Processing RIF and OWL2RL within DLVHEX*

Marco Marano[1], Philipp Obermeier[2], and Axel Polleres[2]

[1] Universita' della Calabria mmarano@deis.unical.it
[2] Digital Enterprise Research Institute, National University of Ireland, Galway
{philipp.obermeier, axel.polleres}@deri.org

Abstract. We present an extension of the DLVHEX system to support RIF-Core, a dialect of W3C's Rule Interchange Format (RIF), as well as combinations of RIF-Core and OWL2RL ontologies. DLVHEX is a plugin system on top of DLV, a disjunctive Datalog engine which enables higher-order and external atoms, as well as input rewriting capabilities, which are provided as plugins and enable DLVHEX to bidirectionally exchange data with external knowledge bases and consuming input in different Semantic Web languages. In fact, there already exist plugins for languages such as RDF and SPARQL. Our new plugin facilitates consumption and processing of RIF rule sets, as well as OWL2RL reasoning by a 2-step-reduction to DLVHEX via embedding in RIF-Core. The current version implements the translation from OWL2RL to RIF by a static rule set [12] and supports the RIF built-ins mandatory for this reduction trough external atoms in DLVHEX. For the future we plan to switch to a dynamic approach for RIF embedding of OWL2RL [2] and extend the RIF reasoning capabilities to more features of RIF-BLD. We provide a description of our current system, its current development status as well as an illustrative example, and conclude future plans to complete the Semantic Web library of plugins for DLVHEX.

1 Introduction

The W3C is currently developing *RIF* (*Rule Interchange Format*) [6], a universal layer designed for exchanging rules between different and possibly heterogeneous systems over the Semantic Web. It is focused on the exchange more than on the development of a single system to fit all needs of all the already available rule systems, because it appears clear that a system which fits all needs is very difficult, if not impossible to build, due to the large syntactic and semantic differences between different systems or even in different modules of the same system. The RIF working group divided the language into *dialects* which are meant to be used in different situations, while maintaining the largest subset of rules in common. They are called *RIF profiles*: *Core*, *BLD* and *PRD*. While Core is formed by the base constructs of the language, BLD (Basic Logic Dialect) is focused on logic, while PRD (Production Rules Dialect) is based on the concept of production rules. Among other features, by treating F-Logic like frames equivalently to RDF triples, particularly the RIF Core and RIF BLD fragments, promise a standard format for publishing and exchanging rules on top of RDF.

Likewise, ontologies in *OWL2RL*[10], a rule-based sublanguage of the Web ontology language *OWL2* [7], enables the support of inference over ontologies directly in rule-based system. This is achieved by giving a partial axiomatisation of the RDF OWL2 semantics in terms of first-order implications that can be encoded as rules.

* This work is partly funded by Science Foundation Ireland (SFI) project Lion-2 (SFI/08/CE/I1380) and an IRCSET postgraduate scholarship.

At the moment few implementations of OWL2RL and RIF-Core exist since both languages are quite new. Moreover, we are not aware of any implementations – as of yet – that implement the combinations of RIF and OWL as standardised [2].

To fill this gap we propose and implemented a reduction of those languages to *DLVHEX* [4], a powerful disjunctive logic reasoner based on the Answer Set Programming paradigm. DLVHEX has it roots in *DLV*, a disjunctive Datalog system, but adds several features to the base language. The most interesting of them is the possibility to use natively higher order atoms and external atoms, which are added to the core language by means of a plug-in architecture. Through external atoms it is possible to inject procedural code in the otherwise purely declarative semantics of the language. This concept is very similar to libraries for other reasoners which enable interaction with external data sources, such as, e.g., the integration of RDF support in SWI-Prolog [13]. There already exist a rich collection of DLVHEX plugins for Semantic Web languages, such as SPARQL [11], RDF and OWL DL [5]. Our new plugin for RIF-Core and OWL2RL not only expands the interoperability of DLVHEX with these two new standards, but also enables the combination of both with the other data models and extensions, already accessible by plugins, for an evaluation, experiments and new applications by combining these languages with the expressive features of Answer Set Programming [1, 3].

Our plugin allows DLVHEX to load and process RIF rule sets as well as OWL2RL ontologies. These are transformed to DLVHEX programs in a two-step translation: we first rewrite from OWL2RL to RIF-Core, and then perform a translation into DLVHEX. To this end there exist two different OWL2RL-to-RIF reduction methods, though, a static RIF rule set [12, Appendix 8.1] or dynamic a translation function from OWL2RL ontologies to RIF documents which yields RIF rules specifically to the input ontology [12, Appendix 8.2]. In comparison, the former approach bears some limitations in relation to interoperability with other RIF rule sets, and the combination of RIF with OWL ontologies as specified in [2] is rather based on the latter. Despite these restrictions, our current version of the OWL2RL reasoner transforms OWL2RL ontologies into RIF rules by the static rule set for the sake of a rapid first implementation. We will explain the limitations of this approach when doing it naively, and approximate the full dynamic combination of [2] by some extensions of the naive first translation.

In the following we give a description of our system, its current development status as well as accompanying examples in Section 2, and conclude with a report on our future plans in Section 3.

2 System Description

Our plugin consists of three parts: the OWL2RL to RIF-Core translation following [12], a RIF-Core to DLVHEX translator component, and the DLVHEX reasoner. In sequel we will provide more details to these components while we describe the system's workflow partitioned into its three essential stages:

Phase I - Translation from OWL2RL to RIF-Core An OWL2RL ontology, given in RDF/XML, as input is forwarded to the OW2RL to RIF-Core translator which translates RDF triples of the input ontology to RIF frames and merges them with the static rule set from [12] to a RIF-Core document. The application of the static rule set to the RIF frames gained from the input will be performed during the evaluation of this RIF document later on.

Phase II - Reduction of RIF-Core to DLVHEX The previously obtained RIF-Core document is preliminary reduced to a DLVHEX program. For that, the document is first parsed into an abstract syntax tree that is translated into a HEX program by a tree walking algorithm which gradually generates, adherent to a predefined set of translation rules, the corresponding HEX expressions from the visited tree nodes. This transformation includes reduction of features from RIF not directly expressible in our system to the processable input language of DLVHEX, e.g. Lloyd-Topor [8] transformation of rule bodies with discjunction, static type checking, or unnesting of external predicates, i.e. built-ins. Eventually, the generated program is forwarded to DLVHEX which returns a collection of answer sets.

Phase III - Answer Construction from DLVHEX to OWL2RL Eventually, the answer sets, which are basically sets of ground facts, are simply transformed into a set of RIF ground atomic formulas.

Example – RIF to DLVHEX

The OW2RL to RIF-Core translation, executed in Phase I is straightforward. We give here only a small example for the RIF-Core to DLVHEX translation, occurring in Phase II. We apply it here to a test case from the RIF development group, `http://www.w3.org/2005/rules/wiki/Factorial_Forward_Chaining`:

```
Document(
Prefix(pred <http://www.w3.org/2007/rif-builtin-predicate#>)
Prefix(func <http://www.w3.org/2007/rif-builtin-function#>)
Prefix(ex   <http://example.org/example#>)
Group
  (
    ex:factorial(0 1)

    Forall ?N ?F? ?N1 ?F1 (
        ex:factorial(?N ?F) :-
            And(External(pred:numeric-greater-than-or-equal(?N1 0))
              ?N = External(func:numeric-add(?N1 1))
                   ex:factorial(?N1 ?F1)
                   ?F = External(func:numeric-multiply(?N ?F1)) )
)  )  )
```

This document describes the computation of the factorial for a positive integer n. Our DLVHEX plugin rewrites the above RIF document into the following two DLVHEX rules:

```
"ex:factorial"("0", "1") :-  .
"ex:factorial"(VAR_N, VAR_F) :- &pred_numeric_geq[VAR_N1, "0"](),
                   equal(VAR_N, VAR_extOutput_1),
                   &func_numeric_add[VAR_N1, "1"](VAR_extOutput_1),
                   "ex:factorial"(VAR_N1, VAR_F1),
                   equal(VAR_F,VAR_extOutput_2),
                   &func_numeric_multiply[VAR_N,VAR_F1](VAR_extOutput_2) .
```

The translation generates two rules, a fact and a *proper* rule, corresponding to the two input RIF rules. The universal quantifier of the second RIF rule is omitted here since DLVHEX rules are per se universal. RIF constants (CURIes, typed literals, quoted unicode strings, etc.), such as `ex:factorial` or `1`, are embraced by double quotes. Prefix names in curies will generally be expanded, but for better readability we didn't resolve them here. RIF built-in predicates and functions, such as `pred:numeric-greater-than-or-equal` and `func:numeric-add`, are rewritten to an corresponding external DLVHEX atoms[3]. So far we support all RIF built-ins which may appear in a RIF

[3] Actually, for this particular example, we could have also exploited the built-in predicates of DLVHEX, which supports natively simple arithmetic functions such as sum, multiply and com-

document yielded by the OWL2RIF to RIF-Core translation. Beyond that, we also support all numeric predicates and functions implementable via calls to an XPath/XQuery Functions&Operators library. Besides, the lack of higher-order atoms in the resulting HEX program is no coincidence. In fact, those are not needed for a pure RIF-Core implementation. Our planned support for RIF-BLD as well as future RIF extensions similar to [5] will potentially demand higher-order features though.

Handling RIF-OWL2RL Combinations

The choice of a translation via the static rule set, applied in Phase I, seemed more convenient to implement at first view. since it supports a fast implementation. However, several limitations arise when translating OWL2RL into RIF via the static rules. Firstly, this method is rather inefficient compared to Reynolds's dynamic, pattern based approach [12, Appendix8.2], which creates more efficient RIF rules containing fewer free variables thus smaller grounding. Further and more problematic, the static rules as such are not suitable for RIF-OWL2RL combinations [2], i.e, a blend of OWL2RL rules with arbitrary RIF-Core rules. As pointed out in [2] the static rules create problems w.r.t. equality if applied to a RIF-OWL2RL combination, even if the RIF component is of RIF-Core. The reason lies in the possible introduction of equality through OWL2RL (via `[Object|Data]MaxCardinality` and `{Universe}FunctionalObjectProperty`) that can also affect the predicates existing in the RIF-Core component. In RIF-Core equality is only allowed in rule bodies and, thus, implications of equalities are not natively expressible. Likewise, our base system, DLVHEX, does not support equality natively, so we represent equality (which may only appear in rule bodies in RIF-Core) using `owl:sameAs`. This works out perfectly for the equality resembled by `owl:sameAs` on the level of RDF triples in the OWL2RL component [10, rules `eq-ref, eq-sym, eq-trans, eq-rep-s, eq-rep-p, eq-rep-o`], by axiomatisation in the OWL2RL rule set, but it is not comprehensively applicable in an analogous way to terms in RIF-Core, since arbitrary predicates or deeply nested external functions might occur in RIF rule sets which are unaffected by this axiomatisation.

Since we use the static rule set for the OWL2RL to RIF translation, at least for the time being, we developed a approximative rewriting for RIF rule sets for RIF-OWL2RL combination that allows us to catch these effects of equality. For a given RIF-OWL2RL combination $< R, G >$, where R is a RIF rule set and G is an RDF Graph, potentially encoding an OWL2RL ontology, our algorithm runs through the following steps and outputs a rewritten RIF-Core program S:

1. Initialise S with R. Flatten all nestings of external predicates and functions in S by recursive substitution of nested terms with variables. For that, we need to express various equalities between arbitrary function terms. However, owl:sameAs is only applicable to express equality between simple terms. Thus, we need to introduce a new equality symbol '$\doteq$' which expresses equality between arbitrary terms. Since the value of each function term, by definition, belongs to an XML datatype we can think of $\doteq$ as equality as evaluated by XPath.[4]

parisons between variables. For the sake of the example, though, we decided to show how the systems can handle such external predicates and functions, in a simple way

[4] In fact, on the stage of DLVHEX '$\doteq$' is evaluated by an external equality predicate implemented through XPath equality checks.

2. Add the static RIF-Core rule set of Reynolds to S.
3. Add G in form of frame facts to S.
4. For any constant c that appears in R but not in G add the fact `c[owl:sameAs->c] .` to S.
5. For each rule of R in S rewrite any occurring atom $p(t_1, ..., t_n)$ where p is a constant and t_i is a simple RIF term ($1 \leq i \leq n$) to an atom $p(X_1, ..., X_n)$ where $X_i = t_i$ if t_i is a variable, else (i.e., t_i is a constant) X_i is a fresh variable.
6. Apply Lloyd-Topr rewriting for non-conjunctive rule bodies in S.
7. Optimisation by removing unnecessary `owl:sameAs` and $\doteq$ statements from the rule bodies in S.

Let us illustrate the effects of this algorithm by an example. Say R[5] contains

```
p(?x) :- Or( q(?x) r(?x,b) ) .
r(c(2 * 2 + 2)).
q(a).
q(d) :- s( 1.3 + 0.7 ).
s(1+1).
```

and $G = \{(\texttt{a}, \texttt{owl:sameAs}, \texttt{b})\}$. Then we get the following intermediate results for S:

After step 1:

```
p(?x) :- Or (q(?x)  r(?x,b) ) .
r(c,?Y1) :- And( (?Y2 ≐ 2 * 2) (?Y1 ≐ ?Y2 + 2)   ).
q(a).
q(d) :- And( s( ?Y1 ) (?Y1 ≐ 1.3 + 0.7) ).
s(?Y1) :- (?Y1 ≐ 1 + 1).
```

After step 2: $S := S \cup$ "Static Rule Set"

After step 3: $S := S \cup \{\texttt{a[owl:sameAs->b]}\}$

After step 4: $S := S \cup \{\texttt{c[owl:sameAs->c]}, \texttt{2[owl:sameAs->2]}\}$

After step 5:

```
p(?x) :- And ( Or (q(?x)  r(?x,?X1) ) ?X1[owl:sameAs->b] ) .
r(?X1,?Y1) :- And( ?X1[owl:samAs->c] (?Y2 ≐ 2 * 2) (?Y1 ≐ ?Y2 + 2) ).
q(a).
q(?X1) :- And( ?X1[owl:samAs->d]  s( ?Y1 ) (?Y1 ≐ 1.3 + 0.7) ).
s(?Y1) :- (?Y1 ≐ 1 + 1).
```

After step 7:

```
p(?x) :- q(?x) .
p(?x) :- And ( r(?x,?X1) ?X1[owl:sameAs->b] ) .
r(?X1,?Y1) :- And( ?X1[owl:samAs->c] (?Y2 ≐ 2 * 2) (?Y1 ≐ ?Y2 + 2) ).
q(a).
q(?X1) :- And( ?X1[owl:samAs->d]  s( ?Y1 ) (?Y1 ≐ 1.3 + 0.7) ).
s(?Y1) :- (?Y1 ≐ 1 + 1).
```

Our translation is realised as a plugin[6] to the DLVHEX system[7]. Furthermore, RIF-Core contains many built-ins in form of external predicates and functions. These external

[5] Please note, that R deviates from the formal RIF syntax as we use here '+' and '$*$' for the built-in functions `func:numeric-add` and `func:numeric-multiply` in infix-notation for better readability

[6] For the source code and installation/usage instructions, please refer to `http://sourceforge.net/projects/dlvhex-semweb/` as well as `http://dlvhex-semweb.svn.sourceforge.net/viewvc/dlvhex-semweb/dlvhex-rifplugin/`.

[7] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/`

functions are computed by use of a standard XML Library that implements most of the common XPath/XQuery Functions& Operators [9]. At present, we support a subset of those, as we focused our attention on the built-ins which are mandatory for the reduction of OWL2RL reasoning to DLVHEX via RIF.

3 Conclusion and Future Work

We presented a DLVHEX plugin for OWL2RL and RIF-Core reasoning. The former is based on a 2-step reduction to DLVHEX via RIF-Core. This is, to our knowledge, the first attempt to implement RIF-OWL combinations a la [2], At our current stage of development we facilitate the translation to RIF by the static rule set of [12] which, as we have explained earlier, imposes restrictions on reasoning in combination with other RIF-Core documents. For the future we, therefore, will consider to modify the implementation of the first phase, switching from the static rule set to the dynamic rewriting by [12, Appendix8.2] similarly used in [2] which is based on RIF-BLD. Consequently, we will also try to extend the RIF-Core to DLVHEX translation in Phase II to more features of RIF-BLD. Moreover we plan to implement the remaining RIF built-ins to have a more complete translation from RIF-BLD to DLVHEX.

References

1. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2002.
2. J. De Bruijn. RIF rdf and OWL compatibility. Proposed recommendation, W3C, October 2009. http://www.w3.org/TR/2010/PR-rif-rdf-owl-20100511/.
3. T. Eiter, G. Ianni, A. Polleres, and R. Schindlauer. Answer set programming for the semantic web, June 2006. Tutorial at ESWC2006.
4. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *ESWC*, pages 273–287, 2006.
5. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining answer set programming with description logics for the semantic web. In *KR*, 2004.
6. M. Kifer and H. Boley. RIF basic logic dialect. Proposed recommendation, W3C, October 2009. http://www.w3.org/TR/2010/PR-rif-bld-20100511/.
7. M. Krötzsch, P. F. Patel-Schneider, S. Rudolph, P. Hitzler, and B. Parsia. OWL 2 web ontology language primer. Technical report, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.
8. J. W. Lloyd and R. W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.
9. A. Malhotra, J. Melton, and N. Walsh. XQuery 1.0 and XPath 2.0 functions and operators. Recommendation, W3C, January 2007. http://www.w3.org/TR/xpath-functions/.
10. B. Motik, A. Fokoue, I. Horrocks, Z. Wu, C. Lutz, and B. Cuenca Grau. OWL 2 web ontology language profiles. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/.
11. A. Polleres and R. Schindlauer. dlvhex-sparql: A sparql-compliant query engine based on dlvhex. In *2nd Int. Workshop on Applications of Logic Programming to the Web, Semantic Web and Web Services (ALPSWS2007*, pages 332–347. Springer, 2007.
12. D. Reynolds. OWL 2 RL in RIF. W3C working draft, W3C, October 2009. http://www.w3.org/TR/2009/WD-rif-owl-rl-20091001/.
13. J. Wielemaker, M. Hildebrand, and J. van Ossenbruggen. Prolog as the fundament for applications on the semantic web. In *ALPSWS*, 2007.