Cautiously Approaching SWRL

Bijan Parsia [a] , Evren Sirin [b] , Bernardo Cuenca Grau [a] ,
Edna Ruckhaus [a] , Daniel Hewlett [b]

[a] *University of Maryland, MIND Lab, 8400 Baltimore Ave, College Park MD 20742, USA*

[b] *University of Maryland, Computer Science Department, College Park MD 20742, USA*

Abstract

The Semantic Web Rules Language (SWRL) has recently been proposed as the basic rules language for the Semantic Web. SWRL is unusual in being an undecidable superset of a very expressive description logic - OWL DL ($\mathcal{SHOIN}(\mathbf{D})$) and a simple rules language, Datalog. In this paper, we attempt to evolve OWL DL toward SWRL, but cautiously and carefully. We strive to retain practical decidability. We seek to meet identifiable needs as simply as possible. We also attempt to make use of existing work on integrating rules systems, especially Datalog, with expressive description logics. We want to build a rules system that people with a large commitment to OWL will find understandable and useful. With such an aim, we explore different ways to give the users what they want, while staying "close" to OWL, and "cautiously far" from the full expressivity of SWRL.

Key words: Rules, Datalog, SWRL, Conjunctive ABox Query

1 Introduction

In Semantic Web circles it is commonly acknowledged that "rules are next," perhaps shortly after query (although, of course, rules and query are tightly connected). Since the Semantic Web is widely acknowledged to be a standards oriented initiative — with prime custodianship belonging to the World Wide Web Consortium (W3C) — there is a sense that the agenda is set, that capital-R Rules are urgently

Email addresses: bparsia@isr.umd.edu (Bijan Parsia), evren@cs.umd.edu (Evren Sirin), bernardo@mindlab.umd.edu (Bernardo Cuenca Grau), ruckhaus@ldc.usb.ve (Edna Ruckhaus), dhewlett@wam.umd.edu (Daniel Hewlett).

needed to "make progress" according to popular notions of Semantic Web "Architecture", and that their moment is now (or soon). That is, there is a plan, there is a need, and the time is now for a standard rules language for the Semantic Web.

There is also a sense of it being past time; of belatedness. Harold Boley's call to the logic programming community (1) is nearing its first decade. DAML-L(ogic) never materialized, but is a specter hanging over much of the DARPA Agent Markup Language program, with traces still visible in briefings, reports, and long neglected action items. Semantic Web Services (SWS) researchers complain of lacking critical expressivity.

The Resource Description Framework (RDF) and Web Ontology Language (OWL), now W3C recommendations, themselves complicate the landscape. For all their problems, oversights, and omissions, [1] they are firmly ensconced as the core Semantic Web languages. On certain views of Semantic Web architecture, OWL (in its various species) is (and was required to be) "layered" on RDF(S). Subsequent more (or other) expressive languages should be at least layered on RDF(S), and probably on OWL as well. [2] Even if the layering requirement is lifted, there will presumably be a great deal of RDF(S) and OWL knowledge bases, tools, experience, and mindshare to deal with. A rules proposal which does not take these into account essentially starts the Semantic Web effort over again. Unfortunately, while RDF(S) is perhaps inexpressive enough to be compatible with most rules proposals in some form or another, the OWL species are far too expressive to coexist with ease with standard rules systems. While there has historically been some work on integrating description logic systems with rules (e.g., production, a.k.a. "trigger" rules in Classic) it is only recently that effort has been made in understanding the various possible relationships between highly expressive description logics like OWL DL (roughly, $\mathcal{SHOIN}(\mathbf{D})$) and, e.g., Datalog or Horn logic in general (3) (4) (5).

Consider two recent proposals coming from the Semantic Web community for integrating rules and OWL: Description Logic Programs(6) (DLP) and the Semantic Web Rules Language(7) (SWRL). These proposals embody diametrically opposed integration approaches. DLP is the *intersection* of Horn logic and OWL, where as SWRL is (roughly) the *union* of them. In DLP, the resultant language is a very peculiar looking description logic and rather inexpressive language overall. It's hard to see the restrictions are either natural or satisfying. Contrariwise, SWRL retains the full power of OWL DL, but at the price of decidability and practical implementations. In general, we find the intersection approach to be a non-starter. Users tend, wisely or not, to use expressivity that is offered to them. For example, it tends to

[1] Commonly cited on the expressivity front as missing are, for RDF and OWL Full, literals in the subject position, and, for OWL overall, qualified number restrictions. In the rules community, the loudest complaint seems to be the lack of non-monotonicity. The Web Services Modeling Ontology initiative has abandoned RDF and OWL, at least to begin with, for the seemingly more rule friendly F-Logic.

[2] See the RDFS thesis in (2).

require a fair bit of discipline to stay within the OWL Lite species, and even within OWL DL. For OWL DL, the advantage is clear (decidability and excellent practical complete reasoners) over OWL Full. For OWL Lite, much use seems driven by implementation considerations, and often rather confused one (e.g., not recognizing the expressive implication of General Concept Inclusions axioms (GCIs)). If a rules language is to fit in with OWL, it has to minimize the compromises on the OWL side. Thus, we come at the rules problem as OWL people looking for something more, an expressive enhancement to our current, heavily OWL based toolkit. This is the natural perspective of an important chunk of the Web rules audience.

Thus, we take SWRL as our starting point. SWRL is our overarching framework in these investigations as it adds "a simple form of Horn-style rules" to OWL DL "in a syntactically and semantically coherent manner." As we are most concerned with preserving OWL knowledge bases and human expertise, SWRL is a natural fit. We would also like to reuse as much as our current infrastructure as possible, e.g., we have a DL reasoner (Pellet) and we do not really want to build a new from scratch (again). We want to *incrementally* add features from SWRL to OWL DL in such a way that our tools remain practical and useful for each extension. In our experience, without good tool support it is *very* difficult to really appreciate the capabilities and appropriate use of a Web knowledge representation (or any KR) language. Also, as a central theme — perhaps the dominant theme — of the Semantic Web is securing widespread adoption (a.k.a., "web scalability"), deployable systems are not a luxury, even in the exploratory stages.

We attempt to evolve OWL DL toward SWRL, but cautiously and carefully. We strive to retain practical decidability. We seek to meet identifiable needs as simply as possible. We also attempt to make use of existing work on integrating rules systems, especially Datalog, with expressive description logics. We want to build a rules system that people with a large commitment to OWL will find understandable and useful.

1.1 Some Desiderata for a Web Rules Language

While building on OWL is a core requirement for us (and, we believe, any future W3C Rules working group), there are a number of other, sometimes conflicting, desiderata articulated for a Web Rules Language. One criterion is the mirror image of our OWL requirement: People want to reuse their existing rulesets, implementations, and expertise. Call this the "reuse" criterion. Another is to add expressivity to OWL, for example, n-ary predicates, or role value maps, Call this, the "expressivity" criterion. Finally, people attribute certain desirable pragmatic features to rules, e.g., that they are easier to write and read and understand, or that they mesh better with our ways of thinking about certain problems, and so on. Call this the

"pragmatics" criterion. [3]

The pragmatics criterion is particularly interesting. OWL is a *very* expressive language, and description logics have, historically, claimed a number of pragmatic advantages for their notation (e.g., variable free syntax and counting quantifiers are obvious wins). But it is also our experience that even seasoned logicians and knowledge representation experts find description logics perplexing, misleading, and simply confusing, especially when it comes to what they can or cannot express. A clear win, then, would be to provide some sort of rule syntactic sugar for OWL DL axioms. We discuss one such approach in section 3. This approach has the advantage of remaining decidable, indeed, remaining entirely in OWL DL. No fundamental revision of reasoners required to process such rules, although the affordances of knowledge bases developed under the influence of rule sugar might require novel optimizations. The disadvantage, of course, is that no new expressivity is added. We discuss the use of conjunctive ABox query (8) as a means for supplying rule sugar in Section 3.3.

We can address the reuse criterion and the expressivity criterion together, and retain decidability, if we can loosely combine OWL ontologies with Datalog databases. In 4 we provide a framework for analyzing a number of proposals for combining description logics and Datalog, and relate them to SWRL. This family of formalisms, as subsets of SWRL, do add significant expressivity to OWL, and allow for an number of plausible techniques for integrating rule knowledge bases with terms from OWL ontologies.

In 5, we address some ways of dealing with the extensive set of SWRL built-in predicates, many of which would be useful in OWL right now.

Finally, in 6, we do a preliminary analysis of one of the most commonly cited applications of a rules language — policies, in particular, for Web Services.

1.2 Some Non-desiderata

A prominent use case for rule systems has been for the *implementation* of OWL itself. Seven out of thirteen OWL reasoner implementations listed in the Web Ontology Working Group's implementation report (9) are axiomatic, that is, "rule based".

[3] Note that the much of the pragmatics criterion can be seen as an aspect of the reuse criterion. For example, if I have a lot of skill in writing Prolog-like rules and a deep understanding of how they work, these are infrastructural advantages for Prolog-like rules. However, that experience, skill, and understanding contributes to the transparency and familiarity of Prolog-like rules for notable populations. We shall remain neutral on whether there are inherent psychological advantages to rules, or whether any pragmatic benefit is merely due to prior exposure and training.

(This list does not include the venerable, oft cited and used DAML/OWLJessKB. (10)) There is a lot of debate about this general approach to implementing OWL reasoning, but the general flavor is of *implementing* a reasoner using logic programming (often in Turing complete logic programming languages) rather than a translation of OWL into a more or equi-expressive formalism, with the appropriate additional background axioms (that is, they are very unlike the two uses of full first order logic reasoners listed in the report). We do *not* take it as a criterion that a Semantic Web Rules Language should be able to model or express OWL semantics, or to implement a decision procedure for OWL.

We also do not address non-monotonicity, defaults, conflict resolution, procedural attachments, or similar logic programming features. SWRL is a fragment of first order logic with first order semantics, as are RDF(S) and OWL, so there is momentum in that direction. Here, we are more concerned with seeing what we *can* do with simple (first order) extensions to OWL.

2 OWL DL and SWRL

OWL-DL corresponds to the expressive Description Logic $\mathcal{SHOIN}(\mathbf{D})$. In this section, we briefly define the syntax and semantics of OWL-DL and of SWRL, as an extension of it.

Definition 1 *(Syntax of $\mathcal{SHOIN}(\mathbf{D})$)*

Let V_C, V_{IP}, V_{DP}, V_D, V_I, V_{DV} be countable and pair-wise disjoint sets of class, object property, datatype property, datatype, individuals, and datatype value names respectively. An object property is either an object property name or its inverse, $R_i \cup \{R^-|R \in V_{IP}\}$. An object property axiom is either an inclusion axiom of the form $R_1 \sqsubseteq R_2$, for R_1, R_2 object properties, or a transitivity axiom of the form $Trans(R)$, for $R \in V_{IP}$. A datatype property axiom is an axiom of the form $T \sqsubseteq U$, where $T, U \in V_{DP}$. A role box $\Re$ is a finite set of object or datatype property axioms. An object property R is simple if it is not transitive and none of its sub-properties are transitive. Finally, a datatype D is either a datatype name in V_D or an enumeration of datatype values $\{d_1, ..., d_m\}, d_i \in V_{DV}$.

The set of $\mathcal{SHOIN}(\mathbf{D})$ classes is the smallest set such that:

(1) Every class name $A \in V_C$ is a class
(2) If C, D are classes and R is an object property, S is a simple object property, $o \in V_I$ is an individual name, $D \in V_D$ is a datatype, T is a datatype property, $d \in V_{DV}$ is a datatype value and n is a non-negative integer, then the following are also classes:
- *$\neg C$ (Negation), $C \sqcap D$(Conjunction), $C \sqcup D$ (Disjunction), $\{o\}$ (Nominals)*

Construct Name	OWL Syntax	DL Syntax & Semantics
Atomic Class	A(URI)	$A^I \subseteq \Delta^I$
Universal Class	owl:Thing	$\top^I = \Delta^I$
Object Property	R (URI)	$R^I \subseteq \Delta^I \times \Delta^I$
Datatype Property	U (URI)	$U^I \subseteq \Delta^I \times \Delta^D$
Datatype Name	D (URI)	$D^D \subseteq \Delta^D$
Data Range	OneOf($d_1, ..., d_n$)	$\{d_1, ..., d_n\}^D = \{d_1^D\} \cup ... \cup \{d_n^D\}$
Conjunction	intersectionOf(C,D)	$(C \sqcap D)^I = C^I \cap D^I$
Disjunction	unionOf(C,D)	$(C \sqcup D)^I = C^I \cup D^I$
Negation	ComplementOf(C)	$(\neg C)^I = \Delta^I / C^I$
someValues Restr.	restriction(R someValuesFrom(C))	$(\exists R.C)^I = \{x \in \Delta^I \mid \exists y \in \Delta^I, (x,y) \in R^I, y \in C^I\}$
allValues Restr.	restriction(R allValuesFrom(C))	$(\forall R.C)^I = \{x \in \Delta^I \mid \forall y \in \Delta^I, (x,y) \in R^I \rightarrow y \in C^I\}$
Transitive Prop.	ObjectProperty(R [Transitive])	$I \models Trans(R) \leftrightarrow R^I = (R^I)^+$
Object Prop. Hierarchy	subPropertyOf(R_1, R_2)	$I \models (R_1 \sqsubseteq R_2) \leftrightarrow R_1^I \subseteq R_2^I$
Dat. Prop. Hierarchy	subPropertyOf(U_1, U_2)	$I \models (U_1 \sqsubseteq U_2) \leftrightarrow U_1^I \subseteq U_2^I$
Inverse Property	ObjectProperty(Q [inverseOf(R)])	$(Q)^I = \{(x,y) \mid (y,x) \in R^I\}$
Nominals	OneOf($o_1, ..., o_n$)	$\{o_1, ..., o_n\}^D = \{o_1^I\} \cup ... \cup \{o_n^I\}$
Func. Object Prop.	ObjectProperty(R [Functional])	$I \models Funct(R) \leftrightarrow \forall a, b, c \in \Delta^I,$ $R^I(a,b) \wedge R^I(a,c) \rightarrow b = c$
Func. Data. Prop.	DatatypeProperty(U [Functional])	$I \models Funct(U) \leftrightarrow \forall a \in \Delta^I \wedge \forall b, c \in \Delta^D,$ $U^I(a,b) \wedge U^I(a,c) \rightarrow b = c$
Cardinality	restriction(S minCardinality(n))	$(\geq nS)^I = \{x \in \Delta^I, \|\|y, (x,y) \in S^I\|\| \geq n\}$
Restrictions	restriction(S maxCardinality(n))	$(\leq nS)^I = \{x \in \Delta^I, \|\|y, (x,y) \in S^I\|\| \leq n\}$
someValues Restr.	restriction(U someValuesFrom(C))	$(\exists U.D)^I = \{x \in \Delta^I \mid \exists y \in \Delta^D, (x,y) \in R^I, y \in D^I\}$
allValues Rest.	restriction(U allValuesFrom(C))	$(\forall U.D)^I = \{x \in \Delta^I \mid \forall y \in \Delta^D, (x,y) \in R^I \rightarrow y \in D^I\}$
Cardinality	restriction(U minCardinality(n))	$(\geq nS)^I = \{x \in \Delta^I, \|\|y, (x,y) \in U^I\|\| \geq n\}$
Restrictions	restriction(U maxCardinality(n))	$(\leq nS)^I = \{x \in \Delta^I, \|\|y, (x,y) \in U^I\|\| \leq n\}$

Table 1 OWL-DL Syntax and Semantics

- $\exists R.C$ *(someValues restriction),* $\forall R.C$ *(allValues restriction)*
- $\exists T.D$ *(someValues restriction),* $\forall T.D$ *(allValues restriction)*
- $\geq nS$*,* $\geq nU$ *(minCardinality restriction),* $\leq nS, \leq nU$ *(maxCardinality restriction)*

We use $\top$ *as an abbreviation for* $A \sqcup \neg A$*, and* $\bot$ *as an abbreviation for* $A \sqcap \neg A$*. Let* C, D *be classes, then the expression* $C \sqsubseteq D$ *is called a general concept inclusion axiom (GCI). A TBox,* T *is a finite set of GCIs. An ABox* A *is a finite set of assertions of the form* $C(a), R(a,b), U(a,d), a = b, a \neq b$*, where* $a, b \in V_I$*. A* $\mathcal{SHOIN}(\mathbf{D})$ *knowledge base (equivalent to an OWL-DL ontology) consists of a TBox* T*, an RBox* R *and an ABox* A*.*

Definition 2 *(Semantics)*

An interpretation I *is tuple* $I = (\Delta^I, \Delta^D, .^I, .^D)$*, where* Δ^I, Δ^D *are the object and datatype interpretation domains, which are disjoint* $\Delta^I \cap \Delta^D = \emptyset$*. The interpreta-*

tion functions $.^I$ and $.^D$ map each class name $A \in V_C$ to a subset A^I of Δ^I, each datatype name $D \in V_D$ to a subset D^D of Δ^D, each object property name R to a subset R^I of $\Delta^I \times \Delta^I$, and each datatype property name U to a subset U^I of $\Delta^I \times \Delta^D$

The interpretation functions can be inductively extended to complex constructs as shown in Table 1.

An interpretation I satisfies an object property inclusion axiom $R_1 \sqsubseteq R_2$ iff $R_1^I \subseteq R_2^I$, it satisfies a datatype property inclusion axiom $U_1 \sqsubseteq U_2$ if $U_1^I \subseteq U_2^I$, it satisfies a transitivity axiom $Trans(R)$ iff $R^I = (R^I)^+$, and it satisfies the GCI $C \sqsubseteq D$ iff $C^I \subseteq D^I$. The interpretation I satisfies the ABox assertion $C(a)$ if $a^I \in C^I$, the assertion $R(a, b)$ if $(a^I, b^I) \in R^I$, the assertion $U(a, d)$ if $(a^I, d^D) \in U^I$, and the assertions $a = b (a \neq b)$ if $a^I = b^I (a^I \neq b^I)$. The interpretation I is a model of the knowledge base iff it satisfies all the axioms in the RBox, TBox and ABox.

Throughout the paper we will use an example ontology that describes computers, monitors, computer accessories, and so on. As we go along and describe different methods for combining ontology and rule languages we will provide additional definitions for services that sell computers, preferences of customers who want to buy computers, and so on.

Here we provide the basic axioms that will help the reader understand the examples presented in later sections [4]:

$Computer \sqsubseteq Product$
$Monitor \sqsubseteq Product$
$Computer \sqsubseteq \exists hasCPU.CPU$
$CPU \sqsubseteq \exists hasSpeed.CPUSpeed$

$Customer \sqsubseteq Person$
$SalesService \equiv Service \sqcap \exists sells.Product$
$ExpensiveComputer = Computer \sqcap \exists hasPrice.HighPrice$

Definition 3 *(SWRL syntax) Let V_C, V_{IP}, V_{DP}, V_D, V_I, V_{DV} be countable and pairwise disjoint sets of class, object property, datatype property, datatype, individuals, and datatype value names respectively.*

Let V_{IX}, V_{DX} be a set of object and datatype variables respectively and let $V_{Built-In}$ a set of built-in names.

*A **SWRL object term** is either an object variable name or an individual name. Analogously, a **SWRL datatype term** is either a datatype value name or a datatype*

[4] The complete ontology is available at [http://www.mindswap.org/dav/ontologies/computer]

variable name.

*Let C, R, U, D be a $\mathcal{SHOIN}(\mathbf{D})$ class, object property, datatype property and datatype respectively. Let i, j be SWRL object terms, and $v, v_1, ..., v_n$ be SWRL datatype terms and let p be a built-in name. Then, the set of **SWRL atoms** is defined by the following grammar:*

$$Atom \leftarrow C(i)|D(v)|R(i,j)|U(i,v)|builtIn(p, v_1, ..., v_n)|i = j|i \neq j$$

*Let a and $b_1, ..., b_n$ be SWRL atoms. A **SWRL rule** r is an expression of the form:*

$$a \leftarrow b_1, ..., b_n$$

The atom a is the head of the rule, denoted by $H(r)$, while $b_1, ..., b_n$ is the body or antecedent of r, denoted by $A(r)$.

*Let Σ be a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base and P a set of SWRL rules. A **SWRL Knowledge Base** is a pair $K = (\Sigma, P)$*

Definition 4 *(SWRL semantics)*

*Let $I = (\Delta^I, \Delta^D, .^I, .^D)$ be a $\mathcal{SHOIN}(\mathbf{D})$ interpretation; a **binding** $B(I)$ is a $\mathcal{SHOIN}(\mathbf{D})$ interpretation that extends the interpretation functions $.^I$ and $.^D$ such that:*

$$V_{IX} \rightarrow P(\Delta^I)\ ;\ V_{DX} \rightarrow P(\Delta^D)$$

Where P is the powerset operator.

A Binding $B(I)$ satisfies the SWRL atoms according to Table 2, where C, R, U, D are a $\mathcal{SHOIN}(\mathbf{D})$ class, object property, datatype property and datatype respectively, i, j are SWRL object terms, v is a SWRL datatype term and p is a built-in name.

A Binding $B(I)$ satisfies the antecedent $A(r)$ of the rule r if it is empty or $B(I)$ satisfies every atom in $A(r)$. A Binding satisfies the consequent $H(r)$ if H is not empty and $B(I)$ satisfies the atom in it. A rule is satisfied by an interpretation I iff for every binding B such that $B(I)$ satisfies the antecedent, $B(I)$ also satisfies the consequent.

3 Rules as Syntactic Sugar

In this section we define a method of representing a subset of SWRL directly in OWL, while preserving its semantics. We do this by utilizing techniques used in

SWRL atom	Condition on Interpretation
$C(i)$	$i^I \in C^I$
$R(i,j)$	$(i^I, j^I) \in R^I$
$U(i,v)$	$(i^I, v^D) \in U^I$
$D(v)$	$v^D \in D^D$
$builtIn(p, v_1, ..., v_n)$	$(v_1^D, ..., v_n^D) \in p^D$
$i = j$	$i^I = j^I$
$i \neq j$	$i^I \neq j^I$

Table 2 SWRL atoms

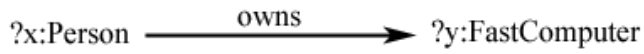
the processing of conjunctive ABox queries to transform rules into class axioms in DL. This approach allows for some rules to be treated as syntactic sugar for complex DL class expressions and axioms, but it imposes significant restrictions on the structure of these rules.

3.1 ABox Conjunctive Query

A conjunctive query is composed of a conjunction of terms, each of the form $x : C$ or $\langle x, y \rangle : R$, where x is either a variable or a named individual, y is a variable or a named individual, or literal value, C is a named concept, and R is a role. For example, the following query asks for all the people who own a fast computer:

$?x : Person \wedge \langle ?x, ?y \rangle : owns \wedge ?y : FastComputer$

This conjunction of terms can also be considered as a directed graph where each node is a variable, named individual, or literal value, and each edge is a role relation. This conception of the query is needed to describe the rolling-up technique. The graph representation of the query above is:



3.2 Rolling-up: Transforming Queries into Classes

Like other operations in DL systems, the answering of conjunctive queries can be reduced to KB satisfiability problem. The rolling-up technique is a procedure to generate a DL class expression that contains the constraints placed on a single variable $?x$ in a conjunctive query(8). This class expression is called the rolled-up class for $?x$. In the case of boolean queries, it is checked if the rolled-up class is satisfiable in every model of the KB. If we want to get the bindings for the

variables, we can replace the variables with individual names from the KB, and add that binding to the answer set if the resulting boolean query is a logical consequence of the KB.

Each application of the rolling-up technique has a target variable. The rolled-up class will capture the conditions placed on this target variable by the query graph. Using the example above, the class generated by rolling-up to the variable $?x$ would be $Person \sqcap \exists owns.FastComputer$.

The rolling-up technique works by traversing the query graph and representing each edge as an existential restriction. This traversal begins at the target variable, and moves outward in a depth-first manner. Each outgoing edge $\langle ?x, ?y \rangle : R$ is transformed into the class expression $\exists R.Y$, where Y is the conjunction of the named class restrictions on $?y$ in the query and the class generated by rolling-up all other edges into and out of $?y$. If $?y$ has no class restrictions and no unprocessed edges, Y is simply $\top$. This recursive process results in a single, complex class expression that incorporates all of the edges and class restrictions in the original query graph.

In order for the rolling-up technique to be applicable, constraints must be imposed on the query graph. The primary condition is that the query graph must be acyclic. There are more sophisticated methods that handle some cycles, such as cycles involving individual constants, that are beyond the scope of our approach (11).

3.3 Transforming Rules into Class Axioms

The rolling-up technique used in answering conjunctive queries can also be applied to a subset of SWRL, to gain some of the syntactic expressivity of rules without extending the semantic expressivity of OWL-DL. In this approach, the antecedent and consequent of a rule are each treated as a conjunctive query, and transformed into DL class expressions using the rolling-up technique. The addition of the assertion that the antecedent class expression is a subclass of the consequent class expression ensures the intended rule semantics.

This method exploits an overlap of OWL and SWRL expressivity. For rules where the antecedent and consequent both consist only of a single class expression applied to a shared variable, the rule is equivalent to a DL axiom asserting that the antecedent class is a subclass of the consequent class. Thus, the rule $Computer(x) :- FastComputer(x)$ is equivalent to the DL axiom $FastComputer \sqsubseteq Computer$. For this reason, rules of this form are often considered trivial within the DL context. However, when the class expressions involved are complex, the resulting axioms become non-trivial.

There are two ways to exploit this intersection of rule and DL expressivity. The first

would be to create, in DL, named classes that are equivalent to complex expressions in DL, and use these named classes in rules. However, after creating the complex rules within DL, the work required to create the subclass relation is trivial, making the rule formulation useless in general. Our approach is to create these complex classes dynamically from the rule definition, through the following process: First, the antecedent and consequent of the rule are each transformed into a conjunctive query, with the same single distinguished variable. Second, the resulting queries are transformed into class expressions using the rolling-up technique. Finally, these classes are added to the DL KB with the assertion that the antecedent class is a subclass of the consequent class. Consider the following rule, which describes a set of conditions that imply a given computer is a fast computer.

$$FastComputer(?c) :-$$
$$Computer(?c), hasCPU(?c, ?cpu), hasSpeed(?cpu, ?sp), HighSpeed(?sp).$$

The initial step, the transformation of the rule into two conjunctive queries, is straightforward. For the consequent and antecedent, each predicate in the rule maps directly to a conjunctive query term by the mapping below:

$$C(?x) \rightarrow ?x : C$$
$$R(?x, ?y) \rightarrow \langle ?x, ?y \rangle : R$$

Applying this mapping to the consequent of the example rule above simply yields $?c$:$FastComputer$. However, for the more complex antecedent, we obtain the following query:

$$?c : Computer \wedge \langle ?c, ?cpu \rangle : hasCPU \wedge \langle ?cpu, ?sp \rangle : hasSpeed \wedge ?sp : HighSpeed$$

Applying the rolling-up technique to each of the queries will yield the class expressions needed to represent the rule in DL. Both queries contain the variable $?c$, so this will be the only distinguished variable in each of the queries. Rolling-up the consequent produces the simple class expression $FastComputer$. Rolling-up the antecedent query to this variable generates the class expression $Computer \sqcap \exists hasCPU.\exists hasSpeed.HighSpeed$. To complete the representation of the original rule, we simply need to add the following subclass axiom to the KB:

$$Computer \sqcap \exists hasCPU.\exists hasSpeed.HighSpeed \sqsubseteq FastComputer$$

Note that, in SWRL semantics, an empty consequent is defined to be trivially false and an empty antecedent is defined to be trivially true. For these cases, we simply generate the expressions $\bot$ and $\top$, respectively.

We must define the set of rules (subset of SWRL) that can be translated directly into DL class axioms using this technique. Since the rolling-up technique is to be applied to both the antecedent and consequent, it is clear that each must satisfy the requirements of a conjunctive query, described above. However, further conditions

must be imposed to ensure that the subclass relation between the two resulting classes will have its intended semantics. These conditions are the following, depending on the number of variables shared between the consequent and antecedent:

- If 0 variables are shared, then the rule can be represented in OWL if at least one individual is shared. All variables in each query are undistinguished.
- If 1 variable is shared, then the rule can be represented in OWL. Only the shared variable is distinguished, and so is used as the target of rolling-up.
- If 2 or more variables are shared, then the rule can not be represented directly in OWL.

To understand the need for this condition, consider that a single application of the rolling-up technique results in a class expression. Classes are essentially one-place predicates that describe the conditions placed on a single variable. No single class can describe the conditions placed on multiple variables. It is for this reason that one class must be generated for each distinguished variable in a query.

3.4 Beyond Syntactic Sugar: Representing More Complex Rules

While the restrictions above still allow for the expression of non-trivial rules, a large subset of SWRL remains. Consider one of the generic examples that is used to show the use of rules, that is the definition of the `hasUncle` property:

$$hasUncle(?nephew, ?uncle) :- \\ hasParent(?nephew, ?parent), hasBrother(?parent, ?uncle).$$

We cannot simply use the above technique in this case because there are two distinguished variables in the head of the rule. Generating one subsumption axiom for each variable is not sufficient, because the resulting axioms will, even when taken collectively, not result in semantics equivalent to that of the original rule. For the `hasUncle` example, treating both shared variables, $?nephew$ and $?uncle$, as distinguished and applying the rolling-up technique yields the following class axioms, respectively:

For $?nephew$: $\exists hasParent.\exists hasBrother.\top \sqsubseteq \exists hasUncle.\top$
For $?uncle$: $\exists hasBrother^{-}.\exists hasParent^{-}.\top \sqsubseteq \exists hasUncle^{-}.\top$

These axioms are not sufficient to infer $hasUncle(Bob, Bill)$ from a KB that has the assertions $\{hasParent(Bob, Mary),\ hasBrother(Mary, Bill)\}$. Therefore, it is not possible to treat the definition of `hasUncle` property as syntactic sugar. However, this does not mean that we cannot express any of the definitions related to uncles. The following table shows some definitions that use the `hasUncle` property and which can be transformed into semantically equivalent OWL axioms that do not use hasUncle property explicitly:

Definition	**With explicit hasUncle property**	**Without explicit hasUncle property**
People who are uncles	$\exists hasUncle^{-}.\top$	$\exists hasBrother^{-}.\exists hasParent^{-}.\top$
People who have a funny uncle	$\exists hasUncle.Funny$	$\exists hasParent.\exists hasBrother.Funny$
People whose uncles are all lawyers	$\forall hasUncle.Lawyer$	$\forall hasParent.\forall hasBrother.Lawyer$
Bob has an uncle Bill	$\langle Bob, Bill\rangle$:$hasUncle$	Bob:$\exists hasParent.\exists hasBrother.\{Bill\}$

If the only reason for defining the hasUncle relation was to express these axioms then it is obvious that we do not need the power of rules to achieve that. It seems plausible to automate these transformations so a wider range of rules can be treated as syntactic sugar. As a preprocessing step, we could examine the rules in the ontology, find the rules that have a similar form to hasUncle definition and then transform all the axioms that involve these properties as shown above. The same transformation would also be done when a query is being answered so that the query $hasUncle(Bob, ?uncle)$ can automatically be expanded to $hasParent(Bob, ?parent), hasBrother(?parent, ?uncle)$.

It is possible that some ontologies that use the hasUncle definition can be handled this way while others cannot. If the given ontology has the subproperty assertion $hasUncle \sqsubseteq hasRelative$ then transformation is not possible because we cannot express this axiom without using the explicit hasUncle name. Another example would be the use of number restrictions. For example, the definition for "all the people who have exactly two uncles" can be written as $= 2hasUncle$. It is not generally possible to transform such number restrictions to the expanded version – there could be arbitrary number of parents (considering step parents) and each parent may have arbitrary number of brothers. The preprocessing step described above needs to detect such definitions and should apply the transformation only if it is possible to transform all the definitions.

4 Combining Ontology and Rules Languages

In this section we review the different approaches for combining rules formalisms and Description Logics. We present a general framework for combining Description Logics and Datalog. We show that SWRL, as well as many other proposals for combining DLs and Datalog are within this framework. Finally, we compare these "SWRL-compatible" approaches with other proposals (3) that define a different semantics for the coupling between the rules and the Description Logics components and discuss their advantages and drawbacks.

4.1 Combining Description Logics and Datalog

In this section, we provide a common framework, which includes many of the most prominent proposals for combining rules and Datalog, such as AL-Log (12) CARIN (4) , DL-safe rules (5) and, most importantly, SWRL. We show that, under certain conditions, families of combined formalisms, like AL-Log and CARIN languages, can be seen as fragments of SWRL.

All the proposals analyzed in this section are based on the same simple idea, namely, that the coupling between a DL Knowledge Base K and a Datalog program P is achieved by allowing the use of classes, object and datatype properties defined in K (called DL-atoms) in the Datalog rules in P. Different formalisms, with different expressivity and computational properties, can be defined by:

- Changing the expressivity of the DL language: For certain combinations, the use of $\mathcal{SHOIN}(\mathbf{D})$(OWL-DL) will result in a decidable language, whereas for others decidability can only be achieved if a less expressive DL language is used.
- Restricting the places where the DL-atoms can be used in the Datalog rules: For example, one may allow DL-atoms only in the antecedent of the Datalog rules
- Restricting the way variables can be combined in the Datalog rules using the so-called *safety conditions*. For example, one may require that all variables appearing in a DL-atom must also appear in a non-DL atom of the same rule.
- Restricting the use of predicates in the Datalog rules with a certain arity. For example, as in SWRL, one may disallow the use of predicates with arity greater than 2.

Different choices in the above points will yield to different combination languages. This way of combining DLs and Datalog does not provide a robust decidability of the combination; in other words, although the DL language being used and Datalog are decidable logics, the combination using this approach may (and frequently will) result in an undecidable formalism.

All the approaches described in this section are based on Datalog. Datalog (13) is a simple rule-based language, defined as follows:

Definition 5 *(Datalog Syntax)*

Let V_X be a set of variables, V_{cons} a set of constants, and V_{RP} a set of predicate symbols. A ***Datalog term*** *is any variable from V_X or constant symbol from V_{cons}. A* ***Datalog atom*** *is of the form $q(t_1, ..., t_n)$, where q is a predicate symbol of arity $n \geq 0$ in V_{RP} and $t_1, ..., t_n$ are datalog terms. A rule r is of the form:*

$$a \leftarrow b_1, ..., b_k$$

Where $a, b_1, ..., b_k$ are datalog atoms. The atom a is the head (consequent) of the

rule, denoted by $H(r)$, while $b_1, ..., b_k$ is the body (antecedent) of r, denoted by $A(r)$. A Datalog program P is a finite set of rules.

Datalog can be given a model-theoretic semantics by means of *interpretations*. An interpretation, in the context of Datalog, assigns meaning to constants and predicates. Rules are interpreted as true or false under different interpretations. In case a rule r is evaluated as true under the interpretation I, we say that I satisfies r, denoted by $I \models r$. In the case of Datalog, it is often more convenient not to define the semantics in terms of arbitrary interpretations, but to use only Herbrand interpretations [5], which in case of Datalog Programs, have a finite domain, called the Herbrand interpretation of the program.

Definition 6 *(Semantics)*

An interpretation is a pair $I = (\Delta^I, .^I)$, where Δ^I is a non-empty set, called the domain and $.^I$ is the interpretation function, which maps each constant $a \in V_{cons}$ to itself ($a^I = a \in \Delta^I$), each variable $x \in V_X$ to an element of the domain ($x^I \in \Delta^I$), and each predicate symbol q of arity n to an n-ary relation over the domain Δ^I ($p^I \subseteq \Delta^I \times .^n .. \Delta^I$).

An interpretation I satisfies a Datalog atom $q(t_1, ..., t_n)$, where p is an n-ary Datalog predicate symbol and the t_i are Datalog terms, if $(t_1^I, ..., t_n^I) \in p^I$. The interpretation I satisfies the antecedent $A(r)$ if it is empty or I satisfies every Datalog atom in $A(r)$. The interpretation I satisfies the consequent $H(r)$ if H is non-empty and I satisfies every Datalog atom in H. The interpretation I satisfies the Datalog rule r iff whenever I satisfies the antecedent, it also satisfies the consequent.

The Herbrand Base of a Datalog program P, denoted by HB_P is the set of all ground atoms with predicate and constant symbols appearing in P. We denote by $ground(P)$ the grounding of P w.r.t. HB_P. An (Herbrand) Interpretation I relative to P is a subset of the Herbrand Base. A model of a program P is an interpretation $I \subseteq HB_P$ such that $A(r) \subseteq I$ implies $H(r) \in I$ for every $r \in ground(P)$. An answer set is the least model of P w.r.t. set inclusion. Note that two different Herbrand interpretations share the same domain (which is finite in the case of Datalog) and only differ in the interpretation of the predicate symbols.

Now that we have defined the Description Logic $\mathcal{SHOIN}(\mathbf{D})$ and Datalog, we can define a general formalism that combines both as follows:

Definition 7 *Let V_C, V_{IP}, V_{DP}, V_D, V_I, V_{DV} be countable and pair-wise disjoint sets of class, object property, datatype property, datatype, individuals, and datatype value names respectively. Let V_{RP} be a set of predicate symbols.*

Let V_{IX}, V_{DX} be a set of object and datatype variables respectively and let $V_X =$

[5] This can always be done due to Herbrand Theorem

$V_{IX} \cup V_{DX}$

*An **object term** is either an object variable name or an individual name. Analogously, a **datatype term** is either a datatype value name or a datatype variable name. A **term** is either an object term or a datatype term.*

*Let C, R, U, D be a $\mathcal{SHOIN}(\mathbf{D})$ class, object property, datatype property and datatype respectively. Let i, j be object terms, and $v, v_1, ..., v_n$ be datatype terms Then, the set of **DL atoms** is defined by the following grammar:*

$$DL - Atom \leftarrow C(i)|D(v)|R(i,j)|U(i,v)$$

*A **Datalog atom** is of the form $q(t_1, ..., t_n)$, where q is a predicate symbol of arity $n \geq 0$ in V_{RP} and $t_1, ..., t_n$ are terms. An **atom** is either a DL or a Datalog atom.*

*Let a and $b_1, ..., b_n$ be atoms. A **rule** r is an expression of the form:*

$$a \leftarrow b_1, ..., b_n$$

The atom a is the head of the rule, denoted by $H(r)$, while $b_1, ..., b_n$ is the body or antecedent of r, denoted by $A(r)$.

*Let Σ be a $\mathcal{SHOIN}(\mathbf{D})$ knowledge base and P a finite set of rules. A **combined Knowledge Base** is a pair $K = (\Sigma, P)$.*

There are two equivalent ways for defining the semantics of the combined formalism. First, DL interpretations can be extended in order to deal with explicit variables and Datalog predicates of arbitrary arity. Another way to go would be to consider a DL interpretation for the DL KB and the DL-atoms in the rules, and an Herbrand Datalog interpretation for the Datalog rules, without considering the DL-atoms.

Definition 8 *(Semantics) Let $I = (\Delta^I, \Delta^D, .^I, .^D)$ be a $\mathcal{SHOIN}(\mathbf{D})$ interpretation; a **Binding** $B(I)$ is a $\mathcal{SHOIN}(\mathbf{D})$ interpretation that extends the interpretation functions $.^I$ and $.^D$ such that:*

$$V_{IX} \rightarrow P(\Delta^I) \text{ ; } V_{DX} \rightarrow P(\Delta^D) \text{ ; } V_X \rightarrow P(V_{DX} \cup V_{IX})$$

$$q^I \subseteq \Delta \times ..^n.. \times \Delta, \; q \in V_{RP} \text{ with arity } n$$

Where P is the powerset operator and $\Delta = \Delta^I \cup \Delta^D$.

A Binding $B(I)$ satisfies the DL-atoms according to Table 2, where C C, R, U, D are a $\mathcal{SHOIN}(\mathbf{D})$ class, object property, datatype property and datatype respectively, i, j are object terms, v is a datatype term. A Binding $B(I)$ satisfies the Datalog atom $q(t_1, ..., t_n)$ if $(t_1^I, ..., t_n^I) \in q^I$

A Binding $B(I)$ satisfies the antecedent $A(r)$ of the rule r if it is empty or $B(I)$

satisfies every atom in $A(r)$. A Binding satisfies the consequent $H(r)$ if H is not empty and $B(I)$ satisfies the atom in it. A rule is satisfied by an interpretation I iff for every binding B such that $B(I)$ satisfies the antecedent, $B(I)$ also satisfies the consequent.

An equivalent way to define the semantics of the combination is as follows. Let I be a $\mathcal{SHOIN}(\mathbf{D})$ interpretation. Let $K = (\Sigma, P)$ be a combined KB and P_D be the set of Datalog clauses obtained from the clauses in P by deleting in each clause all the DL atoms. Let $V_I(K), V_{DV}(K), V_{RP}(K)$ be respectively the set of individual, data value and Datalog predicate names appearing in K. The Herbrand Base of P_D, denoted by HB_{P_D} is the set of all Datalog atoms of the form $q(c_1, ..., c_n)$, where $c_i \in V_I(K) \cup V_{DV}(K)$, $q \in V_{RP}(K)$. Let H be a Herbrand interpretation with domain HB_{P_D}. An Interpretation J for K is the union of a $\mathcal{SHOIN}(\mathbf{D})$ interpretation I for Σ and d interpretation H for P_D. The interpretation $J = (I, H)$ is a model of K if $I \models \Sigma$ and for each rule in P, and for each of its ground instances either there exists one DL atom that is not satisfied by J or the rule obtained by suppressing the DL-atoms is satisfied by J.

4.1.1 Hybrid Systems: AL-Log and CARIN

In the mid and late 90s, many proposals for combining DLs and Datalog were presented under the name of "Hybrid Systems". The most prominent ones are AL-Log (12) and CARIN (4) . Here, we show that both can be seen as subsets of the general language presented above, and hence can be studied and understood within our framework.

$\mathcal{AL}$-Log was originally conceived as an Hybrid Knowledge Representation combining the Description Logic ALC and Datalog. The interaction between the subsystems is done through the specification of "constraints" in the Datalog rules, which are expressed in terms of $\mathcal{ALC}$ classes. $\mathcal{AL}$-Log constraints can be simply seen as unary DL atoms in our framework, and hence $\mathcal{AL}$-Log results in one of the simplest combined formalisms we can define. Here, we will slightly generalize $\mathcal{AL}$-Log in two ways: first, we will use $\mathcal{SHOIN}(\mathbf{D})$ instead of ALC on the DL side, and secondly we will also allow the use of OWL datatypes, and SWRL built-ins in the antecedent of Datalog rules.

Definition 9 *Let $K = (\Sigma, P)$ be a combined KB. We say that K is an **AL-Log KB** if DL atoms in P (if any) appear only in the antecedent of the rules, and are only of the form $C(t)$, $D(v)$, or $builtIn(q, v_1, ..., v_n)$, where C is a $\mathcal{SHOIN}(\mathbf{D})$ class, D an OWL datatype, q is a built-in predicate, t is an object term and $v, v_1, ..., v_n$ are datatype terms.*

Example 1 ($\mathcal{AL}$-Log rule) *Suppose that we want to define the set of services that are convenient for a customer as the set of services that provide fast delivery to the location the customer lives in. The following rule describes the predicate*

`convenient` and refers to the $Customer$ and $SalesService$ concepts from the DL component and uses the $livesIn$ and $fastDelivery$ predicates that are defined in the Datalog component.

$$convenient(?cust, ?serv) :-$$
$$livesIn(?cust, ?loc), fastDelivery(?serv, ?loc),$$
$$Customer(?cust), SalesService(?serv).$$

CARIN was presented as a Hybrid formalism that allowed the use of both classes and object properties in the antecedent of the Datalog rules, and hence it represents the most natural extension of $\mathcal{AL}$-Log. Here, we will also slightly extend CARIN by allowing OWL datatypes, OWL datatype properties and SWRL built-ins in the antecedent of the Datalog rules

Definition 10 *Let $K = (\Sigma, P)$ be a combined KB. We say that K is a **CARIN KB** if DL atoms in P (if any) appear only in the antecedent of the rules, and are only of the form $C(t)$, $D(v)$, $R(t_1, t_2)$, $U(u_1, u_2)$, $builtIn(q, v_1, ..., v_n)$, where C is a $\mathcal{SHOIN}(\mathbf{D})$ class, D an OWL datatype, R is an OWL object property, U is an OWL datatype property, q is a built-in predicate, t is an object term and $u_1, u_2, v, v_1, ..., v_n$ are datatype terms.*

Example 2 (Carin rule) *Consider a rule that says a customer will get a discount on printers if he/she has previously bought an expensive computer of the same brand. This rule can be defined as:*

$$discountAvailable(?cust, ?printer) :-$$
$$previouslyBought(?cust, ?comp), sameBrand(?comp, ?printer),$$
$$hasPrice(?comp, ?price), Customer(?cust), Printer(?printer),$$
$$Computer(?comp), HighPrice(?price).$$

Although, this rule looks quite similar to the $\mathcal{AL}$-Log rule note that we are now referring to `hasPrice` property which was defined in the DL component. In $\mathcal{AL}$-Log it was not allowed to use DL roles in the rule body.

4.1.2 SWRL

Definition 11 *Let $K = (\Sigma, P)$ be a combined KB. We say that K is a **SWRL KB** if the rules contain only DL-atoms.*

An AL-Log or a CARIN Knowledge Base can be seen as a SWRL Knowledge Base, provided that all the Datalog predicates it contains have at most arity 2 and that there is no binary predicate in the Datalog rules of the form $p(u, t)$, where u is a datatype term and t is an object term.

In this case, the unary Datalog predicates can be seen either as atomic classes or

datatypes, while the binary Datalog predicates could be treated as atomic object or datatype properties, or as built-ins. The Datalog facts concerning those predicates can then be handled as ABox assertions.

4.1.3 Safety Conditions

Safety conditions restrict the way variables can be combined within a Datalog rule. These restrictions are used in order to ensure the decidability of certain combinations. Safety conditions, when applied to a certain combined formalism, *do restrict* its expressive power. The most useful safety conditions are those that provide a convenient trade-off between the expressivity and computational properties, i.e. those that provide an additional expressive power useful for applications, while keeping the decidability of the formalism.

The first condition we define is *Datalog safety*, which is commonly appended to the definition of Datalog itself, since it ensures that the set of logical consequences of a Datalog program will always be finite. We will *always* assume this condition along the paper.

Definition 12 *(Datalog safety) Let P be a Datalog program. A rule r in P is Datalog safe if every variable appearing in the head of the rule also appears in the body.*

DL-safety conditions restrict the way variables can be used in DL-atoms in the Datalog rules. Thus, these conditions impose restrictions in the way the coupling between the DL and the Datalog components can be done in a combined KB. We distinguish three different safety conditions: the conditions we call *strong DL safety* and *weak DL safety* were originally defined in (5) and (12) respectively; finally, *role-safety* was used in (4) to ensure the decidability in some CARIN languages.

Definition 13 *(Strong DL-safety)*

*Let Σ be an OWL-DL ontology and P a Datalog program. A rule r in P is **strongly DL-safe** if each variable in r occurs in a non-DL atom in the rule body. The program P is strongly DL-safe if all its rules are strongly DL-safe.*

Definition 14 *(Weak DL-safety)*

*Let Σ be an OWL-DL ontology and P a datalog program. A rule r in P is **weakly DL-safe** if each variable in r occurs in a non-DL atom in the rule. The program P is weakly DL-safe if all its rules are weakly DL-safe.*

Note that strong DL-safety implies weak DL-safety, i.e. *weak* DL-safety is less restrictive than *strong* DL-safety.

	AL-Log	CARIN	CARIN	General Case
Unary DL-Atoms in Antecedent	$\surd$	$\surd$	$\surd$	strongly DL-safe
Unary DL-Atoms in Consequent	$\times$	$\times$	$\times$	strongly DL-safe
Binary DL-Atoms in Antecedent	$\times$	role-safe	$\surd$	strongly DL-safe
Binary DL-Atoms in Consequent	$\times$	$\times$	$\times$	strongly DL-safe
n-ary non-DL atoms	$\surd$	$\surd$	$\surd$	$\surd$
Most expressive OWL subset	$\mathcal{SHOIN}(\mathbf{D})$	$\mathcal{SHI}(\mathbf{D})$	DL-Lite$^-$	$\mathcal{SHOIN}(\mathbf{D})$

Table 3 Decidable Combinations

Definition 15 *(Role safety)*

Let Σ be an OWL-DL ontology and P a datalog program. A rule r in P is ***role safe*** *if, for each DL-atom p with arity 2 in the antecedent of r* ***at least one variable*** *in p also appears in a non-DL atom q in the antecedent of r and q never appears in the consequent of any rule in the program P*

The decidability of a combined formalism depends on the DL language used, the places where DL-atoms are allowed in rules and the safety conditions imposed. Table 3 summarizes the decidability results. $\mathcal{AL}$-Log is decidable even if no safety condition is required. If role-safety is imposed, then $CARIN$ is decidable for $\mathcal{SHI}(\mathbf{D})$. In (4) it was shown that, under DL-safety, *CARIN* was decidable for the logic $\mathcal{ALCNR}$. We argue, that the decision procedure presented in (4) as a proof of decidability can be adapted to $\mathcal{SHI}(\mathbf{D})$. The reason is that, on the one hand, the tableau expansion used in (4) for $\mathcal{ALCNR}$ can be replaced by the known tableau expansion for $\mathcal{SHI}(\mathbf{D})$, and, on the other hand, the finite canonical interpretation for each tableau completion can be constructed using the techniques presented in (4), since $\mathcal{SHI}(\mathbf{D})$ enjoys the finite model property.

If role-safety is not imposed, in order to regain decidability, the expressivity of the DL component must be restricted in CARIN. As a direct consequence of the results presented in (4), it is straightforward to show that CARIN would be decidable for DL-Lite, a sub-boolean DL with nice computational properties and very interesting expressivity for Semantic Web applications presented in (14), if functional properties are disallowed.

Finally, in (5) it was shown that if DL atoms are allowed at any position of the rules, but strong DL-safety is required, then query answering is decidable for OWL-DL.

4.2 Implementation

We have implemented a prototype of $\mathcal{AL}$-Log. The reasoner computes answers to queries based on the specification of both components and is based on the notion of *constrained SLD-derivation* and *constrained SLD-refutation*, as presented in (12). The system has been implemented in Prolog, coupled to our OWL reasoner Pellet. (15). We have tried two different strategies for the implementation, which are amenable to different kinds of optimizations:

- "Pre-processing approach": The key idea of this implementation is to pre-process all of the DL atoms that appear in the Datalog rules, and include them as facts in the relational subsystem. In order to cover all the possible models, if two or more rules have the same (obviously non-DL) atom in the head, and they also share a DL-atom in the body, whose (single) variable appears in the same argument of the head in the rules, then the disjunction of all those (unary) DL-atoms must also be computed and realized by the DL reasoner. Once the pre-processing is done, any query can be answered by the relational component using any of the known techniques for Datalog query evaluation.
- "On-the-fly" approach: This mode of implementation follows the query answering method described in (12). We have a meta-interpreter that models the computation process of a constrained SLD-refutation. For all of the constrained empty clauses in a constrained SLD-refutation of a query Q, disjunctions of classes may be constructed, and queries are processed using the Pellet reasoner. The derivation terminates once all the queries are processed.

The first approach shows that the procedure is independent of the DL language under consideration, and only relies on the existence of a decision procedure for such a DL. This shows that $\mathcal{AL}$-Log is decidable when combined with any decidable DL.

4.3 Alternatives to SWRL

We have seen so far that many of the approaches for combining rules with Description Logics can be seen as fragments of SWRL. However, one of the main drawbacks of SWRL and its fragments is that they are not robustly decidable, i.e, any tiny extension of a decidable fragment of SWRL can easily result in an undecidable combination. Another major drawback of SWRLisms is that, although the DL component is very expressive, the rules component is not, since it is restricted to Datalog.

It would be desirable to investigate a more flexible and robust way for combining DLs and rule languages. In (3) a new method is suggested for combining logic programming under the answer set semantics with the Description Logics $\mathcal{SHIF}(\mathbf{D})$

(OWL-Lite) and $\mathcal{SHOIN}(\mathbf{D})$(OWL-DL). This method provides a *technical separation* between the inferences in the Description Logic and the Logic Programming components, which result in formalisms that are robustly decidable, even in the case where the rules language is far more expressive than Datalog.

The combinations that can be defined using this method *cannot* be seen neither as supersets nor as subsets of SWRL or of any of its fragments, since the semantics of the coupling between the Description Logics and the rules components is *different* and *incompatible with* the semantics of SWRL.

In order to compare SWRL and DL-Programs, let us define the following DL-Program formalism:

Definition 16 *A DL atom $Q(t)$ is either an expression of the form $C(t)$ or $\neg C(t)$, where C is a concept and t is an object term, or an expression of the form $R(t_1, t_2)$, where R is an object property and t_1, t_2 are object terms. A DL-rule has the form:*

$$a \leftarrow b_1, ..., b_k$$

A Datalog rule where any b_i may be a DL-atom. A DL-Program $KB = (\Sigma, P)$ consists of a DL KB and a finite set of DL rules P

Syntactically, this language can also be seen as the (undecidable) CARIN fragment of SWRL. The difference is in the definition of the semantics:

Definition 17 *Let $KB = (\Sigma, P)$ a DL program. The Herbrand Base of P is the Herbrand Base of P ignoring the DL atoms. An interpretation I relative to P is a subset of HB_P. We say that I is a model of $l \in HB_P$ iff $l \in I$. We denote this by $I \models_\Sigma l$*

I is a model of a ground DL atom $a = Q(\bar{c})$ under Σ iff $\Sigma \models Q(\bar{c})$. We denote this $I \models_\Sigma a$. We say that I is a model of a DL-rule r iff $I \models_\Sigma H(r)$ whenever $I \models_\Sigma l$ for all $l \in B(r)$ (any of the atoms in $B(r)$ can be a DL atom). I is a model of a DL-Program $KB = (\Sigma, P)$ ($I \models KB$) iff $I \models_\Sigma r$ for all $r \in ground(P)$

The problem of determining whether a DL program has an answer set is decidable for the DLs $\mathcal{SHIF}(\mathbf{D})$ and $\mathcal{SHOIN}(\mathbf{D})$.

Lemma 1 *Let $K = (\Sigma, P)$ be a DL-Program. Let $Mod_1(K), Mod_2(K)$ be respectively the set of models of K under DL-programs and SWRL semantics. Then $Mod_1(K) \subseteq Mod_2(K)$*

The lemma is a consequence of the fact that, under DL-programs semantics, in order for a rule r in P to be satisfied, whenever the consequent is satisfied, *all* the models of Σ *must* satisfy all the DL-atoms in r. Under SWRL semantics, however, it suffices that at least one model of Σ satisfies the DL atoms in r.

The following lemma is a direct consequence of the above:

Lemma 2 *Let $Q = q_1, ..., q_n$ be a query composed of a conjunction of atoms. Let S_1, S_2 be respectively the set of correct answers to Q w.r.t the knowledge base $K = (\Sigma, P)$ under DL-programs and SWRL semantics, then $S_1 \subseteq S_2$*

This implies that, when using the DL-programs semantics, we are missing in general some answers to a given query. Let's illustrate this with an example. Consider the combined knowledge base shown in Figure 1 that has definitions for a computer sales services follows. The concept HQS reads as `High Quality Sales Service`, while DCS refers to a `Discount Computer Sales Service` which sells computers at low prices and does not have high ratings. The values 500 and 900 belong to LowPrice, 4000 is a HighPrice, 5 is a HighRating, and 2 is a LowRating.

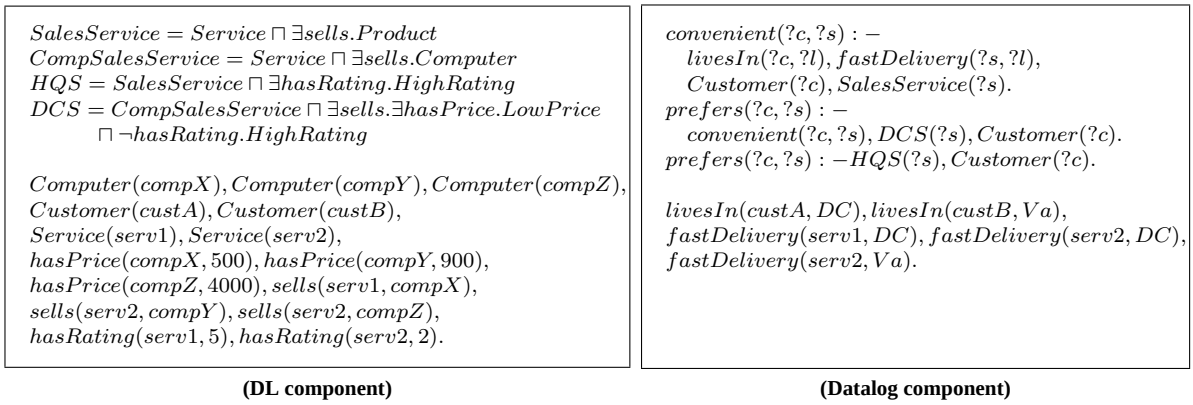


Fig. 1. A combined knowledge base for a computer sales services

Notice that in the $\mathcal{AL}$-log semantics, according to the second clause of $prefers$, both customers prefer $serv1$ because it has a high rating. However, no single rule for $prefers$ entails that customers prefer $serv2$. Customers prefer $serv2$ in either the set of models Mod_1 where it is true that it has a high rating, or the set of models Mod_2 where it is a convenient discount computer sales service. Therefore, $prefer(custA, serv2)$ and $prefer(custB, serv2)$ are entailed in the models $Mod_1 \cup Mod_2$ although both ground facts can not be entailed in any single set of models of K (*serv2* sells low price computers, and it has a low rating, but it can not be entailed that **all** of its rating values are not high).
On the other hand, in the Answer Set and DL Combination semantics, we would only entail $prefer(custA, serv1)$ and $prefer(custB, serv1)$ which are true in all of the models of Σ, therefore some of the ground facts are missed.

5 Adding Builtins

5.1 Built-ins in Datalog

Built-in predicates are special predicate symbols, like $>, <, \leq, \geq, =, \neq, +$, for concrete domains, such as integers and strings, that may occur in the body of a rule. Built-ins have a predefined logical meaning (a fixed interpretation) and can be considered as dynamically evaluated predicates, since they are implemented as procedures which are evaluated during the execution of a Datalog program. Built-in predicates are not evaluated until all its arguments are bound to constants.

The use of built-in predicates may compromise the safety of the rules, since they may yield to infinite relations. Safety can be ensured by requiring that each variable occurring as an argument in a built-in predicate has to appear in an "ordinary" predicate in the body of the rule, or it must be bound by an equality.

Built-in predicates can be used to define restrictions on single values:

$$SmallMonitor(?monitor) :-$$
$$Monitor(?monitor),$$
$$hasScreenSize(?monitor, ?size),$$
$$?size <= 15$$

or can be used to define a relation between two different values:

$$WideScreenMonitor(?monitor) :-$$
$$Monitor(?monitor),$$
$$hasWidth(?monitor, ?width), hasHeight(?monitor, ?height),$$
$$?height/?width < 0.75$$

5.2 Built-ins in DL's and OWL

The built-in datatype predicates can be used in DLs that support concrete datatypes.There are two basic ways proposed to combine concrete domains with DLs, the *concrete domain* approach proposed by Baader and Hanschke (16) and the *type system* approach proposed by Horrocks and Sattler (17).

The *type system* approach provides an easy way to combine DLs with XML Schema datatypes and closest to how OWL is combined with XML Schema datatypes. In this approach, datatypes are considered to be unary predicates over a universal concrete domain which is disjoint from the abstract domain (domain of individuals). Built-in predicates such as $>, <, \leq, \geq, =$ are considered to be part of the type sys-

tem that are used to derive new datatypes from existing ones. In this approach there is a separate datatype reasoner that checks if a conjunction of (possibly negated) datatypes is satisfiable or not. For example, the conjunction `xsd:PositiveInteger`$\wedge$ `xsd:NegativeInteger` is not satisfiable because there are no common values between two datatypes.

In OWL it is possible to use arbitrary XML Schema datatypes on datatype property restrictions. For example, we can say that `hasScreenSize` property is allowed to take values only from `xsd:PositiveInteger` datatype with the following global restriction $\top \sqsubseteq \forall hasScreenSize.$`xsd:PositiveInteger`. It is also possible to have the functionality of some of the built-in datatype predicates by using the constructors provided in XML Schema. In XML Schema, it is possible to derive new datatypes by defining restrictions on *facets*. For example, it is possible to define the set of positive integers less than 15 by defining a restriction on the `minInclusive` facet of `xsd:PositiveInteger`. Then, the definition for `SmallMonitor` predicate above can be expressed as:

$$Monitor \sqcap \exists hasScreenSize.\texttt{<=}_{15} \sqsubseteq SmallMonitor$$

where `<=15` is the corresponding derived XSD datatype description. Unfortunately, XSD facets are limited to express min and max value restrictions and regular expression based patterns for strings.

It is easy to extend the OWL DL reasoners with more built-in datatype predicates, especially when predicates are unary because it is a straightforward extension to XSD support. Note that the way XSD allows to define restrictions on datatypes effectively makes the built-in predicate, e.g. "$<=$", unary because all but one of the arguments of the built-in predicate are constant values, e.g. "$<= 15$". It is also easy to implement these extensions on existing reasoners because in the *type system* approach datatype reasoning is done by a separate datatype reasoner that can be modified without changing the reasoning procedure for the abstract domain.

It is also possible to have support for multi-arity datatype predicates in OWL by using extensions such as the one proposed in $\mathcal{SHOQ}(\mathbf{D_n})$ (18). $\mathcal{SHOQ}(\mathbf{D_n})$ extends the $\mathcal{SHOQ}(\mathbf{D})$ DL by allowing the use of n-ary predicates over concrete datatypes. Reasoning with $\mathcal{SHOQ}(\mathbf{D_n})$ is similar to $\mathcal{SHOQ}(\mathbf{D})$ but now we have the ability to express concepts where a relation between two datatype values is stated. For example, we can define the *Wide Screen Monitor* rule from previous section as:

$$Monitor \sqcap \exists hasHeight, hasWidth.\texttt{divide<}_{0.75} \sqsubseteq WideScreenMonitor$$

In the current specification of OWL and XML Schema, it is not possible to write such expressions but proposals like OWL-E show how OWL can be extended to allow such expressive datatype expressions.

5.3 Difference between built-ins in Datalog and DL

Although the semantics of built-ins in Datalog and DL's look quite similar there is a fundamental difference between two approaches. The Datalog approach applies built-in predicates only to existing values while DL reasoners can do reasoning with datatypes without any existing value. Therefore, DL reasoners can infer additional information even though there is no asserted value for a datatype property. Lets illustrate this with an example: Suppose that the monitors whose screen size is less than 20 inches are eligible for standard shipping (whereas heavier items need to be shipped with priority shipping). In Datalog, this fact would be with the following rule:

$$
\begin{aligned}
&EligibleForStandardShipping(?monitor) :- \\
&\quad Monitor(?monitor), \\
&\quad hasScreenSize(?monitor, ?size), \\
&\quad ?size <= 20
\end{aligned}
$$

which can be equivalently expressed in OWL with the following axiom:

$$Monitor \sqcap \exists hasScreenSize.\texttt{<=}_{20} \sqsubseteq EligibleForStandardShipping$$

A Datalog reasoner can infer that a monitor is eligible for standard shipping only if the value of the screen size for the monitor is known. On the other hand, if a DL reasoner knows that `SmallMonitor(monitor)` is true then it can infer `EligibleForStandardShipping(monitor)` is also true without knowing the exact value for the screen size. This is because the conjunction of the datatype expression $size <= 15$ (used in the definition of the `SmallMonitor`) and the datatype expression $size > 20$ (the negation of the expression used in the definition of the `EligibleForStandardShipping`) is not satisfiable. Thus, the DL reasoner can infer that there is no possible value for the `hasScreenSize` property such that a monitor is a `SmallMonitor`) but not `EligibleForStandardShipping`.

6 Policies

Policies on the Web are important because it is required to have access and security control on the resources in the distributed environment of the Web.

One area where policies are considered to be essential is the Web Services context. It is necessary to specify the conditions under which a user is allowed to invoke a service. It is generally argued that "rules" are essential for expressing policies. Let us examine one simple example to investigate this issue in more detail: There is a research lab in a university that performs astronomical observations and provides

the data via several different Web Services, S1, S2, S3, ... The access policy for S1 says that only lab members, the employees of the lab's university or an employee of an organization which funds the lab can use the services. These conditions can be expressed with the following rules:

$hasPermission(?p, S1) : -memberOf(?p, Lab).$
$hasPermission(?p, S1) : -$
$\quad hasAffiliation(?p, ?univ), hasSubOrganization(?univ, Lab).$
$hasPermission(?p, S1) : -employeeOf(?p, ?funder), funds(?funder, Lab).$

As we have already shown in Section 3, these rules can easily be expressed with the following OWL axioms:

$\exists memberOf.\{Lab\} \sqsubseteq \exists hasPermission.\{S1\}$
$\exists hasAffiliation.\exists hasSubOrganization.\{Lab\} \sqsubseteq \exists hasPermission.\{S1\}$
$\exists employeeOf.\exists funds.\{Lab\} \sqsubseteq \exists hasPermission.\{S1\}$

Although both representations encode exactly the same semantics, the rule representation looks more intuitive to humans than the OWL representation. It should also be noted that trying to write such policy rules in OWL is much harder. For example, one would be inclined to write the following axiom to encode the same policy rule:

$\{S1\} \sqsubseteq \forall hasPermission^{-}.(\exists memberOf.\{Lab\} \sqcup$
$\quad \exists hasAffiliation.\exists hasSubOrganization.\{Lab\} \sqcup$
$\quad \exists employeeOf.\exists funds.\{Lab\})$

This axiom obviously has a different meaning since it only specifies the necessary conditions for having the permission but does not specify them as sufficient conditions.

Treating rules as syntactic sugar for OWL lets us to have easier to understand policies and write while still using the sound and complete reasoners designed for OWL to evaluate the conditions in the policy. This approach also has a potential use for dealing with conflicting rules. A policy has a conflict if there is one rule giving permission to someone to perform a specific action and yet there is another rule prohibiting the same person performing that action [6] . Such conflicts would easily be detected by a DL reasoner as unsatisfiable concept descriptions. And theoretically, with the help of explanation/debugging tools being developed for OWL, the information about which two rules conflict can be computed and presented to users.

Despite the advantages of mapping the rule syntax to OWL axioms as we have discussed already discussed in Section 3 the rules that can be handled with this ap-

[6] These kind of conflicts are called "Conflicts of Modality" and there may be other kinds of conflicts if the policy language allows constructs to express obligations and dispensations

proach are limited. For example, we might want to write more generalized policy rules to specify permissions on a set of services. The research lab may be collaborating with other organizations on certain projects. The members of such partner organizations would be given access only to the services related to the specific joint project. The following rule encodes this policy:

$$hasPermission(?person, ?service) :- \\ relatedTo(?service, ?project), JointProject(?project), \\ participates(?org, ?project), memberOf(?person, ?org).$$

This rule cannot be directly mapped to OWL axioms but can easily be handled by the hybrid DL-Datalog systems. The consequence is that, in the hybrid system, the `hasPermission` property needs to be defined as a Datalog predicate which means no restrictions on this property can be defined in the DL side.

7 Conclusion

It is a common belief in the Semantic Web community that rules are needed for prominent Semantic Web applications, such as Web Services and Policies. The Semantic Web Rules Language (SWRL) can be roughly seen as the union of two decidable, yet orthogonal in expressivity, fragments of First Order Logic: OWL-DL ($\mathcal{SHOIN}(\mathbf{D})$) and Datalog. However, the expressive power provided by SWRL comes at the price of decidability, and hence of practical, sound and complete reasoning, which can be a critical requirement for the very same applications SWRL was designed for.

In this paper, we have shown that many of the user's needs can be satisfied without even getting close to the full expressivity of SWRL. We have shown that, actually, many of the rules that modelers would like to be able to write can be expressed in plain OWL-DL. However, although in these cases rules would be just "syntactic sugar" for OWL, the equivalent OWL expressions may become quite complex and counter-intuitive. We have presented the rolling-up technique as a method for transforming these rules into OWL class expressions in a way that can be implemented in OWL-based tools. We have shown that, even in the case where a certain property, such as *hasUncle* cannot be expressed in OWL, some of the restrictions in which such a property is used can still be expressed in OWL, without the need of defining such a property. Again, the resulting class expression can be non-obvious, since DLs were not originally conceived for such kind of modeling.

However, there are situations in which OWL-DL does not suffice anymore. We have presented a framework in which many of the most prominent proposals for integrating Description Logics and Datalog, including SWRL, can be expressed and understood. We have refined some of the existing decidability results for such

formalisms and developed a new technique for implementing $\mathcal{AL}$-Log formalisms using our DL reasoner Pellet. We have shown that many rules do actually fall within decidable formalisms, with the right modeling decisions.

In case more expressivity is required, even if more than plain Datalog is needed in the rules component, there are still ways to get along. We have compared SWRL semantics with the semantics presented in (3). We have shown that robust decidability and ease of implementation comes at a price of missing entailments, i.e, of incompleteness. For many applications the missing entailments may be critical, while for others they may not be.

Finally, we have shown the differences between the way a DL and a Datalog reasoner handle built-ins and discussed policies on the Web as an interesting use case for testing the expressivity required by rules.

References

[1] H. Boley, Knowledge bases in the world wide web: A challenge for logic programming (second, revised edition), Tech. Rep. TM-96-02, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Germany (1996).

[2] I. Horrocks, P. F. Patel-Schneider, Three theses of representation in the semantic web, in: Proc. of the WWW 2003, ACM, 2003, pp. 39–47.

[3] T. Eiter, T. Lukasiewicz, R. Schindlauer, H. Tompits, Combining answer set programming with description logics for the semantic web, in: Proc. of KR 2004, pp. 141–151.

[4] A. Levy, M.-C. Rousset, CARIN: A representation language combining horn rules and description logics, Artificial Intelligence 104 (1-2) (1998) 165–209.

[5] B. Motik, U. Sattler, R. Studer, Query answering for owl-dl with rules, in: Proc. of ISWC 2004, pp. 549–563.

[6] B. Grosof, I. Horrocks, R. Volz, S. Decker, Description logic programs: Combining logic programs with description logic, in: Proc. of WWW 2003.

[7] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A semantic web rule language combining OWL and RuleML, W3C Submission http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/ (2004).

[8] I. Horrocks, S. Tessaris, A conjunctive query language for description logic aboxes, in: Proc. of AAAI 2000, 2000, pp. 399–404.

[9] D. Connolly, J. Hendler, S. Hawke, Owl implementations, http://www.w3.org/2001/sw/WebOnt/impls (2003).

[10] J. Kopena, W. Regli, DAMLJessKB: A tool for reasoning with the semantic web, IEEE Intelligent Systems 18 (3) (2003) 74–77.

[11] S. Tessaris, Questions and answers: reasoning and querying in description logic, Ph.D. thesis, University of Manchester (2001).

[12] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, Al-log: integrating datalog

and description logics, Journal of Intelligent Information Systems 10 (1998) 227–252.

[13] S. Ceri, G. Gottlob, L. Tanca, What you always wanted to know about datalog (and never dared to ask), IEEE Transactions on Knowledge and Data Engineering 1 (1) (1989) 146–166.

[14] D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, G. Vetere, Dl-lite: Practical reasoning for rich DLs, in: Proc. of DL 2004, 2004.

[15] Pellet - OWL DL Reasoner, http://www.mindswap.org/2003/pellet.

[16] F. Baader, P. Hanschke, A scheme for integrating concrete domains into concept languages, in: Proc. of IJCAI-91, Australia, 1991, pp. 452–457.

[17] I. Horrocks, U. Sattler, Ontology reasoning in the $\mathcal{SHOQ}(\mathbf{D})$ description logic, in: B. Nebel (Ed.), Proc. of the IJCAI 2001, pp. 199–204.

[18] J. Z. Pan, I. Horrocks, Extending Datatype Support in Web Ontology Reasoning, in: Proc. of the ODBASE 2002), 2002.