Exercise 4.1  Reflexive, Transitive Closure

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

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

Formalize the reflexive transitive closure as an inductive predicate:

\[
\text{inductive } star :: \left( ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool \right)
\]

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

\[
\text{lemma } star\_prepend: \left[ \left[ r \; x \; y ; \; \text{star } r \; y \; z \right] \right] \Rightarrow \text{star } r \; x \; z
\]
\[
\text{lemma } star\_append: \left[ \left[ \text{star } r \; x \; y ; \; r \; y \; z \right] \right] \Rightarrow \text{star } r \; x \; z
\]

Formalize the star predicate again, this time the other way around:

\[
\text{inductive } star' :: \left( ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a \Rightarrow 'a \Rightarrow bool \right)
\]

Prove the equivalence of your two formalizations:

\[
\text{lemma } \text{“star } r \; x \; y = star' \; r \; x \; y”
\]

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

Exercise 4.2  Elements of a List

Please give all your proofs in Isar, not apply style.

Define a recursive function \( \text{elems} \) returning the set of elements of a list:

\[
\text{fun } \text{elems} :: \left( 'a \; \text{list} \Rightarrow 'a \; \text{set} \right)
\]

To test your definition, prove:

\[
\text{lemma } \text{“elems } [1, \; 2, \; 3, \; (4::nat)] = \{1, \; 2, \; 3, \; 4\}”
\]
Now prove for each element \( x \) in a list \( xs \) that we can split \( xs \) in an \( x \)-free prefix, \( x \) itself, and a suffix:

\[
\text{lemma} \quad \neg x \in \text{elems} \; xs \implies \exists \; ys \; zs. \; xs = ys \@ x \# zs \wedge x \notin \text{elems} \; ys
\]

**Exercise 4.3  Rule Inversion**

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

\[
\text{inductive} \; ev :: \; \text{n}
\]

\[
\begin{align*}
\text{ev}0 & : \text{ev} \; 0 \mid \\
\text{evSS} & : \text{ev} \; n = \rightarrow \text{ev} \; (\text{Suc} \; (\text{Suc} \; n))
\end{align*}
\]

Prove the converse of rule \( \text{evSS} \) using rule inversion.

Hint: There are two ways to proceed. First, you can write a structured Isar proof using the \text{cases} method:

\[
\text{lemma} \quad \text{ev} \; (\text{Suc} \; (\text{Suc} \; n)) = \rightarrow \text{ev} \; n
\]

\[
\textproof ~
\begin{align*}
\text{assume} & \quad \text{ev} \; (\text{Suc} \; (\text{Suc} \; n)) \\
\text{then show} & \quad \text{ev} \; n
\end{align*}
\]

\[
\text{proof (cases)}
\]

\[
\quad \text{qed}
\]

\[
\text{qed}
\]

Alternatively, you can write a more automated proof by using the \text{inductive_cases} command to generate elimination rules. These rules can then be used with \text{“auto elim:”}.

(If given the \text{[elim]} attribute, \text{auto} will use them by default.)

\[
\text{inductive_cases evSS_elim} : \text{ev} \; (\text{Suc} \; (\text{Suc} \; n))
\]

Prove that the natural number three (i.e., \( \text{Suc} \; (\text{Suc} \; (\text{Suc} \; 0)) \)) 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 \text{inductive_cases}.

\[
\text{lemma} \quad \neg \text{ev} \; (\text{Suc} \; (\text{Suc} \; (\text{Suc} \; 0)))
\]

**Homework 4.1  A Binary Tree (5 points)**

*Submission until Monday 01.06.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE4]” in the subject of your email. If your name is Prénom Nom-de-Famille, please call your theory file Nom_de_Famille_Prenom.thy.*

Please give all your proofs in Isar, not apply style.
Consider the following inductive datatype of complete binary trees over ‘a’:

```haskell
datatype 'a tree =
  Leaf 'a
| Node "'a tree" "'a tree"
```

A tree is either a 'a-labeled leaf node or an inner node with two subtrees, a left and a right subtree.

The alphabet of a tree is the set of 'a elements it contains:

```haskell
primrec alphabet :: "'a tree ⇒ 'a set" where
  "alphabet (Leaf a) = {a}"
  "alphabet (Node t u) = alphabet t ∪ alphabet u"
```

The depth of an element a in a tree is the distance from this element to the root of the tree. If the element occurs several times, the maximal distance is taken. If it does not occur, 0 is returned:

```haskell
primrec depth :: "'a tree ⇒ 'a ⇒ nat" where
  "depth (Leaf _) a = 0"
  "depth (Node t u) a =
    max (if a ∈ alphabet t then depth t a + 1 else 0)
    (if a ∈ alphabet u then depth u a + 1 else 0)"
```

The height of a tree is the maximal distance from a leaf to the root of the tree. Define it:

```haskell
primrec height :: "'a tree ⇒ nat"
```

Check that it has the expected properties:

```haskell
value "height (Leaf a) = 0"
value "height (Node (Leaf a) (Leaf b)) = 1"
value "height (Node (Leaf a) (Node (Leaf b) (Leaf c))) = 2"
```

Prove the following lemmas:

```haskell
lemma depth_le_height: "depth t a ≤ height t"
lemma exists_at_height: "∃a ∈ alphabet t. depth t a = height t"
```

Hint: For the second lemma, make sure you have an intuitive understanding of why it holds before you attempt an Isar proof. Try to figure out which reasoning principles (induction, case analysis, etc.) you will need before even launching Isabelle.

**Homework 4.2 A Bounded-Loop Imperative Programming Language (10 points)**

Submission until Monday 01.06.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE4]” in the subject of your email. If your name is Prénom Nom-de-Famille, please call your theory file Nom_de_Famille_Prenom.thy.
Please give all your proofs in Isar, not apply style.

Anticipating on Chapters 6 and 7, we introduce a small imperative programming language and its semantics. One of the issues we will face in the second half of the course is that of nontermination. Here, we circumvent the problem by allowing only bounded loops. The construct $\text{Repeat } n \ c$ executes $c$ exactly $n$ times.

```isar
datatype com = 
  SKIP | Assign vname aexp | Seq com com | If bexp com com | Repeat nat com
```

Define a reasonable evaluation function for the above commands and a corresponding inductive predicate. Normally, we would be satisfied with one or the other style, but here we want to exercise and compare both styles.

```isar
fun eval :: "com ⇒ state ⇒ state" 
inductive ival :: "com ⇒ state ⇒ state ⇒ bool"
```

Hint: Recall that functions can be iterated (or “repeated” if you like) using $\circ$—e.g., $f \circ 2 = f \circ f$.

Prove that the two styles are equivalent, in the following precise sense:

```isar
lemma eval_sound: "ival c s (eval c s)" 
lemma eval_complete: "ival c s s' ⇒ s' = eval c s"
```

Finally, show that $ival$ is total and deterministic, reusing already proved results as much as possible:

```isar
lemma ival_total: "∃s'. ival c s s'" 
lemma ival_determ: 
  assumes s': "ival c s s'" and s'': "ival c s s''" 
  shows "s' = s''"
```

---

**Homework 4.3** The Reflexive, Symmetric, Transitive Closure (10 points)

Submission until Monday 01.06.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE4]” in the subject of your email. If your name is Prénom Nom-de-Famille, please call your theory file Nom_de_Famille_Prenom.thy.

Please give all your proofs in Isar, not apply style.

Binary relations are sometimes modeled as binary predicates $r$, sometimes as sets of pairs $r$. The notation '$a rel' abbreviates ('$a \times 'a) set$. Isabelle’s libraries support both views. For example, the transitive closure exists both as tranclp :: ("'a ⇒ 'a ⇒ bool") ⇒ 'a ⇒ 'a ⇒ bool and tranclp :: 'a rel ⇒ 'a rel.
Given a relation \( r :: 'a \text{ rel} \), the reflexive, symmetric, transitive closure of \( r \) is the least relation that includes \( r \) and that is reflexive, symmetric, and transitive.

Define this concept as an inductive set:

\[
\text{inductive set } \text{rst\_close} :: \quad \text{\textquote{\textquote{'a rel} \Rightarrow \textquote{'a rel}}} \quad \text{for } \text{\textquote{r :: \textquote{'a rel}}} 
\]

The \( \text{rst\_close} \) constant is not such a useful concept, because it can be expressed quite directly in terms of \( \text{rtrancl} \) (the reflexive, transitive closure—Isabelle’s built-in \textit{star}), the \textit{converse} of a relation (written as \( ^{-} \), \( ^{-1} \), or \( \text{	extless\textit{inverse}\textgreater} \)), and set union. Prove this:

\[
\text{lemma} \quad \text{\textquote{\textquote{rst\_close \, r = rtrancl \,(r \cup r^{-1})}}} 
\]

Hint: The proof has two subproofs, one for each direction of the biimplication (equivalence). Use the right rule among \( \text{rst\_close\_induct} \) and \( \text{rtrancl\_induct} \) for each direction.

---

**Homework 4.4  Kaminski’s Theorem (3 bonus points)**

Submission until Monday 01.06.2015 10:00 AM. Please send your submissions to jasmin.blanchette@mpi-inf.mpg.de and mathias.fleury@mpi-inf.mpg.de with “[CONCRETE4]” in the subject of your email. If your name is Prénom Nom-de-Famille, please call your theory file \textit{Nom\_de\_Famille\_Prenom.thy}.

Please give all your proofs in Isar, not apply style.

Prove the following theorem, by case distinction on \( b, f \text{ True} \), and \( f \text{ False} \). The theorem is due to Mark Kaminski, a former student at the Universität des Saarlandes (now at Oxford). For this theorem, you are exceptionally not allowed to use the \textit{metis}, \textit{smt}, \textit{smt2}, and \textit{smt3} proof methods.

\[
\text{theorem} \\
\text{fixes } f :: \text{\textquote{\textquote{bool} \Rightarrow \textquote{bool}}} \\
\text{shows} \quad \text{\textquote{f \,(f \,(fb)) = fb}} 
\]

And don’t forget the

\[
\text{end} 
\]