

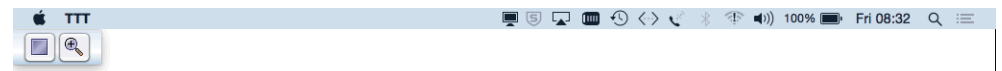
Script generated by TTT

Title: FDS (12.05.2017)

Date: Fri May 12 08:32:41 CEST 2017

Duration: 88:27 min

Pages: 32



Chapter 3

Case Study: Binary Search Trees

84



Preview: sets

Type: *'a set*



Preview: sets

Type: *'a set*

Operations: $a \in A, A \cup B, \dots$



Preview: sets

Type: $'a\ set$

Operations: $a \in A, A \cup B, \dots$

Bounded quantification: $\forall a \in A. P$

86



Preview: sets

Type: $'a\ set$

Operations: $a \in A, A \cup B, \dots$

Bounded quantification: $\forall a \in A. P$

Proof method *auto* knows (a little) about sets.

86



The (binary) tree library

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

87



The (binary) tree library

`~~/src/HOL/Library/Tree.thy`
datatype $'a\ tree = Leaf \mid Node\ ('a\ tree)\ 'a\ ('a\ tree)$

87



The (binary) tree library

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

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

Abbreviations:

$\langle \rangle \equiv \text{Leaf}$

87



The (binary) tree library

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

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

Abbreviations:

$\langle \rangle \equiv \text{Leaf}$
 $\langle l, a, r \rangle \equiv \text{Node } l \ a \ r$

87



The (binary) tree library

Size = number of nodes:

$\text{size} :: 'a \ \text{tree} \Rightarrow \text{nat}$

88



The (binary) tree library

Size = number of nodes:

$\text{size} :: 'a \ \text{tree} \Rightarrow \text{nat}$

$\text{size } \langle \rangle = 0$

$\text{size } \langle l, _, r \rangle = \text{size } l + \text{size } r + 1$

88



The (binary) tree library

Size = number of nodes:

$size :: 'a\ tree \Rightarrow nat$

$size \langle \rangle = 0$

$size \langle l, _, r \rangle = size\ l + size\ r + 1$

Inorder listing:

$inorder :: 'a\ tree \Rightarrow 'a\ list$

88



The (binary) tree library

Size = number of nodes:

$size :: 'a\ tree \Rightarrow nat$

$size \langle \rangle = 0$

$size \langle l, _, r \rangle = size\ l + size\ r + 1$

Inorder listing:

$inorder :: 'a\ tree \Rightarrow 'a\ list$

$inorder \langle \rangle = []$

$inorder \langle l, x, r \rangle = inorder\ l @ [x] @ inorder\ r$

88



The (binary) tree library

The set of elements in a tree:

$set_tree :: 'a\ tree \Rightarrow 'a\ set$

89



The (binary) tree library

The set of elements in a tree:

$set_tree :: 'a\ tree \Rightarrow 'a\ set$

$set_tree \langle \rangle = \{\}$

$set_tree \langle l, a, r \rangle = set_tree\ l \cup \{a\} \cup set_tree\ r$

89



The (binary) tree library

The set of elements in a tree:

$set_tree :: 'a\ tree \Rightarrow 'a\ set$

$set_tree\ \langle \rangle = \{\}$

$set_tree\ \langle l, a, r \rangle = set_tree\ l \cup \{a\} \cup set_tree\ r$

Applying a function to all elements a tree:

$map_tree :: ('a \Rightarrow 'b) \Rightarrow 'a\ tree \Rightarrow 'b\ tree$

89



The (binary) tree library

The set of elements in a tree:

$set_tree :: 'a\ tree \Rightarrow 'a\ set$

$set_tree\ \langle \rangle = \{\}$

$set_tree\ \langle l, a, r \rangle = set_tree\ l \cup \{a\} \cup set_tree\ r$

Applying a function to all elements a tree:

$map_tree :: ('a \Rightarrow 'b) \Rightarrow 'a\ tree \Rightarrow 'b\ tree$

$map_tree\ f\ \langle \rangle = \langle \rangle$

$map_tree\ f\ \langle l, a, r \rangle = \langle map_tree\ f\ l, f\ a, map_tree\ f\ r \rangle$

89



The (binary) tree library

Binary search tree invariant:

$bst :: 'a\ tree \Rightarrow bool$

90



The (binary) tree library

Binary search tree invariant:

$bst :: 'a\ tree \Rightarrow bool$

$bst\ \langle \rangle = True$

$bst\ \langle l, a, r \rangle =$

$(bst\ l \wedge$

$bst\ r \wedge$

$(\forall x \in set_tree\ l. x < a) \wedge (\forall x \in set_tree\ r. a < x))$

90



The (binary) tree library

Binary search tree invariant:

$bst :: 'a \text{ tree} \Rightarrow \text{bool}$

$bst \langle \rangle = \text{True}$

$bst \langle l, a, r \rangle =$

$(bst \ l \wedge$

$bst \ r \wedge$

$(\forall x \in \text{set_tree } l. x < a) \wedge (\forall x \in \text{set_tree } r. a < x))$

For any type $'a$?

90



Isabelle's type classes

91



Isabelle's type classes

A *type class* is defined by

- a set of required functions (the interface)

91



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

91



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 \leq , $<$



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 \leq , $<$

A type belongs to some class if

- the interface functions are defined on that type



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 \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 \leq , $<$

A type belongs to some class if

- the interface functions are defined on that type
- and satisfy the axioms of the class (proof needed!)



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 \leq , $<$

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 τ belongs to class C

91



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 \leq , $<$

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 τ belongs to class C

Example: $bst :: ('a :: linorder) tree \Rightarrow bool$

91



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 \leq , $<$

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 τ belongs to class C

Example: $bst :: ('a :: linorder) tree \Rightarrow bool$
 $\Rightarrow 'a$ must be a linear order!

91



Case study

BST_Demo.thy

92



Case study

