TDDD08 — Tutorial 6

Who? Victor Lagerkvist

From? Theoretical Computer Science Laboratory, Linköpings Universitet, Sweden

When? 9 oktober 2015

Efficient Prolog

There are many techniques that can be used to make a Prolog program more efficient. Some of them are given in this tutorial.

- Basic guidelines.
- Indexing.
- Cuts, if-then-else.
- Tail recursion.
- Open structures and difference lists.

Basic guidelines

Complexity.

- First and most important issue: which algorithm is being implemented? Worst-case running time ($O(n)$ notation) still applies for Prolog but with a few additional complications.
- Sorting and reverse examples on whiteboard.

Basic guidelines

Order of queries. The ordering of the clauses for a predicate and the subqueries within a clause matters. Consider these two versions of a query:

- `| ?- student(X), father(X).`
- `| ?- father(X), student(X).`

Which is better? If we are working with a database of (1) the people in Linköping or (2) the people at LiU? In general: put the most restrictive query first! Also, put facts and base cases before recursive cases in a program.

Use the libraries

Be lazy whenever possible!

- Builtins and library methods are usually faster than hand-written code.
- `use_module(module_name)` at the prompt.
- `:- use_module(module_name)` in the code.

Lots of functions for free: `append/3`, `member/2`, `map/2`, `reverse/2` etc. See the SICStus Prolog documentation.

Unification and indexing

Unification is very fast in most Prolog systems. Failing in the unification step is faster than using a rule and then failing. Consider:

```
len3_a([_X,_Y,_Z|_Xs]).

len3_b(Xs) :- length(Xs, L), L >= 3.
```

Imagine what happens when we call `len3_a/1` or `len3_b/1` with a list of thousands of elements.

Unification and indexing

Prolog identifies rules to try based on:

- Predicate name and arity (e.g. append/3).
- Type of first argument (e.g. empty list, non-empty list, `f(X)`, symbol `anna`, number 0...)

Place identifiers, etc, in first argument! Silly example: with the program

```
p(1, 1).
p(2, 2).
p(3, 3).
...
p(10000, 10000).
```

Queries of the form `p(999, X)` are substantially faster than queries of the form `p(X, 999)`. Indexing on multiple arguments are available in some Prolog system but is not yet something that you can take for granted.

Pruning branches of the search tree: the cut!

The cut (`!/0`) is used to remove remove superflous branches in the search tree. Examples on whiteboard.

Tail recursion

Well-known trick in functional programming. Replace recursion with iteration. Possible in Prolog when:

- The recursive call is the last one in its clause.
- There are no untried alternatives for the clause (e.g. the clause is the last rule for the predicate).
- There are no untried alternatives for the other subqueries of the clause. In this case, Prolog does not have to perform a full recursion (create a stack frame, keep track of traceback points) but can "skip" back into the clause.

Consider `middle/2` from lab 2.

```
middle(X, [X]).
middle(X, [_First|Xs]) :-
    append(Middle, [_Last], Xs),
    middle(X, Middle).
```

Is `middle/2` tail-recursive?

Differences lists and open structures

We have already seen difference lists in relation to definite clause grammars. More examples of open structures on the whiteboard.