Debugging prolog

Intro to AI - tutorial by Nicolas Höning
April 12, 2006

What is debugging?

- debugging originally means "finding errors in your program"
- as programs became more abstract, it now also means: "follow what your program is doing"
- in prolog the red line you will want to follow is the backtracking (you'll learn more about that later)

ways of debugging in prolog

we'll talk about 2 of them:

- tracing (prolog-style)
- writing (your style)

diving in

we'll use this family relation program (here are just a few lines):

```
mann(johannes).
mann(klaus).
mann(manuel).
...
...
frau(elisabeth).
frau(christa).
frau(margret).
...
```

```
elter(johannes,christa).
elter(johannes,margret).
...
elter(elisabeth,christa).
elter(elisabeth,margret).
...
elter(christa, manuel).
...
grosselter(G,E) :-
    elter(G,X),
    elter(X,E).
```

The example query

our query will be:

```
grosselter(X,manuel).
```

the result is always:

```
?- grosselter(X,manuel).

X = johannes ;

X = elisabeth ;

No
```

but what is prolog doing?

What prolog does (in prose)

prolog is trying to fill the variables such that some grosselter - relation is provable.
That means:

1. some X might be „elter“ of some Y (that is: finding a model for X and Y)
2. and if that Y actually is „elter“ of manuel, then X is a winner (here Prolog decides if that model is valid)

tracing

type trace(grosselter/2,+all). and run the query again.
We can see that prolog is doing something.
but there should be more:
type trace(elter/2,+all). and run the query again.
now that is a history. We are now tracing what happens to two predicates while prolog tries to prove our query.

Note:

- Four "ports" of a predicate can be traced: call,exit,fail,redo (you can turn each of them on or off, e.g. trace(elter/2,-call) (here „/2“ describes the arity of that predicate)
- If you follow Prolog here, you'll see „live“ why the order of your clauses makes a difference!
- Redo goes up to the last step that did not fail and tries to go on from there on another path (yes, that is backtracking), but:
- elter(christa,manuel) is visited often, but only evaluated UNLESS it is part of an unvisited branch (so prolog keeps track of this –of course-).
- to turn tracing off, type nodebug. important: your trace points will stay (type debugging. when in debug/tracing mode to see them)
- type help. to read about predefined predicates like trace yourself

writing

sometimes you might be interested in other things like the value of variables at a specific point. you can write your own output then.
Just add another goal to the „grosselter“ - predicate:

```
grosselter(G,E) :-
    elter(G,X),
    writeln('trying '+G+' for grandparent and '+X+' for parent'),
    elter(X,E).
```

(there is no harm done to the truthfulness of your program: „writeln“ always returns true)

takeaways for writing your own output:

- use single quotation marks
- use the "+" operator to incorporate variables
- this is always ressource-consuming, so keep that in mind for later, bigger programs!
- hint: use an extra predicate (e.g. my_writeln()) where you can switch all your debugging on or off at one line like this:

```
my_writeln(A)
    %:- writeln(A)
    .
```