Exercise 4.1  Reflexive Transitive Closure

Theory \textit{Star} (available on the course website) defines a binary relation \textit{star} \( r \), which is the reflexive, transitive closure of the binary relation \( r \). It is defined inductively with the rules \( \text{"star} \ r \ x \ x \)" and \( \left[ \left[ \text{star} \ r \ x \ y; \ r \ y \ z \right] \right] \Rightarrow \text{star} \ r \ x \ z \).

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

\begin{verbatim}
inductive star' :: 
"\('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool" for r where
"star' r x x"
\[star' r x y; r y z\] \Rightarrow \text{star'} r x z
\end{verbatim}

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

\textbf{lemma} \textit{"star} \ r \ x \ y \Rightarrow \text{star'} \ r \ x \ y"

Exercise 4.2  Proving That Numbers Are Not Even

Recall the evenness predicate \textit{ev} from the lecture:

\begin{verbatim}
inductive ev :: 
"nat \Rightarrow bool" where
ev0: \textit{ev} 0
evSS: \textit{ev} \ n \Rightarrow \textit{ev} \ (Suc \ (Suc \ n))
\end{verbatim}

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

\textbf{lemma} \textit{ev} \ (Suc \ (Suc \ n)) \Rightarrow \textit{ev} \ n

\textbf{proof} =
\begin{verbatim}
assume \textit{ev} \ (Suc \ (Suc \ n))" then show \textit{ev} \ n"
proof (cases)
... 
qed
qed
\end{verbatim}
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] ⇒ 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`. 

2
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 \( a \).

\textbf{type\_\textit{synonym}} \( ('Q,\Sigma)\ LTS = '('Q\times\ '\Sigma\times\ 'Q)\ set\)

First define an inductive predicate \textit{accept}, that characterizes the words accepted from a
given state \( q \), i.e., \textit{accept} \( F \delta q\ w \) holds iff word \( w \) is accepted from state \( q \).

\textbf{inductive} \textit{accept} :: \( 'Q \ set \Rightarrow ('Q,\Sigma)\ LTS \Rightarrow 'Q \Rightarrow \Sigma\ list \Rightarrow bool\)

for \( F \delta \) where

The product construction is a standard construction for the intersection of two FSMs.
Define a function \textit{prod}\(\delta\) that returns the transition relation of the product FSM of two
given FSMs:

\textbf{definition} \textit{prod}\(\delta\) :: \( ('Q1,\Sigma)\ LTS \Rightarrow ('Q2,\Sigma)\ LTS \Rightarrow ('Q1\times'Q2,\Sigma)\ LTS\)

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

\textbf{lemma} \textit{prod\_complete}:\n\hspace{1cm} assumes \( A \): \textit{accept} \( F1 \delta 1 q1\ w \)
\hspace{1cm} assumes \( B \): \textit{accept} \( F2 \delta 2 q2\ w \)
\hspace{1cm} shows \textit{accept} \( ('F1\times F2)\ (\textit{prod}\delta \delta1 \delta 2)\ (q1,q2)\ w \)
\hspace{1cm} using \( A \) \( B \)
\hspace{1cm} proof (induction arbitrary: q2 rule: \textit{accept}\_\textit{induct}[\textit{case\_names\ base\ step}])
\hspace{1cm} case (base q1)

next
\hspace{1cm} case (step q1 a q1' w q2)\textbf{qed}

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

\textbf{lemma} \textit{prod\_sound}:\n\hspace{1cm} assumes \textit{accept} \( ('F1\times F2)\ (\textit{prod}\delta \delta1 \delta 2)\ (q1,q2)\ w \)
\hspace{1cm} shows \textit{accept} \( F1 \delta 1 q1\ w \land \textit{accept} F2 \delta 2 q2\ w \)

Hint to get the induction through:

\hspace{1cm} \textbf{proof} –
\hspace{2cm} \{ fix q12
\hspace{3cm} \textbf{assume} \textit{accept} \( ('F1\times F2)\ (\textit{prod}\delta \delta1 \delta 2)\ q12\ w \)
\hspace{3cm} \textbf{hence} \textit{accept} \( F1 \delta 1 (\textit{fst} q12)\ w \land \textit{accept} F2 \delta 2 (\textit{snd} q12)\ w \)

Insert your inductive proof here

\}

3
thus \texttt{thesis using assms by auto}

\texttt{qed}

\texttt{end}