universal quantifiers and implications (rules) to form compound N3 formulas.

Here is the N3 representation of the *neighborSlot* rule introduced in the preamble of Section 2.2:

```

@forall B,Out,In.
{
  B a :betweenObj;
    :outer Out;
    :inner In.
} => { Out :neighborSlot In. }

```

## 2.3 Object-Relational Rule Languages

In Sections 2.1 and 2.2, we introduced relational and object-centered rules. Rule languages have also been developed to combine these two paradigms, either in a heterogeneous or homogeneous way. The *heterogeneous* approach simply allows the mixed use of both relational and frame formulas, e.g. in rule conditions. In contrast, the *homogeneous* approach defines new integrated formulas that generalize the formulas of both paradigms. For example, generalizing the *betweenRel* facts in Section 2.1 and the *betweenObj* facts in Section 2.2, we allow combined facts with a class *betweenObjRel* that types OIDs described by both positional and slotted arguments, as shown in the following *b3* description:

$$b3\#betweenObjRel(\textit{canada}, \textit{usa}, \textit{mexico}, \\ \textit{dimensionality} \rightarrow 2, \textit{orientation} \rightarrow \textit{SouthNorth}).$$

The fact describes the instance *b3* using a 3-tuple and two slots, as well as using *betweenObjRel* as the class for typing *b3*. The tuple with its fixed positions is useful to model the three ‘required’ arguments introduced in *betweenRel*,

while the easily extended set of slots with their extant name tags model two newly introduced ‘optional’ arguments. The *dimensionality* slot parameterizes betweenness as a 2D (planar) rather than, e.g., 1D (linear) or 3D (spatial) relationship. The *orientation* slot indicates betweenness reading in the top-down (SouthNorth) rather than, e.g., the strict bottom-up (SouthNorth) or a composite direction.

Homogeneous object-relational representation has certain advantages compared to purely relational or purely object-centered representations: 1) Compared to the purely relational representation, it allows to freely add ‘optional’ slots to a relation without extending existing argument tuples, which may lead to changes to other KB facts. 2) Compared to the purely object-centered representation, it can fix the position and co-occurrence of some or all of the ‘required’ arguments so that the syntax (and storage) becomes more compact and the representation becomes more precise.

Using the *orientation* slot of *betweenObjRel*, we can define a new predicate *orientedNeighbor* (with an object-existential, slot-parameterized rule conclusion) which specializes the relational predicate *neighborRel* of Section 2.1:

$$\begin{aligned} \forall B, Outer1, Inner, Outer2, Orient : \\ (\exists A : A\#orientedNeighbor(Outer1, Inner, orientation \rightarrow Orient)) \Leftarrow \\ B\#betweenObjRel(Outer1, Inner, Outer2, orientation \rightarrow Orient) \end{aligned}$$

Note that only the relevant slot, *orientation*, is accessed by the rule condition, whose value, *Orient*, is specified to be the same in the rule conclusion.

### 2.3.1 Heterogeneous Combination: F-logic and RIF

Frame logic (F-logic) [48, 49] is the most well-known approach for combining object-centered modeling with logic formalisms. F-logic uses first-order ground terms to represent object identifiers of frames. A *molecular formula* in F-logic, analogous to atomic formulas in FOL, can be of the following forms: (1) an *object molecule* describing an object with its single-valued or set-valued attributes; (2) a *signature expression* describing the types of attributes for classes; (3) a *class membership* or a *subclass relationship* formula. Compound formulas are built on top of molecular formulas using first-order connectives  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\exists$ , and  $\forall$ .

W3C's *Rule Interchange Format* (RIF) [22], built on top of F-logic, is a standard for exchanging rules among rule-based applications and systems. RIF defines a family of rule languages called dialects. Each dialect has its presentation syntax and XML syntax, as well as well-defined semantics. In this dissertation we focus on the Basic Logic Dialect (RIF-BLD) [21], which constitutes a rule language that has the expressiveness of the Horn subset of FOL with equality. RIF-BLD allows frames with slotted arguments as in F-logic as well as functions and predicates applied to positional arguments as in FOL and to slot-like 'named' arguments as in RuleML/POSL [42]. Basically, named arguments and slots are "attribute-value" pairs of the form  $p \rightarrow v$  as introduced in Section 2.2, which used " $\rightarrow$ " instead of " $->$ ".

In RIF, constants are expressed by IRIs, and variables are strings starting with '?'. Terms can be of the following forms: (1) Simple terms which are constants or variables; (2) Positional terms, which are functions/predicates applied to positional arguments; (3) Named-argument terms, which are func-

tions/predicates applied to slotted arguments; (4) External terms of the form `External(t)`, where `t` is a positional or named-argument term. They refer to built-ins or ‘foreign’ (externally defined) procedures.

Atomic formulas in RIF are of the following forms: (1) Predicate applications `p(...)` describing a predicate `p` applied to positional or slotted arguments; (2) Equalities `a=b`; (3) Class memberships `a#b`; (4) Subclass relationships `a##b`; (5) (Untyped) frames `o[p1->v1 . . . pk->vk]` describing an object `o` with `k` attribute-value pairs; (6) External formulas `External( $\phi$ )`, which are used to call externally defined functions such as arithmetic functions. Compound formulas in RIF are built on atomic formulas using similar constructs as in FOL, mostly with a prefix syntax: `And(...)` for conjunction, `Or(...)` for disjunction, `Exists ?X1 . . . ?Xn (...)` and `Forall ?X1 . . . ?Xn (...)` for existential and universal quantification, and `:-` for rule implication. A universal rule is of the form `Forall ?X1 . . . ?Xn ( $\phi$  :-  $\psi$ )`, where the *condition formula*  $\psi$  can be an atomic, a conjunction, a disjunction, or an existential, and the *conclusion formula*  $\phi$  can be an atomic or a conjunction of atomic formulas.

As we can see, in RIF, atomic formulas with positional and slotted arguments can be used, but only separately, making it a heterogeneous object-relational language.

### 2.3.2 Homogeneous Combination: PSOA RuleML

PSOA RuleML [15] is an object-relational Web rule language that generalizes, e.g., RIF-BLD and POSL [42] by a homogeneous integration of relationships and frames into positional-slotted object-applicative (psoa)<sup>13</sup> terms, where the

<sup>13</sup>We use the upper-cased “PSOA” as a qualifier for the language and the lower-cased “psoa” for its terms.

most often used single-tuple subset has the general form (the syntax of arbitrary psoa terms, using explicit square brackets for tuples, will be introduced in Section 3.2):

$$\mathbf{Oidless}: f(t_1 \dots t_n p_1 \rightarrow v_1 \dots p_k \rightarrow v_k) \quad (2.1)$$

$$\mathbf{Oidful}: o\#f(t_1 \dots t_n p_1 \rightarrow v_1 \dots p_k \rightarrow v_k) \quad (2.2)$$

Both (2.1) and (2.2) apply a function or predicate<sup>14</sup>  $f$  – in (2.2) identified by an OID  $o$  via a membership,  $o\#f$ , of  $o$  typed by  $f$  – to a tupled (positional) descriptor having arguments  $t_1 \dots t_n$  and to a bag of slotted descriptors  $p_j \rightarrow v_j$ ,  $j = 1, \dots, k$ , each pairing a slot name (attribute)  $p_j$  with a slot filler (value)  $v_j$ .

A psoa term can be interpreted as a (psoa) expression, denoting an individual, or a (psoa) atom, denoting a truth value. The interpretation as an expression was earlier allowed for both oidless and oidful psoa terms (i.e., (2.1) and (2.2) above) but will be restricted to oidless psoa terms in the dissertation, as explained in Section 3.1. The interpretation as an atom is permitted for both (2.1) and (2.2) on the top-level, while restricted to (2.2) when embedded, as explained in Section 3.1.

An OID  $o$  typed by the *root* (“universal”, “any”, ...) *predicate*  $f = \text{Top}$  is considered as untyped. Untyped atoms of the form  $o\#\text{Top}(\dots)$  are a generalization of untyped frames of the form  $o[\dots]$  in Section 2.3.

The OID, tuples, and slots in a psoa atom are all optional. For an oidless psoa atom, not provided with a ‘user’ OID, *objectification* can generate a ‘sys-

---

<sup>14</sup>PSOA’s notion of ‘predicate’ can be regarded as a generalization of RDF’s notion of ‘class’. However, RDF’s synonymous notions of ‘property’ and ‘predicate’ correspond to F-logic’s, RIF’s, and PSOA’s notion of ‘slot name’.

tem’ OID to make it oidful, as detailed in Section 5.3. Given an atom with a (user-provided or system-generated) OID, the slotribution transformation introduced in Section 2.2 for slots can be generalized to description for both tuples and slots, which will be explained in Section 5.6.

The alphabet of PSOA RuleML includes a single set **Const** of individual, function, and predicate constants – to prepare functional-logic integration as, e.g., in Relfun, Hilog, and RIF – as well as a set **Var** of variables.

Constants have the form `"literal"^^symspace`, where `literal` is a Unicode string and `symspace` is a symbol space identifier. Six kinds of constant shortcuts are defined in [62], including numbers, strings, Internationalized Resource Identifiers (IRIs), and ‘\_’-prefixed *local constants* (e.g., `_a`) whose `symspace` is specialized to `rif:local`. In PSOA RuleML, number shortcuts with a decimal point ‘.’ are interpreted as in the symbol space `xs:double` instead of `xs:decimal` like in RIF. This is because double-precision floating-point numbers are more often used in programming and specification languages. An IRI constant `"..."^^rif:iri` has the angular-form shortcut `<...>`. It can be further abbreviated using a namespace prefix ending with ‘:’, e.g., an IRI `<http://ex.org/a>` is a shortcut of `<http://ex.org/a^^rif:iri` and can be abbreviated as `:a` if the KB contains a declaration `Prefix(:<http://ex.org/>)` for the prefix ‘:’. In contrast to other kinds of ‘global’ constants that can be reused among different KBs, local constants in different KBs are regarded as different and will be renamed apart when KBs are merged, as explained in Section 5.9. A stand-alone ‘\_’ is not a local constant but a local-constant generator. `Top` is regarded as a special constant in PSOA RuleML, denoting the root class.

Variables in PSOA are ‘?’-prefixed symbols, e.g., ?X.

The syntax of PSOA RuleML is built on *terms*. A *base term* is a *simple term*, that is a constant or variable, a psOA expression, or an *external term* `External(f(...))`<sup>15</sup>. An *atomic formula* is a psOA atom in the form of terms (2.1) or (2.2), a subclass<sup>16</sup> term `c1##c2`, an equality term `t1=t2`, or an external term `External(f(...))`<sup>17</sup>, where `c1`, `c2`, `t1`, and, `t2` are all base terms. Like in RIF, external terms are usually employed for built-in function and predicate calls. Compound formulas can be constructed using the Horn-like subset of FOL, including conjunctions `And( $\tau_1 \dots \tau_n$ )`, disjunctions `Or( $\tau_1 \dots \tau_n$ )` in the rule body, rule implications  $\tau_1 :- \tau_2$  on the top-level of a quantification, existential quantification `Exists ?X1 ... ?Xn ( $\tau_1$ )`, and universal quantification `Forall ?X1 ... ?Xn ( $\tau_1$ )` on the top-level of a KB. Here, each  $\tau_i$  is an atomic or compound formula.

A *group formula* `Group( $\tau_1 \dots \tau_n$ )` wraps a set of fact and rule clauses into a KB. A *document formula* `Document( $d_1 \dots d_n \Phi$ )` wraps a group formula  $\Phi$  with a sequence of directives  $d_1, \dots, d_n$  for prefix and import declarations.<sup>18</sup>

The model-theoretic semantics of PSOA RuleML is defined through semantic structures [15,17]. A semantic structure  $\mathcal{I}$  is a tuple  $\langle \mathbf{TV}, \mathbf{DTS}, \mathbf{D}, \mathbf{D}_{\text{ind}}, \mathbf{D}_{\text{func}}, \mathbf{I}_C, \mathbf{I}_V, \mathbf{I}_{\text{psoa}}, \mathbf{I}_{\text{sub}}, \mathbf{I}_=, \mathbf{I}_{\text{external}}, \mathbf{I}_{\text{truth}} \rangle$ . Here  $\mathbf{D}$  is a non-empty set called the domain of  $\mathcal{I}$ . The components  $\mathbf{D}_{\text{ind}}$  and  $\mathbf{D}_{\text{func}}$  are subsets of  $\mathbf{D}$  for the interpretation of individuals and functions.  $\mathbf{I}_C$  and  $\mathbf{I}_V$  interpret constants and variables.  $\mathbf{I}_{\text{psoa}}$  interprets predicates/functions of psOA terms as

<sup>15</sup>Here,  $\mathbf{f}$  is interpreted as a function.

<sup>16</sup>“Subclass” is pars pro toto for “subpredicate” in [27].

<sup>17</sup>Here,  $\mathbf{f}$  is interpreted as a predicate.

<sup>18</sup>Since PSOATransRun 1.3, the document root RuleML, can be used for the original Document, as well as Assert for the original Group, complementing it with Query, which was originally absent.

semantic functions, which will be shown in Section 3.4.  $I_{\text{sub}}$ ,  $I_{=}$ , and  $I_{\text{external}}$  interpret subclass, equality, and external terms. A generic mapping  $I$  from terms to  $D$  can be defined using the above interpretation mappings.  $I_{\text{truth}}$  maps domain elements to the set of truth values  $TV = \{\mathbf{t}, \mathbf{f}\}$ . We will write  $\mathcal{I}.D$ ,  $\mathcal{I}.I_V$ , etc., for the components of  $\mathcal{I}$  in the rest of the dissertation.

Truth evaluation for formulas is determined by a recursive evaluation function  $TVal_{\mathcal{I}}$  defined in [15]. A semantic structure  $\mathcal{I}$  is called a *model* of a KB  $\phi$  if  $TVal_{\mathcal{I}}(\phi) = \mathbf{t}$  and  $\mathcal{I}$  conforms to all semantic restrictions (e.g., subclass and description restrictions), denoted by  $\mathcal{I} \models \phi$ . A PSOA KB  $\phi$  is said to *entail* a formula  $\psi$ , denoted by  $\phi \models \psi$ , if for every model  $\mathcal{I}$  of  $\phi$ ,  $\mathcal{I} \models \psi$  holds.

A *query* has the same form as a condition formula. Similarly to Prolog, a query is said to be *ground* if it has no variables while *non-ground* otherwise. Unlike Prolog queries that use only free variables, in PSOA RuleML query variables can be explicitly quantified in an existential scope, for which the bindings will not be shown as answers.

### 2.3.2.1 Exemplifying PSOA RuleML with Startup Examples

PSOA RuleML allows the use of different kinds of psOA atoms in clauses. In this sub-subsection we discuss a classification of rule conclusion atoms (ConcAtom) and demonstrate it with a series of KBs modeling core aspects of startup-company samples in the PSOA RuleML language. Our queries to these KBs, like all the examples in PSOA presentation syntax (PSOA/PS) given in the dissertation, can be executed in our PSOATransRun implementation of PSOA RuleML (see Chapter 9).

In a rule that has a conjunctive conclusion composed of multiple conjoined

ConcAtoms, each of those atoms can belong to a different category in the classification. In this sub-subsection, those conjunctive conclusions will not be further considered, because they do not contribute to the classification of ConcAtoms themselves.

The top-level classification of ConcAtoms reuses the oidless/oidful separation explained in Section 2.3.2. Oidful ConcAtoms can be further classified based on the kind of OID term used: constant OID, variable OID, or function-application OID, explained in the following.

1. Without explicit OID (oidless)

Such an atom can be seen as having an implicit existential OID, which is normally replaced with a constant (analogous to 2.1) or a function application (a special case of 2.3 where the arguments include all universal variables) during a reasoning process.

2. With explicit OID (oidful)

- 2.1 Constant OID

- 2.2 Variable OID

Variables in rules can be further described from different dimensions (applied to both 2.2 and each variable parameter in 2.3):

- existential/universal
- conclusion-only/condition
- OID/non-OID variables
- whether it is bound to the result of an external function call

The following is a further classification of 2.2, based on the first two dimensions.

2.2.1 taken unchanged from a universal condition variable (for ‘immediate enrichment’ within a single rule application)

2.2.2 occurs only in the conclusion

2.2.2.1 existential variable

2.2.2.2 universal variable

### 2.3 Function-application OID

A function-application OID’s parameters can be a subset of universal variables in the scope of the rule. We call the OID to be *fully parameterized* if it takes the entire set of universal variables as arguments. Otherwise, it is called *partially parameterized*. A partially parameterized function-application OID is used for ‘incremental enrichment’ of earlier applications of the same rule. A fully parameterized function-application OID having a function name occurring only in the rule is analogous to Skolem function applications.

From the modeling perspective, function-application OIDs can be underdetermined, well-determined, or overdetermined depending on its parameters. An OID is said to be well-determined if its parameter set is the exact set of arguments that determines the object identity, corresponding to primary keys in databases. It is underdetermined (resp. overdetermined) if its parameters is a subset/superset of the parameters of a well-determined version.

In the following, we give a series of Startup examples, each of which has a rule with a single-atom conclusion. The classification of the conclusion atom is shown in square brackets of the form [ConcAtom *i*. . . *k*] in the header text of each example.

Example 2.1 gives a small KB with a fact and a simple rule. The fact is a `_cofounders` atom with two positional arguments for the CEO and the CTO. Unifying the fact with its condition, the rule derives an oidless `_startup` atom `_startup(_ceo->_Ernie _cto->_Tony)` with the same arguments.

**Example 2.1.** ([ConcAtom 1]: Without Conclusion OID)

```
Document (
  Group (
    Forall ?CEO ?CTO (
      _startup(_ceo->?CEO _cto->?CTO) :- _cofounders(?CEO ?CTO)
    )

    _cofounders(_Ernie _Tony)
  )
) ◇
```

Example 2.2 enriches Example 2.1 by introducing `_equity` atoms describing the equity shares of the CEO and the CTO. The condition of the rule uses external calls to ensure that the sum of the equity percentages of the CEO and the CTO is not greater than 100. Unifying the facts with its condition, the rule derives the same atom `_startup(_ceo->_Ernie _cto->_Tony)` as in Example 2.2.

**Example 2.2.** ([ConcAtom 1]: Without Conclusion OID and With Condition External Built-in)

```
Document (
  Prefix(func: <http://www.w3.org/2007/rif-builtin-function#>)
```

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

```
Group (
```

```
  Forall ?CEO ?CTO ?CEOeqy ?CTOeqy (
```

```
    _startup(_ceo->?CEO _cto->?CTO) :-
```

```
      And(_cofounders(?CEO ?CTO)
```

```
        _equity(?CEO ?CEOeqy)
```

```
        _equity(?CTO ?CTOeqy)
```

```
      External(
```

```
        pred:numeric-less-than-or-equal(
```

```
          External(func:numeric-add(?CEOeqy ?CTOeqy)) 100)))
```

```
    )
```

```
  _cofounders(_Ernie _Tony)
```

```
  _equity(_Ernie 50)
```

```
  _equity(_Tony 30)
```

```
)
```

```
)
```

◇

Example 2.3 further enriches Example 2.2 with `_hire` atoms having the CEO and an employee as the arguments. The rule is enriched by adding `_hire` atoms in the condition and transferring the employee argument to a slot of the `_startup` atom via the rule. Applying the rule to the four facts yields the atom `_startup(_ceo->_Ernie _cto->_Tony _employee->_Kate)`.

**Example 2.3.** ([ConcAtom 1]: Without Conclusion OID, With Conclusion Slot and Condition External Built-in)

```
Document (
```

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

```
Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    _startup(_ceo->?CEO _cto->?CTO _employee->?Emp) :-
      And(_cofounders(?CEO ?CTO)
        _hire(?CEO ?Emp)
        _equity(?CEO ?CEOeqy)
        _equity(?CTO ?CTOeqy)
        External(
          pred:numeric-less-than-or-equal(
            External(func:numeric-add(?CEOeqy ?CTOeqy)) 100)))
  )

  _cofounders(_Ernie _Tony)
  _hire(_Ernie _Kate)
  _equity(_Ernie 50)
  _equity(_Tony 30)
)
)
```

◇

Example 2.4 is a variant of Example 2.1 that changes `_cofounders` to a ternary `_conamers` having an additional third argument. The conclusion `_startup` atom uses `?Comp` directly as the OID. In the other examples of the sub-subsection, we assume that every pair of CEO and CTO can only found one company, so we will keep using the binary `_cofounders`. Apply-

ing the rule to the fact yields the atom `_Chi4corp#_startup(_ceo->_Ernie _cto->_Tony)`.

**Example 2.4.** ([ConcAtom 2.2.1]: With Conclusion OID Taken Unchanged from Universal Condition non-OID Argument)

```
Document (
  Group (
    Forall ?CEO ?CTO ?Comp (
      ?Comp#_startup(_ceo->?CEO _cto->?CTO) :-
        _conamers(?CEO ?CTO ?Comp)
    )

    _conamers(_Ernie _Tony _Chi4corp)
  )
)

```

◇

Example 2.5 has an oidful `_startup` fact with an explicit OID `_Chi4corp`. The rule can derive extra `_employee` slots attached to the same OID using a conjunction of an oidful `_startup` atom and a `_hire` relationship between the CEO and the employee. Applying the rule to the `_cofounders` fact and each of the two `_hire` facts separately yields `_Chi4corp#Top(_employee->_Kate)` and `_Chi4corp#Top(_employee->_Fred)`, which can be merged with the fact `_Chi4corp#_startup(_ceo->_Ernie _cto->_Tony)`, resulting in the single atom `_Chi4corp#_startup(_ceo->_Ernie _cto->_Tony _employee->_Kate _employee->_Fred)`.

**Example 2.5.** ([ConcAtom 2.2.1]: With Conclusion OID Taken Unchanged from Universal Condition OID)

```

Document (
  Group (
    Forall ?Comp ?CEO ?CTO ?Emp (
      ?Comp#Top(_employee->?Emp) :-
        And(?Comp#_startup(_ceo->?CEO _cto->?CTO)
          _hire(?CEO ?Emp))
    )
    _Chi4corp#_startup(_ceo->_Ernie _cto->_Tony)
    _hire(_Ernie _Kate)
    _hire(_Ernie _Fred)
  )
)

```

◇

Example 2.6 is a variant of Example 2.3 that uses a function-application OID `_startupID(?CEO ?CTO)` as well as omitting the modeling of equity shares. Here, the parameters `?CEO` and `?CTO` constitute the exact subset that determines the identity of the company, so that the OID is well-determined and the the same OID is formed for derived company atoms having the same `?CEO` and `?CTO`. Applying the rule to the `_cofounders` fact and each of the two `_hire` facts separately yields

```

_startupID(_Ernie _Tony)#_startup(_ceo->_Ernie _cto->_Tony
                                     _employee->_Kate)

```

and

```
_startupID(_Ernie _Tony)#_startup(_ceo->_Ernie _cto->_Tony
                                     _employee->_Fred)
```

The atoms share the same OID and can be merged into a single atom

```
_startupID(_Ernie _Tony)#_startup(_ceo->_Ernie _cto->_Tony
                                     _employee->_Kate _employee->_Fred)
```

If the OID is replaced with `_startupID(?CEO ?CTO ?Emp)`, taking also the employee argument `?Emp`, it is overdetermined, resulting in different OIDs to be formed for derived company atoms having the same `?CEO` and `?CTO` but with two different employee slot. If the OID is replaced with `_startupID(?CEO)`, it is underdetermined, where the same OID is formed for derived company atoms having only the same `?CEO`.

**Example 2.6.** ([ConcAtom 2.3]: With Conclusion Function-Application OID)

```
Document (
  Group (
    Forall ?CEO ?CTO ?Emp (
      _startupID(?CEO ?CTO)#_startup(_ceo->?CEO _cto->?CTO
                                     _employee->?Emp) :-
      And(_cofounders(?CEO ?CTO) _hire(?CEO ?Emp))
    )

    _cofounders(_Ernie _Tony)
    _hire(_Ernie _Kate)
    _hire(_Ernie _Fred)
  )
)
```

◇

## 2.4 Interoperation of Rules

Along with the rapid increase of the number of Web applications, interoperation between various knowledge formalisms used by different applications becomes more and more important for integrating and reusing knowledge. A systematics of rule interoperation has evolved in recent years [18]. The interoperation of rules can be intra-language or inter-language. *Intra-language* interoperation [8, 41] is usually employed for upgrading between different versions of a standard, or performing normalization of a rule KB. *Inter-language* interoperation, however, is needed for reusing existing KBs written in other formats, developing applications supporting multiple languages, or providing services for a new rule language through existing tools in another language.

Providing interoperation between multiple languages always relies on translators at its core but can be done in various ways. A brute-force approach is all-to-all translations, i.e. the implementation of bidirectional translations for each pair of languages. This approach can result in the highest efficiency for each of the customized translators. However, it requires  $n(n-1)$  translators to be implemented for  $n$  languages, which is not practical when  $n$  is large. An alternative approach is distinguishing a language as canonical interchange format, and implementing bidirectional translations between each language and the canonical language. This approach is much more practical, as it requires  $2n - 2$  translators to be implemented for  $n$  languages and allows better reuse of modules among different translators. A third approach is implementing a ring translation. Let  $L_1, L_2, \dots, L_n$  be a set of rule languages, this approach implements translators from  $L_i$  to  $L_{i+1}$ ,  $1 \leq i < n$ , as well as one from  $L_n$  to  $L_1$ . It is easy to see that a translation from any  $L_i$  to  $L_j$  can be done

by a composition of translators [14]. For  $n$  languages, this method requires to implement  $n$  translators, which is the fewest among the three approaches. However, the efficiency of translation with up to  $n-1$  composed translators can be low when  $n$  is large. Moreover, any incorrectness of a single translator can affect many translations using this approach, and small errors can multiply across up to  $n - 1$  steps. Hence, the second approach is the one most widely studied and used.

There have been several standardization efforts to define interchange languages for rule interoperation. The Rule Markup Language (RuleML) [26] is a unifying family of XML-serialized rule languages spanning across all industrially relevant kinds of Web rules.<sup>19</sup> The interoperation between RuleML and RDF [23] and between RuleML and OWL/SWRL [8] have been studied. Reaction RuleML has been used as the interchange language of Rule Responder [25]. The European Network of Excellence REWERSE (Reasoning on the Web with Rules and Semantics) proposed R2ML, the REWERSE Rule Markup Language, which offers a solution for interchanging rules between heterogeneous systems combining RuleML, SWRL, and OCL. ISO Common Logic [69] provides a framework for a family of logic-based languages, e.g. FOL, Horn Prolog, Conceptual Graphs, etc. It defines Common Logic Interchange Format (CLIF) and Conceptual Graph Interchange Format (CGIF) as its main dialects. W3C's RIF [58] is another interchange format for rule languages, which has been introduced in Section 2.3. OntoIOP<sup>20</sup> [54] is a recent standardization effort by OMG, which defines a Distributed Ontology Language (DOL) [55] that makes heterogenous ontologies interoperable under one

---

<sup>19</sup>[http://wiki.ruleml.org/index.php/Introducing\\_RuleML](http://wiki.ruleml.org/index.php/Introducing_RuleML)

<sup>20</sup><http://ontoiop.org/>

framework.

In the literature, translations between various logic formalisms have been studied. RuleML-facilitated translation between N3 and Prolog was proposed [24], where n-ary relations in Prolog are translated into frames in N3 through objectification, while positionalization is used for the inverse translation. A ring translation was proposed for interoperation of RIF presentation syntax, RIF XML syntax, RuleML XML syntax, and RuleML/POSL syntax [14]. A translation of frames for the implementation of the knowledge machine language to slottribution-like conjunctions of triples as answer set programs (ASPs), a variant of logic programs, has been recently proposed [9]. Translations from various subsets of declarative FOL theories to procedural logic programs was studied [29, 50, 56, 72], using different approaches. Translations between various Description Logic (DL) fragments and rule languages were also presented and studied [46, 51, 52]. In particular, Krötzsch *et.al.* [52] identified a maximal fragment of DL  $\mathcal{ALC}$  that can be translated into Datalog.

Semantic data access or Ontology-Based Data Access (OBDA), complemented by Rule-Based Data Access (RBDA) [19], is a research area that aims at facilitating interoperation between Semantic Web applications and relational data stored in databases. The idea is to expose data in a relational database as a set of facts,  $A$ , through a pre-defined mapping. Combined with a set of rules,  $T$ , describing a high-level conceptual view of the data in a knowledge representation language, one can pose semantic query  $q$  using the language and get an inferential answer. A widely used query answering approach is to first rewrite  $q$  into a database query  $q'$  based on  $T$ , and then execute  $q'$  on the relational data using highly optimized and mature relational database

engines. RDB2RDF is an ongoing W3C standardization effort with the aim of exposing relational data as RDF for Semantic Web applications [5,34]. The mapping from relational schemas to object-centered schemas is defined via the R2RML language [34]. In order to map a database record into RDF triples, a globally unique OID representing the record needs to be created [5]. If the database table containing the record has a primary key, the OID is an encoding of the key value(s) of the record. Usually, the encoding comes with a table-specific prefix to ensure the global uniqueness of the OID. Otherwise, if the table has no primary key, an OID is generated and the record is identified as a (local) RDF blank node.

Besides for declarative Semantic Web applications focused here, data stored in relational databases can also be accessed and procedurally updated in an object-oriented manner for object-oriented programming applications via object-relational mappings, which are widely used by software developers. Object-relational mapping systems in widespread use include Java Hibernate<sup>21</sup>, Microsoft's ADO.NET Entity Framework [3], EclipseLink<sup>22</sup>, Oracle's TopLink<sup>23</sup>, and Ruby on Rails<sup>24</sup>. To build an application on top of an object-relational mapping system, one needs to specify the mapping between the database schema and the programming-language-specific object abstraction. Basically, this mapping defines the correspondence between the properties of classes and the columns of database tables. With such mappings in place, object-relational mapping systems permit specifying queries and updates on the object-oriented level by first translating the object-oriented requests to corresponding SQL

---

<sup>21</sup><http://www.hibernate.org/>

<sup>22</sup><http://www.eclipse.org/eclipselink/>

<sup>23</sup><http://www.oracle.com/technology/products/ias/toplink/index.html>

<sup>24</sup><http://rubyonrails.org/>

commands and then executing the commands through database engines. The systems also re-organize query answers from database engines into objects for programmatic access.

There have also been studies on the formalization of logic translations [59, 60]. Logic systems can be abstracted using a meta-notion named *institutions* [38, 59], which formalizes signatures, sentences, models and satisfaction relations based on category theory. Language translations are modeled as *institution comorphisms*, which characterize mappings between two institutions.

# Chapter 3

## Revising the PSOA RuleML Language for Version 1.0

In this chapter we discuss a revision of the knowledge representation, syntax, and semantics of the original PSOA RuleML language defined in [15], achieving PSOA RuleML 1.0. Most of the work has been presented in our papers [74] and [27]. Section 3.1 explains the syntax and semantics of embedded psoa terms. Section 3.2 explains the addition of perspectival knowledge modeling. Section 3.3 explains the revision of the EBNF grammar that defines PSOA syntax. Section 3.4 explains the revision of the model-theoretic semantics.

### 3.1 Syntax and Semantics of Embedded Psoa Terms

In PSOA RuleML, embedded psoa terms can be semantically classified into expressions or atoms, and syntactically classified into oidful or oidless terms.

We will discuss different combinations of the two dimensions in the following.

- **Expressions**

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

The semantics for expressions uses the  $I$  and  $I_{\text{psoa}}$  mappings, whose definitions in [15] will be revised in Section 3.4. The new definition of  $I_{\text{psoa}}$  will interpret  $I(f)$  into a semantic function that takes the empty set for oidless psoa terms, besides a bag of tuples and a bag of slots, and returns an arbitrary domain element. The lack of an OID for an expression also prevents its description, thus avoiding that its arbitrary value gets ‘distributed over’ the resulting conjunction of single-slot/tuple expressions or ‘absorbed by’ its truth value.<sup>25</sup>

- **Atoms**

A psoa atom is a psoa term interpreted as a predicate application. Embedded atoms have been widely used in object-centered and object-relational languages such as RDF, N3, and Flora-2/F-logic as a shorthand notation. An atomic formula containing an embedded atom can be unnested into a conjunction of trimmed formulas, which will be illustrated in Section 5.2.

---

<sup>25</sup>Relfun’s *valued conjunctions* of functional-logic expressions only retain the value of the right-most, ‘&’-prefixed conjunct: <http://www.relfun.org>.

In RDF and N3, embedded atoms are oidless and can be objectified and extracted using blank node identifiers, while in Flora-2/F-logic they are oidful. Because in PSOA/PS both functions and predicates share the same alphabet `Const` of constants (cf. Section 2.3.2), oidless embedded atoms and (always) oidless embedded expressions have the same syntactical form and cannot be distinguished. Thus, we prohibit oidless embedded atoms, instead always using an embedded oidless term as an expression and an embedded oidful term as an atom. The distinction applies only to embedded psoa terms but not to top-level terms, where both oidful and oidless forms are interpreted as atoms. To indicate an RDF/N3-like embedded blank node, the atom can use an explicit OID, thus: (1) `_#f(...)` in ground facts, where ‘\_’ is the fresh-constant generator; (2) `Exists ?0 (... ?0#f(...))` in non-ground facts and rule conclusions; and (3) `?#f(...)` in queries / rule premises, where ‘?’ is the anonymous variable. The modeling is also employed in the interoperation from N3 to PSOA in Section 6.2.

## 3.2 Representing Perspectival Knowledge

In Section 2.3.2, we introduced the original version of PSOA RuleML from [15], where descriptors of psoa atoms can be either tupled or slotted. In this section, orthogonally to the tupled/slotted distinction, we allow a descriptor in an atom to be independent or dependent on the atom’s predicate. A descriptor dependent on a predicate exploits the predicate scope and can only be successfully queried with the same predicate. In contrast, an independent descriptor does not exploit the predicate scope and can also be successfully queried with a different predicate. Incorporating the dependent vs. independent distinction

of descriptors, the dimensions of psoa atoms are introduced in Section 3.2.1, followed by examples in Section 3.2.2.

### 3.2.1 Dimensions of Psoa Atoms

In the original PSOA, a psoa atom was characterized by two orthogonal dimensions [17]: (1) whether the atom is oidful or oidless and (2) whether the atom’s descriptors are tupled,<sup>26</sup> slotted, or tupled+slotted. In the following, a metamodel, shown in Figure 3.1, is introduced for PSOA RuleML 1.0, refining the original dimensions (see  $D_1$  and  $D_2$  below) and adding a new perspectivity dimension (see  $D_3$  below) for dependent and independent descriptors [27].

Before introducing dimensions  $D_1$ – $D_3$ , we first start with a quantitative dimension  $D_0$ . It 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. 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 and perspeneutral/perspectival distinctions non-applicable to them. For non-empty atoms,  $D_0$  is used to define  $D_2$ , as shown by the  $D_0 \rightarrow D_2$  mapping in Figure 3.1.

There are three orthogonal qualitative dimensions  $D_1$ – $D_3$  for non-empty atoms generating eighteen subcubes. Refining the original six quadrants of the “psoa table” [17], the PSOA RuleML 1.0 subcubes will be labeled according to

---

<sup>26</sup>To emphasize the option of multiple tuples, the original terms “positional/slotted” have been replaced by “tupled/slotted” in most cases.

three layers of six, for non-empty atoms that are **perspeneutral**, **perspectival**, and **perspeneutral+perspectival**, 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,  $D_0(m=1,k=0)$ -specialized to oidless, single-tupled, perspectival atoms, having the common name *relationships*.

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 original psoa table are also created by these dimensions as the above-mentioned pv1 subcube (specialized 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,<sup>27</sup> a frame affords one or more ( $k \geq 1$ ) 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 PSOA RuleML 1.0, dimensions  $D_0$ - $D_2$  are augmented by the *perspectivity* dimension  $D_3$  defined based on the dependency kind (i.e., independent or dependent) of its descriptors: A *perspeneutral* atom comprises one or more (predicate-)independent descriptors; a *perspectival* atom comprises

---

<sup>27</sup>Here, “plain” refers to an ordinary slot with a non-tuple-valued filler; cf. Footnote 28.

one or more (predicate-)dependent descriptors; a *perspeneutral+perspectival* atom combines one or more independent plus one or more dependent descriptors. In the systematics of Figure 3.1, the zeroth dimension is indicated by inequality or equality pairs involving  $(m,k)$ , the first and second dimensions are constituted by the columns and rows of each layer, and the third dimension is unraveled layer-wise. As explained 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) can be  $D_0(m=1,k\geq 1)$ -specialized to *shelframes*; likewise, oidless, tupled+slotted, perspectival atoms (pv5) can be  $D_0(m=1,k\geq 1)$ -specialized 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,  $k\geq 0$  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,  $k\geq 0$  slot, perspectival atoms* can be constructed as the union of relationship and relpairship atoms.

**Origin:**

$$D_0(m=0,k=0) \quad \quad \quad \} \quad \text{descriptorless}$$
**Mapping:**

$$\left. \begin{array}{l} D_0(m \geq 1, k=0) \longrightarrow D_2: \text{tupled} \\ D_0(m=0, k \geq 1) \longrightarrow D_2: \text{slotted} \\ D_0(m \geq 1, k \geq 1) \longrightarrow D_2: \text{tupled+slotted} \end{array} \right\} \quad \text{descriptorful}$$

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

<b>D<sub>3</sub>: perspeneutral</b>	D <sub>1</sub> : oidless	D <sub>1</sub> : oidful
D <sub>2</sub> : tupled	pn1	pn2. D <sub>0</sub> (m=1,k=0): shelves
D <sub>2</sub> : slotted	pn3	pn4: <b>frames</b>
D <sub>2</sub> : tupled+slotted	pn5	pn6. D <sub>0</sub> (m=1,k≥1): shelfframes

<b>D<sub>3</sub>: perspectival</b>	D <sub>1</sub> : oidless	D <sub>1</sub> : oidful
D <sub>2</sub> : tupled	pv1. D <sub>0</sub> (m=1,k=0): <b>relationships</b>	pv2
D <sub>2</sub> : slotted	pv3: pairships	pv4
D <sub>2</sub> : tupled+slotted	pv5. D <sub>0</sub> (m=1,k≥1): relpairships	pv6

<b>D<sub>3</sub>: perspeneutral+perspectival</b>	D <sub>1</sub> : oidless	D <sub>1</sub> : oidful
D <sub>2</sub> : tupled	pp1	pp2
D <sub>2</sub> : slotted	pp3	pp4
D <sub>2</sub> : tupled+slotted	pp5	pp6

Figure 3.1: Basic metamodel of PSOA RuleML 1.0: Multi-dimensional psaa atoms.

### 3.2.2 Formal Facts and Queries

In this section, we introduce the syntax of dependent and independent descriptors and demonstrate the use of them through example KBs and queries.

Dependent vs. independent descriptors are uniformly notated using, respectively, the dual “+” vs. “-” marks, leading to four kinds of descriptors:

- For tuples, “+” vs. “-” are used as prefixes for the square brackets, yielding the syntaxes  $+[ \dots ]$  vs.  $-[ \dots ]$ .
- For slots, “+” vs. “-” are used as shafts of the infix arrows, yielding the syntaxes  $\dots + > \dots$  vs.  $\dots - > \dots$ .

The general kinds of psosa terms have these two forms ( $m^+, m^-, k^+, k^- \geq 0$ ,  $n_{i^+}^+, n_{i^-}^- \geq 0$  for any  $i^+$  and  $i^-$  such that  $1 \leq i^+ \leq m^+$  and  $1 \leq i^- \leq m^-$ ):

$$\begin{aligned}
 \text{Oidless :} \quad & f(+[t_{1,1}^+ \dots t_{1,n_1^+}] \dots +[t_{m^+,1}^+ \dots t_{m^+,n_{m^+}^+}] \\
 & \quad -[t_{1,1}^- \dots t_{1,n_1^-}] \dots -[t_{m^-,1}^- \dots t_{m^-,n_{m^-}^-}] \\
 & \quad p_1^+ > v_1^+ \dots p_{k^+}^+ > v_{k^+}^+ \\
 & \quad p_1^- > v_1^- \dots p_{k^-}^- > v_{k^-}^-) \tag{3.1}
 \end{aligned}$$

$$\begin{aligned}
 \text{Oidful :} \quad & o\#f(+[t_{1,1}^+ \dots t_{1,n_1^+}] \dots +[t_{m^+,1}^+ \dots t_{m^+,n_{m^+}^+}] \\
 & \quad -[t_{1,1}^- \dots t_{1,n_1^-}] \dots -[t_{m^-,1}^- \dots t_{m^-,n_{m^-}^-}] \\
 & \quad p_1^+ > v_1^+ \dots p_{k^+}^+ > v_{k^+}^+ \\
 & \quad p_1^- > v_1^- \dots p_{k^-}^- > v_{k^-}^-) \tag{3.2}
 \end{aligned}$$

Note that in terms having or specializing the forms (3.1) and (3.2) any tuple can be empty, an assumption made for the rest of the dissertation unless

explicitly restricted. For the commonly used special case of single-dependent-tuple psoa terms ( $m^+ = 1, m^- = 0$ ), the square brackets enclosing the tuple may be omitted as shown for single-dependent-tuple, independent-slot psoa terms of the forms (2.1) and (2.2). Note that in the original PSOA RuleML [15], no prefix was used on any (square-bracketed) tuple, which was assumed to be independent by the semantics. A tuple without brackets was a shortcut for the non-prefixed bracketed tuple in a single-tuple psoa term, hence was also treated as independent. In PSOA RuleML 1.0, the shortcut is employed for the more often used single-dependent-tuple psoa terms. Independent tuples must therefore always be explicitly marked as such in psoa terms. Also note that in the original PSOA RuleML, slots always used the “-” shaft and were interpreted as independent.

Next we demonstrate the revised syntax using examples.

**Example 3.1.** The following is the first example KB for PSOA RuleML 1.0:

```
Document (
  Group (
    _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])                %           Fact 7
    _John#Top(_gender->_male)              %           Fact 8
    _John#Top(_income->29400)              %           Fact 9
  )
)

```

◇

The upper four facts show a taxonomy of four predicates `_TA`, `_Teacher`, `_Student`, and `_Scholar` connected through the “##” infix indicating the binary subclass relation, where `_TA` is a subclass of both `_Teacher` and `_Student`.

Data (i) Facts 1 to 9 constitute data about `_John`. Data (i) Fact 1 represents that under the perspective of being a `_TA`, `_John`’s workload is high. Data (i) Facts 2 to 4 indicate that under the perspective of being a `_Teacher`, `_John` is characterized by (a length-2 tuple for) `_Wed` followed by `_Thu`, is in the `_dep(artmen)t` of `_Physics`, and has a `_salary` of 29400. Data (i) Facts 5 and 6 give `_Student`-dependent information of `_John`. Data (i) Facts 7 to 9 give the date of birth (represented as a length-3 tuple [1995 8 17]),<sup>28</sup> `_gender`, and total `_income` of `_John`, which are independent of, e.g., the `_Teacher`, `_TA`, and `_Student` perspectives.

According to the metamodel of Section 3.2.1, Data (i) Facts 1 to 6 – with only dependent descriptors – are perspectival atoms, while Data (i) Facts 7 to 9 – with only independent descriptors – are perspeneutral atoms.

Descriptors associated with the same OID can be grouped into atoms in different ways. Besides various other groupings, **two complementary methods** of creating atoms from descriptors are (i) *single-descriptor atoms* and (ii)

---

<sup>28</sup>A tuple can be seen as a shortcut for a tuple-valued slot having the Top-predicate-complementing implicit ‘vacuous’ name (label) `prop(erty)`, which in our examples could be specialized to the slot names `_days` – for the weekdays a scholar is present in a department – and `_dob` – for the date-of-birth of a person.

*perspectival-concentrated atoms*. Method (i) creates one atom for each descriptor, thus maximizing the atom count. Method (ii) creates one atom for each predicate that acts as the class of the OID, thus minimizing the atom count. In method (ii), each atom collects all descriptors that are dependent on the predicate of the atom as well as, optionally, some independent descriptors of the OID.

The above Example 3.1 is modeled according to method (i) using only single-descriptor atoms. Particularly, in the Data (i) Facts 7 to 9, the unique root predicate `Top` is employed, hence keeping this representation unique. These `Top`-typed atoms can also be regarded as untyped atoms, as often used in F-logic and RIF.

**Example 3.2.** The second version of RichTA example changes (only) the data part of Example 3.2 according to above-described method (ii) such that there are both single-descriptor and multiple-descriptor atoms.

```
Document (
  Group (
    . . .                               % 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)
  )
)

```

◇

Here, the lower three ground facts represent the data as perspectival-

concentrated psoa atoms. 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 such 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 Section 3.2.1, both of Example 3.2’s last two facts are perspeneutral+perspectival psoa atoms.

Example 3.2’s data part distributes the independent descriptors across the `_Teacher` and `_Student` atoms, in one of several possible ways according to 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`),<sup>29</sup> obtaining the following unique *perspectivity-concentrated form* of the data.

### Example 3.3.

```
Document (
  Group (
    . . .                                % Taxonomy
    _John#_TA(_workload+>_high)          % Data (iii)
    _John#_Teacher(+[_Wed _Thu] _dept+>_Physics _salary+>29400)
    _John#_Student(+[_Mon _Tue _Fri] _dept+>_Math)
```

---

<sup>29</sup>If a non-Top predicate such as `_Teacher` were used, 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.

```

    _John#Top(-[1995 8 17] _gender->_male _income->29400)
  )
)

```

◇

Posing ground queries to the above KBs of ground atoms exemplifies a *prerequisite for psoa-term unification*, which generalizes oidless-positional-term unification (this prerequisite applies also to non-ground atoms in queries and KB clauses): To unify, two psoa terms must “pair up” [42] descriptors of the same dependency kind – either both independent or both dependent – after Top-dependent descriptors have been “reversed to” 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 Example 3.2
> _John#_Student(_gender->_male)
success
> _John#_Student(_gender+>_male)
fail

_John#_Student(... _dept+>_Math ...)    % Fragment of Example 3.2
> _John#_Student(_dept->_Math)
fail
> _John#_Student(_dept+>_Math)
success

```

```

% Tuples
_John#_Student(... -[1995 8 17] ...)      % Fragment of Example 3.2
> _John#_Student(-[1995 8 17])
success
> _John#_Student(+[1995 8 17])
fail

_John#_Student(+[_Mon _Tue _Fri] ...)      % Fragment of Example 3.2
> _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

```

In general, for determining the above `success` outcomes – besides the same-dependency-kind prerequisite – proof procedures need to be applied. 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.

### 3.2.3 Formal Rules and Queries

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

In Examples 3.1–3.3 above, the `_John`-specific `_TA`-dependent fact `_John#_TA(_workload+>_high)` can be replaced by the following more versatile perspectival rule over dependent slots:

```

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 any OID `?o` that is a 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.

Augmenting Example 3.2’s `_Teacher/_Student` descriptors by corresponding dependent quantitative `_coursehours` slots,

```

_John#_Teacher(... _coursehours+>12 ...)
_John#_Student(... _coursehours+>20 ...)

```

and adding the rule (R1) we arrive at the following prototype KB.

#### Example 3.4.

Document (

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

Group (

```

    _Teacher##_Scholar          % Taxonomy
    _Student##_Scholar
    _TA##_Teacher
    _TA##_Student

```



```
> ?who#_TA(_workload+>?level) % (Q+1?)
```

succeed, with bindings `?who = _John` and `?level = _high`.<sup>30</sup>

A perspeneutral rule mapping from an independent tuple to independent slots can be used to test whether the three tuple elements 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 class `_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 subclass fact and `ValidDate`-checking rule<sup>31</sup>:

```
_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)
```

---

<sup>30</sup>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.

<sup>31</sup>The “...” abridges subrule queries for leap years etc. Finite subsets of triples from the infinite virtual date table, including `_ValidDate(1995 8 17)`, could also be materialized as facts.

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.

### 3.3 Revision of EBNF Grammar for Syntax

The original EBNF grammar for PSOA/PS is shown in the following.

#### Rule Language:

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

#### Condition Language:

```
FORMULA ::= 'And' '(' FORMULA* ')' |  
           'Or' '(' FORMULA* ')' |  
           'Exists' Var+ '(' FORMULA ')' |  
           ATOMIC |  
           'External' '(' Atom ')'  
ATOMIC ::= Atom | Equal | Subclass
```

```

Atom ::= PSOA
Equal ::= TERM '=' TERM
Subclass ::= TERM '##' TERM
PSOA ::= TERM '#' TERM '(' TUPLE* (TERM '->' TERM)* ')'
TUPLE ::= '[' TERM* ']'
TERM ::= Const | Var | Expr | 'External' '(' Expr ')'
Expr ::= PSOA
Const ::= '"' UNICODESTRING '^' SYMSPACE | CONSTSHORT
Var ::= '?' UNICODESTRING?
SYMSPACE ::= ANGLEBRACKIRI | CURIE

```

The condition language defines the formulas that can be used in rule premises or queries. The rule language defines rules and KB documents.

This grammar has several shortcomings:

- (1) The grammar is not closed under description. Description replaces a psoa atom with a conjunction (cf. Section 5.6). The current definitions of the `Implies` and `HEAD` productions allow a head formula to be an atom, an existentially quantified atom, or an conjunction of these two. However, if a head formula is an existentially quantified atom, then its description yields an existentially quantified conjunction, which cannot be accepted by the grammar. Also, if an atom is used in a standalone or universal fact, the description yields a top-level or universally quantified conjunction, which is also not accepted by the grammar.
- (2) The grammar is not closed under objectification. The objectification of a universal fact yields a top-level KB formula having the form `Forall ?X1 ... ?Xn (Exists ?Y1 ... ?Ym (...))`. Such formulas cannot be ac-

cepted by the current grammar. This is because universal quantification is matched by the first case of the `RULE` production, where the `CLAUSE` production accepts only an atom or a rule implication but not an existential quantification.

- (3) The grammar does not cover oidless psOA terms. The `TERM` production accepts only oidful psOA terms. Oidless terms are only treated as shortcuts which need to be given OIDs before semantic interpretation. In Section 3.4 we will give semantics to oidless psOA terms directly, so it is necessary to also change the grammar to accept them.
- (4) Besides the above shortcomings, the grammar also needs to be updated to incorporate the new syntax for dependent and independent descriptors explained in Section 3.2.

In order to address these shortcomings, we introduce the following changes to the EBNF grammar.

- For (1) and (2), we change the `CLAUSE`, `Implies`, and `HEAD` productions as follows:

```

CLAUSE ::= Implies | HEAD
Implies ::= HEAD ':-' FORMULA
HEAD ::= ATOMIC | 'Exists' Var+ '(' HEAD ')' | 'And' '(' HEAD* ')'

```

This grammar constructs the `HEAD` production recursively from `ATOMIC` using `Exists` and `And`, so that any formula accepted by the production would be closed under description and objectification. The `Implies` production is defined directly using `HEAD` to match all head formulas. The `CLAUSE` production

replaces `ATOMIC` with `HEAD` so that all head formulas can also be used as top-level formulas. This makes all top-level formulas closed under description and objectification.

- For (3) and (4), we separate the definition of `PSOA` into `PSOAOIDLESS` and `PSOAOIDFUL`. We also separate the definition of `Atom` into `ATOMOIDLESS` and `ATOMOIDFUL`. The non-terminals `ATOMOIDLESS`, `ATOMOIDFUL`, and `Expr` are defined using `PSOAOIDLESS` and `PSOAOIDFUL`. In the last case of the `FORMULA` production, the `PSOA` non-terminal is changed to `Atom` to indicate the meaning of `psoa` terms in external formulas. The `PSOA` non-terminal in the `TERM` production is also updated so that the production can match both oidless expressions and embedded oidful atoms (see Section 5.2) using the `PSOA` non-terminal.

```
PSOA ::= (TERM '#')? TERM '(' TUPLE* (TERM '->' TERM)* ')'
```

```
TERM ::= Const | Var | PSOA | 'External' '(' PSOA ')'
```

```
FORMULA ::= ... | 'External' '(' Atom ')'
```

```
Atom ::= ATOMOIDLESS | ATOMOIDFUL
```

```
ATOMOIDLESS ::= PSOAOIDLESS
```

```
ATOMOIDFUL ::= PSOAOIDFUL
```

```
PSOA ::= PSOAOIDLESS | PSOAOIDFUL
```

```
PSOAOIDLESS ::= TERM '(' (TERM* | tuple*) slot* ')'
```

```
PSOAOIDFUL ::= TERM '# ' PSOAOIDLESS
```

```
TERM ::= Const | Var | ATOMOIDFUL | Expr | 'External' '(' Expr ')'
```

```
Expr ::= PSOAOIDLESS
```

```
TUPLE ::= ('+' | '-')? '[' TERM* ']'
```

```
slot ::= TERM ('+>' | '->') TERM
```

The revision makes explicit the syntactic shortcut of a single tuple without square brackets by adding the `TERM*` choice. The non-terminal `TUPLE` becomes `TUPLEDI` for a dependent or independent tuple, which has an optional dependency mark in front of the brackets. A new non-terminal `SLOTDI` is introduced for a dependent or independent slot, where the slot arrow can be either `+>` or `->`.

The complete updated grammar after applying the above changes is shown as follows:

### Rule Language:

```
Document ::= 'Document' '(' Base? Prefix* Import* Group? ')'
```

```
Base ::= 'Base' '(' ANGLEBRACKIRI ')'
```

```
Prefix ::= 'Prefix' '(' Name ANGLEBRACKIRI ')'
```

```
Import ::= 'Import' '(' ANGLEBRACKIRI PROFILE? ')'
```

```
Group ::= 'Group' '(' (RULE | Group)* ')'
```

```
RULE ::= ('Forall' Var+ '(' CLAUSE ')') | CLAUSE
```

```
CLAUSE ::= Implies | HEAD
```

```
Implies ::= HEAD ':-' FORMULA
```

```
HEAD ::= ATOMIC | 'Exists' Var+ '(' HEAD ')') | 'And' '(' HEAD* ')'
```

```
PROFILE ::= ANGLEBRACKIRI
```

### Condition Language:

```
FORMULA ::= 'And' '(' FORMULA* ')') |
```

```
           'Or' '(' FORMULA* ')') |
```

```
           'Exists' Var+ '(' FORMULA ')') |
```

```
           ATOMIC |
```

```
           'External' '(' Atom ')')
```

```

ATOMIC ::= Atom | Equal | Subclass
Atom ::= ATOMOIDLESS | ATOMOIDFUL
ATOMOIDLESS ::= PSOAOIDLESS
ATOMOIDFUL ::= PSOAOIDFUL
Equal ::= TERM '=' TERM
Subclass ::= TERM '##' TERM
PSOA ::= PSOAOIDLESS | PSOAOIDFUL
PSOAOIDLESS ::= TERM '(' (TERM* | TUPLEDI*) SLOTDI* ')'
PSOAOIDFUL ::= TERM '#' PSOAOIDLESS
TUPLEDI ::= ('+' | '-') '[' TERM* ']'
SLOTDI ::= TERM ('+>' | '->') TERM
TERM ::= Const | Var | ATOMOIDFUL | Expr | 'External' '(' Expr ')'
Expr ::= PSOAOIDLESS
Const ::= '"' UNICODESTRING '"^~' SYMSPACE | CONSTSHORT
Var ::= '?' UNICODESTRING?
SYMSPACE ::= ANGLEBRACKIRI | CURIE

```

An *external-occurrence-restrained* sublanguage of PSOA 1.0 will be implemented in Section 9.1. It can be defined by specializing the productions such that (i) the left-hand side TERMS of the infix operations in the `Equal`, `Subclass`, and `PSOAOIDFUL` productions are non-external terms and (ii) the predicate term in `PSOAOIDLESS` is a simple term, i.e. a special case of a non-external term.

### 3.4 Revision of Model-Theoretic Semantics

The original semantics of PSOA RuleML [15] defined the interpretation of a psOA term as follows, where  $I$  is a generic mapping for interpreting any term:

$$\begin{aligned}
 I(\text{o\#f}([\mathfrak{t}_{1,1} \dots \mathfrak{t}_{1,n_1}] \dots [\mathfrak{t}_{m,1} \dots \mathfrak{t}_{m,n_m}] \text{p}_1 \text{->} \text{v}_1 \dots \text{p}_k \text{->} \text{v}_k)) = \\
 I_{\text{psoa}}(I(\mathfrak{f}))(I(\text{o}), \\
 \{\langle I(\mathfrak{t}_{1,1}), \dots, I(\mathfrak{t}_{1,n_1}) \rangle, \dots, \langle I(\mathfrak{t}_{m,1}), \dots, I(\mathfrak{t}_{1,n_m}) \rangle\}, \quad (3.3) \\
 \{\langle I(\text{p}_1), I(\text{v}_1) \rangle, \dots, \langle I(\text{p}_k), I(\text{v}_k) \rangle\})
 \end{aligned}$$

$I$  first gives a domain element interpretation in  $D$  to each component of the psOA term, including: the OID  $\text{o}$ ; the predicate/function  $\mathfrak{f}$ ; each positional argument  $\mathfrak{t}_{i,j}$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n_i$ ; each slot name  $\text{p}_h$  and filler  $\text{v}_h$ ,  $1 \leq h \leq k$ . Then  $I_{\text{psoa}}$  maps  $I(\mathfrak{f}) \in D$  to a semantic function of the general form

$$D_{\text{ind}} \times \text{SetOfFiniteBags}(D_{\text{ind}}^*) \times \text{SetOfFiniteBags}(D_{\text{ind}} \times D_{\text{ind}}) \rightarrow D$$

which takes the following three arguments and returns an element  $\mathfrak{d} \in D$ :

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

$I_{\text{psoa}}$  can be applied to  $I(\mathfrak{f})$  no matter whether  $\mathfrak{f}$  is a predicate or function

symbol, where  $I_{\text{psoa}}(\mathbf{I}(\mathbf{f}))$  must be a  $D_{\text{ind}}$ -valued semantic function if  $\mathbf{f}$  is a function. Notice that in the above definitions, the mappings  $\mathbf{I}$  and  $I_{\text{psoa}}$  are only defined for oidful psOA terms and their predicates/functions, hence all psOA terms need to be objectified before applying the semantics. This has several shortcomings:

- Expressions should not have OIDs, which will be explained in Section 5.2.
- Atoms are required to have OIDs before they can be semantically evaluated, which creates overhead for each atom whose predicate in the KB clauses is used only as a Prolog-like relation.

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

1. The definition of  $\mathbf{I}$  for **oidful** psOA terms becomes

$$\begin{aligned}
& \mathbf{I} \left( \begin{array}{l} \text{o\#f} (+ [\mathbf{t}_{1,1}^+ \dots \mathbf{t}_{1,n_1}^+] \dots + [\mathbf{t}_{m^+,1}^+ \dots \mathbf{t}_{m^+,n_{m^+}}^+] \\ - [\mathbf{t}_{1,1}^- \dots \mathbf{t}_{1,n_1}^-] \dots - [\mathbf{t}_{m^-,1}^- \dots \mathbf{t}_{m^-,n_{m^-}}^-] \\ \mathbf{p}_1^+ \> \mathbf{v}_1^+ \dots \mathbf{p}_{k^+}^+ \> \mathbf{v}_{k^+}^+ \\ \mathbf{p}_1^- \> \mathbf{v}_1^- \dots \mathbf{p}_{k^-}^- \> \mathbf{v}_{k^-}^- \end{array} \right) = \\
& I_{\text{psoa}}(\mathbf{I}(\mathbf{f}))(\{\mathbf{I}(\text{o})\}, \\
& \quad \{\langle \mathbf{I}(\mathbf{t}_{1,1}^+), \dots, \mathbf{I}(\mathbf{t}_{1,n_1}^+) \rangle, \dots, \langle \mathbf{I}(\mathbf{t}_{m^+,1}^+), \dots, \mathbf{I}(\mathbf{t}_{m^+,n_{m^+}}^+) \rangle\}, \\
& \quad \{\langle \mathbf{I}(\mathbf{t}_{1,1}^-), \dots, \mathbf{I}(\mathbf{t}_{1,n_1}^-) \rangle, \dots, \langle \mathbf{I}(\mathbf{t}_{m^-,1}^-), \dots, \mathbf{I}(\mathbf{t}_{m^-,n_{m^-}}^-) \rangle\}, \quad (3.4) \\
& \quad \{\langle \mathbf{I}(\mathbf{p}_1^+), \mathbf{I}(\mathbf{v}_1^+) \rangle, \dots, \langle \mathbf{I}(\mathbf{p}_{k^+}^+), \mathbf{I}(\mathbf{v}_{k^+}^+) \rangle\} \\
& \quad \{\langle \mathbf{I}(\mathbf{p}_1^-), \mathbf{I}(\mathbf{v}_1^-) \rangle, \dots, \langle \mathbf{I}(\mathbf{p}_{k^-}^-), \mathbf{I}(\mathbf{v}_{k^-}^-) \rangle\}
\end{aligned}$$

Here, the first argument of the semantic function  $I_{\text{psoa}}(I(\mathbf{f}))$  is wrapped into a singleton set  $\{I(\mathbf{o})\}$ , the second and third arguments are respective interpretations of dependent and independent tuples, and the fourth and fifth arguments are respective interpretations of dependent and independent slots. The first-argument wrapping method in (3.4) allows defining  $I$  for **oidless** psOA terms separately, by using the empty set  $\{\}$  as the first argument:

$$\begin{aligned}
& I \left( \begin{array}{l} \mathbf{f} (+ [\mathbf{t}_{1,1}^+ \dots \mathbf{t}_{1,n_1}^+] \dots + [\mathbf{t}_{m^+,1}^+ \dots \mathbf{t}_{m^+,n_{m^+}}^+] \\ - [\mathbf{t}_{1,1}^- \dots \mathbf{t}_{1,n_1}^-] \dots - [\mathbf{t}_{m^-,1}^- \dots \mathbf{t}_{m^-,n_{m^-}}^-] \\ \mathbf{p}_1^+ \mathbf{v}_1^+ \dots \mathbf{p}_{k^+}^+ \mathbf{v}_{k^+}^+ \\ \mathbf{p}_1^- \mathbf{v}_1^- \dots \mathbf{p}_{k^-}^- \mathbf{v}_{k^-}^- \end{array} \right) = \\
& I_{\text{psoa}}(I(\mathbf{f}))(\{\}, \\
& \quad \{\langle I(\mathbf{t}_{1,1}^+), \dots, I(\mathbf{t}_{1,n_1}^+) \rangle, \dots, \langle I(\mathbf{t}_{m^+,1}^+), \dots, I(\mathbf{t}_{m^+,n_{m^+}}^+) \rangle\}, \\
& \quad \{\langle I(\mathbf{t}_{1,1}^-), \dots, I(\mathbf{t}_{1,n_1}^-) \rangle, \dots, \langle I(\mathbf{t}_{m^-,1}^-), \dots, I(\mathbf{t}_{m^-,n_{m^-}}^-) \rangle\}, \\
& \quad \{\langle I(\mathbf{p}_1^+), I(\mathbf{v}_1^+) \rangle, \dots, \langle I(\mathbf{p}_{k^+}^+), I(\mathbf{v}_{k^+}^+) \rangle\} \\
& \quad \{\langle I(\mathbf{p}_1^-), I(\mathbf{v}_1^-) \rangle, \dots, \langle I(\mathbf{p}_{k^-}^-), I(\mathbf{v}_{k^-}^-) \rangle\}
\end{aligned} \tag{3.5}$$

2. The definition of  $I_{\text{psoa}}$  is changed to map  $D$  to semantic functions of the form

$$\begin{aligned}
& \text{SetOfPhiSingletons}(D_{\text{ind}}) \\
& \times \text{SetOfFiniteBags}(D_{\text{ind}}^*) \times \text{SetOfFiniteBags}(D_{\text{ind}}^*) \\
& \times \text{SetOfFiniteBags}(D_{\text{ind}} \times D_{\text{ind}}) \times \text{SetOfFiniteBags}(D_{\text{ind}} \times D_{\text{ind}}) \\
& \hspace{20em} \rightarrow D
\end{aligned}$$

where  $\text{SetOfPhiSingletons}(D_{\text{ind}})$  is defined as  $\{\{\}\} \cup \{\{c\} \mid c \in D_{\text{ind}}\}$ ,

whose elements contains the empty set  $\{\}$  and a singleton set  $\{c\}$  for each  $c \in D_{\text{ind}}$ . With this definition, the semantic function  $I_{\text{psoa}}(I(f))$  can be correctly applied to the arguments in equations (3.4) and (3.5).

3. Define truth evaluation for oidless psoa terms used as atoms as follows.

$$TVal_{\mathcal{I}}(f(\dots)) = I_{\text{truth}}(I(f(\dots)))$$

The truth evaluation for oidful psoa terms is kept unchanged.

$$TVal_{\mathcal{I}}(\text{o}\#f(\dots)) = I_{\text{truth}}(I(\text{o}\#f(\dots)))$$

4. A model of a PSOA KB is required to conform to the *objectification restriction* to capture the logical equivalence between an oidless atom (notice the absence of “o#” in front of “f(…)”) and its existentially objectified form:

$$TVal_{\mathcal{I}}(f(\dots)) = \mathbf{t} \text{ if and only if } TVal_{\mathcal{I}}(\text{Exists } ?0 (?0\#f(\dots))) = \mathbf{t}$$

where  $?0$  is a fresh variable representing the postulated virtual OID of the atom. Notice that the restriction applies only to atoms – after unnesting (cf. Section 5.2) occurring only the top-level – but not to – embedded – expressions, which do not have a truth value.

5. The description restriction is updated to incorporate dependent and independent slots and tuples:

$$TVal_{\mathcal{I}} \left( \begin{array}{l} \text{o\#f} (+ [t_{1,1}^+ \dots t_{1,n_1^+}] \dots + [t_{m^+,1}^+ \dots t_{m^+,n_{m^+}^+}] \\ - [t_{1,1}^- \dots t_{1,n_1^-}] \dots - [t_{m^-,1}^- \dots t_{m^-,n_{m^-}^-}] \\ p_1^+ \> v_1^+ \dots p_{k^+}^+ \> v_{k^+}^+ \\ p_1^- \> v_1^- \dots p_{k^-}^- \> v_{k^-}^- \end{array} \right) = \mathbf{t}$$

if and only if

$$\begin{aligned} & TVal_{\mathcal{I}} (\text{o\#f}) \\ = & TVal_{\mathcal{I}} \left( \text{o\#f} (+ [t_{1,1}^+ \dots t_{1,n_1^+}]) \right) \\ = & \dots \\ = & TVal_{\mathcal{I}} \left( \text{o\#f} (+ [t_{m^+,1}^+ \dots t_{m^+,n_{m^+}^+}] \right) \\ = & TVal_{\mathcal{I}} \left( \text{o\#Top} (- [t_{1,1}^- \dots t_{1,n_1^-}]) \right) \\ = & \dots \\ = & TVal_{\mathcal{I}} \left( \text{o\#Top} (- [t_{m^-,1}^- \dots t_{m^-,n_{m^-}^-}] \right) \\ = & TVal_{\mathcal{I}} (\text{o\#f} (p_1^+ \> v_1^+)) \quad = \dots = \quad TVal_{\mathcal{I}} (\text{o\#f} (p_{k^+}^+ \> v_{k^+}^+)) \\ = & TVal_{\mathcal{I}} (\text{o\#Top} (p_1^- \> v_1^-)) \quad = \dots = \quad TVal_{\mathcal{I}} (\text{o\#Top} (p_{k^-}^- \> v_{k^-}^-)) \\ = & \mathbf{t} \end{aligned}$$

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

# Chapter 4

## Interoperation and Portation

### Architecture

In this chapter we discuss an interoperation and portation architecture based on translators. The architecture is presented here in two parts; each specializes a metaframework to a framework. Section 4.1 will explain the framework of interoperation among multiple rule languages, using PSOA RuleML as the canonical interchange language. Section 4.2 will cover the framework of portation of KBs and queries for the execution of object-relational knowledge.

#### 4.1 PSOA-Centered Interoperation

The goal of rule interoperation is to facilitate the reuse of knowledge expressed in different rule languages as well as software built on top of these languages.

We first define an abstract notion of a knowledge representation language in the following.

**Definition 4.1.** (*Knowledge representation language*) A knowledge representation language  $L = (\Phi_L, Q_L)$  is characterized by the following components:

1. a set of KBs  $\Phi_L$ ;
2. a set of queries  $Q_L$ , where the subset  $BQ_L$  without free variables is called **boolean queries**.

A language has a system of syntaxes and is associated with a system of semantics, where one syntax is distinguished for specifying the semantics. Each semantics defines an entailment relation  $\models_L \subseteq \Phi_L \times BQ_L$ . For each  $\phi \in \Phi_L$  and  $q \in BQ_L$ ,  $\phi$  entails  $q$  is denoted as  $\phi \models_L q$ . Here, the notation  $\models_L$  generically refers to entailments defined using model-theoretic or proof-theoretic semantics, denoted as  $\models_L$  or  $\vdash_L$  respectively.

A language  $L' = (\Phi_{L'}, Q_{L'})$  is called a **sublanguage** of  $L$ , written as  $L' \preceq L$ , if  $\Phi_{L'} \subseteq \Phi_L$  and  $Q_{L'} \subseteq Q_L$ .

An intersection of two sublanguages  $L'_1$  and  $L'_2$  of  $L$ , denoted as  $L'_1 \cap L'_2$ , is defined as  $(\Phi_{L'_1} \cap \Phi_{L'_2}, Q_{L'_1} \cap Q_{L'_2})$ . Similarly, a union of two sublanguages  $L'_1 \cup L'_2$  is defined as  $(\Phi_{L'_1} \cup \Phi_{L'_2}, Q_{L'_1} \cup Q_{L'_2})$ . ◇

As explained in Section 2.4, interoperation between  $n$  knowledge representation (e.g., rule) languages can be achieved through an interoperation metaframework choosing a canonical language  $L_c$  from the  $n$  languages and implementing bidirectional translations between  $L_c$  and any designated language  $L_{d_i}, i = 1, \dots, n - 1$ . The interoperation can be expressed as a visual pattern in Figure 4.1, where each node represents knowledge expressed in one language and each edge represents a translation of knowledge expressed in one language to another.

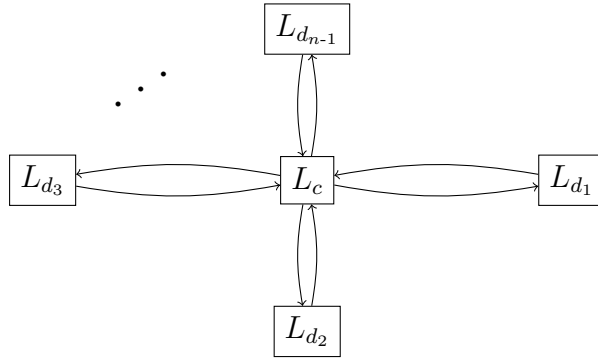


Figure 4.1: Interoperation metaframework.

The metaframework may be specialized, e.g., fixing  $L_c$  to RIF or PSOA, leading to RIF- or PSOA-centered interoperation frameworks. Figure 4.2 shows the PSOA-centered interoperation framework.

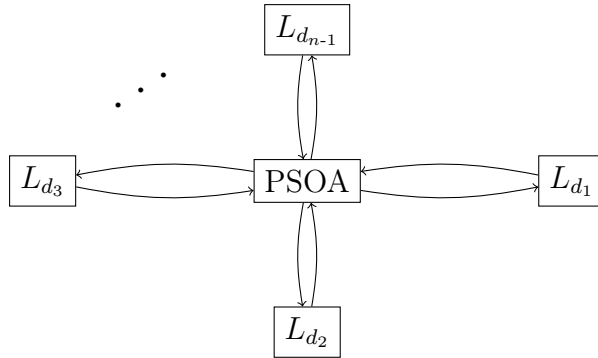


Figure 4.2: PSOA-centered interoperation framework.

Further specializing the framework for  $n = 4$ ,  $L_{d_1} = \text{N3}$ ,  $L_{d_2} = \text{TPTP}$ , and  $L_{d_3} = \text{(Pure) Prolog}$ , we obtain the instantiation in Figure 4.3, which is focused in this dissertation.

In Figure 4.3, the translations that are implemented in our work are expressed using solid lines while others are expressed using dash lines. The translations from PSOA to Prolog and to TPTP will also be needed for portation in the next section while the N3-to-PSOA translation is only needed for

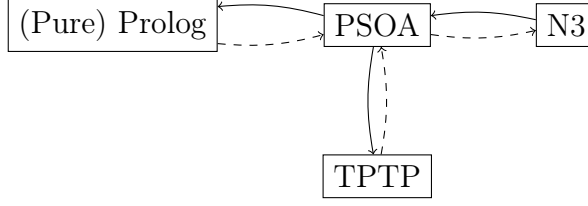


Figure 4.3: An instantiation of PSOA-centered interoperation framework.

interoperation.

Next we give a formal definition of the general concept of a translation between rule languages (i.e., for their KBs and queries).

**Definition 4.2.** (*Translation for KBs and queries*) Given languages  $L_s = (\Phi_{L_s}, Q_{L_s})$  and  $L_t = (\Phi_{L_t}, Q_{L_t})$ , an  $L_s$ -to- $L_t$  **translation**  $\text{tr}_{L_s, L_t}$  is a pair  $(\text{trK}_{L_s, L_t}, \text{trQ}_{L_s, L_t})$  where  $\text{trK}_{L_s, L_t}$  is a mapping  $\Phi_{L_s} \rightarrow \Phi_{L_t}$  for KB translation, and  $\text{trQ}_{L_s, L_t}$  is a mapping  $\Phi_{L_s} \times Q_{L_s} \rightarrow Q_{L_t}$  for query translation. The arguments of the  $\text{trQ}_{L_s, L_t}$  mapping include the KB  $\phi$  to allow query translations to be KB-specific. For convenience, we abbreviate  $\text{trK}_{L_s, L_t}(\phi)$  as  $\text{tr}_{L_s, L_t}(\phi)$  and  $\text{trQ}_{L_s, L_t}(\phi, q)$  as  $\text{tr}_{L_s, L_t}(\phi, q)$  in the rest of the dissertation whenever appropriate.

When  $L_s$  and  $L_t$  are different,  $\text{tr}_{L_s, L_t}$  is an external translation. When  $L_s$  and  $L_t$  are the same language  $L$ , the translation  $\text{tr}_{L, L}$  is an  $L$ -internal translation, also called a **transformation**, and can be abbreviated as  $\text{tr}_L$ .

$\text{trK}_{L_s, L_t}$  and  $\text{trQ}_{L_s, L_t}$  are often defined using a recursive mapping  $\text{tr}'_{L_s, L_t}$  from constructs in  $L_s$  to constructs in  $L_t$  of the same kind. A construct  $\chi_{L_s}$  in  $L_s$  can be a KB, a formula, or a base term. Each construct  $\chi_{L_s}$  is augmented by information about its surrounding context, including:

1.  $\text{KB}(\chi_{L_s})$ , denoting: (1) the surrounding KB, if  $\chi_{L_s}$  is a KB construct; or (2) the KB that the surrounding query is posed to, if  $\chi_{L_s}$  is a query

construct

2. An indication of whether – in case  $\chi_{L_s}$  is a formula or a base term –  $\chi_{L_s}$  occurs: in a fact, in a rule conclusion, in a rule condition, or in a query

Other context information will be explained for specific translations whenever needed in the rest of the dissertation.

Based on  $\text{tr}'_{L_s, L_t}$ , for each KB  $\phi \in \Phi_{L_s}$  and query  $q \in Q_{L_s}$ ,  $\text{trK}_{L_s, L_t}$  and  $\text{trQ}_{L_s, L_t}$  can be defined as

- $\text{trK}_{L_s, L_t}(\phi) = \text{tr}'_{L_s, L_t}(\phi)$
- $\text{trQ}_{L_s, L_t}(\phi, q) = \text{tr}'_{L_s, L_t}(cq)$ , where  $cq$  augments  $q$  with 1.  $\phi$  as the context KB( $cq$ ) and 2. an indicator that  $cq$  occurs in a query.  $\diamond$

A translation can be a sequential *composition* of multiple translations. An elementary translation that is not composed from other translations is called a *translation step*. The composition of translations is defined as follows.

**Definition 4.3.** (*Translation composition*) Given two translations  $\text{tr}_{L_1, L_2} = (\text{trK}_{L_1, L_2}, \text{trQ}_{L_1, L_2})$  and  $\text{tr}_{L_2, L_3} = (\text{trK}_{L_2, L_3}, \text{trQ}_{L_2, L_3})$ , the notation  $\text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2}$  denotes a sequential composition of the two translations, which is a pair  $(\text{trK}_{L_1, L_3}, \text{trQ}_{L_1, L_3})$ . For any KB  $\phi \in \Phi_{L_1}$  and query  $q \in Q_{L_1}$ ,  $\text{trK}_{L_1, L_3}$  is defined as

$$\text{trK}_{L_1, L_3}(\phi) = \text{trK}_{L_2, L_3}(\text{trK}_{L_1, L_2}(\phi))$$

and  $\text{trQ}_{L_1, L_3}$  is defined as

$$\text{trQ}_{L_1, L_3}(\phi, q) = \text{trQ}_{L_2, L_3}(\text{trK}_{L_1, L_2}(\phi), \text{trQ}_{L_1, L_2}(\phi, q))$$

A translation composed from more than two translations  $\text{tr}_{L_1, L_2}, \text{tr}_{L_2, L_3}, \dots, \text{tr}_{L_{n-1}, L_n}$  is defined as

$$\text{tr}_{L_{n-1}, L_n} \circ \dots \circ \text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2} = \text{tr}_{L_{n-1}, L_n} \circ (\dots (\text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2})) \quad \diamond$$

An external  $L_s$ -to- $L_t$  translation can be a composition of one or more  $L_s$ -internal translation steps and one external  $L_s$ -to- $L_t$  translation step. The  $L_s$ -internal translation steps will also be called  $L_s$  transformation steps in the rest of the dissertation. They rewrite formulas having constructs or special semantics in  $L_s$  but not in  $L_t$ . The composition of these steps provides a *normalization* of KBs and queries in  $L_s$ . The last (external)  $L_s$ -to- $L_t$  translation step is called a *conversion*, which transliterates constructs of  $L_s$  into constructs of  $L_t$ . Besides the top-level modularization of translation into transformation followed by transliteration, the division of normalization into smaller steps allows fine-grained modularity. While this highly modular design may be slightly less efficient than a monolithic approach, it makes the translators easier to create, test, maintain, and reuse.

As the centerpiece of the interoperation metaframework shown in Figure 4.1, an  $L_s$ -to- $L_t$  translation where neither  $L_s$  nor  $L_t$  is  $L_c$  can be obtained by composing an  $L_s$ -to- $L_c$  translation and an  $L_c$ -to- $L_t$  translation. As an example, with  $L_c = \text{PSOA}$ , Figure 4.4 shows compositions for N3-to-Prolog and N3-to-TPTP translations. The figure expands the corresponding boxes and solid lines in Figure 4.3. The N3-to-Prolog translation can be composed from N3-to-PSOA and PSOA-to-Prolog (PSOA2Prolog) translations. The latter is composed from a normalization within PSOA targeting LP and a conversion from the normalized PSOA to Prolog. The N3-to-TPTP translation can be

composed similarly.

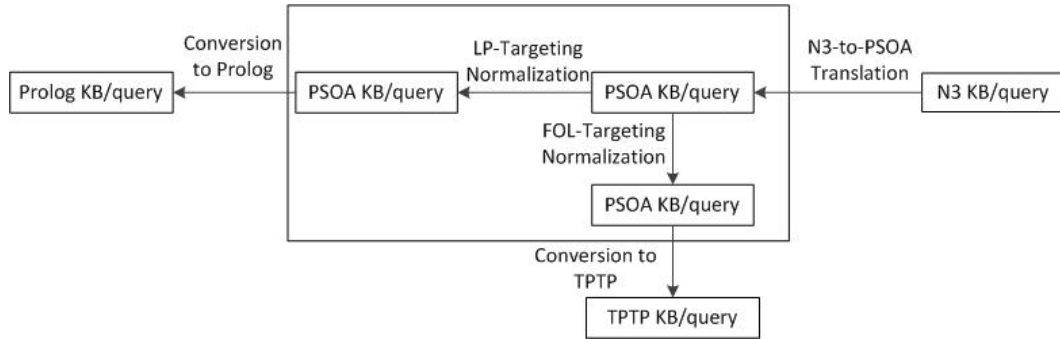


Figure 4.4: Expanded view of translation network.

## 4.2 Translator-Based Portation

Portation of KBs/queries in a rule language  $L_s$  to a reasoning system for a rule language  $L_t$  can be done using the portation metaframework shown in Figure 4.5.

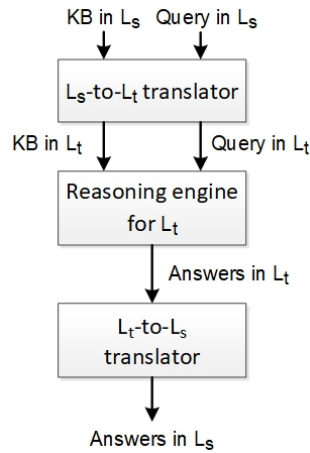


Figure 4.5: Translator-based metaframework for portation.

Given a KB  $\phi_s$  and a query  $q_s$  in  $L_s$ , the metaframework has three sequential phases:

1. Translate  $\phi_s$  and  $q_s$  into a KB  $\phi_t$  and a query  $q_t$  in  $L_t$  using an  $L_s$ -to- $L_t$  translator which implements a translation  $\text{tr}_{L_s, L_t}$ ;
  2. Execute  $q_t$  against  $\phi_t$  in a reasoning system for  $L_t$  (also called a reasoning engine as part of the enclosing system for  $L_s$ ) and get the answers;
  3. Translate the answers in  $L_t$  back to  $L_s$ , using a translator from  $L_t$  to  $L_s$  that acts only on base terms but not necessarily KBs and formulas.
- When  $\text{tr}_{L_s, L_t}$  is built on top of a recursive mapping  $\text{tr}'_{L_s, L_t}$  as explained in Definition 4.2, the back-translation can be seen as a mapping that restricts  $(\text{tr}'_{L_s, L_t})^{-1}$  to base terms and extends it with symbols introduced by the reasoning engine.

The portation metaframework can be notated using the bracketed notation  $L_s\text{TransRun}[L_s2L_t, runtime]$ . Here, the patterns  $L_s\text{TransRun}$  and  $L_s2L_t$  are concatenations of the language variables  $L_s$  and  $L_t$  with the string constants “TransRun” and “2” (in the sense of “-to-”). The first component  $L_s2L_t$  implements the translation from  $L_s$  to  $L_t$  in the first phase. The second component, *runtime*, provides an execution environment for  $L_t$  in the second phase, performing query answering using the output of  $L_s2L_t$ . Since the component for the third phase is dependent on the translator  $L_s2L_t$  and can be implemented as a supplement of  $L_s2L_t$ , it is omitted from the bracketed notation.

Compared to a reasoning system that implements query answering by compiling  $L_s$  to an instruction level and implementing a runtime system for it, a translator-based implementation allows faster prototyping and is easier to maintain. However, it may be harder for a translator-based implementation to incorporate reasoning procedures and optimizations that are not already

implemented by the specific underlying  $L_t$  engine.

Specializing the portation metaframework with  $L_s = \text{PSOA}$  leads to the  $\text{PSOATransRun}[\text{PSOA2}L_t, \text{runtime}]$  framework for providing query answering in PSOA RuleML. In our work, two instantiations of  $\text{PSOATransRun}$  will be discussed and realized:  $\text{PSOATransRun}[\text{PSOA2TPTP}, \text{VampirePrime}]$ , where  $L_t$  is TPTP, and  $\text{PSOATransRun}[\text{PSOA2Prolog}, \text{XSBProlog}]$ , where  $L_t$  is Prolog. The VampirePrime and XSB Prolog engines are used as ‘black boxes’ in our work while the translators  $\text{PSOA2TPTP}$  and  $\text{PSOA2Prolog}$  are our focus and will be explained in Chapters 7 and 8, respectively.

The portation metaframework can also be specialized to other languages. For example, fixing  $L_s = \text{RIF}$  leads to the  $\text{RIFTransRun}[\text{RIF2}L_t, \text{runtime}]$  framework, which can be further specialized, e.g. for  $L_t = \text{Prolog}$ .

The correctness of such portation relates to the *semantics-preservation* property of the translation. Intuitively, a translation is semantics-preserving if entailments are maintained before and after the translation. Sometimes, a translation may be not semantics-preserving with respect to all inputs but can be changed to a semantics-preserving translation by restricting the inputs to a (small enough) sublanguage. Restricting the inputs can also allow a partial translation (interoperation) to become total translation (interoperation). The property can be defined via the notions of *soundness* and *completeness* (analogous to, respectively, perfect precision and perfect recall in information retrieval) as shown in the following.<sup>32</sup>

**Definition 4.4.** (*Semantics-preserving translation*) *Given languages  $L_s = (\Phi_{L_s}, Q_{L_s})$  and  $L_t = (\Phi_{L_t}, Q_{L_t})$ , a translation  $\text{tr}_{L_s, L_t}$  **preserves sound-***

---

<sup>32</sup>Semantics preservation is also used in the RIF standard to define conformant RIF-BLD consumers and producers: [http://www.w3.org/TR/rif-bld/#Conformance\\_Clauses](http://www.w3.org/TR/rif-bld/#Conformance_Clauses)

*ness* or is **sound** with respect to a sublanguage  $L'_s$  of  $L_s$  and entailments  $\Vdash_{L_s}$  and  $\Vdash_{L_t}$ , if for any  $\phi \in \Phi_{L'_s}, q \in BQ_{L'_s}$  such that  $\text{tr}_{L_s, L_t}(\phi) \Vdash_{L_t} \text{tr}_{L_s, L_t}(\phi, q)$ ,  $\phi \Vdash_{L_s} q$  holds. Also,  $\text{tr}_{L_s, L_t}$  **preserves completeness** or is **complete** with respect to  $L'_s$ ,  $\Vdash_{L_s}$  and  $\Vdash_{L_t}$  if for any  $\phi \in \Phi_{L'_s}, q \in BQ_{L'_s}$  such that  $\phi \Vdash_{L_s} q$ ,  $\text{tr}_{L_s, L_t}(\phi) \Vdash_{L_t} \text{tr}_{L_s, L_t}(\phi, q)$  holds.  $\text{tr}_{L_s, L_t}$  is said to **preserve semantics** or to be **semantics-preserving** with respect to  $L'_s$ ,  $\Vdash_{L_s}$  and  $\Vdash_{L_t}$  if it is both sound and complete with respect to  $L'_s$ ,  $\Vdash_{L_s}$  and  $\Vdash_{L_t}$ . When  $L_s = L_t = L$  and  $\Vdash_{L_s}$  and  $\Vdash_{L_t}$  are the same entailment in  $L$ , we say  $\text{tr}_{L, L}$  is semantics-preserving with respect to  $L$  and  $\Vdash_L$ .  $\diamond$

The semantics-preservation property can be visualized through the commutative diagram in Figure 4.6.

$$\begin{array}{ccc}
 & \text{tr}_{L_s, L_t} & \\
 BQ_{L_s} & \overset{\text{tr}_{L_s, L_t}}{\dashrightarrow} & BQ_{L_t} \\
 \uparrow \Vdash_{L_s} & & \uparrow \Vdash_{L_t} \\
 \Phi_{L_s} & \overset{\text{tr}_{L_s, L_t}}{\dashrightarrow} & \Phi_{L_t}
 \end{array}$$

Figure 4.6: Semantics-preserving translation from  $L_s$  to  $L_t$ .

The following corollary follows trivially from Definition 4.4.

**Corollary 4.1.** *If  $L''_s \preceq L'_s$  and a translation  $\text{tr}_{L_s, L_t}$  is semantics-preserving with respect to  $L'_s$ ,  $\Vdash_{L_s}$ , and  $\Vdash_{L_t}$ , then  $\text{tr}_{L_s, L_t}$  is semantics-preserving with respect to  $L''_s$ ,  $\Vdash_{L_s}$ , and  $\Vdash_{L_t}$ .*

Theorem 4.1 gives a sufficient condition for the semantics preservation of a translation composition.

**Theorem 4.1.** *Let  $L_1$ ,  $L_2$ , and  $L_3$  be languages,  $\models_{L_1}$ ,  $\models_{L_2}$ , and  $\models_{L_3}$  be entailments in these languages,  $L'_1 = (\Phi_{L'_1}, Q_{L'_1})$  and  $L'_2 = (\Phi_{L'_2}, Q_{L'_2})$  be respective sublanguages of  $L_1$  and  $L_2$ ,  $\text{tr}_{L_1, L_2}$  be a translation from  $L_1$  to  $L_2$ , and  $\text{tr}_{L_2, L_3}$  be a translation from  $L_2$  to  $L_3$ .*

*The composition  $\text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2}$  is semantics-preserving with respect to  $L'_1$ ,  $\models_{L_1}$ ,  $\models_{L_3}$  if the following holds: (1)  $\text{tr}_{L_1, L_2}$  is semantics-preserving with respect to  $L'_1$ ,  $\models_{L_1}$ , and  $\models_{L_2}$ ; (2)  $\text{tr}_{L_2, L_3}$  is semantics-preserving with respect to  $L'_2$ ,  $\models_{L_2}$ , and  $\models_{L_3}$ ; (3) for any  $\phi \in \Phi_{L'_1}$  and  $q \in BQ_{L'_1}$ ,  $\text{tr}_{L_1, L_2}(\phi) \in \Phi_{L'_2}$  and  $\text{tr}_{L_1, L_2}(\phi, q) \in BQ_{L'_2}$  respectively.*

*Proof.* Given  $\phi \in \Phi_{L'_1}$  and  $q \in BQ_{L'_1}$ , since  $\text{tr}_{L_1, L_2}$  is semantics-preserving with respect to  $L'_1$ ,  $\models_{L_1}$ , and  $\models_{L_2}$ ,  $\phi \models_{L_1} q$  iff  $\text{tr}_{L_1, L_2}(\phi) \models_{L_2} \text{tr}_{L_1, L_2}(\phi, q)$  holds. Since  $\text{tr}_{L_1, L_2}(\phi) \in \Phi_{L'_2}$ ,  $\text{tr}_{L_1, L_2}(\phi, q) \in BQ_{L'_2}$ , and  $\text{tr}_{L_2, L_3}$  is semantics-preserving with respect to  $L'_2$ ,  $\models_{L_2}$ , and  $\models_{L_3}$ ,  $\text{tr}_{L_1, L_2}(\phi) \models_{L_2} \text{tr}_{L_1, L_2}(\phi, q)$  iff  $\text{tr}_{L_2, L_3}(\text{tr}_{L_1, L_2}(\phi)) \models_{L_3} \text{tr}_{L_2, L_3}(\text{tr}_{L_1, L_2}(\phi), \text{tr}_{L_1, L_2}(\phi, q))$ .

By Definition 4.3,  $\phi \models_{L_1} q$  iff  $\text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2}(\phi) \models_{L_3} \text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2}(\phi, q)$  holds. Hence  $\text{tr}_{L_2, L_3} \circ \text{tr}_{L_1, L_2}$  is semantics-preserving with respect to  $L'_1$ .  $\square$

Next we discuss the connection between the correctness of the portation metaframework for query answering in  $L_s$  and the semantics-preservation property of  $\text{tr}_{L_s, L_t}$ . We first define the notion of certain answers, commonly used in knowledge representation [71].<sup>33</sup>

**Definition 4.5.** *(Certain answers) Given a KB  $\phi_L$  and a query  $q_L$  in language  $L$ , a certain answer ans with respect to  $\phi_L$  and  $q_L$  is a substitution, denoted as  $\{X_1 = t_1, \dots, X_n = t_n\}$ , where  $X_1, \dots, X_n$  are free variables in*

<sup>33</sup>This sense of '(un)certain answers' in the literature (e.g., [71]) is unrelated to "(un)certain values/factors", as used in probabilistic/fuzzy/... logics.

$q_L$  and  $t_1, \dots, t_n$  are base terms constructed from constants in  $\phi_L$ , such that  $\phi_L \Vdash_L q_L(ans)$ . The notation  $q_L(ans)$  denotes the formula obtained by applying the substitution  $ans$  to  $q_L$ . The set of all certain answers with respect to  $\phi_L$  and  $q_L$  is written as  $CAns(\phi_L, q_L)$ .  $\diamond$

In brief, certain answers are the answers that use only constants from the original KB. Note that although an actual reasoning system for query-answering may not return only certain answers – e.g., in our PSOATransRun system, ‘uncertain’ answers may use generated constants for existential variables – the discussion of the connection to the semantics-preservation property here is limited to certain answers since ‘uncertain’ answers cannot be directly defined via a Tarski-style model-theoretic semantics of a language and they are often specific to each implementation of a language.

**Theorem 4.2.** *Given a KB  $\phi_{L_s}$  in  $L_s$ , a query  $q_{L_s}$  in  $L_s$ , and a translation  $\text{tr}_{L_s, L_t}$  defined via a recursive mapping  $\text{tr}'_{L_s, L_t}$  according to Definition 4.2,  $ans \in CAns(\phi_{L_s}, q_{L_s})$  iff  $\text{tr}'_{L_s, L_t}(ans) \in CAns(\text{tr}_{L_s, L_t}(\phi_{L_s}), \text{tr}_{L_s, L_t}(q_{L_s}))$  holds if  $\text{tr}_{L_s, L_t}$  satisfies the following: (1)  $\text{tr}_{L_s, L_t}$  is semantics-preserving; (2) for any substitution  $st$ ,  $\text{tr}_{L_s, L_t}(\phi_{L_s}, q_{L_s}(st)) = \text{tr}_{L_s, L_t}(\phi_{L_s}, q_{L_s})(\text{tr}'_{L_s, L_t}(st))$ . Here, the mapping  $\text{tr}'_{L_s, L_t}$  of Definition 4.2 is extended for substitutions, where  $\text{tr}'_{L_s, L_t}(st)$  is defined as  $\{\text{tr}'_{L_s, L_t}(X_i) = \text{tr}'_{L_s, L_t}(t_i) \mid X_i = t_i \in st\}$ .*

*Proof.*

$$\begin{aligned}
& ans \in CAns(\phi_{L_s}, q_{L_s}) \\
\text{iff } & \phi_{L_s} \Vdash_{q_{L_s}}(ans) \\
\text{iff } & \mathbf{tr}_{L_s, L_t}(\phi_{L_s}) \Vdash_{\mathbf{tr}_{L_s, L_t}(\phi_{L_s}, q_{L_s}(ans))} \text{ (by semantics preservation of } \mathbf{tr}_{L_s, L_t} \text{)} \\
\text{iff } & \mathbf{tr}_{L_s, L_t}(\phi_{L_s}) \Vdash_{\mathbf{tr}_{L_s, L_t}(\phi_{L_s}, q_{L_s})(\mathbf{tr}'_{L_s, L_t}(ans))} \\
\text{iff } & \mathbf{tr}'_{L_s, L_t}(ans) \in CAns(\mathbf{tr}_{L_s, L_t}(\phi_{L_s}), \mathbf{tr}_{L_s, L_t}(\phi_{L_s}, q_{L_s})) \quad \square
\end{aligned}$$

According to the theorem, the combined reasoning system in Figure 4.5 is sound and complete with respect to certain answers as long as the translation  $\mathbf{tr}_{L_s, L_t}$  in phase 1 is semantics-preserving and preserves free variables (cf. requirement (2) above), the underlying engine used in phase 2 is sound and complete, and the backward translation in phase 3 is an inverse of  $\mathbf{tr}_{L_s, L_t}$  specialized to base terms. In later chapters, we will expand on the semantics preservation of the translations rather than the soundness and completeness of the underlying engines.

# Chapter 5

## PSOA Transformation Modules and their Composition

In this chapter, we discuss PSOA transformation steps that can be reused across different translators from PSOA RuleML to other rule languages. The steps include unnesting, objectification, Skolemization, subclass transformation, description, flattening external expressions, and splitting rules with conjunctive conclusions. The KB merging operation will also be explained. Section 5.1 explores semantics-preserving PSOA transformations and sufficient conditions to prove semantics preservation. Each of the Sections 5.2 to 5.8 defines a transformation step and discusses the PSOA sublanguage within which the transformation is semantics-preserving. Most of the steps refine those of our earlier papers [73] and [74]. Section 5.9 discusses the KB merging operation needed for processing KBs that use `Import` statements. Section 5.10 discusses sequential composition of PSOA transformations.

## 5.1 PSOA Transformations and their Semantics

### Preservation

In this section, we explore the construction of PSOA transformations and the proof of their semantics-preservation property.

A PSOA transformation  $\text{tr}_{\text{PSOA}}$  is defined based on Definition 4.2, where  $L_s = L_t = \text{PSOA}$ . As explained in Definition 4.2,  $\text{tr}_{\text{PSOA}}$  can be defined on top of a recursive mapping  $\text{tr}'_{\text{PSOA}}$ , which needs to be defined over every PSOA construct. In order to simplify future definitions of  $\text{tr}_{\text{PSOA}}$ , we define a schema for constructing  $\text{tr}'_{\text{PSOA}}$  from a preliminary version  $\text{tr}^*_{\text{PSOA}}$  defined only for certain constructs that are the focus of the transformation.

**Definition 5.1.** *Let  $\text{tr}^*_{\text{PSOA}}$  be a transformation defined for certain PSOA constructs. We can define  $\text{tr}'_{\text{PSOA}}$  as follows:*

- (1) *If  $\text{tr}^*_{\text{PSOA}}(\tau)$  is defined,  $\text{tr}'_{\text{PSOA}}(\tau) = \text{tr}^*_{\text{PSOA}}(\tau)$ ;*
- (2) *if  $\text{tr}^*_{\text{PSOA}}(\tau)$  is not defined and  $\tau$  is a simple term,  $\text{tr}'_{\text{PSOA}}(\tau) = \tau$  is just an identity mapping;*
- (3) *otherwise, i.e.  $\text{tr}^*_{\text{PSOA}}(\tau)$  is not defined and  $\tau$  is not a simple term,  $\text{tr}'_{\text{PSOA}}(\tau)$  is obtained by recursively applying  $\text{tr}'_{\text{PSOA}}$  to each subformula or subterm of  $\tau$ , while keeping the surrounding formula or term structure unchanged.*

$\text{tr}_{\text{PSOA}}$  is defined on top of  $\text{tr}'_{\text{PSOA}}$  according to Definition 4.2. ◇

Note that the construct obtained by  $\text{tr}'_{\text{PSOA}}(\tau)$  could also be obtained by replacing every subconstruct  $\tau_1$  in  $\tau$  with  $\text{tr}^*_{\text{PSOA}}(\tau_1)$  if  $\text{tr}^*_{\text{PSOA}}(\tau_1)$  is defined.

In the rest of the chapter, some transformations are introduced to realize restrictions of the PSOA semantics by transforming KBs and queries so that entailments can be established under a *relaxed PSOA semantics* where one or more semantic restrictions for models are excluded from the original PSOA semantics. In this chapter entailments under the original PSOA semantics will be denoted by  $\models$  for convenience. The notation  $\models_{-r_1 r_2 \dots r_m}$  denotes an entailment under a relaxed PSOA semantics in which the semantic restrictions  $r_1, \dots, r_m$  are excluded (denoted by a preceding ‘-’ sign which applies to each  $r_i$ ). We use  $o$  for the objectification restriction,  $d$  for the description restriction, and  $s$  for the subclass restrictions. For example,  $\models_{-od}$  represents entailment under a relaxed PSOA semantics in which the objectification and description restrictions are excluded. The notation  $S_{\models}$  denotes the set of entailments under either the original PSOA semantics ( $\models$ ) or a relaxed PSOA semantics ( $\models_{-r_1 \dots r_m}$ ). Given  $\models^{\bullet} \in S_{\models}$ ,  $\models^{\bullet}_{-r}$  denotes the semantics where the restriction  $r$  is excluded from  $\models^{\bullet}$ . In the following, we define entailments under relaxed PSOA semantics.

**Definition 5.2.** (*Entailment under relaxed PSOA semantics*) *An interpretation  $\mathcal{I}$  is a model of  $\phi$  under a relaxed PSOA semantics excluding restrictions  $r_1, \dots, r_m$ , written as  $\mathcal{I} \models_{-r_1 \dots r_m} \phi$ , iff  $TVal_{\mathcal{I}}(\phi) = \mathbf{t}$  and  $TVal_{\mathcal{I}}$  conforms to all semantic restrictions except  $r_1, \dots, r_m$ , which are not guaranteed by  $TVal_{\mathcal{I}}$ . A KB  $\phi$  entails a boolean query  $q$  under the relaxed PSOA semantics, written as  $\phi \models_{-r_1 \dots r_m} q$ , iff for every  $\mathcal{I}$  such that  $\mathcal{I} \models_{-r_1 \dots r_m} \phi$ ,  $\mathcal{I} \models_{-r_1 \dots r_m} q$  holds.  $\diamond$*

The semantics preservation for PSOA transformations follows Definition 4.4, where  $L_s = L_t = \text{PSOA}$  and  $\models_{L_s}$  and  $\models_{L_t}$  are entailments from  $S_{\models}$ . For the discussion of translations and semantics preservation, in the rest of the disser-

tation we focus on a PSOA sublanguage named *PSOA Kernel*, defined in the following, and its sublanguages.

**Definition 5.3.** (*PSOA Kernel*) *The PSOA sublanguage  $\mathcal{P}_K = (\Phi_K, Q_K)$  is defined as the subset of PSOA RuleML where  $\Phi_K$  and  $Q_K$  satisfy the following:*

- *No equalities or external terms occur in facts or rule conclusions;*
- *Expressions have the form of a constant function symbol applied to a single dependent tuple.*

*The notation  $BQ_K$  denotes the subset of all boolean queries in  $Q_K$ . ◇*

In order to prove a transformation  $\text{tr}_{\text{PSOA}}$  is semantics-preserving with respect to  $L, \models^\bullet, \models^{\bullet\bullet}$ , where  $L \preceq \mathcal{P}_K$  and  $\models^\bullet, \models^{\bullet\bullet} \in S_{\models}$ , one needs to show that for every KB  $\phi$  and boolean query  $q$  in  $L$ ,  $\phi \models^\bullet q$  iff  $\text{tr}_{\text{PSOA}}(\phi) \models^{\bullet\bullet} \text{tr}_{\text{PSOA}}(\phi, q)$  as required by Definition 4.4. One can first prove  $\phi \models^\bullet q$  iff  $\text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$  and then, if  $\models^\bullet$  and  $\models^{\bullet\bullet}$  are not the same, prove  $\text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$  iff  $\text{tr}_{\text{PSOA}}(\phi) \models^{\bullet\bullet} \text{tr}_{\text{PSOA}}(\phi, q)$ . The next theorem gives a sufficient condition for the first step, where we write  $\text{Var}(\tau)$  for the set of all free variables in a construct  $\tau$ ,  $\mathbf{M}(\text{Var}(\tau), \mathbf{D})$  for the set of all mappings from variables in  $\text{Var}(\tau)$  to domain elements in  $\mathbf{D}$ , and  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V)$  for an interpretation that coincides with  $\mathbf{I}_V$  on all variables it interprets, and with  $\mathcal{I}$  on everything else.

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

**Theorem 5.1.** *Given a KB  $\phi \in \Phi_K$ , a boolean query  $q \in BQ_K$ , an entailment  $\models^\bullet \in S_{\models}$ , and a transformation  $\text{tr}_{\text{PSOA}}$  constructed from  $\text{tr}_{\text{PSOA}}^*$  via  $\text{tr}'_{\text{PSOA}}$*

according to Definition 5.1,

$$\phi \models^\bullet q \quad \text{iff} \quad \text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$$

holds if  $\text{tr}'_{\text{PSOA}}$  has the following properties:

*Property (1) (Soundness)* If  $\phi \not\models^\bullet q$ , there exists a counter-model  $\mathcal{I}$  for  $\phi \models^\bullet q$  and a semantic structure  $\mathcal{I}'$  s.t.  $\mathcal{I}'.\mathbf{D} = \mathcal{I}.\mathbf{D}$  and for every  $\tau$  being an atomic formula or a formula changed by  $\text{tr}^*_{\text{PSOA}}$  in  $\phi$  and  $q$ , and every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}.\mathbf{D})$ ,

*Property (1.a)* if  $\tau$  is positive and  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$ , then  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$ ;

*Property (1.b)* if  $\tau$  is negative and  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$ , then  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$ .

*Property (2) (Completeness)* If  $\text{tr}_{\text{PSOA}}(\phi) \not\models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$ , there exists a counter-model  $\mathcal{I}'$  for  $\text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$  and a semantic structure  $\mathcal{I}$  s.t.  $\mathcal{I}.\mathbf{D} = \mathcal{I}'.\mathbf{D}$  and for every  $\tau$  being an atomic formula and every formula changed by  $\text{tr}^*_{\text{PSOA}}$  in  $\phi$  and  $q$ , and every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}'.\mathbf{D})$ ,

*Property (2.a)* if  $\tau$  is positive and  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$ , then  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$ ;

*Property (2.b)* if  $\tau$  is negative and  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$ , then  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$ .

*Proof.* Proving  $\phi \models^\bullet q$  iff  $\text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$  is equivalent to proving  $\phi \not\models^\bullet q$  iff  $\text{tr}'_{\text{PSOA}}(\phi) \not\models^\bullet \text{tr}'_{\text{PSOA}}(cq)$ , where  $cq$  is the context-aware version of  $q$  as explained in Definition 4.2. We start with the ‘if’ part. If  $\text{tr}'_{\text{PSOA}}(\phi) \not\models^\bullet \text{tr}'_{\text{PSOA}}(cq)$ , by Property (2) there exists a counter-model  $\mathcal{I}'$  for  $\text{tr}'_{\text{PSOA}}(\phi) \models^\bullet \text{tr}'_{\text{PSOA}}(cq)$  and a semantic structure  $\mathcal{I}$  having Properties (2.a) and (2.b).

We first prove by induction that

(i) for every positive formula  $\tau$  and  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}'.D)$ , if

$$TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t} \text{ then } TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t}.$$

(ii) for every negative formula  $\tau$  and  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}'.D)$ , if

$$TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t} \text{ then } TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}.$$

Since  $\mathcal{I}'$  is a model,  $\mathcal{I}'$  conforms to all semantic restrictions required by  $\models^\bullet$ .

So  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V)$  also conforms to these restrictions. Thus if for any formula  $\tau'$ ,

$TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\tau') = \mathbf{t}$  holds, then  $\mathcal{I}' \models^\bullet \tau'$  also holds. Here we give the proof of

(i) using Property (2.a), and the proof of (ii) using (2.b) can be done similarly.

- Base case:

If  $\tau$  is an atomic formula or a formula changed by  $\text{tr}^*_{\text{PSOA}}$ , then it has Property

(2.a). If  $TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ , then  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$ . By (2.a)

we have  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$ , so  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t}$ .

- Inductive step:

If  $\tau$  is neither an atomic formula nor a formula changed by  $\text{tr}^*_{\text{PSOA}}$ , then

$\text{tr}'_{\text{PSOA}}(\tau)$  is defined by Case 3 of Definition 5.1. There are three cases:

- $\tau = \text{And}(\tau_1 \dots \tau_n)$

In this case  $\text{tr}'_{\text{PSOA}}(\tau) = \text{And}(\text{tr}'_{\text{PSOA}}(\tau_1) \dots \text{tr}'_{\text{PSOA}}(\tau_n))$ .

If  $TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ , then  $TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau_i)) = \mathbf{t}$  holds for

$i = 1, \dots, n$ . By induction hypothesis,  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau_i) = \mathbf{t}$  holds for each

$i$ . Hence  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t}$  holds.

- $\tau = \text{Or}(\tau_1 \dots \tau_n)$

In this case  $\text{tr}'_{\text{PSOA}}(\tau) = \text{Or}(\text{tr}'_{\text{PSOA}}(\tau_1) \dots \text{tr}'_{\text{PSOA}}(\tau_n))$ .

If  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}})}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ , then  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}})}(\text{tr}'_{\text{PSOA}}(\tau_i)) = \mathbf{t}$  holds for some  $i$  in  $1 \dots n$ . By induction hypothesis,  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_i) = \mathbf{t}$ . Hence  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau) = \mathbf{t}$  holds.

–  $\tau = \text{Exists } ?X_1 \dots ?X_n (\tau_1)$

In this case  $\text{tr}'_{\text{PSOA}}(\tau) = \text{Exists } ?X_1 \dots ?X_n (\text{tr}'_{\text{PSOA}}(\tau_1))$ .

If  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}})}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ , then there exists  $\mathcal{I}_{\mathcal{V}^*} \in \mathcal{M}(\{?X_1, \dots, ?X_n\}, \mathcal{I}.D)$  such that  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}} \cup \mathcal{I}_{\mathcal{V}^*})}(\tau_1) = \mathbf{t}$ . By induction hypothesis,  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}} \cup \mathcal{I}_{\mathcal{V}^*})}(\tau_1) = \mathbf{t}$ . So  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau) = \mathbf{t}$  holds.

Built on top of (i) and (ii), we next prove for each top-level formula  $\tau$  in  $\phi$ ,  $TVal_{\mathcal{I}}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ .

- If  $\tau$  is a ground fact,  $\tau$  is a positive formula so  $TVal_{\mathcal{I}}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ .
- If  $\tau$  is a rule of the form  $\text{Forall } ?X_1 \dots ?X_n (\tau_1 :- \tau_2)$ ,  $TVal_{\mathcal{I}'}(\tau) = \mathbf{t}$  means for every  $\mathcal{I}_{\mathcal{V}^*} \in \mathcal{M}(\{?X_1, \dots, ?X_n\}, \mathcal{I}.D)$ ,  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}^*})}(\tau_1) = \mathbf{t}$  or  $TVal_{\mathcal{V}(\mathcal{I}', \mathcal{I}_{\mathcal{V}^*})}(\tau_2) = \mathbf{f}$  holds. Since  $\tau_1$  is positive and  $\tau_2$  is negative, either  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}^*})}(\tau_1) = \mathbf{t}$  or  $TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}^*})}(\tau_2) = \mathbf{f}$  holds for every  $\mathcal{I}_{\mathcal{V}^*}$ . So  $TVal_{\mathcal{I}}(\tau) = \mathbf{t}$  holds.
- If  $\tau$  is a universal fact, it can be seen as a rule with an empty condition, so that the above proof for general rules still apply.

Hence  $\mathcal{I} \models^{\bullet} \phi$  holds. Also, since  $\mathcal{I}' \not\models^{\bullet} \text{tr}'_{\text{PSOA}}(cq)$  and  $\mathcal{I}'$  conforms to the semantic restrictions,  $TVal_{\mathcal{I}'}(cq) = \mathbf{f}$ . Since  $cq$  is a negative formula, by the above conclusion (ii) we have  $TVal_{\mathcal{I}}(cq) = \mathbf{f}$ . Thus  $\mathcal{I} \not\models^{\bullet} cq$  and  $\mathcal{I} \not\models^{\bullet} q$  hold. So  $\mathcal{I}$  is a counter-model for  $\phi \models^{\bullet} q$ , and  $\phi \not\models^{\bullet} q$  holds.

The ‘only if’ part can be proved similarly by showing  $\mathcal{I}'$  is a counter-model for  $\text{tr}'_{\text{PSOA}}(\phi) \models^{\bullet} \text{tr}'_{\text{PSOA}}(cq)$ . □

**Corollary 5.1.** *Given a KB  $\phi$ , a boolean query  $q$ , an entailment  $\models^\bullet \in S_{\models}$ , and a transformation  $\text{tr}_{\text{PSOA}}$  constructed from  $\text{tr}_{\text{PSOA}}^*$  via  $\text{tr}'_{\text{PSOA}}$ ,*

$$\phi \models^\bullet q \quad \text{iff} \quad \text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$$

*holds if for every formula  $\tau$  changed by  $\text{tr}_{\text{PSOA}}^*$  in  $\phi$  and  $q$ , and every semantic structure  $\mathcal{I}^*$  that conforms to the semantic restrictions required by  $\models^\bullet$ ,  $TVal_{\mathcal{I}^*}(\tau) = \mathbf{t}$  iff  $TVal_{\mathcal{I}^*}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$  holds.*

*Proof.* We will show that  $\text{tr}_{\text{PSOA}}$  has Properties (1) and (2) in Theorem 5.1. For Property (1), if  $\mathcal{I}$  is a counter-model for  $\phi \models q$ , then we can choose  $\mathcal{I}' = \mathcal{I}$ . For every  $\tau$  being atomic formula or a formula changed by  $\text{tr}_{\text{PSOA}}^*$  in  $\phi$  and  $q$ ,

- if  $\tau$  is an atomic formula that is not changed by  $\text{tr}_{\text{PSOA}}^*$ , then  $\text{tr}'_{\text{PSOA}}(\tau) = \tau$  according to Definition 5.1, so (1.a) and (1.b) are fulfilled trivially.
- otherwise  $\tau$  is changed by  $\text{tr}_{\text{PSOA}}^*$ . Since  $\mathcal{I}$  is a model,  $\mathcal{I}$  conforms to all semantic restrictions. For every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}.D)$ ,  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V)$  also conforms to all semantic restrictions. Based on the assumption where  $\mathcal{I}^*$  becomes  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V)$ ,  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t}$  iff  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\text{tr}'_{\text{PSOA}}(\tau)) = \mathbf{t}$ . Thus  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$  iff  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \text{tr}'_{\text{PSOA}}(\tau)$  so that (1.a) and (1.b) are also fulfilled.

Thus  $\text{tr}_{\text{PSOA}}$  has Property (1).

That  $\text{tr}_{\text{PSOA}}$  has Property (2) can be proved similarly by choosing  $\mathcal{I} = \mathcal{I}'$ . So the corollary holds according to Theorem 5.1.  $\square$

The following theorem provides a sufficient condition for proving  $\text{tr}_{\text{PSOA}}(\phi) \models^\bullet \text{tr}_{\text{PSOA}}(\phi, q)$  iff  $\text{tr}_{\text{PSOA}}(\phi) \models^{\bullet\bullet} \text{tr}_{\text{PSOA}}(\phi, q)$ .

**Theorem 5.2.** *Given a KB  $\phi'$ , a boolean query  $q'$ , and  $\models^\bullet, \models^{\bullet\bullet} \in S_{\models}$  where  $\models^{\bullet\bullet}$  has fewer semantic restrictions than  $\models^\bullet$ ,*

$$\phi' \models^\bullet q' \quad \text{iff} \quad \phi' \models^{\bullet\bullet} q'$$

*holds if for each counter-model  $\mathcal{I}''$  for  $\phi' \models^{\bullet\bullet} q'$  there exists a counter-model  $\mathcal{I}'$  for  $\phi' \models^\bullet q'$ .*

*Proof.* We first prove the ‘if’ part. For every  $\mathcal{I}'$  s.t.  $\mathcal{I}' \models^\bullet \phi'$ ,  $\mathcal{I}' \models^{\bullet\bullet} \phi'$  holds since  $\models^{\bullet\bullet}$  has fewer semantic restrictions than  $\models^\bullet$ . Also, since  $\phi' \models^{\bullet\bullet} q'$ ,  $\mathcal{I}' \models^{\bullet\bullet} q'$ . Because  $\mathcal{I}' \models^\bullet \phi'$ ,  $\mathcal{I}'$  guarantees the semantic restrictions required by  $\models^\bullet$ . Hence  $\mathcal{I}' \models^\bullet q'$  always holds and  $\phi' \models^\bullet q'$  is proved.

Next we prove the ‘only if’ part, which is equivalent to  $\phi' \not\models^\bullet q'$  if  $\phi' \not\models^{\bullet\bullet} q'$ . If  $\phi' \not\models^{\bullet\bullet} q'$ , there exists a counter-model  $\mathcal{I}''$  for  $\phi' \models^{\bullet\bullet} q'$ . According to the assumption of the theorem, there exists a counter-model  $\mathcal{I}'$  for  $\phi' \models^\bullet q'$ , hence  $\phi' \not\models^\bullet q'$  is proved. So the theorem holds.  $\square$

## 5.2 Unnesting

In this section we will define the unnesting transformation  $\text{unnest}(\alpha)$  for a given atomic formula  $\alpha$ , which is extended from the definition in [20]. Before performing  $\text{unnest}(\alpha)$ , any ‘\_’ and ‘?’ used as OIDs in  $\alpha$  need to be eliminated, because they cannot be used as co-references for the same constant/variable in two separate formulas. The elimination is done by replacing each ‘\_’ with a fresh constant and each ‘?’ with a fresh variable in the universal scope of the enclosing clause.

In the following, we define  $\text{unnest}(\alpha)$  based on the recursive  $\text{Atoms}$ . Here,

$\text{oid}(t)$  denotes the OID of an oidful term  $t$ . Also,  $\text{Parts}(t)$  denotes the set of top-level components of an atomic formula or a term  $t$ , including, optionally,  $\text{oid}(t)$ , its tuples, slot names, slot fillers, as well as its predicate/function.

**Definition 5.4.** (*Unnesting*) Given an atomic formula  $\alpha$ , its unnesting  $\text{unnest}^*(\alpha)$  is defined as

$$\begin{aligned} \text{unnest}^*(\alpha) &::= \text{And}(\sigma_1 \dots \sigma_n) \quad \text{s.t. } \{\sigma_1, \dots, \sigma_n\} = \cup_{t \in \text{Parts}(\alpha)} \text{Atoms}(t) \cup \{\text{trim}_{\text{unn}}(\alpha)\} \\ \text{Atoms}(t) &::= \begin{cases} \emptyset & t \text{ is a simple term} \\ \cup_{s \in \text{Parts}(t)} \text{Atoms}(s) & t \text{ is oidless (expression)} \\ \cup_{s \in \text{Parts}(t)} \text{Atoms}(s) \cup \{\text{trim}_{\text{unn}}(t)\} & t \text{ is oidful (atom)} \end{cases} \\ \text{trim}_{\text{unn}}(t) &::= \text{Construct obtained by replacing every } s \in \text{Parts}(t) \text{ in } t \text{ with } \text{retain}_{\text{unn}}(s) \\ \text{retain}_{\text{unn}}(t) &::= \begin{cases} t & t \text{ is a simple term} \\ \text{trim}_{\text{unn}}(t) & t \text{ is oidless (expression)} \\ \text{retain}_{\text{unn}}(\text{oid}(t)) & t \text{ is oidful (atom)} \end{cases} \end{aligned}$$

$\text{unnest}^*$  is extended to a PSOA transformation  $\text{unnest}$  via  $\text{unnest}'$  according to Definition 5.1. ◇

For each atomic formula  $\alpha$ ,  $\text{unnest}^*(\alpha)$  is a conjunction of formulas  $\sigma_i$  without embedded atoms. Each  $\sigma_i$  is a trimmed version of the top-level formula  $\alpha$  or of some embedded atom. The set  $\text{Atoms}(t)$  contains each  $\sigma_i$  trimmed from an atom embedded in  $t$  or  $t$  itself. It is constructed by recursively traversing through each component  $s \in \text{Parts}(t)$ , collecting  $\text{Atoms}(s)$  into  $\text{Atoms}(t)$ , and then adding  $\text{trim}_{\text{unn}}(t)$  to  $\text{Atoms}(t)$  if  $t$  is oidful, which indicates that  $t$  is an atom. The transformation  $\text{trim}_{\text{unn}}(t)$  splits off all embedded atoms from  $t$  and leaves behind its ‘ultimate’ OID for each of them. It is constructed by replacing each  $s \in \text{Parts}(t)$  with  $\text{retain}_{\text{unn}}(s)$ , which defines the left-behind term for each embedded term  $s$ .

Since the definition of PSOA semantics can be applied only after applying the unnesting transformation, the transformation is trivially semantics-preserving. In the following, we use an example to explain the unnesting transformation.

**Example 5.1.** Let the input formula  $\alpha$  be  $\text{o1\#c(p}\rightarrow\text{f(o2\#c\#d))}$ . Note that ‘#’ is left-associative, hence the embedded atom  $\text{o2\#c\#d}$  is interpreted to have the OID  $\text{o2\#c}$  and the class  $\text{d}$ . The conjuncts of  $\text{unnest}(\alpha)$  are constructed as follows:

$$\begin{aligned}
& \{\sigma_1, \dots, \sigma_n\} \\
&= \cup_{t \in \text{Parts}(\alpha)} \text{Atoms}(t) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= (\text{Atoms}(\text{o1}) \cup \text{Atoms}(\text{c}) \cup \text{Atoms}(\text{p}) \cup \text{Atoms}(\text{f}(\text{o2\#c\#d})) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= \text{Atoms}(\text{f}(\text{o2\#c\#d})) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= (\text{Atoms}(\text{f}) \cup \text{Atoms}(\text{o2\#c\#d})) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= \text{Atoms}(\text{o2\#c\#d}) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= (\text{Atoms}(\text{o2\#c}) \cup \text{Atoms}(\text{d}) \cup \{\text{trim}_{\text{unn}}(\text{o2\#c\#d})\}) \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= \text{Atoms}(\text{o2\#c}) \bigcup \{\text{trim}_{\text{unn}}(\text{o2\#c\#d})\} \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= (\text{Atoms}(\text{o2}) \cup \text{Atoms}(\text{c}) \cup \{\text{trim}_{\text{unn}}(\text{o2\#c})\}) \bigcup \{\text{trim}_{\text{unn}}(\text{o2\#c\#d})\} \bigcup \{\text{trim}_{\text{unn}}(\alpha)\} \\
&= \{\text{trim}_{\text{unn}}(\text{o2\#c}), \text{trim}_{\text{unn}}(\text{o2\#c\#d}), \text{trim}_{\text{unn}}(\alpha)\}
\end{aligned}$$

The  $\text{trim}_{\text{unn}}$  transformations work as follows, using the recursive  $\text{retain}_{\text{unn}}$  transformation.

$$\begin{aligned}
\text{trim}_{\text{unn}}(\text{o2\#c}) &= \text{retain}_{\text{unn}}(\text{o2})\#\text{retain}_{\text{unn}}(\text{c}) = \text{o2\#c} \\
\text{trim}_{\text{unn}}(\text{o2\#c\#d}) &= \text{retain}_{\text{unn}}(\text{o2\#c})\#\text{retain}_{\text{unn}}(\text{d}) \\
&= \text{retain}_{\text{unn}}(\text{oid}(\text{o2\#c}))\#\text{d} = \text{retain}_{\text{unn}}(\text{o2})\#\text{d} = \text{o2\#d}
\end{aligned}$$

$$\begin{aligned}
\text{trim}_{\text{unn}}(\alpha) &= \text{trim}_{\text{unn}}(\text{o1\#c(p}\rightarrow\text{f(o2\#c\#d))}) \\
&= \text{retain}_{\text{unn}}(\text{o1})\#\text{retain}_{\text{unn}}(\text{c})(\text{retain}_{\text{unn}}(\text{p})\rightarrow\text{retain}_{\text{unn}}(\text{f(o2\#c\#d)})) \\
&= \text{o1\#c(p}\rightarrow\text{retain}_{\text{unn}}(\text{f})(\text{retain}_{\text{unn}}(\text{o2\#c\#d}))) \\
&= \text{o1\#c(p}\rightarrow\text{f}(\text{retain}_{\text{unn}}(\text{oid}(\text{o2\#c\#d})))) \\
&= \text{o1\#c(p}\rightarrow\text{f}(\text{retain}_{\text{unn}}(\text{o2\#c}))) \\
&= \text{o1\#d(p}\rightarrow\text{f(o2))}
\end{aligned}$$

Hence, the unnesting  $\text{unnest}(\alpha)$  results in  $\text{And}(\text{o2\#c o2\#d o1\#c(p}\rightarrow\text{f(o2))})$ .

◇

## 5.3 Objectification

In PSOA RuleML, each oidless atom  $\sigma$  is understood as having an implicit OID, which allows the interchangeable use of the respective term forms (3.1) and (3.2) in Section 3.2.2.

In this section, we will discuss an objectification systematics for oidless atoms, including undifferentiated and differentiated static objectification transformations, as well as a novel static/dynamic transformation. In contrast to the static objectification, which generates explicit OIDs for every oidless atom, the static/dynamic objectification generates as few explicit OIDs as possible, instead constructing virtual OIDs as query variable bindings.

### 5.3.1 Static Objectification Transformations

The static objectification  $\text{obj}_s(\alpha)$  of a KB/query  $\alpha$  is obtained by replacing each oidless atom  $\sigma$  with its objectified form  $\text{obj}_s(\sigma)$  having a generated OID. The

generation can adopt either an undifferentiated method  $\text{obj}_{s=}$ , which uniformly transforms  $\sigma$  everywhere, or a differentiated method  $\text{obj}_{s\neq}$ , which transforms  $\sigma$  differently based on its occurrence.

**Definition 5.5.** (*Undifferentiated static objectification*) *The undifferentiated static objectification transformation  $\text{obj}_{s=}$  is built from an auxiliary transformation  $\text{obj}_{s=}^*$  via  $\text{obj}_{s=}^{\prime}$  according to Definition 5.1.  $\text{obj}_{s=}^*(\sigma)$  is defined for each oidless atom  $\sigma$  as  $\text{Exists } ?i \text{ } (?i\#f(\dots))$ , where  $?i$  is a fresh variable in the enclosing clause chosen from  $?1, ?2, \dots$*   $\diamond$

The following theorem shows that  $\text{obj}_{s=}$  preserves semantics.

**Theorem 5.3.** *For any  $\models^\bullet \in S_{\models}$  that requires the objectification restriction,  $\text{obj}_{s=}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_K$ ,  $\models^\bullet$ , and  $\models_{-o}^\bullet$ .*

*Proof.* We first show that for every KB  $\phi$  and boolean query  $q$  in  $\mathcal{P}_K$ , the condition of Corollary 5.1 is fulfilled, where  $\text{tr}_{\text{PSOA}}^*$  is  $\text{obj}_{s=}^*$ ,  $\text{tr}'_{\text{PSOA}}$  is  $\text{obj}_{s=}^{\prime}$ , and  $\text{tr}_{\text{PSOA}}$  is  $\text{obj}_{s=}$ . For any semantic structure  $\mathcal{I}^*$  and any  $\tau$  being an oidless atom, if  $\mathcal{I}^*$  guarantees the objectification restriction required by  $\models^\bullet$ , then  $TVal_{\mathcal{I}^*}(\tau) = \mathbf{t}$  iff  $TVal_{\mathcal{I}^*}(\text{obj}_{s=}^{\prime}(\tau)) = \mathbf{t}$  holds trivially based on the definition of objectification restriction. By Corollary 5.1,  $\phi \models^\bullet q$  iff  $\text{obj}_{s=}(\phi) \models^\bullet \text{obj}_{s=}(\phi, q)$ .

Next we will show that the condition of Theorem 5.2 is fulfilled for  $\models^{\bullet\bullet}$  being  $\models_{-o}^\bullet$ ,  $\phi'$  being  $\text{obj}_{s=}(\phi)$ , and  $q'$  being  $\text{obj}_{s=}(\phi, q)$ . If there exists a counter-model  $\mathcal{I}''$  for  $\text{obj}_{s=}(\phi) \models_{-o}^\bullet \text{obj}_{s=}(\phi, q)$ , then it can be modified into a model  $\mathcal{I}'$  that guarantees the objectification restriction, by redefining  $TVal_{\mathcal{I}'}(\sigma)$  for every oidless atom  $\sigma$  to be the same as  $TVal_{\mathcal{I}''}(\text{obj}_{s=}^{\prime}(\sigma))$ . Since the change only affects oidless atoms, which neither exist in  $\text{obj}_{s=}(\phi)$  nor in

$\text{obj}_{s=}(\phi, q)$ ,  $TVal_{\mathcal{T}'}(\text{obj}_{s=}(\phi)) = TVal_{\mathcal{T}''}(\text{obj}_{s=}(\phi)) = \mathbf{t}$  and  $TVal_{\mathcal{T}'}(\text{obj}_{s=}(\phi, q)) = TVal_{\mathcal{T}''}(\text{obj}_{s=}(\phi, q)) = \mathbf{f}$ . Hence  $\mathcal{T}'$  is a counter-model for  $\text{obj}_{s=}(\phi) \models^\bullet \text{obj}_{s=}(\phi, q)$  and the condition of Theorem 5.2 is fulfilled. Thus  $\text{obj}_{s=}(\phi) \models^\bullet \text{obj}_{s=}(\phi, q)$  iff  $\text{obj}_{s=}(\phi) \models_{-o}^\bullet \text{obj}_{s=}(\phi, q)$ .

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

**Definition 5.6.** (*Differentiated static objectification*) *The differentiated static objectification transformation  $\text{obj}_{s\neq}$  is built from a preliminary version  $\text{obj}_{s\neq}^*$  via  $\text{obj}'_{s\neq}$  according to Definition 5.1.  $\text{obj}_{s\neq}^*(\sigma)$  is defined for each oidless atom  $\sigma$  having the form  $\mathbf{f}(\dots)$  as follows.*

*Case 1: If  $\sigma$  is a ground fact,  $\text{obj}_{s\neq}^*(\sigma) = \_i \# \mathbf{f}(\dots)$ , where  $\_i$  is a fresh local constant symbol chosen from  $\_1, \_2, \dots$ , which neither occurs elsewhere in  $\text{KB}(\sigma)$  nor is used for the objectification of other atoms. Here, the set of reserved names  $\{\_1, \_2, \dots\}$  could be replaced with any other alphabet containing a countably infinite number of reserved constant symbols.*

*Case 2: If  $\sigma$  is a universal fact, a rule conclusion atom, or a query atom,  $\text{obj}_{s\neq}^*(\sigma) = \text{Exists } ?j ( ?j \# \mathbf{f}(\dots) )$  where  $?j$  is a fresh variable in the enclosing clause chosen from  $?1, ?2, \dots$*

*Case 3: If  $\sigma$  is a rule condition atom,  $\text{obj}_{s\neq}^*(\sigma) = ?j \# \mathbf{f}(\dots)$ , where  $?j$  is a fresh variable chosen from  $?1, ?2, \dots$  and scoped universally by the enclosing rule.*  $\diamond$

Next we prove the semantics preservation for  $\text{obj}_{s\neq}$  by showing that it leads to the same entailments as  $\text{obj}_{s=}$ .

**Lemma 5.1.** *Let  $\models^\bullet \in S_{\models}$  be a PSOA semantics that requires the objectification restriction. Given a KB  $\phi$  and a boolean query  $q$  in  $\mathcal{P}_K$ , if  $q$  does not use*

a generated OID constant symbol  $\_1, \_2, \dots$ , then  $\text{obj}_{s=}(\phi) \models_{-o}^{\bullet} \text{obj}_{s=}(\phi, q)$  iff  $\text{obj}_{s\neq}(\phi) \models_{-o}^{\bullet} \text{obj}_{s\neq}(\phi, q)$  holds.

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

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

**Theorem 5.4.** *Let  $\mathcal{P}_{\text{obj}_{s\neq}} = (\Phi_K, Q_{\text{obj}_{s\neq}})$  be the sublanguage of  $\mathcal{P}_K$  where  $Q_{\text{obj}_{s\neq}}$  is the subset of  $Q_K$  that does not use constants  $\_1, \_2, \dots$ . Given any  $\models_{\perp}^{\bullet} \in S_{\perp}$  that requires the objectification restriction, then  $\text{obj}_{s\neq}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_{\text{obj}_{s\neq}}, \models_{\perp}^{\bullet}$ , and  $\models_{-o}^{\bullet}$ .*

*Proof.* For any  $\phi \in \Phi_K$  and  $q \in Q_{\text{obj}_{s\neq}}$ , by Theorem 5.3 we have  $\phi \models_{\perp}^{\bullet} q$  iff  $\text{obj}_{s=}(\phi) \models_{-o}^{\bullet} \text{obj}_{s=}(\phi, q)$ . By Lemma 5.1,  $\text{obj}_{s=}(\phi) \models_{-o}^{\bullet} \text{obj}_{s=}(\phi, q)$  iff  $\text{obj}_{s\neq}(\phi) \models_{-o}^{\bullet} \text{obj}_{s\neq}(\phi, q)$ . Hence  $\phi \models_{\perp}^{\bullet} q$  iff  $\text{obj}_{s\neq}(\phi) \models_{-o}^{\bullet} \text{obj}_{s\neq}(\phi, q)$ . So, by Definition 4.4, the theorem holds.  $\square$

### 5.3.2 Static/Dynamic Objectification Transformation

For KBs in which most or all of the predicates are Prolog-like relations, it is often not necessary to generate OIDs for their oidless atoms explicitly. In this subsection, a novel static/dynamic objectification approach is introduced to keep unchanged as many of the KB's oidless atoms as possible, instead constructing virtual OIDs at query time if and when bindings for OID variables are being queried. In particular, the approach avoids certain single-dependent-tuple atoms being given explicit OIDs and distributed to a conjunction as explained in Section 5.6.

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

**Definition 5.7.** (*Non-relational and relational predicates*) Given a KB  $\phi$ , a predicate  $\mathbf{f} \in \text{PredKB}(\phi)$  is **non-relational** in  $\phi$  if  $\mathbf{f}$  occurs at least once in an oidful, independent-tuple-containing, multi-tuple, descriptorless, or slotted atom of  $\phi$ , or in a subclass formula of  $\phi$ . Conversely,  $\mathbf{f}$  is **relational** in  $\phi$  if it has no such occurrence.  $\mathbf{f}$  is said to have an arity  $\mathbf{k}$  in  $\phi$  if there exists an atom  $\mathbf{f}([t_1 \dots t_k])$  in  $\phi$ . The sets of non-relational and relational predicates of  $\phi$  are written as  $\text{PredKB}_{NR}(\phi)$  and  $\text{PredKB}_R(\phi)$ , respectively.  $\diamond$

For atoms using a relational predicate in  $\text{PredKB}_R(\phi)$ , their OIDs can be virtualized by dynamic objectification.

**Definition 5.8.** (*Dynamic objectification*) The dynamic objectification trans-

formation  $\text{obj}_d^*(\omega)$  of an atom  $\omega$  having a relational KB predicate in  $\phi = \text{KB}(\omega)$ , i.e.  $\text{Pred}(\omega) \in \text{PredKB}_R(\phi)$ , is defined as follows.

Case 1 If  $\omega$  is a KB atom,  $\text{obj}_d^*(\omega) = \omega$ .

Case 2 If  $\omega$  is a query atom, there are five subcases:

Case 2.1 If  $\omega$  is a relationship,  $\text{obj}_d^*(\omega) = \omega$ .

Case 2.2 If  $\omega$  has a non-variable (e.g., constant or expression) OID, a slot, or an independent tuple,  $\text{obj}_d^*(\omega) = \text{Or}()$ , where  $\text{Or}()$  is an encoding of explicit falsity.

Case 2.3 If  $\omega$  has a variable OID and  $m > 0$  dependent tuples, being of the form  $?0\#f(+[\mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1}] \dots + [\mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m}])$ , equivalent to a conjunction separately applying  $?0\#f$  to all tuples,

$$\text{And}(?0\#f(+[\mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1}]) \dots ?0\#f(+[\mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m}])),$$

then  $\text{obj}_d^*(\omega)$  is a relational conjunction querying the oidless versions of these applications while using explicit equalities between the OID variable  $?0$  and a OID-constructor function  $\_oidcons$  applied to the predicate and the elements of each tuple (where the  $?0$  equalities also enforce tuple unification, so that the  $m$  single-tuple  $f$  relationships can be satisfied by a single KB clause, thus realizing special ‘virtual multi-tuple’ atoms as queries):

$$\begin{aligned} &\text{And}(f(+[\mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1}]) ?0 = \_oidcons(f \mathbf{t}_{1,1} \dots \mathbf{t}_{1,n_1})) \\ &\dots \\ &f(+[\mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m}]) ?0 = \_oidcons(f \mathbf{t}_{m,1} \dots \mathbf{t}_{m,n_m})) \end{aligned}$$

In the special case of  $m = 1$ , the query atom  $?0\#f(+[t_1 \dots t_n])$  becomes

$$\text{And}(f(+[t_1 \dots t_n]) \ ?0 = \_oidcons(f \ t_1 \dots t_n))$$

The function `\_oidcons` is employed to universally construct OIDs for all relationships so that it needs to take the predicate symbol as the first argument to distinguish between relationships with different predicates. Here, the reserved name `\_oidcons` could be replaced with any other reserved constant symbol.

*Case 2.4* If  $\omega$  is a membership being of the form  $?0\#f()$ ,  $\text{obj}_d^*(\omega)$  is a disjunction of  $k$  formulas  $\text{obj}_d^*(?0\#f(+[?X_1 \dots ?X_{n_1}]))$ , where  $n_1, \dots, n_k$  are the  $k$  different arities of  $f$  in the KB:

$$\text{Or}(\text{obj}_d^*(?0\#f(+[?X_1 \dots ?X_{n_1}])) \dots \text{obj}_d^*(?0\#f(+[?X_1 \dots ?X_{n_k}])))$$

*Case 2.5* If  $\omega$  being of the form  $f(\dots)$  has no OID but  $m$  dependent tuples,  $m \neq 1$ ,  $\text{obj}_d^*(\omega) = \text{Exists } ?0 (\text{obj}_d^*(?0\#f(\dots)))$ , where  $?0$  is a fresh variable in the enclosing query and  $\text{obj}_d^*(?0\#f(\dots))$  is computed according to either *Case 2.3* or *Case 2.4*. ◇

**Definition 5.9.** (Static/dynamic objectification) Let  $\text{obj}_s^*$  be a transformation chosen from  $\{\text{obj}_{s \neq}^*, \text{obj}_{s =}^*\}$  and  $\phi$  be a KB. The static/dynamic objectification transformation  $\text{obj}_{s+d}^*(\omega)$  of an atom is defined as

$$\text{obj}_{s+d}^*(\omega) = \begin{cases} \text{obj}_d^*(\omega) & \text{Pred}(\omega) \in \text{PredKB}_R(\text{KB}(\omega)) \\ \text{obj}_s^*(\omega) & \text{Pred}(\omega) \in \text{PredKB}_{NR}(\text{KB}(\omega)) \end{cases} \quad (5.1)$$

where  $\text{KB}(\omega)$  must comply to the following restrictions for the application of

*dynamic objectification:*

(i) For every KB clause, universal variables occurring in its conclusion must also occur in its condition (e.g., prohibiting universal, “non-ground” facts).

(ii) There does not exist a predicate variable.

$\text{obj}_{s+d}^*$  is extended to a PSOA transformation  $\text{obj}_{s+d}$  via  $\text{obj}'_{s+d}$  according to Definition 5.1. The two  $\text{obj}_{s+d}$  versions using  $\text{obj}_{s=}$  and  $\text{obj}_{s\neq}$  for the static part are notated as  $\text{obj}_{s=+d}$  and  $\text{obj}_{s\neq+d}$ , respectively.  $\diamond$

In the following, we discuss the semantics preservation of  $\text{obj}_{s=+d}$  and  $\text{obj}_{s\neq+d}$ , for the respective PSOA sublanguages  $\mathcal{P}_{\text{obj}_{s=+d}}$  and  $\mathcal{P}_{\text{obj}_{s\neq+d}}$ .  $\mathcal{P}_{\text{obj}_{s=+d}} = (\Phi_{\text{obj}_{s=+d}}, Q_{\text{obj}_{s=+d}})$  is a PSOA sublanguage where: (1)  $\Phi_{\text{obj}_{s=+d}}$  is the subset of  $\Phi_K$  that conforms to conditions (i) and (ii) above and does not use `_oidcons`; (2)  $Q_{\text{obj}_{s=+d}}$  is the subset of  $Q_K$  that has no predicate variable and does not use `_oidcons`. In contrast,  $\mathcal{P}_{\text{obj}_{s\neq+d}} = (\Phi_{\text{obj}_{s\neq+d}}, Q_{\text{obj}_{s\neq+d}})$  is a PSOA sublanguage where: (1)  $\Phi_{\text{obj}_{s\neq+d}}$  is the same as  $\Phi_{\text{obj}_{s=+d}}$ ; (2)  $Q_{\text{obj}_{s\neq+d}}$  is the subset of  $Q_{\text{obj}_{s=+d}}$  that does not use constants `_1`, `_2`, `...`. The notations  $BQ_{\text{obj}_{s=+d}}$  and  $BQ_{\text{obj}_{s\neq+d}}$  denote the boolean query subset of  $Q_{\text{obj}_{s=+d}}$  and  $Q_{\text{obj}_{s\neq+d}}$  respectively.

**Lemma 5.2.** *Let  $\models^\bullet$  be an entailment in  $S_{\models}$  that requires the objectification restriction and the description restriction. For any  $\phi \in \Phi_{\text{obj}_{s=+d}}$  and  $q \in BQ_{\text{obj}_{s=+d}}$ ,  $\phi \models^\bullet q$  iff  $\text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$ .*

*Proof.* We will show that  $\text{obj}'_{s=+d}$  has Properties (1) and (2) in Theorem 5.1 for  $\text{tr}_{\text{PSOA}}^*$  being  $\text{obj}_{s=+d}^*$ ,  $\text{tr}'_{\text{PSOA}}$  being  $\text{obj}'_{s=+d}$ , and  $\text{tr}_{\text{PSOA}}$  being  $\text{obj}_{s=+d}$ .

- Property (1)

Let  $\mathcal{I}$  be a counter-model for  $\phi \models^\bullet q$ . This  $\mathcal{I}$  can be modified into a semantic structure  $\mathcal{I}'$  by adding the interpretation of `_oidcons`. For every atom  $\omega$  of the form  $f(+[\mathbf{t}_1 \dots \mathbf{t}_n])$  s.t.  $TVal_{\mathcal{I}}(\omega) = \mathbf{t}$  and  $f \in \text{PredKB}_R(\phi)$ , since  $\mathcal{I} \models^\bullet \phi$ ,  $\mathcal{I}$  must guarantee the objectification restriction required by  $\models^\bullet$ , thus  $TVal_{\mathcal{I}}(\text{Exists } ?0 (?0\#f(+[\mathbf{t}_1 \dots \mathbf{t}_n]))) = \mathbf{t}$  and there exists a mapping  $I_{V^*} \in \mathbf{M}(\{?0\}, \mathcal{I}.D)$  s.t.  $v(\mathcal{I}, I_{V^*}) \models^\bullet ?0\#f(+[\mathbf{t}_1 \dots \mathbf{t}_n])$ . We can define  $\mathcal{I}'.I_{\text{psoa}}(\mathcal{I}'.I(\text{\_oidcons}))$  to make  $\mathcal{I}'.I(\text{\_oidcons}(f \mathbf{t}_1 \dots \mathbf{t}_n)) = I_{V^*}(?0)$ .

We will prove that  $\mathcal{I}'$  satisfies the conditions (1.a) and (1.b) for every atom  $\omega$  and every  $I_V \in \mathbf{M}(\text{Var}(\omega), \mathcal{I}.D)$  via a case-by-case analysis for  $\omega$ .

Case 1 If  $\text{Pred}(\omega) \in \text{PredKB}_{NR}(\phi)$ , there are two subcases.

Case 1.1 If  $\omega$  is oidful, then  $\text{obj}'_{s=+d}(\omega) = \omega$ . Since  $\mathcal{I}'$  and  $\mathcal{I}$  only differ in the interpretation of `_oidcons`,  $v(\mathcal{I}, I_V) \models^\bullet \omega$  iff  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$  holds. Hence conditions (1.a) and (1.b) are satisfied.

Case 1.2 If  $\omega$  is oidless, then  $\text{obj}'_{s=+d}(\omega)$  is  $\omega$ 's existential form  $\text{obj}'_{s=}(\omega)$ . Since  $\models^\bullet$  requires the objectification restriction,  $v(\mathcal{I}, I_V) \models^\bullet \omega$  iff  $v(\mathcal{I}, I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$  holds. Since  $\mathcal{I}'$  and  $\mathcal{I}$  only differ in the interpretation of `_oidcons`,  $v(\mathcal{I}, I_V) \models^\bullet \omega$  iff  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$  holds. Hence conditions (1.a) and (1.b) are satisfied.

Case 2 If  $\text{Pred}(\omega) \in \text{PredKB}_R(\phi)$ ,  $\text{obj}'_{s=+d}(\omega) = \text{obj}^*_d(\omega)$ . There are the following subcases, each of which corresponds to a case in Definition 5.8.

Case 2.1 If  $\omega$  is a KB atom,  $\text{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, conditions (1.a) and (1.b) are satisfied.

Case 2.2 If  $\omega$  is a query atom, then  $\omega$  has a negative occurrence and we need to show that (1.b) is satisfied for every subcase of Case 2 in Definition 5.8.

Case 2.2.1 In this case,  $\text{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, condition (1.b) is satisfied.

Case 2.2.2 In this case,  $\text{obj}'_{s=+d}(\omega) = \text{Or}()$ , so  $v(\mathcal{I}', I_V) \not\models^\bullet \text{obj}'_{s=+d}(\omega)$ . Thus condition (1.b) is satisfied.

Case 2.2.3 In this case  $\text{obj}'_{s=+d}(\omega)$  is a conjunction of relationships and equalities. If  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$ , then  $v(\mathcal{I}', I_V) \models^\bullet f(+[\mathfrak{t}_{i,1} \dots \mathfrak{t}_{i,n_i}])$  and  $v(\mathcal{I}', I_V) \models^\bullet ?0 = \text{\_oidcons}(f \mathfrak{t}_{i,1} \dots \mathfrak{t}_{i,n_i})$  hold for  $i = 1, \dots, n$ . Hence for each  $i$ ,  $v(\mathcal{I}', I_V) \models^\bullet \text{\_oidcons}(f \mathfrak{t}_{i,1} \dots \mathfrak{t}_{i,n_i}) \# f(+[\mathfrak{t}_{i,1} \dots \mathfrak{t}_{i,n_i}])$  holds. According to the definition of  $\text{\_oidcons}$ 's interpretation in  $\mathcal{I}'$ ,  $v(\mathcal{I}, I_V) \models^\bullet \omega$  holds and condition (1.b) is satisfied.

Case 2.2.4 In this case  $\text{obj}'_{s=+d}(\omega)$  is a disjunction. If  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$ , then there exists  $i$  s.t.  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(?0 \# f(+[\mathfrak{?X}_1 \dots \mathfrak{?X}_{n_i}]))$ . According to Case 2.2.3, we have  $v(\mathcal{I}, I_V) \models^\bullet ?0 \# f(+[\mathfrak{?X}_1 \dots \mathfrak{?X}_{n_i}])$ . Since  $\models^\bullet$  requires the description restriction,  $v(\mathcal{I}, I_V) \models^\bullet ?0 \# f()$  holds and condition (1.b) is satisfied.

Case 2.2.5 In this case  $\text{obj}'_{s=+d}(\omega) = \text{Exists } ?0 (\text{obj}'_{s=+d}(?0 \# f(\dots)))$ . If  $v(\mathcal{I}', I_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$ , then there exists  $I_{V^*} \in \mathbf{M}(\{\mathfrak{?0}\}, \mathcal{I}'.D)$  s.t.  $v(\mathcal{I}', I_V \cup I_{V^*}) \models^\bullet \text{obj}'_{s=+d}(?0 \# f(\dots))$ . According to Cases 2.2.3 and 2.2.4,  $v(\mathcal{I}, I_V \cup I_{V^*}) \models^\bullet ?0 \# f(\dots)$  holds. Thus  $v(\mathcal{I}, I_V) \models^\bullet \text{Exists } ?0 (?0 \# f(\dots))$ . Since  $\models^\bullet$  requires the objectification restriction, we have  $v(\mathcal{I}, I_V) \models^\bullet \omega$ . So condition (1.b) is satisfied.

For non-atom atomic formulas (cf. Sections 2.3.2),  $\text{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, conditions (1.a) and (1.b) are also satisfied. Hence  $\text{obj}'_{s=+d}$  has

Property (1).

- Property (2)

Let  $\mathcal{I}'$  be a counter-model for  $\text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$ . We can modify  $\mathcal{I}'$  into a semantic structure  $\mathcal{I}$  by making the following changes to the semantic function  $\mathbf{I}_{\text{psoa}}(\mathbf{I}(\mathbf{r}))$  for every relational predicate  $\mathbf{r} \in \text{PredKB}_R(\phi)$ :

- (a) Make  $\mathbf{I}_{\text{truth}}(\mathbf{I}_{\text{psoa}}(\mathbf{I}(\mathbf{r}))(\mathbf{S}_o, \mathbf{B}_{\text{tups}}^+, \mathbf{B}_{\text{tups}}^-, \mathbf{B}_{\text{slots}}^+, \mathbf{B}_{\text{slots}}^-)) = \mathbf{f}$  if  $\mathbf{B}_{\text{tups}}^-, \mathbf{B}_{\text{slots}}^+$  or  $\mathbf{B}_{\text{slots}}^-$  is non-empty,  $\mathbf{B}_{\text{tups}}^+$  contains different interpreted tuples or an interpreted tuple whose length is not an arity of  $\mathbf{r}$  in the KB  $\phi$ . With this change, for every atom  $\omega$  and every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\omega), \mathcal{I}.D)$  s.t.  $\text{Pred}(\omega) = \mathbf{r}$ ,  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\omega) = \mathbf{t}$  is possible only if  $\mathbf{B}_{\text{tups}}^-, \mathbf{B}_{\text{slots}}^+$  or  $\mathbf{B}_{\text{slots}}^-$  are all empty and the length of every dependent tuple of  $\omega$  is an arity of  $\mathbf{r}$  in the KB  $\phi$ .
- (b) For every interpreted tuple  $\langle \mathbf{e}_1, \dots, \mathbf{e}_n \rangle$  where  $\mathbf{e}_1, \dots, \mathbf{e}_n \in \mathcal{I}.D$ , if  $\mathbf{I}_{\text{truth}}(\mathbf{I}_{\text{psoa}}(\mathbf{I}(\mathbf{r}))(\{\}, \{\langle \mathbf{e}_1, \dots, \mathbf{e}_n \rangle\}, \{\}, \{\}, \{\})) = \mathbf{t}$  after modification (a) is applied and  $\mathbf{I}(\_oidcons)$  is defined, then we make  $\mathbf{I}_{\text{truth}}(\mathbf{I}_{\text{psoa}}(\mathbf{I}(\mathbf{r}))(\{\mathbf{e}_o\}, \{\langle \mathbf{e}_1, \dots, \mathbf{e}_n \rangle\}, \{\}, \{\}, \{\}))$  as well as  $\mathbf{I}_{\text{truth}}(\mathbf{I}_{\text{psoa}}(\mathbf{I}(\mathbf{r}))(\{\mathbf{e}_o\}, \{\}, \{\}, \{\}, \{\}))$  true if and only if  $\mathbf{e}_o = \mathbf{I}_{\text{psoa}}(\mathbf{I}(\_oidcons))(\{\}, \{\langle \mathbf{I}(\mathbf{r}), \mathbf{e}_1, \dots, \mathbf{e}_n \rangle\}, \{\}, \{\}, \{\})$ . With this change, for every atom  $\omega$  having the form  $\mathbf{r}(+[\mathbf{t}_1 \dots \mathbf{t}_n])$  and every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\omega) \cup \{?0\}, \mathcal{I}.D)$ , if  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(?0\#\mathbf{r}(+[\mathbf{t}_1 \dots \mathbf{t}_n])) = \mathbf{t}$  then  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V)(?0) = \mathbf{v}(\mathcal{I}', \mathbf{I}_V)(\_oidcons(\mathbf{r} \mathbf{t}_1 \dots \mathbf{t}_n))$ .

We will prove that  $\mathcal{I}$  satisfies conditions (2.a) and (2.b) for every atom  $\omega$  and every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\omega), \mathcal{I}.D)$  via a case-by-case analysis for  $\omega$ .

Case 1 If  $\text{Pred}(\omega) \in \text{PredKB}_{NR}(\phi)$ , since  $\mathcal{I}'$  and  $\mathcal{I}$  only differ in the truth evaluation of atoms having a relational predicate and it can be verified that  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V)$  guarantees the same semantic restrictions as  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V)$ ,  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \omega$  iff  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \omega$  holds. In order for  $\mathcal{I}$  to satisfy conditions (2.a) and (2.b), we only need to show that  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \omega$  iff  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$ , which will be shown in the following two subcases.

Case 1.1 If  $\omega$  is oidful, then  $\text{obj}'_{s=+d}(\omega) = \omega$ . Hence  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \omega$  iff  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$  holds trivially.

Case 1.2 If  $\omega$  is oidless, then  $\text{obj}'_{s=+d}(\omega)$  is  $\omega$ 's existential form  $\text{obj}'_s(\omega)$ . Since  $\models^\bullet$  requires the objectification restriction,  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{obj}'_{s=+d}(\omega)$  iff  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V) \models^\bullet \omega$  also holds.

Case 2 If  $\text{Pred}(\omega) \in \text{PredKB}_R(\phi)$ ,  $\text{obj}'_{s=+d}(\omega) = \text{obj}^*_d(\omega)$ . There are the following subcases, each of which corresponds to a case in Definition 5.8.

Case 2.1 If  $\omega$  is a KB atom,  $\text{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, conditions (2.a) and (2.b) are satisfied.

Case 2.2 If  $\omega$  is a query atom, then  $\omega$  has a negative occurrence and we need to show that (2.b) is satisfied for every subcase of Case 2 in Definition 5.8. Since  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V)$  guarantees semantic restrictions required by  $\models^\bullet$ , we only need to show that if  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \omega$ ,  $TVal_{\mathbf{v}(\mathcal{I}', \mathbf{I}_V)}(\text{obj}'_{s=+d}(\omega)) = \mathbf{t}$ .

Case 2.2.1 In this case,  $\text{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, condition (2.b) is satisfied.

Case 2.2.2 In this case, since  $\omega$  does not use the symbol `_oidcons`,  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \not\models^\bullet \omega$  holds according to the definition of  $\mathcal{I}$ , hence condition (2.b) is satisfied.

Case 2.2.3 In this case, if  $v(\mathcal{I}, \mathbf{I}_V) \models^\bullet \omega$  holds, then for  $i = 1, \dots, m$ ,  $v(\mathcal{I}, \mathbf{I}_V) \models^\bullet ?0\#f(+[\mathbf{t}_{i,1} \dots \mathbf{t}_{i,n_i}])$  holds. According to the above modification (b) that yields  $\mathcal{I}$ ,

$$\begin{aligned} & TVal_{v(\mathcal{I}', \mathbf{I}_V)}(f(+[\mathbf{t}_{i,1} \dots \mathbf{t}_{i,n_i}])) \\ &= TVal_{v(\mathcal{I}', \mathbf{I}_V)}(?0=_oidcons(f \mathbf{t}_{i,1} \dots \mathbf{t}_{i,n_i})) \\ &= \mathbf{t} \end{aligned}$$

Hence  $TVal_{v(\mathcal{I}', \mathbf{I}_V)}(\mathbf{obj}'_{s=+d}(\omega)) = \mathbf{t}$  and condition (2.b) is satisfied.

Case 2.2.4 In this case, if  $v(\mathcal{I}, \mathbf{I}_V) \models^\bullet \omega$ , then according to the modifications (a) and (b) that yields  $\mathcal{I}$ , there exists  $i$  and  $\mathbf{I}_V^* \in M(\{?X_1 \dots ?X_{n_i}\}, \mathcal{I}.D)$  s.t.  $v(\mathcal{I}, \mathbf{I}_V \cup \mathbf{I}_V^*) \models^\bullet ?0\#f(+[?X_1 \dots ?X_{n_i}])$  and  $n_i$  is an arity of  $f$  in  $\phi$ . According to Case 2.2.3, we have  $v(\mathcal{I}', \mathbf{I}_V \cup \mathbf{I}_V^*) \models^\bullet \mathbf{obj}'_{s=+d}(?0\#f(+[?X_1 \dots ?X_{n_i}]))$ . Thus  $v(\mathcal{I}', \mathbf{I}_V \cup \mathbf{I}_V^*) \models^\bullet \mathbf{obj}'_{s=+d}(?0\#f())$  and condition (2.b) is satisfied.

Case 2.2.5 In this case, if  $v(\mathcal{I}, \mathbf{I}_V) \models^\bullet \omega$ , then according to the modification (b) that yields  $\mathcal{I}$ , there exists  $\mathbf{I}_V^* \in M(\{?0\}, \mathcal{I}.D)$  s.t.  $v(\mathcal{I}, \mathbf{I}_V \cup \mathbf{I}_V^*) \models^\bullet ?0\#f(\dots)$ . By Cases 2.2.3 and 2.2.4,  $v(\mathcal{I}', \mathbf{I}_V \cup \mathbf{I}_V^*) \models^\bullet \mathbf{obj}'_{s=+d}(?0\#f(\dots))$ . Hence  $v(\mathcal{I}', \mathbf{I}_V) \models^\bullet \text{Exists } ?0(\mathbf{obj}'_{s=+d}(?0\#f(\dots)))$ . So condition (2.b) is satisfied.

For any non-atom atomic formula  $\omega$ ,  $\mathbf{obj}'_{s=+d}(\omega) = \omega$ . Analogous to Case 1.1, conditions (2.a) and (2.b) are also satisfied. Hence  $\mathbf{obj}'_{s=+d}$  has Property (2).

So, by Theorem 5.1, the lemma holds.  $\square$

**Lemma 5.3.** *Let  $\models^\bullet$  be an entailment in  $S_{\models}$  that requires the objectification restriction and the description restriction. For any  $\phi \in \Phi_{\mathbf{obj}_{s=+d}}$  and  $q \in$*

$BQ_{\text{obj}_{s=+d}}, \text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$  iff  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$ .

*Proof.* We need to show that the conditions of Theorem 5.2 can be fulfilled for  $\phi'$  being  $\text{obj}_{s=+d}(\phi)$ ,  $q'$  being  $\text{obj}_{s=+d}(\phi, q)$ , and  $\models^{\bullet\bullet}$  being  $\models_{-o}^\bullet$ . If  $\mathcal{I}''$  is a counter-model for  $\phi' \models_{-o} q'$ , then  $\mathcal{I}'' \models_{-o} \phi'$  and  $\mathcal{I}'' \not\models_{-o} q'$ . We modify  $\mathcal{I}''$  into a semantic structure  $\mathcal{I}'$  that guarantees the objectification restriction by making the following changes to the semantic function  $I_{\text{psoa}}(I(\mathbf{f}))$  for every predicate  $\mathbf{f}$ :

a) If  $\mathbf{f} \in \text{PredKB}_{NR}(\phi)$ , make

$$I_{\text{truth}}(I_{\text{psoa}}(I(\mathbf{f}))(\{\}, B_{\text{tups}}^+, B_{\text{tups}}^-, B_{\text{slots}}^+, B_{\text{slots}}^-)) = \mathbf{t}$$

if there exists a domain element  $\mathbf{e}$  such that

$$I_{\text{truth}}(I_{\text{psoa}}(I(\mathbf{f}))(\{\mathbf{e}\}, B_{\text{tups}}^+, B_{\text{tups}}^-, B_{\text{slots}}^+, B_{\text{slots}}^-)) = \mathbf{t}$$

Otherwise, make  $I_{\text{truth}}(I_{\text{psoa}}(I(\mathbf{f}))(\{\}, B_{\text{tups}}^+, B_{\text{tups}}^-, B_{\text{slots}}^+, B_{\text{slots}}^-)) = \mathbf{f}$ .

b) If  $\mathbf{f} \in \text{PredKB}_R(\phi)$ , make the same changes as explained in the proof of Property (2) in Lemma 5.2.

Since these changes affect only the truth evaluation of atoms that do not occur in  $\text{obj}_{s=+d}(\phi)$  or  $\text{obj}_{s=+d}(\phi, q)$  and for every  $I_V, \mathbf{v}(I', I_V)$  guarantees the objectification restriction, both  $TVal_{\mathcal{I}'}(\text{obj}_{s=+d}(\phi)) = TVal_{\mathcal{I}''}(\text{obj}_{s=+d}(\phi)) = \mathbf{t}$  and  $TVal_{\mathcal{I}'}(\text{obj}_{s=+d}(\phi, q)) = TVal_{\mathcal{I}''}(\text{obj}_{s=+d}(\phi, q)) = \mathbf{f}$  hold. Hence  $\mathcal{I}'$  is a counter-model for  $\text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$ . By Theorem 5.2, the lemma holds.  $\square$

**Theorem 5.5.**  $\text{obj}_{s=+d}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_{\text{obj}_{s=+d}}$ ,  $\models^\bullet$ , and  $\models_{-o}^\bullet$ , where  $\models^\bullet$  is an entailment in  $S_{\models}$  that requires the objectification and description restrictions.

*Proof.* For any  $\phi \in \Phi_{\text{obj}_{s=+d}}$  and  $q \in BQ_{\text{obj}_{s=+d}}$ , by Lemma 5.2  $\phi \models^\bullet q$  iff  $\text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$ . By Lemma 5.3,  $\text{obj}_{s=+d}(\phi) \models^\bullet \text{obj}_{s=+d}(\phi, q)$  iff  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$ . Hence  $\phi \models^\bullet q$  iff  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$  holds and by Definition 4.4 the theorem is proved.  $\square$

**Lemma 5.4.** Let  $\models^\bullet$  be an entailment in  $S_{\models}$  that requires the objectification and description restrictions. For any  $\phi \in \Phi_{\text{obj}_{s \neq +d}}$  and  $q \in BQ_{\text{obj}_{s \neq +d}}$ ,  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$  iff  $\text{obj}_{s \neq +d}(\phi) \models_{-o}^\bullet \text{obj}_{s \neq +d}(\phi, q)$ .

*Proof.* This can be proved based on Theorem 5.5 using a similar approach to proof of Lemma 5.1.  $\square$

**Theorem 5.6.**  $\text{obj}_{s \neq +d}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_{\text{obj}_{s \neq +d}}$ ,  $\models^\bullet$ , and  $\models_{-o}^\bullet$ , where  $\models^\bullet$  is an entailment in  $S_{\models}$  that requires the objectification and description restrictions.

*Proof.* For any  $\phi \in \Phi_{\text{obj}_{s \neq +d}}$  and  $q \in BQ_{\text{obj}_{s \neq +d}}$ ,  $\phi \in \Phi_{\text{obj}_{s=+d}}$  and  $q \in BQ_{\text{obj}_{s=+d}}$  hold. By Theorem 5.5, we have  $\phi \models^\bullet q$  iff  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$ . By Lemma 5.4,  $\text{obj}_{s=+d}(\phi) \models_{-o}^\bullet \text{obj}_{s=+d}(\phi, q)$  iff  $\text{obj}_{s \neq +d}(\phi) \models_{-o}^\bullet \text{obj}_{s \neq +d}(\phi, q)$ . Hence  $\phi \models q$  iff  $\text{obj}_{s \neq +d}(\phi) \models_{-o}^\bullet \text{obj}_{s \neq +d}(\phi, q)$ . So, by Definition 4.4, the theorem holds.  $\square$

## 5.4 Skolemization

PSOA RuleML allows the use of *head existentials*, i.e. existentially quantified formulas in rule conclusions. They can be system-provided during objectifi-

cation or user-provided in the KB. These head existentials can be eliminated by *Skolemization* to target the Horn subset of PSOA, which can be mapped to target languages that have only the expressivity of Horn logic, e.g. Pure Prolog. Our employed Skolemization defined in the following is a specialized FOL Skolemization [31] adapted for PSOA RuleML clauses, whose universals are already in prenex form and whose existentials are confined to (conjunctive) conclusions, hence does not require preprocessing of formulas.

**Definition 5.10.** (*Skolemization*) *The Skolemization transformation  $\text{sk}^*(\tau)$  of an existential formula in rule conclusion, which has the form  $\text{Exists } ?X_1 \dots ?X_n (\tau_1)$ , is a variant of  $\tau_1$  where each existential variable  $?X_i$  is replaced with a Skolem function  $\_skolemk_i$  applied to all universally quantified variables from the surrounding clause's quantifier prefix. For each  $?X_i$ , a fresh Skolem function name  $\_skolemk_i$  is chosen from the first name in the sequence  $\_skolem1, \_skolem2, \dots$  that neither occurs elsewhere in the KB nor has been used yet for Skolemization.  $\text{sk}^*$  is extended to a PSOA transformation via  $\text{sk}'$  according to Definition 5.1. Here, the set of reserved names  $\{\_skolem1, \_skolem2, \dots\}$  could be replaced with any other alphabet containing a countably infinite number of reserved constant symbols.  $\diamond$*

Following is a simple example of Skolemization of an OID existential.

**Example 5.2.** (Skolemization example) The following PSOA rule

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

has an existentially quantified formula in the rule conclusion. Skolemization removes the existential quantifier and replaces the variable  $?1$  with  $\_skolem1(?v)$ ,

which is a unary Skolem function `_skolem1` applied to the variable `?v` in the clause's quantifier prefix, yielding

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

Next we prove the semantics preservation of our Skolemization approach.

**Theorem 5.7.** *Let  $\mathcal{P}_{\text{sk}} = (\Phi_K, Q_{\text{sk}})$  be the PSOA sublanguage where  $Q_{\text{sk}}$  is the subset of  $Q_K$  that does not use the Skolem function names `_skolem1`, `_skolem2`, ... For any  $\models^\bullet \in S_{\models}$ , `sk` is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_{\text{sk}}$  and  $\models^\bullet$ .*

*Proof.* We need to show that for any  $\phi \in \mathcal{P}_{\text{sk}}$  and  $q \in BQ_{\text{sk}}$ , `sk` has Properties (1) and (2) in Theorem 5.1 where  $\text{tr}_{\text{PSOA}}^*$ ,  $\text{tr}'_{\text{PSOA}}$ , and  $\text{tr}_{\text{PSOA}}$  become `sk`<sup>\*</sup>, `sk`' , and `sk`, respectively.

- Property (1)

If  $\mathcal{I}$  is a counter-model for  $\phi \models^\bullet q$ , we can extend  $\mathcal{I}$  into  $\mathcal{I}'$  by defining interpretations of Skolem functions as explained in the following. For every head existential  $\tau = \text{Exists } ?X_1 \dots ?X_n (\tau_1)$  and every  $I_V \in M(\text{Var}(\tau), \mathcal{I}.D)$  s.t.  $v(\mathcal{I}, I_V) \models^\bullet \tau$ , there exists  $I_{V^*} \in M(\{?X_1, \dots, ?X_n\}, \mathcal{I}.D)$  s.t.  $v(v(\mathcal{I}, I_V), I_{V^*}) \models^\bullet \tau_1$ . For each `?Xi`, if it is replaced with `_skolemki(...)` in `sk`'( $\tau$ ), then we can define  $\mathcal{I}'.I_{\text{psoa}}(\text{\_skolemk}_i)$  to make  $v(\mathcal{I}', I_V).I(\text{\_skolemk}_i(\dots)) = I_{V^*} (?X_i)$ .

According to this definition,  $v(\mathcal{I}', I_V) \models^\bullet \text{sk}'(\tau)$  and condition (1.a) is fulfilled for every head existential. For any other atomic formula  $\tau$  in  $\phi$  that is not in a head existential, since  $\tau$  does not use the Skolem function names as required by the definitions of `sk` and  $Q_{\text{sk}}$ ,  $\mathcal{I}'$  retains the truth evaluation of  $\tau$  in  $\mathcal{I}$ , thus conditions (1.a) or (1.b) are also fulfilled.

- Property (2)

If  $\mathcal{I}'$  is a counter-model for  $\text{sk}(\phi) \models^\bullet \text{sk}(\phi, q)$ , we can make  $\mathcal{I} = \mathcal{I}'$ . We will show in the following that conditions (2.a) and (2.b) are fulfilled.

- Property (2.a)

If  $\tau$  is an atomic formula, then  $\text{sk}'(\tau) = \tau$  so that condition (2.a) is fulfilled trivially. If  $\tau$  is a head existential  $\text{Exists } ?X_1 \dots ?X_n (\tau_1)$ , and  $\mathfrak{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \text{sk}'(\tau)$ , then  $\mathfrak{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \text{sk}'(\tau)$ . We define  $\mathbf{I}_{V^*} \in \mathbf{M}(\{?X_1, \dots, ?X_n\}, \mathcal{I}.D)$  such that for each variable  $?X_i$ ,  $\mathbf{I}_{V^*}(?X_i) = \mathfrak{v}(\mathcal{I}, \mathbf{I}_V)(\text{\_skolem}_i(\dots))$ . We can see that  $\mathfrak{v}(\mathfrak{v}(\mathcal{I}, \mathbf{I}_V), \mathbf{I}_{V^*}) \models^\bullet \tau_1$ . Hence  $\mathfrak{v}(\mathcal{I}, \mathbf{I}_V) \models^\bullet \tau$  and (2.a) is fulfilled.

- Property (2.b)

For every negative formula  $\tau$ ,  $\text{sk}'(\tau) = \tau$  and condition (2.b) is fulfilled.

By Theorem 5.1,  $\phi \models^\bullet q$  iff  $\text{sk}(\phi) \models^\bullet \text{sk}(\phi, q)$ . So, by Definition 4.4, the theorem holds.  $\square$

## 5.5 Subclass Transformation

In PSOA RuleML, subclass formulas of the form  $\mathbf{c1}\#\#\mathbf{c2}$  are used to express subsumptions between two predicates (acting as classes)  $\mathbf{c1}$  and  $\mathbf{c2}$ . Their semantics is captured by the following two semantic restrictions for the truth evaluation  $TVal_{\mathcal{I}}$  of a semantic structure  $\mathcal{I}$ , which we will call *subclass restrictions*.

$$\text{If } TVal_{\mathcal{I}}(\mathbf{o}\#\mathbf{c1}) = TVal_{\mathcal{I}}(\mathbf{c1}\#\#\mathbf{c2}) = \mathbf{t}, \text{ then } TVal_{\mathcal{I}}(\mathbf{o}\#\mathbf{c2}) = \mathbf{t}. \quad (5.2)$$

$$\text{If } TVal_{\mathcal{I}}(\mathbf{c1}\#\#\mathbf{c2}) = TVal_{\mathcal{I}}(\mathbf{c2}\#\#\mathbf{c3}) = \mathbf{t}, \text{ then } TVal_{\mathcal{I}}(\mathbf{c1}\#\#\mathbf{c3}) = \mathbf{t}. \quad (5.3)$$

Some systems that support subclass formulas, e.g. reasoning systems for order sorted-logic [45], have special optimizations to implement this semantics directly. Meanwhile, other systems adopt an indirect approach to implement this semantics using rules. Here we discuss two different methods. The first method, defined in the following, is to axiomatize the semantic restrictions with two rules and keep the subclass formulas unchanged.

**Definition 5.11.** (*Subclass axiomatization*) *The subclass axiomatization transformation  $\text{sub}_a$  of a KB  $\phi \in \Phi_K$  is defined as  $\text{sub}_a(\phi) = \phi \cup \{AX_{\text{sub1}}, AX_{\text{sub2}}\}$ , where  $AX_{\text{sub1}}$  and  $AX_{\text{sub2}}$  are defined as follows.*

$$AX_{\text{sub1}} : \text{Forall } ?O \ ?C1 \ ?C2 \ (?O\#\?C2 \ :- \ \text{And}(\?O\#\?C1 \ ?C1\#\#\?C2))$$

$$AX_{\text{sub2}} : \text{Forall } ?C1 \ ?C2 \ ?C3 \ (?C1\#\#\?C3 \ :- \ \text{And}(\?C1\#\#\?C2 \ ?C2\#\#\?C3))$$

For queries, the transformation  $\text{sub}_a(\phi, q) = q$ . ◇

The semantics preservation of subclass axiomatization is shown in the following theorem.

**Theorem 5.8.** *For any  $\models^\bullet \in S_{\models}$ ,  $\text{sub}_a$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_K$ ,  $\models^\bullet$ , and  $\models_{-s}^\bullet$ .*

*Proof.* We need to show for every  $\phi \in \Phi_K$  and every  $q \in Q_K$ ,  $\phi \models^\bullet q$  iff  $\text{sub}_a(\phi) \models_{-s}^\bullet \text{sub}_a(\phi, q)$ . This is equivalent to proving  $\phi \not\models^\bullet q$  iff  $\text{sub}_a(\phi) \not\models_{-s}^\bullet q$ . The proof is given in the following:

- $\phi \not\models^\bullet q$
- iff there exists  $\mathcal{I}$  s.t.  $\mathcal{I} \models^\bullet \phi$  and  $\mathcal{I} \not\models^\bullet q$
- iff there exists  $\mathcal{I}$  s.t.  $\mathcal{I} \models_{-s}^\bullet \phi$ ,  $\mathcal{I}$  guarantees the subclass restrictions (5.2) and (5.3), and  $\mathcal{I} \not\models^\bullet q$
- iff there exists  $\mathcal{I}$  s.t.  $\mathcal{I} \models_{-s}^\bullet \phi \cup \{AX_{sub1}, AX_{sub2}\}$  and  $\mathcal{I} \not\models_{-s}^\bullet q$
- iff  $\phi \cup \{AX_{sub1}, AX_{sub2}\} \not\models_{-s}^\bullet q$
- iff  $\text{sub}_a(\phi) \not\models_{-s}^\bullet \text{sub}_a(\phi, q)$

So the theorem holds. □

The second method, defined in the following, is to rewrite each subclass formula into a rule. This method can be applied when subclass formulas are only used as top-level KB formulas for expressing class hierarchies but not in other places, e.g. nested in a rule or used in queries.

**Definition 5.12.** (*Subclass rewriting*) *The subclass rewriting transformation  $\text{sub}_r^*(\tau)$  for a subclass formula  $\tau$  having the form  $c1\#\#c2$  results in the following rule*

$$\text{forall } ?X \text{ (?X\#c2 :- ?X\#c1)} \quad (5.4)$$

$\text{sub}_r^*$  is extended to a PSOA transformation  $\text{sub}_r$  via  $\text{sub}_r'$  according to Definition 5.1. ◇

In the following theorem we show the semantics preservation of subclass rewriting for a PSOA sublanguage where subclass formulas are not used in

rules or in queries (i.e. schema-level information can only be asserted as top-level KB formulas).

**Theorem 5.9.** *Let  $\mathcal{P}_{\text{sub}_r} = (\Phi_{\text{sub}_r}, Q_{\text{sub}_r})$  be the PSOA sublanguage such that (1)  $\Phi_{\text{sub}_r}$  is the set of KBs where subclass formulas are only used as top-level KB formulas; (2)  $Q_{\text{sub}_r}$  is the set of queries that do not use subclass formulas. For any  $\models^\bullet \in S_{\models}$ ,  $\text{sub}_r$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_{\text{sub}_r}$ ,  $\models^\bullet$ , and  $\models_{-s}^\bullet$ .*

*Proof.* We need to show for every KB  $\phi \in \Phi_{\text{sub}_r}$  and boolean query  $q \in Q_{\text{sub}_r}$ ,  $\phi \models^\bullet q$  iff  $\text{sub}_r(\phi) \models_{-s}^\bullet \text{sub}_r(\phi, q)$ . We first prove  $\phi \models^\bullet q$  iff  $\text{sub}_r(\phi) \models^\bullet \text{sub}_r(\phi, q)$  by showing  $\text{sub}_r$  has Properties (1) and (2) in Theorem 5.1, where  $\text{tr}_{\text{PSOA}}^*$  is  $\text{sub}_r^*$ ,  $\text{tr}'_{\text{PSOA}}$  is  $\text{sub}'_r$ , and  $\text{tr}_{\text{PSOA}}$  is  $\text{sub}_r$ . We need to prove the properties are fulfilled for  $\tau$  being a subclass formula or any other atomic formula.

- Property (1)

If there exists a counter-model  $\mathcal{I}$  for  $\phi \models q$ , then we can choose  $\mathcal{I}' = \mathcal{I}$ . For every  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}.D)$ ,

- if  $\tau$  is a subclass formula of the form  $\mathbf{c1}\#\#\mathbf{c2}$ , according to the assumption  $\tau$  can only occur positively as a top-level KB formula, so we only need to show that condition (1.a) is fulfilled. If  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models \tau$ , then  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\tau) = \mathbf{t}$  and  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V)$  guarantees subclass restriction (5.2). Hence for any  $\mathbf{o}$  such that  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\mathbf{o}\#\mathbf{c1}) = \mathbf{t}$ ,  $TVal_{\mathbf{v}(\mathcal{I}, \mathbf{I}_V)}(\mathbf{o}\#\mathbf{c2}) = \mathbf{t}$  holds. Thus  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models \text{sub}'_r(\tau)$  and condition (1.a) is fulfilled.
- if  $\tau$  is any other atomic formula, then  $\text{sub}'_r(\tau) = \tau$ , so conditions (1.a) and (1.b) are fulfilled trivially.

- Property (2)

If there exists a counter-model  $\mathcal{I}^*$  for  $\mathbf{sub}_r(\phi) \models \mathbf{sub}_r(\phi, q)$ , we can extend it to a semantic structure  $\mathcal{I}'$  by defining  $TVal_{\mathcal{I}'}(\mathbf{c1}\#\#\mathbf{c2}) = \mathbf{t}$  as long as  $TVal_{\mathcal{I}'}(\mathbf{sub}'_r(\mathbf{c1}\#\#\mathbf{c2})) = \mathbf{t}$  holds. It is easy to verify that  $\mathcal{I}'$  guarantees subclass restrictions. And since  $\mathcal{I}'$  changes only the truth evaluation of subclass formulas, we still have  $TVal_{\mathcal{I}'}(\mathbf{sub}'_r(\phi)) = \mathbf{t}$  and  $TVal_{\mathcal{I}'}(\mathbf{sub}'_r(q)) = \mathbf{f}$ . Hence  $\mathcal{I}'$  is still a counter-model for  $\mathbf{sub}'_r(\phi) \models \mathbf{sub}'_r(q)$ . Next we define  $\mathcal{I} = \mathcal{I}'$  and show that conditions (2.a) and (2.b) are fulfilled.

For every  $I_V \in \mathbf{M}(\mathbf{Var}(\tau), \mathcal{I}.D)$ ,

- if  $\tau$  is a subclass formula of the form  $\mathbf{c1}\#\#\mathbf{c2}$ , it can only occur positively and the above construction of  $\mathcal{I}'$  ensures that condition (2.a) is fulfilled.
- if  $\tau$  is any other atomic formula, then  $\mathbf{sub}_r(\tau) = \tau$ , so conditions (2.a) and (2.b) are fulfilled.

Hence  $\phi \models^\bullet q$  iff  $\mathbf{sub}_r(\phi) \models_{\cdot_s}^\bullet \mathbf{sub}_r(\phi, q)$  holds by Theorem 5.1.

Next we need to prove  $\mathbf{sub}_r(\phi) \models^\bullet \mathbf{sub}_r(\phi, q)$  iff  $\mathbf{sub}_r(\phi) \models_{\cdot_s}^\bullet \mathbf{sub}_r(\phi, q)$  by showing the condition of Theorem 5.2 is fulfilled for  $\phi'$  being  $\mathbf{sub}_r(\phi)$ ,  $q'$  being  $\mathbf{sub}_r(\phi, q)$ , and  $\models^{\bullet\bullet}$  being  $\models_{\cdot_s}^\bullet$ . If  $\mathcal{I}''$  is a counter-model for  $\mathbf{sub}_r(\phi) \models_{\cdot_s}^\bullet \mathbf{sub}_r(\phi, q)$ , we can modify it into  $\mathcal{I}'$  by defining  $TVal_{\mathcal{I}'}(\mathbf{c1}\#\#\mathbf{c2}) = \mathbf{f}$  for any subclass formulas, so that  $\mathcal{I}'$  guarantees subclass restriction. This change would not affect the truth evaluation for  $\mathbf{sub}_r(\phi)$  and  $\mathbf{sub}_r(\phi, q)$  since they do not have subclass formulas. Thus  $\mathcal{I}'$  is a counter-model for  $\mathbf{sub}_r(\phi) \models^\bullet \mathbf{sub}_r(\phi, q)$  and the condition of Theorem 5.2 is fulfilled.

Thus we have  $\phi \models^\bullet q$  iff  $\mathbf{sub}_r(\phi) \models_{\cdot_s}^\bullet \mathbf{sub}_r(\phi, q)$  and the theorem is proved. □

In our PSOATransRun implementation (cf. Chapter 9), we focus on the sublanguage  $\mathcal{P}_{\text{sub}_r}$  and choose the second method since it does not require the axiomatization rules to be imported, prelude-like, to every KB.

## 5.6 Description

In PSOA RuleML, the truth evaluation of an atom in a model  $\mathcal{I}$  must guarantee the revised description restriction in Section 3.4. According to that definition, we define the description transformation as follows.

**Definition 5.13.** (*Description*) *The description transformation  $\text{desb}^*(\tau)$  for an oidful atom  $\tau$ , which has the general form*

$$\begin{aligned} & \text{of} (+ [\mathbf{t}_{1,1}^+ \dots \mathbf{t}_{1,n_1^+}] \dots + [\mathbf{t}_{m^+,1}^+ \dots \mathbf{t}_{m^+,n_{m^+}^+}] \\ & \quad - [\mathbf{t}_{1,1}^- \dots \mathbf{t}_{1,n_1^-}] \dots - [\mathbf{t}_{m^-,1}^- \dots \mathbf{t}_{m^-,n_{m^-}^-}] \\ & \quad \mathbf{p}_1^+ \rightarrow \mathbf{v}_1^+ \dots \mathbf{p}_{k^+}^+ \rightarrow \mathbf{v}_{k^+}^+ \\ & \quad \mathbf{p}_1^- \rightarrow \mathbf{v}_1^- \dots \mathbf{p}_{k^-}^- \rightarrow \mathbf{v}_{k^-}^- ) \end{aligned}$$

*containing  $m^+$  dependent tuples,  $m^-$  independent tuples,  $k^+$  dependent slots,  $k^-$  dependent slots is a conjunction of  $1 + m^+ + m^- + k^+ + k^-$  subformulas, including*

one membership atom,  $m^+ + m^-$  single-tuple atoms, and  $k^+ + k^-$  single-slot atoms:

$$\begin{aligned}
& \text{And}(\text{o}\#f \\
& \quad \text{o}\#f(+[\mathbf{t}_{1,1}^+ \dots \mathbf{t}_{1,n_1}^+]) \dots \text{o}\#f(+[\mathbf{t}_{m^+,1}^+ \dots \mathbf{t}_{m^+,n_{m^+}}^+]) \\
& \quad \text{o}\#\text{Top}(-[\mathbf{t}_{m^-,1}^- \dots \mathbf{t}_{m^-,n_{m^-}}^-]) \dots \text{o}\#\text{Top}(-[\mathbf{t}_{m^-,1}^- \dots \mathbf{t}_{m^-,n_{m^-}}^-]) \quad (5.5) \\
& \quad \text{o}\#f(\mathbf{p}_1^+ \rightarrow \mathbf{v}_1^+) \dots \text{o}\#f(\mathbf{p}_{k^+}^+ \rightarrow \mathbf{v}_{k^+}^+) \\
& \quad \text{o}\#\text{Top}(\mathbf{p}_1^- \rightarrow \mathbf{v}_1^-) \dots \text{o}\#\text{Top}(\mathbf{p}_{k^-}^- \rightarrow \mathbf{v}_{k^-}^-))
\end{aligned}$$

For the special case where  $f = \text{Top}$  but  $m^+ + m^- + k^+ + k^- > 0$ , the tautological membership  $\text{o}\#\text{Top}$  is omitted from the conjunction for efficiency.<sup>34</sup>

$\text{desb}^*$  can be extended to a PSOA transformation  $\text{desb}$  via  $\text{desb}'$  based on Definition 5.1. ◇

In the following we discuss the semantics preservation of  $\text{desb}$ .

**Theorem 5.10.** *For any  $\models^\bullet$  in  $S_{\models}$  that requires the description restriction and does not require the objectification restriction,  $\text{desb}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_K$ ,  $\models^\bullet$ , and  $\models_{-d}^\bullet$ .*

*Proof.* We need to show for every  $\phi \in \Phi_K$  and  $q \in Q_K$ ,  $\phi \models^\bullet q$  iff  $\text{desb}(\phi) \models_{-d}^\bullet \text{desb}(\phi, q)$ . We first prove  $\phi \models^\bullet q$  iff  $\text{desb}(\phi) \models^\bullet \text{desb}(\phi, q)$  by showing  $\text{desb}$  fulfills the condition of Corollary 5.1 where  $\text{tr}_{\text{PSOA}}^*$  is  $\text{desb}^*$ ,  $\text{tr}'_{\text{PSOA}}$  is  $\text{desb}'$ , and  $\text{tr}_{\text{PSOA}}$  is  $\text{desb}$ . For any semantic structure  $\mathcal{I}^*$ , if  $\mathcal{I}^*$  conforms to the description restriction required by  $\models^\bullet$ , then  $TVal_{\mathcal{I}^*}(\tau) = \mathbf{t}$  iff  $TVal_{\mathcal{I}^*}(\text{desb}(\tau)) = \mathbf{t}$  holds for every atom  $\tau$  changed by  $\text{desb}$ . Hence by Corollary 5.1,  $\phi \models^\bullet q$  iff  $\text{desb}(\phi) \models^\bullet \text{desb}(\phi, q)$  holds.

---

<sup>34</sup>For  $m^+ + m^- + k^+ + k^- = 0$ , the descriptorless  $\text{o}\#\text{Top}$  does not undergo description.

Next we prove  $\text{desb}(\phi) \models^\bullet \text{desb}(\phi, q)$  iff  $\text{desb}(\phi) \models_{-d}^\bullet \text{desb}(\phi, q)$  by showing the condition of Theorem 5.2 is fulfilled where  $\phi'$  is  $\text{desb}(\phi)$ ,  $q'$  is  $\text{desb}(\phi, q)$ ,  $\models^{\bullet\bullet}$  is  $\models_{-d}^\bullet$ . Let  $\mathcal{I}''$  be a counter-model for  $\text{desb}(\phi) \models_{-d}^\bullet \text{desb}(\phi, q)$ . We modify  $\mathcal{I}''$  into  $\mathcal{I}'$  by redefining the truth evaluation  $TVal_{\mathcal{I}''}$  for every oidful atom  $\tau$  so that  $TVal_{\mathcal{I}'}(\tau) = TVal_{\mathcal{I}''}(\text{desb}(\tau))$ . This makes  $\mathcal{I}'$  guarantee the description restriction. Since  $\text{desb}(\phi)$  and  $\text{desb}(\phi, q)$  contain only simple atoms, the redefinition would not affect their truth evaluation, thus we have  $TVal_{\mathcal{I}'}(\text{desb}(\phi)) = TVal_{\mathcal{I}''}(\text{desb}(\phi)) = \mathbf{t}$  and  $TVal_{\mathcal{I}'}(\text{desb}(\phi, q)) = TVal_{\mathcal{I}''}(\text{desb}(\phi, q)) = \mathbf{f}$ . Also, since  $\models^\bullet$  does not require the objectification restriction, the truth evaluation of oidless atoms can be kept unchanged in  $\mathcal{I}'$ . Hence  $\mathcal{I}' \models^\bullet \text{desb}(\phi)$  and  $\mathcal{I}' \not\models^\bullet \text{desb}(\phi, q)$  holds. Thus  $\mathcal{I}'$  is a counter-model of  $\text{desb}(\phi) \models^\bullet \text{desb}(\phi, q)$  and by Theorem 5.2  $\text{desb}(\phi) \models^\bullet \text{desb}(\phi, q)$  iff  $\text{desb}(\phi) \models_{-d}^\bullet \text{desb}(\phi, q)$  holds.

Thus  $\phi \models^\bullet q$  iff  $\text{desb}(\phi) \models_{-d}^\bullet \text{desb}(\phi, q)$  holds and the theorem is proved.  $\square$

This transformation is central to the normalization in both PSOA2TPTP (see Chapter 7) and PSOA2Prolog (see Chapter 8).

## 5.7 Flattening Embedded External Expressions

An expression in PSOA RuleML can use a constructor function with no definition, a user-defined function specified by conclusion equalities in the KB, or an externally defined function such as an arithmetic built-in. Since in Definition 5.3 we have restricted the use of equalities to rule conditions and queries, user-defined functions need not be dealt with in our transformations. In particular, flattening is needed only for external expressions: it creates a conjunction extracting each embedded external expression as a separate equality, defined

in the following.

**Definition 5.14.** *The flattening transformation  $\text{flt}'(\alpha)$  for an atomic formula  $\alpha$  is defined as follows, where  $\text{genVar}(t)$  generates a fresh variable for each expression  $t$  by choosing the first variable in ?1, ?2, ... that does not occur in the enclosing rule. The  $\text{genVar}(t)$  calls in the definitions of  $\text{Eqs}(t)$  and  $\text{Retain}_{\text{flt}}(t)$  returns the same variable for the same external expression  $t$ .*

$$\text{flt}'(\alpha) ::= \begin{cases} \alpha & m = 1 \\ \text{And}(\sigma_1 \dots \sigma_m) & m > 1, \alpha \text{ is in the KB} \\ \text{Exists } ?X_1 \dots ?X_n (\text{And}(\sigma_1 \dots \sigma_m)) & m > 1, \alpha \text{ is in the query} \end{cases}$$

$$(\{\sigma_1, \dots, \sigma_m\} = \cup_{t \in \text{Parts}(\alpha)} \text{Eqs}(t) \cup \{\text{Trim}_{\text{flt}}(\alpha)\},$$

$?X_1, \dots, ?X_n$  are the variables generated by  $\text{genVar}$  for  $\alpha$ )

$$\text{Eqs}(t) ::= \begin{cases} \emptyset & t \text{ is a simple term} \\ \cup_{s \in \text{Parts}(t)} \text{Eqs}(s) & t \text{ is a non-external expression} \\ \cup_{s \in \text{Parts}(t)} \text{Eqs}(s) \cup \{\text{Retain}_{\text{flt}}(t) = \text{Trim}_{\text{flt}}(t)\} & t \text{ is an external expression} \end{cases}$$

$$\text{Trim}_{\text{flt}}(t) ::= \text{Construct obtained by replacing every } s \in \text{Parts}(t) \text{ in } t \text{ with } \text{Retain}_{\text{flt}}(s)$$

$$\text{Retain}_{\text{flt}}(t) ::= \begin{cases} t & t \text{ is a simple term} \\ \text{Trim}_{\text{flt}}(t) & t \text{ is a non-external expression} \\ \text{genVar}(t) & t \text{ is an external expression} \end{cases}$$

For a non-atomic formula  $\tau$ ,

- if  $\tau$  is not a rule,  $\text{flt}'(\tau)$  is obtained by recursively applying  $\text{flt}'$  to each subformula of  $\tau$ , while keeping the surrounding formula structure unchanged.

- if  $\tau$  is a rule of the form  $\text{Forall } ?V_1 \dots ?V_n (\tau_1 :- \tau_2)$ ,  $\text{flt}'(\tau)$  is

$$\text{Forall } ?V_1 \dots ?V_n ?X_1 \dots ?X_k (\text{flt}'(\tau_1) :- \text{flt}'(\tau_2)),$$

where  $?X_1, \dots, ?X_k$  are the fresh variables generated during the computation of  $\text{flt}'(\tau_2)$ .

The transformation  $\text{flt}$  is defined via  $\text{flt}'$  according to Definition 4.2.  $\diamond$

The definition of the flattening transformation is similar to the definition of unnesting. For an atomic formula  $\alpha$ ,  $\text{flt}(\alpha)$  is a conjunction of formulas  $\sigma_i$  without embedded external expressions. Each  $\sigma_i$  is a trimmed version of the top-level formula  $\alpha$  or an equality in  $\text{Eqs}$ . The set  $\text{Eqs}$  contains equalities of the form  $\text{Retain}_{\text{flt}}(t) = \text{Trim}_{\text{flt}}(t)$ , where  $\text{Retain}_{\text{flt}}(t)$  generates a variable for  $t$  using  $\text{genVar}$  and the equality binds it to the evaluation result of  $\text{Trim}_{\text{flt}}(t)$ .  $\text{Eqs}$  is constructed by recursively traversing through each component  $s \in \text{Parts}(t)$ , collecting  $\text{Eqs}(s)$  into  $\text{Eqs}(t)$ , and then adding  $\text{Retain}_{\text{flt}}(t) = \text{Trim}_{\text{flt}}(t)$  to  $\text{Eqs}(t)$  if  $t$  is an external expression. The transformation  $\text{Trim}_{\text{flt}}(t)$  splits off every embedded expression from  $t$  and leaves behind a generated variable bound to the evaluation result of the expression. Hence the evaluation of  $\text{Trim}_{\text{flt}}(t)$  is the same as the evaluation result of  $t$ , which will be shown later in the proof of Theorem 5.11.

The semantic preservation of  $\text{flt}$  is shown in the following theorem.

**Theorem 5.11.** *For any  $\models^\bullet \in S_{\models}$ ,  $\text{flt}$  is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_K$  and  $\models^\bullet$ .*

*Proof.* We need to show that for every  $\phi \in \Phi_K$  and  $q \in Q_K$ ,  $\phi \models q$  iff  $\text{flt}(\phi) \models \text{flt}(\phi, q)$  holds. This is equivalent to proving  $\phi \models q$  iff  $\text{flt}'(\phi) \models \text{flt}'(q)$ .

For this proof we define an auxiliary transformation  $\text{flt}''$  which is modified from  $\text{flt}'$  by transforming atomic formulas in KBs in the same way as in queries using existentials, and handle rules in the same way as non-rules. Since  $\text{flt}'(\phi)$  can be obtained from  $\text{flt}''(\phi)$  by changing all existential variables in rule conditions into universal variables on the top-level of the rule,  $\text{flt}'(\phi)$  and  $\text{flt}''(\phi)$  are equivalent. Since  $\text{flt}'(q) = \text{flt}''(q)$ , we have  $\text{flt}'(\phi) \models \text{flt}'(q)$  iff  $\text{flt}''(\phi) \models \text{flt}''(q)$ .

Next we will show that  $\phi \models q$  iff  $\text{flt}''(\phi) \models \text{flt}''(q)$  by showing the condition of Corollary 5.1 is fulfilled for  $\models^\bullet$  being  $\models$ ,  $\text{tr}_{\text{PSOA}}$  being  $\text{flt}''$ , and  $\text{tr}_{\text{PSOA}}^*$  being the restricted version of  $\text{flt}''$  applying only to an atomic formula  $\alpha$  that uses an embedded external expression. We write  $\text{AE}(\alpha)$  for all external expressions in  $\alpha$ . For any  $\mathcal{I}^*$  that conforms to the semantic restrictions required by  $\models^\bullet$ , if  $TVal_{\mathcal{I}^*}(\text{flt}''(\alpha)) = \mathbf{t}$ , then there exists  $\mathbf{I}_V \in \mathbf{M}(\{?X_1, \dots, ?X_n\}, \mathcal{I}^*.D)$ ,  $\mathcal{I}^{**} = \mathbf{v}(\mathcal{I}^*, \mathbf{I}_V)$  s.t.

$$\begin{aligned}
& TVal_{\mathcal{I}^{**}}(\text{And}(\sigma_1 \dots \sigma_m)) = \mathbf{t} \\
\text{iff } & TVal_{\mathcal{I}^{**}}(\sigma_1) = \dots = TVal_{\mathcal{I}^{**}}(\sigma_m) = \mathbf{t} \\
\text{iff } & TVal_{\mathcal{I}^{**}}(\text{Trim}_{\text{flt}}(\alpha)) = \mathbf{t} \text{ and } \forall t \in \text{AE}(\alpha), TVal_{\mathcal{I}^{**}}(\text{Retain}_{\text{flt}}(t) = \text{Trim}_{\text{flt}}(t)) = \mathbf{t} \\
\text{iff } & TVal_{\mathcal{I}^{**}}(\text{Trim}_{\text{flt}}(\alpha)) = \mathbf{t} \text{ and } \forall t \in \text{AE}(\alpha), \mathcal{I}^{**}(\text{Retain}_{\text{flt}}(t)) = \mathcal{I}^{**}(\text{Trim}_{\text{flt}}(t)) \\
\text{iff } & TVal_{\mathcal{I}^{**}}(\text{Trim}_{\text{flt}}(\alpha)) = \mathbf{t} \text{ and } \forall t \in \text{AE}(\alpha), \mathcal{I}^{**}(\text{Retain}_{\text{flt}}(t)) = \mathcal{I}^{**}(t) \\
& \text{(by the definition of } \text{Trim}_{\text{flt}}(t))
\end{aligned}$$

By the definition of  $\text{Trim}_{\text{flt}}(\alpha)$ ,  $TVal_{\mathcal{I}^{**}}(\alpha) = \mathbf{t}$  holds. Since  $\{?X_1, \dots, ?X_n\}$  are fresh variables that do not occur in  $\alpha$ ,  $TVal_{\mathcal{I}^*}(\alpha) = \mathbf{t}$  holds.

Conversely, if  $TVal_{\mathcal{I}^*}(\alpha) = \mathbf{t}$ , then we can choose  $\mathbf{I}_V$  s.t.  $\mathbf{I}_V(\text{Retain}_{\text{flt}}(t)) = \mathcal{I}^*(t)$  and  $\mathcal{I}^{**} = \mathbf{v}(\mathcal{I}^*, \mathbf{I}_V)$ . So  $\mathcal{I}^{**}(\text{Retain}_{\text{flt}}(t)) = \mathcal{I}^{**}(t)$ . By the definition of  $\text{Trim}_{\text{flt}}(\alpha)$ ,  $TVal_{\mathcal{I}^{**}}(\text{Trim}_{\text{flt}}(\alpha)) = \mathbf{t}$ . Hence according to the ‘if’ part of the

above ‘iff’ derivation sequence we have  $TVal_{\mathcal{T}^{**}}(\text{And}(\sigma_1 \dots \sigma_m)) = \mathbf{t}$ . Thus  $TVal_{\mathcal{T}^*}(\text{flt}''(\alpha)) = \mathbf{t}$ .

So  $TVal_{\mathcal{T}^*}(\alpha) = \mathbf{t}$  iff  $TVal_{\mathcal{T}^*}(\text{flt}''(\alpha)) = \mathbf{t}$  and the condition of Corollary 5.1 is fulfilled. Thus  $\phi \models q$  iff  $\text{flt}''(\phi) \models \text{flt}''(q)$ . Hence  $\phi \models q$  iff  $\text{flt}'(\phi) \models \text{flt}'(q)$  holds and the theorem is proved.  $\square$

Because of the similarity of unnesting and flattening transformations, they could also be merged into one single transformation.

## 5.8 Splitting Rules with Conjunctive Conclusions

In order to target rule languages that do not support rules that use a conjunction as the conclusion, e.g. ISO Prolog, we need to split such rules into multiple rules using the transformation **ccsp** defined in the following, which is part of the Lloyd-Topor transformations [57]. In the definition, we write  $\text{conj}(\tau)$  for all top-level and nested conjuncts in a formula  $\tau$ , defined as

$$\text{conj}(\tau) ::= \begin{cases} \bigcup_{i=1}^n \text{conj}(\tau_i) & \tau = \text{And}(\tau_1 \dots \tau_n) \\ \{\tau\} & \tau \text{ is of any other form} \end{cases} \quad (5.6)$$

Note that if  $\tau$  has an existential conjunct,  $\text{conj}(\tau)$  would keep it unchanged without looking further into formulas embedded in the existential quantifier. Hence conjunctions embedded in an existential quantifier will not be extracted.

**Definition 5.15.** (*Conjunctive conclusion splitting*) Given a KB  $\phi$ , the conjunctive conclusion splitting transformation  $\text{ccsp}^*(\phi)$  of  $\phi$  is obtained by re-

placing every clause  $\tau$ , which has the general form

$$\text{Forall } ?X_1 \dots ?X_n (\tau_h :- \tau_p) \tag{5.7}$$

with  $k$  rules

$$\text{Forall } ?X_1 \dots ?X_n (\tau_h^i :- \tau_p) \quad i = 1, \dots, k, \{\tau_h^1, \dots, \tau_h^k\} = \text{conj}(\tau_h)$$

$\text{ccsp}^*$  is extended to  $\text{ccsp}$  via  $\text{ccsp}'$  based on Definition 5.1. ◇

According to the definition, if the rule conclusion  $\tau_h$  is not a conjunction, then  $\text{conj}(\tau_h) = \{\tau_h\}$  and the rule is kept unchanged. Otherwise, the rule is split into  $k$  rules with each (top-level or nested) conjunct becoming the conclusion of one rule, and with the condition and the quantification copied unchanged.

The semantics preservation of  $\text{ccsp}$  is shown in the following.

**Lemma 5.5.** *Let  $\tau$  be a rule,  $\tau_i, i = 1, \dots, k$  be the split rules, and  $\mathcal{I}$  be a semantic structure.  $TVal_{\mathcal{I}}(\tau) = \mathbf{t}$  holds iff  $TVal_{\mathcal{I}}(\tau_i) = \mathbf{t}, i = 1, \dots, k$ .*

*Proof.* Let  $\tau$  have the form (5.7). If  $\tau_h$  is not a conjunction then  $k = 1, \tau_1 = \tau$ , and the lemma holds trivially. Otherwise, for every  $I_V \in M(\{?X_1, \dots, ?X_n\}, \mathcal{I}.D)$ ,

we have

$$\begin{aligned}
& TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h :- \tau_p) = \mathbf{t} \\
\text{iff } & TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h) = \mathbf{t} \text{ or } TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_p) = \mathbf{f} \\
\text{iff } & TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\text{And}(\tau_h^1 \dots \tau_h^k)) = \mathbf{t} \text{ or } TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_p) = \mathbf{f} \\
\text{iff } & TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^1) = \dots = TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^k) = \mathbf{t} \text{ or } TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_p) = \mathbf{f} \\
\text{iff } & (TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^1) = \mathbf{t} \text{ or } TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_p) = \mathbf{f}) \text{ and } \dots \text{ and} \\
& (TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^k) = \mathbf{t} \text{ or } TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_p) = \mathbf{f}) \\
\text{iff } & TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^1 :- \tau_p) = \dots = TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^k :- \tau_p) = \mathbf{t}
\end{aligned}$$

Hence

$$\begin{aligned}
& TVal_{\mathcal{I}}(\tau) = \mathbf{t} \\
\text{iff } & \forall \mathcal{I}_{\mathcal{V}} \in \mathcal{M}(\{\?X_1, \dots, ?X_n\}, \mathcal{I}.D), TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h :- \tau_p) = \mathbf{t} \\
\text{iff } & \forall \mathcal{I}_{\mathcal{V}} \in \mathcal{M}(\{\?X_1, \dots, ?X_n\}, \mathcal{I}.D), TVal_{\mathcal{V}(\mathcal{I}, \mathcal{I}_{\mathcal{V}})}(\tau_h^i :- \tau_p) = \mathbf{t}, i = 1, \dots, k \\
\text{iff } & TVal_{\mathcal{I}}(\tau_i) = \mathbf{t}, i = 1, \dots, k
\end{aligned}$$

So the lemma holds.  $\square$

**Theorem 5.12.** *For any  $\models^\bullet \in S_{\models}$ , ccsp is a semantics-preserving PSOA transformation with respect to  $\mathcal{P}_K$  and  $\models^\bullet$ .*

*Proof.* We need to show for every  $\phi \in \Phi_K$  and  $q \in Q_K$ ,  $\phi \models q$  iff  $\text{ccsp}(\phi) \models \text{ccsp}(\phi, q)$ . We first show that for every  $\mathcal{I}$ ,  $\mathcal{I} \models \phi$  iff  $\mathcal{I} \models \text{ccsp}(\phi)$ . If  $\mathcal{I} \models \phi$ , then  $\mathcal{I}$  guarantees all semantic restrictions and for every top-level formula  $\tau$  in  $\phi$ ,  $TVal_{\mathcal{I}}(\tau) = \mathbf{t}$ . Let  $\tau'$  be a top-level formula in  $\text{ccsp}(\phi)$ . If  $\tau'$  is not a

rule, it occurs in  $\phi$  and  $TVal_{\mathcal{I}}(\tau') = \mathbf{t}$  holds. Otherwise there exists  $\tau$  s.t.  $\tau'$  is split from  $\tau$ . By Lemma 5.5  $TVal_{\mathcal{I}}(\tau') = \mathbf{t}$  also holds. Thus  $\mathcal{I} \models \text{ccsp}(\phi)$ .

The converse direction can also be proved in similarly by showing  $\mathcal{I} \models \tau$  holds for every  $\tau$  in  $\phi$ .

Thus

$$\phi \models q$$

iff for every  $\mathcal{I}$ , if  $\mathcal{I} \models \phi$  then  $\mathcal{I} \models q$

iff for every  $\mathcal{I}$ , if  $\mathcal{I} \models \text{ccsp}(\phi)$  then  $\mathcal{I} \models q$

iff for every  $\mathcal{I}$ , if  $\mathcal{I} \models \text{ccsp}(\phi)$  then  $\mathcal{I} \models \text{ccsp}(q)$  (by Definition 5.15,  
 $\text{ccsp}(q) = q$  holds)

iff  $\text{ccsp}(\phi) \models \text{ccsp}(q)$

So the theorem is proved. □

## 5.9 Local-Constant Renaming and KB Merging

In some applications, knowledge may be distributed over multiple KBs, so that a preprocessing phase is needed to merge them into one KB for further processing. This procedure is based on the KB merging transformation, which itself is built on of the renaming and the adjoining transformations.

**Definition 5.16.** (*Renaming*) Given a KB  $\phi$ , the renaming transformation  $\text{ren}(\phi)$  replaces each local constant with a fresh constant name generated by a local constant generator  $\text{gencon}()$ . The parameterless  $\text{gencon}()$  on every invocation returns a fresh constant that has not been used in an earlier such

generation. ◇

**Definition 5.17.** (*Adjoining*) Given  $m$  KBs  $\phi_1, \dots, \phi_m$ , the  $m$ -ary adjoining transformation  $\text{adj}(\phi_1, \dots, \phi_m)$  collects all clauses of  $\phi_1 \dots \phi_m$  into a single KB as the output.<sup>35</sup> ◇

**Definition 5.18.** (*KB merging*) Given  $1+n$  KBs  $\phi_0, \phi_1, \dots, \phi_n$ , the  $(1+n)$ -ary KB merging transformation  $\text{merge}(\phi_0, \phi_1, \dots, \phi_n)$  is defined as  $\text{adj}(\phi_0, \text{ren}(\phi_1), \dots, \text{ren}(\phi_n))$ , where all renaming transformations utilize the same local constant generator  $\text{gencon}()$  that generates names not occurring in  $\phi_0$ . ◇

The `merge` transformation is asymmetric in that the renaming transformation is not applied to the argument  $\phi_0$ , so that queries can still use the original local constant names in  $\phi_0$  after merging. When a symmetric transformation for KBs is needed, one can use the empty KB for  $\phi_0$  and let  $\phi_1, \dots, \phi_n$  be the input KBs so that all of them will undergo the renaming transformation.

The generator `gencon()` is implemented simply by returning the first name in `_1, _2, ...` that has not already been used in  $\phi_0$  or in the generation.

The main use of the merging transformation is in the importing transformation of PSOA documents, which combines all directly or indirectly imported KBs specified by `Import` declarations into the a document. The procedure is specified as pseudocode in Algorithm 5.1.

In the algorithm, the set `ImportedIRIs` stores the IRIs of all dereferenced KBs to avoid the same KB being dereferenced again. Thus, the algorithm can also deal with cyclic imports.

---

<sup>35</sup>We do not specify the order of clauses in the new KB since the model-theoretic semantics does not change for different orders of KB clauses.

---

**Algorithm 5.1:** Importing transformation

---

**Input:** A PSOA KB  $\phi_0$   
**Output:** A PSOA KB with all imported KBs merged

- 1 Initialize ToProcess with IRIs of all imported KBs in  $\phi_0$ ;
- 2 Initialize ImportedIRIs and ImportedKBs with the empty set;
- 3 **while** ToProcess *is not empty* **do**
- 4     Select an arbitrary IRI  $iri$  from ToProcess and remove  $iri$  from ToProcess;
- 5     **if**  $iri$  *is not in* ImportedIRIs **then**
- 6         Add  $iri$  to ImportedIRIs;
- 7         Dereference  $iri$  to the KB  $\phi$ ;
- 8         **if**  $\phi$  *is a KB in* PSOA **then**
- 9             Add  $\phi$  to ImportedKBs;
- 10            Parse  $\phi$  and add the IRIs of its imported KBs into ToProcess;
- 11         **else**
- 12             Translate  $\phi$  into a PSOA KB  $\phi'$  and add  $\phi'$  to ImportedKBs;
- 13 **return** merge( $\phi_0, \phi_1, \dots, \phi_n$ ), where  $\{\phi_1, \dots, \phi_n\} = \text{ImportedKBs}$ ;

---

## 5.10 Composition of PSOA Transformations

The transformation steps defined in the previous sections can be sequentially composed in different ways based on Definition 4.3. One desired property of a transformation composition is that every transformation step is applied only once. In order to construct such transformation compositions, we need to constrain the order of applying the steps. In particular, we need to find the transformation pairs  $\langle \text{tr}_{\text{PSOA}}^1, \text{tr}_{\text{PSOA}}^2 \rangle$  such that  $\text{tr}_{\text{PSOA}}^2$  must be applied after  $\text{tr}_{\text{PSOA}}^1$  if they are used in the same transformation composition, defined in the following.

**Definition 5.19.** *We say a transformation  $\text{tr}_{\text{PSOA}}^2$  must be applied after  $\text{tr}_{\text{PSOA}}^1$  in a composition, written as  $\text{tr}_{\text{PSOA}}^2 \leftarrow \text{tr}_{\text{PSOA}}^1$ , if and only if  $\text{tr}_{\text{PSOA}}^1 \circ \text{tr}_{\text{PSOA}}^2 \circ \text{tr}_{\text{PSOA}}^1 = \text{tr}_{\text{PSOA}}^2 \circ \text{tr}_{\text{PSOA}}^1 \circ \text{tr}_{\text{PSOA}}^2 \neq \text{tr}_{\text{PSOA}}^1 \circ \text{tr}_{\text{PSOA}}^2$ .*  $\diamond$

The constraint  $\text{tr}_{\text{PSOA}}^2 \circ \text{tr}_{\text{PSOA}}^1 \circ \text{tr}_{\text{PSOA}}^2 \neq \text{tr}_{\text{PSOA}}^1 \circ \text{tr}_{\text{PSOA}}^2$  indicates that if

$\text{tr}_{\text{PSOA}}^1$  is applied after  $\text{tr}_{\text{PSOA}}^2$ , then the output can still be changed by another application of  $\text{tr}_{\text{PSOA}}^2$ . This usually happens when a formula produced by  $\text{tr}_{\text{PSOA}}^1$  can be further changed by  $\text{tr}_{\text{PSOA}}^2$ .

Note that the  $\leftarrow$  relation is not transitive. If  $\text{tr}_{\text{PSOA}}^1 \leftarrow \text{tr}_{\text{PSOA}}^2$  and  $\text{tr}_{\text{PSOA}}^2 \leftarrow \text{tr}_{\text{PSOA}}^3$ ,  $\text{tr}_{\text{PSOA}}^1 \leftarrow \text{tr}_{\text{PSOA}}^3$  may not hold. This means if a composition uses both  $\text{tr}_{\text{PSOA}}^1$  and  $\text{tr}_{\text{PSOA}}^3$  but not  $\text{tr}_{\text{PSOA}}^2$ , any order of applying  $\text{tr}_{\text{PSOA}}^1$  and  $\text{tr}_{\text{PSOA}}^3$  may be correct.

In the following, we list the ‘ $\leftarrow$ ’ relations between transformation steps and give analyses.

- **desb**  $\leftarrow$  **unnest**: description via **desb** must be applied after unnesting via **unnest**, since **unnest** can produce a conjunction that contains an oidful atom that can be transformed via **desb**, e.g. an atom having more than one descriptor.
- **ccsp**  $\leftarrow$  **unnest**: conjunctive conclusion splitting via **ccsp** must be applied after unnesting via **unnest**, since **unnest** can create a top-level conjunction in a rule conclusion, which can be transformed via **ccsp**.
- **sk**  $\leftarrow$  **obj**: Skolemization via **sk** must be applied after objectification via **obj**. This is because **obj** may introduce a head existential which needs to be transformed via **sk**.
- **desb**  $\leftarrow$  **obj**: description via **desb** must be applied after objectification via **obj**, since oidless atoms can be given OIDs via **obj** and then be transformed via **desb**.
- **ccsp**  $\leftarrow$  **desb**: conjunctive conclusion splitting via **ccsp** must be applied

after description via `desb`, because `desb` can create a top-level conjunction in a rule conclusion, which can be transformed via `ccsp`.

- `ccsp`  $\leftarrow$  `sk`: conjunctive conclusion splitting via `ccsp` must be applied after Skolemization via `sk`, because the elimination of head existentials in `sk` can make an existentially quantified conjunction into a top-level conjunction in a rule conclusion, which can be transformed via `ccsp`.

Table 5.1 summarizes all the ‘ $\leftarrow$ ’ constraints between transformation steps.

Table 5.1: Constraints on applying transformation steps.

	unnest	obj	sk	desb	ccsp	flt	sub
unnest							
obj							
sk		$\leftarrow$					
desb	$\leftarrow$	$\leftarrow$					
ccsp	$\leftarrow$		$\leftarrow$	$\leftarrow$			
flt							
sub							

## 5.11 PSOA Transformations Demonstrated with Startup Example

We will demonstrate the transformations steps using another variant of Section 2.3.2.1’s Startup examples. The KB updates Example 2.3 by (1) adding an atom `_hire(_Ernie _Fred)` and a subclass formula `_startup##_company`; (2) changing the independent `_ceo` and `_cto` slots in the rule conclusion to an independent tuple using the syntax of PSOA RuleML 1.0 explained in Chap-

ter 3.<sup>36</sup>

**Example 5.3.** (Startup Variant with Subclass Formula)

```
Document (  
  Prefix(func: <http://www.w3.org/2007/rif-builtin-function#>  
  Prefix(pred: <http://www.w3.org/2007/rif-builtin-predicate#>  
  
  Group (  
    Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (  
      _startup(-[?CEO ?CTO] _employee->?Emp) :-  
        And(_cofounders(?CEO ?CTO) _hire(?CEO ?Emp)  
          _equity(?CEO ?CEOeqy) _equity(?CTO ?CTOeqy)  
        External(  
          pred:numeric-less-than-or-equal(  
            External(func:numeric-add(?CEOeqy ?CTOeqy)) 100))  
        )  
  
      _cofounders(_Ernie _Tony)  
      _hire(_Ernie _Kate)  
      _equity(_Ernie 50)  
      _equity(_Tony 30)  
      _startup##_company  
    )  
  )  
)
```

◇

Amongst the many possible queries over the KB in Example 5.3, our running query will be

```
_company(-[?X ?Y])
```

---

<sup>36</sup> This example updates the example from our earlier paper [73] to PSOA RuleML 1.0 by explicitly marking the independent tuple as explained in Section 3.2.2.

It asks for the independent tuple  $-[?X ?Y]$  of any `_company`, omitting the `_employee` slot. The desired answer is `?X=_Ernie ?Y=_Tony`.

Next we demonstrate how transformation steps can be applied to the KB and the query. In the following, for simplicity we omit the `Document` wrapper and the prefix declarations for KBs.

We start with subclass transformations in Section 5.5. The subclass axiomatization approach adds two axiomatization rules to the KB, while the subclass rewriting approach rewrites the subclass formula `_startup##_company` into a rule, as shown in the following.

#### **Subclass-rewritten KB:**

```
Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    ...
  )
  ...
  _equity(_Tony 30)

  Forall ?X (
    ?X#_company :- ?X#_startup
  )
)
```

Acting on the subclass-rewritten KB, the objectification step can adopt various approaches from Section 5.3.

Undifferentiated static objectification creates an existential quantification for every oidless atom.

#### **Objectified KB (undifferentiated, static):**

```

Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    Exists ?1 (?1#_startup(-[?CEO ?CTO] _employee->?Emp)) :-
      And(Exists ?2 (?2#_cofounders(?CEO ?CTO))
        Exists ?3 (?3#_hire(?CEO ?Emp))
        Exists ?4 (?4#_equity(?CEO ?CEOeqy))
        Exists ?5 (?5#_equity(?CTO ?CTOeqy))
        External(
          pred:numeric-less-than-or-equal(
            External(func:numeric-add(?CEOeqy ?CTOeqy)) 100)
          )
        )
    )
  )

  _1#_cofounders(_Ernie _Tony)
  _2#_hire(_Ernie _Kate)
  _3#_equity(_Ernie 50)
  _4#_equity(_Tony 30)

  Forall ?X ( ?X#_company :- ?X#_startup )
)

```

### Objectified Query (undifferentiated, static):

```
Exists ?1 (?1#_company(-[?X ?Y]))
```

Differentiated static objectification introduces OIDs  $_1, \dots, _4$  for the ground facts. It also introduces an existentially quantified variable  $?1$  for the oidless atom in the rule conclusion while introducing four universally quantified variables  $?2, \dots, ?5$  for the oidless atoms in the rule condition. Moreover, the query is objectified using a variable  $?1$ , which is encapsulated in an existential scope to avoid being treated as a free query variable.

## Objectified KB (differentiated, static):

```
Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy ?2 ?3 ?4 ?5 (
    Exists ?1 (?1#_startup(-[?CEO ?CTO] _employee->?Emp)) :-
      And(?2#_cofounders(?CEO ?CTO)
          ?3#_hire(?CEO ?Emp)
          ?4#_equity(?CEO ?CEOeqy)
          ?5#_equity(?CTO ?CTOeqy)
          External(
            pred:numeric-less-than-or-equal(
              External(func:numeric-add(?CEOeqy ?CTOeqy)) 100)))
  )

  _1#_cofounders(_Ernie _Tony)
  _2#_hire(_Ernie _Kate)
  _3#_equity(_Ernie 50)
  _4#_equity(_Tony 30)

  Forall ?X ( ?X#_company :- ?X#_startup )
)
```

## Objectified Query (differentiated, static):

```
Exists ?1 (?1#_company(-[?X ?Y]))
```

In contrast to the static objectification approaches, static/dynamic objectification introduces explicit OIDs only for the atom with the non-relational predicate `_startup`, which occurs in a slotted atom and in an (oidful) membership atom. It does not introduce explicit OIDs for the atoms with relational predicates `_cofounders`, `_hire`, and `_equity`. Since the atoms for which an explicit OID needs to be introduced occur only in the rule conclusion and

the query, undifferentiated static/dynamic and differentiated static/dynamic objectification coincide, leading to the following KB.

### Objectified KB (static/dynamic):

```

Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    Exists ?1 (
      ?1#_startup(-[?CEO ?CTO] _employee->?Emp)) :-
      And(_cofounders(?CEO ?CTO)
        _hire(?CEO ?Emp)
        _equity(?CEO ?CEOeqy)
        _equity(?CTO ?CTOeqy)
        External(
          pred:numeric-less-than-or-equal(
            External(func:numeric-add(?CEOeqy ?CTOeqy)) 100)))
    )

    _cofounders(_Ernie _Tony)
    _hire(_Ernie _Kate)
    _equity(_Ernie 50)
    _equity(_Tony 30)

    Forall ?X ( ?X#_company :- ?X#_startup )
  )

```

### Objectified Query (static/dynamic):

```

Exists ?1 (?1#_company(-[?X ?Y]))

```

Next, description transforms each oidful atom into a conjunction. The result, which is shown below, contains a rule with an existentially quantified conjunctive conclusion.

## Described KB:

```
Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    Exists ?1 (
      And(?1#_startup
        ?1#Top(-[?CEO ?CTO] _employee->?Emp)) :-
      And(_cofounders(?CEO ?CTO)
        _hire(?CEO ?Emp)
        _equity(?CEO ?CEOeqy)
        _equity(?CTO ?CTOeqy)
      External(
        pred:numeric-less-than-or-equal(
          External(func:numeric-add(?CEOeqy ?CTOeqy)) 100))
    )

    _cofounders(_Ernie _Tony)
    _hire(_Ernie _Kate)
    _equity(_Ernie 50)
    _equity(_Tony 30)

    Forall ?X ( ?X#_company :- ?X#_startup )
  )
)
```

## Described Query:

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

The application of the external predicate `pred:numeric-less-than-or-equal` is then flattened, with the argument `External(func:numeric-add(...))` being replaced by a fresh universal variable `?2`, obtaining the conjunction

```

And(?2=External(func:numeric-add(?CEOeqy ?CTOeqy))
    External(pred:numeric-less-than-or-equal(?6 100))

```

The Skolemization step eliminates the existential in the rule conclusion by replacing every occurrence of the variable ?1 with a Skolem function application `_skolem1(?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy ?2)`.

### Skolemized KB:

```

Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    And(_skolem1(?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy ?2)#_startup
        _skolem1(...)#Top(-[?CEO ?CTO] _employee->?Emp) :-
    And(_cofounders(?CEO ?CTO)
        _hire(?CEO ?Emp)
        _equity(?CEO ?CEOeqy)
        _equity(?CTO ?CTOeqy)
        ?2 = External(func:numeric-add(?CEOeqy ?CTOeqy))
        External(pred:numeric-less-than-or-equal(?2 100)))
  )

  _cofounders(_Ernie _Tony)
  _hire(_Ernie _Kate)
  _equity(_Ernie 50)
  _equity(_Tony 30)

  Forall ?X ( ?X#_company :- ?X#_startup )
)

```

Finally, the conjunction in the rule conclusion is split to obtain the normalized KB as shown on the next page. The four output rules have the same

condition and the same argument list for the Skolem function. The normalized query after rule splitting is the same as the described version shown above.

### Conjunctive-conclusion-split KB:

```

Group (
  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    _skolem1(?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy ?2)#_startup :-
      And(_cofounders(?CEO ?CTO)
        _hire(?CEO ?Emp)
        _equity(?CEO ?CEOeqy)
        _equity(?CTO ?CTOeqy)
        ?2 = External(func:numeric-add(?CEOeqy ?CTOeqy))
        External(pred:numeric-less-than-or-equal(?2 100)))
  )

  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    _skolem1(...)#Top(-[?CEO ?CTO]) :- And(...)
  )

  Forall ?CEO ?CTO ?Emp ?CEOeqy ?CTOeqy (
    _skolem1(...)#Top(_employee->?Emp) :- And(...)
  )

  _cofounders(_Ernie _Tony)
  _hire(_Ernie _Kate)
  _equity(_Ernie 50)
  _equity(_Tony 30)

  Forall ?X ( ?X#_company :- ?X#_startup )
)

```

To further exemplify the difference between static and static/dynamic objectification, we demonstrate the transformations of another two queries that use relational predicates, as shown in the following.

- `_cofounders(?X ?Y)`

This query is oidless. According to Definitions 5.5 and 5.6, both the undifferentiated and differentiated static objectification approaches transform it to `Exists ?1 (?1#_cofounders(?X ?Y))` using an explicit existential. In contrast, the static/dynamic approach keeps the query unchanged according to Definition 5.9.

- `?0#_hire(?X ?Y)`

This query is oidful. Both undifferentiated and differentiated static objectification approaches keep it unchanged since it is already oidful. In contrast, the static/dynamic objectification approach transforms it to the conjunction `And(_hire(?X ?Y) ?0=_oidcons(_hire ?X ?Y))`, where the first conjunct asks for a relationship while the second conjunct binds the OID query variable to virtual OID constructed using the `_oidcons` function, which takes the predicate `_hire` as the first argument according to Case 2.3 of Definition 5.8.

# Chapter 6

## Interoperation from N3 to PSOA

In this chapter, we study the interoperation from N3 to PSOA as shown in context by Figure 4.3. For the translational core of interoperation, Section 6.1 defines the N3Basic sublanguage of N3 used for the translation, and Section 6.2 introduces its translation to PSOA.

### 6.1 N3Basic: An N3 Sublanguage for Interoperation

In this section we introduce the N3 sublanguage, named N3Basic, suited for bidirectional interoperation between N3 and PSOA (used here for interoperation from N3 to PSOA). N3Basic corresponds to a rule language that expands RDF with (head-)existential rules. It restricts the N3 language defined in the W3C Team Submission [11] according to the following.

- Formulas of the form  $\{ \dots \}$  can only be used as the condition and the conclusion of a top-level N3 rule. Any other uses, e.g. as the subject or

the property value of a statement, are disallowed.

- The content in  $\{\dots\}$  is restricted to a possibly existentially quantified sequence of statements.
- Path expressions and lists are disallowed.
- Explicit existential quantification is disallowed on the top level of a document. This is because N3 does not have delimiters for the scope of quantification. An explicit top-level existential quantification would be regarded as having an implicit scope of the entire document, which would allow an existential quantification surrounding a rule and thus the semantics goes beyond normal Horn logic with existential rules.
- Rule conclusions cannot contain (action) primitives hence we have only derivation rules.

Following is the EBNF grammar to generate the N3Basic sublanguage of N3, which is modified from the BNF grammar of the N3-rules sublanguage<sup>37</sup>:

```
document      ::= declaration* (universal | statement | rule)*
declaration   ::= '@base' explicituri |
                '@keywords' barename (',' barename)* |
                '@prefix' prefix explicituri
existential   ::= '@forSome' symbol* '.'
universal     ::= '@forAll' symbol* '.'
rule          ::= '{' formulacontent '}' '=>' '{' formulacontent '}' '.'
formulacontent ::= existential* statement*
statement     ::= subject propertylist '.'
propertylist  ::= property (',' property)*
```

---

<sup>37</sup><https://www.w3.org/2000/10/swap/grammar/n3-rules.n3>

```

property      ::= property object (',' object)*
subject       ::= expression
property      ::= expression |
               'a' |
               ',='
object        ::= expression
expression    ::= pathitem
pathitem      ::= literal |
               numericliteral |
               symbol |
               quickvariable |
               '[' propertylist ']'
literal       ::= '"' UNICODESTRING '"'^^' SYMSPACE |
               ''' UNICODESTRING ''' '@' langtag |
               ''' UNICODESTRING '''
symbol        ::= explicitURI |
               qname

```

## 6.2 Translation from N3Basic to PSOA RuleML

The N3-to-PSOA transformation is composed of a normalization within N3 and a conversion from N3 to PSOA. The normalization within N3 transforms the input so that it consists of only triples or rules built on top of triples. This can be done by a process analogous to unnesting followed by description in PSOA. Meanwhile, compact IRIs using namespace prefixes are expanded to full IRIs without prefixes. Literals are expanded to the general form "literal"^^symospace. In our implementation, the normalization is provided

by the RDF4J library.<sup>38</sup>

The conversion from the normalized N3 constructs to PSOA constructs is described in the following, where  $\rho_{\text{N3,PSOA}}$  denotes the mapping function. First, we explain the conversion of terms in the following.

- IRI constants

In N3, an IRI quantified in `@forAll` or `@forsome` is regarded as a variable and its conversion will be explained later. Otherwise, the IRI is a constant  $c$  and  $\rho_{\text{N3,PSOA}}(c) = c$ .

- Literals

If  $c$  is a literal,  $\rho_{\text{N3,PSOA}}(c) = c$ .

- Blank nodes

If  $b$  is a blank node having the prefix `_:`, it is converted as follows:

- If  $b$  is in a fact, the conversion  $\rho_{\text{N3,PSOA}}(b)$  is a local constant in PSOA obtained by replacing the prefix `_:` of  $b$  with `_`.
- Otherwise,  $\rho_{\text{N3,PSOA}}(b)$  is a fresh variable `genvar(b)` generated to replace all occurrences of  $b$  within the scope of a query, a rule condition or a rule conclusion. The generated variables are quantified by an existential quantifier in the corresponding scope.

- Variables

If  $v$  is a ‘?’-prefixed variable,  $\rho_{\text{N3,PSOA}}(v) = v$ . If  $v$  is an IRI quantified by `@forAll` or `@forsome`, the conversion  $\rho_{\text{N3,PSOA}}(v) = \text{genvar}(v)$  is a fresh

---

<sup>38</sup><http://rdf4j.org/>

‘?’-prefixed variable generated to replace all occurrences of  $v$  in the corresponding scope.

The conversion of formulas are explained in the following.

- Triples

An N3 triple  $t$  has the form

$$o \quad p \quad v.$$

If  $p$  is `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`, then  $\rho_{N3,PSOA}(t)$  is a membership term in PSOA

$$\rho_{N3,PSOA}(o) \# \rho_{N3,PSOA}(v)$$

Otherwise,  $\rho_{N3,PSOA}(t)$  is a single-slot untyped frame

$$\rho_{N3,PSOA}(o) \# \text{Top}(\rho_{N3,PSOA}(p) \rightarrow \rho_{N3,PSOA}(v))$$

- Query, rule condition, and rule conclusion

If  $f$  is the top-level query, rule condition, or rule conclusion consisting of  $m$  triples  $t_1, \dots, t_m$ , the conversion  $\rho_{N3,PSOA}(f)$  is defined as

$$\text{Exists } ?X_1 \dots ?X_n (\text{And}(\rho_{N3,PSOA}(t_1) \dots \rho_{N3,PSOA}(t_m)))$$

The existential variables  $?X_1, \dots, ?X_n$  are the variables generated to replace blank nodes in  $f$ . If there are no blank nodes in  $f$ , the existential quantification is omitted. If  $m = 1$ , the conjunction **And** is omitted.

- Rules

An N3 rule  $l$  has the form  $\{f1\} \Rightarrow \{f2\}$ . or

$$\text{@forall } v_1, \dots, v_k. \{f1\} \Rightarrow \{f2\}.$$

The conversion  $\rho_{N3,PSOA}(l)$  is defined as the PSOA rule

$$\text{forall } \rho_{N3,PSOA}(v_1), \dots, \rho_{N3,PSOA}(v_n) (\rho_{N3,PSOA}(f2) :- \rho_{N3,PSOA}(f1))$$

The variables  $v_1, \dots, v_n, n \geq k$  are all universal variables occurring in  $f1$  and  $f2$ , including explicit quantified variables  $v_1, \dots, v_k$  and implicitly quantified variables having a '?' prefix.

**Example 6.1.** Consider the following N3 KB:

```
@prefix ex: <http://ex.com/>
```

```
ex:john ex:fatherOf ex:tom.
```

```
ex:tom ex:fatherOf ex:mary, [:age 19].
```

```
ex:mary ex:age 10.
```

```
{ ?X ex:fatherOf [ex:fatherOf ?Y]. }
```

```
=>
```

```
{ ?X ex:grandFatherOf ?Y. }.
```

```
{ ?X ex:fatherOf ?Y. }
```

```
=>
```

```
{ ?X ex:married [ex:motherOf ?Y]. }. ◇
```

The unnested KB contains three facts and two rules. The first fact shows `ex:john` is the father of `ex:tom`. The second fact tells `ex:tom` is the father

of mary and someone whose age is 19. The third fact shows the `ex:age` of `ex:mary` is 10. The first rule defines `ex:grandFatherOf` via compositions of `ex:fatherOf`. The second rule means if `?X` is the father of `?Y`, then `?X` married someone who is the mother of `?Y`.

The normalization within N3 creates explicit blank node names for the implicit square-bracketed blank nodes. Comma-separated property values will also be separated into separate triples. For example, the second fact introduces an OID `_:1` for the blank node, resulting in

```
ex:Tom ex:fatherOf ex:Mary, _:1.
_:1 ex:age 19.
```

The first of the above two facts will be further separated into two triples. After these steps, the KB in Example 6.1 becomes:

```
ex:John ex:fatherOf ex:Tom.
ex:Tom ex:fatherOf ex:Mary.
ex:Tom ex:fatherOf _:1.
_:1 ex:age 19.
ex:Mary ex:age 10.
{ ?X ex:fatherOf _:2. _:2 ex:fatherOf ?Y. }
=>
{ ?X ex:grandFatherOf ?Y }.
{ ?X ex:fatherOf ?Y. }
=>
{ ?X ex:married _:3. _:3 ex:motherOf ?Y. }.
```

The translation to PSOA RuleML is shown in the following, where the prefixed IRIs are expanded to full IRIs.

```

Document (
  Group (
    <http://ex.com/John>#Top(<http://ex.com/fatherOf>-><http://ex.com/Tom>)
    <http://ex.com/Tom>#Top(<http://ex.com/fatherOf>-><http://ex.com/Mary>)
    <http://ex.com/Tom>#Top(<http://ex.com/fatherOf>->_1)
    _1#Top(<http://ex.com/age>->19)
  Forall ?X ?Y (
    ?X#Top(<http://ex.com/grandFatherOf>->?Y) :-
      Exists ?2 (
        And(?X#Top(<http://ex.com/fatherOf>->?2)
          ?2#Top(<http://ex.com/fatherOf>->?Y))
      )
    )
  Forall ?X ?Y (
    Exists ?3 (
      And (?X#Top(<http://ex.com/married>->?3)
        ?3#Top(<http://ex.com/motherOf>->?Y))
    ) :-
      ?X#Top(<http://ex.com/fatherOf>->?Y)
    )
  )
)

```

The implementation of the above translation is confined to facts that are ground or existential. This implemented sublanguage of N3Basic constitutes an N3 sublanguage identical to RDF/Turtle [10]. This is needed for the inter-operation of “Linked Data”, “Graph Database”, and “Triple Store” applications as well as for use cases such as our OfficeProspector discussed in Section 10.2.

The translation is incorporated into PSOA’s Import statement. When a binary import statement `Import(<d1> <e1>)` is specified in a KB  $\phi$  with  $e1$

being `<http://www.w3.org/ns/entailment/Simple>`, the imported KB `d1` will undergo N3-to-PSOA translation and be merged with  $\phi$ .

# Chapter 7

## Interoperation from PSOA to TPTP (PSOA2TPTP)

In this chapter we explain the translation for interoperating from PSOA to the TPTP language in the context of Figure 4.3. The translation is composed of an FOL-targeting normalization within PSOA RuleML and a conversion from normalized PSOA to TPTP. Section 7.1 and Section 7.2 explain the normalization and the conversion, respectively. The conversion is a refinement of the one in our earlier paper [76]. Section 7.3 proves semantics preservation of the translation.

### 7.1 FOL-Targeting Normalization of the PSOA Kernel Source

In this section we introduce the transformation chain  $\eta_{\text{FOL}}$  for normalizing the source KBs/queries in the PSOA Kernel  $\mathcal{P}_K$  (cf. Definition 5.3) when

targeting FOL, which is the basis of the TPTP-FOF language. The chain is a composition of transformations `unnest`, `subr`, `obj`, and `desb` in as explained in Section 5.10, where `obj` covers all objectification variants explained in Section 5.3:

$$\eta_{\text{FOL}} = \text{desb} \circ \text{obj} \circ \text{sub}_r \circ \text{unnest}$$

All four transformations are needed for KBs while for queries, the chain can be simplified to `desb`  $\circ$  `obj`  $\circ$  `unnest` since `subr` does not change query formulas.  $\eta_{\text{FOL}}$  first eliminates embedded atoms via `unnest`, and then performs `subr`, `obj`, and `desb` to allow entailments to be evaluated under the relaxed PSOA semantics without the semantic restrictions, which can be aligned with FOL semantics. Since the transformation `obj` have multiple variants as explained in Sections 5.3,  $\eta_{\text{FOL}}$  also have different variants depending on which variants of `obj` is chosen. For the current version of PSOA RuleML, the expressivity of the TPTP-FOF translation output is the subset of FOL corresponding to Hornlog with existentials. However, the chain  $\eta_{\text{FOL}}$  can still be reused for future extensions of PSOA RuleML that has a full first-order expressivity.

Next we study the semantics preservation of the composition. In the following,  $\mathcal{P}_{\text{obj}} = (\Phi_{\text{obj}}, Q_{\text{obj}})$  denotes the PSOA sublanguage corresponding to the chosen objectification method (e.g.  $\mathcal{P}_{\text{obj}_{s \neq +d}}$  for `objs $\neq$ +d`).

**Theorem 7.1.** *The FOL-targeting normalization  $\eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{\text{-sod}}$ .*

*Proof.* Let  $\phi$  be a KB and  $q$  be a boolean query in  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ . By the explanation of Definition 4.1,  $\phi \in \Phi_{\text{obj}} \cap \Phi_{\text{sub}_r}$  and  $q \in Q_{\text{obj}} \cap Q_{\text{sub}_r}$ . It is easy

to verify that

$$\text{unnest}(\phi), \text{sub}_r \circ \text{unnest}(\phi), \text{obj} \circ \text{sub}_r \circ \text{unnest}(\phi),$$

$$\text{desb} \circ \text{obj} \circ \text{sub}_r \circ \text{unnest}(\phi) \in \phi \in \Phi_{\text{obj}} \cap \Phi_{\text{sub}_r}$$

and

$$\text{unnest}(\phi, q), \text{sub}_r \circ \text{unnest}(\phi, q), \text{obj} \circ \text{sub}_r \circ \text{unnest}(\phi, q),$$

$$\text{desb} \circ \text{obj} \circ \text{sub}_r \circ \text{unnest}(\phi, q) \in Q_{\text{obj}} \cap Q_{\text{sub}_r}$$

Since  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}} \preceq \mathcal{P}_K$  and  $\text{unnest}$  is semantics-preserving with respect to  $\mathcal{P}_K$  and  $\models$ , by Corollary 4.1  $\text{unnest}$  is also semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$  and  $\models$ . Also, since  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}} \preceq \mathcal{P}_{\text{sub}_r}$  and  $\text{sub}_r$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r}$ ,  $\models$ , and  $\models_{-s}$  by Theorem 5.9,  $\text{sub}_r$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{-s}$ . So, by Theorem 4.1,  $\text{sub}_r \circ \text{unnest}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{-s}$ .

Again, since  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}} \preceq \mathcal{P}_{\text{obj}}$  and  $\text{obj}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{obj}}$ ,  $\models_{-s}$ , and  $\models_{-so}$  by Theorems 5.3, 5.4, 5.5, and 5.6 where  $\models^\bullet$  is  $\models_{-s}$ ,  $\text{obj}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models_{-s}$ , and  $\models_{-so}$ . So, by Theorem 4.1,  $\text{obj} \circ \text{sub}_r \circ \text{unnest}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{-so}$ .

Finally, since  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}} \preceq \mathcal{P}_K$  and  $\text{desb}$  is semantics-preserving with respect to  $\mathcal{P}_K$ ,  $\models_{-so}$ , and  $\models_{-sod}$  by Theorem 5.10 where  $\models^\bullet$  is  $\models_{-so}$ ,  $\text{desb}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models_{-so}$ , and  $\models_{-sod}$ . Hence by

Theorem 4.1  $\text{desb} \circ \text{obj} \circ \text{sub}_r \circ \text{unnest}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{\text{-sod}}$  so that the theorem is proved.  $\square$

## 7.2 Converting Normalized External-Free PSOA Kernel to TPTP

In this section we deal with an **External-free** sublanguage of PSOA Kernel, called **External-free**  $\mathcal{P}_K$ , since built-ins are not available in the targeted FOF dialect of TPTP. We explain the conversion  $\zeta$  from the normalized KBs/queries in **External-free**  $\mathcal{P}_K$  to TPTP, which is defined using a recursive conversion  $\zeta'$ . We first define  $\zeta'$  for base terms in Section 7.2.1 and then define  $\zeta'$  for formulas and  $\zeta$  for KBs and queries in Section 7.2.2.

### 7.2.1 Base Terms

Base terms include constants, variables, and expressions. Their conversions are defined in the following.

- Constants

All constants must be parsed into a normalized form before the conversion, where

- Numbers (of datatype `xs:integer` or `xs:double`), strings (of datatype `xs:string`), and local constants (of datatype `rif:local`) are parsed into the corresponding shortcut forms.
- IRIs are normalized into the angle-bracket-enclosed form  $\langle \dots \rangle$ . Compact IRIs using namespace prefixes are expanded to full IRIs.

- Other constants are expanded to the full form " $\dots$ ".

The conversion  $\zeta'(c)$  of a normalized constant  $c$  is determined as follows:

- If  $c$  is a number,  $\zeta'(c)$  is the same number in TPTP.
- If  $c$  is a string,  $\zeta'(c)$  is the same double-quoted string in TPTP.
- Otherwise,  $\zeta'(c)$  is the single-quoted version of  $c$ . For example, a constant `_a` is converted to `'_a'`.

- Variables

The conversion  $\zeta'(v)$  of a '?'-prefixed variable  $v$  to TPTP replaces '?' with the upper-case letter 'Q' (Question mark). For example, a PSOA variable `?v` is mapped to a TPTP variable `Qv`.

- Expressions

The conversion of a nullary expression  $\mathbf{f}([\ ])$  is  $\zeta'(\mathbf{f})$ . When  $\mathbf{n} > 0$ , the conversion of an expression  $\mathbf{f}([\mathbf{t}_1 \dots \mathbf{t}_\mathbf{n}])$  is  $\zeta'(\mathbf{f}) (\zeta'(\mathbf{t}_1), \dots, \zeta'(\mathbf{t}_\mathbf{n}))$ , where the function and its arguments are all recursively mapped using  $\zeta'$ .

## 7.2.2 Formulas

After FOL-targeting normalization, subclass formulas are eliminated via `subr`, so that a PSOA input KB/query consists of two kinds of atomic formulas: atoms and equalities. Their conversions are explained in the following.

- Atoms

Atoms in a normalized PSOA KB/query can be further separated into three kinds: memberships, oidful single-descriptor atoms, and oidless single-dependent-tupled atoms. Their conversions are explained below.

– Memberships

A membership having the form  $o \# c()$  (abridged  $o \# c$ ) means  $o$  is an instance of  $c$ . Its conversion uses a reserved predicate, `memterm`,<sup>39</sup> resulting in a binary term `memterm( $\zeta'(o)$ ,  $\zeta'(c)$ )`.

– Oidful Single-Descriptor Atoms

An oidful single-descriptor atom can have either a tuple or a slot as the descriptor.

A single-tupled atom can be either dependent-tupled (having the form `o#f(+ [t1 ... tn])`) or independent-tupled (having the form `o#Top(- [t1 ... tn])` after normalization). Their conversions use the reserved predicates `prdtupterm` and `tupterm` correspondingly. Both of the reserved predicates take the OID conversion  $\zeta(o)$  as the first argument. `prdtupterm` takes the predicate conversion  $\zeta(f)$  as the second argument. The conversions of the  $n$  components of the tuple follow as the sequence of remaining arguments for both `prdtupterm` and `tupterm`. The result is a  $(2+n)$ -ary term for the dependent case

$$\text{prdtupterm}(\zeta'(o), \zeta'(f), \zeta'(t_1) \dots \zeta'(t_n))$$

and a  $(1+n)$ -ary term for the independent case

$$\text{tupterm}(\zeta'(o), \zeta'(t_1) \dots \zeta'(t_n))$$

Note that the predicates `prdtupterm` and `tupterm` are polyadic – i.e.,

---

<sup>39</sup>In the earlier PSOA2TPTP translation [76], the predicate name `member` was used. We changed the name in both PSOA2TPTP and PSOA2Prolog, because `member` is a built-in predicate in some Prolog systems such as SWI Prolog.

representing predicates of different arities – as allowed in FOL and by the TPTP syntax.

Similarly, a single-slotted atom can also be either dependent (having the form  $\text{o}\#f(\text{p}+\text{v})$ ) or independent (having the form  $\text{o}\#\text{Top}(\text{p}-\text{v})$  after normalization). Their conversions use the reserved predicates `prdsloterm` and `sloterm` correspondingly. Both of the reserved predicates take the OID conversion  $\zeta(\text{o})$  as the first argument. `prdsloterm` takes the predicate conversion  $\zeta(f)$  as the second argument. The conversions of the slot name `p` and slot filler `v` follow as the sequence of remaining arguments for both `prdsloterm` and `sloterm`.

– Oidless Single-Dependent-Tupled Atoms

An oidless single-dependent-tupled atom has the form  $f(+[\tau_1 \dots \tau_n])$ . It can only occur when `f` is a relational predicate and static/dynamic objectification is chosen for the normalization  $\eta_{\text{FOL}}$ . The atom is converted in the same way as an expression, leading to a proposition (when  $n = 0$ ) or an  $n$ -ary relationship (when  $n > 0$ ) in TPTP-FOF.

- Equalities

In `External`-free  $\mathcal{P}_K$ , an equality  $\tau_1 = \tau_2$  cannot be used for built-in calls but only for unifying two base terms. The formula is converted to  $\zeta'(\tau_1) = \zeta'(\tau_2)$  in TPTP.

Non-atomic formulas in PSOA, including rule implications, conjunctions, disjunctions, existential quantification, and universal quantification are converted to corresponding TPTP formulas. Table 7.1 lists the conversions of all formulas in FOL-normalized PSOA KBs/queries to TPTP.

Table 7.1: Conversion from PSOA/PS formulas to TPTP formulas.

PSOA/PS Formulas	TPTP Formulas
$\text{o\#Top}(-[t_1 \dots t_n])$	$\text{tupterm}(\zeta'(o), \zeta'(t_1), \dots, \zeta'(t_n))$
$\text{o\#f}([t_1 \dots t_n])$	$\text{prdtupterm}(\zeta'(o), \zeta'(f), \zeta'(t_1), \dots, \zeta'(t_n))$
$\text{o\#Top}(p \rightarrow v)$	$\text{sloterm}(\zeta'(o), \zeta'(p), \zeta'(v))$
$\text{o\#f}(p \rightarrow v)$	$\text{prdsloterm}(\zeta'(o), \zeta'(f), \zeta'(p), \zeta'(v))$
$\text{o\#c}$	$\text{memterm}(\zeta'(o), \zeta'(c))$
$\text{f}([t_1 \dots t_n])$	$\begin{cases} \zeta'(f) & n = 0 \\ \zeta'(f) (\zeta'(t_1), \dots, \zeta'(t_n)) & n > 0 \end{cases}$
$\text{And}(\tau_1 \dots \tau_n)$	$(\zeta'(\tau_1) \& \dots \& \zeta'(\tau_n))$
$\text{Or}(\tau_1 \dots \tau_n)$	$(\zeta'(\tau_1) \mid \dots \mid \zeta'(\tau_n))$
$\text{Exists } ?X_1 \dots ?X_n (\tau)$	$?[\zeta'(X_1), \dots, \zeta'(X_n)] : \zeta'(\tau)$
$\text{Forall } ?X_1 \dots ?X_n (\tau)$	$![\zeta'(X_1), \dots, \zeta'(X_n)] : \zeta'(\tau)$
$\tau_1 :- \tau_2$	$\zeta'(\tau_1) \leq \zeta'(\tau_2)$
$\tau_1 = \tau_2$	$\zeta'(\tau_1) = \zeta'(\tau_2)$

Using the recursive  $\zeta'$ , the conversion  $\zeta$  can be defined as follows. A normalized PSOA KB  $\phi$  consisting of clauses  $\psi_1, \dots, \psi_n$  has the form

```
Document (
  Group (
     $\psi_1$ 
    ...
     $\psi_n$ 
  )
)
```

Its conversion  $\zeta(\phi)$  is

```
f of(ax1, axiom,  $\zeta'(\psi_1)$ ).
...
f of(axn, axiom,  $\zeta'(\psi_n)$ ).
```

In the KB conversion, each clause conversion  $\zeta'(\psi_i)$ ,  $1 \leq i \leq n$  is annotated as a TPTP-FOF axiom having the label *axi*.

Regarding query conversion, in spite of a proposed (non-official) syntax for answer extraction [68], different syntax has been used by engines. Here we

present a query conversion for the VampirePrime engine we used in our TPTP instantiation of PSOATransRun. The conversion  $\zeta(\phi, q)$  uses a reserved answer predicate `ans` for answer extraction as shown in the following.

$$\begin{aligned} & ![\zeta'(?X1), \zeta'(?X2), \dots]: \\ & (\text{ans}("?X1 = ", \zeta'(?X1), "?X2 = ", \zeta'(?X2), \dots) \Leftarrow \zeta'(q)). \end{aligned}$$

In the above formula, `?X1`, `?X2`, `...` are the free variables in `q`. Answers from VampirePrime are of the form

$$\text{ans}("?X1 = ", v1, "?X2 = ", v2, \dots)$$

where `v1`, `v2`, `...` are bindings for the variables in TPTP. When the query has no variables, `ans` is used alone as the conclusion.

### 7.3 Proof of Semantics Preservation

In this section we discuss the semantics preservation of the translation from the PSOA sublanguage  $\mathcal{P}_K$  to TPTP-FOF. Since TPTP-FOF adopts FOL semantics, for convenience, in our discussion we use the conversions  $\zeta_{\text{FOL}}$  and  $\zeta'_{\text{FOL}}$  from PSOA to FOL, where  $\zeta'_{\text{FOL}}$  is the counterpart of  $\zeta'$  using the FOL syntax of formulas (cf. Section 2.1.1 and Table 2.1) and  $\zeta_{\text{FOL}}$  is defined on top of  $\zeta'_{\text{FOL}}$  according to Definition 4.2.

In this section, we use  $\models_{\text{FOL}}$  to denote the first-order semantics explained in Section 2.1.1.

**Lemma 7.1.**  *$\zeta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\zeta_{\text{FOL}}}$ ,  $\models_{\text{-sod}}$ , and  $\models_{\text{FOL}}$ , where  $\mathcal{P}_{\zeta_{\text{FOL}}} = (\Phi_{\zeta_{\text{FOL}}}, Q_{\zeta_{\text{FOL}}})$  is the PSOA sublanguage where  $\Phi_{\zeta_{\text{FOL}}}$*

and  $Q_{\zeta_{\text{FOL}}}$  are subsets of  $\Phi_K$  and  $Q_K$  that are free of external terms, subclass formulas, and have only formulas of the forms in Table 7.1.

*Proof.* We need to show that for any KB  $\phi$  in  $\Phi_{\zeta_{\text{FOL}}}$  and boolean query  $q$  in  $Q_{\zeta_{\text{FOL}}}$ ,  $\phi \models_{\text{-sod}} q$  iff  $\zeta_{\text{FOL}}(\phi) \models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$  holds. This is equivalent to proving  $\phi \not\models_{\text{-sod}} q$  iff  $\zeta_{\text{FOL}}(\phi) \not\models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$ .

We first prove the ‘only if’ part. If  $\phi \not\models q$ , there exists a counter-model  $\mathcal{I}$  for  $\phi \not\models q$ . We can define an FOL interpretation  $\mathcal{I}'$  as follows:

- $\mathcal{I}'$  has the same domain as  $\mathcal{I}$ ;
- For any individual  $c'$  in  $\zeta_{\text{FOL}}(\phi)$ , there exists a constant  $c$  in  $\phi$  s.t.  $c' = \zeta_{\text{FOL}}(c)$ . We define  $c^{\mathcal{I}'} = c^{\mathcal{I}}$ .
- For any function  $f'$  in  $\zeta_{\text{FOL}}(\phi)$ , there exists a constant  $f$  in  $\phi$  s.t.  $f' = \zeta_{\text{FOL}}(f)$ .  $f'^{\mathcal{I}'}$  is defined to be the semantic function such that  $f'^{\mathcal{I}'}(e_1, \dots, e_n) = f^{\mathcal{I}}(\{\{\}, \langle e_1 \dots e_n \rangle, \{\}, \{\}, \{\})$  for any  $e_1 \dots e_n \in \mathcal{I}.D$ ;
- For any predicate constant  $r$  in  $\zeta_{\text{FOL}}(\phi)$ , there are two cases:
  - $r'$  is a reserved predicate. In this case  $r^{\mathcal{I}'}$  is defined as follows:

$$\text{tupterm}^{\mathcal{I}'}(o, e_1, \dots, e_n) = \text{Top}^{\mathcal{I}}(\{o\}, \{\}, \{\langle e_1, \dots, e_n \rangle\}, \{\}, \{\})$$

$$\text{prdtupterm}^{\mathcal{I}'}(o, f, e_1, \dots, e_n) = f^{\mathcal{I}}(\{o\}, \{\langle e_1, \dots, e_n \rangle\}, \{\}, \{\}, \{\})$$

$$\text{sloterm}^{\mathcal{I}'}(o, p, v) = \text{Top}^{\mathcal{I}}(\{o\}, \{\}, \{\}, \{\}, \{\langle p, v \rangle\})$$

$$\text{prdsloterm}^{\mathcal{I}'}(o, f, p, v) = f^{\mathcal{I}}(\{o\}, \{\}, \{\}, \{\langle p, v \rangle\}, \{\})$$

$$\text{memterm}^{\mathcal{I}'}(o, c) = c^{\mathcal{I}}(\{o\}, \{\}, \{\}, \{\}, \{\})$$

- $r' = \zeta_{\text{FOL}}(r)$  is converted from a relational predicate  $r$  in  $\phi$ . In this case  $r'^{\mathcal{I}'}$  is defined to be the semantic function such that  $r'^{\mathcal{I}'}(e_1, \dots, e_n) = TVal_{\mathcal{I}'}(r^{\mathcal{I}}(\{\}, \{\langle e_1 \dots e_n \rangle\}, \{\}, \{\}, \{\}))$  for any  $e_1 \dots e_n \in \mathcal{I}.D$ .

With this definition, we can prove by induction that for any formula  $\tau$  in  $\phi$  and any  $\mathbf{I}_V \in \mathbf{M}(\text{Var}(\tau), \mathcal{I}.D)$ ,  $\mathbf{v}(\mathcal{I}', \mathbf{I}_V') \models_{\text{FOL}} \zeta_{\text{FOL}}(\tau)$  iff  $\mathbf{v}(\mathcal{I}, \mathbf{I}_V) \models_{\text{-sod}} \tau$ , where  $\mathbf{I}_V' = \{(\zeta_{\text{FOL}}(v), e) \mid (v, e) \in \mathbf{I}_V\}$ .

- Base case

If  $\tau$  is an atomic formula in the normalized KB,  $\tau$  can be either an atom or an equality. If  $\tau$  is an atom, the statement holds by the definition of predicate interpretation in  $\mathcal{I}'$ . If  $\tau$  is an equality, the statement also holds since  $\mathcal{I}'$  retains the interpretation of base terms from  $\mathcal{I}$ .

- Inductive case

If  $\tau$  is a non-atomic formula, it can be a conjunction, a disjunction, an existential quantification, a rule implication, or a universal quantification. Since these PSOA constructs have the same meaning as in FOL, the statement can be proved by aligning the FOL semantics of each construct with the PSOA semantics.

Based on the statement, we have  $\mathcal{I}' \models_{\text{FOL}} \zeta_{\text{FOL}}(\phi)$  and  $\mathcal{I}' \not\models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$ . Thus  $\mathcal{I}'$  is a counter-model of  $\zeta_{\text{FOL}}(\phi) \not\models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$ . So  $\phi \models_{\text{-sod}} q$  only if  $\zeta_{\text{FOL}}(\phi) \models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$  holds.

Next we prove the ‘if’ part. If  $\zeta_{\text{FOL}}(\phi) \not\models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$ , there exists a counter-model  $\mathcal{I}'$  s.t.  $\mathcal{I}' \models_{\text{FOL}} \zeta_{\text{FOL}}(\phi)$  and  $\mathcal{I}' \not\models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$ . We can construct the PSOA interpretation  $\mathcal{I}$  similarly by applying the equations in the ‘only if’ in the opposite direction. It can also be proved inductively that

$\mathcal{I} \models_{-sod} \phi$  and  $\mathcal{I} \not\models_{-sod} q$ . Hence  $\mathcal{I}$  is a counter model of  $\phi \not\models_{-sod} q$ . So  $\phi \models_{-sod} q$  if  $\zeta_{\text{FOL}}(\phi) \models_{\text{FOL}} \zeta_{\text{FOL}}(\phi, q)$  also holds.  $\square$

**Theorem 7.2.** *The PSOA-to-TPTP translation  $\eta_{\text{FOL}} \circ \zeta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{FOL}}$ ,  $\models$ , and  $\models_{\text{FOL}}$ , where  $\mathcal{P}_{\text{FOL}}$  is the **External-free** subset of  $\mathcal{P}_{\text{sub},r} \cap \mathcal{P}_{\text{obj}}$ .*

*Proof.* Since  $\mathcal{P}_{\text{FOL}} \preceq \mathcal{P}_{\text{sub},r} \cap \mathcal{P}_{\text{obj}}$ , by Theorem 7.1 and Corollary 4.1,  $\eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{FOL}}$ ,  $\models$ , and  $\models_{-sod}$ . By Lemma 7.1,  $\zeta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\zeta_{\text{FOL}}}$ ,  $\models_{-sod}$ , and  $\models_{\text{FOL}}$ .

For any KB  $\phi$  and boolean query  $q$  in  $\mathcal{P}_{\text{FOL}}$ ,  $\zeta_{\text{FOL}}(\phi)$  and  $\zeta_{\text{FOL}}(\phi, q)$  are in  $\mathcal{P}_{\zeta_{\text{FOL}}}$ . Hence by Theorem 4.1  $\eta_{\text{FOL}} \circ \zeta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{FOL}}$ ,  $\models$ , and  $\models_{\text{FOL}}$ .  $\square$

# Chapter 8

## Interoperation from PSOA to Prolog (PSOA2Prolog)

In this chapter we explain the translation from PSOA Kernel to the ISO Prolog language for PSOA-to-Prolog interoperation in the context of Figure 4.3. Section 8.1 discusses the LP-targeting normalization within PSOA RuleML. Section 8.2 discusses the conversion of normalized PSOA input to Prolog. The normalization and conversion are refinements of the corresponding phases in our earlier paper [73]. Section 8.3 proves semantics preservation of the translation.

### 8.1 LP-Targeting Normalization of the PSOA Kernel Source

This section introduces the transformation chain  $\eta_{LP}$  for performing Logic-Programming-targeting (LP-targeting) normalization of the source KBs/queries

in the PSOA Kernel  $\mathcal{P}_K$  (cf. Definition 5.3).

In contrast to FOL, LP does not support existentials and conjunctions in rule conclusions, thus  $\eta_{LP}$  needs to perform **sk** and **ccsp** after  $\eta_{FOL}$ . Moreover, since LP systems usually support built-ins,  $\eta_{LP}$  also includes **flt** to extract embedded built-in calls in PSOA so that they can be mapped to the LP-correspondences in the target language. The resulting  $\eta_{LP}$  is the following composition of the transformations defined in Chapters 5 and 7:

$$\begin{aligned}\eta_{LP} &= \text{flt} \circ \text{ccsp} \circ \text{sk} \circ \eta_{FOL} \\ &= \text{flt} \circ \text{ccsp} \circ \text{sk} \circ \text{desb} \circ \text{obj} \circ \text{sub}_r \circ \text{unnest}\end{aligned}$$

Since – like  $\text{sub}_r$  in Chapter 7 – **ccsp** and **sk** do not act on queries, for queries the transformation  $\eta_{LP}(\phi, q)$  can be simplified as follows:

$$\begin{aligned}\eta_{LP}(\phi, q) &= (\text{flt} \circ \eta_{FOL})(\phi, q) \\ &= (\text{flt} \circ \text{desb} \circ \text{obj} \circ \text{unnest})(\phi, q)\end{aligned}$$

**Theorem 8.1.** *The LP-targeting normalization  $\eta_{LP}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$ ,  $\models$ , and  $\models_{\text{-sod}}$ .*

*Proof.* Let  $\phi$  be a KB and  $q$  be a boolean query in  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$ . By the explanation of Definition 4.1,  $\phi \in \Phi_K \cap \Phi_{\text{obj}} \cap \Phi_{\text{sub}_r} = \Phi_{\text{obj}} \cap \Phi_{\text{sub}_r}$  and  $q \in Q_{\text{sk}} \cap Q_{\text{obj}} \cap Q_{\text{sub}_r}$ . It is easy to verify that

$$\begin{aligned}\eta_{FOL}(\phi), \text{sk} \circ \eta_{FOL}(\phi), \text{ccsp} \circ \text{sk} \circ \eta_{FOL}(\phi), \\ \text{flt} \circ \text{ccsp} \circ \text{sk} \circ \eta_{FOL}(\phi) \in \Phi_{\text{obj}} \cap \Phi_{\text{sub}_r}\end{aligned}$$

and

$$\eta_{\text{FOL}}(\phi, q), \text{sk} \circ \eta_{\text{FOL}}(\phi, q), \text{ccsp} \circ \text{sk} \circ \eta_{\text{FOL}}(\phi, q),$$

$$\text{flt} \circ \text{ccsp} \circ \text{sk} \circ \eta_{\text{FOL}}(\phi, q) \in Q_{\text{sk}} \cap Q_{\text{obj}} \cap Q_{\text{sub}_r}$$

Since  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r} \preceq \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$  and  $\eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_K$  and  $\models$ , by Corollary 4.1  $\eta_{\text{FOL}}$  is also semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$  and  $\models$ . Also, since  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r} \preceq \mathcal{P}_{\text{sk}}$  and  $\text{sk}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}}$  and  $\models_{\text{-sod}}$  by Theorem 5.7 where  $\models^\bullet$  is  $\models_{\text{-sod}}$ ,  $\text{sk}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$  and  $\models_{\text{-sod}}$ . So, by Theorem 4.1,  $\text{sk} \circ \eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{\text{-sod}}$ .

Again, since  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r} \preceq \mathcal{P}_K$  and  $\text{ccsp}$  is semantics-preserving with respect to  $\mathcal{P}_K$  and  $\models_{\text{-sod}}$  by Theorem 5.12,  $\text{ccsp}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$  and  $\models_{\text{-sod}}$ . So, by Theorem 4.1,  $\text{ccsp} \circ \text{sk} \circ \eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sub}_r} \cap \mathcal{P}_{\text{obj}}$ ,  $\models$ , and  $\models_{\text{-sod}}$ .

Finally, since  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r} \preceq \mathcal{P}_K$  and  $\text{flt}$  is semantics-preserving with respect to  $\mathcal{P}_K$  and  $\models_{\text{-sod}}$  by Theorem 5.10 where  $\models^\bullet$  is  $\models_{\text{-sod}}$ ,  $\text{flt}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$  and  $\models_{\text{-sod}}$ . Hence by Theorem 4.1  $\text{flt} \circ \text{ccsp} \circ \text{sk} \circ \eta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$ ,  $\models$ , and  $\models_{\text{-sod}}$  so that the theorem is proved.  $\square$

## 8.2 Converting Normalized PSOA Kernel to Prolog

In this section, we explain the conversion  $\rho$  from the normalized KBs/queries in  $\mathcal{P}_K$  to Prolog, which is defined using a recursive mapping function  $\rho'$ . In the following, we first discuss the conversion of base terms and then discuss the conversion of formulas.

### 8.2.1 Base Terms

Conversions of based terms in PSOA are defined in the following.

- Constants

Before the conversion, any constant  $c$  must be normalized in the same way as explained in Section 7.2.1 for the TPTP translation. The conversion  $\rho'(c)$  of a normalized constant  $c$  is determined as follows:

- If  $c$  is a number,  $\rho'(c)$  is the corresponding Prolog number.
- If  $c$  is an arithmetic built-in function/predicate adopted from RIF [62],  $\rho'(c)$  is the corresponding Prolog built-in function/predicate, as listed in Table 8.1. Since Prolog's `rem` and `'//'` functions accept only integer arguments, the PSOA2Prolog translation supports only integer arguments for `func:numeric-integer-divide` and `func:numeric-mod`.
- If  $c$  is a string having the form `"..."`,  $\rho'(c)$  replaces every single quote character `'` with the Prolog escaped sequence<sup>40</sup> `''` and surrounds it

---

<sup>40</sup>An alternative form of an escaped single quote in Prolog is `\'`. We do not choose it because the XSB engine does not support it.

Table 8.1: Conversion of PSOA (arithmetic) built-ins to Prolog.

PSOA/PS Built-in	Prolog Built-in
<code>func:numeric-add</code>	<code>'+'</code>
<code>func:numeric-subtract</code>	<code>'-'</code>
<code>func:numeric-multiply</code>	<code>'*'</code>
<code>func:numeric-divide</code>	<code>'/'</code>
<code>func:numeric-integer-divide</code>	<code>'//'</code>
<code>func:numeric-mod</code>	<code>rem</code>
<code>pred:numeric-equal</code>	<code>'=='</code>
<code>pred:numeric-less-than</code>	<code>'&lt;'</code>
<code>pred:numeric-less-than-or-equal</code>	<code>'=&lt;'</code>
<code>pred:numeric-greater-than</code>	<code>'&gt;'</code>
<code>pred:numeric-greater-than-or-equal</code>	<code>'&gt;='</code>
<code>pred:numeric-not-equal</code>	<code>'\='</code>
<code>pred:is-literal-integer</code>	<code>integer</code>

with a pair of single quotes, yielding a Prolog constant having the form `'...'`. We do not convert PSOA strings directly to Prolog strings because in Prolog double-quoted strings are stored and returned in bindings as integer lists, so that the inverse conversion to PSOA would not be unique should lists be added to PSOA in the future.

– Otherwise,  $\rho'(c)$  is the single-quoted version of  $c$ .

- Variables

The conversion  $\rho'(v)$  of a `'?'`-prefixed variable  $v$  is the same as  $\zeta(v)$ , which replaces `'?'` with `'Q'`.

- Expressions

The conversion of an expression  $f(+[t_1 \dots t_n])$  is  $\rho'(f)$  when  $n = 0$  and  $\rho'(f)(\rho'(t_1), \dots, \rho'(t_n))$  when  $n > 0$ , where the function and its arguments are all recursively mapped using  $\rho'$ . The conversion of an external expression in the form of `External(f(+[t1 ... tn]))` is the same as  $\rho'(f(+[t_1 \dots t_n]))$ .

## 8.2.2 Formulas

The conversion of PSOA formulas are explained below, starting with atomic formulas followed by non-atomic formulas. After normalization, atomic formulas include atoms and equalities. Their conversions are explained below.

- Atoms

After normalization, a PSOA KB or query atom can be a membership, an oidful single-descriptor atom, or an oidless single-dependent-tupled atom. Its conversion is the same as its TPTP conversion in Section 7.2, using the reserved predicate `memterm` for a membership, `prdtupterm` for a single-dependent-tupled atom, `tupterm` for a Top-typed, single-independent-tupled atom, `prdsloterm` for a single-dependent-slotted atom, and `sloterm` for a Top-typed, single-independent-slotted atom.

- Equalities

The conversion of an equality  $\tau_1 = \tau_2$  depends on the right-hand side formula  $\tau_2$ . If  $\tau_2$  is a built-in function call having the form `External(f(t1...tn))`, the left-hand side  $\tau_1$  is unified with the evaluation result of  $\tau_2$ . The formula is mapped to a binary `is`-primitive invocation in Prolog. If  $\tau_2$  is not a built-in function call, the formula is a unification between two base terms in PSOA and is mapped using Prolog's unification predicate `'='`.

Next we proceed to the conversion of non-atomic formulas.

The conversion of a conjunction `And(...)` is a parenthesized comma-separated Prolog formula, which can be nested inside an outer formula. Similarly, the conversion of a disjunction `Or(...)` is a semicolon-separated Prolog formula.

Table 8.2: Conversion from PSOA/PS formulas to Prolog formulas.

PSOA/PS Formulas	Prolog Formulas
$\text{o\#Top}(-[t_1 \dots t_n])$	$\text{tupterm}(\rho'(o), \rho'(t_1), \dots, \rho'(t_n))$
$\text{o\#f}([t_1 \dots t_n])$	$\text{prdtupterm}(\rho'(o), \rho'(f), \rho'(t_1), \dots, \rho'(t_n))$
$\text{o\#Top}(p \rightarrow v)$	$\text{sloterm}(\rho'(o), \rho'(p), \rho'(v))$
$\text{o\#f}(p \rightarrow v)$	$\text{prdsloterm}(\rho'(o), \rho'(f), \rho'(p), \rho'(v))$
$\text{o\#c}$	$\text{memterm}(\rho'(o), \rho'(c))$
$f(+[t_1 \dots t_n])$	$\begin{cases} \rho'(f) & n = 0 \\ \rho'(f) (\rho'(t_1), \dots, \rho'(t_n)) & n > 0 \end{cases}$
$\tau_1 = \tau_2$	$\begin{cases} \text{is}(\rho'(\tau_1), \rho'(\tau_2)) & \text{if } \tau_2 \text{ is External}(\dots) \\ \text{'='}(\rho'(\tau_1), \rho'(\tau_2)) & \text{otherwise} \end{cases}$
$\text{And}(\tau_1 \dots \tau_n)$	$(\rho'(\tau_1), \dots, \rho'(\tau_n))$
$\text{Or}(\tau_1 \dots \tau_n)$	$(\rho'(\tau_1); \dots ; \rho'(\tau_n))$
$\text{Exists } ?X_1 \dots ?X_n (\tau)$	$\rho'(\tau)$
$\text{Forall } ?X_1 \dots ?X_n (\tau)$	$\rho'(\tau)$
$\text{External}(\tau)$	$\rho'(\tau)$
$\tau_1 :- \tau_2$	$\rho'(\tau_1) :- \rho'(\tau_2)$

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

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

The conversion of an external formula  $\text{External}(\tau)$  is the same as the conversion of the formula inside the  $\text{External}$  wrapper.

Table 8.2 lists the conversions from PSOA/PS formulas to Prolog formulas.

Using the recursive  $\rho'$ , the conversion  $\rho$  can be defined as follows. A normalized PSOA KB  $\phi$  consisting of clauses  $\psi_1, \dots, \psi_n$  has the following form:

```

Document (
  Group (
     $\psi_1$ 
    ...
     $\psi_n$ 
  )
)

```

Its conversion  $\rho(\phi)$  is

```

 $\rho'(\psi_1)$ 
...
 $\rho'(\psi_n)$ 

```

For a normalized query  $q$ , the conversion  $\rho(\phi, q)$  appends a period ‘.’ to the end of  $\rho'(q)$ .

### 8.3 Proof of Semantics Preservation

In this section we discuss the semantics preservation of  $\rho$  as well as our PSOA-to-Prolog translation  $\text{tr}_{\text{PSOA,Prolog}} = \eta_{\text{LP}} \circ \rho$ . In the discussion, we use the declarative semantics of definite logic programs in [57], which is defined using Herbrand models and is proved to be equivalent to FOL semantics for definite logic programs. Since the semantics of external built-in calls is not captured by the declarative semantics, we focus on an **External**-free sublanguage of PSOA in this section, where the translation output is in the Pure Prolog subset of ISO Prolog, introduced in Section 2.1.2.

**Lemma 8.1.**  *$\rho$  is semantics-preserving with respect to  $\mathcal{P}_\rho$ ,  $\models_{\text{-sod}}$ , and  $\models_{\text{LP}}$ , where  $\mathcal{P}_\rho$  has only formulas shown in Table 8.2 and is **External**-free.*

*Proof.* By Lemma 7.1, the FOL correspondence of  $\rho$  is semantics-preserving with respect to  $\mathcal{P}_{\zeta_{\text{FOL}}}$ ,  $\models_{\text{-sod}}$ , and  $\models_{\text{FOL}}$ . Since  $\mathcal{P}_\rho \preceq \mathcal{P}_{\zeta_{\text{FOL}}}$ ,  $\rho$  is semantics-preserving with respect to  $\mathcal{P}_\rho$ ,  $\models_{\text{-sod}}$ , and  $\models_{\text{FOL}}$ . And since  $\models_{\text{LP}}$  coincides with  $\models_{\text{FOL}}$  on definite logic programs, the lemma holds.  $\square$

**Theorem 8.2.**  $\eta_{\text{LP}} \circ \rho$  is semantics-preserving with respect to  $\mathcal{P}_{\text{LP}}$ ,  $\models$ , and  $\models_{\text{LP}}$ , where  $\mathcal{P}_{\text{LP}}$  is the **External-free** sublanguage of  $\mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$ .

*Proof.* Since  $\mathcal{P}_{\text{LP}} \preceq \mathcal{P}_{\text{sk}} \cap \mathcal{P}_{\text{obj}} \cap \mathcal{P}_{\text{sub}_r}$ , by Theorem 8.1 and Corollary 4.1,  $\eta_{\text{LP}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{LP}}$ ,  $\models$ , and  $\models_{\text{-sod}}$ . By Lemma 7.1,  $\rho$  is semantics-preserving with respect to  $\mathcal{P}_\rho$ ,  $\models_{\text{-sod}}$ , and  $\models_{\text{LP}}$ .

For any KB  $\phi$  and boolean query  $q$  in  $\mathcal{P}_{\text{LP}}$ ,  $\rho(\phi)$  and  $\rho(\phi, q)$  are in  $\mathcal{P}_{\zeta_{\text{FOL}}}$ . Hence by Theorem 4.1  $\eta_{\text{FOL}} \circ \zeta_{\text{FOL}}$  is semantics-preserving with respect to  $\mathcal{P}_{\text{LP}}$ ,  $\models$ , and  $\models_{\text{-sod}}$ .  $\square$

According to Chapter 2 of [57], SLD-resolution ( $\vdash_{\text{SLD}}$ ) is sound and complete with respect to the declarative semantics  $\models_{\text{LP}}$ . Hence the following theorem follows.

**Theorem 8.3.**  $\eta_{\text{FOL}} \circ \rho$  is semantics-preserving with respect to  $\mathcal{P}_{\text{LP}}$ ,  $\models$ , and  $\vdash_{\text{SLD}}$ .

Note that for SLD-resolution, there is no textual ordering of clauses, and the order of conjunct execution is neither fixed in a conjunctive rule condition nor in a conjunctive query. For a specific implementation (e.g., by depth-first exploration of the search tree), these orders are usually fixed, hence the corresponding proof procedure may not terminate (e.g., because it diverges into an infinite path of the search tree), leading to implementation incompleteness. In particular, for the Prolog instantiation of PSOATransRun (see Section 9.2.2),

the orders in PSOA KBs/queries (within `Group` as well as rule-condition- and query-`And` constructs) need to comply to the ordering required by (ISO and XSB) Prolog.

Also note that Prolog makes a closed-world assumption while PSOA's semantics makes an open-world assumption. However, since PSOA RuleML currently does not support any form of negation, the distinction in the assumptions would not lead to different entailments or answers. Hence, the "No" answers obtained from PSOATransRun should be interpreted as non-entailments, for boolean queries, or 'no (more) answers derivable', for queries having free variables.

# Chapter 9

## Implementation of PSOA RuleML (PSOATransRun)

In this chapter we discuss the prototype implementation of PSOA RuleML based on the PSOATransRun framework explained in Section 4.2. Section 9.1 explains the implementation of translators in ANTLR. Section 9.2 explains the combination of translators and runtime engines. Section 9.3 explains the structure of the PSOATransRun program. Sections 9.1 and 9.2 revise and expand our earlier paper [76], which explained an initial version of the TPTP instantiation of PSOATransRun.

### 9.1 ANTLR-based Translator Implementation

The  $\text{PSOA}2L_t$  translator, employed by PSOATransRun to translate PSOA KBs/queries to a target language  $L_t$  as explained in Section 4.2, is implemented

in Java<sup>41</sup> according to the schematic chart shown in Figure 9.1. Specializing the schema to  $L_t$ =TPTP and  $L_t$ =Prolog results in two concrete implementations of the PSOA2TPTP and PSOA2Prolog translators explained in Chapters 7 and 8, respectively. This chart is realized using Version 3 of the ANTLR software tool, which can construct translators from grammatical descriptions containing (e.g., rewriting and generation) actions in a variety of programming languages.<sup>42</sup>

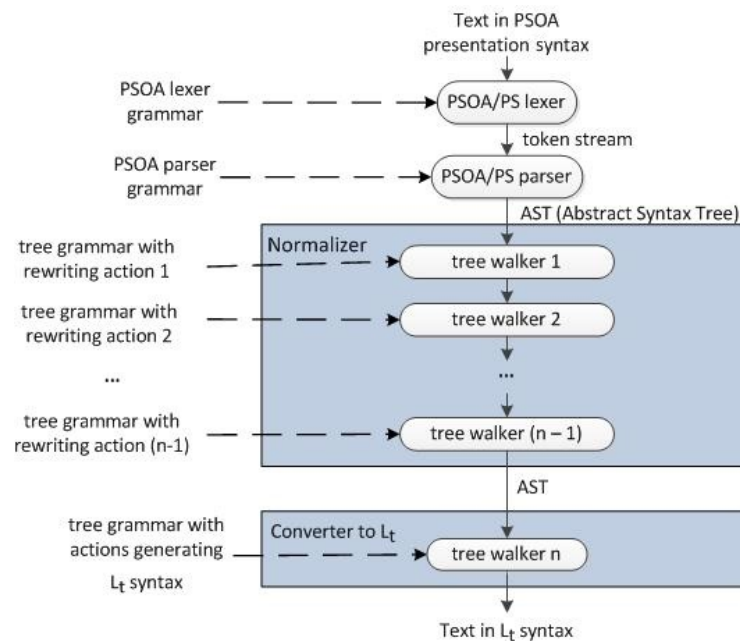


Figure 9.1: Schematic ANTLR-based translator implementation.

As shown in Figure 9.1, the chart consists of a lexer, a parser, and  $n$  tree walkers. The solid arrows represent data flow while the dashed arrows represent the grammars used to generate the translator components shown in the rounded boxes. The translation is performed in the following three phases,

<sup>41</sup>We chose Java for better reusability of components of the implementation in other applications.

<sup>42</sup><http://www.antlr3.org>

using ANTLR's Abstract Syntax Trees (ASTs) as the internal representation of a PSOA KB/query.

1. Parsing text into AST

This phase uses a lexer and a parser generated from corresponding grammars. The lexer breaks up the text presentation of any PSOA input KB/query into a stream of tokens. The parser transforms the token stream into an ANTLR AST.

2. AST-to-AST transformation

This phase uses  $n - 1$  tree walkers generated from tree grammars consisting of the same grammar rules combined with different embedded rewriting actions. Each tree walker implements a PSOA transformation step explained in Chapter 5, where each output AST represents a transformed PSOA KB/query, e.g., as exemplified in Section 5.11. The composition of these tree walkers performs a normalization of the input PSOA KB/query, e.g., as explained in Sections 7.1 and 8.1. The output AST represents a normalized PSOA KB/query for the translation.

3. Converting AST into text

This phase uses one tree walker to convert the AST representation of the normalized PSOA input into the syntax of  $L_t$ .

The ANTLR-generated parser uses an  $LL$  parsing mechanism. It constructs a deterministic finite automaton which can look ahead an arbitrary number of lexer tokens and choose to match one of the candidate patterns in a production. However, the PSOA RuleML grammar in Section 3.3 is a

non-*LL* grammar and cannot be used directly by ANTLR to generate the parser. Hence we transcribe it into an *LL*(1) grammar, which is accepted by ANTLR. Such grammars are efficient in that a single-token lookahead tells the parser which alternative to choose. We follow the standard approach of compilers [4] to do the transcription: (1) ambiguity resolution; (2) elimination of left recursion; (3) left-factoring.

We start with the external-occurrence-restrained sublanguage explained in Section 3.3, where:

- (i) The left-hand side of a subclass formula, the left-hand side of an equality formula, and the OID of a *psoa* term must not be an external term.
- (ii) The predicate of a *psoa* term must not be an external term but a simple term, which is either a constant or a variable.

Before transcribing Section 3.3's grammar, we slightly modify it to incorporate the parenthesis-omitting shortcut for memberships. This is done by making the parenthesized descriptor sequence of the *PSOAIDLESS* production optional as follows:

```
PSOAIDLESS ::= TERM ('(' (TERM* | TUPLEDI*) SLOTDI* ')')?
```

Next we show the transcription of the grammar into an *LL*(1) grammar.

- Ambiguity resolution

In Section 3.3's grammar, the *PSOA* production is ambiguous. The term  $o\#f()$  can be accepted in two ways: (1)  $o$  is accepted as the OID and  $f$  as the predicate term; (2)  $o\#f$  is accepted as a predicate term and  $o\#f()$  as an oidless *psoa* term. To resolve the ambiguity, we realize the above

external-occurrence restraint (ii) by transcribing the `PSOAIDLESS` and `TERM` productions as follows:

```
PSOA ::= PSOAIDLESS | PSOAIDFUL
PSOAIDLESS ::= SIMPLE_TERM ('(' (TERM* | TUPLEDI*) SLOTDI* ')')?
SIMPLE_TERM ::= Const | Var
TERM ::= PSOA | 'External' '(' Expr ')'
```

The `TERM` production is modified since the modified `PSOAIDLESS` production accepts also stand-alone simple terms.

- Elimination of left recursion

Left recursion is one of the main causes of a grammar to become a non-*LL* grammar. A simple example of a left-recursive grammar with a single terminal `'a'` is:

```
P := (P 'a') | 'a'
```

This grammar of non-terminal `P` accepts a string of the form `'a'+`. However, no *LL*-based parser is capable of parsing such a production since it would not be able to consume any token when it applies the first alternative.

In `PSOA/PS`, the production for `PSOAIDFUL`, shown in the following, is implicitly left-recursive since its first non-terminal `TERM` can also be expanded to a `PSOAIDFUL`.

```
PSOAIDFUL ::= TERM '#' PSOAIDLESS
```

In order to eliminate this left recursion, we employ a transcription of the `PSOAIDFUL` production in the following steps.

1. Group everything after the non-terminal `TERM` using a new non-terminal `PSOAOIDFUL_REST` and expand the `TERM` non-terminal in the `PSOAOIDFUL` production:

$$\begin{aligned} \text{PSOAOIDFUL} & ::= \text{PSOA } \text{PSOAOIDFUL\_REST} \\ & \quad | \text{'External' '(' Expr ')'} \text{PSOAOIDFUL\_REST} \quad (1) \\ \text{PSOAOIDFUL\_REST} & ::= \text{'\#'} \text{PSOAOIDLESS} \quad (2) \end{aligned}$$

2. Further expand the `PSOA` non-terminal in the first alternative, yielding production (3):

$$\begin{aligned} \text{PSOAOIDFUL} & ::= \text{PSOAOIDFUL } \text{PSOAOIDFUL\_REST} \\ & \quad | \text{PSOAOIDLESS } \text{PSOAOIDFUL\_REST} \\ & \quad | \text{'External' '(' Expr ')'} \text{PSOAOIDFUL\_REST} \quad (3) \end{aligned}$$

3. Transcribe (3) to remove left recursion.

$$\begin{aligned} \text{PSOAOIDFUL} & ::= \text{PSOAOIDLESS } \text{PSOAOIDFUL\_REST}^+ \\ & \quad | \text{'External' '(' Expr ')'} \text{PSOAOIDFUL\_REST}^+ \quad (4) \end{aligned}$$

Realizing the restraint (i) for `psoa` terms, the grammar is simplified by keeping the first alternative for the `PSOAOIDFUL` production.

$$\text{PSOAOIDFUL} ::= \text{PSOAOIDLESS } \text{PSOAOIDFUL\_REST}^+ \quad (5)$$

- Left factoring

After removing the left recursion, the third step of transcription is left factoring, which makes the grammar an  $LL(1)$  grammar. Left factoring means to combine multiple alternatives into one by merging their common prefix. Following is an example consisting of non-terminals `P`, `Q` and terminals `'a'`, `'b'`:

```

P ::= (Q 'a') | Q
Q ::= 'b'+

```

While parsing a sentence of  $P$ , the parser needs to reach the end of the string to decide on the two alternatives which have a common prefix  $Q$ . Since  $Q$  can match an arbitrary number of tokens, no  $LL(k)$  parser is able to distinguish the alternatives of  $P$ . By merging the common prefix, we can transcribe the production into  $P ::= Q 'a'?$  and the grammar becomes an  $LL(1)$  grammar.

One of the examples of common prefixes in the original parser grammar is the production for `PSOA` where the `PSOAOIDFUL` production, transcribed to (5), starts with `PSOAOIDLESS`. Merging the common prefixes yields

```

PSOA ::= PSOAOIDLESS PSOAOIDFUL_REST*

```

The `ATOMIC` production also needs to undergo transcription to remove common prefixes. Under the restraint (i) for `Equal` and `Subclass`, the transcription yields

```

ATOMIC ::= PSOA (('=' | '##') TERM)?

```

The final implementation of the parser grammar is an ANTLR version of the transcribed grammar with some minor changes.<sup>43</sup>

The above-described  $LL(1)$  grammar is easier for generating the parser but tends to be less reusable and more difficult for future developers to read and evolve. Thus, we embed rewrite rules into the parser grammar to construct

---

<sup>43</sup>The implemented parser grammar was developed based on the original version of `PSOA` and adapted for Version 1.0. It has been using non-terminal names slightly different from the transcribed grammar.

ASTs whose structure is specified using a relatively simple and reusable tree grammar. The ASTs retain the meaningful input tokens and encodes them into a tree structure, while some auxiliary tokens like '(' and ')' are removed. Some “imaginary tokens”, in ANTLR terminology, are also added for easy recognition and navigation.

## 9.2 Combining Translators and Reasoning Engines

In this section we discuss the combinations of the translators with TPTP and Prolog engines into two PSOATransRun instantiations. The PSOATransRun 1.3.1 release<sup>44</sup>, which resulted from this dissertation, uses the Prolog instantiation by default while the TPTP instantiation can be accessed via the command line option `-l tptp`.

### 9.2.1 Combining PSOA2TPTP and a TPTP Engine

The TPTP instantiation combines the PSOA2TPTP translator with the VampirePrime engine<sup>45</sup>. The engine is provided as a Linux executable and is packaged into the PSOATransRun release along with a configuration file. The configuration file declares a polyadic answer predicate `ans` with arities 0, 2, 4, 6, ..., 20. As indicated in Section 7.2.2, query variable names are employed as the first, third, fifth, ... arguments of the answer predicates while the returned variable bindings are the second, fourth, sixth, ... arguments.

---

<sup>44</sup><http://psoa.ruleml.org/transrun/1.3.1/local/>

<sup>45</sup><http://riazanov.webs.com/software.htm>

Since the maximum declared arity of `ans` is 20, our TPTP instantiation currently supports a maximum of 10 free variables in each query. Should queries with more query variables be needed in the future, this limitation can be easily relaxed by adding more declarations for arities 22, 24, ... into the TPTP configuration file.

Each input PSOA KB or query is translated by `PSOA2TPTP` and stored in a file<sup>46</sup>. A Java class is written to store the translated KBs and queries as files, start `VampirePrime` with these files, and parse the output answers into (variable, binding) pairs.

In both of the `PSOATransRun` instantiations, similarly as in Prolog, there are two settings for extracting answers of user queries that might have multiple answers: the one-answer setting and the all-answer setting. The former is the default setting. For each query, it returns the first answer and, if the user requests more, returns one more answer at a time. The latter is configured using the command line option `-a` and returns all answers together for each query. For the use cases and test cases in Chapters 10 and 11, this setting is employed. Since `VampirePrime` can only return all answers together,<sup>47</sup> in the TPTP instantiation, the one-answer setting is achieved by iterating over the returned answers one by one.

---

<sup>46</sup>The output file path of translated KBs can be declared by the user via the command line. If absent, a temporary file path will be used by the system.

<sup>47</sup>`VampirePrime` imposes a limit on the maximum number of answers, which is currently set to 400 by our program, so that a (semantically) infinite number of answers to a query cannot become a cause of non-termination.

### 9.2.2 Combining PSOA2Prolog and a Prolog Engine

The Prolog instantiation combines the PSOA2Prolog translator with the XSB Prolog engine. Unlike VampirePrime, the XSB engine must be pre-installed by users before using the Prolog instantiation of PSOATransRun. The engine is accessed from Java through the InterProlog Java API<sup>48</sup>. There is no limitation to the number of free query variables in this instantiation.

XSB allows to use tabling to memoize intermediate derivation results of specific predicates to achieve better efficiency and prevent infinite loops during query execution. In order to exploit XSB's tabling capability, each translated KB is complemented with tabling directives for the reserved predicates. The three reserved predicates with a fixed arity, `memterm/2`, `sloterm/3`, and `prdsloterm/4` are tabled with their corresponding arity. For the polyadic reserved predicates `tupterm` and `prdtupterm`, we table the arities that correspond to any KB tuple with a maximum length of 10. Hence, if a KB contains atoms whose tuple length is greater than 10, the efficiency may decrease since the mapping of the atom is not tabled during reasoning. This limitation can be relaxed by just increasing the maximum-tuple-length number. Alternatively, it could be entirely removed in the future by tabling the reserved predicates based on their arities in each translated KB.

In the Prolog instantiation, the one-answer setting is directly implemented while the all-answer setting is realized using Prolog's `findall` predicate. For Hornlog-KB-and-query pairs for which the number of answers is infinite (cf. *nat* example in Section 2.1), the one-answer setting can give answers one by one while the all-answer setting runs out of memory without giving any answer.

---

<sup>48</sup><http://interprolog.com>

## 9.3 Overall Program Structure

The Java source of the PSOATransRun system is hosted on GitHub.<sup>49</sup> It is organized into the following four components, each being a Java project:

- PSOACore

This project includes the PSOA/PS lexer, parser, as well as the PSOA-internal transformers.

- PSOA2X

This project includes the implementation of translators from PSOA to other languages using transformers in the PSOACore project as well as containing language-specific converters.

- PSOATransRun

This project (1) provides Java wrappers for runtime engines and (2) combines the translators in the PSOA2X project and the wrapped runtime engines to PSOATransRun instantiations as explained in Section 4.2. It also implements the command-line interface for PSOATransRun.

- PSOATranRunWebService

This project wraps translators and runtime engines into Web services and implements a Web-based GUI for an earlier version of the TPTP instantiation of PSOATransRun.

The following table lists the main Java classes and ANTLR grammar files in the program.

---

<sup>49</sup><https://github.com/RuleML/PSOATransRunComponents>

Table 9.1: Major classes in each component of PSOATransRun.

File name	Description
PSOACore Project	
RDF2PSOA.java	Class for translating RDF/N3 KBs to PSOA presentation syntax
X2PSOA.java	Superclass for translators from other languages or syntaxes to PSOA presentation syntax
PSOAPS.g	Lexer and parser grammars for parsing PSOA RuleML presentation syntax
Concatenater.g	Tree grammar for concatenating multiple KBs into one as explained in Section 5.9
ConjunctiveHeadSplitter.g	Tree grammar for splitting rules having conjunctive conclusions as explained in Section 5.8
ExternalFlattener.g	Tree grammar for flattening nested external built-in function applications as explained in Section 5.7
Objectifier.g	Tree grammar for objectification as explained in Section 5.3
Renamer.g	Tree grammar for renaming the local constants as explained in Section 5.9
Skolemizer.g	Tree grammar for Skolemization as explained in Section 5.4
Descributor.g	Tree grammar for description as explained in Section 5.6
SubclassRewriter.g	Tree grammar for subclass rewriting as explained in Section 5.5
Unnester.g	Tree grammar for unnesting as explained in Section 5.2
PSOAKB.java	Class for storing PSOA KB information and performing KB transformations
PSOAQuery.java	Class for performing PSOA query transformation

File name	Description
PSOA2X Project	
Translator.java	Superclass for all translators from PSOA RuleML to other languages
ANTLRBasedTranslator.java	Subclass of Translator using an ANTLR-based implementation
PrologTranslator.java	PSOA2Prolog translator
TPTPTranslator.java	PSOA2TPTP translator
PrologConverter.g	Tree grammar for Prolog conversion as explained in Section 8.2
TPTPConverter.g	Tree grammar for TPTP conversion as explained in Section 7.2
PSOATransRun Project	
PSOATransRunCmdLine.java	PSOATransRun system for question answering in PSOA RuleML
QueryResult.java	Query result from engines
XSBEngine.java	Wrapper for XSB engine
VampirePrimeEngine.java	Wrapper for VampirePrime engine
PSOATransRunWebService Project	
TranslateResource.java	Web service wrapper for translator
ExecutionResource.java	Web service wrapper for runtime engine

# Chapter 10

## Use Cases

In this chapter, we present two use cases for PSOA RuleML 1.0 realized in PSOATransRun 1.3.1, as practical ‘proof-of-concept’ applications.

The first use case, Port Clearance Rules, explained in Section 10.1 and published in our paper [75], is developed based on the Decision Management Community Challenge of March 2016. The challenge asks for a decision model to determine whether a ship can enter a Dutch port on a certain date based on English rules. We formalized the rules in PSOA, added facts directly in PSOA, and queried the resulting KB for results in PSOATransRun.

The second use case, OfficeProspector, explained in Section 10.2, aims to help companies find office suites suitable for their businesses. In the use case, office data is enriched with public data sets in different modeling paradigms in order to enable user queries, e.g. for finding office suites based on information about building surroundings. The use case KB contains rules and an internal data set as facts, both in PSOA syntax, as well as external data sets – some extracted from an external (CSV) format – imported into KB facts – some translated on-the-fly from an external (N3) syntax.

## 10.1 Port Clearance Rules

This section presents the Port Clearance Rules use case. Section 10.1.1 explains the background of this use case. Section 10.1.2 describes the PSOA RuleML formalization of ten independently provided controlled English rules. Section 10.1.3 complements the PSOA rules by facts and demonstrates queries. Some readers might want to preview the shape of ship facts in Section 10.1.3 before seeing those shapes in rules in Section 10.1.2.

### 10.1.1 Background

The Decision Management (DM) Community<sup>50</sup> has been running Challenges about decision modeling problems<sup>51</sup> since 2014. The DM Community Challenge of March 2016 consisted of creating decision models from the structured text of ten English Port Clearance Rules available online<sup>52</sup>, inspired by the International Ship and Port Facility Security Code. They were originally developed for “The Game of Rules” [66]<sup>53</sup>.

The English of each one of the independently given Port Clearance Rules – as a kind of “legalese”<sup>54</sup> – is moderately controlled, some having a structured ‘if’ part, which supports their formalization. On the other hand, the textual order of the rules – although logically immaterial – does not reflect their most human-readable arrangement in the created decision models. At the time of this writing, there are eight online solutions<sup>55</sup>, some using decision tables or

---

<sup>50</sup><https://dmcommunity.org>

<sup>51</sup><https://dmcommunity.org/challenge/>

<sup>52</sup><https://dmcommunity.org/challenge/challenge-march-2016/>

<sup>53</sup><http://www.buildingbusinesscapability.com/presentations/2015/2200.pdf>

<sup>54</sup>While these core rules are not meant as a law-interpretation challenge, they are rich enough for development into a legal-informatics use case

<sup>55</sup><https://dmcommunity.org/challenge/challenge-march-2016/#Solutions>

decision trees as their main modeling technique.

Based on an analysis of the English Port Clearance Rules and several solutions, we formalized the rules in PSOA RuleML, complemented them by port clearance data facts directly in PSOA RuleML, explored the resulting KB with systematic queries in PSOATransRun, and propose generalized decision models.

### 10.1.2 Formalizing the Port Clearance Rules in PSOA

To arrive at a PSOA RuleML decision model for Port Clearance Rules, the specific object-relational knowledge representation was chosen and the English Port Clearance Rules were formalized in the PSOA RuleML presentation syntax. As indicated in Section 10.1.1, the original textual rule order calls for a more human-readable version, although this entails that – to maintain the correspondence with the original – the PSOA RuleML model’s rule numbers are not consecutive (e.g., the original Rule 4 becomes one of the last rules in PSOA’s textual as well as visual forms). Note that the model-theoretic semantics of PSOA RuleML – not relying on rule order – is associated with the presentation syntax rather than any decision-model visualization.<sup>56</sup>

The PSOA RuleML decision model for Port Clearance Rules is visualized in Figure 10.1, an object-relational And-Or DAG (‘And’ branches are horizontally cross-linked) with rule names as nodes, e.g. Rule 2, and conclusion predicates as side labels of nodes, e.g. `:MayEnterDutchPortUnloaded`. For the not side-labeled nodes Rule 9, 1&5, and 4, the root-class predicate `Top` is understood,

---

<sup>56</sup>While these two notions of “model” are quite unrelated, PSOA’s model theory applied to PSOA’s Port Clearance Rules formalized in this subsection provides a semantics for this decision model. Likewise, for any other decision model in PSOA RuleML.

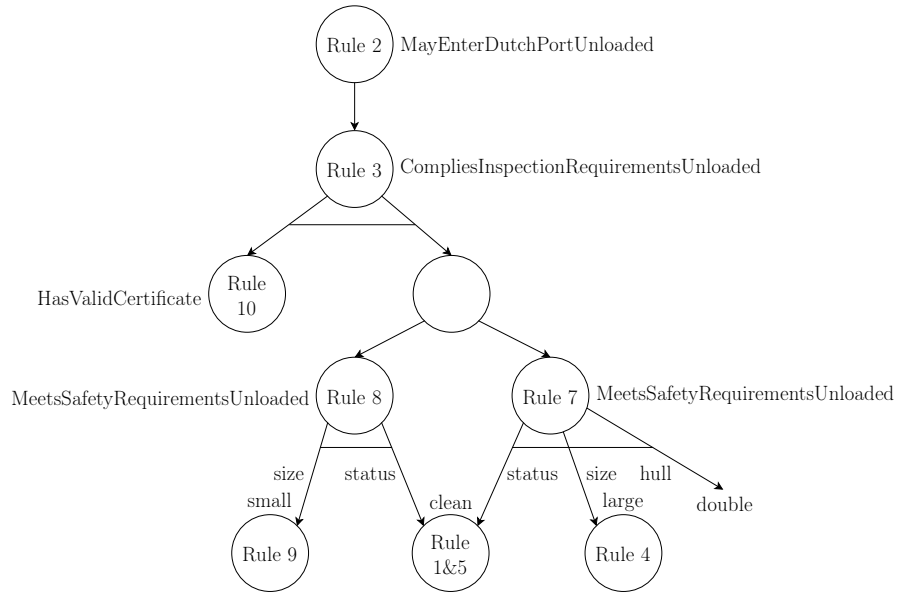


Figure 10.1: Visualization of PSOA RuleML decision model for Port Clearance Rules.

while slot names, e.g. `:size`, and slot fillers, e.g. `:small`, are shown as, respectively, labels of incoming arcs and top labels of the rule nodes (for the slot name `:hull` the filler `:double` does not require any further rule).

The blank, unlabeled node represents the only ‘Or’ branch in this model, where Rules 8 and 7 are – operationally speaking – ‘pre-invoked’ via the conclusion predicate `:MeetsSafetyRequirementsUnloaded`, having conditions with a first conjunct immediately determining whether the slot `:size` is `:small` or `:large`, so that only either Rule 8 or Rule 7, respectively, can be ‘fully invoked’, causing *near-deterministic* behavior.

The model is object-relational in that the upper part running to the conclusions of Rules 8 and 7 involves unary relations applied to ships while the lower part from the conditions of Rules 8 and 7 downward involves frames with ship OIDs described by slots. Hence, the upper ‘And’ branch between Rule 10 and the blank node corresponds to an explicit `And` in PSOA/PS, while the

two lower ‘And’ branches between Rules 9 and 1&5 as well as between Rule 1&5, Rule 4 and slot `:hull` being `:double` correspond to the implicit conjunction of PSOA RuleML slots<sup>57</sup>, which can be made explicit by the description transformation in Section 5.6.

We now describe the rules, top-down, in five subgroups, referring to the first half of Appendix A for the complete PSOA KB.

In the **first subgroup** of rules, the original Rule 2 is the (DAG-)root of the decision model, passing on its ship argument to Rule 3, which conjoins the two requirements for compliance:

2. An unloaded ship may only enter a Dutch port if the ship complies with the requirements of the Inspection for unloaded ships.
3. A ship must comply with the requirements of the Inspection for unloaded ships if the ship complies with all of the following: a) the ship meets the safety requirements for unloaded ships; b) the ship has a certificate of registry that is valid.

In PSOA/PS, the subgroup is formalized as follows:

```
% Main relational rule invokes inspection rule for certificate And safety

% Rule 2
Forall ?s (
  :MayEnterDutchPortUnloaded(?s) :-
    :CompliesInspectionRequirementsUnloaded(?s)
)
```

---

<sup>57</sup>We will later see that, of the conjoined slots, `:size` occurs directly in a `:Ship` frame while `:status` and `:hull` occur in its embedded `:ShipHold` frame.

```

% Rule 3
Forall ?s (
  :CompliesInspectionRequirementsUnloaded(?s) :-
    And(:IsValidCertificate(?s)
        :MeetsSafetyRequirementsUnloaded(?s))
)

```

While the original Rule 2's conclusion-side word “may” – is part of the predicate name `:MayEnterDutchPortUnloaded` – creates a top-level context that can be given a modal – specifically, a deontic – interpretation, the original Rule 3's conclusion-side “must” is not again part of the predicate name `:CompliesInspectionRequirementsUnloaded`. Both rules are relational, on the Datalog level of expressiveness (predicates only have a variable, `?s`, no function application, as their argument). The `:Ship`-type test for `?s` is postponed to the later object-centered rules, where `:Ship` becomes a class.

The **second subgroup** consists of just the original Rule 10, which looks up a ship's certificate-validity date and checks whether the current date is before that:

10. A ship's certificate of registry must be considered valid if the date up to which the registration is valid of the certificate of registry is after the current date.

In PSOA/PS, this subgroup has two formal options (local/fixed date):

```

% Object-relational certificate rule compares ship's registry expiration
% with current date

% Rule 10
Forall ?s ?d ?e (

```

```

:HasValidCertificate(?s) :-
    And(?s#:Ship(:registryExpirationDate->?e)
%       phys:currentDate(?d) % Uncomment for local date (deployment)
       :currentDate(?d)      % Uncomment for fixed date (reproducibility)
       phys:lessThanDate(?d ?e))
)

```

As in Rule 3, the word “must” is not explicit in the predicate name here or later on. Rule 10 transits from the relational to the object-centered paradigm: The relational conclusion argument `?s` becomes, in the first condition conjunct, the OID of class `:Ship` of a query frame with a `:registryExpirationDate` slot, whose filler is a date encoded as a Hornlog-expressiveness-level (constructor-)function application `phys:date(year month day)` – this depth-1 nesting could be easily eliminated, hence the rule stays on what we refer to as the (Datalog-transformable) *near-Datalog* expressiveness level.<sup>58</sup> The second conjunct queries the current date in that encoding, optionally yielding the local date or a fixed date.<sup>59</sup> The third conjunct then checks `phys:lessThanDate` between the expiration date and the current date.

The **third subgroup** contains the complementary original Rules 8 resp. 7, which – for `:MeetsSafetyRequirementsUnloaded` of small resp. large ships – give two resp. three condition conjuncts, where the b) conjunct is shared (leading to the DAG structure of the decision model visualized in Figure 10.1<sup>60</sup>):

---

<sup>58</sup>The ternary function `phys:date` becomes unnecessary when dividing the slot `:registryExpirationDate` into three slots `:registryExpirationYear`, `:registryExpirationMonth`, and `:registryExpirationDay`.

<sup>59</sup>While for deployment the local date is computed by the `phys:currentDate` predicate from the physics library, imported via `http://psoa.ruleml.org/lib/phys.psoa`, for (test-suite) reproducibility a fixed date is looked up as a `:currentDate` ground fact.

<sup>60</sup>For layout reasons, in the DAG visualization the b) conjunct is shown as the second ‘And’ branch of Rule 8 but as the first ‘And’ branch of Rule 7.

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

In PSOA/PS, this subgroup has the following formalization:

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

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

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

Rules 8 and 7 each includes a disjunct of the original Rule 6, which uses two intermediate predicates that are not needed for realizing the decision logic. Both Rules 8 and 7 transit from the relational to the object-centered paradigm

with their frame conditions: The relational conclusion argument `?s` becomes the OID of class `:Ship` of a frame with `:size` and `:hold` slots, where the slot value of `:hold` is an embedded frame with OID `?h` of class `:ShipHold` and slots `:status` and `:clean`. The `:hold`-embedded `:ShipHold` frame – corresponding to an embedded `:ShipHold` function application – can be regarded as raising the expressiveness level to Hornlog, but – being only a depth-1 nesting – can be easily unnested, hence stays on the near-Datalog level. The nested frames of Rules 8 and 7 in `PSOATransRun` will be transformed into conjunctions via unnesting followed by description. For example, this is the unnesting result for Rule 7’s condition (the two `?h` occurrences are used for the referenced OID of the extracted frame moved to the top-level and for the referencing OID in the `:hold`-filler position of the main frame):

```
And(
  ?h#:ShipHold(:status->:clean :hull->:double)
  ?s#:Ship(:size->:large :hold->?h)
)
```

The subsequent description result is as follows (all but one of the `?h` and `?s` occurrences just use the root class `Top`):

```
And(
  And(?h#:ShipHold ?h#Top(:status->:clean) ?h#Top(:hull->:double))
  And(?s#:Ship ?s#Top(:size->:large) ?s#Top(:hold->?h))
)
```

The **fourth subgroup** includes the similar original Rules 9 resp. 4, which determine whether a ship is small resp. large:

9. A ship must be categorized as small if the total length of the ship is less than 80 meters.

4. A ship must be categorized as large if the total length of the ship is at least 80 meters.

In PSOA/PS, this subgroup’s formalization is:

```
% Object-centered (except for math) rules to get qualitative size by
% thresholding length
```

```
% Rule 9
```

```
Forall ?s ?l (
  ?s#Top(:size->:small) :-
    And(?s#:Ship(:totalLength->?l)
      math:lessThan(?l 80))
)
```

```
% Rule 4
```

```
Forall ?s ?l (
  ?s#Top(:size->:large) :-
    And(?s#:Ship(:totalLength->?l)
      math:greaterEq(?l 80))
)
```

Both rules are object-centered except for the relational `math:lessThan` and `math:greaterEq` calls in their second conjuncts.<sup>61</sup> Note that units of measure – here, “meter” and, in Rule 5, “mg dry weight per cm” – are omitted on this near-Datalog level of expressiveness, but could become Hornlog function applications in slot fillers – here, `:m(?l)` and, in Rule 5, `:mgDryWeightPerSqcm(?c)`.

The **fifth subgroup** consists of the original Rules 1 and 5, where the condition of Rule 1 negates the conclusion of Rule 5:

---

<sup>61</sup>The predicates having the `math:` prefix are defined in the imported mathematics library <http://psoa.ruleml.org/lib/math.psoa>. They are shortcuts for external built-in calls in PSOA.

1. The hold of a ship must be considered clean if the hold does not contain remainders of cargo.
  
5. A ship's hold contains remainders of cargo if the residual cargo measurement is higher than 0.5 mg dry weight per cm<sup>2</sup>.

In PSOA/PS, this subgroup is formalized as one combined Rule 1&5:

```
% Object-centered (except for math) rule to get qualitative status by
% thresholding residual

% Rule 1&5 (combines Rule 1 and Rule 5)
Forall ?h ?c (
  ?h#Top(:status->:clean) :-
    And(?h#:ShipHold(:residualCargoMeasurement->?c)
      math:lessEq(?c 0.5))
)
```

By propagating the negation into Rule 5's condition, the negation of a `math:greaterThan` call is simplified to a `math:lessEq` call, so that negation is eliminated.<sup>62</sup> The resulting Rule 1&5 is again object-centered except for the relational `math:lessEq`.

### 10.1.3 Enrichment by Port Clearance Facts and Queries

Since the March 2016 DM Community Challenge has introduced only ship rules, we have developed ship facts for systematic testing of the rules in Section 10.1.2, as shown in the second half of Appendix A. Each

---

<sup>62</sup>Coincidentally, while for PSOATransRun 1.3.1 the implementation of negation is restricted to numeric disequality (`math:notEq`), in a future PSOATransRun a Negation-as-failure (Naf) primitive could be mapped to the underlying Prolog's Naf.

ship fact is a frame having an OID `:ship $k$` ,  $k = 1, 2, \dots$ , and three slots, `:registryExpirationDate`, `:totalLength`, and `:hold`. The value of the `:hold` slot is an embedded frame having an OID `:h $k$` ,  $k = 1, 2, \dots$ , and two slots, `:residualCargoMeasurement` and `:hull`. Following is an example of a ship fact.

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

The ship facts are covering all cases with qualitative slot-filler distinctions. Each ship fact comes with a comment on whether the ship instance should be allowed to enter a Dutch port, as of 2017-05-06, as well as the main reason.

In the following, we pose representative copy&paste-ready queries to the KB and demonstrate the answers obtained by `PSOATransRun`. The queries that answer Port Clearance questions such as for the DM Challenge are ground queries using the top-level predicate `:MayEnterDutchPortUnloaded` applied to specific ship instances, e.g. to `:ship1` and `:ship7`, as shown below.

```
> :MayEnterDutchPortUnloaded(:ship1)
No

> :MayEnterDutchPortUnloaded(:ship7)
Yes
```

We can also pose a generalized, symbolic-execution-style non-ground query to `:MayEnterDutchPortUnloaded`, using output variable `?w` to deduce ships

that may enter a Dutch port. Such non-ground queries can make predicates behave non-deterministically: The multiple (here, five) ?w-answer bindings are shown as full IRIs expanded from the ‘:’-prefixed :ship*k* abbreviations in the KB.<sup>63</sup>

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

The supporting computations for these top-level decision-making questions can be revealed by specialized object-centered and relational queries. Let us thus explore queries over the KB in a bottom-up traversal of parts of the DAG in Figure 10.1, explaining the answers obtained by PSOATransRun.

We start with the object-centered-query portion of this traversal. The following query asks whether :h7 is a ship hold and it is clean, which can be proved by Rule 1&5 and the :h7 frame embedded inside the :ship7 fact.

```
> :h7#:ShipHold(:status->:clean)    % Query centered on OID :h7
Yes
```

Building on that, a query then asks whether the hold ?h of :ship7 is clean. This can be proved by first binding ?h to (the expanded IRI of) the hold :h7 of :ship7 and then check whether :h7 is clean using the previous (sub)query.

```
> :ship7#:Ship(:hold->?h#:ShipHold(:status->:clean)) % Main OID is :ship7
?h=<http://psoa.ruleml.org/usecases/PortClearance#h7>
```

---

<sup>63</sup>For queries with variable-binding results, Yes answers will be understood. Since multiple variable-binding answers – like KB clauses – are semantically unordered, implementations such as PSOATransRun can choose any enumeration order.

The next query asks whether `:ship7` is a large ship, which can be proved by Rule 4 using its `:totalLength` slot in the `:ship7` fact.

```
> :ship7#:Ship(:size->:large)
```

Yes

An extended query then asks whether `:ship7` is a large ship and its hold is clean and double hulled. The size and hold status of `:ship7` have been proved by the previous two (sub)queries while the hold-hull information can be proved directly through the `:ship7` fact.

```
> :ship7#:Ship(:size->:large :hold->?h#:ShipHold(:status->:clean :hull->:double))
?h=<http://psoa.ruleml.org/usecases/PortClearance#h7>
```

We now proceed to the relational-query portion of this traversal. The following query asks whether `:ship7` meets the safety requirements. It can be proved by Rule 7 and the previous (sub)query.

```
> :MeetsSafetyRequirementsUnloaded(:ship7) % Query with unary safety relator
```

Yes

The sibling query in Figure 10.1 asks whether `:ship7` has a valid certificate, which can be proved by Rule 10 based on its `:registryExpirationDate` slot in a fact.

```
> :HasValidCertificate(:ship7) % As of 2017-05-06
```

Yes

The penultimate query of this traversal asks whether `:ship7` complies with the requirements of the inspection for unloaded ships. It can be proved by Rule 3 based on the previous two (sub)queries.

```
> :CompliesInspectionRequirementsUnloaded(:ship7) % As of 2017-05-06
```

Yes

The top-level query `:MayEnterDutchPortUnloaded(:ship7)` can now be proved by Rule 2 and the previous (sub)query.

Next we explore some non-ground queries that can extract interesting content from the KB.

The following query asks for the `:size` of `:ship1`.

```
> :ship1#:Ship(:size->?z)
?z=<http://psoa.ruleml.org/usecases/PortClearance#small>
```

The subsequent query asks for any large ship whose hold is clean. Here, the stand-alone ‘?’ is an (anonymous) variable whose bindings are not needed.

```
> ?s#:Ship(:size->:large :hold->?#:ShipHold(:status->:clean))
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship7>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship13>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship10>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship14>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship6>
```

The next query asks for any ship that is large and has a valid certificate.

```
> :IsValidCertificate(?s#:Ship(:size->:large))
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship5>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship14>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship6>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship9>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship13>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship7>
```

The final query asks for any ship that is small and meets the safety requirements for unloaded ships.

```
> :MeetsSafetyRequirementsUnloaded(?s#:Ship(:size->:small))
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship4>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship1>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship12>
?s=<http://psoa.ruleml.org/usecases/PortClearance#ship2>
```

## 10.2 OfficeProspector

This section presents the OfficeProspector<sup>64</sup> use case. Section 10.2.1 explains the modeling of knowledge in the use case, including a partonomy-taxonomy. Sections 10.2.2 and 10.2.3 explain data facts as well as rules. Section 10.2.4 demonstrates example queries for various use scenarios.

### 10.2.1 Knowledge Modeling

The knowledge in the OfficeProspector use case is stored as facts in both the relational and object-centered paradigms. It is queried and exchanged by rules using mainly the object-centered paradigm, which provides high flexibility, e.g. for range queries. It can be classified into internal knowledge, which is created by the developers of the use case, and external knowledge, which is obtained from external data sources and incorporated via `Import` statements. In the rest of this chapter, constants used in the internal knowledge are prefixed with “:”, expanded to `<http://psoa.ruleml.org/usecases/OfficeProspector/>`, while constants used in the external knowledge have the prefix “`opd:`”, expanded to `<http://psoa.ruleml.org/usecases/OfficeProspector/data-external#>`.

The classes used in our object-centered modeling of knowledge can be or-

---

<sup>64</sup>`<http://psoa.ruleml.org/usecases/OfficeProspector/`

```

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

```

Figure 10.2: Partonomy-taxonomy for office suites.

ganized into a partonomy-taxonomy<sup>65</sup> shown in Figure 10.2. The subsumption between classes is expressed using normal indentation with only blanks, e.g. `Office` and `ClosedOffice`, while the part-whole relation between classes is expressed using an indentation with the `|_` notation, e.g. `Suite` and `Office`.

The partonomy has four different levels: neighborhoods, buildings, suites, and spaces inside suites. Objects of directly related classes in the partonomy are connected via `:hasPart` and `:partOf` slots. A neighborhood is a geographically localized community within a city or town. A building in our KB is used to model an office building containing office suites for rent. This modeling permits later refinement through a distinction between apartment and

---

<sup>65</sup>“Partonomy” is often called “meronomy” [47].

office buildings (which would allow the use case to be extended for also renting apartment buildings). A suite in our model is the main substructure in a building that can be rented by users. Spaces inside a suite can be specified by their functionality – using `Office`, `MeetingSpace`, `Kitchen`, and `Reception` classes – their openness – using (closed) `Room` and `OpenSpace` classes – or in a combined manner – using, e.g., `ClosedOffice`.

Besides using `:hasPart` and `:partOf` slots, the interior 2D structure of a suite is further specified using orientation slots to connect objects denoting spaces inside the suites. For each suite, a pair of orthogonal axes X and Y can be established while a visitor walks through the entrance – the Y axis orients towards the direction that he/she faces (which will be called `:plusY`) and the X axis orients towards the right-hand side direction (which will be called `:plusX`).<sup>66</sup> Given an object representing a space inside a suite, its four orientation slots `:plusX`, `:plusY`, `:minusX`, and `:minusY` point to adjacent spaces inside the suite located on the corresponding side of this space object according to the directions of axes. For each space, the facts store only `:plusX` and `:plusY` slots while `:minusX` and `:minusY` slots are derived by rules using `:plusX` and `:plusY` slots attached to its adjacent spaces. This modeling permits future refinement for spatial reasoning. Figure 10.3 gives an example floor plan of an office suite which we will call `:suite1`.<sup>67</sup> Figure 10.4 gives example facts that describe the interior structures of three suites, including `:suite1`, together with some additional information such as the available utilities. In the PSOA facts, `:space1`, ..., `:space5` denote the spaces labeled with the

---

<sup>66</sup>For simplicity, we assume that each suite has only one main entrance.

<sup>67</sup>Modified from <http://www.conceptdraw.com/solution-park/building-office-layout-plans>

same name in Figure 10.3. The positional arguments of `:Suite` constitute a (dependent) tuple of the floor number, room number, area in square meters, and monthly rent in Canadian dollars (CAD). They are followed by slots `:entrance`, `:hasPart`, etc. that describe the inner structure and facilities of the suite. Besides the orientation slots explained above, each space frame has one independent tuple representing its spans on the X and Y axes.

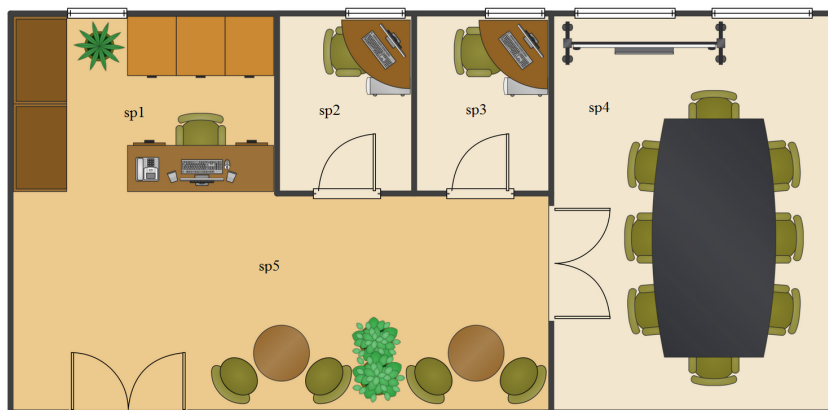


Figure 10.3: Example floor plan of Office Suite 101.

## 10.2.2 Data Facts

The data facts in the use case mainly consist of one internal data set and three external data sets, which will be explained in the following sub-subsections. The external data sets are from the city of Toronto and permit extension for other cities in the future. These data sets are included into the use case using `Import` declarations in the main KB.

```

:suite1#:Suite(1 101 226 2000
    :entrance->:door1
    :hasPart->:space1 :hasPart->:space2 :hasPart->:space3
    :hasPart->:space4 :hasPart->:space5
    :totalClosedOffices->2
    :hvac->:h1
    :utility->:phone :utility->:internet)
:door1#:Door(:plusY->:space5)
:space1#:Reception(-[6.91 4.6] :plusX->:space2)
:space2#:ClosedOffice(-[3.45 4.6] :plusX->:space3)
:space3#:ClosedOffice(-[3.45 4.6] :plusX->:space4)
:space4#:ClosedMeetingSpace(-[7.35 10.5])
:space5#:OpenWorkSpace(-[9.6 5.61] :plusX->:space4
    :plusY->:space1 :plusY->:space2 :plusY->:space3)
:h1#:HVAC(:rating->op-rtg:advanced)

:suite2#:Suite(2 205 150 1500
    :entrance->:door2
    :hasPart->:space6 :hasPart->:space7 :hasPart->:space8 :hasPart->:space9
    :totalOpenOffices->2
    :hvac->:h2
    :utility->:phone :utility->:internet)
:door2#:Door(:plusY->:space6)
:space6#:OpenSpace(-[24 1] :plusY->:space7 :plusY->:space8 :plusY->:space9)
:space7#:OpenOffice(-[6.2 5.1] :plusX->:space8)
:space8#:ClosedKitchen(-[6.2 5.1] :plusX->:space9)
:space9#:OpenMeetingSpace(-[10.3 5.1])
:h2#:HVAC(:rating->op-rtg:regular)

:suite3#:Suite(3 303 100 1000
    :entrance->:door3
    :hasPart->:space10 :hasPart->:space11 :hasPart->:space12
    :totalClosedOffices->2
    :hvac->:h3
    :utility->:phone :utility->:internet)
:door3#:Door(:plusY->:space10)
:space10#:OpenSpace(-[24 1] :plusY->:space11 :plusY->:space12)
:space11#:ClosedOffice(-[7.2 6.1] :plusX->:space12)
:space12#:ClosedOffice(-[7.2 6.1])
:h3#:HVAC(:rating->op-rtg:basic)

```

Figure 10.4: Example of facts describing interior structures of office suites.

Table 10.1: Slots of building objects.

<code>:hasPart</code>	any part, e.g., a suite, of the building
<code>:totalElevators</code>	total number of elevators in the building
<code>:availableSuites</code>	available suites for rent in the building
<code>:totalSuites</code>	total number of suites in the building
<code>:frontDesk</code>	front desk information object of the building
<code>:parking</code>	parking information object of the building

### 10.2.2.1 Internal Data Set

The internal data set<sup>68</sup> contains facts describing buildings, office suites, and spaces inside suites. Associated with real addresses, the facts are generated by a data generator program using other random information, e.g. the monthly rent and contact information is generated using a uniform distribution.

Each building fact is modeled as a tuple+slotted psoa atom having a tuple of seven arguments. The first five arguments describe the address of the building, including street number, street, city, province, and country. The sixth and seventh arguments describe the completion year and the contact information of the building. The slots of a building object are shown in Table 10.1.

Figure 10.5 is an example fact of a building `:bldg1` in which the suites `:suite1`, `:suite2`, and `:suite3` in Figure 10.4 locate.

```
:bldg1#:Building("876" "Adelaide St W" "Toronto" "ON" "CA"
 1983 "416-787-830"
 :frontdesk->:f1
 :hasPart->:suite1
 :hasPart->:suite2
 :hasPart->:suite3
 :totalElevators->1
 :availableSuites->3
 :totalSuites->10)
```

Figure 10.5: A building fact.

<sup>68</sup><http://psoa.ruleml.org/usecases/OfficeProspector/office.psoa>

### 10.2.2.2 External Data Sets

The external data sets contain facts obtained from external sources, including the following.

- A relational data set contains information of Toronto’s 140 neighborhoods.<sup>69</sup> It is extracted from multiple data sets from Toronto’s Web portal of open data.<sup>70</sup> Each relationship in the data set has the predicate `opd:Neighborhood` and seven arguments, including the neighborhood’s ID, name, total area, total population, total major crime incidents, total number of businesses, total number of employments, and total number of public transport access points. The ID argument serves as a ‘primary key’ of the relationship. The following is an example fact of the Niagara neighborhood.

```
opd:Neighborhood(82 "Niagara" 3.1 21000 417 591 15621 64) (10.1)
```

- A relational data set contains coordinates of addresses in WGS84<sup>71</sup>, the reference coordinate system used by the Global Positioning System.<sup>72</sup> The data are extracted from the address point data of Toronto<sup>73</sup>. Each relationship in the data set has the predicate `opd:Geocode` and eight arguments, including street number, street address, city, province, country name, latitude, longitude, and neighborhood ID. The neighborhood ID argument denotes the ID of the neighborhood that the address belongs to. It can be seen as a ‘foreign key’ that refers to the neighborhood relationships in the above data set. The following

---

<sup>69</sup><http://psoa.ruleml.org/usecases/OfficeProspector/neighborhood.psoa>

<sup>70</sup><https://web.toronto.ca/city-government/data-reports-maps/open-data/>

<sup>71</sup><https://confluence.qps.nl/pages/viewpage.action?pageId=29855173>

<sup>72</sup><http://psoa.ruleml.org/usecases/OfficeProspector/geocode.psoa>

<sup>73</sup>[http://opendata.toronto.ca/gcc/address\\_points\\_wgs84.zip](http://opendata.toronto.ca/gcc/address_points_wgs84.zip)

is an example fact which gives the details of the address of the building `:bldg1` shown above. This information can be used to derive that the building `:bldg1` is in the neighborhood with the number 82, which corresponds to the Niagara neighborhood as formalized in (10.1).

```
opd:Geocode("876" "Adelaide St W" "Toronto" "ON" "CA"
            43.64338054290 -79.41296295790 82) (10.2)
```

- An object-centered data set in N3 syntax contains information of amenities in Toronto.<sup>74</sup> The data set is a fragment of LinkedGeoData<sup>75</sup> containing the information of public transport access points, restaurants, and hotels that are modeled as geospatial points in the data set.

The following is an example fact.

```
ns1:node2111375373 rdf:type    ns3:Restaurant, ns2:Node ;
  rdfs:label      "Asaka Sushi" ;
  foaf:homepage   <http://www.asakasushi.ca/> ;
  ...
  geo:lat         43.5633118 ;
  geo:long        -79.572139 ;
  ns3:cuisine     "japanese" ;
  ns3:opening_hours "Mo-Th 11:00-22:00;Fr-Sa 11:00-20:30;Su 12:00-22:00".
```

---

<sup>74</sup><http://psoa.ruleml.org/usecases/OfficeProspector/amenitiesToronto.n3>

<sup>75</sup><http://linkedgeodata.org/>

### 10.2.3 Rules

Rules in the use case can be categorized as integration rules, vocabulary-extension rules, slot-constraining rules, and miscellaneous rules, e.g. for providing range queries. The integration rules expose atoms in the data sets as frames using our own vocabulary. The vocabulary-extension rules derive new slots from the primitive slots obtained from data facts. Slot-constraining rules expand slot-constrained queries. In the following sub-subsections we will explain these different kinds of rules.

#### 10.2.3.1 Object-Centered Integration Rules for Knowledge Enrichment

The integration rule for the neighborhood data set, shown in Figure 10.6, exposes a neighborhood relationship in the data sets as a frame. The OID of the frame is constructed from the input relationship using an `:oidkey` function applied to the predicate `opd:Neighborhood` and the `?neighborhoodNum` argument, which is the primary key of the relationship. Unlike the `_oidcons` function used in static/dynamic objectification (cf. Definition 5.8), which constructs an OID for each relationship from its predicate and *all* of its arguments, the `:oidkey` function constructs OIDs for each relationship from its predicate and the arguments that constitute *a simple or composite key* of the relationship.

```

Forall ?id ?name ?area ?popul ?crimes ?tb ?te ?ts ?cr ?trd
(
  :oidkey(opd:Neighborhood ?neighborhoodNum)#:Neighborhood(
    :name->?name
    :totalBusiness->?tb
    :totalEmployments->?te
    :totalPublicTransAccesses->?ts
    :crimeRate->?cr
    :publicTransDensity->?trd
  ) :-
  And(
    opd:Neighborhood(?neighborhoodNum ?name ?area ?popul ?crimes
      ?tb ?te ?ts)
    ?cr = External(func:numeric-multiply(100
      External(func:numeric-divide(?crimes ?popul))))
    ?trd = External(func:numeric-multiply(100
      External(func:numeric-divide(?ts ?popul))))
  )
)
)

```

Figure 10.6: Integration rule for buildings.

The first four slots of the conclusion frame are directly from arguments of the condition relationship while the last two slots are obtained through arithmetic computation of the arguments.

The integration of the building data set is done in two rules. The first rule shown in the following simply expresses the positional arguments of a building object as slots attached to the same object. This rule would allow these arguments to be queried in both positional or slotted forms.

```

forall ?b ?streetnum ?street ?city ?province ?country
    ?yearBuilt ?contact
(
  ?b#Top(
    :streetNum->?streetnum
    :street->?street
    :city->?city
    :prov->?province
    :country->?country
    :yearBuilt->?yearBuilt
    :contact->?contact
  ) :-
  ?b#:Building(?streetnum ?street ?city ?province ?country
    ?yearBuilt ?contact)
)

```

Figure 10.7: Integration rule for buildings.

Another integration rule for buildings uses a join between the address arguments of a building and the corresponding `Geocode` relationship to add two slots `:coord` and `:partOf` to the building `?b`. The value of the `:coord` slot is a `:Point` application to the latitude and longitude. The value of the `:partOf` slot is an OID having the same form as those constructed in the conclusion of the above neighborhood rule, thus creating the connection between building objects and neighborhood objects.

The integration rule for suites is similar to the first one for buildings.

The integration rules for the object-centered data set extracts the subset of slots from the imported `LinkedGeoData` into slots of that are needed for our use case. In these rules, the `:oidlgd` function is employed to construct an

OID in our use case from each LinkedGeoData OID. The following are some examples. Each slot from the original OID is transferred to the new OID using our own vocabulary in a separate rule.

```
Forall ?o ?lat ?long
(
  :oidlgd(?o)#GeoEntity(:coord->:Point(?lat ?long)) :-
    ?o#lgdmeta:Node(geo:lat->?lat geo:long->?long)
)
```

```
Forall ?o ?desc
(
  :oidlgd(?o)#Top(:name->?desc) :-
    ?o#Top(rdfs:label->?desc)
)
```

```
Forall ?o ?cus
(
  :oidlgd(?o)#Top(:cuisine->?cus) :-
    ?o#Top(lgdo:cuisine->?cus)
)
```

### 10.2.3.2 Object-Centered Vocabulary-Extension Rules

The vocabulary-extension rules derive new slots from the primitive slots obtained from the integration rules. Some examples will be shown in the next few paragraphs.

The following rule obtains the area of a (rectangular) space from its tuple of spans on the X and Y axes measured in meters. The slot value of `:area` is a `:measure` function application with two arguments denoting the quantity

and the unit-of-measure. This representation allows application users to pose query using different kinds of unit-of-measures.

```
Forall ?o ?ar
(
  ?o#Top(:area->:measure(?ar :sqm)) :-
    And(
      ?o#:Space(-[?x ?y])
      ?ar = External(func:numeric-multiply(?x ?y))
    )
)
```

The following rules derive `:minusX` and `:minusY` slots from `:plusX` and `:plusY` slots.

```
Forall ?o1 ?o2
(
  ?o2#Top(:minusX->?o1) :-
    ?o1#Top(:plusX->?o2)
)
```

```
Forall ?o1 ?o2
(
  ?o2#Top(:minusY->?o1) :-
    ?o1#Top(:plusY->?o2)
)
```

The following rule derives an `:adjacent` slot of `?o1` if it has either one of the `:plusX`, `:plusY`, `:minusX`, `:minusY` slots.

```
Forall ?o1 ?o2
(
```

```

?o1#Top(:adjacent->?o2) :-
  Or(
    ?o1#Top(:plusX->?o2)
    ?o1#Top(:plusY->?o2)
    ?o1#Top(:minusX->?o2)
    ?o1#Top(:minusY->?o2)
  )
)

```

### 10.2.3.3 Slot-Constraining Rules

In this sub-subsection we explain slot-constraining rules, which expand slot-constrained queries with user-provided slot name/filler specifications to retrieve OIDs.

We start with a general axiomatization of a PSOA language extension, Slot-Constrained PSOA RuleML, employed for our use case in the office domain. The following rule expands a slot with an `:atmost`-constrained integer parameter. The rule condition retrieves any object `?o` that has a slot `?p` and the slot filler `?v` fulfills the specified constraint `:atmost(?valcstrt)`.

```

Forall ?o ?p ?v ?valcstrt
(
  ?o#Top(:constrain(?p)->:atmost(?valcstrt)) :-
    And(
      External(pred:is-literal-integer(?valcstrt))
      ?o#Top(?p->?v)
      External(pred:numeric-less-than-or-equal(?v ?valcstrt))
    )
)

```

The value of the `:atmost` constraint can also have a `:measure` (constructor-)function application as its argument. The corresponding constraining rule is analogous to the previous rule, except that the second conjunct requires the slot filler to be a `:measure` function application with the same unit-of-measure.

```
Forall ?o ?p ?v ?valcstrt
(
  ?o#Top(:constrain(?p)->:atmost(:measure(?valcstrt ?unit))) :-
  And(
    External(pred:is-literal-integer(?valcstrt))
    ?o#Top(?p->:measure(?v ?unit))
    External(pred:numeric-less-than-or-equal(?v ?valcstrt))
  )
)
```

Besides these two variants of (unmeasured and measured) `:atmost` constraints, we have axiomatized corresponding `:atleast` constraints, while similar `:lessthan` and `:greaterthan` constraints could be easily added to a library of (unmeasured and measured) arithmetic slot-constraining rules.

The above two rules allow users to ask slot-constrained queries in the form of

```
:bldg1#:Building(:constrain(:yearBuilt)->:atleast(1980))
```

and

```
?s#:Suite(:constrain(:monthlyRent)->:atmost(:measure(1500 :cad)))
```

These queries employ slots in the form of `:constrain(p)->cf(v)`, where *p* is the name of the slot that is constrained, *cf* is the name of a function

representing a specific constraint, e.g., `:atleast`, and  $v$  is the constraint value. The first query is ground, asking whether `:bldg1` is built after 1980. It can be proved by the fact in Figure 10.5, the rule in Figure 10.7 and the `:atleast` counterpart of the unmeasured slot-constraining rule above. The second query is non-ground, asking for any suite `?s` whose monthly rent is at most CAD 1500. It can be answered by the facts in Figure 10.4, a suite-integration rule, similar to Figure 10.7, that re-expresses the positional monthly-rent argument as a slot, and the `:atmost` version of the unmeasured slot-constraining rule above. The answer bindings are shown in the following, where the abbreviated IRIs are expanded to full IRIs.

```
?b=<http://psoa.ruleml.org/usecases/OfficeProspector/suite2>
```

```
?b=<http://psoa.ruleml.org/usecases/OfficeProspector/suite3>
```

The queries are similar to OCL constraints<sup>76</sup>, except that the constraint values can also be constructor function applications such as `:measure(1500 :cad)`.

In contrast to the above rules, which are not specific to any particular domain, next we will explain some slot-constraining rules specific to this use case. The following rules expand `:publicTransAccessDistance`-constrained queries of a building `?b` using the auxiliary predicate `:withinDistance`, which retrieves the global coordinates of geospatial entities `?o1` and retrieves any nearby entity `?o2` that is within the distance of `?dist` meters from `?o1`. Two rules similar to the `:publicTransAccessDistance` rule are defined for slots `:restaurantDistance` and `:hotelDistance`.

```
forall ?b ?s ?dist
```

```
(
```

---

<sup>76</sup><http://www.omg.org/spec/OCL/>

```

?b#Top(:constrain(:publicTransAccessDistance)->:atmost(?dist)) :-
  And(
    ?b#:Building
    ?s#:PublicTransAccess
    :withinDistance(?b ?s ?dist)
  )
)

Forall ?o1 ?o2 ?lat1 ?long1 ?lat2 ?long2 ?dist
(
  :withinDistance(?o1 ?o2 :measure(?dist :m)) :-
    And(
      ?o1#:GeoEntity(:coord->:Point(?lat1 ?long1))
      ?o2#:GeoEntity(:coord->:Point(?lat2 ?long2))
      :distanceLessEqual(?lat1 ?long1 ?lat2 ?long2 ?dist)
    )
)
)

```

#### 10.2.3.4 Miscellaneous Rules

In this sub-subsection we explain a few other rules employed by the use case.

The following rule is used to convert measures, e.g. from meters to feet.

```

Forall ?o ?p ?v1 ?v2
(
  ?o#Top(?p->:measure(?v1 :ft)) :-
    And(
      ?o#Top(?p->:measure(?v2 :m))
      ?v1 = External(func:numeric-multiply(?v2 0.3048))
    )
)
)

```

The following rules provide a recursive definition of slot `:hasPartTrans`, which is the transitive closure of `:hasPart`.

```

Forall ?X ?Y
(
  ?X#Top(:hasPartTrans->?Y) :-
  ?X#Top(:hasPart->?Y)
)

Forall ?X ?Y ?Z
(
  ?X#Top(:hasPartTrans->?Z) :-
  And(?X#Top(:hasPart->?Y)
    ?Y#Top(:hasPartTrans->?Z))
)

```

## 10.2.4 Use Scenarios

In this subsection we give examples of user queries at different levels of the partonomy – neighborhoods, buildings, suites, as well as spaces inside a suite. All of these queries can be successfully executed in PSOATransRun 1.3.1.

The following is an example query for finding any suite `?s` that satisfies: 1) the monthly rent of `?s` is at most CAD 1200; 2) the HVAC system of `?s` has a rating of at least basic; 3) `?s` has Internet.

```

?s#:Suite(
  :constrain(:monthlyRent)->:atmost(:measure(1200 :cad))
  :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)
  :utility->:internet
)

```

The answers to the query list OIDs of suites that fulfill the constraints. If the internal dataset in OfficeProspector is replaced with facts in Figures 10.4 and 10.5, the answer is the following:

```
?s=<http://psoa.ruleml.org/usecases/OfficeProspector/suite3>
```

If the constraint value for `:monthlyRent` is changed to `:measure(999 :cad)`, no suites can satisfy the modified query

```
?s#:Suite(  
  :constrain(:monthlyRent)->:atmost(:measure(999 :cad))  
  :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)  
  :utility->:internet  
)
```

PSOATransRun will print `No` in this case.

Built on top of the first query, the next query further constrains the building `?b` of the suite `?s` to being built after 1985 and within 1000 meters of a public transport access point. This query needs to be answered using facts from both the internal data set and the external Geocode and LinkedGeoData data sets.

```
And(  
  ?s#:Suite(  
    :constrain(:monthlyRent)->:atmost(:measure(1200 :cad))  
    :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)  
    :utility->:internet  
  )  
  ?b#:Building(  
    :hasPart->?s  
    :constrain(:publicTransAccessDistance)->:atmost(:measure(1000 :m))  
    :constrain(:yearBuilt)->:atleast(1985)
```

```
)  
)
```

Further increasing the geospatial context, more constraints can be added to the neighborhood of the suite ?s, e.g. to require the neighborhood of the suite to have a crime rate lower than 3 percent.

```
And(  
  ?s#:Suite(  
    :constrain(:monthlyRent)->:atmost(:measure(1200 :cad))  
    :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)  
    :utility->:internet  
  )  
  ?b#:Building(  
    :hasPart->?s  
    :constrain(:publicTransAccessDistance)->:atmost(:measure(1000 :m))  
    :constrain(:yearBuilt)->:atleast(1985)  
  )  
  ?n#:Neighborhood(  
    :hasPart->?b  
    :constrain(:crimeRate)->:atmost(3)  
  )  
)
```

Constraints can also be posed on neighborhoods without referring to the intermediate :Building level, using the recursive :hasPartTrans slot.

```
And(  
  ?s#:Suite(  
    :constrain(:monthlyRent)->:atmost(:measure(1200 :cad))  
    :constrain(:hvac)->op-rtg:atleast(op-rtg:basic)
```

```

    :utility->:internet
  )
  ?n#:Neighborhood(
    :hasPartTrans->?s
    :constrain(:crimeRate)->:atmost(3)
  )
)

```

After a suite OID is returned, users can pose queries to explore its interior structure. The following query finds all pairs of adjacent spaces ?o1, ?o2 inside suite :suite2.

```

And(
  :suite2#:Suite(:hasPart->?o1 :hasPart->?o2)
  ?o1#:Space(:adjacent->?o2#:Space)
)

```

The partonomy-taxonomy root class of ?o1 and ?o2 in this query can be refined to specific subclasses in the adjacency structure. For example, the following query finds any kitchen ?o1 and open office ?o2 of suite :suite2 that are adjacent.

```

And(
  :suite2#:Suite(:hasPart->?o1 :hasPart->?o2)
  ?o1#:Kitchen(:adjacent->?o2#:OpenOffice)
)

```

Queries can also be posed to find amenities and their information nearby the building of a suite. The following query finds the name and cuisine of any restaurant ?r within 1000 meters distance of the suite :suite2.

```
And(  
  ?b#Building(:hasPart->:suite2)  
  :withinDistance(?b ?r :measure(1000 :m))  
  ?r#Restaurant(:name->?name :cuisine->?cus)  
)
```

For OfficeProspector, the generated office data consists of 50 buildings, 142 (available) suites, and 2878 spaces inside suites. Using the generated data, PSOATransRun can execute queries that are almost the same as the above ones except that their OID constants are replaced by OIDs to be found in the generated data, e.g. `:suite2` is replaced by `:st2`. Despite some queries returning more than 20 answers, for every query of this subsection, all answers are obtained immediately (with the all-answer setting) in interactive PSOATransRun sessions. Quantitative evaluations of PSOATransRun using test cases can be found in Chapter 11.

# Chapter 11

## Evaluation

In this chapter we evaluate Version 1.0 of the PSOA RuleML language as well as the PSOATransRun instantiations. PSOA RuleML 1.0 is compared to Flora-2/F-logic and RIF-BLD in Section 11.1. Evaluations of PSOATransRun instantiations will be conducted in Section 11.2.

### 11.1 Evaluation of PSOA RuleML 1.0

In this section we compare language features of PSOA RuleML 1.0 and two other object-relational rule languages: Flora-2/F-logic and RIF-BLD.

On the level of atoms, PSOA RuleML adopts a homogeneous approach to combine relationships and frames. This allows more kinds of atoms and more flexibility for fine-grained knowledge representation, as shown in Figure 3.1, to be used besides relationships and frames in Flora-2/F-logic and RIF-BLD. For example, relationships can be used when one wants to extend a relationship tuple with slots for optional arguments and/or multi-valued arguments. Conversely, shelframes can be used when one wants a subset of slotted arguments

of a frame to co-occur, ‘locked together’, in a tuple (cf. the CEO-CTO tuple in Example 5.3). Complementing the typical dependent tuples and independent slots in, respectively, relationships and frames, independent tuples and dependent slots can also be used for atoms (cf. Examples 3.1 – 3.4).

PSOA RuleML supports objectification, which allows oidless atoms (e.g., pairships) to be given a system-generated OID and to be described. This feature is not supported by Flora-2/F-logic and RIF-BLD. RIF-BLD’s named-argument terms are syntactically similar to pairships, yet their semantics disallows objectification and description.

Besides atoms, all three languages include subclass and equality formulas. Flora-2/F-logic supports also schema-level formulas, e.g. for signature declarations. Although these formulas do not have direct counterparts in PSOA and RIF-BLD, their usage as top-level KB formulas can be expressed as rules.

Regarding language expressivity, PSOA RuleML 1.0 corresponds to Hornlog with head existentials, equality, and import statements. Similarly, RIF-BLD corresponds to Hornlog with equality and import directives. Flora-2/F-logic corresponds to Hornlog with equality extended by various kinds of negations and, in Flora-2, many other features (some extra-logical).

The semantics of PSOA, RIF-BLD, and the F-logic kernel of Flora-2 are defined via different varieties of model theories.

## 11.2 Evaluation of PSOATransRun Instantiations

We evaluated the PSOATransRun instantiations in Version 1.3.1 by conducting experiments 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.

The first experiment was performed on a test suite, named UnitTest-PSOA<sup>77</sup>, of 54 test cases and 302 queries, which covers a wide spectrum of PSOA constructs that we have implemented. Each test case consists of one KB, multiple queries and user-provided answers to each query. Answers to each query are obtained from PSOATransRun instantiations and compared to the expected answers automatically. The Prolog instantiation passed all test cases, while the TPTP instantiation failed on 11 test cases that contains external built-ins, which it does not support. For the test cases on which both instantiations succeeded, the Prolog instantiation takes 75ms on average for each query while the TPTP instantiation takes 5.3ms. Here, the Prolog instantiation is slower largely due to the communication overhead between the InterProlog API and the XSB engine.

The second experiment, employing the Chain test cases, explores performance differences between differently modeled KBs. In a series of test cases, we measured the runtime of tupled vs. slotted and perspectival (dependent) vs. perspeneutral (independent) variations of PSOA KBs in both PSOATransRun instantiations. Since PSOA RuleML's main area of differentiation is in offering alternative kinds of atoms, as systematized in the metamodel explained in Section 3.2.1, we focus this experiment on querying single (rather than conjunctions/joins of) atoms through rule chains of increasing lengths.

We used Python-based generators to create four groups of test cases,<sup>78</sup> each using one of the four major kinds of atoms: dependent-tuple, independent-tuple, dependent-slot, and independent-slot. Each group has test cases dis-

---

<sup>77</sup><https://github.com/RuleML/PSOATransRunComponents/tree/master/PSOATransRun/test/>

<sup>78</sup><http://psoa.ruleml.org/testcases/>

tinguished by the number  $k$  of KB rules, which is a parameter of the group's 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 ( $i = 1, \dots, k, i' = i - 1$ ):

```
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 ( $i = 1, \dots, k, i' = i - 1$ ):

```
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 independent-tuple and independent-slot groups are constructed by reversing the '+' signs of the two dependent groups.

Starting with  $k = 0$  rules and incrementing in steps of 50 rules until reaching  $k = 500$  rules, we generated eleven test cases for each group and measured the average query execution time in 10 runs of each test case except for the ones in the dependent-tuple group under the static/dynamic objectification

setup, which will be explained next. For the dependent-tuple group, we also compared the query execution time between static vs. static/dynamic objectification setups as explained in Section 5.3. For the other three groups, where none of the predicates can be relational, static/dynamic objectification degenerates to static objectification, hence we do not compare the two setups. Since the query execution time is relatively short for the dependent-tuple group under the static/dynamic setup, each test case is executed 100 times to obtain the average. For this group, we also estimated the extra time besides reasoning in both instantiations by measuring the average query execution time for the query `And(?X1=_a1 ?X2=_a2 ?X3=_a3)` as baselines, which returns the same answer as the test queries without the need for any inference. Such an *identity query* consists of a conjunction of (equational) answer bindings, whose processing time before (unchanged) printing is negligible. For the Prolog instantiation, the extra time comes from the communication overhead between the XSB engine and the InterProlog API. For the TPTP instantiation, the extra time is the KB loading time in VampirePrime.<sup>79</sup>

For the Prolog instantiation, the results of the tupled groups are shown in Table 11.1 and Figure 11.1 while the results of the slotted groups are shown in Table 11.2 and Figure 11.2. For the TPTP instantiation, the results are shown in Table 11.3 and Table 11.4.<sup>80</sup> Tables 11.1 and 11.3 also includes the results of identity queries. In the tables, the shortcut “*query-err*” means the query yields an execution error (e.g., because of running out of memory).

---

<sup>79</sup>VampirePrime does not separate KB loading and query execution phases like XSB. Hence the measured time includes the KB loading time.

<sup>80</sup>The TPTP instantiation is executed under the undifferentiated objectification setup rather than the differentiated setup (the default), in which the VampirePrime engine fails to get any answer for  $k \geq 50$ .

Table 11.1: Execution time of eleven Tupled Chain test cases for Prolog instantiation.

		Dependency & Objectification Choices			
		Indep	Dep		
			Stat	Stat/Dyn	IdentQry
Number of Rules	0	47	49	49	46
	50	72	47	47	49
	100	161	52	46	43
	150	403	59	48	50
	200	858	71	44	46
	250	1636	81	44	42
	300	2834	82	45	46
	350	<i>query-err</i>	95	46	47
	400		106	44	43
	450		131	47	44
	500		143	44	46

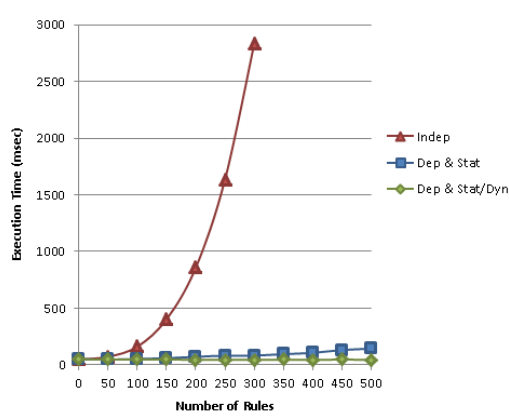


Figure 11.1: Execution time of eleven Tupled Chain test cases for Prolog instantiation.

Table 11.2: Execution time of eleven Slotted Chain test cases for Prolog instantiation.

		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	<i>query-err</i>	202
	400		239
	450		289
	500		352

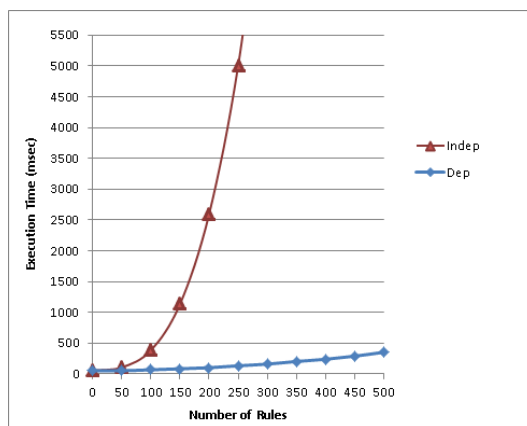


Figure 11.2: Execution time of eleven Slotted Chain test cases for Prolog instantiation.

Table 11.3: Execution time of eleven Tupled Chain test cases for TPTP instantiation.

		Dependency & Objectification Choices			
		Indep	Dep		
			Stat	Stat/Dyn	IdentQry
Number of Rules	0	8	8	8	8
	50	60	24	11	10
	100	244	41	19	16
	150	707	57	22	18
	200	1574	84	32	25
	250	3038	110	41	31
	300	5597	140	54	41
	350	9185	179	65	53
	400	14173	216	78	64
	450	20921	261	91	74
	500	30626	311	107	88

Table 11.4: Execution time of eleven Slotted Chain test cases for TPTP instantiation.

		Dependency Choices	
		Indep	Dep
Number of Rules	0	8	9
	50	1655	80
	100	10203	182
	150	27902	345
	200	<i>query-err</i>	561
	250		848
	300		1206
	350		1655
	400		2166
	450		2810
	500		3487

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 description, 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 considerably 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, description 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.

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

Comparing the last two columns in Table 11.1, we can see that in the dependent-tuple group, under static/dynamic objectification, the query execution time of the Prolog instantiation is entirely dominated by the communication overhead between the InterProlog API and XSB – as shown in the last (baseline) column – while the actual reasoning time is almost negligible. In this group of tests, we also tried to increase  $k$  up to 100000. The results show

that the average actual reasoning time grows very slowly, remains under 7ms, and constitutes only a very small portion of the query execution time. From the last two columns in Table 11.3, we can see that for the TPTP instantiation, despite the KB loading time constituting a major part of the query execution time in the group of tests, the actual reasoning time still slowly grows from  $k = 0$  to  $k = 500$ . This shows that, with static/dynamic objectification, the translated Chain KBs can be efficiently queried in both XSB and VampirePrime. This provides further evidence (cf. Section 5.3) that the efficiency of relational rules can be fully retained by static/dynamic objectification.

Comparing the performance of Prolog and TPTP instantiations from the tables, we can see that: (1) In simple test cases, which take less than 50ms, the Prolog instantiation is slower because the communication overhead dominates. In other test cases, the Prolog instantiation is faster. (2) The TPTP instantiation is able to get answers on independent-tuple test cases when  $k \geq 350$ , while the Prolog instantiation fails to get any answer for those cases.

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.

The third experiment, employing NDChain test cases, extends each tupled test case in the second experiment with one fact and  $k$  non-deterministic rules. For the dependent group, the added fact is `_s0(_a1 _a2 _a3)` and each added rule has the following form ( $i = 1, \dots, k, i' = i - 1$ ):

```

Forall ?X1 ?X2 ?X3 (
  _si(?X1 ?X2 ?X3) :- Or(_ri(?X1 ?X2 ?X3) _si'(?X1 ?X2 ?X3))
)

```

Here, each `_si` is derived from either `_ri` or `_si'`. The query is `_sk(?X1 ?X2 ?X3)`, which has one binding `?X1=_a1, ?X2=_a2, ?X3=_a3`.<sup>81</sup> Since every `_si`-call leads to an `_ri` call, the reasoning time increases mostly because of the possible repeated `_ri`-calls during the search of answers.

Starting with  $k = 0$ , we first increase in steps of one until reaching  $k = 10$ , then increase in steps of 10 until reaching  $k = 50$ , and finally increase in steps of 50 until reaching  $k = 350$ . The results for the Prolog and TPTP instantiations are shown in, respectively, Tables 11.5 and 11.6.

---

<sup>81</sup> In this test case, `_si(_a1 _a2 _a3)` can be proved by different `Or`-branches, hence can lead to redundant `?X1=_a1, ?X2=_a2, ?X3=_a3` answers. In our testing, all answers are collected at once and redundant answers are eliminated.

Table 11.5: Execution time of Tupled NDChain test cases for Prolog instantiation.

		Dependency & Objectification Choices		
		Indep	Dep	
			Stat	Stat/Dyn
$k$	0	50	50	46
	1	54	52	47
	2	60	60	48
	3	77	74	50
	4	237	231	48
	5	1861	1759	48
	6	<i>query-err</i>	<i>query-err</i>	50
	7			50
	8			47
	9			58
	10			56
	20			60
	30			72
	40			94
	50			117
	100			270
	150			540
	200			947
	250			1458
	300			2165
350	<i>query-err</i>			

Table 11.6: Execution time of Tupled NDChain test cases for TPTP instantiation.

		Dependency & Objectification Choices		
		Indep	Dep	
			Stat	Stat/Dyn
$k$	0	7	6	5
	1	5	4	4
	2	5	5	4
	3	7	6	5
	4	9	11	6
	5	9	8	4
	6	10	8	5
	7	11	9	6
	8	12	9	4
	9	14	10	5
	10	16	10	5
	20	49	16	9
	30	110	22	10
	40	203	29	13
	50	358	38	16
	100	2423	86	31
	150	8627	150	56
	200	22194	241	81
	250	45263	352	117
	300	82799	510	160
350	<i>query-err</i>	678	206	

The results again show that static/dynamic objectification is faster than static objectification, sometimes by an order of magnitude as  $k$  increases. Moreover, even under static objectification, dependent test cases are faster than their independent counterparts in both the Prolog and TPTP instantiations when  $k \geq 3$ , sometimes by one or two orders of magnitude as  $k$  increases.

The results also show that the TPTP instantiation is 5 to 12 times faster than the Prolog instantiation on NDChain tests under static/dynamic objectification. When  $k = 350$ , the Prolog instantiation cannot get answers, indicated by an execution error, *query-err*, in the last row of Table 11.5, while the TPTP instantiation can. Besides the communication overhead between InterProlog and XSB, the main cause of the performance difference is that VampirePrime’s bottom-up computation avoids repeated  $r_i$ -calls of a top-down reasoner. In XSB Prolog, tabling can also avoid repeated  $r_i$ -calls. However, PSOATransRun turns on tabling only for the reserved predicates, e.g. `memterm`, introduced by our translation. Under static/dynamic objectification, after translation, user predicates in PSOA KBs are handed through to the XSB engine almost unchanged (except for an additional pair of single quotes). Hence, e.g. for safety reasons, PSOATransRun does not enable tabling for these translated user predicates.

# Chapter 12

## Conclusions and Future Work

In this chapter, we first summarize the dissertation in Section 12.1. Next, Section 12.2 lists the major contributions of the dissertation. Finally, Section 12.3 discusses future work.

### 12.1 Summary

In this section we will revisit the objectives we set up in Section 1.3 and consider their fulfillment.

The first objective is to study the interoperation from PSOA to TPTP and to Prolog, as well as the interoperation from N3 to PSOA. The objective is fulfilled in Chapters 6, 7, and 8: Chapter 6 discusses the interoperation from N3 to PSOA. Chapter 7 explains the interoperation from PSOA to TPTP. Chapter 8 discusses the interoperation from PSOA to Prolog.

The second objective is to design an architecture for interoperating and porting object-relational knowledge. The objective is fulfilled in Chapter 4. An interoperation and portation architecture is introduced. Both the interop-

eration and portation parts specialize a metaframework to a framework.

The third objective is to characterize sublanguages of PSOA within which the translations are semantics-preserving. This is achieved – with sublanguages moderately restricting the use of constructs in KBs/queries – for the PSOA transformations in Chapter 5 as well as the translations for PSOA2TPTP and PSOA2Prolog in Chapters 7 and 8. Chapter 5 defines the PSOA transformation steps. For each step, the PSOA sublanguage within which the step is semantics-preserving is explained. Built on top of the transformation steps, translations from PSOA to TPTP and to Prolog are explained and the corresponding PSOA sublanguages for semantics preservation are characterized.

The fourth objective, to realize translators, is fulfilled by the N3-to-PSOA, PSOA2TPTP, and PSOA2Prolog translator implementations. Built on top of the last two translators, the fifth objective, of realizing reasoning systems for query answering in PSOA RuleML, is fulfilled by two instantiations of the PSOATransRun framework: `PSOATransRun[PSOA2TPTP,VampirePrime]` and `PSOATransRun[PSOA2Prolog,XSBProlog]`. Chapter 9 explains the implementation.

The sixth objective is to apply the PSOA RuleML language as well as its implementation to use cases. This is fulfilled in Chapter 10 via two use cases, Port Clearance Rules and OfficeProspector.

The seventh objective, of developing a test suite and evaluating PSOA as well as the realized reasoning systems, is fulfilled in Chapter 11. The PSOA language is compared to Flora-2/F-logic and RIF-BLD. A test suite, `UnitTestPSOA`, is developed to validate – for KBs with a wide spectrum of PSOA constructs – current and future reasoning systems. Moreover, we eval-

uate the performance of PSOATransRun on two other test cases exemplifying rules over the basic kinds of atoms of the PSOA RuleML 1.0 metamodel: Chain and NDChain.

The eighth objective is to revise the syntax and semantics of PSOA based on the findings in translator development and evaluation of translator-based reasoning systems. This is achieved in Chapter 3 by advancing the original version defined in [15] to Version 1.0, which constitutes a succinct yet expressive language mostly due to its orthogonal overall design according to the three-dimensional metamodel for atoms in Figure 3.1.

## 12.2 Dissertation Contributions

The major contributions of this dissertation are captured in the following hierarchical structure:

1. Revise the PSOA RuleML language to achieve Version 1.0.

In PSOA RuleML 1.0, a new (predicate-)independent vs. (predicate-)dependent distinction for descriptors is introduced and a novel perspectivity dimension of atoms is defined on top of it. The new dimension is illustrated with a Rich TA example systematically describing the same individual differently under different perspectives using atoms having different predicates. Moreover, the interpretation of embedded psoa terms as oidless expressions or oidful atoms is clarified in the revision. The EBNF-defined syntax and model-theoretic semantics are revised to incorporate the changes.

2. Create an architecture for interoperating and porting object-relational knowledge.

The architecture has two parts, each consisting of a framework that specializes an existing metaframework. In the architecture, each translation from PSOA RuleML is a composition of a normalization within PSOA and a conversion from normalized PSOA to the target language. The normalization for each target language is itself composed of multiple PSOA transformation steps, several of which are shared between different normalizations.

3. Formalize and realize translations, as well as prove their semantics preservation.

(a) Formalize nine PSOA transformation steps, which can be reused by other PSOA-sourced translations, and realized corresponding modules.

The transformation steps are unnesting, objectification, Skolemization, subclass transformation, description, flattening external expressions, and splitting conjunctive-conclusion rules. For objectification, static/dynamic objectification is invented to keep unchanged as many relationships as possible, instead constructing virtual OIDs at query time if and when bindings for their OID variables are being queried. The principle for composing the transformation steps is discussed.

(b) Formalize and realize the translations from **External**-free PSOA Kernel to TPTP and from PSOA Kernel to Prolog, using the PSOA transformation steps.

The transformation steps are sequentially composed into FOL- and LP-targeting normalizations for the TPTP and Prolog translations, respectively. The LP-targeting normalization is an extension of the FOL-targeting normalization with additional transformation steps.

(c) Formalize the translation from an N3 sublanguage, N3Basic, to PSOA and realized the translation for N3 facts.

The N3-to-PSOA translator can be composed with PSOA2TPTP and PSOA2Prolog to form N3-to-TPTP and N3-to-Prolog translators, respectively.

(d) Study the semantics preservation of each PSOA transformation step as well as of the TPTP and Prolog translations and identified the sublanguages for semantics preservation.

A sufficient condition for proving semantics preservation of PSOA transformations is given in Section 5.1. For each transformation step, the appropriate PSOA sublanguage is defined and semantics-preservation theorems are proved. Based on that, semantics preservation of their compositions is proved for FOL- and LP-targeting normalizations with respect to appropriate sublanguages. Finally, semantics preservation is proved for the PSOA2TPTP translation with respect to the FOL semantics and for the PSOA2Prolog translation with respect to the declarative semantics of logic programs.

4. Combine the PSOA2TPTP and PSOA2Prolog translators with, respectively, the runtime engines VampirePrime and XSB Prolog into two PSOATransRun instantiations to provide query answering for PSOA.

5. Create use cases for real-world problems and performed evaluations on test cases.

(a) Apply PSOA and PSOATransRun to the two use cases Port Clearance Rules and OfficeProspector.

In Port Clearance Rules, we formalized existing moderately controlled

English rules in PSOA RuleML, added facts directly in PSOA, and queried results in PSOATransRun. In OfficeProspector, generated office data are enriched with public data sets in different modeling paradigms in order to enable advanced user queries. The use cases show that PSOA RuleML is well-suited to capture real-world problems and PSOATransRun is well-suited for KB development.

- (b) Evaluate the PSOA language as well as the PSOATransRun instantiations through experiments on test cases.

The evaluation show the following (categories are given before explanations):

- (1) Modeling: For descriptors that need to be defined across different predicates via different rules, dependent modeling is more efficient than independent modeling.
- (2) Modeling: For arguments that are often used together in a KB and in queries, tupled modeling is more efficient than slotted modeling.
- (3) Translation: Static/dynamic objectification is always more efficient than static objectification in KBs that contain relational predicates. It can fully retain the efficiency of relational rules.
- (4) Instantiation: For simple test cases, the TPTP instantiation is always faster because of the communication overhead in the Prolog instantiation. On the other hand, for non-simple test cases, the Prolog instantiation is sometimes faster.

## 12.3 Future Work

On the language design level, the PSOA RuleML language can be further extended to include interesting features from other rule languages. Staying in the realm of Hornlog expressivity, schema-level declarations from RDF Schema using `rdfs:subPropertyOf`, `rdfs:domain`, and `rdfs:range` could be allowed. Possible extensions beyond Hornlog expressivity include classical negation and negation-as-failure. Moreover, the current model-theoretic semantics of PSOA could be complemented with a proof-theoretic semantics and proof procedure.

For rule interoperation, the dashed lines in Figure 4.3 can be fully implemented. Other PSOA-related translations can be studied, e.g. between PSOA and subsets of DL such as OWL 2 RL [61]. The transformation steps in Chapter 5 may be further optimized. For example, the ‘relationalness test’ for dynamic objectification could be restricted to only conclusion atoms since they are similar to query atoms. In this way, condition atoms can also benefit from virtual OIDs. Also, to improve translation efficiency, some (directly composable) transformations, e.g. unnesting and flattening, could be merged into one single transformation.

On the implementation level, the current PSOA/PS-based translators and reasoning systems can be extended for the XML serialization of PSOA RuleML. The current translator-based PSOATransRun instantiations could be complemented with a reasoning system that directly implements the above-mentioned proof procedure for PSOA.

# Bibliography

- [1] *Semantics of business vocabulary and business rules v1.0*, January 2008, OMG Specification, <http://www.omg.org/spec/SBVR/1.0/>.
- [2] Elie Abi-Lahoud, Tom Butler, Donald Chapin, and John Hall, *Interpreting regulations with SBVR*, Proceedings of Rule Challenge, Human Language Technology and Doctoral Consortium, at the 7th International Symposium on Rules (Paul Fodor, Dumitru Roman, Darko Anicic, Adam Wyner, Monica Palmirani, Davide Sottara, and François Lévy, eds.), CEUR-1004, 2013.
- [3] Atul Adya, José A. Blakeley, Sergey Melnik, and S. Muralidhar, *Anatomy of the ADO.NET entity framework*, Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007 (Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, eds.), ACM, 2007, pp. 877–888.
- [4] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman (eds.), *Compilers: Principles, techniques, and tools*, second ed., Pearson/Addison Wesley, Boston, MA, USA, 2007.

- [5] Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, and Juan Sequeda, *A Direct Mapping of Relational Data to RDF*, W3C Candidate Recommendation. <http://www.w3.org/TR/2012/CR-rdb-direct-mapping-20120223/>, February 2012.
- [6] Dörthe Arndt, Ruben Verborgh, Jos De Roo, Hong Sun, Erik Mannens, and Rik Van de Walle, *Semantics of notation3 logic: A solution for implicit quantification*, Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings (Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke, and Dumitru Roman, eds.), Lecture Notes in Computer Science, vol. 9202, Springer, 2015, pp. 127–143.
- [7] Tara Athan, Harold Boley, Guido Governatori, Monica Palmirani, Adrian Paschke, and Adam Wyner, *LegalRuleML: From metamodel to use cases*, Theory, Practice, and Applications of Rules on the Web, Proc. 7th International Symposium, RuleML 2013 (Leora Morgenstern, Petros Stefanias, François Lévy, Adam Wyner, and Adrian Paschke, eds.), Lecture Notes in Computer Science, vol. 8035, Springer, 2013, pp. 13–18.
- [8] Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer, *Implementing RuleML Using Schemas, Translators, and Bidirectional Interpreters*, W3C Workshop on Rule Languages for Interoperability, <http://www.w3.org/2004/12/rules-ws/paper/49/>, April 2005.
- [9] Chitta Baral and Shanshan Liang, *From knowledge represented in frame-based languages to declarative representation and reasoning via ASP*, Principles of Knowledge Representation and Reasoning: Proceedings of the

- Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012 (Gerhard Brewka, Thomas Eiter, and Sheila A. McIlraith, eds.), AAAI Press, 2012.
- [10] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers, *RDF 1.1 Turtle – Terse RDF Triple Language*, February 2014, W3C Recommendation. <https://www.w3.org/TR/turtle/>.
- [11] Tim Berners-Lee and Dan Connolly, *Notation3 (N3): A readable RDF syntax*, March 2011, W3C Team Submission. <http://www.w3.org/TeamSubmission/n3/>.
- [12] Tim Berners-Lee, Dan Connolly, and Sandro Hawke, *Semantic web tutorial using N3*, 2003.
- [13] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler, *N3Logic: A logical framework for the world wide web*, Theory and Practice of Logic Programming (TPLP) **8** (2008), no. 3.
- [14] Harold Boley, *RIF RuleML Rosetta Ring: Round-Tripping the Dlex Subset of Datalog RuleML and RIF-Core*, Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings (Guido Governatori, John Hall, and Adrian Paschke, eds.), Lecture Notes in Computer Science, vol. 5858, Springer, 2009, pp. 29–42.
- [15] Harold Boley, *A RIF-Style Semantics for RuleML-Integrated Positional-Slotted, Object-Applicative Rules*, Rule-Based Reasoning, Programming, and Applications - 5th International Symposium, RuleML 2011 - Europe,

- Barcelona, Spain, July 19-21, 2011. Proceedings (Nick Bassiliades, Guido Governatori, and Adrian Paschke, eds.), LNCS, vol. 6826, Springer, 2011, pp. 194–211.
- [16] Harold Boley, *PSOA RuleML: Linked Objects and Rules*, Semantic Days 2011, Oslo, Norway, June 2011.
- [17] Harold Boley, *PSOA RuleML: Integrated Object-Relational Data and Rules*, Reasoning Web. Web Logic Rules (RuleML 2015) - 11th Int’l Summer School 2015, Berlin, Germany, July 31- August 4, 2015, Tutorial Lectures (Wolfgang Faber and Adrian Paschke, eds.), LNCS, vol. 9203, Springer, 2015.
- [18] Harold Boley, *The RuleML Knowledge-Interoperation Hub*, Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6–9, 2016. Proceedings (José Júlio Alferes, Leopoldo E. Bertossi, Guido Governatori, Paul Fodor, and Dumitru Roman, eds.), Lecture Notes in Computer Science, vol. 9718, Springer, 2016, pp. 19–33.
- [19] Harold Boley, Rolf Grütter, Gen Zou, Tara Athan, and Sophia Etzold, *A Datalog<sup>+</sup> RuleML 1.01 architecture for rule-based data access in ecosystem research*, Rules on the Web. From Theory to Applications (Antonis Bikakis, Paul Fodor, and Dumitru Roman, eds.), Lecture Notes in Computer Science, vol. 8620, Springer International Publishing, 2014, pp. 112–126.

- [20] Harold Boley and Michael Kifer, *RIF Basic Logic Dialect (Working Draft)*, October 2007, W3C Working Draft, <https://www.w3.org/TR/2007/WD-rif-bld-20071030/>.
- [21] ———, *A guide to the basic logic dialect for rule interchange on the web*, IEEE Transactions on Knowledge and Data Engineering **22** (2010), no. 11, 1593–1608.
- [22] ———, *RIF Basic Logic Dialect*, W3C Recommendation. <http://www.w3.org/TR/2013/REC-rif-bld-20130205/>, February 2013.
- [23] Harold Boley, Jing Mei, Michael Sintek, and Gerd Wagner, *RDF/RuleML Interoperability*, W3C Workshop on Rule Languages for Interoperability, <http://www.w3.org/2004/12/rules-ws/paper/93/>, W3C, April 2005.
- [24] Harold Boley, Taylor Michael Osmun, and Benjamin Larry Craig, *WellnessRules: A Web 3.0 Case Study in RuleML-Based Prolog-N3 Profile Interoperation*, Rule Interchange and Applications, International Symposium, RuleML 2009, Las Vegas, Nevada, USA, November 5-7, 2009. Proceedings (Guido Governatori, John Hall, and Adrian Paschke, eds.), Lecture Notes in Computer Science, vol. 5858, Springer, 2009, pp. 43–52.
- [25] Harold Boley and Adrian Paschke, *Rule Responder Agents: Framework and Instantiations*, Semantic Agent Systems: Foundations and Applications (Atilla Elci, Mamadou Tadiou Kone, and Mehmet A. Orgun, eds.), Springer Studies in Computational Intelligence, Vol. 344, 2011.
- [26] Harold Boley, Adrian Paschke, and Omair Shafiq, *RuleML 1.0: The Overarching Specification of Web Rules*, Proc. 4th International Web

Rule Symposium: Research Based and Industry Focused (RuleML-2010), Washington, DC, USA, October 2010, Lecture Notes in Computer Science, Springer, 2010.

- [27] Harold Boley and Gen Zou, *Perspectival knowledge in PSOA RuleML: Representation, model theory, and translation*, ArXiv e-prints (2017), <https://arxiv.org/pdf/1712.02869>.
- [28] Piero Bonatti, *Rule languages for security and privacy in cooperative systems*, Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International, vol. 1, 2005, pp. 268–269.
- [29] Pedro Cabalar, *Existential quantifiers in the rule body*, Proceedings of the 23rd Workshop on (Constraint) Logic Programming 2009, Universitätsverlag Potsdam, 2010, p. 59.
- [30] Vinay K. Chaudhri, Stijn Heymans, Michael Wessel, and Son Cao Tran, *Object-oriented knowledge bases in logic programming*, Technical Communication of International Conference in Logic Programming, 2013.
- [31] C.L. Chang and R.C.T. Lee, *Symbolic logic and mechanical theorem proving*, Academic Press, 1973.
- [32] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks, *Computing Datalog rewritings beyond Horn ontologies*, Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence (IJCAI 2013), 2013.
- [33] Richard Cyganiak and David Wood, *RDF 1.1 Concepts and Abstract Syntax*, W3C Working Draft. <http://www.w3.org/TR/2013/WD-rdf11-concepts-20130723/>, July 2013.

- [34] Souripriya Das, Seema Sundara, and Richard Cyganiak, *R2RML: RDB to RDF Mapping Language*, W3C Candidate Recommendation. <http://www.w3.org/TR/2012/CR-r2rml-20120223/>, February 2012.
- [35] Pierre Deransart, AbdelAli Ed-Dbali, and Laurent Cervoni, *Prolog: The standard (reference manual)*, Springer, 1996.
- [36] H.B. Enderton, *A mathematical introduction to logic*, Academic Press, 2001.
- [37] M. Geneserith and R. Fikes, *Knowledge interchange format, version 3.0. Reference manual*, Computer Science Department, Technical Report Logic 92-1, Stanford University, 1992.
- [38] Joseph A. Goguen and Rod M. Burstall, *Institutions: abstract model theory for specification and programming*, Journal of ACM **39** (1992), no. 1, 95–146.
- [39] Yiwei Gong and Marijn Janssen, *Adaptive and compliant policy implementation: Creating administrative processes using semantic web services and business rules*, Collaborative, Trusted and Privacy-Aware e/m-Services (Christos Douligeris, Nineta Polemi, Athanasios Karantjias, and Winfried Lamersdorf, eds.), IFIP Advances in Information and Communication Technology, vol. 399, Springer Berlin Heidelberg, 2013, pp. 298–310.
- [40] Georg Gottlob, Giorgio Orsi, Andreas Pieris, and Mantas Šimkus, *Datalog and its extensions for semantic web databases*, Reasoning Web. Semantic Technologies for Advanced Query Answering, Springer, 2012, pp. 54–77.

- [41] Stephen Greene, *Object-oriented to positional RuleML translators*, <http://ruleml.org/ooruleml-xslt/oo2prml.html>.
- [42] Harold Boley, *Integrating Positional and Slotted Knowledge on the Semantic Web*, *Journal of Emerging Technologies in Web Intelligence* **4** (2010), no. 2, 343–353.
- [43] Patrick Hayes and Peter F. Patel-Schneider, *RDF 1.1 Semantics*, July 2013, W3C Working Draft. <http://www.w3.org/TR/rdf11-mt/>.
- [44] ISO/IEC 13211-1, *Prolog – part 1: General core*, 1995.
- [45] Ken Kaneiwa, *Order-sorted logic programming with predicate hierarchy*, *Artificial Intelligence* **158** (2004), no. 2, 155–188.
- [46] Yevgeny Kazakov, *Saturation-based decision procedures for extensions of the guarded fragment*, Ph.D. thesis, 2007.
- [47] C Maria Keet, *Introduction to part-whole relations: mereology, conceptual modelling and mathematical aspects*, KRDB Research Centre Technical Report KRDB06-3 (Tutorial reader), Faculty of Computer Science, Free University of Bozen-Bolzano, Italy **2** (2006).
- [48] Michael Kifer, *Rules and ontologies in F-logic*, Proceedings of the First international conference on Reasoning Web (Berlin, Heidelberg), Springer-Verlag, 2005, pp. 22–34.
- [49] Michael Kifer, Georg Lausen, and James Wu, *Logical foundations of object-oriented and frame-based languages*, *Journal of ACM* **42** (1995), no. 4, 741–843.

- [50] Tae-Won Kim, Joohyung Lee, and Ravi Palla, *Circumscriptive event calculus as answer set programming*, Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 823–829.
- [51] Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler, *ELP: Tractable rules for OWL 2*, The Semantic Web - ISWC 2008 (Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, eds.), Lecture Notes in Computer Science, vol. 5318, Springer Berlin Heidelberg, 2008, pp. 649–664.
- [52] Markus Krötzsch, Sebastian Rudolph, and Peter H. Schmitt, *On the semantic relationship between datalog and description logics*, Web Reasoning and Rule Systems (Pascal Hitzler and Thomas Lukasiewicz, eds.), Lecture Notes in Computer Science, vol. 6333, Springer Berlin Heidelberg, 2010, pp. 88–102.
- [53] Markus Krötzsch, František Simančík, and Ian Horrocks, *A description logic primer*, ArXiv e-prints (2012).
- [54] Oliver Kutz, Till Mossakowski, Christian Galinski, and Christoph Lange, *Towards a standard for heterogeneous ontology integration and interoperability*, International Conference on Terminology, Language and Content Resources (LaRC), 2011.
- [55] Christoph Lange, Oliver Kutz, Till Mossakowski, and Michael Grüninger, *The distributed ontology language (DOL): ontology integration and interoperability applied to mathematical formalization*, Intelligent Computer Mathematics, Springer, 2012, pp. 463–467.

- [56] Joohyung Lee and Ravi Palla, *System F2LP – computing answer sets of first-order formulas*, Logic Programming and Nonmonotonic Reasoning (Esra Erdem, Fangzhen Lin, and Torsten Schaub, eds.), Lecture Notes in Computer Science, vol. 5753, Springer Berlin Heidelberg, 2009, pp. 515–521.
- [57] J.W. Lloyd, *Foundations of logic programming (second edition)*, Springer-Verlag, 1987.
- [58] Leora Morgenstern, Chris Welty, Harold Boley, and Gary Hallmark, *RIF Primer*, W3C Recommendation. <http://www.w3.org/2005/rules/wiki/Primer>, June 2010.
- [59] Till Mossakowski, Joseph Goguen, Răzvan Diaconescu, and Andrzej Tarlecki, *What is a logic?*, Logica universalis, Springer, 2007, pp. 111–133.
- [60] Till Mossakowski and Oliver Kutz, *The onto-logical translation graph*, Modular Ontologies - Proceedings of the Fifth International Workshop (WoMO) 2011 (Oliver Kutz and Thomas Schneider, eds.), vol. 230, IOS Press, 2011, pp. 94–109.
- [61] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz, *OWL 2 web ontology language — profiles (second edition)*, W3C Recommendation. <http://www.w3.org/TR/2012/REC-owl2-profiles-20121211>, December 2012.
- [62] Axel Polleres, Harold Boley, and Michael Kifer, *RIF Datatypes and Built-ins 1.0 (Second Edition)*, February 2013, W3C Recommendation, <http://www.w3.org/TR/2013/REC-rif-dtb-20130205/>.

- [63] David A. Randell, Zhan Cui, and Anthony Cohn, *A spatial logic based on regions and connection*, KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (Cambridge, MA, USA) (Bernhard Nebel, Charles Rich, and William Swartout, eds.), Morgan Kaufmann, 1992, pp. 165–176.
- [64] Alexandre Riazanov, Gregory W. Rose, Artjom Klein, Alan J. Forster, Christopher J. O. Baker, Arash Shaban-Nejad, and David L. Buckeridge, *Towards Clinical Intelligence with SADI Semantic Web Services: a Case Study with Hospital-Acquired Infections Data*, Proceedings of the 4th International Workshop on Semantic Web Applications and Tools for the Life Sciences (New York, NY, USA), SWAT4LS '11, ACM, 2012, pp. 106–113.
- [65] Gert Smolka, *Feature-constraint logics for unification grammars*, The Journal of Logic Programming **12** (1992), no. 1, 51–87.
- [66] Silvie Spreeuwenberg, Charlotte Bouvy, and Martijn Zoet, *Het spel met de regels*, Uitgave van BRPN, ISBN 978-90-815568-2-8, 2013.
- [67] Geoff Sutcliffe, *The tptp problem library and associated infrastructure*, Journal of Automated Reasoning **43** (2009), no. 4, 337–362.
- [68] Geoff Sutcliffe, Mark Stickel, Stephan Schulz, and Josef Urban, *Answer extraction for TPTP*, <http://www.cs.miami.edu/~tptp/TPTP/Proposals/AnswerExtraction.html>.
- [69] *ISO/IEC 24707:2007. the ISO common logic standard*, 2007, <http://common-logic.org/>.

- [70] Stanislav Vojir, Tomáš Kliegr, Andrej Hazucha, Radek Škrabal, and Milan Šimuněk, *Transforming association rules to business rules: Easyminer meets Drools*, Proceedings of Rule Challenge, Human Language Technology and Doctoral Consortium, at the 7th International Symposium on Rules (Paul Fodor, Dumitru Roman, Darko Anicic, Adam Wyner, Monica Palmirani, Davide Sottara, and François Lévy, eds.), CEUR-1004, July 2013.
- [71] Jef Wijsen, *Certain conjunctive query answering in first-order logic*, ACM Trans. Database Syst **37** (2012), no. 2, 9:1–9:35.
- [72] Heng Zhang, Yan Zhang, Mingsheng Ying, and Yi Zhou, *Translating first-order theories into logic programs*, Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two, IJCAI'11, 2011, pp. 1126–1131.
- [73] Gen Zou and Harold Boley, *PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog*, Proc. 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany, Lecture Notes in Computer Science, Springer, August 2015.
- [74] ———, *Minimal objectification and maximal unnesting in PSOA ruleML*, Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings (José Júlio Alferes, Leopoldo E. Bertossi, Guido Governatori, Paul Fodor, and Dumitru Roman, eds.), Lecture Notes in Computer Science, vol. 9718, Springer, 2016, pp. 130–147.

- [75] ———, *Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic*, Proceedings of the RuleML+RR 2017 Challenge, vol. 1875, CEUR, July 2017.
- [76] Gen Zou, Reuben Peter-Paul, Harold Boley, and Alexandre Riazanov, *PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners*, Rules on the Web: Research and Applications, Proc. 6th International Symposium, RuleML 2012, Montpellier, France (Antonis Bikakis and Adrian Giurca, eds.), Lecture Notes in Computer Science, vol. 7438, Springer, August 2012, pp. 264–279.

# Appendix A

## Port Clearance KB in PSOA

### RuleML

```
Document (  
  Prefix(: <http://psoa.ruleml.org/usecases/PortClearance#>)  
  Prefix(math: <http://psoa.ruleml.org/lib/math#>)  
  Prefix(phys: <http://psoa.ruleml.org/lib/phys#>)  
  
  Import(<http://psoa.ruleml.org/lib/math.psoa>)  
  Import(<http://psoa.ruleml.org/lib/phys.psoa>)  
  
Group (  
  
  % Port Clearance Knowledge Base  
  % Version: 2017-06-08  
  
  % Copy&paste-ready KB file in presentation syntax (PSOA RuleML/PS):  
  % http://psoa.ruleml.org/usecases/PortClearance/PortClearance.psoa  
  % To be complemented by KB file in serialization syntax (PSOA RuleML/XML):  
  % http://psoa.ruleml.org/usecases/PortClearance/PortClearance.ruleml  
  
  % PSOA RuleML/PS can be directly used as a PSOATransRun input file:  
  % http://wiki.ruleml.org/index.php/PSOA\_RuleML#PSOATransRun (Current Release)
```

```

% Reordering, subgrouping, and explaining the rules from
% https://dmcommunity.org/challenge/challenge-march-2016/

% Main relational rule invokes inspection rule for certificate And safety

% Rule 2
Forall ?s (
  :MayEnterDutchPortUnloaded(?s) :-
    :CompliesInspectionRequirementsUnloaded(?s)
)

% Rule 3
Forall ?s (
  :CompliesInspectionRequirementsUnloaded(?s) :-
    And(:HasValidCertificate(?s)
      :MeetsSafetyRequirementsUnloaded(?s))
)

% Object-relational certificate rule compares ship's registry
% expiration with current date

% Rule 10
Forall ?s ?d ?e (
  :HasValidCertificate(?s) :-
    And(?s#:Ship(:registryExpirationDate->?e)
      % phys:currentDate(?d) % Uncomment for local date (deployment)
      :currentDate(?d)      % Uncomment for fixed date (reproducibility)
      phys:lessThanDate(?d ?e))
)

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

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

```

```

)

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

% Object-centered (except for math) rules to get qualitative size by
% thresholding length

% Rule 9
Forall ?s ?l (
  ?s#Top(:size->:small) :-
    And(?s#:Ship(:totalLength->?l)
         math:lessThan(?l 80))
)

% Rule 4
Forall ?s ?l (
  ?s#Top(:size->:large) :-
    And(?s#:Ship(:totalLength->?l)
         math:greaterEq(?l 80))
)

% Object-centered (except for math) rule to get qualitative status
% by thresholding residual

% Rule 1&5 (combines Rule 1 and Rule 5)
Forall ?h ?c (
  ?h#Top(:status->:clean) :-
    And(?h#:ShipHold(:residualCargoMeasurement->?c)
         math:lessEq(?c 0.5))
)

```

```

:currentDate(phys:date(2017 5 6)) % Uncomment for fixed date (reproducibility)

% Ship facts (No or Yes refer to answers for queries, as of 2017-05-06,
% with :ship1, :ship2, ... as arguments)

% Facts covering all cases with qualitative slot-filler distinctions
% Explanatory comments for Yes answers focus on the most relevant slots

% Distinction for :registryExpirationDate

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

% Ship 2 - Yes, registry is valid
:ship2#:Ship(:registryExpirationDate->phys:date(2017 10 1)
            :totalLength->20
            :hold->:h2#:ShipHold(:residualCargoMeasurement->0.2
                                :hull->:single))

% Distinction for :residualCargoMeasurement

% Ship 3 - No, hold not clean
:ship3#:Ship(:registryExpirationDate->phys:date(2020 1 1)
            :totalLength->70
            :hold->:h3#:ShipHold(:residualCargoMeasurement->0.6
                                :hull->:single))

% Ship 4 - Yes, hold clean (qualitatively the same as for Ship 2)
:ship4#:Ship(:registryExpirationDate->phys:date(2020 1 1 )
            :totalLength->70
            :hold->:h4#:ShipHold(:residualCargoMeasurement->0.4
                                :hull->:single))

% Distinctions for :residualCargoMeasurement and :hull

```

```

% Ship 5 - No, hold not clean
:ship5#:Ship(:registryExpirationDate->phys:date(2020 1 1)
            :totalLength->90
            :hold->:h5#:ShipHold(:residualCargoMeasurement->0.6
                                :hull->:double))

% Ship 6 - No, size large yet hold single-hulled
:ship6#:Ship(:registryExpirationDate->phys:date(2020 1 1)
            :totalLength->90
            :hold->:h6#:ShipHold(:residualCargoMeasurement->0.4
                                :hull->:single))

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

% Facts with multiple reasons for No or Yes

% Three reasons for No

% Ship 8 - No, registry expired, hold not clean, and size large yet
% hold single-hulled
:ship8#:Ship(:registryExpirationDate->phys:date(2017 1 1)
            :totalLength->90
            :hold->:h8#:ShipHold(:residualCargoMeasurement->0.9
                                :hull->:single))

% Two reasons for No

% Ship 9 - No, hold not clean and size large yet hold single-hulled
:ship9#:Ship(:registryExpirationDate->phys:date(2018 1 1)
            :totalLength->90
            :hold->:h9#:ShipHold(:residualCargoMeasurement->0.9
                                :hull->:single))

```

```

% Ship 10 - No, registry expired and size large yet hold single-hulled
:ship10#:Ship(:registryExpirationDate->phys:date(2017 1 1)
              :totalLength->90
              :hold->:h10#:ShipHold(:residualCargoMeasurement->0.2
                                      :hull->:single))

% Ship 11 - No, registry expired and hold not clean
:ship11#:Ship(:registryExpirationDate->phys:date(2017 1 1)
              :totalLength->90
              :hold->:h11#:ShipHold(:residualCargoMeasurement->0.9
                                      :hull->:double))

% Two reasons for Yes

% Ship 12 - Yes, size small nevertheless hold double-hulled
:ship12#:Ship(:registryExpirationDate->phys:date(2020 1 1)
              :totalLength->60
              :hold->:h12#:ShipHold(:residualCargoMeasurement->0.1
                                      :hull->:double))

% Facts probing special cases

% Ship 13 - No, large ship must have some (a double) hull
:ship13#:Ship(:registryExpirationDate->phys:date(2020 1 1)
              :totalLength->120
              :hold->:h13#:ShipHold(:residualCargoMeasurement->0.2))

% Ship 14 - Yes, date, length, and measurement are at the threshold
:ship14#:Ship(:registryExpirationDate->phys:date(2017 5 7)
              :totalLength->80
              :hold->:h14#:ShipHold(:residualCargoMeasurement->0.5
                                      :hull->:double))
)
)

```

# Glossary

<b>argument, positional</b>	An argument whose meaning is determined by its position in a tuple
<b>argument, slotted</b>	See entry “slot”
<b>atom</b>	By default, it is the same as a psoa atom. In the context of FOL and Prolog, it is the same as an atomic formula.
<b>atom, oidless</b>	An atom that has no OID
<b>atom, oidful</b>	An atom that has an OID
<b>atom, psoa</b>	An atomic formula in the form of a psoa term
<b>conversion</b>	A translation step that transliterates constructs of the source language into constructs of the target language
<b>describution</b>	A transformation in PSOA that distributes the OID of an atom over its descriptors, see Definition 5.13
<b>descriptor</b>	A tuple or a slot of an atom

<b>descriptor, dependent</b>	A descriptor that is sensitive to a specific non-Top predicate in whose scope it occurs within an atom
<b>descriptor, independent</b>	A descriptor that is not sensitive to any specific non-Top predicate in whose scope it occurs within an atom
<b>engine</b>	A reasoning system used as an execution environment for the portation target in the portation metaframework and its specializations, see Section 4.2
<b>expression</b>	A function application, which in PSOA is an embedded oidless psOA term denoting an individual, see Section 3.1
<b>formula</b>	A language construct that denotes a truth value
<b>formula, atomic</b>	An elementary form of formulas that can be used to construct other formulas
<b>frame</b>	A structure that describes the information of an object in the form of attribute-value pairs
<b>normalization</b>	A transformation of constructs within a language, here in preparation for conversion to another language, see Section 4.1

<b>object-centered</b>	Object-centered modeling is a modeling paradigm of rule languages where knowledge is organized around objects. The difference between object-centered languages and object-oriented languages is that object-oriented languages usually allow information to be updated.
<b>object identifier</b>	A term in a psoa atom that is typed and can be associated with descriptors, abbreviated as OID
<b>objectification</b>	A transformation in PSOA that realizes the objectification restriction by transforming KBs and queries such that entailments can be established under a relaxed semantics in which the restriction is no longer required
<b>perspectivity</b>	The dimension of psoa atoms defined based on whether their descriptors are dependent or independent, see Section 3.2.1
<b>predicate</b>	In relational languages, a predicate is a relation. In RDF, a predicate (synonymous to ‘property’) corresponds to F-logic’s, RIF’s, and PSOA’s notion of ‘slot name’. In PSOA, the notion of predicate can be regarded as a generalization of RDF’s notion of class and FOL’s notion of relation.

<b>predicate, reserved</b>	A predicate in a translation's target language that is reserved to encode constructs from the source language. To avoid unintended modeling, if a translated KB needs to be extended with other user-defined KBs, the latter should not contain these reserved predicates.
<b>PSOA RuleML</b>	A homogeneous object-relational rule language that features the use of psOA terms, see Section 2.3.2
<b>reasoning system</b>	A system that performs logical reasoning (specifically, query answering in this dissertation)
<b>relational</b>	Relational modeling is a modeling paradigm of rule languages whose central idea is to model entity connections as a predicate over tuples
<b>restriction, description</b>	See Section 3.4, Change 5
<b>restriction, objectification</b>	A semantic restriction that requires each oidless atom to be equivalent to its existentially objectified form, see Section 3.4, Change 4
<b>restriction, semantic</b>	A restriction in a model-theoretic semantics that a model must conform to
<b>restriction, subclass</b>	See Section 5.5

<b>rule interoperation</b>	Methods that can make implementations of different rule languages, and knowledge represented in different languages work together. In this dissertation we focus on interoperation through translation between languages.
<b>slot</b> (also called “slotted argument”)	An attribute-value pair where the attribute describes the meaning of the value
<b>slot, dependent</b>	A dependent descriptor in the form of a slot
<b>slot, independent</b>	An independent descriptor in the form of a slot
<b>slotribution</b>	A specialized description transformation that distributes an OID of a frame over its slots
<b>term</b>	By default, it is the elementary building construct for all formulas. In the context of FOL, it is restricted to the elementary construct that denotes a domain individual.
<b>term, base</b>	A construct that can denote a domain individual
<b>term, psoa</b>	A term that can be interpreted as an expression or an atom. It, respectively, applies a function or predicate, for a predicate optionally identified by an OID, to a bag of tupled or slotted descriptors, see Sections 2.3.2 and 3.2.2.
<b>transformation</b>	A translation within the same language, see Definition 4.2

<b>transformation step</b>	A translation step within the same language
<b>translation</b>	See Definition 4.2
<b>translation step</b>	A translation that is not composed from other translations
<b>tuple</b>	A (possibly bracketed) ordered sequence of terms
<b>tuple, dependent</b>	A dependent descriptor in the form of a tuple
<b>tuple, independent</b>	An independent descriptor in the form of a tuple

# Vita

Candidate's full name: Gen Zou

Universities attended:

Tsinghua University, 2008, Master

Tsinghua University, 2004, Bachelor

Publications:

1. 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.), Proceedings of RuleML 2012, LNCS, Springer, Berlin, Heidelberg, vol. 7438, pp. 264–279
2. Zou, G., Peter-Paul, R., Boley, H., Riazanov, A.: PSOATransRun: Translating and Running PSOA RuleML via the TPTP Interchange Language for Theorem Provers. In: Ait-Kaci, H., Hu, Y.J., Nalepa, G.J., Palmirani, M., Roman, D. (eds.), RuleML2012@ECAI Challenge, CEUR Workshop Proceedings, vol. 874
3. Zou, G.: GeospatialRules: A Datalog<sup>+</sup> RuleML Rulebase for Geospatial Reasoning. In Patkos, T., Wyner, A., Giurca, A. (eds.): Challenge+DC@RuleML 2014, CEUR Workshop Proceedings, vol. 1211
4. Zou, G., Boley, H.: PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., Roman, D. (eds.) Proceedings of RuleML 2015, LNCS, Springer, Cham, vol. 9202, pp. 176–192
5. 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+DC@RuleML 2015, CEUR Workshop Proceedings, vol. 1417
6. Zou, G., Boley, H.: Minimal Objectification and Maximal Unnesting in PSOA RuleML. In: Alferes, J.J., Bertossi, L., Governatori, G., Fodor, P.,

Roman, D. (eds.), Proceedings of RuleML 2016, LNCS, Springer, Cham, vol. 9718, pp. 130–147

7. Zou, G., Boley, H., Wood, D., Lea, K.: Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic, In Bassiliades, N., Bikakis, A., Costantini, S., Franconi, E., Giurca, A., Kontchakov, R., Patkos, T., Sadri, F., Van Woensel, W. (eds.) Doctoral Consortium, Challenge, Industry Track, Tutorials and Posters@RuleML+RR, CEUR Workshop Proceedings, vol. 1875

Conference Presentations:

1. PSOA2TPTP: A Reference Translator for Interoperating PSOA RuleML with TPTP Reasoners, RuleML 2012, Montpellier, France.
2. PSOATransRun: Translating and Running PSOA RuleML via the TPTP Interchange Language for Theorem Provers, RuleML 2012, Montpellier, France.
3. A Translator Framework for the Interoperation of Graph plus Relational Data and Rules on the Web, 5th Atlantic Workshop on Semantics and Services, February 2014
4. GeospatialRules: A Datalog<sup>+</sup> RuleML Rulebase for Geospatial Reasoning, RuleML 2014, Prague, Czech Republic.
5. PSOA2Prolog: Object-Relational Rule Interoperation and Implementation by Translation from PSOA RuleML to ISO Prolog, RuleML 2015, Berlin, Germany.
6. PSOA RuleML Integration of Relational and Object-Centered Geospatial Data, RuleML 2015, Berlin, Germany.
7. Zou, G.: The PSOATransRun 1.0 System for Object-Relational Reasoning in RuleML. 6th Atlantic Workshop on Semantics and Services, December 2015
8. Minimal Objectification and Maximal Unnesting in PSOA RuleML, RuleML 2016, New York, USA
9. Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic, RuleML 2017, London
10. Port Clearance Rules in PSOA RuleML: From Controlled-English Regulation to Object-Relational Logic, Invited tutorial at OnTheMove Federated Conferences & Workshops 2017, Industry Case Studies Program