Functional Data Structures
Exercise Sheet 3

Exercise 3.1 Membership Test with Less Comparisons

In worst case, the isin function performs two comparisons per node. In this exercise, we
want to reduce this to one comparison per node, with the following idea:
One never tests for >, but always goes right if not <. However, one remembers the value
where one should have tested for =, and performs the comparison when a leaf is reached.

fun isin2 :: "('a:linorder) tree ⇒ 'a option ⇒ 'a ⇒ bool"
"The second parameter stores the value for the deferred comparison.

Show that your function is correct.

Hint: Auxiliary lemma for isin2 t (Some y) z !

lemma isin2_None:

"let t → isin2 t None x ⇒ x = "
Type unification failed

Type error in application: incompatible operand type
Operator: \texttt{op} \equiv \texttt{k} \ctx \texttt{a} \to \texttt{bool}
Operand: \texttt{f} \equiv \texttt{a} \to \texttt{option}

However, one remembers the value where one should have tested for `\texttt{==}`, and perform the comparison when a leaf is reached.

Fun: \texttt{isin2 :: `(a\texttt{\|}\texttt{linorder})\ tree \to \texttt{a\ option} \to \texttt{a} \to \texttt{bool}'}
- (The second parameter stores the value for the deferred comparison)
- \texttt{where: isin2 Leaf \texttt{r \equiv (case\ \texttt{r\ of\ Some\ a \to \texttt{mk| None \to \texttt{False)}}}$)

Test: Show that your function is correct.

Hints: Auxiliary lemma for `isin2 t (Some y) x`: 

\texttt{consts}

\texttt{isin2 :: `(a\ texttt{\|}\texttt{linorder})\ tree \to \texttt{a\ option} \to \texttt{a} \to \texttt{bool}'}
- (The second parameter stores the value for the deferred comparison)
- \texttt{where: isin2 Leaf \texttt{r \equiv (case\ \texttt{r\ of\ Some\ a \to \texttt{mk| None \to \texttt{False)}}}$)

Test: Show that your function is correct.
Inner syntax error: unexpected end of input
Failed to parse prop

Inner syntax errors
Failed to parse prop

Type unification failed
Type error in application: incompatible operand type

Operators: isin2 r :: 'a option => 'a => bool
Operands: a :: 'a
text : Show that your function is correct.

Hint: Auxiliary lemma for \( \text{isin2} \) \( \text{t} \) (Some x) x:

\[ \text{lemma isin2_None:} \]
\[ \begin{array}{c}
\text{fun isin2 :: } \langle m:\text{listorder} \rangle \text{ tree } \rightarrow \langle o\text{ption } a = \text{bool}\rangle \\
\text{The second parameter stores the value for the deferred comparison. where} \\
\text{isin2 Leaf ren k } = \langle \text{case ren of Some a } \rightarrow \text{mk } | \text{None } \rightarrow \text{False}\rangle \\
\text{isin2 (Node a l r) ren k } = \\
\text{if } k = a \text{ then isin2 l ren k } \\
\text{else isin2 r ren k } \\
\end{array} \]

\[ \text{consts} \]
\[ \text{isin2 :: } \langle a\text{ tree } \rightarrow \langle o\text{ption } a = \text{bool}\rangle \]
\[ \text{Found termination order: } \langle \text{up, size [fst p]} \rangle < \text{inlex} = () \]

proof (prove)
\[ \text{goal (1 subgoal):} \]
\[ 1. \text{fun isin2 :: } \langle \text{m:\text{listorder}} \rangle \text{ tree } \rightarrow \langle \text{o\text{ption } a = \text{bool}} \rangle \]
\[ \text{The second parameter stores the value for the deferred comparison.} \]
\[ \text{where} \]
\[ \text{isin2 Leaf ren k } = \langle \text{case ren of Some a } \rightarrow \text{mk } | \text{None } \rightarrow \text{False}\rangle \]
\[ \text{isin2 (Node a l r) ren k } = \\
\text{if } k = a \text{ then isin2 l ren k } \\
\text{else isin2 r ren k } \\
\]

\[ \text{test : Show that your function is correct.} \]

\[ \text{consts} \]
\[ \text{isin2 :: } \langle a\text{ tree } \rightarrow \langle o\text{ption } a = \text{bool}\rangle \]
\[ \text{Found termination order: } \langle \text{up, size [fst p]} \rangle < \text{inlex} = () \]
test: Show that your function is correct.

HINT: Auxiliary lemma for `isin2 t (Some y) x`!

lemma isin2_None:
  `bt == isin2 t (Some a) k = isin t k`
  apply (induction t)
  apply auto

proof (prove)
goal: (8 subgoals):
1. \( \forall x \forall y. (x < y) \land (y < x) \rightarrow \text{set_tree} t x y \rightarrow \text{set_tree} t y x \)

inner syntax error: unexpected end of input:
Failed to parse prop.
```plaintext
proof (prove)
  goal (1 subgoal):
  1. [bt : Vect_set_tree t. a < x] \implies \text{isin2} t (\text{Some} a) k = (a = k \lor \text{isin} t k)

proof (prove)
  goal (1 subgoal):
  1. [bt : Vect_set_tree t. a < x] \implies \text{isin2} t (\text{Some} a) k = (a = k \lor \text{isin} t k)

lemma
  "bt \implies \text{set_tree} t. a \rightarrow \text{isin2} t (\text{Some} a) k \rightarrow (a = k \lor \text{isin} t k)"

lemma isin2_Name:
  "bt \implies \text{isin2} t \text{ None } k = \text{isin} t k"

apply (induction t)

lemma isin2_Name:
  "bt \implies \text{isin2} t \text{ None } k = \text{isin} t k"

apply (induction t)

apply (auto simp: a)

done

test

Exercise (Height-Preserving In-Order Join)

theorem isin2_Name: [bt : \text{isin2} t t' t' None k = \text{isin} t t' k]
```
Functional Data Structures

Exercise Sheet 3

Exercise 3.1 Membership Test with Less Comparisons

In worst case, the isin function performs two comparisons per node. In this exercise, we want to reduce this to one comparison per node, with the following idea:

One never tests for \( > \), but always goes right if not \( < \). However, one remembers the value where one should have tested for \( = \), and performs the comparison when a leaf is reached.

```haskell
fun isin2 :: "(a, v :: a) tree => 'a option => 'a => bool"
    The second parameter stores the value for the deferred comparison.

Show that your function is correct.
Hint: Auxiliary lemma for isin2 t (Some y) x !

lemmas isin2 None:
    "bt t \rightarrow isin2 t None k = isin t k"
```

Exercise 3.2 Weight-Preserving In-Order Join

Write a function that joins two binary trees such that:
- The in-order traversal of the new tree is the concatenation of the in-order traversals of the original trees.
Write a function that joins two binary trees such that

- The in-order traversal of the new tree is the concatenation of the in-order traversals of the original trees.
- The new tree is at most one higher than the highest original tree.

Hint: Once you get the function right, proofs are easy!

```haskell
fun join :: "a tree → a tree → a tree"

where

  "join Leaf t = t"

  "join t Leaf = t"

  "join (Node l1 l2 r1) (Node l2 a2 r2) = (case join l1 r1 of |
  Leaf a1 → undefined |
  Node l a r → Node l (join l1 r1) (join l2 r2))"

lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 # inorder t2"
```

```haskell
consts

(leaf) (tree) (join) (inorder)
```

```haskell
lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 # inorder t2"
```
Lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 @ inorder t2"
by (auto split: tree.splits)

Lemma "height\(\)join t1 t2\) \leq max (height t1, height t2) + 1"
apply (induction t1 t2 rules: join.induct)
by (auto split: tree.splits)

Exercise (Implement Delete)
Implement delete using the \(\text{join}\) function from last exercise.

Theorem height\(\)join t1.0 t1.0\) \leq max (height t1.0, height t1.0) + 1
Lemma join_inorder[simp]: "inorder(join t1 t2) = inorder t1 # inorder t2"
by (auto split: tree.splits)

Lemma height[join t1 t2] ≤ max (height t1) (height t2) + 1
apply (induction t1 t2 rules: injuneduct)
by (auto split: tree.splits)

Exercise (Implement Delete): Implement delete using the \emph{join} function from last exercise.

Note: At this point, we are not interested in the implementation details of join any more, but just in its specification, i.e., what it does to trees. Thus, as first step, we declare its equations to not being automatically unfolded.

declare join_simps[simp del]
Note: At this point, we are not interested in the implementation details of join any more, but just in its specification, i.e., what it does to trees.

Thus, as first step, we declare its equations to not being automatically unfolded.

```

thm join.simps

declare join.simps[simp del]
```

Both, \( \text{set_tree} \) and \( \text{set} \) can be expressed by the inorder traversal over trees:

```

thm set_inorder[symmetric] \text{bst_iff_sorted_wrt_less}
```

Note: As \( \text{set} \) \( \text{set_inorder} \) is declared as simp.

Be careful not to have both directions of the lemma in the simpset at the same time, otherwise the simpifier is likely to loop.

```

set_tree \texttt{lt} = \text{set} \{ \text{inorder} \texttt{lt} \}

\text{bst} \texttt{lt} = \text{sorted_wrt op} \< \{ \text{inorder} \texttt{lt} \}
```

Alternatively, you can write: declare set_inorder(simp_del); to remove it once and for all.

test "For the sorted_wrt predicate, you might want to use these lemmas as simp."
then sorted_wrt_append sorted_wrt_concl

test "Show that join preserves the set of entries:"
    lemma [simp]: "set_tree (join t1 t2) = set_tree t1 \cup set_tree t2"

test "Show that joining the left and right child of a BST is again a BST:"
    lemma [simp]: "bst (Node l (x::l) r) \implies bst (join l r)"

proof (prove)
  goal (1 subgoal):  
    1. set_tree (join t1 t2) = set_tree t1 \cup set_tree t2
\begin{verbatim}
join (\text{simp})

declare join (\text{simp}) (\text{simp del})

\textbf{text:} Both \texttt{set_tree} and \texttt{bst} can be expressed by the inorder traversal over trees.

\textbf{proof:} Note: As \texttt{set\_inorder} is declared as \texttt{simp}, be careful not to have both directions of the lemma in the simpset at the same time, otherwise the simplifier is likely to loop.

\begin{itemize}
  \item \texttt{set\_tree} \texttt{P} = \texttt{set\_inorder} \texttt{P}
  \item \texttt{bst\_P} = \texttt{sorted\_wrt} \texttt{op} \texttt{< \_inorder \_P}
\end{itemize}

\begin{verbatim}
proof (prove)
goal (1 subgoal):
1. \texttt{bst} (\texttt{L}, \texttt{X}, \texttt{R}) \texttt{==} \texttt{bst} (\texttt{join} \texttt{L} \texttt{R})
\end{verbatim}
\end{verbatim}
show that join preserves the set of entries:
  \[ \text{lemma } \text{(simp add set_inorder[symmetric])} \]
  \[ \text{text } \text{(simp add set_inorder[symmetric])} \]

show that joining the left and right child of a BST is again a BST:
  \[ \text{lemma } \text{(simp add set_inorder[symmetric])} \]
  \[ \text{text } \text{(simp add set_inorder[symmetric])} \]

apply (simp add bst_iff_sorted_wrt_less_del: join_inorder)

show that joining the left and right child of a BST is again a BST:
  \[ \text{lemma } \text{(simp add set_inorder[symmetric])} \]
  \[ \text{text } \text{(simp add set_inorder[symmetric])} \]

apply (simp add bst_iff_sorted_wrt_less del: join_inorder)

proof (prove)

\begin{enumerate}
  \item sorted_wrt op \langle inorder \rangle \\
    sorted_wrt op \langle inorder \rangle \land (\forall x. set_tree l. x < x) \\
    sorted_wrt op \langle inorder \rangle \land \text{join_inorder} \rangle \\
  \end{enumerate}
proof (prove)

goal (1 subgoal):
1. \( \text{bst} \{ l, k, r \} \rightarrow \text{bst} \{ \text{join} l, r \} \)

by (simp add \{set inorder\}[symmetric])


proof (prove)

goal (1 subgoal):
1. \( \forall x : x x x \)

\[
\begin{align*}
\{ x : \text{set inorder} \{ l \} ; x x x : \text{set inorder} \{ r \} ; \text{sorted wrt op < (inorder l)} ; \\
\text{sorted wrt op < (inorder r)} ; \quad \text{Value set tree l} \ l \ x x \times ; \quad \text{Value set tree r} \ r \ x x x \}
\rightarrow x x x
\end{align*}
\]


proof (prove)

goal (1 subgoal):
1. \( \forall x : x x x \)

\[
\begin{align*}
\{ x : \text{set inorder} \{ l \} ; x x x : \text{set inorder} \{ r \} ; \text{sorted wrt op < (inorder l)} ; \\
\text{sorted wrt op < (inorder r)} ; \quad \text{Value set tree l} \ l \ x x \times ; \quad \text{Value set tree r} \ r \ x x x \}
\rightarrow x x x
\end{align*}
\]
text: Show that joining the left and right child of a BST is again a BST.

lemma [simp]: \( \text{bst (Node l x r) \rightarrow bst (join l r}) \)

apply (auto simp add: bst_iff_sorted_wrt_less sorted_wrt_append set_inorder[symmetric])

text: Implement a delete function using the idea contained in the lemmas above.

fun delete :: "'a :: linorder \\
        tree \rightarrow tree \rightarrow tree"
where
    "delete _ x = undefined"

text: Prove it correct! Note: You'll need the first lemma to prove the second one.

lemma [simp]: \( \text{bst t \rightarrow set_tree (delete x t) = set_tree t - \{x\}} \)

proof (prove)
  goal (1 subgoal):
  1. \( \forall x \in \text{set (inorder l)}; x \neq \text{set (inorder r)}; \text{sorted_wrt op < (inorder l)}; \text{sorted_wrt op < (inorder r)}\) \: \( \forall x \in \text{set (inorder l)}; x < x\) \: \( \forall x \in \text{set (inorder r)}; x < x\)
     \( \rightarrow x < x\)

proof (prove)
  goal (1 subgoal):
  1. \( \forall x \in \text{set (inorder l)}; x \neq \text{set (inorder r)}; \text{sorted_wrt op < (inorder l)}; \text{sorted_wrt op < (inorder r)}\) \: \( \forall x \in \text{set (inorder l)}; x < x\) \: \( \forall x \in \text{set (inorder r)}; x < x\)
     \( \rightarrow x < x\)
Theorem: \( \text{bst} (\text{ll}, \text{lx}, \text{lr}) \implies \text{bst} (\text{join} \text{ ll, lr}) \)

Proof:

Goal: \( \text{bst} (\text{ll}, \text{lx}, \text{lr}) \implies \text{bst} (\text{join} \text{ ll, lr}) \)

1. \( \text{bst} (\text{ll}, \text{lx}, \text{lr}) \)
2. \( \text{bst} (\text{join} \text{ ll, lr}) \)