Script generated by TTT

Title: FDS (09.06.2017)

Date: Fri Jun 09 08:34:14 CEST 2017

Duration: 81:16 min

Pages: 93

Principle: Count function calls

For every function $f :: \tau_1 \Rightarrow ... \Rightarrow \tau_n \Rightarrow \tau$

define a *timing function* $t\_f :: \tau_1 \Rightarrow ... \Rightarrow \tau_n \Rightarrow nat$:

Principle: Count function calls

For every function $f :: \tau_1 \Rightarrow ... \Rightarrow \tau_n \Rightarrow \tau$

define a *timing function* $t\_f :: \tau_1 \Rightarrow ... \Rightarrow \tau_n \Rightarrow nat$:

Translation of defining equations:

$$\frac{e \rightsquigarrow e'}{f\ p_1 \ldots p_n = e \ \rightsquigarrow\ t\_f\ p_1 \ldots p_n = e' + 1}$$

Principle: Count function calls

For every function $f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$
define a *timing function* $t\_f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow nat$:
Translation of defining equations:

$$\frac{e \rightsquigarrow e'}{f\ p_1 \ldots p_n = e \ \rightsquigarrow\ t\_f\ p_1 \ldots p_n = e' + 1}$$

Translation of expressions:

$$\frac{s_1 \rightsquigarrow t_1 \quad \ldots \quad s_k \rightsquigarrow t_k}{g\ s_1 \ldots s_k \rightsquigarrow t_1 + \cdots + t_k + t\_g\ s_1 \ldots s_k}$$

Principle: Count function calls

For every function $f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$
define a *timing function* $t\_f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow nat$:
Translation of defining equations:

$$\frac{e \rightsquigarrow e'}{f\ p_1 \ldots p_n = e \ \rightsquigarrow\ t\_f\ p_1 \ldots p_n = e' + 1}$$

Translation of expressions:

$$\frac{s_1 \rightsquigarrow t_1 \quad \ldots \quad s_k \rightsquigarrow t_k}{g\ s_1 \ldots s_k \rightsquigarrow t_1 + \cdots + t_k + t\_g\ s_1 \ldots s_k}$$

- Variable $\rightsquigarrow$ 0, Constant $\rightsquigarrow$ 0

Principle: Count function calls

For every function $f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow \tau$
define a *timing function* $t\_f :: \tau_1 \Rightarrow \ldots \Rightarrow \tau_n \Rightarrow nat$:
Translation of defining equations:

$$\frac{e \rightsquigarrow e'}{f\ p_1 \ldots p_n = e \ \rightsquigarrow\ t\_f\ p_1 \ldots p_n = e' + 1}$$

Translation of expressions:

$$\frac{s_1 \rightsquigarrow t_1 \quad \ldots \quad s_k \rightsquigarrow t_k}{g\ s_1 \ldots s_k \rightsquigarrow t_1 + \cdots + t_k + t\_g\ s_1 \ldots s_k}$$

- Variable $\rightsquigarrow$ 0, Constant $\rightsquigarrow$ 0
- Constructor calls and primitive operations on $bool$ and numbers cost 1

Example

$app\ []\ ys = ys$

Example

$app\ []\ ys = ys$

$\leadsto$

$t\_app\ []\ ys = 0 + 1$

Example

$app\ []\ ys = ys$

$\leadsto$

$t\_app\ []\ ys = 0 + 1$

$app\ (x\#xs)\ ys = x\ \#\ app\ xs\ ys$

Example

$app\ []\ ys = ys$

$\leadsto$

$t\_app\ []\ ys = 0 + 1$

$app\ (x\#xs)\ ys = x\ \#\ app\ xs\ ys$

$\leadsto$

$t\_app\ (x\#xs)\ ys = 0 + (0 + 0 + t\_app\ xs\ ys) + 1 + 1$

A compact formulation of $e \leadsto t$

t is the sum of all $t\_g\ s_1\ \ldots\ s_k$
such that $g\ s_1\ \ldots\ s_n$ is a subterm of e

A compact formulation of $e \rightsquigarrow t$

t is the sum of all $t\_g\ s_1\ \ldots\ s_k$
such that $g\ s_1\ \ldots\ s_n$ is a subterm of e

If g is

- a constructor or
- a predefined function on $bool$ or numbers

then $t\_g\ \ldots = 1$.

if and *case*

So far we model a call-by-value semantics

if and *case*

So far we model a call-by-value semantics

Conditionals and case expressions are evaluated lazily.

if and *case*

So far we model a call-by-value semantics

Conditionals and case expressions are evaluated lazily.
Translation:

$$\frac{b \rightsquigarrow t \quad s_1 \rightsquigarrow t_1 \quad s_2 \rightsquigarrow t_2}{\textit{if } b \textit{ then } s_1 \textit{ else } s_2 \rightsquigarrow t + (\textit{if } b \textit{ then } t_1 \textit{ else } t_2)}$$

A compact formulation of $e \rightsquigarrow t$

t is the sum of all $t\_g\ s_1\ \dots\ s_k$
such that $g\ s_1\ \dots\ s_n$ is a subterm of e

If g is

- a constructor or
- a predefined function on $bool$ or numbers

then $t\_g\ \dots = 1$.

if and *case*

So far we model a call-by-value semantics

Conditionals and case expressions are evaluated lazily.
Translation:

$$\frac{b \rightsquigarrow t \quad s_1 \rightsquigarrow t_1 \quad s_2 \rightsquigarrow t_2}{\textit{if } b \textit{ then } s_1 \textit{ else } s_2 \rightsquigarrow t + (\textit{if } b \textit{ then } t_1 \textit{ else } t_2)}$$

$O(.)$ is enough

$O(.)$ is enough

$\Longrightarrow$ Reduce all additive constants to 1

Example

$t\_app\ (x\#xs)\ ys = t\_app\ xs\ ys + 1$

Discussion

- The definition of t_f from f can be automated.

Discussion

- The definition of t_f from f can be automated.
- The correctness of t_f could be proved w.r.t. a semantics that counts computation steps.

Discussion

- The definition of t_f from f can be automated.
- The correctness of t_f could be proved w.r.t. a semantics that counts computation steps.
- Precise complexity bounds (as opposed to $O(.)$) would require a formal model of (at least) the compiler and the hardware.

`Thys/Sorting.thy`

Insertion sort complexity

$merge :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$

$merge :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$

$merge\ []\ ys = ys$
$merge\ xs\ []\ = xs$
$merge\ (x\ \#\ xs)\ (y\ \#\ ys) =$
(if $x \leq y$ then $x\ \#\ merge\ xs\ (y\ \#\ ys)$
else $y\ \#\ merge\ (x\ \#\ xs)\ ys$)

$merge :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$

$merge\ []\ ys = ys$
$merge\ xs\ []\ = xs$
$merge\ (x\ \#\ xs)\ (y\ \#\ ys) =$
(if $x \leq y$ then $x\ \#\ merge\ xs\ (y\ \#\ ys)$
else $y\ \#\ merge\ (x\ \#\ xs)\ ys$)

$msort :: 'a\ list \Rightarrow 'a\ list$

$msort\ xs =$
(let $n = length\ xs$
in if $n \leq 1$ then xs
else $merge\ (msort\ (take\ (n\ div\ 2)\ xs))$
$(msort\ (drop\ (n\ div\ 2)\ xs))$)

Thys/Sorting.thy

Merge sort

Chapter 7

Binary Trees

`HOL/Library/Tree.thy`
`Thys/Tree_Additions.thy`

Binary trees

datatype $'a\ tree = Leaf \mid Node\ ('a\ tree)\ 'a\ ('a\ tree)$

Tree traversal

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

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

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

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

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

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

Size

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

$|\langle\rangle| = 0$
$|\langle l,\ \_,\ r\rangle| = |l| + |r| + 1$

Size

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

$|\langle\rangle| = 0$
$|\langle l,\ \_,\ r\rangle| = |l| + |r| + 1$

29

Size

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

$|\langle\rangle| = 0$
$|\langle l,\ \_,\ r\rangle| = |l| + |r| + 1$

$size1 :: {}'a\ tree \Rightarrow nat$
$|t|_1 = |t| + 1$

Size

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

$|\langle\rangle| = 0$
$|\langle l,\ \_,\ r\rangle| = |l| + |r| + 1$

$size1 :: {}'a\ tree \Rightarrow nat$
$|t|_1 = |t| + 1$

$\Longrightarrow$
$|\langle\rangle|_1 = 1$
$|\langle l,\ x,\ r\rangle|_1 = |l|_1 + |r|_1$

Height

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

$h(\langle\rangle) = 0$
$h(\langle l, \_, r\rangle) = max\ (h(l))\ (h(r)) + 1$

Height

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

$h(\langle\rangle) = 0$
$h(\langle l, \_, r\rangle) = max\ (h(l))\ (h(r)) + 1$

Warning: $h(.)$ only on slides

Height

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

$h(\langle\rangle) = 0$
$h(\langle l, \_, r\rangle) = max\ (h(l))\ (h(r)) + 1$

Warning: $h(.)$ only on slides

Lemma $h(t) \leq |t|$

Lemma $|t|_1 \leq 2^{h(t)}$

Minimal height

$min\_height :: 'a\ tree \Rightarrow nat$

Minimal height

$min\_height :: 'a\ tree \Rightarrow nat$

$mh(\langle\rangle) = 0$
$mh(\langle l,\ \_,\ r\rangle) = min\ (mh(l))\ (mh(r)) + 1$

Minimal height

$min\_height :: 'a\ tree \Rightarrow nat$

$mh(\langle\rangle) = 0$
$mh(\langle l,\ \_,\ r\rangle) = min\ (mh(l))\ (mh(r)) + 1$

Warning: $mh(.)$ only on slides

Internal path length

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

$ipl\ \langle\rangle = 0$
$ipl\ \langle l,\ \_,\ r\rangle = ipl\ l + |l| + ipl\ r + |r|$

Internal path length

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

$ipl\ \langle\rangle = 0$
$ipl\ \langle l,\ \_,\ r\rangle = ipl\ l + |l| + ipl\ r + |r|$

Why relevant?

Complete tree

$complete :: {}'a\ tree \Rightarrow bool$

Complete tree

$complete :: {}'a\ tree \Rightarrow bool$

$complete\ \langle\rangle = True$
$complete\ \langle l, \_, r\rangle =$
$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Complete tree

$complete :: {}'a\ tree \Rightarrow bool$
$complete\ \langle\rangle = True$
$complete\ \langle l, \_, r\rangle =$
$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Complete tree

$complete :: {'a}\ tree \Rightarrow bool$

$complete\ \langle\rangle\ =\ True$

$complete\ \langle l,\ \_,\ r\rangle\ =$

$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Complete tree

$complete :: {'a}\ tree \Rightarrow bool$

$complete\ \langle\rangle\ =\ True$

$complete\ \langle l,\ \_,\ r\rangle\ =$

$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Lemma $|t|_1 = 2^{h(t)} \Longrightarrow complete\ t$

Complete tree

$complete :: {'a}\ tree \Rightarrow bool$

$complete\ \langle\rangle\ =\ True$

$complete\ \langle l,\ \_,\ r\rangle\ =$

$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Lemma $|t|_1 = 2^{h(t)} \Longrightarrow complete\ t$

Lemma $|t|_1 = 2^{mh(t)} \Longrightarrow complete\ t$

Corollary $\neg\ complete\ t \Longrightarrow |t|_1 < 2^{h(t)}$

Complete tree

$complete :: {'a}\ tree \Rightarrow bool$

$complete\ \langle\rangle\ =\ True$

$complete\ \langle l,\ \_,\ r\rangle\ =$

$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Lemma $|t|_1 = 2^{h(t)} \Longrightarrow complete\ t$

Lemma $|t|_1 = 2^{mh(t)} \Longrightarrow complete\ t$

Corollary $\neg\ complete\ t \Longrightarrow |t|_1 < 2^{h(t)}$

Complete tree

$complete :: 'a\ tree \Rightarrow bool$
$complete\ \langle\rangle = True$
$complete\ \langle l,\ \_,\ r\rangle =$
$(complete\ l \wedge complete\ r \wedge h(l) = h(r))$

Lemma $complete\ t = (mh(t) = h(t))$

Lemma $complete\ t \Longrightarrow |t|_1 = 2^{h(t)}$

Lemma $|t|_1 = 2^{h(t)} \Longrightarrow complete\ t$
Lemma $|t|_1 = 2^{mh(t)} \Longrightarrow complete\ t$

Corollary $\neg\ complete\ t \Longrightarrow |t|_1 < 2^{h(t)}$
Corollary $\neg\ complete\ t \Longrightarrow 2^{mh(t)} < |t|_1$

Complete tree: ipl

Lemma A complete tree of height h has internal path length $(h - 2) * 2^h + 2$.

Complete tree: ipl

Lemma A complete tree of height h has internal path length $(h - 2) * 2^h + 2$.

In a search tree, finding the node labelled x takes as many steps as the path from the root to x is long. Thus the average time to find an element that is in the tree is $ipl\ t\ /\ |t|$.

Complete tree: ipl

Lemma A complete tree of height h has internal path length $(h - 2) * 2^h + 2$.

In a search tree, finding the node labelled x takes as many steps as the path from the root to x is long. Thus the average time to find an element that is in the tree is $ipl\ t\ /\ |t|$.

Lemma Let t be a complete search tree of height h.

Complete tree: *ipl*

Lemma A complete tree of height h has internal path length $(h-2)*2^h+2$.

In a search tree, finding the node labelled x takes as many steps as the path from the root to x is long. Thus the average time to find an element that is in the tree is $ipl\ t\ /\ |t|$.

Lemma Let t be a complete search tree of height h. The average time to find a random element that is in the tree is asymptotically $h-2$ (as h approaches ∞):

Complete tree: *ipl*

Lemma A complete tree of height h has internal path length $(h-2)*2^h+2$.

In a search tree, finding the node labelled x takes as many steps as the path from the root to x is long. Thus the average time to find an element that is in the tree is $ipl\ t\ /\ |t|$.

Lemma Let t be a complete search tree of height h. The average time to find a random element that is in the tree is asymptotically $h-2$ (as h approaches ∞):

$$ipl\ t\ /\ |t| \sim h-2$$

Complete tree: *ipl*

A problem: $(h-2)*2^h+2$ is only correct if interpreted over type int, not nat.

Complete tree: *ipl*

A problem: $(h-2)*2^h+2$ is only correct if interpreted over type int, not nat.

Correct version:
Lemma $complete\ t \Longrightarrow$
$int\ (ipl\ t) = (int\ (h(t)) - 2) * 2^{h(t)} + 2$

Balanced tree

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

Balanced tree

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

$balanced\ t = (h(t) - mh(t) \leq 1)$

Balanced tree

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

$balanced\ t = (h(t) - mh(t) \leq 1)$

Balanced trees have optimal height:
Lemma *If* $balanced\ t \wedge |t| \leq |t'|$ *then* $h(t) \leq h(t')$.

Warning

- The terms *complete* and *balanced* are not defined uniquely in the literature.

Chapter 8

Search Trees

Most of the material focuses on
BSTs = binary search trees

BSTs represent sets

Any tree represents a set:

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

$set\_tree\ \langle\rangle = \{\}$
$set\_tree\ \langle l,\ x,\ r\rangle = set\_tree\ l \cup \{x\} \cup set\_tree\ r$

BSTs represent sets

Any tree represents a set:

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

$set\_tree\ \langle\rangle = \{\}$
$set\_tree\ \langle l,\ x,\ r\rangle = set\_tree\ l \cup \{x\} \cup set\_tree\ r$

A BST represents a set that can be searched in time $O(h(t))$

BSTs represent sets

Any tree represents a set:

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

$set\_tree\ \langle\rangle = \{\}$
$set\_tree\ \langle l,\ x,\ r\rangle = set\_tree\ l \cup \{x\} \cup set\_tree\ r$

A BST represents a set that can be searched in time $O(h(t))$

Function set_tree is called an *abstraction function*
because it maps the implementation
to the abstract mathematical object

bst

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

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge bst\ r \wedge$
$(\forall x \in set\_tree\ l.\ x < a) \wedge$
$(\forall x \in set\_tree\ r.\ a < x))$

bst

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

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge bst\ r \wedge$
$(\forall x \in set\_tree\ l.\ x < a) \wedge$
$(\forall x \in set\_tree\ r.\ a < x))$

Type $'a$ must be in class $linorder$ ($'a :: linorder$) where $linorder$ are *linear orders* (also called *total orders*).

bst

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

$bst\ \langle\rangle = True$
$bst\ \langle l,\ a,\ r\rangle =$
$(bst\ l \wedge bst\ r \wedge$
$(\forall x \in set\_tree\ l.\ x < a) \wedge$
$(\forall x \in set\_tree\ r.\ a < x))$

Type $'a$ must be in class $linorder$ ($'a :: linorder$) where $linorder$ are *linear orders* (also called *total orders*).

Note: nat, int and $real$ are in class $linorder$

Set interface

An implementation of sets of elements of type $'a$ must provide

Set interface

An implementation of sets of elements of type $'a$ must provide

- An implementation type $'s$

Set interface

An implementation of sets of elements of type $'a$ must provide

- An implementation type $'s$
- $empty :: 's$
- $insert :: 'a \Rightarrow 's \Rightarrow 's$
- $delete :: 'a \Rightarrow 's \Rightarrow 's$
- $isin :: \quad 's \Rightarrow 'a \Rightarrow bool$

Map interface

Instead of a set, a search tree can also implement a map from $'a$ to $'b$:

Map interface

Instead of a set, a search tree can also implement a map from $'a$ to $'b$:

- An implementation type $'m$
- $empty :: 'm$
- $update :: 'a \Rightarrow 'b \Rightarrow 'm \Rightarrow 'm$

Map interface

Instead of a set, a search tree can also implement a map from $'a$ to $'b$:

- An implementation type $'m$
- $empty :: 'm$
- $update :: 'a \Rightarrow 'b \Rightarrow 'm \Rightarrow 'm$
- $delete :: 'a \Rightarrow 'm \Rightarrow 'm$
- $lookup :: 'm \Rightarrow 'a \Rightarrow 'b\ option$

Map interface

Instead of a set, a search tree can also implement a map from $'a$ to $'b$:

- An implementation type $'m$
- $empty :: 'm$
- $update :: 'a \Rightarrow 'b \Rightarrow 'm \Rightarrow 'm$
- $delete :: 'a \Rightarrow 'm \Rightarrow 'm$
- $lookup :: 'm \Rightarrow 'a \Rightarrow 'b\ option$

Sets are a special case of maps

Comparison of elements

We assume that the element type $'a$ is a linear order

Instead of using $<$ and $\leq$ directly:

datatype $cmp\_val = LT \mid EQ \mid GT$

$cmp\ x\ y =$
(if $x < y$ then LT else if $x = y$ then EQ else GT)

Implementation

Implementation type: $'a\ tree$

$empty = Leaf$

$$
\begin{aligned}
&insert\ x\ \langle\rangle = \langle\langle\rangle,\ x,\ \langle\rangle\rangle\\
&insert\ x\ \langle l,\ a,\ r\rangle = (\text{case } cmp\ x\ a \text{ of}\\
&\qquad\qquad LT \Rightarrow \langle insert\ x\ l,\ a,\ r\rangle\\
&\qquad\quad\ \mid EQ \Rightarrow \langle l,\ a,\ r\rangle\\
&\qquad\quad\ \mid GT \Rightarrow \langle l,\ a,\ insert\ x\ r\rangle)
\end{aligned}
$$

Implementation

$delete\ x\ \langle\rangle = \langle\rangle$

Implementation

$$
\begin{aligned}
&delete\ x\ \langle\rangle = \langle\rangle\\
&delete\ x\ \langle l,\ a,\ r\rangle =\\
&(\text{case } cmp\ x\ a \text{ of}\\
&\quad LT \Rightarrow \langle delete\ x\ l,\ a,\ r\rangle\\
&\mid EQ \Rightarrow \text{if } r = \langle\rangle \text{ then } l\\
&\qquad\qquad \text{else let } (a',\ r') = del\_min\ r \text{ in } \langle l,\ a',\ r'\rangle\\
&\mid GT \Rightarrow \langle l,\ a,\ delete\ x\ r\rangle)
\end{aligned}
$$

Implementation

$$
\begin{aligned}
&delete\ x\ \langle\rangle = \langle\rangle\\
&delete\ x\ \langle l,\ a,\ r\rangle =\\
&(\text{case } cmp\ x\ a \text{ of}\\
&\quad LT \Rightarrow \langle delete\ x\ l,\ a,\ r\rangle\\
&\mid EQ \Rightarrow \text{if } r = \langle\rangle \text{ then } l\\
&\qquad\qquad \text{else let } (a',\ r') = del\_min\ r \text{ in } \langle l,\ a',\ r'\rangle\\
&\mid GT \Rightarrow \langle l,\ a,\ delete\ x\ r\rangle)
\end{aligned}
$$

$$
\begin{aligned}
&del\_min\ \langle l,\ a,\ r\rangle =\\
&(\text{if } l = \langle\rangle \text{ then } (a,\ r)\\
&\ \text{else let } (x,\ l') = del\_min\ l \text{ in } (x,\ \langle l',\ a,\ r\rangle))
\end{aligned}
$$

8 Unbalanced BST

Correctness

Correctness Proof Method Based on Sorted Lists

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup\ \{.\}$	$-\ \{.\}$	$\in$

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup\ \{.\}$	$-\ \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup\ \{.\}$	$-\ \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$

$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup \{.\}$	$- \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$
$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$
$$set\_tree\ (delete\ x\ t) = set\_tree\ t - \{x\}$$

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup \{.\}$	$- \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$
$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$
$$set\_tree\ (delete\ x\ t) = set\_tree\ t - \{x\}$$
$$isin\ t\ x = (x \in set\_tree\ t)$$

Why is this implementation correct?

Because	*empty*	*insert*	*delete*	*isin*
simulate	$\{\}$	$\cup \{.\}$	$- \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$
$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$
$$set\_tree\ (delete\ x\ t) = set\_tree\ t - \{x\}$$
$$isin\ t\ x = (x \in set\_tree\ t)$$

Under the assumption $bst\ t$

Also: bst must be invariant

$$bst\ empty$$
$$bst\ t \Longrightarrow bst\ (insert\ x\ t)$$
$$bst\ t \Longrightarrow bst\ (delete\ x\ t)$$

Also: bst must be invariant

$$bst\ empty$$
$$bst\ t \Longrightarrow bst\ (insert\ x\ t)$$
$$bst\ t \Longrightarrow bst\ (delete\ x\ t)$$

Why is this implementation correct?

Because	$empty$	$insert$	$delete$	$isin$
simulate	$\{\}$	$\cup\ \{.\}$	$-\ \{.\}$	$\in$

$$set\_tree\ empty = \{\}$$
$$set\_tree\ (insert\ x\ t) = set\_tree\ t \cup \{x\}$$
$$set\_tree\ (delete\ x\ t) = set\_tree\ t - \{x\}$$
$$isin\ t\ x = (x \in set\_tree\ t)$$

Under the assumption