Chapter 3

Case Study: Binary Search Trees

Type: 'a set

Operations: \( a \in A, A \cup B, \ldots \)
Type: 'a set
Operations: \( a \in A, A \cup B, \ldots \)
Bounded quantification: \( \forall a \in A. \ P \)

Proof method \textit{auto} knows (a little) about sets.

The (binary) tree library

\texttt{~/src/HOL/Library/Tree.thy}

datatype 'a tree = Leaf | Node ('a tree) 'a ('a tree)
The (binary) tree library

```haskell
~~/src/HOL/Library/Tree.thy

datatype 'a tree = Leaf | Node ('a tree) 'a ('a tree)
```

Abbreviations:

```haskell
() ≡ Leaf
```

Size = number of nodes:
```
size :: 'a tree ⇒ nat
```

```haskell
size () = 0
size (l, a, r) = size l + size r + 1
```
**The (binary) tree library**

Size = number of nodes:
\[
size :: 'a tree \Rightarrow \text{nat} \\
size \langle \rangle = 0 \\
size \langle l, _, r \rangle = size l + size r + 1
\]

Inorder listing:
\[
inorder :: 'a tree \Rightarrow 'a list \\
inorder \langle \rangle = [] \\
inorder \langle l, x, r \rangle = inorder l @ [x] @ inorder r
\]

**The (binary) tree library**

The set of elements in a tree:
\[
set\_tree :: 'a tree \Rightarrow 'a set \\
set\_tree \langle \rangle = \{\} \\
set\_tree \langle l, a, r \rangle = set\_tree l \cup \{a\} \cup set\_tree r
\]
The (binary) tree library

The set of elements in a tree:
\[ set\_tree : \text{`a tree} \Rightarrow \text{`a set} \]
\[ set\_tree \emptyset = \{\} \]
\[ set\_tree \langle l, a, r \rangle = set\_tree l \cup \{a\} \cup set\_tree r \]

Applying a function to all elements a tree:
\[ map\_tree : (\text{`a} \Rightarrow \text{`b}) \Rightarrow \text{`a tree} \Rightarrow \text{`b tree} \]
\[ map\_tree f \emptyset = \emptyset \]
\[ map\_tree f \langle l, a, r \rangle = \langle map\_tree f l, f a, map\_tree f r \rangle \]

The (binary) tree library

Binary search tree invariant:
\[ bst : \text{`a tree} \Rightarrow \text{bool} \]
\[ bst \emptyset = \text{True} \]
\[ bst \langle l, a, r \rangle = \]
\[ \text{bst } l \land \]
\[ \text{bst } r \land \]
\[ (\forall x \in set\_tree l. x < a) \land (\forall x \in set\_tree r. a < x)) \]
The (binary) tree library

Binary search tree invariant:
\[ bst :: 'a tree \Rightarrow \text{bool} \]

\[ bst \emptyset = \text{True} \]
\[ bst \langle l, a, r \rangle = \]
\[ (bst l \land
\quad bst r \land
\quad (\forall x \in \text{set}_l. x < a) \land (\forall x \in \text{set}_r. a < x)) \]

For any type 'a?

Isabelle’s type classes

Isabelle’s type classes

A type class is defined by
- a set of required functions (the interface)

Isabelle’s type classes

A type class is defined by
- a set of required functions (the interface)
- and a set of axioms about those functions
Isabelle’s type classes

A type class is defined by
- a set of required functions (the interface)
- and a set of axioms about those functions

Example: class \textit{linorder}: linear orders with $\leq$, $<$

A type belongs to some class if
- the interface functions are defined on that type
- and satisfy the axioms of the class
Isabelle’s type classes

A type class is defined by
- a set of required functions (the interface)
- and a set of axioms about those functions
Example: class linorder: linear orders with ≤, <
A type belongs to some class if
- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)
Notation: \( \tau :: C \) means type \( \tau \) belongs to class \( C \)

Example: \( bst :: (\'a :: linorder) \text{ tree} \Rightarrow \text{bool} \)

Isabelle’s type classes

A type class is defined by
- a set of required functions (the interface)
- and a set of axioms about those functions
Example: class linorder: linear orders with ≤, <
A type belongs to some class if
- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)
Notation: \( \tau :: C \) means type \( \tau \) belongs to class \( C \)
Example: \( bst :: (\'a :: linorder) \text{ tree} \Rightarrow \text{bool} \)

Case study

BST_Demo.thy
Case study