Executable specifications with Prolog

Rigorous Software Development
EUROPEAN MASTER IN SOFTWARE ENGINEERING/ MASTER IN SOFTWARE AND SYSTEMS
Universidad Politécnica de Madrid/IMDEA Software

December 10, 2019

To be turned in by January 10, 2020. Send them to me by mail to `jmarino@fi.upm.es`.

If you are fluent in Prolog you can choose any exercise from this sheet to replace the corresponding exercise from the Maude sheet.

Exercise 1. Use the genealogy tree in Figure 1 to populate a knowledge database with predicates mother/2 and father/2. Define predicates grandmother/2, ancestor/2, sibling/2, descendant/2, cousin/2, etc.

Exercise 2. Define a predicate add/3 that relates 3 terms representing Peano naturals such that the third one is the sum of the other two. Use this to define subtraction and a predicate even/1 to test whether a natural number is even.

Exercise 3. A typical application of logic programming is planning and, in general, exploration of a *state space*. This can range from helping an autonomous robot find a plan to achieve some complex task to analyzing a concurrent system in order to find some hazardous behaviour. Here, we will just consider a well-known puzzle as a toy example for this category. Three priests and three children want to cross a river on a boat that allows for a maximum capacity of two people. At any time, if there are children on any of the banks, they should not be outnumbered by priests. Model the problem in Prolog in order to find a plan – a sequence of crossings – that allows them to perform the crossing safely.

Exercise 4. One "killer" app of Prolog is parsing – in fact it was developed for natural language processing. In the case of *context free grammars* (CFGs) we typically define a ternary Prolog predicate for each nonterminal symbol of the grammar, such that the first parameter is the input list of tokens, the second one is the syntax tree resulting from parsing the nonterminal and the third one is the suffix of the list that remains after parsing the nonterminal.

For example, the first rule of the grammar in Figure 2 would be expressed as:

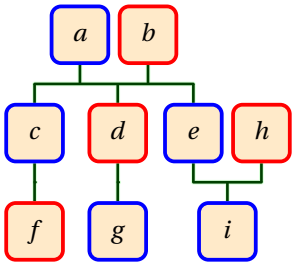

Figure 1: A genealogy. Blue nodes are male, red nodes are female.

$$
\begin{array}{lcl}
E & ::= & E + T \mid E - T \mid T \\
T & ::= & T * F \mid T / F \mid F \\
F & ::= & id \mid n \mid (E)
\end{array}
$$

Figure 2: A simple grammar for arithmetic expressions.

```
e(Tokens, plus(A,B), Rest) :-
    e(Tokens, A, ['+'|Xs]),
    t(Xs, B, Rest).
```

Complete the parser and test it.

Exercise 5. A *binary search tree* can be represented as a Prolog term by using internal choice nodes of the form $nd(left, number, right)$ and *tip* leaves. Define a predicate inorder/2 that relates a search tree with the ordered list of its internal nodes. We will say that a binary tree is balanced iff the number of nodes in its left and right subtrees differ in at most one. Define a predicate balanced/1 that checks whether a binary search tree is balanced. Use the previous definitions to define a predicate that balances a search tree, i.e. returns a tree that is balanced and has the same ordered list of internal nodes.

Exercise 6. Define a predicate sudoku/1 that takes a list of lists of numbers $(1 \ldots 9)$ representing a completed Sudoku puzzle – as list of rows – and checks whether it is a legal solution. Use it to give a quick prototypical solver. Provide an experiment and analyze the efficiency.