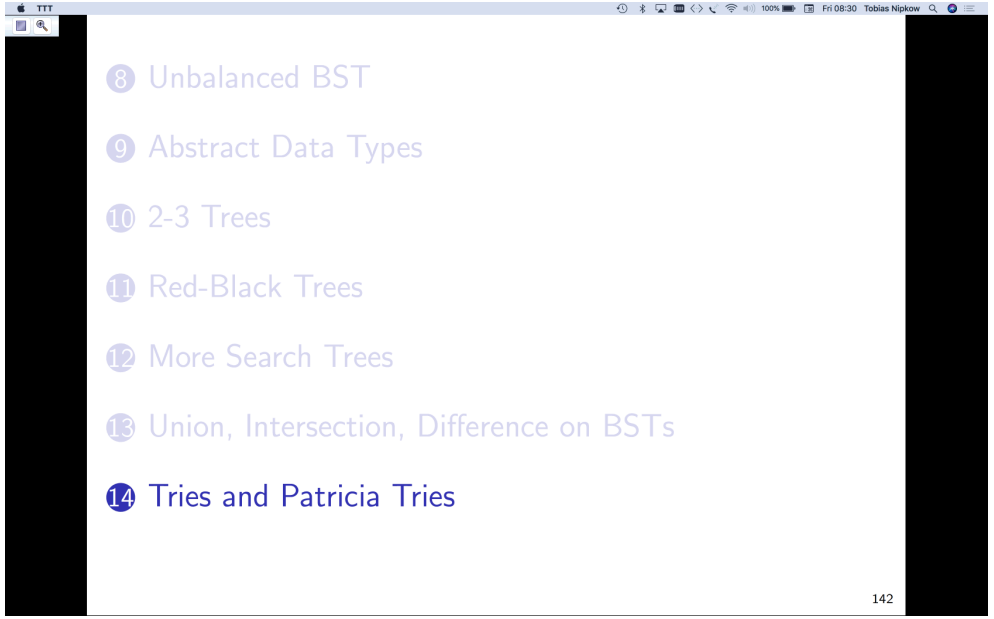
Script **generated by TTT**

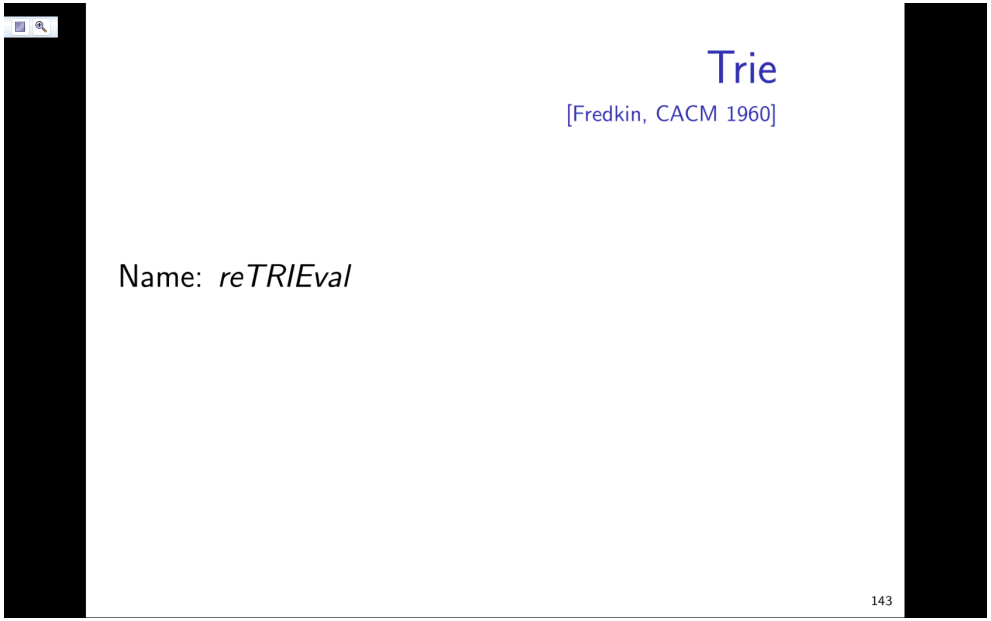
Title: FDS (28.06.2019)

Date: Fri Jun 28 08:30:47 CEST 2019

Duration: 82:04 min

Pages: 73





Trie

[Fredkin, CACM 1960]

Name: *reTRIEval*

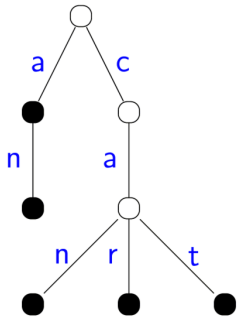
- Tries are search trees indexed by lists
- Tries are tree-shaped DFAs

Example Trie

{ a, an, can, car, cat }

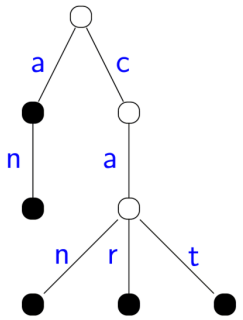
Example Trie

{ a, an, can, car, cat }



Example Trie

{ a, an, can, car, cat }



Trie

datatype $('a\ trie = Nd\ bool\ ('a \Rightarrow 'a\ trie\ option)$

Trie

datatype $('a\ trie = Nd\ bool\ ('a \Rightarrow 'a\ trie\ option)$

Function update notation:

$f(a := b) = (\lambda x.\ \text{if}\ x = a\ \text{then}\ b\ \text{else}\ f\ x)$

Trie

datatype $('a\ trie = Nd\ bool\ ('a \Rightarrow 'a\ trie\ option)$

Function update notation:

$f(a := b) = (\lambda x.\ \text{if}\ x = a\ \text{then}\ b\ \text{else}\ f\ x)$

$f(a \mapsto b) = f(a := Some\ b)$

Trie

datatype $'a\ trie = Nd\ bool\ ('a \Rightarrow 'a\ trie\ option)$

Function update notation:

$f(a := b) = (\lambda x.\ \text{if } x = a \text{ then } b \text{ else } f\ x)$

$f(a \mapsto b) = f(a := Some\ b)$

Next: Implementation of ADT *Set*

empty

$empty =$

empty

$empty =$

empty

$empty = Nd\ False\ (\lambda\_.\ None)$

isin

$isin\ (Nd\ b\ m)\ [] = b$

isin

$isin\ (Nd\ b\ m)\ [] = b$

$isin\ (Nd\ b\ m)\ (k\ \#\ xs) = (\mathsf{case}\ m\ k\ \mathsf{of}$
$\qquad None \Rightarrow False$
$\qquad \mid Some\ t \Rightarrow isin\ t\ xs)$

insert

$insert\ []\ (Nd\ b\ m) = Nd\ True\ m$

insert

$insert\ []\ (Nd\ b\ m) = Nd\ True\ m$

$insert\ (x\ \#\ xs)\ (Nd\ b\ m) =$

insert

$$insert\ []\ (Nd\ b\ m) = Nd\ True\ m$$

$$insert\ (x\ \#\ xs)\ (Nd\ b\ m) =$$
$$Nd\ b\ (m(x \mapsto insert\ xs\ (\textsf{case}\ m\ x\ \textsf{of}$$
$$None \Rightarrow empty$$
$$\mid Some\ t \Rightarrow t)))$$

delete

$$delete\ []\ (Nd\ b\ m) = Nd\ False\ m$$

delete

$$delete\ []\ (Nd\ b\ m) = Nd\ False\ m$$

$$delete\ (x\ \#\ xs)\ (Nd\ b\ m) =$$
$$Nd\ b\ (\textsf{case}\ m\ x\ \textsf{of}$$
$$None \Rightarrow m$$
$$\mid Some\ t \Rightarrow m(x \mapsto delete\ xs\ t))$$

delete

$$delete\ []\ (Nd\ b\ m) = Nd\ False\ m$$

$$delete\ (x\ \#\ xs)\ (Nd\ b\ m) =$$
$$Nd\ b\ (\textsf{case}\ m\ x\ \textsf{of}$$
$$None \Rightarrow m$$
$$\mid Some\ t \Rightarrow m(x \mapsto delete\ xs\ t))$$

Does not shrink trie

Correctness: Abstraction function

$set :: 'a\ trie \Rightarrow 'a\ list\ set$

Correctness theorems

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

Correctness theorems

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

No lemmas required

Correctness theorems

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

Trie

datatype $'a\ trie = Nd\ bool\ ('a \Rightarrow 'a\ trie\ option)$

Function update notation:

$f(a := b) = (\lambda x.\ \text{if } x = a \text{ then } b \text{ else } f\ x)$

$f(a \mapsto b) = f(a := Some\ b)$

Next: Implementation of ADT Set

Trie

datatype $trie = Lf \mid Nd\ bool\ (trie \times trie)$

Trie

datatype $trie = Lf \mid Nd\ bool\ (trie \times trie)$

Auxiliary functions on pairs:

$sel2 :: bool \Rightarrow 'a \times 'a \Rightarrow 'a$

Trie

datatype $trie = Lf \mid Nd\ bool\ (trie \times trie)$

Auxiliary functions on pairs:

$sel2 :: bool \Rightarrow 'a \times 'a \Rightarrow 'a$
$sel2\ b\ (a_1,\ a_2) = (\text{if } b \text{ then } a_2 \text{ else } a_1)$

Trie

datatype $trie = Lf \mid Nd\ bool\ (trie \times trie)$

Auxiliary functions on pairs:

$sel2 :: bool \Rightarrow 'a \times 'a \Rightarrow 'a$
$sel2\ b\ (a_1,\ a_2) = (\text{if } b \text{ then } a_2 \text{ else } a_1)$

$mod2 :: ('a \Rightarrow 'a) \Rightarrow bool \Rightarrow 'a \times 'a \Rightarrow 'a \times 'a$

Trie

datatype $trie = Lf \mid Nd\ bool\ (trie \times trie)$

Auxiliary functions on pairs:

$sel2 :: bool \Rightarrow 'a \times 'a \Rightarrow 'a$
$sel2\ b\ (a_1,\ a_2) = (\text{if } b \text{ then } a_2 \text{ else } a_1)$

$mod2 :: ('a \Rightarrow 'a) \Rightarrow bool \Rightarrow 'a \times 'a \Rightarrow 'a \times 'a$

$mod2\ f\ b\ (a_1,\ a_2) = (\text{if } b \text{ then } (a_1,\ f\ a_2) \text{ else } (f\ a_1,\ a_2))$

empty

$empty = Lf$

isin

$$isin\ Lf\ ks = False$$

$$isin\ (Nd\ b\ lr)\ ks = (\textsf{case}\ ks\ \textsf{of}\ [] \Rightarrow b \mid k\ \#\ x \Rightarrow isin\ (sel2\ k\ lr)\ x)$$

insert

$$insert\ []\ Lf = Nd\ True\ (Lf,\ Lf)$$

insert

$$insert\ []\ Lf = Nd\ True\ (Lf,\ Lf)$$

$$insert\ []\ (Nd\ b\ lr) = Nd\ True\ lr$$

$$insert\ (k\ \#\ ks)\ Lf = Nd\ False\ (mod2\ (insert\ ks)\ k\ (Lf,\ Lf))$$

insert

$$insert\ []\ Lf = Nd\ True\ (Lf,\ Lf)$$

$$insert\ []\ (Nd\ b\ lr) = Nd\ True\ lr$$

$$insert\ (k\ \#\ ks)\ Lf = Nd\ False\ (mod2\ (insert\ ks)\ k\ (Lf,\ Lf))$$

$$insert\ (k\ \#\ ks)\ (Nd\ b\ lr) = Nd\ b\ (mod2\ (insert\ ks)\ k\ lr)$$

delete

$delete\ ks\ Lf = Lf$

Correctness of implementation

Abstraction function:

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

Correctness of implementation

Abstraction function:

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- $isin\ (insert\ xs\ t)\ ys = (xs = ys \vee isin\ t\ ys)$

Correctness of implementation

Abstraction function:

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- $isin\ (insert\ xs\ t)\ ys = (xs = ys \vee isin\ t\ ys)$
  $\Longrightarrow\ set\_trie\ (insert\ xs\ t) = set\_trie\ t \cup \{xs\}$

Correctness of implementation

Abstraction function:

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- $isin\ (insert\ xs\ t)\ ys = (xs = ys \vee isin\ t\ ys)$
  $\Longrightarrow\ set\_trie\ (insert\ xs\ t) = set\_trie\ t \cup \{xs\}$
- $isin\ (delete\ xs\ t)\ ys = (xs \neq ys \wedge isin\ t\ ys)$
  $\Longrightarrow\ set\_trie\ (delete\ xs\ t) = set\_trie\ t - \{xs\}$

Abstraction function via $isin$

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

Abstraction function via $isin$

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- Trivial definition

Abstraction function via $isin$

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- Trivial definition
- Reusing code ($isin$) may complicate proofs.

Abstraction function via *isin*

$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- Trivial definition
- Reusing code (*isin*) may complicate proofs.
- Separate abstract mathematical definition can simplify proofs (see tries with functions)

Abstraction function via *isin*

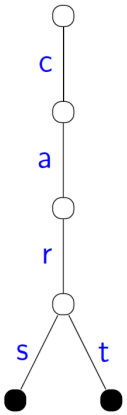
$$set\_trie\ t = \{xs.\ isin\ t\ xs\}$$

- Trivial definition
- Reusing code (*isin*) may complicate proofs.
- Separate abstract mathematical definition can simplify proofs (see tries with functions)

Also possible for some other ADTs, e.g. for Map:
$lookup :: 't \Rightarrow ('a \Rightarrow 'b\ option)$

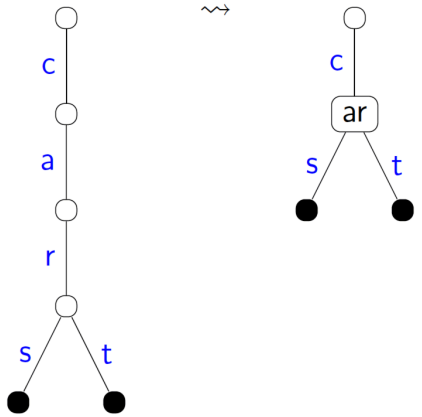
162

From tries to Patricia tries



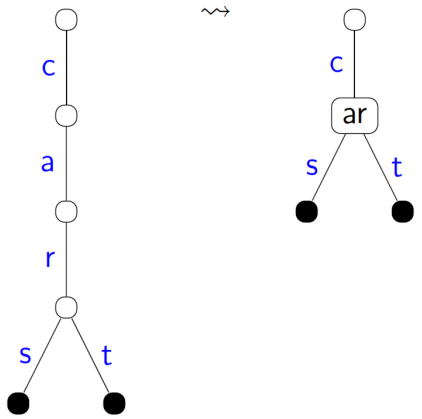
163

From tries to Patricia tries



163

From tries to Patricia tries



Patricia trie

datatype $trieP = LfP$
$\mid NdP\ (bool\ list)\ bool\ (trieP \times trieP)$

Splitting lists

$split\ xs\ ys = (zs,\ xs',\ ys')$

$isinP$

$isinP\ LfP\ ks = False$

$isinP\ (NdP\ ps\ b\ lr)\ ks =$
(let $n = length\ ps$
 in if $ps = take\ n\ ks$
  then case $drop\ n\ ks$ of
    $[] \Rightarrow b$
   $\mid k\ \#\ ks' \Rightarrow isinP\ (sel2\ k\ lr)\ ks'$
  else $False$)

insertP

$insertP\ ks\ LfP = NdP\ ks\ True\ (LfP,\ LfP)$

insertP

$insertP\ ks\ LfP = NdP\ ks\ True\ (LfP,\ LfP)$

$insertP\ ks\ (NdP\ ps\ b\ lr) =$
(case $split\ ks\ ps$ of
$\quad (qs,\ [],\ []) \Rightarrow NdP\ ps\ True\ lr$
$\mid (qs,\ [],\ p\ \#\ ps') \Rightarrow$
$\quad$ let $t = NdP\ ps'\ b\ lr$
$\quad$ in $NdP\ qs\ True$ (if p then $(LfP,\ t)$ else $(t,\ LfP)$)
$\mid (qs,\ k\ \#\ ks',\ []) \Rightarrow NdP\ ps\ b\ (mod2\ (insertP\ ks')\ k\ lr)$
$\mid (qs,\ k\ \#\ ks',\ p\ \#\ ps') \Rightarrow$
$\quad$ let $tp = NdP\ ps'\ b\ lr;\ tk = NdP\ ks'\ True\ (LfP,\ LfP)$
$\quad$ in $NdP\ qs\ False$ (if k then $(tp,\ tk)$ else $(tk,\ tp)$)))

deleteP

$deleteP\ ks\ LfP = LfP$

deleteP

$deleteP\ ks\ LfP = LfP$

$deleteP\ ks\ (NdP\ ps\ b\ lr) =$

deleteP

$deleteP\ ks\ LfP = LfP$

$deleteP\ ks\ (NdP\ ps\ b\ lr) =$

deleteP

$deleteP\ ks\ LfP = LfP$

$deleteP\ ks\ (NdP\ ps\ b\ lr) =$
$(case\ split\ ks\ ps\ of$
$\quad (qs,\ ks',\ p\#ps') \Rightarrow NdP\ ps\ b\ lr\ |$
$\quad (qs,\ k\#ks',\ []) \Rightarrow$
$\quad\quad nodeP\ ps\ b\ (mod2\ (deleteP\ ks')\ k\ lr)\ |$
$\quad (qs,\ [],\ []) \Rightarrow nodeP\ ps\ False\ lr)$

Stepwise data refinement

View *trieP* as an implementation ("refinement") of *trie*

Stepwise data refinement

View *trieP* as an implementation ("refinement") of *trie*

Type	Abstraction function
bool list set	
↑	*set_trie*
trie	
↑	*abs_trieP*
trieP	

Stepwise data refinement

View $trieP$ as an implementation ("refinement") of $trie$

Type	Abstraction function
$bool\ list\ set$	
$\uparrow$	set_trie
$trie$	
$\uparrow$	abs_trieP
$trieP$	

$\Longrightarrow$ Modular correctness proof of $trieP$

$abs\_trieP :: trieP \Rightarrow trie$

$abs\_trieP\ LfP = Lf$

$abs\_trieP\ (NdP\ ps\ b\ (l,\ r)) =$
$prefix\_trie\ ps\ (Nd\ b\ (abs\_trieP\ l,\ abs\_trieP\ r))$

$abs\_trieP :: trieP \Rightarrow trie$

$abs\_trieP\ LfP = Lf$

$abs\_trieP\ (NdP\ ps\ b\ (l,\ r)) =$
$prefix\_trie\ ps\ (Nd\ b\ (abs\_trieP\ l,\ abs\_trieP\ r))$

$prefix\_trie :: bool\ list \Rightarrow trie \Rightarrow trie$

Correctness of $trieP$ w.r.t. $trie$

- $isinP\ t\ ks = isin\ (abs\_trieP\ t)\ ks$
- $abs\_trieP\ (insertP\ ks\ t) = insert\ ks\ (abs\_trieP\ t)$
- $abs\_trieP\ (deleteP\ ks\ t) = delete\ ks\ (abs\_trieP\ t)$

Correctness of *trieP* w.r.t. *trie*

- $isinP\ t\ ks = isin\ (abs\_trieP\ t)\ ks$
- $abs\_trieP\ (insertP\ ks\ t) = insert\ ks\ (abs\_trieP\ t)$
- $abs\_trieP\ (deleteP\ ks\ t) = delete\ ks\ (abs\_trieP\ t)$

$isin\ (prefix\_trie\ ps\ t)\ ks =$
$(ps = take\ (length\ ps)\ ks \wedge isin\ t\ (drop\ (length\ ps)\ ks))$
$prefix\_trie\ ks\ (Nd\ True\ (Lf, Lf)) = insert\ ks\ Lf$
$insert\ ps\ (prefix\_trie\ ps\ (Nd\ b\ lr)) = prefix\_trie\ ps\ (Nd\ True\ lr)$
$insert\ (ks \mathbin{@} ks')\ (prefix\_trie\ ks\ t) = prefix\_trie\ ks\ (insert\ ks'\ t)$
$prefix\_trie\ (ps \mathbin{@} qs)\ t = prefix\_trie\ ps\ (prefix\_trie\ qs\ t)$
$split\ ks\ ps = (qs, ks', ps') \Longrightarrow$
$ks = qs \mathbin{@} ks' \wedge ps = qs \mathbin{@} ps' \wedge (ks' \neq [] \wedge ps' \neq [] \longrightarrow hd\ ks' \neq hd\ ps')$
$(prefix\_trie\ xs\ t = Lf) = (xs = [] \wedge t = Lf)$
$(abs\_trieP\ t = Lf) = (t = LfP)$
$delete\ xs\ (prefix\_trie\ xs\ (Nd\ b\ (l, r))) =$
(if $(l, r) = (Lf, Lf)$ then Lf else $prefix\_trie\ xs\ (Nd\ False\ (l, r))$)
$delete\ (xs \mathbin{@} ys)\ (prefix\_trie\ xs\ t) =$
(if $delete\ ys\ t = Lf$ then Lf else $prefix\_trie\ xs\ (delete\ ys\ t)$)

Correctness of *trieP* w.r.t. *bool list set*

Define $set\_trieP = set\_trie \circ abs\_trieP$

Correctness of *trieP* w.r.t. *bool list set*

Define $set\_trieP = set\_trie \circ abs\_trieP$

$\Longrightarrow$ Overall correctness by trivial composition of correctness theorems for *trie* and *trieP*

Example:

$set\_trieP\ (insertP\ xs\ t) = set\_trieP\ t \cup \{xs\}$

Correctness of *trieP* w.r.t. *bool list set*

Define $set\_trieP = set\_trie \circ abs\_trieP$

$\Longrightarrow$ Overall correctness by trivial composition of correctness theorems for *trie* and *trieP*

Example:

$set\_trieP\ (insertP\ xs\ t) = set\_trieP\ t \cup \{xs\}$

follows directly from

$abs\_trieP\ (insertP\ ks\ t) = insert\ ks\ (abs\_trieP\ t)$
$set\_trie\ (insert\ xs\ t) = set\_trie\ t \cup \{xs\}$

Correctness of $trieP$ w.r.t. $bool\ list\ set$

Define $set\_trieP = set\_trie \circ abs\_trieP$

$\Longrightarrow$ Overall correctness by trivial composition of correctness theorems for $trie$ and $trieP$

Example:

$set\_trieP\ (insertP\ xs\ t) = set\_trieP\ t \cup \{xs\}$

follows directly from

$abs\_trieP\ (insertP\ ks\ t) = insert\ ks\ (abs\_trieP\ t)$

$set\_trie\ (insert\ xs\ t) = set\_trie\ t \cup \{xs\}$

Chapter 9

Priority Queues