Fondements de la Programmation: Meta Logic Programming

Giovanni Coray - Vincenzo Pallotta

June 25, 2000

References:

- "From Logic Programming to PROLOG" - K. Apt, Prentice Hall, 1997
- "The Art of Prolog" - Sterling & Shapiro, MIT Press, 1986

Meta Interpretation of Logic Programs

Base Concepts:

- **Object Language** - the language used to represent the input of meta-interpreters
- **Meta-language** - the language used to write meta-interpreters and to encode the statements of the object language
- **Amalgamation** - to which extent it is possible to reuse objects from the meta-language into the object language.
- **Reflection principles** - when some object level statements can be considered as statements of the meta-language, it is possible to short-cut the meta-interpreter and directly interpret them using the meta-language semantics. Viceversa, since the meta-language may be more powerful, it could be useful to import some construct into the object language.

A simple meta-interpreter for Definite Logic Programs

- The *identity* meta-interpreter:

$$demo(X) \leftarrow X.$$

This program exploit the fact that logic programs are actually terms. Everything that can be proved without the meta-interpreter can be proved using the meta-interpreter (except $demo(X)$!).

- The $vanilla$ meta-interpreter:

$$\begin{aligned} demo(empty) &\leftarrow \\ demo(X,Y) &\leftarrow demo(X), demo(Y). \\ demo(X) &\leftarrow clause(X,Y), demo(Y). \end{aligned}$$

where V is the meta-representation of the object logic program P. V is obtained transforming all the clauses in P of the form:

$$\begin{aligned} p(\overline{X}) &\leftarrow q(\overline{Z}). \\ q(\overline{t}) &\leftarrow \end{aligned}$$

respectively in

$$\begin{aligned} clause(p(\overline{X}), q(\overline{Z})) &\leftarrow \\ clause(q(\overline{t}), empty) &\leftarrow . \end{aligned}$$

Semantic problems of meta-representation in LP

- We used the non-ground representation of object level terms.
- Variable are the same at both levels.
- We want to prove the semantic equivalence between object level programs and their meta-level representation.
- Meta-level variables should not be instantiated with terms not belonging to the Herbrand Base of object program.
  - For instance, in the Herbrand Base of V there can be atoms like $(p(\overline{t}), q(\overline{t}))$ which don't belong to HB_p.
  - A solution is to use a $ground$ representation of terms, where variables are represented by special ground terms.
  - Problems do not arise if we restrict to atoms in the Herbrand base of P.

S-semantics

Definition. *Non-ground immediate consequence operator* $\mathcal{T}$

For each Herbrand *non-ground* interpretation $I \subseteq \mathcal{B}$:

$$\begin{aligned} A \in \mathcal{T}_P(I) \quad \Leftrightarrow \quad & \exists B, \overline{C}, \overline{D}, \vartheta : \\ & B \leftarrow \overline{C} \in P \wedge \\ & \overline{D} \subseteq I \wedge \\ & \vartheta = mgu(\overline{C}, \overline{D}) \wedge \\ & A = B\vartheta. \end{aligned}$$

Theorem. *For every definite logic program P and for every object level atomic formula A:*

$$A \in \mathcal{T}_P^{\omega} \Leftrightarrow demo(P, A) \in \mathcal{T}_V^{\omega}$$

where V is the meta-program containing both the meta-representation of P and the $vanilla$ meta-interpreter.

Corollary. *For every definite logic program P and for every predicate symbol q:*

$$V \vdash_{\vartheta} demo(P, q(\overline{t})) \Leftrightarrow q(\overline{t}) \in \mathcal{T}_P^{\omega}$$

where $V \vdash_{\vartheta} G$ denotes the fact that there exists a SLD refutation for $V \cup \{\leftarrow G\}$ with a computed answer substitution ϑ.

A meta-interpreter of positive PROLOG with occur check

Extend the vanilla meta-interpreter for definite logic programs which performs the occur check during the unification.

Use the following meta-predicates:

- $var(X)$ which succeeds when the term X is a variable.
- $nonvar(X)$ which succeeds when the term X is not a variable.
- $constant(X)$ which succeeds when the term X is a constant.

It can be useful to define a predicate $compound(X)$ which succeeds if X is complex term (a function).

The meta interpreter

```
demo(empty).
demo((G1,G2)) :- demo(G1), demo(G2).
demo(G) :- return_body(G), demo(B).

return_body(G,B) :- functor(A,F,N), functor(C,F,N),
                    clause(C,B), unify_occ(A,C).
```


Unification with occur-check

```
unify_occ(X,Y) :- var(X),var(Y), X=Y.
unify_occ(X,Y) :- var(X), nonvar(Y),
                  not_occurs_in(X,Y), X=Y.
unify_occ(X,Y) :- var(Y), nonvar(X),
                  not_occurs_in(Y,X), X=Y.
unify_occ(X,Y) :- nonvar(X), nonvar(Y),
                  constant(X),constant(Y), X=Y.
unify_occ(X,Y) :- nonvar(X), nonvar(Y),
                  compound(X),compound(Y),
                  term_unify(X,Y).

not_occurs_in(X,Y) :- var(Y), X\==Y.
not_occurs_in(X,Y) :- nonvar(Y),constant(Y).
not_occurs_in(X,Y) :- nonvar(Y), compound(Y),
                      functor(Y,F,N),
                      not_occurs_in(N,X,Y).
not_occurs_in(N,X,Y) :- N>0, arg(N,Y,Arg),
                        not_occurs_in(X,Arg),
                        N1 is N - 1,
                        not_occurs_in(N1,X,Y).
not_occurs_in(0,X,Y).

term_unify(X,Y) :- functor(X,F,N), functor(Y,F,N),
                   unify_args(N,X,Y).

unify_args(N,X,Y) :- N>0, unify_arg(N,X,Y),
                     N1 is N - 1,
                     unify_args(N1,X,Y).
unify_args(0,X,Y).
unify_arg(N,X,Y) :- arg(N,X,ArgX), arg(N,Y,ArgY),
                    unify_occ(ArgX,ArgY).
```