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

Formalize the reflexive transitive closure as inductive predicate:

\begin{verbatim}
inductive star :: "('a => 'a => bool) => 'a => 'a => bool"
\end{verbatim}

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 ] => star r x z"
- lemma star_append: "[ star r x y; r y z ] => star r x z"

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

\begin{verbatim}
inductive star' :: "('a => 'a => bool) => 'a => 'a => bool"
\end{verbatim}

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 $\textit{elems}$ returning the set of elements of a list:

\begin{verbatim}
fun elems :: "'a list => 'a set"
\end{verbatim}

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** \(\forall x \in \text{elems} \; \forall xs. \; \exists ys \; \exists zs. \; xs = ys \; @ \; x \; \# \; zs \; \land \; x \; \notin \; \text{elems} \; ys\)

**Exercise 4.3  Rule Inversion**

Recall the evenness predicate \(ev\) from the lecture:

**inductive** \(ev :: \text{"nat \Rightarrow bool" where}\)

\(ev\_0:: \text{"ev 0"} |\)

\(evSS:: \text{"ev n \Rightarrow ev (Suc (Suc n))"} \)

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** \(\text{"ev (Suc (Suc n)) \Rightarrow ev n"} \)

**proof** —

**assume** \(\text{"ev (Suc (Suc n))" } \text{then show } \text{"ev n"} \)

**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** \(evSS\_elim:: \text{"ev (Suc (Suc n))"} \)

Next, prove that the natural number three \((\text{Suc (Suc (Suc 0)))}\) 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** \(\text{"\neg ev (Suc (Suc (Suc 0)))"} \)

**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** \(\text{"('q,'l) \; lts = "'q \Rightarrow 'l \Rightarrow 'q \Rightarrow bool"} \)

I.e., for an LTS \(\delta\) over nodes of type \(\text{"'q}\) and labels of type \(\text{"'l}\), \(\delta\; q \; l \; q\) 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 \( \text{word} \), such that \( \text{word} \ \delta \ u \ w \ v \) holds iff \( w \) is a word from \( u \) to \( v \).

**inductive** \( \text{word} :: \quad (\lts \to \q \to \q \to \bool) \) for \( \delta \)

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

**definition** \( \text{det} \ \delta \equiv \forall \ q \ a \ q1 \ q2. \ \delta \ q \ a \ q1 \land \delta \ q \ a \ q2 \rightarrow q1=q2 \)

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** \( \text{det} \) : “\( \text{det} \ \delta \)”  
**shows** “\( \text{word} \ \delta \ q \ w \ q' \implies \text{word} \ \delta \ q \ w \ q'' \implies q'=q'' \)”

---

**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** \( ab \) = \( a \mid b \)

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

\[
S \to aSbS|\epsilon \\
T \to TaTb|\epsilon 
\]

**inductive_set** \( S :: \quad \text{“ab list set”} \) where  
**left:** “\( w1 \in S \implies w2 \in S \implies [a] @@ w1 @@ [b] @@ w2 \in S \)”  
**nil:** “\( [] \in S \)”

**inductive_set** \( T :: \quad \text{“ab list set”} \) where  
**right:** “\( w1 \in T \implies w2 \in T \implies w1 @@ [a] @@ w2 @@ [b] \in T \)”  
**nil:** “\( [] \in T \)”

Prove the equivalence \( T = S \).

**Hint:** You need to show that \( S \to SS \) and \( T \to TT \) 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** \( T_{\text{imp}} S \):
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 \( S_{\text{imp}} T T_{\text{imp}} S \) by \textit{auto}