Answer Set Programming for the Semantic Web

Tutorial

Thomas Eiter, Roman Schindlauer (TU Wien)
Giovambattista Ianni (TU Wien, Univ. della Calabria)
Axel Polleres (Univ. Rey Juan Carlos, Madrid)

Supported by IST REWERSE, FWF Project P17212-N04, CICyT project TIC-2003-9001-C02.

Unit 6 – Another ASP Extension: HEX-Programs

R. Schindlauer

KBS Group, Institute of Information Systems, TU Vienna

European Semantic Web Conference 2006

Unit Outline

Motivation

- dl-programs: interfacing external source of knowledge
- Limited flexibility:
  - only one external KB possible
  - only one formalism allowed for KB (OWL)
- Spinning this idea further:
  - Access arbitrary external sources (solvers, services, different knowledge bases, etc.)
  - Standardized interface
  - Entire program: still ASP semantics
- Result: HEX-programs!

Motivation

- dl-programs: interfacing external source of knowledge
- Limited flexibility:
  - only one external KB possible
  - only one formalism allowed for KB (OWL)
- Spinning this idea further:
  - Access arbitrary external sources (solvers, services, different knowledge bases, etc.)
  - Standardized interface
  - Entire program: still ASP semantics
- Result: HEX-programs!

Motivation

- dl-programs: interfacing external source of knowledge
- Limited flexibility:
  - only one external KB possible
  - only one formalism allowed for KB (OWL)
- Spinning this idea further:
  - Access arbitrary external sources (solvers, services, different knowledge bases, etc.)
  - Standardized interface
  - Entire program: still ASP semantics
- Result: HEX-programs!

Motivation

- dl-programs: interfacing external source of knowledge
- Limited flexibility:
  - only one external KB possible
  - only one formalism allowed for KB (OWL)
- Spinning this idea further:
  - Access arbitrary external sources (solvers, services, different knowledge bases, etc.)
  - Standardized interface
  - Entire program: still ASP semantics
- Result: HEX-programs!

Motivation /2

Desirable Features for Rules in the Semantic Web:

- Software Interoperability
  - Importing external knowledge
  - Easily extendable reasoning framework
- Higher-Order Reasoning: rules that talk about predicates
  - Stating generic rules (e.g., general CWA rule)
  - Defining ontology semantics in a program

Our extension: *Higher-order logic programs with EXternal atoms* (HEX-programs) [26]

Motivation /2

Desirable Features for Rules in the Semantic Web:

- Software Interoperability
  - Importing external knowledge
  - Easily extendable reasoning framework
- Higher-Order Reasoning: rules that talk about predicates
  - Stating generic rules (e.g., general CWA rule)
  - Defining ontology semantics in a program

Our extension: *Higher-order logic programs with EXternal atoms* (HEX-programs) [26]

Motivation /2

Desirable Features for Rules in the Semantic Web:

- Software Interoperability
  - Importing external knowledge
  - Easily extendable reasoning framework
- Higher-Order Reasoning: rules that talk about predicates
  - Stating generic rules (e.g., general CWA rule)
  - Defining ontology semantics in a program

Our extension: *Higher-order logic programs with EXternal atoms* (HEX-programs) [26]

Motivation /2

Desirable Features for Rules in the Semantic Web:

- Software Interoperability
  - Importing external knowledge
  - Easily extendable reasoning framework
- Higher-Order Reasoning: rules that talk about predicates
  - Stating generic rules (e.g., general CWA rule)
  - Defining ontology semantics in a program

Our extension: *Higher-order logic programs with EXternal atoms* (HEX-programs) [26]

Syntax

Def. A HEX-program is a finite set P of rules:

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

$m, k \geq 0$, where $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms.

Higher-Order Atoms are expressions of the form

$$(t_0, t_1, \ldots, t_n) \text{ resp. } t_0(t_1, \ldots, t_n),$$

where $t_0, \ldots, t_n$ are (function-free) terms.

Examples: $(x, rdf{:}type, c)$, $node(X)$, $D(a, b)$.

Syntax

Def. A HEX-program is a finite set P of rules:

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

$m, k \geq 0$, where $\alpha_1, \ldots, \alpha_k$ are atoms, and $\beta_1, \ldots, \beta_m$ are either higher-order atoms or external atoms.

Higher-Order Atoms are expressions of the form

$$(t_0, t_1, \ldots, t_n) \ \text{resp.}\ \ t_0(t_1, \ldots, t_n),$$

where $t_0, \ldots, t_n$ are (function-free) terms.

Examples: $(x, \mathit{rdf}{:}\mathit{type}, c)$, $\mathit{node}(X)$, $D(a, b)$.

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&sum[p](X)$ $\Rightarrow$ $I : \{p(2), p(3), q(4)\}$ $\Rightarrow$ output list: 5
input list: p

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&sum[p](X)$ $\Rightarrow$ $I : \{p(2), p(3), q(4)\}$ $\Rightarrow$ output list: 5
input list: p

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&sum[p](X)$ $\Rightarrow$ $I : \{p(2), p(3), q(4)\}$ $\Rightarrow$ output list: 5
input list: p

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&sum[p](X)$ $\Rightarrow$ $I : \{p(2), p(3), q(4)\}$ $\Rightarrow$ output list: 5
input list: p

Syntax /2

External Atoms are expressions of the form

$$\&g[t_1, \ldots, t_n](t'_1, \ldots, t'_m),$$

where
- $\&g$ is an external predicate name, and
- $t_1, \ldots, t_n$ and $t'_1, \ldots, t'_m$ are lists of terms (*input*/*output* lists).

Intuition: Decide membership of $(t'_1, \ldots, t'_m)$ in the output depending on an interpretation I and parameters $t_1, \ldots, t_n$.

Example:

$\&sum[p](X) \quad \Rightarrow \quad I : \{p(2), p(3), q(4)\} \quad \Rightarrow$ output list: 5
input list: p

Examples:

$\&reach[edge, a](X)$ . . . reachable nodes from a in $edge$.

$\Rightarrow$ "Return 1 if $\langle a, X\rangle$ is in the extension of $edge$ in I."

$\&rdf[uri](X, Y, Z)$ . . . RDF-triples found under uri.

$\Rightarrow$ "Return 1 if $\langle X, Y, Z\rangle$ is an RDF-triple in uri."

(As already mentioned in Unit 4)

Examples:

$\&reach[edge, a](X)$ ...reachable nodes from a in $edge$.

$\Rightarrow$ "Return 1 if $\langle a, X \rangle$ is in the extension of $edge$ in I."

$\&rdf[uri](X, Y, Z)$ ...RDF-triples found under uri.

$\Rightarrow$ "Return 1 if $\langle X, Y, Z \rangle$ is an RDF-triple in uri."

(As already mentioned in Unit 4)

Semantics

Define semantics of P in terms of its grounding $grnd(P)$.

⇒ *Herbrand base* HB_P of P: set of all groundings of atoms and external atoms in P (relative to set of constants $\mathcal{C}$).

- $I \subseteq HB_P$ models ground atom a, if $a \in I$
- $I \subseteq HB_P$ models ground $\&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ 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 telling whether $(x_1, \ldots, x_m)$ is in the output for input $I, y_1 \ldots, y_n$.
- $I \subseteq HB_P$ models P iff it models $grnd(P)$
- $I \subseteq HB_P$ is an answer set of P iff I is a minimal model of fP^I, where fP^I is the FLP-reduct of P.

Semantics

Define semantics of P in terms of its grounding $grnd(P)$.

⇒ *Herbrand base* HB_P of P: set of all groundings of atoms and external atoms in P (relative to set of constants $\mathcal{C}$).

- $I \subseteq HB_P$ models ground atom a, if $a \in I$
- $I \subseteq HB_P$ models ground $\&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ 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 telling whether $(x_1, \ldots, x_m)$ is in the output for input $I, y_1 \ldots, y_n$.
- $I \subseteq HB_P$ models P iff it models $grnd(P)$
- $I \subseteq HB_P$ is an answer set of P iff I is a minimal model of fP^I, where fP^I is the FLP-reduct of P.

Semantics

Define semantics of P in terms of its grounding $grnd(P)$.

$\Rightarrow$ *Herbrand base* HB_P of P: set of all groundings of atoms and external atoms in P (relative to set of constants $\mathcal{C}$).

- $I \subseteq HB_P$ models ground atom a, if $a \in I$
- $I \subseteq HB_P$ models ground $\&g[y_1, \ldots, y_n](x_1, \ldots, x_m)$ 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 telling whether $(x_1, \ldots, x_m)$ is in the output for input $I, y_1 \ldots, y_n$.
- $I \subseteq HB_P$ models P iff it models $grnd(P)$
- $I \subseteq HB_P$ is an answer set of P iff I is a minimal model of fP^I, where fP^I is the FLP-reduct of P.

Applications

- Importing external theories, stored in RDF:

$$
\begin{aligned}
triple(X,Y,Z) &\leftarrow \&rdf[<uri1>](X,Y,Z);\\
triple(X,Y,Z) &\leftarrow \&rdf[<uri2>](X,Y,Z);\\
proposition(P) &\leftarrow triple(P, rdfs{:}type, rdf{:}Statement).
\end{aligned}
$$

$\Rightarrow$ Avoid inconsistencies when merging ontologies O_1, O_2.

- Translating and manipulating reified assertions:

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

Applications

- Importing external theories, stored in RDF:

$$\begin{aligned} triple(X,Y,Z) &\leftarrow \&rdf[<uri1>](X,Y,Z);\\ triple(X,Y,Z) &\leftarrow \&rdf[<uri2>](X,Y,Z);\\ proposition(P) &\leftarrow triple(P, rdfs{:}type, rdf{:}Statement). \end{aligned}$$

$\Rightarrow$ Avoid inconsistencies when merging ontologies O_1, O_2.

- Translating and manipulating reified assertions:

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

Applications /2

- Defining ontology semantics:

$$\begin{aligned} D(X) &\leftarrow subClassOf(D, C), C(X). \\ &\leftarrow maxCardinality(C, R, N), C(X), \\ &\quad \&count[R, X](M), M > N. \end{aligned}$$

- Closed World reasoning

$$\begin{aligned} cwa(faculty, project) &\leftarrow . \\ C'(X) &\leftarrow not\ \&DL[C](X), \\ &\quad concept(C), cwa(C, C'), \end{aligned}$$

Applications /2

- Defining ontology semantics:

$$\begin{aligned} D(X) &\leftarrow subClassOf(D, C), C(X). \\ &\leftarrow maxCardinality(C, R, N), C(X), \\ &\quad \&count[R, X](M), M > N. \end{aligned}$$

- Closed World reasoning

$$\begin{aligned} cwa(faculty, project) &\leftarrow . \\ C'(X) &\leftarrow not\ \&DL[C](X), \\ &\quad concept(C), cwa(C, C'), \end{aligned}$$

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety: Each variable in a rule must occur in a positive body literal.

Expansion safety: The input list of an external atom must be bounded:

$$triple(S, P, O) \leftarrow \&rdf[U](S, P, O), uri(U).$$

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety
: Each variable in a rule must occur in a positive body literal.

Expansion safety
: The input list of an external atom must be bounded:

$$triple(S, P, O) \leftarrow \&rdf[U](S, P, O), uri(U).$$

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety: Each variable in a rule must occur in a positive body literal.

Expansion safety: The input list of an external atom must be bounded:

$$triple(S, P, O) \leftarrow \&rdf[U](S, P, O), uri(U).$$

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety
: Each variable in a rule must occur in a positive body literal.

Expansion safety
: The input list of an external atom must be bounded:

$$triple(S,P,O) \leftarrow \&rdf[U](S,P,O), uri(U).$$
$$uri(X) \leftarrow triple(\_,''seeAlso'',X).$$

Unsafe!

Safety

If new values are imported by external atoms, how can we guarantee a finite domain?

By imposing safety restrictions! (see also [28])

"Traditional" safety: Each variable in a rule must occur in a positive body literal.

Expansion safety: The input list of an external atom must be bounded:

$$triple(S,P,O) \leftarrow \&rdf[U](S,P,O), uri(U), known(U).$$
$$uri(X) \leftarrow triple(\_,''seeAlso'',X).$$

Safe.

dlvhex

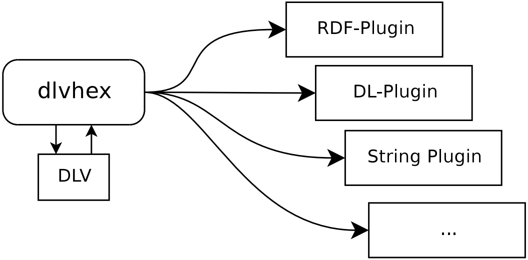
We implemented a reasoner for HEX-programs, called dlvhex [29].
⇒ Command line application, that interfaces DLV and plugins for external atoms used in a program.

Design principle:

dlvhex

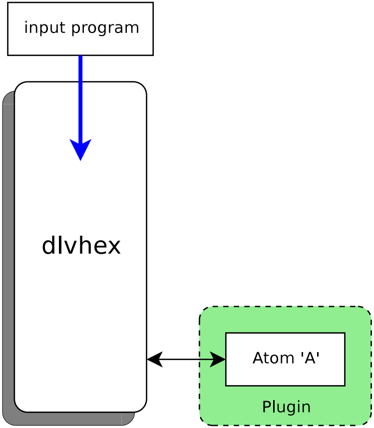
We implemented a reasoner for HEX-programs, called dlvhex [29].
⇒ Command line application, that interfaces DLV and plugins for external atoms used in a program.

Design principle:



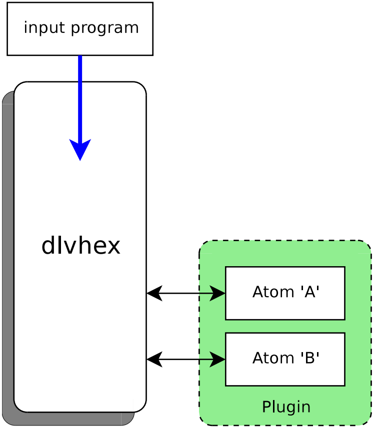
Plugin Mechanism

- A plugin is a shared library, dynamically linked at runtime
- A plugin can provide several Atoms
- A plugin can rewrite the program



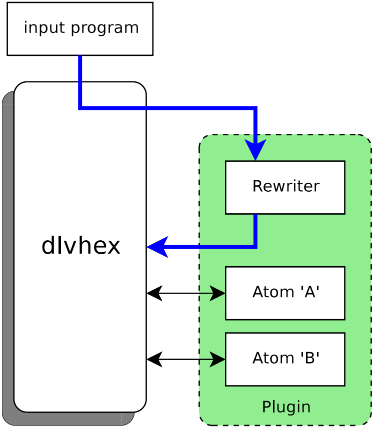
Plugin Mechanism

- A plugin is a shared library, dynamically linked at runtime
- A plugin can provide several Atoms
- A plugin can rewrite the program



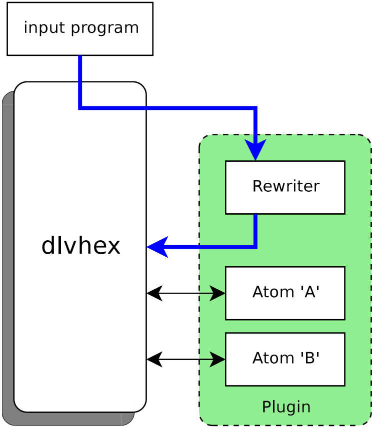
Plugin Mechanism

- A plugin is a shared library, dynamically linked at runtime
- A plugin can provide several Atoms
- A plugin can rewrite the program



Plugin Mechanism

- A plugin is a shared library, dynamically linked at runtime
- A plugin can provide several Atoms
- A plugin can rewrite the program



Plugin development toolkit available!

String Plugin

Purpose: String operations on names.

Available atoms:

`&concat`	Concatenates two strings.
`&cmp`	Compares two strings lexicographically or two numbers arithmetically.
`&strstr`	Tests two strings for substring inclusion.
`&split`	Splits a string along a specified delimiter.
`&sha1sum`	Computes SHA1 checksum from a string.

String Plugin Atoms

`&concat[A,B](C)` builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

String Plugin Atoms

`&concat[A,B](C)`	builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

String Plugin Atoms

`&concat[A,B](C)`	builds a string C from A + B.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

`&cmp[A,B]`	is true if A < B.

Example:
```
before(X,Y) :- &cmp[X,Y], date(X), date(Y).
date("2006-06-18").  date("2006-06-20").
```

String Plugin Atoms

`&concat[A,B](C)`	builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

`&cmp[A,B]`	is true if `A < B`.

Example:
```
before(X,Y) :- &cmp[X,Y], date(X), date(Y).
date("2006-06-18").  date("2006-06-20").
```

String Plugin Atoms

`&concat[A,B](C)` builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

`&cmp[A,B]` is true if `A < B`.

Example:
```
before(X,Y) :- &cmp[X,Y], date(X), date(Y).
date("2006-06-18").  date("2006-06-20").
```

`&strstr[A,B]` is true if A occurs in B.

Example:
```
isURL(X) :- &strstr["http://",X].
```

String Plugin Atoms

`&concat[A,B](C)`	builds a string `C` from `A + B`.

Example:
```
fullURI(X) :- &concat["http://",P](X),
              resourcepath(P).
```

`&cmp[A,B]`	is true if `A < B`.

Example:
```
before(X,Y) :- &cmp[X,Y], date(X), date(Y).
date("2006-06-18").  date("2006-06-20").
```

`&strstr[A,B]`	is true if A occurs in B.

Example:
```
isURL(X) :- &strstr["http://",X].
```

String Plugin Atoms /2

`&split[A,B,C](D)`	splits `A` by delimiter `B` and returns item `C`.

Example: `month(X) :- &split[D,"-",1](X), date(D).`
`date("2006-06-18").`

String Plugin Atoms /2

`&split[A,B,C](D)`	splits `A` by delimiter `B` and returns item `C`.

Example:
```
month(X) :- &split[D,"-",1](X), date(D).
date("2006-06-18").
```

String Plugin Atoms /2

`&split[A,B,C](D)`	splits A by delimiter B and returns item C.

Example:
```
month(X) :- &split[D,"-",1](X), date(D).
date("2006-06-18").
```

`&sha1sum[A](B)`	returns B as SHA1 checksum of A.

Example:
```
ownerID(X) :- &sha1sum[X](Y), mailbox(X).
```

String Plugin Atoms /2

`&split[A,B,C](D)`	splits A by delimiter B and returns item C.

Example:
```
month(X) :- &split[D,"-",1](X), date(D).
date("2006-06-18").
```

`&sha1sum[A](B)`	returns B as SHA1 checksum of A.

Example:
```
ownerID(X) :- &sha1sum[X](Y), mailbox(X).
```

RDF Plugin

Purpose: Access RDF knwoledge bases.

Available atom:

`&rdf` Retrieve RDF-triples from a URI.

RDF Plugin

Purpose: Access RDF knwoledge bases.

Available atom:

`&rdf` Retrieve RDF-triples from a URI.

`&rdf[U](S,P,O)`	returns all triples `<S,P,O>` from `U`.

Example:
```
tr(X,Y,Z) :- &rdf[U](X,Y,Z), uri(U).
uri("http://www.example.org/foaf.rdf").
```

RDF Plugin

Purpose: Access RDF knwoledge bases.

Available atom:

`&rdf` Retrieve RDF-triples from a URI.

`&rdf[U](S,P,O)`	returns all triples <S,P,O> from U.

Example:

```
tr(X,Y,Z) :- &rdf[U](X,Y,Z), uri(U).
uri("http://www.example.org/foaf.rdf").
```

RDF Plugin Example

Program `delicious_a.dlh` retrieves triples from a del.icio.us URI.

Del.icio.us is a social bookmarking service: Users store their bookmarks and tag them with keywords. It has an RDF/RSS-interface: adding a keyword to the URL `http://del.icio.us/rss/tag/` returns all bookmarks with this tag.

- The namespace directive abbreviates long strings, simple syntactic sugar.
- The single URL given returns all bookmarks with the tag `eswc`.

RDF Plugin Example

Program `delicious_a.dlh` retrieves triples from a del.icio.us URI.

Del.icio.us is a social bookmarking service: Users store their bookmarks and tag them with keywords. It has an RDF/RSS-interface: adding a keyword to the URL `http://del.icio.us/rss/tag/` returns all bookmarks with this tag.

- The namespace directive abbreviates long strings, simple syntactic sugar.
- The single URL given returns all bookmarks with the tag `eswc`.

Exercise

Task (1)

1. *Introduce a new predicate* `keyword` *and find a way to append its extension to the string* `"http://del.icio.us/rss/tag/"` *in order to build the URI in a more flexible way.*
2. *To get the actual bookmarks corresponding to a keyword, extract from the triples all resources that have* `"rdf:type"` *as property and* `"rss:item"` *as value.*

?

Exercise

Task (1)

1. *Introduce a new predicate* `keyword` *and find a way to append its extension to the string* `"http://del.icio.us/rss/tag/"` *in order to build the URI in a more flexible way.*
2. *To get the actual bookmarks corresponding to a keyword, extract from the triples all resources that have* `"rdf:type"` *as property and* `"rss:item"` *as value.*

```
tag("eswc").
url(X) :- &concat["http://del.icio.us/rss/tag/",W](X), tag(W).
```

Exercise

Task (1)

1. *Introduce a new predicate* `keyword` *and find a way to append its extension to the string* `"http://del.icio.us/rss/tag/"` *in order to build the URI in a more flexible way.*
2. *To get the actual bookmarks corresponding to a keyword, extract from the triples all resources that have* `"rdf:type"` *as property and* `"rss:item"` *as value.*

```
tag("eswc").

url(X) :- &concat["http://del.icio.us/rss/tag/",W](X), tag(W).
link(X) :- "rdf:type"(X,"rss:item").
```

Solution available as delicious_b.dlh

DL Plugin

Purpose: Query Description Logics knowledge bases.

Available atoms:

`&dlC`	Queries a DL concept.
`&dlR`	Queries a DL role.
`&dlDR`	Queries a DL datatype role.
`&dlConsistent`	Tests a DL KB for consistency.

These atoms descent from the corresponding dl-atoms of our dl-programs and also allow for extending the DL-KB.

DL Plugin

Purpose: Query Description Logics knowledge bases.

Available atoms:

`&dlC`	Queries a DL concept.
`&dlR`	Queries a DL role.
`&dlDR`	Queries a DL datatype role.
`&dlConsistent`	Tests a DL KB for consistency.

These atoms descent from the corresponding dl-atoms of our dl-programs and also allow for extending the DL-KB.

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms

`&dlC[U,a,b,c,d,Q](C)`	Returns all members of `Q` in KB `U`.

`a,b,c,d`: Predicates from the HEX-program, specifying the DL update, in this order:

1. Add `p` to `P` for each tuple `<P,p>` in the extension of `a`.
2. Add `p` to ¬`P` for each tuple `<P,p>` in the extension of `b`.
3. Add `<p,q>` to `R` for each tuple `<R,p,q>` in the extension of `c`.
4. Add `<p,q>` to ¬`R` for each tuple `<R,p,q>` in the extension of `d`.

Example:

```
student(X) :- &dlC[U,x,x,add,x,"PhdStudent"](X), url(U).
add("supervisorOf","Roman","Thomas").
```

DL Plugin Atoms /2

`&dlR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of `Q` in KB `U`.

`Q` has to be an ObjectProperty!

Example:

```
uncle(X,Y) :- &dlR[U,x,x,x,x,"brotherOf"](X,Z),
              &dlR[U,x,x,x,x,"parentOf"](Z,Y), url(U).
```

DL Plugin Atoms /2

`&dlR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of `Q` in KB `U`.

`Q` has to be an ObjectProperty!

Example:

```
uncle(X,Y) :-  &dlR[U,x,x,x,x,"brotherOf"](X,Z),
               &dlR[U,x,x,x,x,"parentOf"](Z,Y), url(U).
```

DL Plugin Atoms /2

`&dlR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of Q in KB U.

Q has to be an ObjectProperty!

Example:

```
uncle(X,Y) :-  &dlR[U,x,x,x,x,"brotherOf"](X,Z),
               &dlR[U,x,x,x,x,"parentOf"](Z,Y), url(U).
```

`&dlDR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of Q in KB U.

Q has to be a DatatypeProperty!

Example:

```
name(X,Y) :- &dlDR[U,a,b,c,d,"name"](X,Y),
             member(X), url(U).
```

DL Plugin Atoms /2

`&dlR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of `Q` in KB `U`.

`Q` has to be an ObjectProperty!

Example:

```
uncle(X,Y) :-  &dlR[U,x,x,x,x,"brotherOf"](X,Z),
               &dlR[U,x,x,x,x,"parentOf"](Z,Y), url(U).
```

`&dlDR[U,a,b,c,d,Q](X,Y)`	Returns all pairs of `Q` in KB `U`.

`Q` has to be a DatatypeProperty!

Example:

```
name(X,Y) :- &dlDR[U,a,b,c,d,"name"](X,Y),
             member(X), url(U).
```

Exercise

Wine example: importing wine preferences from (RDF) foaf-descriptions!

RDF-graph: `X <"foaf:name"> Name`
`X <"foafplus:winePreference"> Wine`

Task (2)

Modify wineCover10a.dlht by creating a predicate `preferredWine` *that associates names to wines.*

Exercise

Wine example: importing wine preferences from (RDF) foaf-descriptions!

RDF-graph:
```
X <"foaf:name"> Name
X <"foafplus:winePreference"> Wine
```

Task (2)

Modify wineCover10a.dlht by creating a predicate `preferredWine` *that associates names to wines.*

Exercise

Wine example: importing wine preferences from (RDF) foaf-descriptions!

RDF-graph:
```
X <"foaf:name"> Name
X <"foafplus:winePreference"> Wine
```

Task (2)

Modify wineCover10a.dlht by creating a predicate `preferredWine` *that associates names to wines.*

```
preferredWine(N,W) :- ?
```

Exercise

Wine example: importing wine preferences from (RDF) foaf-descriptions!

RDF-graph:
```
X <"foaf:name"> Name
X <"foafplus:winePreference"> Wine
```

Task (2)

Modify wineCover10a.dlht by creating a predicate `preferredWine` *that associates names to wines.*

```
preferredWine(N,W) :- "foaf:name"(X,N),
                      "foafplus:winePreference"(X,W).
```

Solution at wineCover10b.dlht

Policy Specification

Recent project using dlvhex: Policy Specification

P. A. Bonatti, D. Olmedilla, and J. Peer.:
Advanced Policy Queries. For: European Commission, IST 2004-506779 (REWERSE), I2-D4, 2005.

Principle:

- Grant access to resources based on disclosed credentials.
- Various combinations of credentials might lead to the same goal.
- Credentials have specific disclosure sensitivities.
- Optimization Problem: Find least sensitive combination of credentials that grant access!

Policy Specification

Recent project using dlvhex: Policy Specification

P. A. Bonatti, D. Olmedilla, and J. Peer.:
Advanced Policy Queries. For: European Commission, IST 2004-506779 (REWERSE), I2-D4, 2005.

Principle:

- Grant access to resources based on disclosed credentials.
- Various combinations of credentials might lead to the same goal.
- Credentials have specific disclosure sensitivities.
- Optimization Problem: Find least sensitive combination of credentials that grant access!

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  $\Rightarrow$ Use aggregates
- Complicated: based on respective credentials, mutual effects
  $\Rightarrow$ Use external atom!

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  $\Rightarrow$ Use aggregates
- Complicated: based on respective credentials, mutual effects
  $\Rightarrow$ Use external atom!

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  $\Rightarrow$ Use aggregates
- Complicated: based on respective credentials, mutual effects
  $\Rightarrow$ Use external atom!

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  $\Rightarrow$ Use aggregates
- Complicated: based on respective credentials, mutual effects
  $\Rightarrow$ Use external atom!

`&policy[sens](S)` returns overall sensitivity of pred. `sens`.

`sens` is binary, associating a sensitivity value to a credential.

Credential Selection

Challenge: Computing the overall sensitivity of a set of credentials.

- Simple: sum, average, maximum
  ⇒ Use aggregates
- Complicated: based on respective credentials, mutual effects
  ⇒ Use external atom!

`&policy[sens](S)` returns overall sensitivity of pred. `sens`.

`sens` is binary, associating a sensitivity value to a credential.

Example:
```
sens(ca,2).  sens(cb,3).
overall(S) :- &policy[sens](S).
```

Function inside the `&policy`-atom easily adaptable.

Policy Function Implementation: Sum

Simple version: Sum of all credential sensitivities—looking inside the plugin:

```
double
PolicySensFunction::eval(const std::vector<double>& values)
{
    double ret(0);

    for (vector<double>::const_iterator di = values.begin();
         di != values.end();
         ++di)
    {
        ret += *di;
    }

    return ret;
}
```

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint):* *For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint): For each* `availableFor(R,_)`*, we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint):* *For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint): For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint):* *For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

```
1 :- not allow(download,R), availableFor(R,_).
2 sensitivity(S) :- &policy[sens](S).
3 :~ sensitivity(S). [S:1]
```

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint): For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

```
1 :- not allow(download,R), availableFor(R,_).
2 sensitivity(S) :- &policy[sens](S).
3 :~ sensitivity(S). [S:1]
```

Exercise

Program policy_a.dlh creates a searchspace for all combinations of credentials → predicate `credential`

Task (3)

1. *remove models without granted access (strong constraint): For each* `availableFor(R,_)`, *we want* `allow(download,R)`*!*
2. *compute model sensitivity:* `sensitivity(S) :- ...`
3. *select least sensitive model with a weak constraint.*

1. `:- not allow(download,R), availableFor(R,_).`
2. `sensitivity(S) :- &policy[sens](S).`
3. `:~ sensitivity(S). [S:1]`

Solution at policy_b.dlh

A larger Example: Reviewer Selection

Let us now take a closer look on the reviewer selection example from Unit 5:

- We have a number of submissions and a program committe
- We have an ontology about publications and researchers:
  - classes like paper, kw, senior researcher, publication, . . .
  - properties like hasAuthor, keyword, publishedIn, firstname, . . .
- We want to assign reviewers combining these knowledge bases with HEX-programs instead of dl-programs now!

Reviewer Selection – Variant 1

Take the original program at reviewers1.dlp as a starting point

Task

Now, reformulate the program as a HEX *program!*

Solution: reviewers1.dlh

We add namespaces:

```
#namespace("rev","http://localhost/asptut/sandbox/reviewers.rdf#")
url("http://localhost/asptut/sandbox/reviewers.rdf").
...
author(subm1,"jdbr"). author(subm1,"htom").
author(subm1,"rev:jdbr"). author(subm1,"rev:htom").
```

We replace dl-atoms by HEX' DL plugin atoms:

```
c1("rev:club100",X) :- pc(X).
cand(X,P) :-  url(U), paper(P), &dlC[U,c1,c2,r1,r2,"rev:senior"](X).
```

Reviewer Selection – Variant 1

Take the original program at reviewers1.dlp as a starting point

Task

Now, reformulate the program as a HEX *program!*

Solution: reviewers1.dlh

We add namespaces:

```
#namespace("rev","http://localhost/asptut/sandbox/reviewers.rdf#")
url("http://localhost/asptut/sandbox/reviewers.rdf").
...
author(subm1,"jdbr"). author(subm1,"htom").
author(subm1,"rev:jdbr"). author(subm1,"rev:htom").
```

We replace dl-atoms by HEX' DL plugin atoms:

```
c1("rev:club100",X) :- pc(X).
cand(X,P) :-  url(U), paper(P), &dlC[U,c1,c2,r1,r2,"rev:senior"](X).
```

Reviewer Selection – Variant 1

Take the original program at reviewers1.dlp as a starting point

Task

Now, reformulate the program as a HEX *program!*

Solution: reviewers1.dlh

We add namespaces:

```
#namespace("rev","http://localhost/asptut/sandbox/reviewers.rdf#")
url("http://localhost/asptut/sandbox/reviewers.rdf").
...
author(subm1,"jdbr"). author(subm1,"htom").
author(subm1,"rev:jdbr"). author(subm1,"rev:htom").
```

We replace dl-atoms by HEX' DL plugin atoms:

```
c1("rev:club100",X) :- pc(X).
cand(X,P) :-  url(U), paper(P), &dlC[U,c1,c2,r1,r2,"rev:senior"](X).
```

Reviewer Selection – Variant 2

Now, let's have a closer look what happens in the DL ontology:

We add another PC member:

```
pc("rev:dknu").
```

File: reviewers2.dlh, filter the result by `cand`.

Task

Question: *Why has `"rev:dknu"` not been included in the candidate reviewers although we know he is in the Club100?*
Check the OWL ontology and find out why!

Solution:

$club100 \equiv person \geq 100 isAuthorOf$

$senior \equiv person \geq 3 isAuthorOf \sqcap \exists isAuthorOf.publication$

$publication \equiv paper \sqcap \geq 1 publishedIn$

There is no publication by dknu in the OWL KB!

Reviewer Selection – Variant 2

Now, let's have a closer look what happens in the DL ontology:

We add another PC member:

```
pc(“rev:dknu”).
```

File: reviewers2.dlh, filter the result by `cand`.

Task

Question: *Why has "rev:dknu" not been included in the candidate reviewers although we know he is in the Club100?*
Check the OWL ontology and find out why!

Solution:

$club100 \equiv person \geq 100 isAuthorOf$

$senior \equiv person \geq 3 isAuthorOf \sqcap \exists isAuthorOf.publication$

$publication \equiv paper \sqcap \geq 1 publishedIn$

There is no publication by dknu in the OWL KB!

Reviewer Selection – Variant 2

Now, let's have a closer look what happens in the DL ontology:

We add another PC member:

```
pc("rev:dknu").
```

File: reviewers2.dlh, filter the result by `cand`.

Task

Question: *Why has* `"rev:dknu"` *not been included in the candidate reviewers although we know he is in the Club100?*
Check the OWL ontology and find out why!

Solution:

$club100 \equiv person \geq 100 isAuthorOf$
$senior \equiv person \geq 3 isAuthorOf \sqcap \exists isAuthorOf.publication$
$publication \equiv paper \sqcap \ \geq 1 publishedIn$

There is no publication by dknu in the OWL KB!

Reviewer Selection – Variant 3

Next variation:

We add information about the authors of submitted papers to the the OWL KB.:

```
r1("rev:hasAuthor",P,A) :- author(P,A).
```

File: reviewers3.dlh, filter the result by `cand`.

Task

Effect: *H. Tompits (htom) and R. Schindlauer (rsch) also become canditates!*
Again: *Check the OWL ontology and find out why!*

Reviewer Selection – Variant 3

Next variation:

We add information about the authors of submitted papers to the the OWL KB.:

```
r1("rev:hasAuthor",P,A) :- author(P,A).
```

File: reviewers3.dlh, filter the result by `cand`.

Task

Effect: *H. Tompits (htom) and R. Schindlauer (rsch) also become canditates!*
Again: *Check the OWL ontology and find out why!*

Reviewer Selection – Variant 4/1

Last variation: Combines results of several queries.

Submissions as before but adding keyword information:

```
paper(subm1).
kw(subm1,"rev:Semantic_Web"). kw(subm1,"rev:OWL").
author(subm1,"rev:jdbr"). author(subm1,"rev:htom").

paper(subm2).
kw(subm2,"rev:Semantic_Web").
kw(subm2,"rev:Answer_Set_Programming").
author(subm2,"rev:teit"). author(subm2,"rev:gian").
author(subm2,"rev:rsch"). author(subm2,"rev:apol").
```

see reviewers4a.dlh

We now want to choose the review candidates candidates depending on keywords occurring in the submitted papers instead.

Reviewer Selection – Variant 4/2

Choose the review candidates candidates depending on keywords:

The OWL KB has properties defining

- keywords of papers "rev:keyword" and overlapping keywords "rev:overlapsWith"

Task

Modify the program reviewers4a.dlh as follows:

1. *A PC member who is author of a paper with the same keyword is a candidate.*
2. *A PC member who is author of a paper with an overlapping keyword as well.*

```
cand(X,P) :- ?
```

Reviewer Selection – Variant 4/2

Choose the review candidates candidates depending on keywords:

The OWL KB has properties defining

- keywords of papers "rev:keyword" and overlapping keywords "rev:overlapsWith"

Task

Modify the program reviewers4a.dlh as follows:

1. *A PC member who is author of a paper with the same keyword is a candidate.*
2. *A PC member who is author of a paper with an overlapping keyword as well.*

```
cand(X,P) :- kw(P,K), pc(X), url(U),
             &dlC[U,c1,c2,r1,r2,"rev:person"](X),
             &dlR[U,c1,c2,r1,r2,"rev:isAuthorOf"](X,P1),
             &dlR[U,c1,c2,r1,r2,"rev:keyword"](P1,K).

cand(X,P) :- kw(P,K), pc(X), url(U),
             &dlR[U,c1,c2,r1,r2,"rev:overlapsWith"](K,K1),
             &dlR[U,c1,c2,r1,r2,"rev:isAuthorOf"](X,P1),
             &dlR[U,c1,c2,r1,r2,"rev:keyword"](P1,K1).
```

Solution: reviewers4b.dlh

Let's continue with the hands-on session!