W. Drabent. **Hints - how to use the Prolog debugger** Version 0.4
Draft, any comments are welcome
October 10, 2017

SICStus Prolog is considered here. We assume that the program dealt with is a definite clause logic program (i.e. without negation and non-logical built-ins of Prolog).

Obtaining all the answers of a given selected atom

Assume that the debugger shows a call of an atomic formula A

`Call:` A

In order to see all the answers for A, repeat the following.

> Type `s`, to skip the details of the execution of A. We may obtain (1) an infinite loop, (2) failure of A:
>
> `Fail:` A
>
> – in this case we are ready, or (3) an answer for A (an instance B of A):
>
> `Exit:` B
>
> In this case type `jr` (jump to the redo port of A):
>
> `Redo:` B
>
> and repeat the above beginning from typing `s`.

At each step you can type `r` to come back to the call of A. At each Exit step typing $\boxed{\text{enter}}$ moves you to the next call.

Looking for the reason of incorrectness

Assume that for a query A_0 you got an answer A, which is wrong w.r.t. your specification for correctness. Now we may start with A_0, but it is better to use A as the initial query (to shrink the SLD-tree to be searched):

```
| ?- trace, A.
% The debugger will first creep -- showing everything (trace)
    Call: A
```

Type $\boxed{\text{enter}}$ to make one step, obtaining a call `Call:` B_1.

We are starting to look at the procedure calls which happened between the call of A and its incorrect success. For each such call, `Call:` B_i, type `s`. If the result is a correct answer, type $\boxed{\text{enter}}$ to obtain the next call. If the result is failure, `Fail:` B_i, type $\boxed{\text{enter}}$ (one or more times) to arrive at a redo port of one of the previously answered calls. At a redo port, type `s` to (try to) obtain a next answer.

Assume that all the obtained answers have been found correct; this means that we arrived to the (incorrect) answer for A, `Exit:` A. Thus we located the error - the clause used in the computation is incorrect. (The head of the clause was unified with A and the calls which contributed to producing the final answer are instances of the body atoms of the clause.)

If any of the obtained answers is incorrect – say an answer B_i' for B_i – then the error will be found by examining the computation between B_i and B_i'. Again, it is useful to start the computation anew by a query that is the incorrect answer, `| ?- trace,`B_i'.

Looking for the reason of incompleteness

We have to find a call $A = p(\ldots)$ such that

(1) A has missing answers (i.e. some answer required by your specification for completeness is not (an instance of) an actually computed answer), and

(2) in the computation for A all the "top level" calls have no missing answers.

. . .