Model Checking, SS2011: Exercise Sheet 9

June 6, 2011

Note. You may want to consult some online references on SICStus Prolog.

1. The SICStus Prolog libraries:
   `http://www.sics.se/sicstus/docs/4.0.4/html/sicstus/The-Prolog-Library.html`
2. Built-In Prolog predicates:
   `http://www.cs.bham.ac.uk/~pjh/prolog_module/sicstus_manual_v3_5/sicstus_10.html#SEC95`
3. CLPQ library:
   `http://www.sics.se/sicstus/docs/4.0.4/html/sicstus/lib_002dclpqr.html#lib_002dclpqr`

Exercise 9.1. Create the following Prolog procedures.

1. Procedure `reverse`/2 that reverses a list.
2. Procedure `last`/2 such that `last(L, E)` succeeds if `E` is the last element of the list `L`.
3. A procedure that removes the k-th element of a list. Example:

   ```
   ?- remove_at(X,[a,b,c,d],2,R).
   X = b
   R = [a,c,d]
   ```

4. Procedure `clpq_inject`/1 that takes a list `L` of CLPQ constraints and injects the conjunction of the elements of `L` in the constraint store.
5. Procedure `gcd`/3 such that `gcd(N1, N2, G)` succeeds if `G` is the greatest common divisor of `N1` and `N2`.
6. Procedure `coprime`/2 that succeeds if its two parameters are coprime integers.
7. Procedure `imp`/2 such that `imp(A,B)` succeeds if `A` and `B` are CLPQ representations of integer linear arithmetic formulas, and `A` entails `B`.

8. Procedure `fib/2` such that a call `fib(N, F)` succeeds if `F` is the `N`-th Fibonacci number.

9. Procedure `sum/2` such that `sum(L, S)` succeeds if `L` is a list of integers and `S` is the sum of the elements of `L`.

10. Procedure `consistent/2` such that `consistent(L, F)` succeeds if `L` is a list of CLPQ formulas, `F` is a CLPQ formula, and the following conjunction is satisfiable.

$$\left(\bigwedge_{p \in \mathtt{L}} p\right) \wedge \mathtt{F}$$

11. Consider the predicate `t(Data, LeftTree, RightTree)` that represents the binary tree whose root is tagged with `Data` and whose left and right subtrees are `LeftTree` and `RightTree`.Give a procedure `binary_search/2` such that `binary_search(T, N)` succeeds if `T` is an ordered integer binary tree, and `N` is the tag of some node.

Exercise 9.2. Assume you are given a program $P = (X,\ pc,\ T,\ \varphi_{init},\ \varphi_{error})$. Consider the following definitions.

$$post(\phi) := \bigvee_{\rho \in T} post(\rho, \phi)$$

$$F(\phi) := \varphi_{init} \vee post(\phi)$$

Prove that F is *monotonic* w.r.t $\models$, i.e. prove that if $\phi_1 \models \phi_2$, then $F(\phi_1) \models F(\phi_2)$.

Exercise 9.3. Consider a directed graph $G = (s, Edges, Nodes)$ with start node s. Consider the following algorithm.

$$ReachMore(R) = R \cup \{n \in Nodes \mid \exists\, n_r \in R\ .\ (n_r, n) \in Edges\}$$

1. Prove that $ReachMore$ is monotonic w.r.t. to $\subseteq$.

2. Encode $ReachMore$ as a Prolog procedure `reach_more/2`.

3. Program in Prolog a concrete reachability algorithm that computes the set of reachable states by computing a fixed point of $ReachMore$.

4. Test your concrete reachability program on the following graph.

```
start(1).
edge(1,2).
edge(2,2).
edge(2,3).
edge(3,4).
edge(3,5).
```

Exercise 9.4. Look up and understand the definition of partial order.

Exercise 9.5. Look up and understand the Knaster-Tarski Theorem.

Exercise 9.6. Do the following modifications to the abstract reachability model checker presented in Lecture 10.

1. Upon termination, print a representation of the abstract reachability tree.
2. Upon reaching an error state, print the potential counter example path.
3. Modify `abst_reach_step` so that new abstract states are printed when discovered.

Exercise 9.7. Consider the Forward-Symbolic-Reachability Prolog program available online[1]. Improve the program by printing upon termination a counterexample path if there is any.

[1] http://www7.in.tum.de/um/courses/mc/ss2011/fsr.pl