Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D., Dr. Peter Lammich
Johannes Hölzl

WS 2014/15
4. 11. 2014

Semantics of Programming Languages

Exercise Sheet 4

Exercise 4.1 Reflexive Transitive Closure

A binary relation is expressed by a predicate of type $R :: 's \Rightarrow 's \Rightarrow bool$. Intuitively, $R\ s\ t$ represents a single step from state s to state t.

The reflexive, transitive closure R^* of R is the relation that contains a step $R^*\ s\ t$, iff R can step from s to t in any number of steps (including zero steps).

Formalize the reflexive transitive closure as inductive predicate:

inductive *star* :: "$('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$"

When doing so, you have the choice to append or prepend a step. In any case, the following two lemmas should hold for your definition:

lemma *star_prepend*: "$[\![r\ x\ y;\ star\ r\ y\ z]\!] \Longrightarrow star\ r\ x\ z$"
lemma *star_append*: "$[\![\ star\ r\ x\ y;\ r\ y\ z\ ]\!] \Longrightarrow star\ r\ x\ z$"

Now, formalize the star predicate again, this time the other way round:

inductive *star'* :: "$('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$"

Prove the equivalence of your two formalizations

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

Hint: The induction method expects the assumption about the inductive predicate to be first.

Exercise 4.2 Elements of a List

Give all your proofs in Isar, not apply style

Define a recursive function *elems* returning the set of elements of a list:

fun *elems* :: "$'a\ list \Rightarrow 'a\ set$"

To test your definition, prove:

lemma "$elems\ [1,2,3,(4::nat)] = \{1,2,3,4\}$"

Now prove for each element x in a list xs that we can split xs in a prefix not containing x, x itself and a rest:

lemma *"$x \in elems\ xs \Longrightarrow \exists\, ys\ zs.\ xs = ys\ @\ x\ \#\ zs \wedge x \notin elems\ ys$"*

Exercise 4.3 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)))"
```

Homework 4.1 (Deterministic) labeled transition systems

Submission until Tuesday, November 11, 10:00am.

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 ⇒ 'l ⇒ 'q ⇒ 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* δ u w v holds iff w is a word from u to v.

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

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

definition *"det δ ≡ ∀ q a q1 q2. δ q a q1 ∧ δ q a q2 ⟶ 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 δ q w q' ⟹ word δ q w q'' ⟹ q'=q''"*

Homework 4.2 Grammars

Submission until Tuesday, November 11, 10:00am.

Give all your proofs in Isar, not apply style
We define two symmetric grammars for all well balanced strings of $\{a, b\}$, defined as the type *ab*:

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

Now we define the language of all balanced occurrences of a and b in two different ways and show that both definitions are equal.

$$S \rightarrow aSbS|\epsilon$$

$$T \rightarrow TaTb|\epsilon$$

inductive_set *S* :: *"ab list set"* **where**
left: *"w1 ∈ S ⟹ w2 ∈ S ⟹ [a] @ w1 @ [b] @ w2 ∈ S"*
| *nil*: *"[] ∈ S"*

inductive_set *T* :: *"ab list set"* **where**
right: *"w1 ∈ T ⟹ w2 ∈ T ⟹ w1 @ [a] @ w2 @ [b] ∈ T"*
| *nil*: *"[] ∈ T"*

Prove the equivalence $T = S$.

Hint: You need to show that $S \rightarrow SS$ and $T \rightarrow TT$ are valid rules. The definitions of S and T show you how these are rules stated in Isabelle/HOL.

lemma *S_imp_T*:

assumes w: *"$w \in S$"*
shows *"$w \in T$"*

Prove this!

lemma *T_imp_S*:
assumes w: *"$w \in T$"*
shows *"$w \in S$"*

Prove this!

With these theorems we finally show the equivalence of S and T:

lemma *"$S = T$"*
using *S_imp_T T_imp_S* **by** *auto*