Script generated by TTT

Title: Nipkow: FDS (15.07.2020)

Date: Wed Jul 15 17:43:08 CEST 2020

Duration: 40:32 min

Pages: 81

One by one (Union)

Let $c(x)$ = cost of adding 1 element to set of size x

Cost of adding m elements to a set of n elements:

$$c(n) + \cdots + c(n + m - 1)$$

One by one (Union)

Let $c(x)$ = cost of adding 1 element to set of size x

Cost of adding m elements to a set of n elements:

$$c(n) + \cdots + c(n + m - 1)$$

$\Longrightarrow$ choose $m \leq n$ $\Longrightarrow$ smaller into bigger

One by one (Union)

Let $c(x)$ = cost of adding 1 element to set of size x

Cost of adding m elements to a set of n elements:

$$c(n) + \cdots + c(n + m - 1)$$

$\Longrightarrow$ choose $m \leq n \Longrightarrow$ smaller into bigger

If $c(x) = \log_2 x \Longrightarrow$
Cost $= O(m * \log_2(n + m))$

One by one (Union)

Let $c(x)$ = cost of adding 1 element to set of size x

Cost of adding m elements to a set of n elements:

$$c(n) + \cdots + c(n + m - 1)$$

$\Longrightarrow$ choose $m \leq n \Longrightarrow$ smaller into bigger

If $c(x) = \log_2 x \Longrightarrow$
Cost $= O(m * \log_2(n + m)) = O(m * \log_2 n)$

Similar for intersection and difference.

- We can do better than $O(m * \log_2 n)$

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$ — better than $O(m * \log_2 n)$ if $m \approx n$

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$ — better than $O(m * \log_2 n)$ if $m \approx n$

- This chapter:

  A parallel divide and conquer approach

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$ — better than $O(m * \log_2 n)$ if $m \approx n$
- This chapter:

  A parallel divide and conquer approach

- Cost: $\Theta(m * \log_2(\frac{n}{m} + 1))$

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$ — better than $O(m * \log_2 n)$ if $m \approx n$
- This chapter:

  A parallel divide and conquer approach
- Cost: $\Theta(m * \log_2(\frac{n}{m} + 1))$

- Works for many kinds of balanced trees

- We can do better than $O(m * \log_2 n)$
- Flatten trees to lists, merge, build balanced tree takes time $O(m + n)$ — better than $O(m * \log_2 n)$ if $m \approx n$
- This chapter:

  A parallel divide and conquer approach
- Cost: $\Theta(m * \log_2(\frac{n}{m} + 1))$
- Works for many kinds of balanced trees
- For ease of presentation: use concrete type $tree$

Uniform $tree$ type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with $'b$-augmented trees:

$('a \times 'b)\ tree$

Uniform $tree$ type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with $'b$-augmented trees:

$('a \times 'b)\ tree$

We work with this type of trees without committing to any particular kind of balancing schema.

Uniform $tree$ type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with $'b$-augmented trees:

$('a \times 'b)\ tree$

We work with this type of trees without committing to any particular kind of balancing schema.

In this chapter: $tree$ abbreviates $('a \times 'b)\ tree$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r \ \approx\ \langle l,\ (a,\ \_),\ r\rangle$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r \ \approx\ \langle l,\ (a,\ \_),\ r\rangle$ + rebalance

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r \ \approx\ \langle l,\ (a,\ \_),\ r\rangle$ + rebalance

Thus *join* determines the balancing schema

Just *join*

Given $join :: tree \Rightarrow {}'a \Rightarrow tree \Rightarrow tree$
we implement

Just *join*

Given $join :: tree \Rightarrow {}'a \Rightarrow tree \Rightarrow tree$
we implement

$insert :: {}'a \Rightarrow tree \Rightarrow tree$
$delete :: {}'a \Rightarrow tree \Rightarrow tree$
$union :: tree \Rightarrow tree \Rightarrow tree$
$inter :: tree \Rightarrow tree \Rightarrow tree$
$diff :: tree \Rightarrow tree \Rightarrow tree$

$split :: tree \Rightarrow {}'a \Rightarrow tree \times bool \times tree$

$split :: tree \Rightarrow {}'a \Rightarrow tree \times bool \times tree$
$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$

(case $cmp\ x\ a$ of

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$

(case $cmp\ x\ a$ of

$LT \Rightarrow$ let $(l_1,\ b,\ l_2) = split\ l\ x$ in

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$

(case $cmp\ x\ a$ of

$LT \Rightarrow$ let $(l_1,\ b,\ l_2) = split\ l\ x$ in $(l_1,\ b,\ join\ l_2\ a\ r)$ |

$split$ :: $tree \Rightarrow {}'a \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$
(case $cmp\ x\ a$ of
  $LT \Rightarrow$ let $(l_1,\ b,\ l_2) = split\ l\ x$ in $(l_1,\ b,\ join\ l_2\ a\ r)$ |
  $EQ \Rightarrow$

$split$ :: $tree \Rightarrow {}'a \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$
(case $cmp\ x\ a$ of
  $LT \Rightarrow$ let $(l_1,\ b,\ l_2) = split\ l\ x$ in $(l_1,\ b,\ join\ l_2\ a\ r)$ |
  $EQ \Rightarrow (l,\ True,\ r)$ |
  $GT \Rightarrow$ let $(r_1,\ b,\ r_2) = split\ r\ x$ in $(join\ l\ a\ r_2,\ b,\ r_1))$

$insert$ :: $'a \Rightarrow tree \Rightarrow tree$

$insert\ x\ t =$

$split$ :: $tree \Rightarrow {}'a \Rightarrow tree \times bool \times tree$

$split\ \langle\rangle\ x = (\langle\rangle,\ False,\ \langle\rangle)$

$split\ \langle l,\ (a,\ \_),\ r\rangle\ x =$
(case $cmp\ x\ a$ of
  $LT \Rightarrow$ let $(l_1,\ b,\ l_2) = split\ l\ x$ in $(l_1,\ b,\ join\ l_2\ a\ r)$ |
  $EQ \Rightarrow (l,\ True,\ r)$ |
  $GT \Rightarrow$ let $(r_1,\ b,\ r_2) = split\ r\ x$ in $(join\ l\ a\ r_2,\ b,\ r_1))$

$insert$ :: $'a \Rightarrow tree \Rightarrow tree$

$insert\ x\ t =$ (let $(l,\ \_,\ r) = split\ t\ x$ in

$split\_min$ :: $tree \Rightarrow {}'a \times tree$

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle\ =$

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle\ =$

(if $l = \langle\rangle$ then $(a,\ r)$ else

let $(m,\ l') = split\_min\ l$ in

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle\ =$

(if $l = \langle\rangle$ then $(a,\ r)$ else

let $(m,\ l') = split\_min\ l$ in $(m,\ join\ l'\ a\ r)$

$join2$ :: $tree \Rightarrow tree \Rightarrow tree$

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle\ =$

(if $l = \langle\rangle$ then $(a,\ r)$ else

let $(m,\ l') = split\_min\ l$ in $(m,\ join\ l'\ a\ r)$

$join2$ :: $tree \Rightarrow tree \Rightarrow tree$

$join2\ l\ r =$

(if $r = \langle\rangle$ then l

else let $(m,\ r') = split\_min\ r$ in

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle =$
(if $l = \langle\rangle$ then $(a,\ r)$ else
let $(m,\ l') = split\_min\ l$ in $(m,\ join\ l'\ a\ r)$)

$join2$ :: $tree \Rightarrow tree \Rightarrow tree$

$join2\ l\ r =$
(if $r = \langle\rangle$ then l
else let $(m,\ r') = split\_min\ r$ in $join\ l\ m\ r'$)

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle =$
(if $l = \langle\rangle$ then $(a,\ r)$ else
let $(m,\ l') = split\_min\ l$ in $(m,\ join\ l'\ a\ r)$)

$join2$ :: $tree \Rightarrow tree \Rightarrow tree$

$join2\ l\ r =$
(if $r = \langle\rangle$ then l
else let $(m,\ r') = split\_min\ r$ in $join\ l\ m\ r'$)

$delete$ :: ${'a} \Rightarrow tree \Rightarrow tree$

$delete\ x\ t =$ (let $(l,\ \_,\ r) = split\ t\ x$ in

$split\_min$ :: $tree \Rightarrow {'a} \times tree$

$split\_min\ \langle l,\ (a,\ \_),\ r\rangle =$
(if $l = \langle\rangle$ then $(a,\ r)$ else
let $(m,\ l') = split\_min\ l$ in $(m,\ join\ l'\ a\ r)$)

$join2$ :: $tree \Rightarrow tree \Rightarrow tree$

$join2\ l\ r =$
(if $r = \langle\rangle$ then l
else let $(m,\ r') = split\_min\ r$ in $join\ l\ m\ r'$)

$delete$ :: ${'a} \Rightarrow tree \Rightarrow tree$

$delete\ x\ t =$ (let $(l,\ \_,\ r) = split\ t\ x$ in $join2\ l\ r$)

$union$ :: $tree \Rightarrow tree \Rightarrow tree$

$union\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then t_2 else
if $t_2 = \langle\rangle$ then t_1 else

$union$ :: $tree \Rightarrow tree \Rightarrow tree$

$union\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then t_2 else
if $t_2 = \langle\rangle$ then t_1 else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$

$union$ :: $tree \Rightarrow tree \Rightarrow tree$

$union\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then t_2 else
if $t_2 = \langle\rangle$ then t_1 else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, \_, r_2) = split\ t_2\ a;$

$union$ :: $tree \Rightarrow tree \Rightarrow tree$

$union\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then t_2 else
if $t_2 = \langle\rangle$ then t_1 else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, \_, r_2) = split\ t_2\ a;$
$l' = union\ l_1\ l_2;$
$r' = union\ r_1\ r_2$

$union$ :: $tree \Rightarrow tree \Rightarrow tree$

$union\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then t_2 else
if $t_2 = \langle\rangle$ then t_1 else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, \_, r_2) = split\ t_2\ a;$
$l' = union\ l_1\ l_2;$
$r' = union\ r_1\ r_2$
in $join\ l'\ a\ r'$)

$inter :: tree \Rightarrow tree \Rightarrow tree$

$inter\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then $\langle\rangle$ else

$inter :: tree \Rightarrow tree \Rightarrow tree$

$inter\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then $\langle\rangle$ else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$

$inter :: tree \Rightarrow tree \Rightarrow tree$

$inter\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then $\langle\rangle$ else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, ain, r_2) = split\ t_2\ a;$

$inter :: tree \Rightarrow tree \Rightarrow tree$

$inter\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then $\langle\rangle$ else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, ain, r_2) = split\ t_2\ a;$
$l' = inter\ l_1\ l_2;$
$r' = inter\ r_1\ r_2$
in

$inter$ :: $tree \Rightarrow tree \Rightarrow tree$

$inter\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then $\langle\rangle$ else
case t_1 of
$\langle l_1, (a, \_), r_1\rangle \Rightarrow$
let $(l_2, ain, r_2) = split\ t_2\ a;$
$l' = inter\ l_1\ l_2;$
$r' = inter\ r_1\ r_2$
in if ain then $join\ l'\ a\ r'$

$diff$ :: $tree \Rightarrow tree \Rightarrow tree$

$diff\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else

$diff$ :: $tree \Rightarrow tree \Rightarrow tree$

$diff\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then t_1 else
case t_2 of
$\langle l_2, (a, \_), r_2\rangle \Rightarrow$

$diff$ :: $tree \Rightarrow tree \Rightarrow tree$

$diff\ t_1\ t_2 =$
(if $t_1 = \langle\rangle$ then $\langle\rangle$ else
if $t_2 = \langle\rangle$ then t_1 else
case t_2 of
$\langle l_2, (a, \_), r_2\rangle \Rightarrow$
let $(l_1, \_, r_1) = split\ t_1\ a;$

$diff :: tree \Rightarrow tree \Rightarrow tree$

```
diff t1 t2 =
(if t1 = ⟨⟩ then ⟨⟩ else
 if t2 = ⟨⟩ then t1 else
 case t2 of
   ⟨l2, (a, _), r2⟩ ⇒
     let (l1, _, r1) = split t1 a;
         l' = diff l1 l2;
         r' = diff r1 r2
     in join2 l' r')
```


$diff :: tree \Rightarrow tree \Rightarrow tree$

```
diff t1 t2 =
(if t1 = ⟨⟩ then ⟨⟩ else
 if t2 = ⟨⟩ then t1 else
 case t2 of
   ⟨l2, (a, _), r2⟩ ⇒
     let (l1, _, r1) = split t1 a;
         l' = diff l1 l2;
         r' = diff r1 r2
     in join2 l' r')
```

Why this way around: t_1/t_2?

13 Union, Intersection, Difference on BSTs
- Implementation
- Correctness
- Join for Red-Black Trees

Specification of *join*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$

Specification of *join*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Specification of *join* and *inv*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Also required: structural invariant *inv*:

Specification of *join* and *inv*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Also required: structural invariant *inv*:

- $inv\ \langle\rangle$
- $inv\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow inv\ l \wedge inv\ r$

Specification of *join* and *inv*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Also required: structural invariant *inv*:

- $inv\ \langle\rangle$
- $inv\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow inv\ l \wedge inv\ r$
- $[\![inv\ l;\ inv\ r]\!] \Longrightarrow inv\ (join\ l\ a\ r)$

Specification of *join* and *inv*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Also required: structural invariant *inv*:

- $inv\ \langle\rangle$
- $inv\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow inv\ l \wedge inv\ r$
- $[\![inv\ l;\ inv\ r]\!] \Longrightarrow inv\ (join\ l\ a\ r)$

Locale context for def of *union* etc

Specification of *union*, *inter*, *diff*

ADT/Locale *Set2* = extension of locale *Set* with

- $union,\ inter,\ diff :: {'s} \Rightarrow {'s} \Rightarrow {'s}$

Specification of *union*, *inter*, *diff*

ADT/Locale *Set2* = extension of locale *Set* with

- $union,\ inter,\ diff :: {'s} \Rightarrow {'s} \Rightarrow {'s}$
- $[\![invar\ s_1;\ invar\ s_2]\!]$
  $\Longrightarrow set\ (union\ s_1\ s_2) = set\ s_1 \cup set\ s_2$

Specification of *join* and *inv*

- $set\_tree\ (join\ l\ a\ r) = set\_tree\ l \cup \{a\} \cup set\_tree\ r$
- $bst\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow$
  $bst\ (join\ l\ a\ r)$

Also required: structural invariant *inv*:

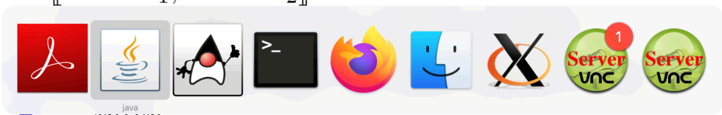
- $inv\ \langle\rangle$
- $inv\ \langle l,\ (a,\ b),\ r\rangle \Longrightarrow inv\ l \wedge inv\ r$
- $[\![inv\ l;\ inv\ r]\!] \Longrightarrow inv\ (join\ l\ a\ r)$

Locale context for def of *union* etc

Specification of *union*, *inter*, *diff*

ADT/Locale *Set2* = extension of locale *Set* with

- $union,\ inter,\ diff :: 's \Rightarrow 's \Rightarrow 's$
- $[\![invar\ s_1;\ invar\ s_2]\!]$



- $\dots\ inter\ \dots$
- $\dots\ diff\ \dots$

Correctness lemmas for *union* etc code

In the context of *join* specification:

- $bst\ t_2 \Longrightarrow$
  $set\_tree\ (union\ t_1\ t_2) = set\_tree\ t_1 \cup set\_tree\ t_2$

Specification of *union*, *inter*, *diff*

ADT/Locale *Set2* = extension of locale *Set* with

- $union,\ inter,\ diff :: 's \Rightarrow 's \Rightarrow 's$
- $[\![invar\ s_1;\ invar\ s_2]\!]$
  $\Longrightarrow set\ (union\ s_1\ s_2) = set\ s_1 \cup set\ s_2$
- $[\![invar\ s_1;\ invar\ s_2]\!] \Longrightarrow invar\ (union\ s_1\ s_2)$
- $\dots\ inter\ \dots$
- $\dots\ diff\ \dots$

We focus on *union*.

See `HOL/Data_Structures/Set_Specs.thy`

Correctness lemmas for *union* etc code

In the context of *join* specification:

- $bst\ t_2 \Longrightarrow$
  $set\_tree\ (union\ t_1\ t_2) = set\_tree\ t_1 \cup set\_tree\ t_2$
- $[\![bst\ t_1;\ bst\ t_2]\!] \Longrightarrow bst\ (union\ t_1\ t_2)$
- $[\![inv\ t_1;\ inv\ t_2]\!] \Longrightarrow inv\ (union\ t_1\ t_2)$

Proofs automatic (more complex for *inter* and *diff*)

Correctness lemmas for *union* etc code

In the context of *join* specification:

- $bst\ t_2 \Longrightarrow$
  $set\_tree\ (union\ t_1\ t_2) = set\_tree\ t_1 \cup set\_tree\ t_2$
- $[\![bst\ t_1;\ bst\ t_2]\!] \Longrightarrow bst\ (union\ t_1\ t_2)$
- $[\![inv\ t_1;\ inv\ t_2]\!] \Longrightarrow inv\ (union\ t_1\ t_2)$

Proofs automatic (more complex for *inter* and *diff*)

Implementation of locale *Set2*:

interpretation *Set2* **where** $union = union \ldots$

Correctness lemmas for *union* etc code

In the context of *join* specification:

- $bst\ t_2 \Longrightarrow$
  $set\_tree\ (union\ t_1\ t_2) = set\_tree\ t_1 \cup set\_tree\ t_2$
- $[\![bst\ t_1;\ bst\ t_2]\!] \Longrightarrow bst\ (union\ t_1\ t_2)$
- $[\![inv\ t_1;\ inv\ t_2]\!] \Longrightarrow inv\ (union\ t_1\ t_2)$

Proofs automatic (more complex for *inter* and *diff*)

Implementation of locale *Set2*:

interpretation *Set2* **where** $union = union \ldots$
and $set = set\_tree$ **and** $invar = (\lambda t.\ bst\ t \wedge inv\ t)$

`HOL/Data_Structures/Set2_Join.thy`

13 Union, Intersection, Difference on BSTs
- Implementation
- Correctness
- Join for Red-Black Trees

join l a r — The idea

Assume l is "smaller" than r :

join l a r — The idea

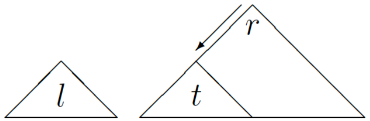
Assume l is "smaller" than r :

- Descend along the left spine of r until you find a subtree t of the same "size" as l:

join l a r — The idea

Assume l is "smaller" than r :

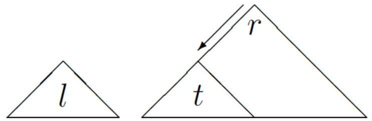
- Descend along the left spine of r until you find a subtree t of the same "size" as l:



join l a r — The idea

Assume l is "smaller" than r :

- Descend along the left spine of r until you find a subtree t of the same "size" as l:



- Replace t by $\langle l, a, t \rangle$.

$join\ l\ x\ r =$
(if $bheight\ r < bheight\ l$
then $paint\ Black\ (joinR\ l\ x\ r)$
else if $bheight\ l < bheight\ r$
then $paint\ Black\ (joinL\ l\ x\ r)$ else $B\ l\ x\ r)$

$join\ l\ x\ r =$
(if $bheight\ r < bheight\ l$
then $paint\ Black\ (joinR\ l\ x\ r)$
else if $bheight\ l < bheight\ r$
then $paint\ Black\ (joinL\ l\ x\ r)$ else $B\ l\ x\ r)$

Need to store black height in each node
for logarithmic complexity

HOL/Data_Structures/
Set2_Join_RBT.thy

HOL/Data_Structures/
Set2_Join_RBT.thy

$join\ l\ x\ r =$
(if $bheight\ r < bheight\ l$
then $paint\ Black\ (joinR\ l\ x\ r)$
else if $bheight\ l < bheight\ r$
then $paint\ Black\ (joinL\ l\ x\ r)$ else $B\ l\ x\ r$)

Need to store black height in each node
for logarithmic complexity

Literature

The idea of "just $join$":

Stephen Adams. *Efficient Sets — A Balancing Act*.
J. Functional Programming, volume 3, number 4, 1993.

The precise analysis:

Guy E. Blelloch, D. Ferizovic, Y. Sun.
Just Join for Parallel Ordered Sets.
ACM Symposium on Parallelism in Algorithms and Architectures 2016.