Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Andrei Popescu, Peter Lammich

WS 2012/13
6. 11. 2012

Semantics of Programming Languages

Exercise Sheet 4

Exercise 4.1 Reflexive Transitive Closure

Theory *Star* (available on the course website) defines a binary relation *star r*, which is the reflexive, transitive closure of the binary relation *r*. It is defined inductively with the rules "*star r x x*" and "$[\![r\ x\ y;\ star\ r\ y\ z]\!] \Longrightarrow star\ r\ x\ z$".

We also could have defined *star* the other way round, i.e., by appending steps rather than prepending steps:

inductive $star'$ :: "$('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$" **for** *r* **where**
  "$star'\ r\ x\ x$" |
  "$[\![star'\ r\ x\ y;\ r\ y\ z]\!] \Longrightarrow star'\ r\ x\ z$"

Prove the following lemma. Hint: You will need an additional lemma for the induction.

lemma "$star\ r\ x\ y \Longrightarrow star'\ r\ x\ y$"

Exercise 4.2 Proving That Numbers Are Not Even

Recall the evenness predicate *ev* from the lecture:

inductive *ev* :: "$nat \Rightarrow bool$" **where**
  ev0: "*ev 0*" |
  evSS: "$ev\ n \Longrightarrow ev\ (Suc\ (Suc\ n))$"

Prove the converse of rule *evSS* using rule inversion. Hint: There are two ways to proceed. First, you can write a structured Isar-style proof using the *cases* method:

lemma "$ev\ (Suc\ (Suc\ n)) \Longrightarrow ev\ n$"
proof –
  assume "$ev\ (Suc\ (Suc\ n))$" **then show** "*ev n*"
  proof (*cases*)

    ...

  qed
qed

Alternatively, you can write a more automated proof by using the **inductive_cases** command to generate elimination rules. These rules can then be used with "*auto elim*:". (If given the [*elim*] attribute, *auto* will use them by default.)

inductive_cases *evSS_elim*: *"ev (Suc (Suc n))"*

Next, prove that the natural number three (*Suc* (*Suc* (*Suc 0*))) is not even. Hint: You may proceed either with a structured proof, or with an automatic one. An automatic proof may require additional elimination rules from **inductive_cases**.

lemma *"¬ ev (Suc (Suc (Suc 0)))"*

Exercise 4.3 Binary Trees with the Same Shape

Consider this datatype of binary trees:

datatype *tree = Leaf int | Node tree tree*

Define an inductive binary predicate *sameshape* :: *tree* ⇒ *tree* ⇒ *bool*, where *sameshape* t_1 t_2 means that t_1 and t_2 have exactly the same overall size and shape. (The elements in the corresponding leaves may be different.)

inductive *sameshape* :: *"tree ⇒ tree ⇒ bool"* **where**

Now prove that the *sameshape* relation is transitive.

theorem "$[\![$*sameshape* t_1 t_2; *sameshape* t_2 $t_3]\!] \Longrightarrow$ *sameshape* t_1 t_3"

Hint: For this proof, we recommend doing an induction over t_1 and t_2 using rule *sameshape.induct*. You will also need some elimination rules from **inductive_cases**. (Look at the subgoals after induction to see which patterns to use.) Finally, note that "*auto elim*:" applies rules tentatively with a limited search depth, and may not find a proof even if you have all the rules you need. You can either try the variant "*auto elim*!:", which applies rules more eagerly, or try another method like *blast* or *force*.

Homework 4 Finite State Machines

Submission until Tuesday, November 13, 10:00am.

Finite state machines (for simplicity without initial states) can be given by a set of final states $F::'Q\ set$ and a transition relation of type $\delta::('Q \times '\Sigma \times 'Q)\ set$. Note that $(q,a,q') \in \delta$ means that there is a transition from q to q' labeled with a.

type_synonym $('Q,'\Sigma)\ LTS$ = "$('Q \times '\Sigma \times 'Q)\ set$"

First define an inductive predicate *accept*, that characterizes the words accepted from a given state q, i.e., *accept F δ q w* holds iff word w is accepted from state q.

inductive *accept* :: "$'Q\ set \Rightarrow ('Q,'\Sigma)\ LTS \Rightarrow 'Q \Rightarrow '\Sigma\ list \Rightarrow bool$"
  for F δ **where**

The product construction is a standard construction for the intersection of two FSMs. Define a function *prod_δ* that returns the transition relation of the product FSM of two given FSMs:

definition *prod_δ* :: "$('Q1,'\Sigma)\ LTS \Rightarrow ('Q2,'\Sigma)\ LTS \Rightarrow ('Q1 \times 'Q2,'\Sigma)\ LTS$"

Now prove that your product accepts enough words. Hint: You will need rule induction and rule inversion.

lemma *prod_complete*:
  assumes *A*: "*accept F1 δ1 q1 w*"
  assumes *B*: "*accept F2 δ2 q2 w*"
  shows "*accept (F1×F2) (prod_δ δ1 δ2) (q1,q2) w*"
  using *A B*
proof (*induction arbitrary*: *q2 rule*: *accept.induct*[*case_names base step*])
  case (*base q1*)

next
  case (*step q1 a q1' w q2*)**qed**

Now prove that your product does not accept too many words.

lemma *prod_sound*:
  assumes "*accept (F1×F2) (prod_δ δ1 δ2) (q1,q2) w*"
  shows "*accept F1 δ1 q1 w* ∧ *accept F2 δ2 q2 w*"

Hint to get the induction through:

proof –
  {
    fix *q12*
    assume "*accept (F1×F2) (prod_δ δ1 δ2) q12 w*"
    hence "*accept F1 δ1 (fst q12) w* ∧ *accept F2 δ2 (snd q12) w*"

Insert your inductive proof here

  }

thus *?thesis* **using** *assms* **by** *auto*
qed

end