Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Kevin Kappelmann, Lukas Stevens

WS 24/25
7.11.2024

Semantics of Programming Languages

Exercise Sheet 4

From this sheet onward, you should write all your (non-trivial) proofs in Isar!

Exercise 4.1 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

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 4.2 (Deterministic) Labelled Transition Systems (LTS)

A *labelled transition system* is a directed graph with labelled edges. We model such systems as functions:

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

For an LTS δ over nodes of type $'q$ and labels of type $'l$, $\delta\ p\ l\ q$ means that there is an edge from p to q labelled with l.

A word from source node u to target node v is the list of edge labels one encounters when going from u to v.
Define an inductive 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* $\delta \equiv \forall p\ l\ q1\ q2.\ \delta\ p\ l\ q1 \wedge \delta\ p\ l\ q2 \longrightarrow q1 = q2$"

Show that for a deterministic LTS, the same word from the same source node leads to at most one target node.

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

Exercise 4.3 Counting Elements

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
  assumes "*count* a xs = *Suc* n"
 shows "$\exists$ *ps ss. xs* = *ps* @ a # *ss* $\wedge$ *count* a *ps* = 0 $\wedge$ *count* a *ss* = n"

Homework 4.1 Product Construction for LTS

Submission until Wednesday, November 13, 23:59pm.
The product construction is a standard construction for the intersection of two *lts*. Define the transition relation *prod* of the product of two given transition systems.

inductive *prod* :: "$('q_1,'l)$ *lts* $\Rightarrow$ $('q_2,'l)$ *lts* $\Rightarrow$ $'q_1 \times 'q_2 \Rightarrow$ $'l \Rightarrow$ $'q_1 \times 'q_2 \Rightarrow$ *bool*" **for** $\delta_1\ \delta_2$

Show that your product only contains those words.

Hint: Make sure to set up the induction properly. When you explicitly state the arguments in a computation or rule induction, it might be necessary to then declare some of the variables in the arguments as arbitrary.

Also make sure to chain in the proper facts into your induction with *using* ....

theorem *prod_sound*:
  assumes *"word (prod* δ_1 δ_2*) (*p_1*,*p_2*) ls (*q_1*,*q_2*)"*
 shows *"word* δ_1 p_1 *ls* q_1 $\wedge$ *word* δ_2 p_2 *ls* q_2*"*

Now prove that your product accepts all words that occur in both *lts*.

Hint: You will need rule induction and rule inversion.

lemma *prod_complete*:
  assumes *"word* δ_1 p_1 *ls* q_1*"*
    and *"word* δ_2 p_2 *ls* q_2*"*
 shows *"word (prod* δ_1 δ_2*) (*p_1*,*p_2*) ls (*q_1*,*q_2*)"*

Finally, the single correctness statement follows:

corollary *"*$\{w.$ *word (prod* δ_1 δ_2*) (*p_1*,*p_2*)* w *(*q_1*,*q_2*)*$\} = \{w.$ *word* δ_1 p_1 w $q_1\} \cap \{w.$ *word* δ_2 p_2 w $q_2\}$*"*
 using *prod_sound prod_complete* **by** *fast*

Homework 4.2 Minimal Compiler

Submission until Wednesday, November 13, 23:59pm.

Recall the *aexp* compiler with exceptions from the last tutorial.

We defined correctness of the compiled code as follows:

correct a ins $\equiv \forall$ *s stk. exec ins s stk = Some* (*aval a s # stk*)

We are also interested in optimal code. Show that any correct list of instructions needs to contain a *LOAD* instruction for every variable of an *aexp*.

Hint: Do a proof by contradiction, without induction. You will need auxiliary lemmas about *aval* and *exec* with changed state (those need induction).

theorem *vars_in_ins*:
  assumes *"x* $\in$ *vars a"*
 shows *"correct a ins* $\Longrightarrow$ *LOAD x* $\in$ *set ins"*