Exercise 4.1  List Elements in Interval

Write a function to in-order list all elements of a BST in a given interval. I.e., \( \text{in-range} \ t \ a \ v \) shall list all elements \( x \) with \( a \leq x \leq v \). Write a recursive function that does not descend into nodes that definitely contain no elements in the given range.

\[
\text{fun in_range :: } \ast \times \ast \times \ast \Rightarrow \ast
\]

Show that you list the right set of elements

\[\text{lemma } "\text{let } t \Rightarrow \text{set (in-range t a v) } = \{ x \mid \text{tree t} \wedge a \leq x \leq v \}\]"\n
Show that your list is actually in-order

\[\text{lemma } "\text{let } t \Rightarrow \text{in-range t a v} = \text{filter (\lambda x. a \leq x \land x \leq v) (inord t)}"\]

Exercise 4.2  Pretty Printing of Binary Trees
You may want to start with an auxiliary function, that takes the BST with the elements seen so far as additional argument, and then define the actual function.

```isabelle
fun bst_rendups_aux :: 
  "'a::{linorder, list} => 'a list => 'a list" where
| "bst_rendups_aux [] t []" | "bst_rendups_aux t [] (i#x#xs)" | "bst_rendups_aux xs i x t xs" | "bst_rendups_aux (ins x t) xs" |
```

Definition: "bst_rendups xs = bst_rendups_aux Leaf xs"

**Lemma:** Show that your function preserves the set of elements, and returns a list with no duplicates (predicate distinct in Isabelle).

**Hint:** Generalization!

```isabelle
lemma bst_rendups_aux_set[simp]: "bst t = set (bst_rendups_aux t xs) = set xs = set_tree t" 
apply (induction xs arbitrary: x) 
apply (auto simp set_tree_isin set_tree_ins bst_ins) 
done
```

```isabelle
theorem bst_rendups_aux_distinct[simp]: "bst t = distinct (bst_rendups_aux t xs)" 
apply (induction xs arbitrary: x) 
apply (auto simp set_tree_isin set_tree_ins bst_ins) 
done
```

**Lemma:** "set (bst_rendups xs) = set xs"
Exercise 4.1 List Elements in Interval

Write a function to in-order list all elements of a BST in a given interval. I.e., \( \text{in-range } t \ u \ v \) shall list all elements \( x \) with \( u \leq x \leq v \). Write a recursive function that does not descend into nodes that definitely contain no elements in the given range.

fun \( \text{in-range } t \ u \ v \) :: "a :: lnode tree \( = \) a \cdot a \cdot l list"

Show that your list is actually in-order

lemma \( \forall t = \text{set } \text{in-range } t \ u \ v = \{ x \in \text{set-tree } t.\ \ u \leq x \leq v \} \)"  

Show that your list is actually in-order

lemma \( \forall t = \text{in-range } t \ u \ v = \text{filter } (x.\ \ u \leq x \leq v) \) (inorder t)"

Exercise 4.2 Pretty Printing of Binary Trees
Write a function to in-order list all elements of a BST in a given interval. I.e., \( \text{in-range } t u v \) shall list all elements \( x \) with \( u \leq x \leq v \).

```haskell
fun in_range :: "a:inorder tree \( \to \) 'a \( \to \) 'a \( \to \) 'a list"
where
  "in_range Leaf u v" = []
  | "in_range (Node l a r) u v" =
    if a \( \leq \) u then "in_range l u v" ++ [a] ++ "in_range r u v" else []
    if a \( \geq \) v then "in_range l u v" ++ [a] ++ "in_range r u v" else []

test = Show that you list the right set of elements:
lemma "bst t \( \Rightarrow \) set (in_range t u v) = in_order t"
```

```haskell
proof
```

```haskell
lemma "bst t \( \Rightarrow \) set (in_range t u v) = filter \( \lambda x . u \leq x \leq v \) (inorder t)"
```

```haskell
proof
```

```haskell
Show that your list is actually in-order:
```

```haskell
test = Show that your list is actually in-order:
```

```haskell
lemma "bst t \( \Rightarrow \) set (in_range t u v) = \{ \text{in-tree } t . u \leq x \leq v \}"
```

```haskell
proof
```
Write a function to in-order list all elements of a BST in a given interval. I.e., \( \text{in-range} \ t \ u \ v \) shall list all elements \( x \) with \( x \in [u,v] \).

Write a recursive function that does not descend into nodes that definitely contain no elements in the given range.

```haskell
run_in_range :: "[a] -> ([a] -> [a] -> [a]) -> [a]
where
  in_range Leaf u v = []
  in_range (Node l a r) u v =
    if u <= a then in_range l u a else []
    ++ if a <= v then a : in_range r a v else []
```

Show that you list the right set of elements:

```haskell
lemma bst t \rightarrow\!\! \text{in-range} \ t \ u \ v \equiv \ (\text{inorder-tree} t) \ u \times \times \times \ v
```

Show that your list is actually in-order:

```haskell
done
```

proof (prove)

goal (1 subgoal):

1. \( \text{bst} t \rightarrow\!\! \text{in-range} \ t \ u \ v \equiv \ (\text{inorder-tree} t) \ u \times \times \times \ v \)

Proof state:

```
  1. \( \text{bst} t \rightarrow\!\! \text{in-range} \ t \ u \ v \equiv \ (\text{inorder-tree} t) \ u \times \times \times \ v \)
```

Use induction on \( t \):
Show that your list is actually in-order.

Lemma: "bst t \in \text{in-range} t u v = \text{filter} (\lambda x, u(x) \land x \leq v) (\text{inorder} t)"

apply (induction t)
apply simp
apply simp
apply simp
apply simp
Sledgehammering...
"search": Timed out
"cyc": Timed out
"x": Timed out
"e": Timed out

proof (prove)
goal (1 subgoal):
1. \( \forall x. x2 \leq x \leq v \rightarrow
   \begin{align*}
   \& (u \leq x) \land (v \leq x) \\
   \& (u \leq x) \land (v \leq x) \\
   \& (u \leq x) \land (v \leq x) \\
   \& (u \leq x) \land (v \leq x) \\
   \& (u \leq x) \land (v \leq x) \\
   \end{align*}
\)
Inner syntax error: unexpected end of input
Failed to parse prop

proof (prove)
goal (2 subgoals):
1. \([x] = 1 \leftrightarrow l1 \land \lnot l2 \implies l1 = \{\}
2. \([x] = 1 \leftrightarrow l1 \land \lnot l2 \implies l2 = \{\]
Outer syntax errors: command expected, but keyword `in` was found

proof (prove):

1. \( \text{[v] } \in \text{in\_range} \ t \ u \ v \Rightarrow \text{filter} \ (x. \text{u \& x \& v)} \ \text{in\_order} \ t \)
2. \( \text{[v] } \in \text{in\_range} \ t \ u \ v \Rightarrow \text{filter} \ (x. \text{u \& x \& v)} \ \text{in\_order} \ t \)

apply simp

proof (prove):

1. \( \text{in\_range} \ t \ u \ v \Rightarrow \text{filter} \ (x. \text{u \& x \& v)} \ \text{in\_order} \ t \)
2. \( \text{in\_range} \ t \ u \ v \Rightarrow \text{filter} \ (x. \text{u \& x \& v)} \ \text{in\_order} \ t \)

apply simp
proof (prove)

goal (1 subgoal):
1. \( \forall t1 \times t2 \times t3 \). \( \text{in_range}(t1, t2, t3) \) \( \Rightarrow \) (property)

apply simp

apply clarsimp

proof (prove)

goal (1 subgoal):
1. \( \forall t1 \times t2 \times t3 \). \( \text{in_range}(t1, t2, t3) \) \( \Rightarrow \) (property)

apply simp

apply clarsimp

proof (prove)

goal (1 subgoal):
1. \( \forall t1 \times t2 \times t3 \). \( \text{in_range}(t1, t2, t3) \) \( \Rightarrow \) (property)

apply simp

apply clarsimp
proof (prove):
goal (1 subgoal):
1. \(\forall t \in t \implies u \in u \implies v \in v \implies (\text{in-order} t, u \cup x \cup v) \implies \text{(in-order} t)\)
   apply (induction t)
   apply simp
   apply (clarsimp simp: filter_empty_conv)

proof (prove):
goal (1 subgoal):
1. \(\forall t \in t \implies u \in u \implies v \in v \implies (\text{in-range} t, u \cup x \cup v) \implies \text{(in-range} t)\)
   apply (induction t)
   apply simp
   apply (clarsimp simp: filter_empty_conv)
   apply safe
   apply simp
   apply simp_all

proof (prove):
goal (1 subgoal):
1. \(\forall t \in t \implies u \in u \implies v \in v \implies (\text{in-range} t, u \cup x \cup v) \implies \text{(in-range} t)\)
   apply (induction t)
   apply simp
   apply (clarsimp simp: filter_empty_conv)
   apply safe
   apply simp
   apply simp_all

proof (prove):
goal (10 subgoals):
1. \(\forall t \in t \implies u \in u \implies v \in v \implies (\text{in-range} t, u \cup x \cup v) \implies \text{(in-range} t)\)
   apply (induction t)
   apply simp
   apply (clarsimp simp: filter_empty_conv)
   apply safe
   apply simp
   apply simp_all

proof (prove):
goal (10 subgoals):
1. \(\forall t \in t \implies u \in u \implies v \in v \implies (\text{in-range} t, u \cup x \cup v) \implies \text{(in-range} t)\)
   apply (induction t)
   apply simp
   apply (clarsimp simp: filter_empty_conv)
   apply safe
   apply simp
   apply simp_all
proof (prove):

goal (10 subgoals):

1. \( \forall t. t, \left[ \text{in_range } t \land \forall x: \text{set_tree } t \wedge x < x_2 \Rightarrow (u \cdot x \land x_2 \neq v) \right] \Rightarrow (u \cdot x_2 \land x_2 \neq v) \)

2. \( \text{filter_empty_conv} \)

3. \( \text{simp_all} \)

4. \( \text{auto simp filter_empty_conv} \)

5. \( \text{done} \)

\[ \text{(filter } P \text{ } X = \{ \} ) = (\forall x: \text{set } X \wedge \neg P x) \]
proof (prove):

goal (10 subgoals):
1. \( x_1 \leq x_2 \leq x_3 \land \ldots \land x_n \leq x_{n+1} \land \ldots \land x_{m-1} \leq x_m \leq x_{m+1} \)
2. \( x_1 \leq x_2 \leq x_3 \land \ldots \land x_n \leq x_{n+1} \land \ldots \land x_{m-1} \leq x_m \leq x_{m+1} \)

proof (prove):

goal (18 subgoals):
1. \( x_1 \leq x_2 \leq x_3 \land \ldots \land x_n \leq x_{n+1} \land \ldots \land x_{m-1} \leq x_m \leq x_{m+1} \)
2. \( x_1 \leq x_2 \leq x_3 \land \ldots \land x_n \leq x_{n+1} \land \ldots \land x_{m-1} \leq x_m \leq x_{m+1} \)
proof (prove)
goal: no subgoals!
proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)

proof (prove)
goal (1 subgoal):
1. \( \text{bst} \rightarrow \text{in-range (t u v)} \ [x \rightarrow \text{inorder t, } u \leq x \wedge x \leq v] \)
text While this function itself is not very useful, the induction rule generated by the function package allows simultaneous induction over two trees:

```
print_statement_bin_tree.induct
```

text Binary trees can be uniquely pretty-printed by emitting a symbol L for a leaf, and a symbol N for a node. Each N is followed by the pretty-prints of the left and right tree. No additional brackets are required.

datatype 'a thw = L | N 'a

fun pretty :: 'a tree -> 'a tchar list
  where
    "pretty" = undefined

where
  "pretty" = undefined

text Show that pretty-printing is actually unique, i.e., no two different trees are pretty-printed the same way.

HINT: Auxiliary lemmas. Simultaneous induction over both trees.

lemma pretty_unique: "pretty t = pretty t'" \Rightarrow t = t'

text Exercise (Enumeration of Trees):
datatype tchar = L | N | (a tchar)

fun pretty :: "a tree ⇒ tchar list"
where
  "pretty Leaf = []"
| "pretty (Node l a r) = N # pretty l # a # pretty r"

lemma pretty_unique: "pretty t = pretty t' ⟷ t = t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto

proof (prove)
goal 1 [subgoal 1]:
1. \(l_1 \cdot r_1 \cdot l_2 \cdot r_2\)
   \[\text{pretty } l_1 \cdot \text{pretty } l_2 \rightarrow l_1 = l_2; \text{ pretty } r_1 \cdot \text{ pretty } r_2 \rightarrow r_1 = r_2;\]
   \[\text{ pretty } l_1 = \text{ pretty } l_2 \land \text{ pretty } r_1 = \text{ pretty } r_2 \rightarrow l_1 = l_2; r_1 = r_2;\]

lemma pretty_unique: "\(\text{pretty } t = \text{ pretty } t'\) ⟷ t = t'"
apply (induction t t' rule: bin_tree2.induct)
apply auto

proof (prove)
goal 2 [subgoal 1]:
1. \(l_1 \cdot r_1 \cdot l_2 \cdot r_2\)
   \[\text{ pretty } l_1 = \text{ pretty } l_2 \land l_1 = l_2; \text{ pretty } r_1 = \text{ pretty } r_2 \rightarrow r_1 = r_2;\]
   \[\text{ pretty } l_1 = \text{ pretty } l_2 \land \text{ pretty } r_1 = \text{ pretty } r_2 \rightarrow l_1 = l_2; r_1 = r_2;\]
proof (prove)
goal (2 subgoals):
1. \( \forall r_1 \ r_2 \ r_2. \)
   \[ \text{pretty } l_1 = \text{pretty } l_2 = l_1 = l_2; \text{ pretty } r_1 = \text{ pretty } r_2 = r_1 = r_2; \]
   \[ \text{ pretty } l_1 \land \text{ pretty } r_1 = \text{ pretty } l_2 \land \text{ pretty } r_2 \]
   \[ l_1 = l_2 \]
2. \( \forall r_1 \ r_2 \ r_2. \)
   \[ \text{pretty } l_1 = \text{pretty } l_2 = l_1 = l_2; \text{ pretty } r_1 = \text{ pretty } r_2 = r_1 = r_2; \]
   \[ \text{ pretty } l_1 \land \text{ pretty } r_1 = \text{ pretty } l_2 \land \text{ pretty } r_2 \]
   \[ l_1 = l_2 \]

lemma pretty_unique: \( \text{pretty } l \land \text{ pretty } t' \equiv \text{ pretty } t \land t' \equiv t \).
apply induction \( t \equiv t' \) rules bin_tree2_induct
apply auto
proof (prove):
1. pretty [] @ xs = pretty [] @ xs' ➔ [] = []
2. \( \langle l_1 \cdot u, r_1 \rangle, l_2 \cdot u, r_2 \rangle \triangleq \langle l_1 \cdot u, r_1 \rangle, l_2 \cdot u, r_2 \rangle \triangleq \langle l_1 \cdot u, r_1 \rangle, l_2 \cdot u, r_2 \rangle \)

lemma pretty_unique: 'pretty t @ xs = pretty t' @ xs' ➔ t = t'
apply induction t' arbitrary: xs xs' rule: bin_tree2.induct
apply auto

proof (prove):
1. \( \forall r_1 \cdot l_2 \cdot r_2 \cdot \exists x \cdot \exists x': \) pretty \( r_1 \cdot l_2 \cdot r_2 \cdot x \triangleq \) \( r_1 \cdot l_2 \cdot r_2 \cdot x' \)
2. \( \langle x, x' \rangle, \) pretty \( r_1 \cdot l_2 \cdot r_2 \cdot x \triangleq \) \( r_1 \cdot l_2 \cdot r_2 \cdot x' \)

Write a function that generates the set of all trees up to a given height.
(The other direction, i.e. that all trees are contained, requires an advanced case split, which has not yet been introduced in the lecture, so it is omitted here.)
Submission until Friday, May 26, 11:59am.

In this homework, we will develop a binary search tree that additionally stores the rank (number of nodes) of the left subtree in each node.

With this auxiliary information, it is easy to implement a rank query, i.e., to return the position of a given element in the inorder traversal.

datatype 'a tree = Leaf | Node 'a tree nat 'a tree

Define a function to count the number of nodes in a tree

fun num_nodes :: "'a tree ⇒ nat" where

Define a function to check for the invariant: search tree property and the correct rank annotation (number of nodes in left subtree)

fun bst :: "'a:order tree ⇒ bool" where

Define the insert function. You may assume that the value to be inserted is not contained in the tree. Note: Double-check to correctly update the rank annotation.

fun insert :: "'a:order ⇒ 'a tree ⇒ 'a tree" where

Show that insert actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.

datatype 'a tree = Leaf | Node 'a tree nat 'a tree

Define a function to count the number of nodes in a tree

fun num_nodes :: "'a tree ⇒ nat" where

Define a function to check for the invariant: search tree property and the correct rank annotation (number of nodes in left subtree)

fun bst :: "'a:order tree ⇒ bool" where

Define the insert function. You may assume that the value to be inserted is not contained in the tree. Note: Double-check to correctly update the rank annotation.

fun insert :: "'a:order ⇒ 'a tree ⇒ 'a tree" where

Show that insert actually inserts, and preserves the invariant. Hint: Auxiliary lemma on number of nodes.
fun join :: "('a::linorder ⇒ 'a tree) ⇒ 'a tree ⇒ 'a tree" where
Show that join actually inserts, and preserves the invariant. Hint: Auxiliary lemma on
number of nodes.

lemma join_set: "join (join t x t) = insert x (join t x t)"
lemma "join (join t x t) = insert t (join t x t)"

Define the membership query function and show it correct.

fun mem :: "('a::linorder ⇒ 'a tree ⇒ bool) where

Show that pretty-printing is actually unique, i.e., no two different trees are pretty-printed the same way. Hint: Auxiliary lemma. Simultaneous induction over both trees.

\textbf{Lemma} pretty_unique: \textit{\texttt{pretty }t = pretty \ \texttt{t'} \Rightarrow \texttt{t} = \texttt{t'}}

\textbf{Exercise 4.3} Enumeration of Trees

Write a function that generates the set of all trees up to a given height. Show that only trees up to the specified height are contained.
(The other direction, i.e., that all trees are contained, requires an advanced case split, which has not yet been introduced in the lecture, so it is omitted here)

\texttt{fun enum : \texttt{nat} \Rightarrow \texttt{unit set}} \ \texttt{where}

\texttt{lemma enum_sound: \forall \texttt{n}. \texttt{enum \ n} \subseteq \texttt{trees \ n}}

\textbf{Homework 4} Rank Annotated Trees

Submission until Friday, May 26, 11:59pm.