Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Simon Wimmer

WS 2016/17
22. 11. 2016

Semantics of Programming Languages

Exercise Sheet 5

Exercise 5.1 Rule Inversion

Recall the evenness predicate *ev* from the lecture:

inductive *ev* :: *"nat ⇒ bool"* **where**
  ev0: *"ev 0"* |
  evSS: *"ev n ⟹ 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)) ⟹ ev n"*
proof –
  assume *"ev (Suc (Suc n))"* **then show** *"ev n"*
  proof (*cases*)

    ...

  qed
qed

Optional: 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 5.2 (Deterministic) labeled transition systems

Give all your proofs in Isar, not apply style

A *labeled transition system* is a directed graph with edge labels. We represent it by a predicate that holds for the edges.

type_synonym $('q,'l)$ *lts* = "$'q \Rightarrow 'l \Rightarrow 'q \Rightarrow$ *bool*"

I.e., for an LTS δ over nodes of type $'q$ and labels of type $'l$, $\delta\ q\ l\ q'$ means that there is an edge from q to q' labeled with l.

A word from source node u to target node v is the sequence of edge labels one encounters when going from u to v.
Define a predicate *word*, such that *word* $\delta\ u\ w\ v$ holds iff w is a word from u to v.

inductive *word* :: "$('q,'l)$ *lts* $\Rightarrow 'q \Rightarrow 'l$ *list* $\Rightarrow 'q \Rightarrow$ *bool*" **for** δ

A *deterministic* LTS has at most one transition for each node and label

definition "*det* $\delta \equiv \forall\, q\ a\ q1\ q2.\ \delta\ q\ a\ q1 \wedge \delta\ q\ a\ q2 \longrightarrow q1 = q2$"

Show: For a deterministic LTS, the same word from the same source node leads to at most one target node, i.e., the target node is determined by the source node and the path

lemma
  assumes *det*: "*det* δ"
  shows "*word* $\delta\ q\ w\ q' \Longrightarrow$ *word* $\delta\ q\ w\ q'' \Longrightarrow q' = q''$"

Exercise 5.3 Counting Elements

Give all your proofs in Isar, not apply style

Recall the count function, that counts how often a specified element occurs in a list:

fun *count* :: "$'a \Rightarrow 'a$ *list* $\Rightarrow$ *nat*" **where**
  "*count* x [] = 0"
| "*count* x $(y\#ys)$ = (*if* $x{=}y$ *then Suc* (*count* x ys) *else count* x ys)"

Show that, if an element occurs in the list (its count is positive), the list can be split into a prefix not containing the element, the element itself, and a suffix containing the element one times less

lemma "*count* x xs = *Suc* $n \Longrightarrow \exists\, p\ s.\ xs = p\ @\ x\ \#\ s \wedge$ *count* $x\ p = 0 \wedge$ *count* $x\ s = n$"

Homework 5.1 Paths in Graphs

Submission until Tuesday, November 29, 10:00am.

Give all your proofs in Isar, not apply style

A graph is specified by a set of edges: E :: $('v \times 'v)$ *set*. A path in a graph from u to v is a list of vertices $[u_1, \ldots, u_n]$ such that $u = u_1$, $(u_i,\ u_{i+1}) \in E$, and $(u_n,\ v) \in E$. Moreover, the empty list is a path from any node to itself.
For example, in the graph: $\{(i, i+1) \mid i \in \mathbb{N}\}$, we have that $[3,4,5]$ is a path from 3 to 6, and [] is a path from 1 to 1.

Note that not including the last node of the path into the list simplifies the formalization. Formalize an inductive predicate *is_path*

inductive *is_path* :: "('v×'v) set ⇒ 'v ⇒ 'v list ⇒ 'v ⇒ bool"

Test your formalization for some examples:

lemma "is_path {(i,i+1) | i::nat. True} 3 [3,4,5] 6"
lemma "is_path {(i,i+1) | i::nat. True} 1 [] 1"

Prove the following two lemmas that allow you to glue together and split paths:

lemma *path_appendI*:
"⟦is_path E u p1 v; is_path E v p2 w⟧ ⟹is_path E u (p1@p2) w"

Hint: For the next lemma, do an induction over *p1*, and, in the induction step, use rule-inversion on *is_path*.

lemma *path_appendE*:
"is_path E u (p1@p2) w ⟹ ∃ v. is_path E u p1 v ∧ is_path E v p2 w"

Homework 5.2 Grammars

Submission until Tuesday, November 29, 10:00am.

Give all your proofs in Isar, not apply style

We define a grammar for strings of the form $a^n b^n$, where a and b are defined via the type *ab*:

datatype *ab* = *a* | *b*

We define the language of all strings of the form $a^n b^n$ by means of the following rules:

$$S \rightarrow aSb \mid \epsilon$$

inductive *S* :: "ab list ⇒ bool" **where**
add: "S w ⟹ S (a # w @ [b])"
| *nil*: "S []"

Your task is to show that the grammar fulfills the informal specification of the language, i.e.

lemma *S_correct*:
"S w ⟷ (∃ n. w = replicate n a @ replicate n b)"

Here, *replicate* is a pre-defined function, with *replicate n x* producing a list consisting of n copies of x.

Hint: you may want to split the proof into proofs for the two directions of ⟷. Your proofs may require additional lemmas on *replicate*.