Introspection Coefficient in Prolog?

Alain Colmerauer

June 2014

What is the introspection coefficient in Prolog?

I would like to write a Prolog programm U such that the query

$$\texttt{:- solve(}t\texttt{,}\alpha P\texttt{)} \tag{1}$$

generates the same substitutions as

$$\texttt{:- } t \tag{2}$$

with program P. Here α is a function with codes the program P by a ground term. Let's call the pair (U, α) a *universal pair.*

Note that the query

$$\texttt{:- solve(solve(t,}\alpha P\texttt{),}\alpha U\texttt{)}$$

with programs U, generates the same substitutions as the query (1) with program U.
More generally

$$\texttt{:- }\overbrace{\texttt{solve(...solve(}}^{n-1}\texttt{solve(}t\texttt{,}\alpha P\texttt{)}\overbrace{\texttt{,}\alpha U\texttt{)...,}\alpha U\texttt{)}}^{n-1} \tag{3}$$

generates the same substitutions as (2).

Let d_n denotes the time necessary for executing (3). Then the *introspection coefficent* for the universal pair (U, α) is

$$C = \lim_{n \to \infty} \frac{d_n}{d_{n-1}} \tag{4}$$

We will exprimentally check that for a given P and a given t, the limit C does exist. If this is true then all the different C schould be the same: as n increases, the number of instructions coming from P becomes negligible to the number of instructions coming from U. That could be the explanation.

I describe now the universal pair. It uses direct access from litteral to the relevant clauses. I will use the follwing built-in predicate:

- $\texttt{once(}t_1\texttt{)}$: the calling of t_1 succeeds atmost once,
- $\texttt{univ(}t_1\texttt{,}t_2\texttt{)}$, in fact $t_1\texttt{=..}\,t_2$: if t_1 is $f\texttt{(}s_1\texttt{,}\ldots\texttt{,}s_n\texttt{)}$ then t_2 is $\texttt{[}f\texttt{,}s_1\texttt{,}\ldots\texttt{,}s_n\texttt{]}$ and vice-versa.

These built-in predicates are taken from the ISO international standard on Prolog. They are described in the book *Prolog: The Standart Reference Manual*, by P. Deransart, A. Ed-Dbali and L. Cervoni, Springer 1996. In this book Prolog is seen as a programming langage which manipulates terms: ground or not ground.

It is easy to program the negation by failure, $\texttt{notprovable(}t\texttt{)}$, and to write many other built-in predicates. See Appendix 2.

Universal pair (U, α)

The coding function α produces the list

$$\texttt{[gives(}s_1\texttt{,}q_1\texttt{),...,gives(}s_m\texttt{,}q_m\texttt{)].}$$

The s_i's are all the different signatures of the head of the clauses. The signature of a term of the form $f\texttt{(}t_1\texttt{,...,}t_n\texttt{)}$ is the term $f\texttt{([],...,[])}$ with n occurencies of `[]`. The q_i's are the non-empty sub-lists of all the coded clauses with signature s_i. Each coded clause is

- the term $t_0\texttt{:- [q(}t_1\texttt{,}p_1\texttt{),...,q(}t_n\texttt{,}p_n\texttt{)]}$ for the clause $t_0\texttt{:- }t_1\texttt{,...,}t_n$,
  the term $t_0\texttt{:- []}$ for the clause t_0,
  where q_i is the possibly empty sub-list of clauses which the same signature than t_i;
- each occurrence of the functional symbol `v` of arity 1 is duplicated;
- each variable is replaced by the ground term `v(`*variablename*`)`;

Thus many q_i are infinite trees. The Prolog compiler should avoid the occur test. But there is no necessity to be able to unify two infinite trees. The program U is in Appendix 1:

Because with use `once(`t`)` the order of the clauses is relevant but does not change the result significally. We will do three tests:

- `:- fail`, with the empty program,
- `:- even(s(s(o)))`, with the program

```
even(o).
even(s(N)) :- odd(N).
odd(s(N)) :- even(N).
```

- `:- inlist(X,[a,b,c])`, with the program

```
inlist(X,[X|L]).
inlist(X,[Xp|L]) :- inlist(X,L).
```

We obtain

`:- fail`

n	d_i in secondes	$\frac{d_i}{d_{i-1}}$
0	0.000009	–
1	0.000010	1.1
2	0.000295	29.5
3	0.019035	64.5
4	1.848905	97.1
5	191.280060	103.5

`:- even(s(s(o)))`

n	d_i in seconds	$\frac{d_i}{d_{i-1}}$
0	0.000022	–
1	0.000093	4.2
2	0.003961	42.6
3	0.378818	95.6
4	36.879262	97.4

`:- inlist(X,[a,b,c])`

n	d_i in seconds	$\frac{d_i}{d_{i-1}}$
0	0.000031	–
1	0.000138	4.5
2	0.014678	106.3
3	1.165718	79.4
4	120.073054	103.0

The coefficient of instrospection may exist and may be

$$C \approx 100$$

It does not depend on the speed of the computer on which we do the tests but it depends on the Prolog implementation. In our case we use a MacBook 13 inches manufactured end of 2008, running under OS X 10.9.2. The Prolog compiler is SWI-Prolog-5.6.59.

Appendice 1: program *U*

```
/* Solve */

solve(T,E) :- once(appropriate(T,P,E)), kernelsolve(T,P,E).

/* Kernel solve */

kernelsolve(univ(T,Tp),[],E) :- univ(T,Tp).
kernelsolve(once(T),[],E) :- once(appropriate(T,P,E)), once(kernelsolve(T,P,E)).
kernelsolve(T,[C|P],E) :- solvenormal(T,[C|P],E).

appropriate(T,P,E) :- univ(T,[A|L]), erase(L,Lp), univ(Tp,[A|Lp]), search(d(Tp,P),E).

erase([],[]).
erase([T|L],[[]|Lp]) :- erase(L,Lp).

search(D,[D|Ep]).
search(D,[Dp|Ep]) :- search(D,Ep).
search(d(T,[]),[]).

/* Solve normal */

solvenormal(T,[(Tp:-Q)|P],E) :- once(instance(Tp,T,[],S)), solveinstances(Q,S,E).
solvenormal(T,[C|P],E) :- solvenormal(T,P,E).

solveinstances([],S,E).
solveinstances([q(T,P)|Q],S,E) :-
   once(instance(T,Tp,S,Sp)), kernelsolve(Tp,P,E), solveinstances(Q,Sp,E).

/* Instance */

instance(v(v(T)),v(Tp),S,Sp) :- instance(T,Tp,S,Sp).
instance(v(X),T,S,Sp) :- instancevariable(X,T,S,S,Sp).
instance(T,Tp,S,Sp) :- univ(T,[I|Q]), instances(Q,Qp,S,Sp), univ(Tp,[I|Qp]).

instances([],[],S,S).
instances([T|Q],[Tp|Qp],S,Spp) :- once(instance(T,Tp,S,Sp)), instances(Q,Qp,Sp,Spp).

instancevariable(X,T,[],S,[gives(X,T)|S]).
instancevariable(X,T,[gives(X,T)|S],Sp,Sp).
instancevariable(X,T,[gives(Xp,Tp)|S],Sp,Spp) :- instancevariable(X,T,S,Sp,Spp).
```

Appendice 2 : other built-in predicates

```
/* Not provable */

notprovable(X) :- once(notp(Y)).

notp(X) :- once(X), fail.
notp(X).

/* Not unifiable terms */

notunifiable(T,Tp) :- notprovable(eg(T,Tp).

eq(T,T).

/* The term is a variable and the term is not a variable */

var(T) :- notprovable(notunifiable(T,yes))), notprovable(notunifiable(T,no))),

notvar(T) :- notprovable(var(T)).

/* Identical terms */

identical(T,Tp) :- var(T), var(Tp), identicalvariables(T,Tp).
identical(T,Tp) :- nonvar(T), nonvar(Tp), univ(T,[I|U]), univ(T,[I|Up])

identicallist([],[]).
identicallist([T|U],[Tp|Up]) :- identical(T,Tp), identicallist(U,Up).

identicalvariables(T,Tp) :- notprovable(duo(X,Y),duo(yes,no)).
```