dlvhex: A System for Integrating Multiple Semantics in an Answer-Set Programming Framework[⋆]

Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits

Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A-1040 Vienna, Austria
{eiter,ianni,roman,tompits}@kr.tuwien.ac.at

Abstract. We briefly report on the development status of dlvhex, a reasoning engine for HEX-programs, which are nonmonotonic logic programs with higher-order atoms and external atoms. Higher-order features are widely acknowledged as useful for various tasks and are essential in the context of meta-reasoning. Furthermore, the possibility to exchange knowledge with external sources in a fully declarative framework such as answer-set programming (ASP) is particularly important in view of applications in the Semantic-Web area. Through external atoms, HEX-programs can deal with external knowledge and reasoners of various nature, such as RDF datasets or description logics bases.

1 Introduction

Nonmonotonic semantics is often requested by Semantic-Web designers in cases where the reasoning capabilities of the *Ontology layer* of the Semantic Web turn out to be too limiting, since they are based on monotonic logics. The widely acknowledged answer-set semantics of nonmonotonic logic programs [6], which is arguably the most important instance of the *answer-set programming* (ASP) paradigm, is a natural host for giving nonmonotonic semantics to the *Rules*, *Logic*, and *Proof* layers of the Semantic Web.

However, for important issues such as *meta-reasoning* in the context of the Semantic Web, no adequate answer-set engines have been available so far. Motivated by this fact and the observation that, furthermore, interoperability with other software is an important issue (not only in this context), in previous work [4], the answer-set semantics has been extended to HEX *programs*, which are *higher-order logic programs* (which accommodate meta-reasoning through *higher-order atoms*) with *external atoms* for software interoperability. Intuitively, a higher-order atom allows to quantify values over predicate names, and to freely exchange predicate symbols with constant symbols, like in the rule

$$C(X) \leftarrow subClassOf(D, C), D(X).$$

An external atom facilitates the assignment of a truth value of an atom through an external source of computation. For instance, the rule

$$t(Sub, Pred, Obj) \leftarrow \&RDF[uri](Sub, Pred, Obj)$$

computes the predicate t taking values from the predicate $\&RDF$. The latter predicate extracts RDF statements from the set of URIs specified by the extension of the predicate uri; this task is delegated to an external computational source (e.g., an external deduction system, an execution library, etc.). External atoms allow for a bidirectional flow of information to and from external sources of computation such as description logics reasoners. By means of HEX-programs, powerful meta-reasoning becomes available in a decidable setting, e.g., not only for Semantic-Web applications, but also for meta-interpretation techniques in ASP itself, or for defining policy languages.

Other logic-based formalisms, like TRIPLE [11] or F-Logic [9], feature also higher-order predicates for meta-reasoning in Semantic-Web applications. Our formalism is fully declarative and offers the possibility of nondeterministic predicate definitions with higher complexity in a decidable setting. This proved already useful for a range of applications with inherent nondeterminism, such as ontology merging (cf. [12]) or matchmaking, and thus provides a rich basis for integrating these areas with meta-reasoning.

[⋆] This work was partially supported by the Austrian Science Fund (FWF) under grant P17212-N04, and by the European Commission through the IST Networks of Excellence REWERSE (IST-2003-506779) and CologNeT (IST-2001-33123).

2 HEX-Programs

2.1 Syntax

HEX programs are built on mutually disjoint sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{G}$ of *constant names*, *variable names*, and *external predicate names*, respectively. Unless stated otherwise, elements from $\mathcal{X}$ (resp., $\mathcal{C}$) are written with first letter in upper case (resp., lower case), and elements from $\mathcal{G}$ are prefixed with "&". Constant names serve both as individual and predicate names. Importantly, $\mathcal{C}$ may be infinite.

Elements from $\mathcal{C} \cup \mathcal{X}$ are called *terms*. A *higher-order atom* (or *atom*) is a tuple $(Y_0, Y_1, \ldots, Y_n)$, where $Y_0, \ldots, Y_n$ are terms and $n \geq 0$ is its *arity*. Intuitively, Y_0 is the predicate name; we thus also use the familiar notation $Y_0(Y_1, \ldots, Y_n)$. The atom is *ordinary*, if Y_0 is a constant. For example, $(x, \mathit{rdf}{:}\mathit{type}, c)$ and $\mathit{node}(X)$ are ordinary atoms, while $D(a, b)$ is a higher-order atom. An *external atom* is of the form

$$\&g[Y_1, \ldots, Y_n](X_1, \ldots, X_m), \tag{1}$$

where $Y_1, \ldots, Y_n$ and $X_1, \ldots, X_m$ are two lists of terms (called *input list* and *output list*, respectively), and $\&g$ is an *external predicate name*.

It is possible to specify *molecules* of atoms in F-Logic-like syntax. For instance, $gi[\mathit{father} \rightarrow X, Z \rightarrow iu]$ is a shortcut for the conjunction $\mathit{father}(gi, X), Z(gi, iu)$.

HEX-programs are sets of rules of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \mathit{not}\, \beta_{n+1}, \ldots, \mathit{not}\, \beta_m, \tag{2}$$

where $m, k \geq 0$, $\alpha_1, \ldots, \alpha_k$ are higher-order atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms. The operator "not" is *negation as failure* (or *default negation*).

2.2 Semantics

The semantics of HEX-programs is given by generalizing the answer-set semantics [4]. The *Herbrand base* of a program P, denoted HB_P, is the set of all possible ground versions of atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{C}$. An *interpretation relative to* P is any subset $I \subseteq HB_P$ containing only atoms.

We say that an interpretation $I \subseteq HB_P$ is a *model* of an atom $a \in HB_P$ iff $a \in I$. Furthermore, I a model of a ground external atom $a = \&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ iff $f_{\&g}(I, y_1 \ldots, y_n, x_1, \ldots, x_m) = 1$, where $f_{\&g}$ is an $(n{+}m{+}1)$-ary Boolean function associated with $\&g$, called *oracle function*, assigning each element of $HB_P \times \mathcal{C}^{n+m}$ either 0 or 1.

This definition of satisfaction, together with a modified notion of a *reduct* as defined by Faber *et al.* [5], enables us to define a conservative extension of the answer-set semantics for HEX-programs. For more details, cf. [4].

Note that the answer-set semantics may yield no, one, or multiple models (i.e., answer sets) in general. Therefore, for query answering, *brave* and *cautious reasoning* (truth in some resp. all models) is considered in practice, depending on the application.

2.3 Usability of HEX-Programs

An interesting application scenario, where several features of HEX-programs come into play, is *ontology alignment*. Merging knowledge from different sources in the context of the Semantic Web is a crucial task [2] that can be supported by HEX-programs in various ways:

Importing external theories. This can be achieved as in the following manner:

$$\begin{aligned}
\mathit{triple}(X, Y, Z) &\leftarrow \&\mathit{RDF}[\mathit{uri}](X, Y, Z),\\
\mathit{triple}(X, Y, Z) &\leftarrow \&\mathit{RDF}[\mathit{uri2}](X, Y, Z),\\
\mathit{proposition}(P) &\leftarrow \mathit{triple}(P, \mathit{rdf}{:}\mathit{type}, \mathit{rdf}{:}\mathit{Statement}).
\end{aligned}$$

Searching in the space of assertions. In order to choose nondeterministically which propositions have to be included in the merged theory and which not, statements like the following can be used:

$$pick(P) \vee drop(P) \leftarrow proposition(P).$$

Translating and manipulating reified assertions. For instance, it is possible to choose how to put RDF triples (possibly including OWL assertions) in an easier manipulable and readable format, and to make selected propositions true such as in the following way:

$$\begin{aligned}(X, Y, Z) &\leftarrow pick(P), triple(P, rdf{:}subject, X),\\ &\qquad triple(P, rdf{:}predicate, Y),\\ &\qquad triple(P, rdf{:}object, Z),\\ C(X) &\leftarrow (X, rdf{:}type, C).\end{aligned}$$

Defining ontology semantics. The semantics of the ontology language at hand can be defined in terms of entailment rules and constraints expressed in the language itself or in terms of external knowledge, like in

$$\begin{aligned}D(X) &\leftarrow subClassof(D, C), C(X),\\ &\leftarrow \&inconsistent[pick],\end{aligned}$$

where the external predicate $\&inconsistent$ takes a set of assertions as input and establishes through an external reasoner whether the underlying theory is inconsistent.

Performing default and closed-world reasoning in a controlled way. Assuming that a generic external atom $\&DL[C](X)$ is available for querying the concept C in a given description logics base, the *closed-world assumption* (CWA) can be stated as follows:

$$C'(X) \leftarrow not\,\&DL[C](X), concept(C), cwa(C, C'),$$

where $concept(C)$ is a predicate which holds for all concepts and $cwa(C, C')$ states that C' is the CWA of C.

Inconsistency of the CWA can be checked by pushing back inferred values to the external knowledge base:

$$\begin{aligned}set\_false(C, X) &\leftarrow cwa(C, C'), C'(X),\\ inconsistent &\leftarrow \&DL1[set\_false](b),\end{aligned}$$

where $\&DL1[N](X)$ effects a check whether a knowledge base, augmented with all negated facts $\neg c(a)$ such $N(c, a)$ holds, entails the empty concept $\bot$ (entailment of $\bot(b)$, for any constant b, is tantamount to inconsistency).

3 Implementation

The evaluation principle of dlvhex is to split the program according to its dependency graph into components and alternately call an answer-set solver (DLV) and the external atom functions for the respective subprograms. The framework takes care of traversing the tree of components in the right order and combining their resulting models. Composing the initial dependency graph from a nonground program is not a trivial task, since higher-order atoms as well as the input list of an external atom have to be considered. To this end, we defined a novel notion of *atom dependency*, which extends the traditional understanding of dependencies within a logic program. This leads to novel types of *stratification* which help splitting a HEX-program and choosing the suitable model generation strategies.

Further methods of increasing the efficiency of computation include a general classification of external atoms regarding their functional properties. For instance, their evaluation functions may be *monotonic* or *linear* with respect to a given input. Formalizing such knowledge allows for an intelligent caching algorithm and thus for a reduction of interactions with the external computation source. Latest developments also include a directive to syntactically handle namespaces and an algorithm for traversing the component graph for disjunctive programs, eventually implementing the full HEX-program semantics.

To keep the development and usage of external atoms as flexible as possible, we decided to embed them into *plug-ins*, i.e., libraries that define and provide one or more external atoms. Such plug-ins are implemented as shared libraries, which link dynamically to the main application at runtime. A lean, object-oriented interface reduces the effort of developing custom plug-ins to a minimum.

Moreover, we devised an XML-based markup language for specifying HEX-programs, based on and extending RuleML (Rule Markup Language) [1]. We intend to integrate RuleML import and export mechanisms into dlvhex, as well as providing a Web-Service interface through standardized access mechanism such as SOAP (Simple Object Access Protocol).

3.1 Available External Atoms

The RDF Plug-In RDF (Resource Description Framework) is a language for representing information about resources in the World-Wide Web and is intended to represent meta-data about Web resources which is machine-readable and -processable. RDF is based on the idea of identifying objects using Web identifiers (called *Uniform Resource Identifiers*, or URIs), and describing resources in terms of simple properties and property values. *The RDF plug-in* provides a single external atom, the $\&RDF$ atom, which enables the user to import RDF-triples from any RDF knowledge base. It takes a single constant as input, which denotes the RDF-source (a file path or Web address).

The Description-Logics Plug-In Description logics are an important class of formalisms for expressing knowledge about concepts and concept hierarchies (often denoted as *ontologies*). The basic building blocks are *concepts*, *roles*, and *individuals*. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates interpreted as sets of objects. Roles are interpreted as binary relations between objects. In previous work [3], we introduced *dl-programs* as a method to interface description-logic knowledge bases with answer-set programs, allowing a bidirectional flow of information. To model dl-programs in terms of HEX-programs, we developed the *description-logics plug-in*, which includes three external atoms (these atoms—in accord to the semantics of dl-programs—also allow for extending the description logic knowledge base prior to the actual query by means of the atoms' input parameters):

– the $\&dlC$ atom, which queries a concept (specified by an input parameter of the atom) and retrieves its individuals,
– the $\&dlR$ atom, which queries a role and retrieves its individual pairs, and
– the $\&dlConsistent$ atom, which tests the (possibly extended) description logic knowledge base for consistency.

The description-logics plug-in can access OWL ontologies, i.e., description logic knowledge bases in the language $\mathcal{SHOIN}(\mathbf{D})$, utilizing the RACER reasoning engine [7].

3.2 Current Prototype

dlvhex has been implemented as a command-line application. It takes one or more HEX-programs as input and directly prints the resultant models as output. Both input and output are given in classical textual logic-programming notation. For the core reasoning process, dlvhex itself needs the answer-set solver DLV [10] (and DLT [8] if F-Logic syntax is used).

For illustrating the syntax and usage of dlvhex, consider the following HEX-program which models the search for personal contacts that stem from a *FOAF-ontology*,[1] which is accessible by a URL. The program uses the external atom $\&RDF$:

```
triple(X,Y,Z) :-
    &RDF["http://www.kr.tuwien.ac.at/staff/roman/foaf.rdf"](X,Y,Z).
iknow(X) :- triple("me","http://xmlns.com/foaf/0.1/knows",X).
friend(Y) :- iknow(X),triple(X,"http://xmlns.com/foaf/0.1/name",Y).
```

[1] "FOAF" stands for "Friend Of A Friend", and is an RDF vocabulary to describe people and their relationships.

Here, the first rule imports all triples found at the given URL into the logic-program predicate `triple`. The second rule singles out all values that are objects of triples with `"me"` as subject and `"http://xmlns.com/foaf/0.1/knows"` as predicate. The third rule further traverses the RDF-tree to single out the name-values of the found individuals of the second rule.

Assuming that this program is represented by the file `rdf.lp`, dlvhex is called as follows:

```
user@host:~> dlvhex --filter=friend rdf.lp
```

The `--filter` switch reduces the output of facts to the given predicate names. The result is a single answer set:

```
{friend("Axel Polleres"), friend("Francesco Calimeri"),
 friend("Wolfgang Faber")}
```

We will make dlvhex available both through source and binary packages. To ease becoming familiar with the system, we also offer a simple Web-interface available at

`http://www.kr.tuwien.ac.at/staff/roman/dlvhex`.

It allows for entering a HEX-program and filter predicates and displays the resultant models. On the same Web-page, we also supply a toolkit for developing custom plug-ins, embedded in the GNU autotools environment, which takes care for the low-level, system-specific build process and lets the plug-in author concentrate his or her efforts on the implementation of the plug-in's actual core functionality.

References

1. H. Boley, S. Tabet, and G. Wagner. Design Rationale for RuleML: A Markup Language for Semantic Web Rules. In *Proc. SWWS 2001*, pages 381–401, 2001.
2. D. Calvanese, G. D. Giacomo, and M. Lenzerini. A Framework for Ontology Integration. In *Proc. SWWS 2001*, pages 303–316, 2001.
3. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. Nonmonotonic Description Logic Programs: Implementation and Experiments. In *Proc. LPAR 2004*, pages 511–527, 2004.
4. T. Eiter, G. Ianni, R. Schindlauer, and H. Tompits. A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In *Proc. IJCAI 2005*. Morgan Kaufmann, 2005.
5. W. Faber, N. Leone, and G. Pfeifer. Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In *Proc. JELIA 2004*, pages 200–212, 2004.
6. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
7. V. Haarslev and R. Möller. RACER System Description. In *Proc. IJCAR 2001*, pages 701–705, 2001.
8. G. Ianni, G. Ielpa, A. Pietramala, M. C. Santoro, and F. Calimeri. Enhancing Answer Set Programming with Templates. In *Proc. NMR 2004*, pages 233–239, 2004.
9. M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.
10. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*. To appear.
11. M. Sintek and S. Decker. TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web. In *Proc. ISWC 2002*, pages 364–378, 2002.
12. K. Wang, G. Antoniou, R. W. Topor, and A. Sattar. Merging and Aligning Ontologies in dl-Programs. In *Proc. RuleML 2005* pages 160–171, 2005.