Script **generated by TTT**

Title: FDS (06.07.2018)

Date: Fri Jul 06 08:33:38 CEST 2018

Duration: 85:01 min

Pages: 70

Chapter 10

Amortized Complexity

20 Amortized Complexity

21 Skew Heap

22 Splay Tree

23 Pairing Heap

24 More Verified Data Structures and Algorithms (in Isabelle/HOL)

Example

n increments of a binary counter starting with 0

Example

n increments of a binary counter starting with 0

- WCC of one increment?

WCC = worst case complexity

Example

n increments of a binary counter starting with 0

- WCC of one increment? $O(\log_2 n)$

WCC = worst case complexity

Example

n increments of a binary counter starting with 0

- WCC of one increment? $O(\log_2 n)$
- WCC of n increments? $O(n * \log_2 n)$
- $O(n * \log_2 n)$ is too pessimistic!
- Every second increment is cheap and compensates for the more expensive increments

WCC = worst case complexity

Example

n increments of a binary counter starting with 0

- WCC of one increment? $O(\log_2 n)$
- WCC of n increments? $O(n * \log_2 n)$
- $O(n * \log_2 n)$ is too pessimistic!
- Every second increment is cheap and compensates for the more expensive increments
- Fact: WCC of n increments is $O(n)$

WCC = worst case complexity

The problem

WCC of individual operations
may lead to overestimation of
WCC of sequences of operations

Amortized analysis

Idea:

Try to determine the average cost of each operation

Amortized analysis

Idea:

Try to determine the average cost of each operation
(in the worst case!)

Amortized analysis

Idea:

Try to determine the average cost of each operation
(in the worst case!)

Use cheap operations to pay for expensive ones

Method:

- Cheap operations pay extra (into a "bank account"), making them more expensive

Bank account = *Potential*

Bank account = *Potential*

- The potential ("credit") is implicitly "stored" in the data structure.

Bank account = *Potential*

- The potential ("credit") is implicitly "stored" in the data structure.
- Potential Φ :: $\textit{data-structure} \Rightarrow \textit{non-neg. number}$ tells us how much credit is stored in a data structure

Bank account = *Potential*

- The potential ("credit") is implicitly "stored" in the data structure.
- Potential Φ :: $\textit{data-structure} \Rightarrow \textit{non-neg. number}$ tells us how much credit is stored in a data structure
- Increase in potential = deposit to pay for *later* expensive operation

Bank account = *Potential*

- The potential ("credit") is implicitly "stored" in the data structure.
- Potential Φ :: *data-structure* $\Rightarrow$ *non-neg. number* tells us how much credit is stored in a data structure
- Increase in potential = deposit to pay for *later* expensive operation
- Decrease in potential = withdrawal to pay for expensive operation

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip
  - take out 1 for each $1 \to 0$ flip

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip
  - take out 1 for each $1 \to 0$ flip

$\Longrightarrow$ increment has amortized cost $2 = 1{+}1$

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip
  - take out 1 for each $1 \to 0$ flip

$\Longrightarrow$ increment has amortized cost $2 = 1{+}1$

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip
  - take out 1 for each $1 \to 0$ flip

$\Longrightarrow$ increment has amortized cost $2 = 1{+}1$

Formalization via potential:

$\Phi\ counter$ = the number of 1's in $counter$

Back to example: counter

Increment:

- Actual cost: 1 for each bit flip
- Bank transaction:
  - pay in 1 for final $0 \to 1$ flip
  - take out 1 for each $1 \to 0$ flip

$\Longrightarrow$ increment has amortized cost $2 = 1{+}1$

Formalization via potential:

$\Phi\ counter$ = the number of 1's in $counter$

Data structure

Given an implementation:

Data structure

Given an implementation:

- Type τ
- Operation(s) $f :: \tau \Rightarrow \tau$

Data structure

Given an implementation:

- Type τ
- Operation(s) $f :: \tau \Rightarrow \tau$
  (may have additional parameters)

Data structure

Given an implementation:

- Type τ
- Operation(s) $f :: \tau \Rightarrow \tau$
  (may have additional parameters)
- Initial value: $init :: \tau$
  (function "empty")

Needed for complexity analysis:

- Time/cost: $t\_f :: \tau \Rightarrow num$
  ($num =$ some numeric type

Data structure

Given an implementation:

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,$$

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

$$a_{i+1} := t\_f_{i+1}\ s_i + \Phi\ s_{i+1} - \Phi\ s_i$$

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

$$a_{i+1} := t\_f_{i+1}\ s_i + \Phi\ s_{i+1} - \Phi\ s_i$$

$$\Longrightarrow$$

Sum of amortized costs $\geq$ sum of real costs

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

$$a_{i+1} := t\_f_{i+1}\ s_i + \Phi\ s_{i+1} - \Phi\ s_i$$

$$\Longrightarrow$$

Sum of amortized costs ≥ sum of real costs

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (t\_f_i\ s_{i-1} + \Phi\ s_i - \Phi\ s_{i-1})$$

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

$$a_{i+1} := t\_f_{i+1}\ s_i + \Phi\ s_{i+1} - \Phi\ s_i$$

$$\Longrightarrow$$

Sum of amortized costs ≥ sum of real costs

$$\begin{aligned}\sum_{i=1}^{n} a_i &= \sum_{i=1}^{n} (t\_f_i\ s_{i-1} + \Phi\ s_i - \Phi\ s_{i-1}) \\ &= (\sum_{i=1}^{n} t\_f_i\ s_{i-1}) + \Phi\ s_n - \Phi\ init\end{aligned}$$

Amortized and real cost

Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:

$$s_0 := init,\ s_1 := f_1\ s_0,\ \ldots,\ s_n := f_n\ s_{n-1}$$

Amortized cost := real cost + potential difference

$$a_{i+1} := t\_f_{i+1}\ s_i + \Phi\ s_{i+1} - \Phi\ s_i$$

$$\Longrightarrow$$

Sum of amortized costs ≥ sum of real costs

$$\begin{aligned}\sum_{i=1}^{n} a_i &= \sum_{i=1}^{n} (t\_f_i\ s_{i-1} + \Phi\ s_i - \Phi\ s_{i-1}) \\ &= (\sum_{i=1}^{n} t\_f_i\ s_{i-1}) + \Phi\ s_n - \Phi\ init \\ &\geq \sum_{i=1}^{n} t\_f_i\ s_{i-1}\end{aligned}$$

Verification of amortized cost

For each operation f:
provide an upper bound for its amortized cost

$$a\_f :: \tau \Rightarrow num$$

and prove

$$t\_f\ s + \Phi(f\ s) - \Phi\ s \leq a\_f\ s$$

Back to example: counter

$incr :: bool\ list \Rightarrow bool\ list$

Back to example: counter

$incr :: bool\ list \Rightarrow bool\ list$

$incr\ [] = [True]$
$incr\ (False\ \#\ bs) = True\ \#\ bs$
$incr\ (True\ \#\ bs) = False\ \#\ incr\ bs$

Back to example: counter

$incr :: bool\ list \Rightarrow bool\ list$
$incr\ [] = [True]$
$incr\ (False\ \#\ bs) = True\ \#\ bs$
$incr\ (True\ \#\ bs) = False\ \#\ incr\ bs$

$init = []$

$\Phi\ bs = length\ (filter\ id\ bs)$

Back to example: counter

$incr :: bool\ list \Rightarrow bool\ list$
$incr\ [] = [True]$
$incr\ (False\ \#\ bs) = True\ \#\ bs$
$incr\ (True\ \#\ bs) = False\ \#\ incr\ bs$

$init = []$

$\Phi\ bs = length\ (filter\ id\ bs)$

Lemma
$t\_incr\ bs + \Phi\ (incr\ bs) - \Phi\ bs = 2$

Back to example: counter

$incr :: bool\ list \Rightarrow bool\ list$
$incr\ [] = [True]$
$incr\ (False\ \#\ bs) = True\ \#\ bs$
$incr\ (True\ \#\ bs) = False\ \#\ incr\ bs$

Proof obligation summary

- $\Phi\ s \geq 0$
- $\Phi\ init = 0$
- For every operation $f :: \tau \Rightarrow ... \Rightarrow \tau$:
  $t\_f\ s\ \overline{x} + \Phi(f\ s\ \overline{x}) - \Phi\ s \leq a\_f\ s\ \overline{x}$

Proof obligation summary

- $\Phi\ s \geq 0$
- $\Phi\ init = 0$
- For every operation $f :: \tau \Rightarrow ... \Rightarrow \tau$:
  $t\_f\ s\ \overline{x} + \Phi(f\ s\ \overline{x}) - \Phi\ s \leq a\_f\ s\ \overline{x}$

If the data structure has an invariant $invar$:
assume precondition $invar\ s$

Proof obligation summary

- $\Phi\ s \geq 0$
- $\Phi\ init = 0$
- For every operation $f :: \tau \Rightarrow ... \Rightarrow \tau$:
  $t\_f\ s\ \overline{x} + \Phi(f\ s\ \overline{x}) - \Phi\ s \leq a\_f\ s\ \overline{x}$

If the data structure has an invariant $invar$:
assume precondition $invar\ s$

If f takes 2 arguments of type τ:
$t\_f\ s_1\ s_2\ \overline{x} + \Phi(f\ s_1\ s_2\ \overline{x}) - \Phi\ s_1 - \Phi\ s_2 \leq a\_f\ s_1\ s_2\ \overline{x}$

Warning: real time

Amortized analysis unsuitable for real time applications:

Warning: real time

Amortized analysis unsuitable for real time applications:

Real running time for individual calls
may be much worse than amortized time

Warning: single threaded

Amortized analysis is only correct for single threaded uses of the data structure.

Warning: single threaded

Amortized analysis is only correct for single threaded uses of the data structure.

Single threaded = no value is used more than once

Warning: single threaded

Amortized analysis is only correct for single threaded uses of the data structure.

Single threaded = no value is used more than once

Otherwise:

let $counter = 0;$
$bad =$ increment $counter\ 2^n - 1$ times;
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\vdots$

Warning: observer functions

Observer function: does not modify data structure

Warning: observer functions

Observer function: does not modify data structure
$\Longrightarrow$ Potential difference = 0

Warning: observer functions

Observer function: does not modify data structure
$\Longrightarrow$ Potential difference = 0
$\Longrightarrow$ amortized cost = real cost

Warning: observer functions

Observer function: does not modify data structure

$\Longrightarrow$ Potential difference = 0

$\Longrightarrow$ amortized cost = real cost

$\Longrightarrow$ Must analyze WCC of observer functions

Warning: observer functions

Observer function: does not modify data structure

$\Longrightarrow$ Potential difference = 0

$\Longrightarrow$ amortized cost = real cost

$\Longrightarrow$ Must analyze WCC of observer functions

This makes sense because

Observer functions do not consume their arguments!

Warning: observer functions

Observer function: does not modify data structure

$\Longrightarrow$ Potential difference = 0

$\Longrightarrow$ amortized cost = real cost

$\Longrightarrow$ Must analyze WCC of observer functions

This makes sense because

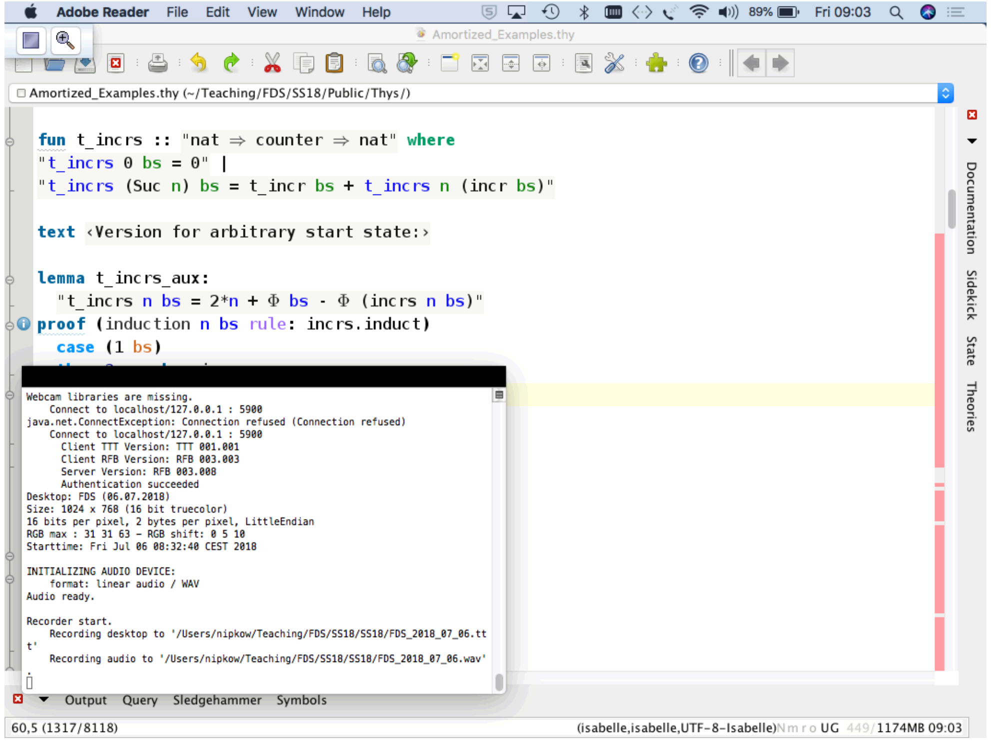
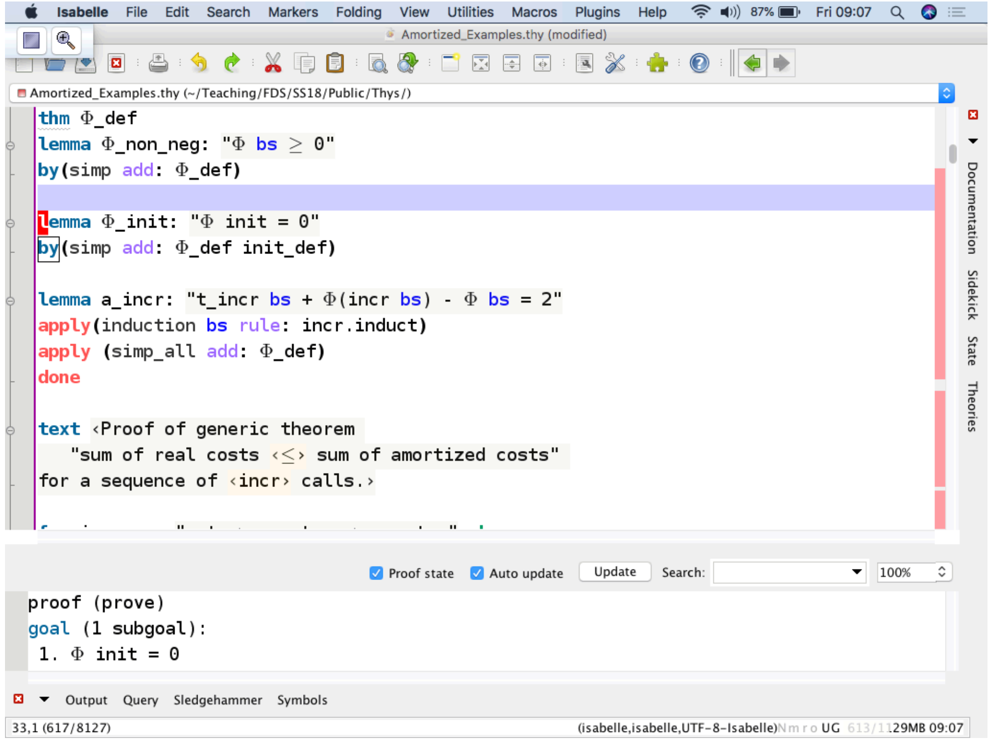
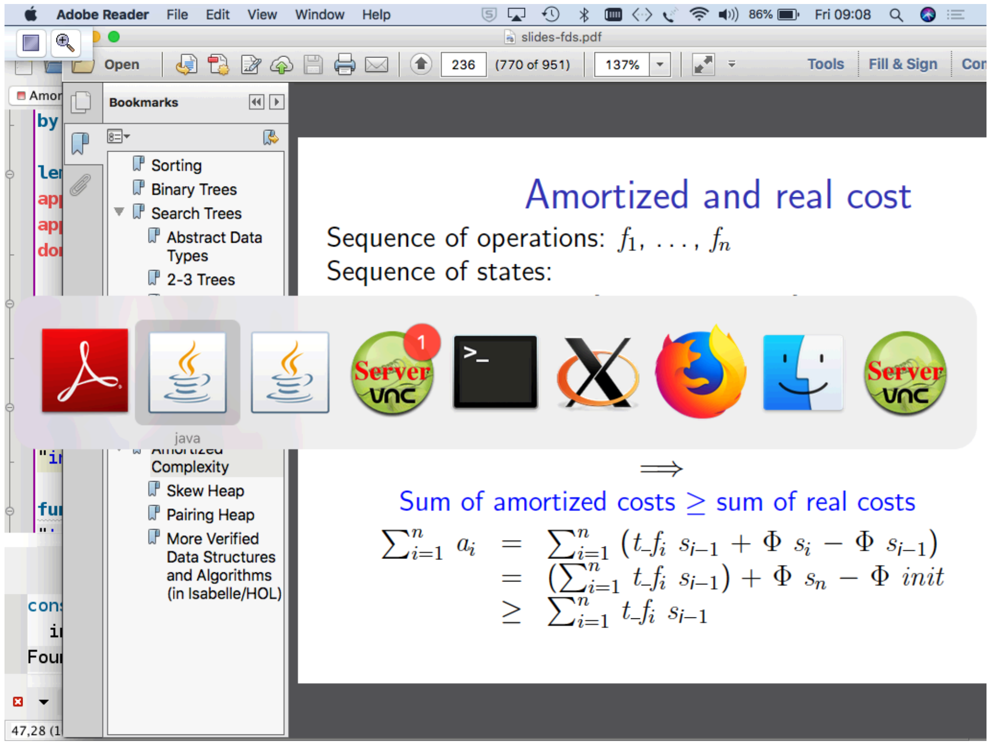
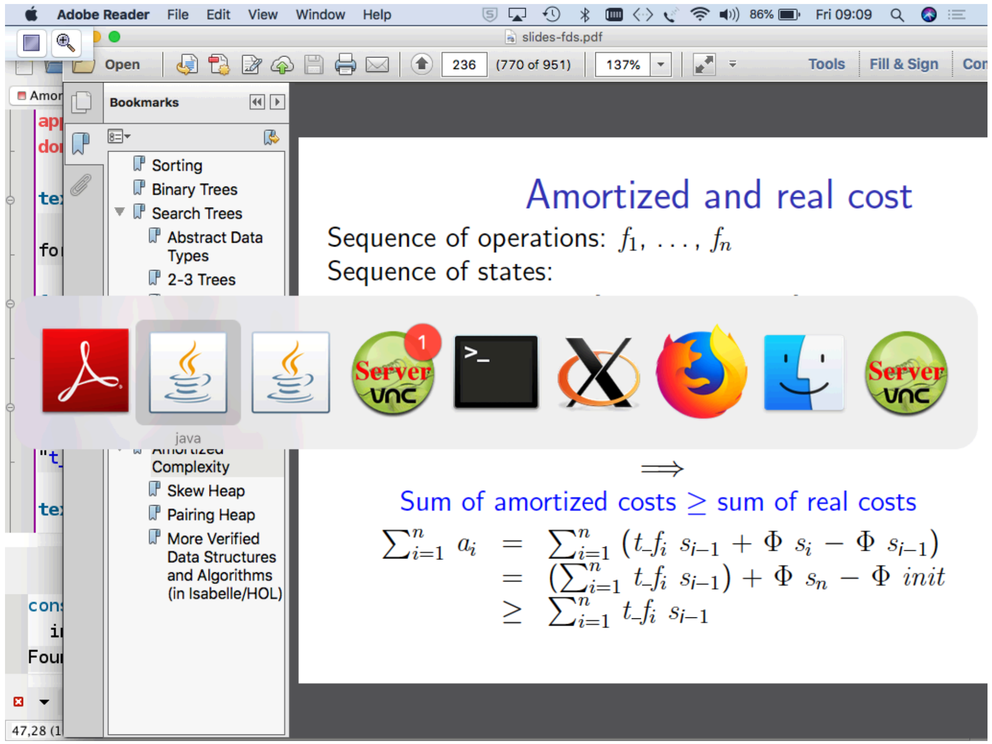
Observer functions do not consume their arguments!

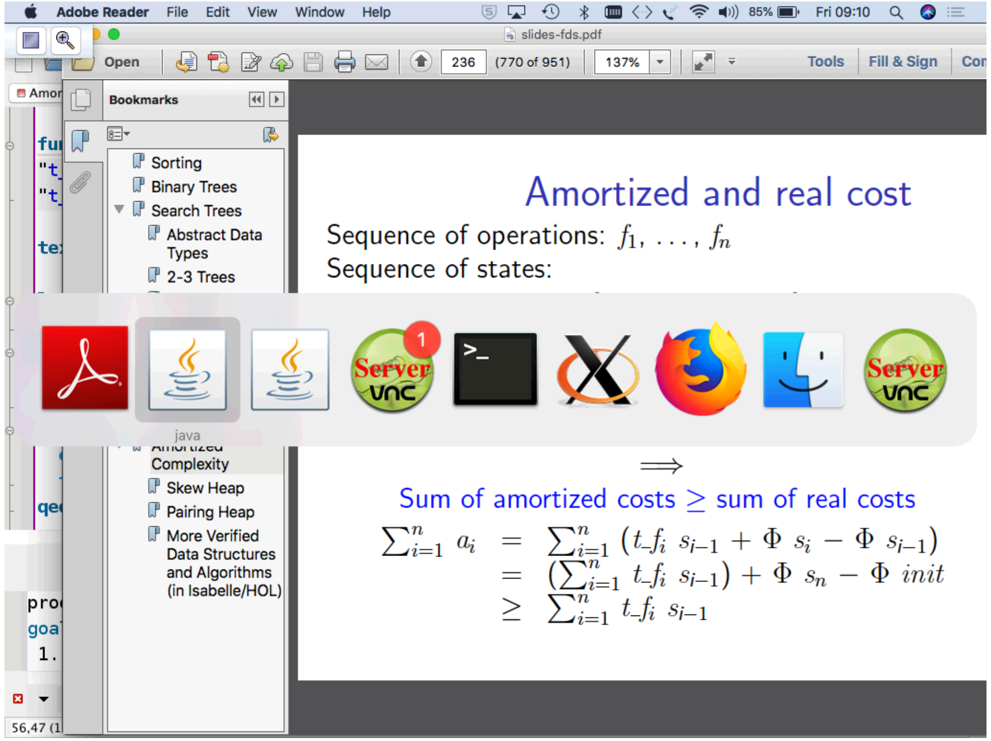
Legal: *let* bad = create unbalanced data structure with high potential;

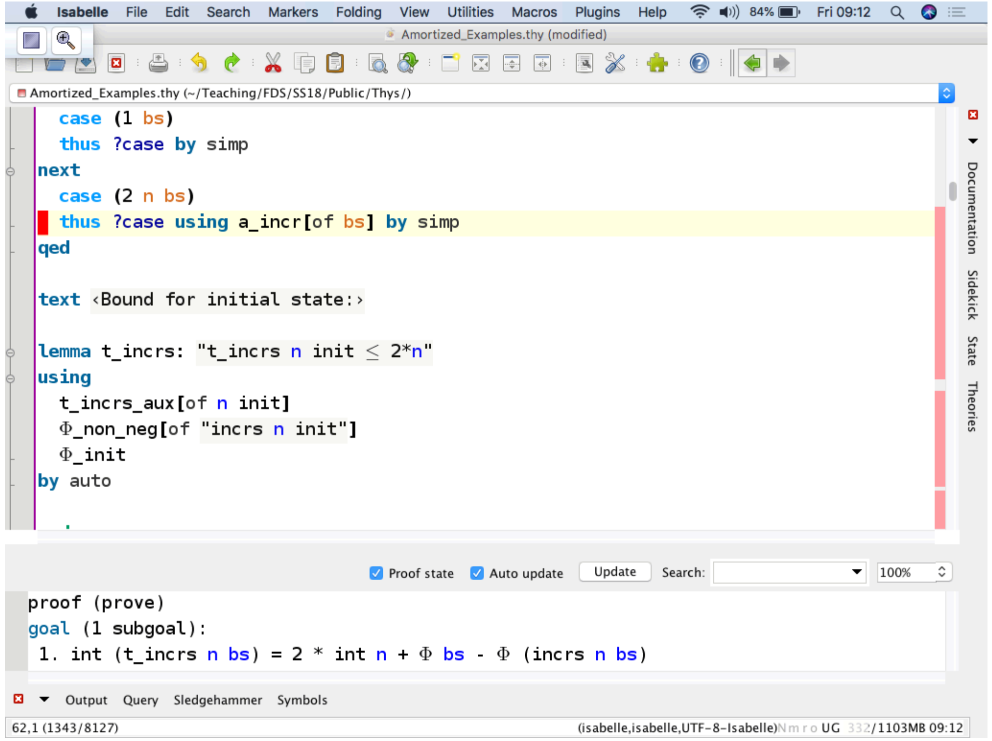
$\_ = observer\ bad;$

$\_ = observer\ bad;$

$\vdots$


Adobe Reader File Edit View Window Help
Amortized_Examples.thy (~/Teaching/FDS/SS18/Public/Thys/)
fun t_incrs :: "nat ⇒ counter ⇒ nat" where
"t_incrs 0 bs = 0" |
"t_incrs (Suc n) bs = t_incr bs + t_incrs n (incr bs)"
text ‹Version for arbitrary start state:›
lemma t_incrs_aux:
"t_incrs n bs = 2*n + Φ bs - Φ (incrs n bs)"
proof (induction n bs rule: incrs.induct)
case (1 bs)
Webcam libraries are missing.
Connect to localhost/127.0.0.1 : 5900
java.net.ConnectException: Connection refused (Connection refused)
Recorder start.
Output Query Sledgehammer Symbols
60,5 (1317/8118)

Isabelle File Edit Search Markers Folding View Utilities Macros Plugins Help
Amortized_Examples.thy (modified)
thm Φ_def
lemma Φ_non_neg: "Φ bs ≥ 0"
by(simp add: Φ_def)
lemma Φ_init: "Φ init = 0"
by(simp add: Φ_def init_def)
lemma a_incr: "t_incr bs + Φ(incr bs) - Φ bs = 2"
apply(induction bs rule: incr.induct)
apply (simp_all add: Φ_def)
done
text ‹Proof of generic theorem
"sum of real costs ‹≤› sum of amortized costs"
for a sequence of ‹incr› calls.›
Proof state Auto update Update Search: 100%
proof (prove)
goal (1 subgoal):
1. Φ init = 0
Output Query Sledgehammer Symbols
33,1 (617/8127)

Adobe Reader File Edit View Window Help
slides-fds.pdf
Open 236 (770 of 951) 137% Tools Fill & Sign
Bookmarks
Sorting
Binary Trees
Search Trees
Abstract Data Types
2-3 Trees
Amortized Complexity
Skew Heap
Pairing Heap
More Verified Data Structures and Algorithms (in Isabelle/HOL)
Amortized and real cost
Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:
java
⟹
Sum of amortized costs ≥ sum of real costs
$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (t\_f_i\ s_{i-1} + \Phi\ s_i - \Phi\ s_{i-1}) = (\sum_{i=1}^{n} t\_f_i\ s_{i-1}) + \Phi\ s_n - \Phi\ init \geq \sum_{i=1}^{n} t\_f_i\ s_{i-1}$
47,28

Adobe Reader File Edit View Window Help
slides-fds.pdf
Open 236 (770 of 951) 137% Tools Fill & Sign
Bookmarks
Sorting
Binary Trees
Search Trees
Abstract Data Types
2-3 Trees
Amortized Complexity
Skew Heap
Pairing Heap
More Verified Data Structures and Algorithms (in Isabelle/HOL)
Amortized and real cost
Sequence of operations: $f_1, \ldots, f_n$
Sequence of states:
java
⟹
Sum of amortized costs ≥ sum of real costs
$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (t\_f_i\ s_{i-1} + \Phi\ s_i - \Phi\ s_{i-1}) = (\sum_{i=1}^{n} t\_f_i\ s_{i-1}) + \Phi\ s_n - \Phi\ init \geq \sum_{i=1}^{n} t\_f_i\ s_{i-1}$
47,28





A *skew heap* is a self-adjusting heap (priority queue)

A *skew heap* is a self-adjusting heap (priority queue)

Functions *insert*, *merge* and *del_min* have amortized logarithmic complexity.

A *skew heap* is a self-adjusting heap (priority queue)

Functions $insert$, $merge$ and del_min have amortized logarithmic complexity.

Functions $insert$ and del_min are defined via $merge$

$merge$

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

$merge$

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

Swap subtrees when descending:

$merge$

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

Swap subtrees when descending:

$merge\ (\langle l_1,\ a_1,\ r_1\rangle =: h_1)\ (\langle l_2,\ a_2,\ r_2\rangle =: h_2) =$
$($if $a_1 \leq a_2$ then $\langle merge\ h_2\ r_1,\ a_1,\ l_1\rangle$
else $\langle merge\ h_1\ r_2,\ a_2,\ l_2\rangle)$

Logarithmic amortized complexity

Theorem

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq 3 * \log_2\ (|t_1|_1 + |t_2|_1) + 1$

Towards the proof

Main proof

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$
$\leq \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1$
$= \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1$
$\leq \log_2\ (|t_1|_1 + |t_2|_1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1$
$\leq \log_2\ (|t_1|_1 + |t_2|_1) + 2 * \log_2\ (|t_1|_1 + |t_2|_1) + 1$
because $\log_2\ x + \log_2\ y \leq 2 * \log_2\ (x + y)$ if $x,y > 0$