

Re: Prolog, memory management and memory leaks

Source: <http://coding.derkeiler.com/Archive/Prolog/comp.lang.prolog/2008-01/msg00047.html>

- *From:* bart demoen <bmd@xxxxxxxxxxxxxxxx>
 - *Date:* Mon, 28 Jan 2008 23:41:00 +0100
-

On Sat, 26 Jan 2008 09:19:20 -0600, A.L wrote:

How is in Prolog?... Are memory leaks possible?

It depends on what you mean by a memory leak.
Of course the programmer can "mess up" – here is an example that causes a memory leak in some Prolog systems:

```
a(N,_) :- N > 0, M is N - 1, a(M,foo(1,2,3,4,5)).  
a(N,_) :- N =< 0.
```

Try the query ?- a(<I>,[]).
with <I> equal to 10000000 in SICStus, Yap, SWI and you will see.

Did the programmer mess up, or the implementation ?
In B-Prolog (and hProlog) this query runs in a flash.

Let's remember what went "wrong" as (1).

- a) some facts are asserted. These facts contain data of some constraint problem
- b) predicate is executed that solves constraint problem. This predicate asserts list with results,
- c) more predicates are executed that retrieve results from this list
- d) When everything is done, cleanup predicate is called than retracts all facts and result list.

<

In some circumstances, this server leaks memory.

Maybe I am assuming too much here, but is this process repeated, and after some repetitions, you get into memory problems ?

Re: Prolog, memory management and memory leaks

If that is the case, it means that the cleanup is not good enough (for the Prolog implementation you are working with). Again there is the question: did you mess up, or are you dealing with a bad implementation.

I did the homework

We know you – you are a serious guy.

I checked about a dozen papers on Prolog GC and memory management. In one, "Understanding Memory Management in Prolog Systems" by Castro and Costa, there is Table 3 that shows "garbage collector efficiency". According to this table, GC efficiency for non-deterministic predicates is not always perfect.

Most often, the problem (as far as I know about GC in Prolog) is not that GC efficiency for non-deterministic predicates is not always perfect, but that Prolog systems consider some predicates as non-deterministic, while the programmer thought they were deterministic. The $a/2$ predicate above shows that.

One thing to realise is: GC can't be perfect (as in "reading the programmers mind about what data is useful in the future"). Being perfect about garbage is equivalent to the Halting Problem. (let me mark this as (a)). So a programmer must help the system, however good it is.

1. Is memory leak in the scheme mentioned above possible? If yes, when and why?
2. What is the strategy to manage memory in long-running Prolog applications?...

I am asking these questions as "generic" ones, not regarding any specific Prolog implementation.

First of all, the generic answer to both of them is by nature very unsatisfactory: (1) yes, because if (a); (2) there is none, because Prolog implementations (can) have their own idea about reachability, usefulness logic, determinism ... In general, you should even be prepared for the odd Prolog implementation that even in the presence of a failure driven loop, refuses to collect any garbage (as you would define it) because it keeps all data about any execution it ever performed so that it can give you instant feedback on those runs.

But you are in luck today, there are some things you can do as a programmer, at least when you use a reasonable system like Yap,

Re: Prolog, memory management and memory leaks

SICStus, SWI ...

Try to find out whether your (local/trail/choicepoint) stacks grow beyond what you expected. If for the $a/2$ predicate above you expected constant size, and you observe something else

- put cuts in your "non-deterministic" predicates
- make predicates tail-recursive

If you observe that the heap (global stack) grows more than you expected, and your other stacks are constant size, then there are two possibilities:

- your program really needs more space than is available
- you are responsible for keeping data in you arguments that is actually garbage

Compare the latter to an implementation of a stack in a garbage collected imperative language (e.g. Java): when you pop, you better also nil the top, otherwise the popped elements potentially remain live long after you expected them to die.

It would be nice to have tools that tell you about predicates that leave choicepoints. In hProlog, I have

```
?- det(somegoal).
```

It is very cheap, but when choicepoints are left, it prints out a backtrace of the choicepoints. For instance:

```
?- det(queens(4,L)).  
/* choice points are left by */  
select/3.  
queens/3.  
select/3.  
queens/3.  
select/3.  
queens/3.  
select/3.  
queens/3.  
range/3.  
range/3.  
range/3.
```

The same could be done for detecting non-tail recursive predicates.

in the presence of dynamic code, your problems only get worse: does the system detect properly that the last clause was selected (this might depend on indexing of dynamic code) ? When does the system

Re: Prolog, memory management and memory leaks

garbage collect dynamic code (this affects detection of the last clause) ? How good is this dyncode gc ? ...

There is no generic answer to those questions. My advice would be: don't use dynamic code – Jan SWI would probably agree to a large extent with that.

And to agree even more with Jan, I would say: use failure driven loops; do your cleanup after backtracking.

And also use tail–recursion and deterministic predicates whenever possible.

Don't use dynamic pred – use findall when you can. Paul Tarau coined the term "poor man's garbage collection" for findall – as usual, he was spot on :-)

Cheers

Bart Demoen

.