Script **generated by TTT**

Title: FDS (01.06.2018)

Date: Fri Jun 01 08:32:59 CEST 2018

Duration: 87:46 min

Pages: 93

TTT 100% Fri 08:33

In Isabelle: **locale**

locale $Set =$
fixes $empty :: {'s}$
fixes $insert :: {'a} \Rightarrow {'s} \Rightarrow {'s}$
fixes $isin :: {'s} \Rightarrow {'a} \Rightarrow bool$
fixes $set :: {'s} \Rightarrow {'a}\ set$
fixes $invar :: {'s} \Rightarrow bool$
assumes $set\ empty = \{\}$
assumes $invar\ s \Longrightarrow isin\ s\ x = (x \in set\ s)$
assumes $invar\ s \Longrightarrow set(insert\ x\ s) = set\ s \cup \{x\}$
assumes $invar\ empty$
assumes $invar\ s \Longrightarrow invar(insert\ x\ s)$

See `HOL/Data_Structures/Set_by_Ordered.thy`

Formally, in general

To ease notation, generalize α and $invar$:
α is the identity and $invar$ is $True$
on types other than T

Specification of each interface function f (on T):

- f must behave like some function f_A (on A):
  $invar\ t_1 \wedge \ldots \wedge invar\ t_n \Longrightarrow$
  $\alpha(f\ t_1\ \ldots\ t_n) = f_A\ (\alpha\ t_1)\ \ldots\ (\alpha\ t_n)$

The purpose of an ADT is to provide a context for implementing generic algorithms parameterized with the interface functions of the ADT.

Example

locale $Set =$
fixes ...
assumes ...
begin

fun set_of_list **where**
$set\_of\_list\ [] = empty\ |$
$set\_of\_list\ (x\ \#\ xs) = insert\ x\ (set\_of\_list\ xs)$

lemma $invar(set\_of\_list\ xs)$
by$(induction\ xs)$
$(auto\ simp\!: invar\_empty\ invar\_insert)$

end

1. Implement interface
2. Prove specification

Example

Define functions *isin* and *insert* on type *'a tree* with invariant *bst*.

In Isabelle: **interpretation**

In Isabelle: **interpretation**

interpretation *Set*
where $empty = Leaf$ **and** $isin = isin$
and $insert = insert$ **and** $set = set\_tree$ **and** $invar = bst$

1. Implement interface
2. Prove specification

Example

Define functions *isin* and *insert* on type $'a\ tree$ with invariant bst.

Now implement locale *Set*:

Formally, in general

To ease notation, generalize α and *invar*:
α is the identity and *invar* is *True*
on types other than T

Specification of each interface function f (on T):

- f must behave like some function f_A (on A):

In Isabelle: **interpretation**

interpretation *Set*
where $empty = Leaf$ **and** $isin = isin$
and $insert = insert$ **and** $set = set\_tree$ **and** $invar = bst$
proof

In Isabelle: **interpretation**

interpretation *Set*
where $empty = Leaf$ **and** $isin = isin$
and $insert = insert$ **and** $set = set\_tree$ **and** $invar = bst$
proof
  show $set\_tree\ empty = \{\}$ $\langle proof \rangle$
next
  fix s **assume** $bst\ s$
  show $set\_tree\ (insert\_tree\ x\ s) = set\_tree\ s \cup \{x\}$
  $\langle proof \rangle$

In Isabelle: **interpretation**

interpretation *Set*
where $empty = Leaf$ **and** $isin = isin$
and $insert = insert$ **and** $set = set\_tree$ **and** $invar = bst$
proof

Formally, in general

To ease notation, generalize α and $invar$:
α is the identity and $invar$ is $True$
on types other than T

In Isabelle: **interpretation**

interpretation *Set*
where *empty* = *Leaf* **and** *isin* = *isin*
and *insert* = *insert* **and** *set* = *set_tree* **and** *invar* = *bst*
proof
  show *set_tree empty* = {} ⟨*proof*⟩
next
  fix *s* **assume** *bst s*
  show *set_tree* (*insert_tree x s*) = *set_tree s* ∪ {*x*}
  ⟨*proof*⟩
next
⋮
qed

In Isabelle: **interpretation**

interpretation *Set*
where *empty* = *Leaf* **and** *isin* = *isin*
and *insert* = *insert* **and** *set* = *set_tree* **and** *invar* = *bst*
proof
  show *set_tree empty* = {} ⟨*proof*⟩
next
  fix *s* **assume** *bst s*
  show *set_tree* (*insert_tree x s*) = *set_tree s* ∪ {*x*}
  ⟨*proof*⟩
next
⋮
qed

In Isabelle: **interpretation**

interpretation *Set*
where *empty* = *Leaf* **and** *isin* = *isin*
and *insert* = *insert* **and** *set* = *set_tree* **and** *invar* = *bst*
proof
  show *set_tree empty* = {} ⟨*proof*⟩
next

2-3 Trees

datatype $'a\ tree23 = \langle\rangle$
$\mid Node2\ ('a\ tree23)\ 'a\ ('a\ tree23)$
$\mid Node3\ ('a\ tree23)\ 'a\ ('a\ tree23)\ 'a\ ('a\ tree23)$

2-3 Trees

datatype $'a\ tree23 = \langle\rangle$
$\mid Node2\ ('a\ tree23)\ 'a\ ('a\ tree23)$
$\mid Node3\ ('a\ tree23)\ 'a\ ('a\ tree23)\ 'a\ ('a\ tree23)$

Abbreviations:

$$\begin{aligned} \langle l,\ a,\ r\rangle &\equiv Node2\ l\ a\ r \\ \langle l,\ a,\ m,\ b,\ r\rangle &\equiv Node3\ l\ a\ m\ b\ r \end{aligned}$$

$isin$

$isin\ \langle l,\ a,\ m,\ b,\ r\rangle\ x =$
(case $cmp\ x\ a$ of
  $LT \Rightarrow isin\ l\ x$
$\mid EQ \Rightarrow True$
$\mid GT \Rightarrow$ case $cmp\ x\ b$ of
    $LT \Rightarrow isin\ m\ x$
    $\mid EQ \Rightarrow True$
    $\mid GT \Rightarrow isin\ r\ x)$

$isin$

$isin\ \langle l,\ a,\ m,\ b,\ r\rangle\ x =$
(case $cmp\ x\ a$ of
  $LT \Rightarrow isin\ l\ x$
$\mid EQ \Rightarrow True$
$\mid GT \Rightarrow$ case $cmp\ x\ b$ of
    $LT \Rightarrow isin\ m\ x$
    $\mid EQ \Rightarrow True$
    $\mid GT \Rightarrow isin\ r\ x)$

Assumes the usual ordering invariant

Structural invariant *bal*

All leaves are at the same level:

Structural invariant *bal*

All leaves are at the same level:

$bal \ \langle\rangle = True$

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

$bal \ \langle l, \_, m, \_, r\rangle =$
$(bal \ l \wedge bal \ m \wedge bal \ r \wedge h(l) = h(m) \wedge h(m) = h(r))$

Structural invariant *bal*

All leaves are at the same level:

$bal \ \langle\rangle = True$

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

$bal \ \langle l, \_, m, \_, r\rangle =$
$(bal \ l \wedge bal \ m \wedge bal \ r \wedge h(l) = h(m) \wedge h(m) = h(r))$

Lemma

$bal \ t \Longrightarrow 2^{h(t)} \leq |t| + 1$

Insertion

The idea:

$Leaf \rightsquigarrow Node2$
$Node2 \rightsquigarrow Node3$
$Node3 \rightsquigarrow$ overflow, pass 1 element back up

Insertion

Two possible return values:

Insertion

Two possible return values:

- tree accommodates new element without increasing height: $T_i\ t$

Insertion

Two possible return values:

- tree accommodates new element without increasing height: $T_i\ t$
- tree overflows: $Up_i\ l\ x\ r$

Insertion

Two possible return values:

- tree accommodates new element without increasing height: $T_i\ t$
- tree overflows: $Up_i\ l\ x\ r$

datatype $'a\ up_i = T_i\ ('a\ tree23)$
$\mid Up_i\ ('a\ tree23)\ 'a\ ('a\ tree23)$

$tree_i :: 'a\ up_i \Rightarrow 'a\ tree23$

Insertion

Two possible return values:

- tree accommodates new element without increasing height: $T_i\ t$
- tree overflows: $Up_i\ l\ x\ r$

datatype $'a\ up_i = T_i\ ('a\ tree23)$
$\mid Up_i\ ('a\ tree23)\ 'a\ ('a\ tree23)$

$tree_i :: 'a\ up_i \Rightarrow 'a\ tree23$
$tree_i\ (T_i\ t) = t$
$tree_i\ (Up_i\ l\ a\ r) = \langle l,\ a,\ r\rangle$

Insertion

$insert :: 'a \Rightarrow 'a\ tree23 \Rightarrow 'a\ tree23$
$insert\ x\ t = tree_i\ (ins\ x\ t)$

Insertion

$insert :: 'a \Rightarrow 'a\ tree23 \Rightarrow 'a\ tree23$
$insert\ x\ t = tree_i\ (ins\ x\ t)$

$ins :: 'a \Rightarrow 'a\ tree23 \Rightarrow 'a\ up_i$

Insertion

$insert :: 'a \Rightarrow 'a\ tree23 \Rightarrow 'a\ tree23$

Insertion

$$ins\ x\ \langle\rangle\ =\ Up_i\ \langle\rangle\ x\ \langle\rangle$$

$$ins\ x\ \langle l,\ a,\ r\rangle\ =$$

Insertion

$$ins\ x\ \langle l,\ a,\ m,\ b,\ r\rangle\ =$$

Insertion

$$\begin{array}{l}
ins\ x\ \langle l,\ a,\ m,\ b,\ r\rangle\ = \\
\text{case}\ cmp\ x\ a\ \text{of} \\
\quad LT \Rightarrow \text{case}\ ins\ x\ l\ \text{of} \\
\qquad\qquad T_i\ l' \Rightarrow T_i\ \langle l',\ a,\ m,\ b,\ r\rangle \\
\qquad\quad\ \mid Up_i\ l_1\ c\ l_2 \Rightarrow Up_i\ \langle l_1,\ c,\ l_2\rangle\ a\ \langle m,\ b,\ r\rangle \\
\mid EQ \Rightarrow T_i\ \langle l,\ a,\ m,\ b,\ r\rangle \\
\mid GT \Rightarrow \\
\qquad \text{case}\ cmp\ x\ b\ \text{of} \\
\qquad\quad LT \Rightarrow \\
\qquad\qquad \text{case}\ ins\ x\ m\ \text{of} \\
\qquad\qquad\quad T_i\ m' \Rightarrow T_i\ \langle l,\ a,\ m',\ b,\ r\rangle \\
\qquad\qquad \mid Up_i\ m_1\ c\ m_2 \Rightarrow Up_i\ \langle l,\ a,\ m_1\rangle\ c\ \langle m_2,\ b,\ r\rangle \\
\qquad \mid EQ \Rightarrow T_i\ \langle l,\ a,\ m,\ b,\ r\rangle \\
\qquad \mid GT \Rightarrow
\end{array}$$

Insertion preserves *bal*

Lemma

$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t))$

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t))$

Proof by induction on t.

Insertion

$ins\ x\ \langle\rangle = Up_i\ \langle\rangle\ x\ \langle\rangle$

$ins\ x\ \langle l,\ a,\ r\rangle =$
case $cmp\ x\ a$ of
  $LT \Rightarrow$ case $ins\ x\ l$ of
    $T_i\ l' \Rightarrow T_i\ \langle l',\ a,\ r\rangle$
    $|\ Up_i\ l_1\ b\ l_2 \Rightarrow T_i\ \langle l_1,\ b,\ l_2,\ a,\ r\rangle$
$|\ EQ \Rightarrow T_i\ \langle l,\ x,\ r\rangle$
$|\ GT \Rightarrow$ case $ins\ x\ r$ of
    $T_i\ r' \Rightarrow T_i\ \langle l,\ a,\ r'\rangle$
    $|\ Up_i\ r_1\ b\ r_2 \Rightarrow T_i\ \langle l,\ a,\ r_1,\ b,\ r_2\rangle$

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t))$

Proof by induction on t.

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t))$
where $h :: 'a\ up_i \Rightarrow nat$

Proof by induction on t.

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t))$
where $h :: 'a\ up_i \Rightarrow nat$
$h(T_i\ t) = h(t)$
$h(Up_i\ l\ a\ r) = h(l)$
Proof by induction on t.

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t)) \wedge h(ins\ a\ t) = h(t)$
where $h :: 'a\ up_i \Rightarrow nat$
$h(T_i\ t) = h(t)$
$h(Up_i\ l\ a\ r) = h(l)$
Proof by induction on t.

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t)) \wedge h(ins\ a\ t) = h(t)$
where $h :: 'a\ up_i \Rightarrow nat$
$h(T_i\ t) = h(t)$
$h(Up_i\ l\ a\ r) = h(l)$
Proof by induction on t. Base and step automatic.

Insertion preserves bal

Lemma
$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t)) \wedge h(ins\ a\ t) = h(t)$
where $h :: 'a\ up_i \Rightarrow nat$
$h(T_i\ t) = h(t)$
$h(Up_i\ l\ a\ r) = h(l)$
Proof by induction on t. Base and step automatic.

Corollary
$bal\ t \Longrightarrow bal\ (insert\ a\ t)$

Insertion preserves bal

Lemma

$bal\ t \Longrightarrow bal\ (tree_i\ (ins\ a\ t)) \wedge h(ins\ a\ t) = h(t)$

where $h :: 'a\ up_i \Rightarrow nat$

$h(T_i\ t) = h(t)$

$h(Up_i\ l\ a\ r) = h(l)$

Proof by induction on t.

Insertion

$ins\ x\ \langle\rangle = Up_i\ \langle\rangle\ x\ \langle\rangle$

$ins\ x\ \langle l,\ a,\ r\rangle =$

Deletion

The idea:

$Node3 \leadsto Node2$

$Node2 \leadsto$ underflow, height decreases by 1

Deletion

The idea:

$Node3 \leadsto Node2$

$Node2 \leadsto$ underflow, height decreases by 1

Underflow: merge with siblings on the way up

Deletion

Two possible return values:

Deletion

Two possible return values:

- height unchanged: $T_d\ t$
- height decreased by 1: $Up_d\ t$

Deletion

Two possible return values:

- height unchanged: $T_d\ t$
- height decreased by 1: $Up_d\ t$

datatype $'a\ up_d = T_d\ ('a\ tree23)\ |\ Up_d\ ('a\ tree23)$

Deletion

Two possible return values:

- height unchanged: $T_d\ t$
- height decreased by 1: $Up_d\ t$

datatype $'a\ up_d = T_d\ ('a\ tree23)\ |\ Up_d\ ('a\ tree23)$

$tree_d\ (T_d\ t) = t$
$tree_d\ (Up_d\ t) = t$

Deletion

$delete :: 'a \Rightarrow 'a\ tree23 \Rightarrow 'a\ tree23$

$delete\ x\ t = tree_d\ (del\ x\ t)$

Deletion

$del\ x\ \langle\rangle = T_d\ \langle\rangle$

$del\ x\ \langle\langle\rangle,\ a,\ \langle\rangle\rangle =$
$(\text{if } x = a \text{ then } Up_d\ \langle\rangle \text{ else } T_d\ \langle\langle\rangle,\ a,\ \langle\rangle\rangle)$

Deletion

$del\ x\ \langle\rangle = T_d\ \langle\rangle$

$del\ x\ \langle\langle\rangle,\ a,\ \langle\rangle\rangle =$
$(\text{if } x = a \text{ then } Up_d\ \langle\rangle \text{ else } T_d\ \langle\langle\rangle,\ a,\ \langle\rangle\rangle)$

$del\ x\ \langle\langle\rangle,\ a,\ \langle\rangle,\ b,\ \langle\rangle\rangle = \ldots$

$del\ x\ \langle l,\ a,\ r\rangle =$
$(\text{case } cmp\ x\ a \text{ of}$
$\quad LT \Rightarrow node21\ (del\ x\ l)\ a\ r$
$\mid EQ \Rightarrow \text{let } (a',\ t) = del\_min\ r \text{ in } node22\ l\ a'\ t$
$\mid GT \Rightarrow node22\ l\ a\ (del\ x\ r))$

$del\ x\ \langle l,\ a,\ r\rangle =$
(case $cmp\ x\ a$ of
$LT \Rightarrow node21\ (del\ x\ l)\ a\ r$
$\mid EQ \Rightarrow$ let $(a',\ t) = del\_min\ r$ in $node22\ l\ a'\ t$
$\mid GT \Rightarrow node22\ l\ a\ (del\ x\ r))$

$del\ x\ \langle l,\ a,\ r\rangle =$
(case $cmp\ x\ a$ of
$LT \Rightarrow node21\ (del\ x\ l)\ a\ r$
$\mid EQ \Rightarrow$ let $(a',\ t) = del\_min\ r$ in $node22\ l\ a'\ t$
$\mid GT \Rightarrow node22\ l\ a\ (del\ x\ r))$

$node21\ (T_d\ t_1)\ a\ t_2 = T_d\ \langle t_1,\ a,\ t_2\rangle$
$node21\ (Up_d\ t_1)\ a\ \langle t_2,\ b,\ t_3\rangle = Up_d\ \langle t_1,\ a,\ t_2,\ b,\ t_3\rangle$
$node21\ (Up_d\ t_1)\ a\ \langle t_2,\ b,\ t_3,\ c,\ t_4\rangle =$
$T_d\ \langle\langle t_1,\ a,\ t_2\rangle,\ b,\ \langle t_3,\ c,\ t_4\rangle\rangle$

Analogous: $node22$

Deletion preserves *bal*

Deletion preserves *bal*

After 13 simple lemmas:

Lemma
$bal\ t \Longrightarrow bal\ (tree_d\ (del\ x\ t))$

Corollary
$bal\ t \Longrightarrow bal\ (delete\ x\ t)$

Beyond 2-3 trees

datatype $'a\ tree234 =$
$Leaf \mid Node2\ \dots \mid Node3\ \dots \mid Node4\ \dots$

Beyond 2-3 trees

datatype $'a\ tree234 =$
$Leaf \mid Node2\ \dots \mid Node3\ \dots \mid Node4\ \dots$

Like 2-3 tress, but with many more cases

Beyond 2-3 trees

datatype $'a\ tree234 =$
$Leaf \mid Node2\ \dots \mid Node3\ \dots \mid Node4\ \dots$

Like 2-3 tress, but with many more cases

The general case:

B-trees and (a, b)-trees

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$\langle\rangle \approx \langle\rangle$$

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$\begin{aligned} \langle\rangle &\approx \langle\rangle \\ \langle t_1, a, t_2\rangle &\approx \langle t_1, a, t_2\rangle \\ \langle t_1, a, t_2, b, t_3\rangle &\approx \langle {\color{red}\langle t_1, a, t_2\rangle}, b, t_3\rangle \end{aligned}$$

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$
\begin{aligned}
\langle\rangle &\approx \langle\rangle \\
\langle t_1, a, t_2\rangle &\approx \langle t_1, a, t_2\rangle \\
\langle t_1, a, t_2, b, t_3\rangle &\approx \langle{\color{red}\langle t_1, a, t_2\rangle}, b, t_3\rangle \;\; \langle t_1, a, {\color{red}\langle t_2, b, t_3\rangle}\rangle
\end{aligned}
$$

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$
\begin{aligned}
\langle\rangle &\approx \langle\rangle \\
\langle t_1, a, t_2\rangle &\approx \langle t_1, a, t_2\rangle \\
\langle t_1, a, t_2, b, t_3\rangle &\approx \langle{\color{red}\langle t_1, a, t_2\rangle}, b, t_3\rangle \;\; \langle t_1, a, {\color{red}\langle t_2, b, t_3\rangle}\rangle \\
\langle t_1, a, t_2, b, t_3, c, t_4\rangle &\approx \langle{\color{red}\langle t_1, a, t_2\rangle}, b, {\color{red}\langle t_3, c, t_4\rangle}\rangle
\end{aligned}
$$

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$
\begin{aligned}
\langle\rangle &\approx \langle\rangle \\
\langle t_1, a, t_2\rangle &\approx \langle t_1, a, t_2\rangle \\
\langle t_1, a, t_2, b, t_3\rangle &\approx \langle{\color{red}\langle t_1, a, t_2\rangle}, b, t_3\rangle \;\; \langle t_1, a, {\color{red}\langle t_2, b, t_3\rangle}\rangle \\
\langle t_1, a, t_2, b, t_3, c, t_4\rangle &\approx \langle{\color{red}\langle t_1, a, t_2\rangle}, b, {\color{red}\langle t_3, c, t_4\rangle}\rangle
\end{aligned}
$$

Red means "I am part of a bigger node"

Structural invariants

Structural invariants

- The root is Black.
- Every $\langle\rangle$ is considered Black.
- If a node is Red,

Structural invariants

- The root is Black.
- Every $\langle\rangle$ is considered Black.
- If a node is Red, its children are Black.
- All paths from a node to a leaf have the same number of

Structural invariants

- The root is Black.
- Every $\langle\rangle$ is considered Black.
- If a node is Red, its children are Black.
- All paths from a node to a leaf have the same number of Black nodes.

Red-black trees

datatype $color = Red \mid Black$

Red-black trees

datatype $color = Red \mid Black$

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

Red-black trees

datatype $color = Red \mid Black$

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

Abbreviations:

$$\begin{aligned} \langle\rangle &\equiv Leaf \\ \langle c,\ l,\ a,\ r\rangle &\equiv Node\ c\ l\ a\ r \\ R\ l\ a\ r &\equiv Node\ Red\ l\ a\ r \end{aligned}$$

Color

$color :: 'a\ rbt \Rightarrow color$
$color\ \langle\rangle = Black$
$color\ \langle c,\ \_,\ \_,\ \_\rangle = c$

Color

$color :: 'a\ rbt \Rightarrow color$
$color\ \langle\rangle = Black$
$color\ \langle c,\ \_,\ \_,\ \_\rangle = c$

$paint :: color \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$
$paint\ c\ \langle\rangle = \langle\rangle$
$paint\ c\ \langle\_,\ l,\ a,\ r\rangle = \langle c,\ l,\ a,\ r\rangle$

Structural invariants

$rbt :: 'a\ rbt \Rightarrow bool$

$rbt\ t = (invc\ t \wedge invh\ t \wedge color\ t = Black)$

Structural invariants

$rbt :: 'a\ rbt \Rightarrow bool$

$rbt\ t = (invc\ t \wedge invh\ t \wedge color\ t = Black)$

$invc :: 'a\ rbt \Rightarrow bool$

$invc\ \langle\rangle = True$

$invc\ \langle c,\ l,\ \_,\ r\rangle =$

$(invc\ l \wedge$

$invc\ r \wedge$

$(c = Red \longrightarrow color\ l = Black \wedge color\ r = Black))$

Structural invariants

$rbt :: 'a\ rbt \Rightarrow bool$

Red-black trees

datatype $color = Red \mid Black$

datatype

$'a\ rbt = Leaf \mid Node\ color\ ('a\ tree)\ 'a\ ('a\ tree)$

Abbreviations:

$$\langle\rangle \equiv Leaf$$

Structural invariants

$invh :: 'a\ rbt \Rightarrow bool$

$invh\ \langle\rangle = True$
$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$

Structural invariants

$invh :: 'a\ rbt \Rightarrow bool$

$invh\ \langle\rangle = True$
$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$

$bheight :: 'a\ rbt \Rightarrow nat$

Structural invariants

$invh :: 'a\ rbt \Rightarrow bool$

$invh\ \langle\rangle = True$
$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$

$bheight :: 'a\ rbt \Rightarrow nat$

$bh(\langle\rangle) = 0$
$bh(\langle c,\ l,\ \_,\ \_\rangle) =$
$(\text{if } c = Black \text{ then } bh(l) + 1 \text{ else } bh(l))$

Logarithmic height

Lemma
$rbt\ t \Longrightarrow h(t) \leq 2 * \log_2 |t|_1$

Structural invariants

$invh :: 'a\ rbt \Rightarrow bool$

$invh\ \langle\rangle = True$
$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$

$bheight :: 'a\ rbt \Rightarrow nat$

Structural invariants

$rbt :: 'a\ rbt \Rightarrow bool$

$rbt\ t = (invc\ t \wedge invh\ t \wedge color\ t = Black)$

$invc :: 'a\ rbt \Rightarrow bool$

$invc\ \langle\rangle = True$
$invc\ \langle c,\ l,\ \_,\ r\rangle =$
$(invc\ l \wedge$
$invc\ r \wedge$
$(c = Red \longrightarrow color\ l = Black \wedge color\ r = Black))$