Functional Programming and Interactive Theorem Proving

with Isabelle/HOL

Peter Lammich

School of Computer Science
The University of Manchester

2019-8-1

Introduction

Programming and proving correct quicksort

Introduction

Programming and proving correct quicksort

using the Theorem Prover Isabelle/HOL
`https://isabelle.in.tum.de/`

Introduction

Programming and proving correct quicksort

using the Theorem Prover Isabelle/HOL
`https://isabelle.in.tum.de/`

download it and follow this lecture on your laptop!
Lecture Material:
`https://github.com/lammich/MCR_SS_2019_FunProgProve`

Introduction

Programming and proving correct quicksort

using the Theorem Prover Isabelle/HOL
`https://isabelle.in.tum.de/`

download it and follow this lecture on your laptop!
Lecture Material:
`https://github.com/lammich/MCR_SS_2019_FunProgProve`

Relax and enjoy! There is no exam on this lecture!

Short Poll

Raise your hand if you

- Have ever written a computer program

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript
  - in PHP, Python, Ruby, PERL

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript
  - in PHP, Python, Ruby, PERL
  - in Haskell, Scala, OCaml, SML, LISP, Scheme

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript
  - in PHP, Python, Ruby, PERL
  - in Haskell, Scala, OCaml, SML, LISP, Scheme
- Know what quicksort is

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript
  - in PHP, Python, Ruby, PERL
  - in Haskell, Scala, OCaml, SML, LISP, Scheme
- Know what quicksort is
- Know what (structural) induction means

Short Poll

Raise your hand if you

- Have ever written a computer program
  - in C, C++, Java, BASIC, PASCAL
  - in JavaScript
  - in PHP, Python, Ruby, PERL
  - in Haskell, Scala, OCaml, SML, LISP, Scheme
- Know what quicksort is
- Know what (structural) induction means
- Have ever used an interactive theorem prover

Lists

$[]$	Empty list
$a \# l$	List with first element a and then list l

Lists

$[]$	Empty list
$a \# l$	List with first element a and then list l
	Example: $1\#2\#3\#4\#[]$

Lists

$[]$	Empty list
$a \# l$	List with first element a and then list l
	Example: $1\#2\#3\#4\#[]$
	Notation: $[1, 2, 3, 4]$

Lists

$[]$ Empty list

$a \# l$ List with first element a and then list l

Example: $1\#2\#3\#4\#[]$

Notation: $[1, 2, 3, 4]$

$l_1 @ l_2$: concatenate two lists
Example: $[1, 2, 3] @ [4, 5, 6] = [1, 2, 3, 4, 5, 6]$

Append @

How to define @ ?

Append @

How to define @ ? Using only [] and #?

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \ @ \ l_2 =$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \mathbin{@} l_2 = l_2$

Append @

How to define $@$? Using only $[]$ and $\#$?

Case distinction whether first list is empty:

$[] \ @ \ l_2 = l_2$
$(x \ \# \ l_1) \ @ \ l_2 =$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \mathbin{@} l_2 = l_2$
$(x \# l_1) \mathbin{@} l_2 = x \# (l_1 \mathbin{@} l_2)$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \mathbin{@} l_2 = l_2$
$(x \# l_1) \mathbin{@} l_2 = x \# (l_1 \mathbin{@} l_2)$

Example:
$([1,2] \mathbin{@} [3])$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[]\ @\ l_2 = l_2$
$(x\ \#\ l_1)\ @\ l_2 = x\ \#\ (l_1\ @\ l_2)$

Example:
$([1,2]\ @\ [3])\ = (1\ \#\ 2\ \#\ [])\ @\ (3\ \#\ [])$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \ @ \ l_2 = l_2$
$(x \# l_1) \ @ \ l_2 = x \# (l_1 \ @ \ l_2)$

Example:
$([1,2] \ @ \ [3]) \ = (1 \# 2 \# []) \ @ \ (3 \# [])$
$= 1 \# ((2 \# []) \ @ \ (3 \# []))$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] \ @ \ l_2 = l_2$
$(x \ \# \ l_1) \ @ \ l_2 = x \ \# \ (l_1 \ @ \ l_2)$

Example:
$([1,2] \ @ \ [3]) \ = (1 \ \# \ 2 \ \# \ []) \ @ \ (3 \ \# \ [])$
$= 1 \ \# \ ((\ 2 \ \# \ []) \ @ \ (3 \ \# \ []))$
$= 1 \ \# \ 2 \ \# \ ([] \ @ \ (3 \ \# \ []))$

Append @

How to define @ ? Using only [] and #?

Case distinction whether first list is empty:

$[] @ l_2 = l_2$
$(x \# l_1) @ l_2 = x \# (l_1 @ l_2)$

Example:
$([1,2] @ [3]) = (1 \# 2 \# []) @ (3 \# [])$
$= 1 \# ((2 \# []) @ (3 \# []))$
$= 1 \# 2 \# ([] @ (3 \# []))$
$= 1 \# 2 \# 3 \# []$

Filter

Erase all elements not ≤ 4 from a list

Filter

Erase all elements not ≤ 4 from a list

*leq*4 [1,42,7,5,2,6,3]

Filter

Erase all elements not ≤ 4 from a list

$\mathit{leq}4$ [1,42,7,5,2,6,3] $=$ [1,2,3]

Filter

Erase all elements not ≤ 4 from a list

*leq*4 [1,42,7,5,2,6,3] $=$ [1,2,3]

*leq*4 [] $=$

Filter

Erase all elements not ≤ 4 from a list

*leq*4 [1,42,7,5,2,6,3] = [1,2,3]

*leq*4 [] = []

Filter

Erase all elements not ≤ 4 from a list

$leq4\ [1,42,7,5,2,6,3] = [1,2,3]$

$leq4\ [] = []$
$leq4\ (x\#l) =$

Filter

Erase all elements not ≤ 4 from a list

$\mathit{leq}4\ [1,42,7,5,2,6,3] \ = [1,2,3]$

$\mathit{leq}4\ [] \ = []$
$\mathit{leq}4\ (x\#l) \ = \mathit{if}\ x{\leq}4\ \mathit{then}\ x \# \mathit{leq}4\ l\ \mathit{else}\ \mathit{leq}4\ l$

Filter

Erase all elements not ≤ 4 from a list

$leq4\ [1,42,7,5,2,6,3] = [1,2,3]$

$leq4\ [] = []$
$leq4\ (x\#l) = \textit{if } x \leq 4 \textit{ then } x \# leq4\ l \textit{ else } leq4\ l$

$leq4\ [1, 42, 7, 5, 2, 6, 3]$

Filter

Erase all elements not ≤ 4 from a list

*leq*4 [1,42,7,5,2,6,3] = [1,2,3]

*leq*4 [] = []
*leq*4 (*x*#*l*) = *if x*≤4 *then x* # *leq*4 *l else leq*4 *l*

*leq*4 [1, 42, 7, 5, 2, 6, 3]
= 1 # *leq*4 [42, 7, 5, 2, 6, 3]

Filter

Erase all elements not ≤ 4 from a list

$leq4\ [1,42,7,5,2,6,3] = [1,2,3]$

$leq4\ [\,] = [\,]$
$leq4\ (x\#l) = if\ x{\leq}4\ then\ x \# leq4\ l\ else\ leq4\ l$

$leq4\ [1, 42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [7, 5, 2, 6, 3]$

Filter

Erase all elements not ≤ 4 from a list

$leq4\ [1,42,7,5,2,6,3] = [1,2,3]$

$leq4\ [] = []$
$leq4\ (x\#l) = \mathit{if}\ x\leq 4\ \mathit{then}\ x \# leq4\ l\ \mathit{else}\ leq4\ l$

$leq4\ [1, 42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [7, 5, 2, 6, 3]$
$= 1 \# leq4\ [5, 2, 6, 3]$

Filter

Erase all elements not ≤ 4 from a list

$leq4$ [1,42,7,5,2,6,3] $=$ [1,2,3]

$leq4$ [] $=$ []
$leq4$ $(x\#l)$ $=$ *if* $x \leq 4$ *then* x # $leq4$ l *else* $leq4$ l

$leq4$ [1, 42, 7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [42, 7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [5, 2, 6, 3]
$=$ 1 # $leq4$ [2, 6, 3]

Filter

Erase all elements not ≤ 4 from a list

*leq*4 [1,42,7,5,2,6,3] = [1,2,3]

*leq*4 [] = []
*leq*4 (*x*#*l*) = *if x*$\leq$4 *then x* # *leq*4 *l else leq*4 *l*

*leq*4 [1, 42, 7, 5, 2, 6, 3]
= 1 # *leq*4 [42, 7, 5, 2, 6, 3]
= 1 # *leq*4 [7, 5, 2, 6, 3]
= 1 # *leq*4 [5, 2, 6, 3]
= 1 # *leq*4 [2, 6, 3]
= 1 # 2 # *leq*4 [6, 3]

Filter

Erase all elements not ≤ 4 from a list

$leq4\ [1,42,7,5,2,6,3] \ = [1,2,3]$

$leq4\ [] \ = []$
$leq4\ (x\#l) \ = \textit{if } x \leq 4 \textit{ then } x \# leq4\ l \textit{ else } leq4\ l$

$leq4\ [1, 42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [42, 7, 5, 2, 6, 3]$
$= 1 \# leq4\ [7, 5, 2, 6, 3]$
$= 1 \# leq4\ [5, 2, 6, 3]$
$= 1 \# leq4\ [2, 6, 3]$
$= 1 \# 2 \# leq4\ [6, 3]$
$= 1 \# 2 \# leq4\ [3]$

Filter

Erase all elements not ≤ 4 from a list

$leq4$ [1,42,7,5,2,6,3] $=$ [1,2,3]

$leq4 \; [] \; = []$
$leq4 \; (x\#l) \; = if \; x{\leq}4 \; then \; x \# leq4 \; l \; else \; leq4 \; l$

$leq4$ [1, 42, 7, 5, 2, 6, 3]
$= 1 \# leq4$ [42, 7, 5, 2, 6, 3]
$= 1 \# leq4$ [7, 5, 2, 6, 3]
$= 1 \# leq4$ [5, 2, 6, 3]
$= 1 \# leq4$ [2, 6, 3]
$= 1 \# 2 \# leq4$ [6, 3]
$= 1 \# 2 \# leq4$ [3]
$= 1 \# 2 \# 3 \# leq4$ []

Filter

Erase all elements not ≤ 4 from a list

$leq4$ [1,42,7,5,2,6,3] $=$ [1,2,3]

$leq4$ [] $=$ []
$leq4$ $(x\#l)$ $=$ *if* $x \leq 4$ *then* x # $leq4$ l *else* $leq4$ l

$leq4$ [1, 42, 7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [42, 7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [7, 5, 2, 6, 3]
$=$ 1 # $leq4$ [5, 2, 6, 3]
$=$ 1 # $leq4$ [2, 6, 3]
$=$ 1 # 2 # $leq4$ [6, 3]
$=$ 1 # 2 # $leq4$ [3]
$=$ 1 # 2 # 3 # $leq4$ []
$=$ 1 # 2 # 3 # []

Filter

Condition as parameter to function

$filter\ P\ [] = []$
$filter\ P\ (x \# l) = (\textbf{if}\ P\ x\ \textbf{then}\ x \#\ filter\ P\ l\ \textbf{else}\ filter\ P\ l)$

Filter

Condition as parameter to function

$filter\ P\ [] = []$
$filter\ P\ (x \# l) = (\textbf{if}\ P\ x\ \textbf{then}\ x \# filter\ P\ l\ \textbf{else}\ filter\ P\ l)$

$filter\ (\lambda x.\ x \leq (4{::}'a))\ l$: Same as $leq4$

Filter

Condition as parameter to function

$\textit{filter}\ P\ [] = []$
$\textit{filter}\ P\ (x \# l) = (\textbf{if}\ P\ x\ \textbf{then}\ x \#\ \textit{filter}\ P\ l\ \textbf{else}\ \textit{filter}\ P\ l)$

$\textit{filter}\ (\lambda x.\ x \leq (4::'a))\ l$: Same as $\textit{leq}4$

$\lambda x.\ x \leq (4::'a)$ is anonymous function, with parameter x .

`Demo.thy`

Functions

Count

count l x How often does element *x* occur in list *l*

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 =$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

Count

$\mathit{count}\ l\ x$ How often does element x occur in list l

Example: $\mathit{count}\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$\mathit{count}\ []\ x\ =$

Count

$\mathit{count}\ l\ x$ How often does element x occur in list l

Example: $\mathit{count}\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$\mathit{count}\ []\ x\ = 0$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y \# l)\ x\ =$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + count\ l\ x\ \textbf{else}\ count\ l\ x$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + count\ l\ x\ \textbf{else}\ count\ l\ x$

$count\ [2, 2, 1, 2]\ 2$
$=$

Count

$\mathit{count}\ l\ x$ How often does element x occur in list l

Example: $\mathit{count}\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$\mathit{count}\ []\ x\ = 0$
$\mathit{count}\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + \mathit{count}\ l\ x\ \textbf{else}\ \mathit{count}\ l\ x$

$\mathit{count}\ [2, 2, 1, 2]\ 2$
$= 1 + \mathit{count}\ [2, 1, 2]\ 2$
$=$

Count

$\mathit{count}\ l\ x$ How often does element x occur in list l

Example: $\mathit{count}\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$\mathit{count}\ []\ x\ = 0$
$\mathit{count}\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + \mathit{count}\ l\ x\ \textbf{else}\ \mathit{count}\ l\ x$

$\mathit{count}\ [2, 2, 1, 2]\ 2$
$= 1 + \mathit{count}\ [2, 1, 2]\ 2$
$= 1 + 1 + \mathit{count}\ [1, 2]\ 2$
$=$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y\ \#\ l)\ x\ = \mathbf{if}\ x{=}y\ \mathbf{then}\ 1 + count\ l\ x\ \mathbf{else}\ count\ l\ x$

$count\ [2, 2, 1, 2]\ 2$
$= 1 + count\ [2, 1, 2]\ 2$
$= 1 + 1 + count\ [1, 2]\ 2$
$= 1 + 1 + count\ [2]\ 2$
$=$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + count\ l\ x\ \textbf{else}\ count\ l\ x$

$count\ [2, 2, 1, 2]\ 2$
$= 1 + count\ [2, 1, 2]\ 2$
$= 1 + 1 + count\ [1, 2]\ 2$
$= 1 + 1 + count\ [2]\ 2$
$= 1 + 1 + 1 + count\ []\ 2$
$=$

Count

$\mathit{count}\ l\ x$ How often does element x occur in list l

Example: $\mathit{count}\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$\mathit{count}\ []\ x\ = 0$
$\mathit{count}\ (y \# l)\ x\ = \mathbf{if}\ x{=}y\ \mathbf{then}\ 1 + \mathit{count}\ l\ x\ \mathbf{else}\ \mathit{count}\ l\ x$

$\mathit{count}\ [2, 2, 1, 2]\ 2$
$= 1 + \mathit{count}\ [2, 1, 2]\ 2$
$= 1 + 1 + \mathit{count}\ [1, 2]\ 2$
$= 1 + 1 + \mathit{count}\ [2]\ 2$
$= 1 + 1 + 1 + \mathit{count}\ []\ 2$
$= 1 + 1 + 1 + 0$
$=$

Count

$count\ l\ x$ How often does element x occur in list l

Example: $count\ [1, 2, 3, 1, 2, 3, 2, 2]\ 2 = 4$

$count\ []\ x\ = 0$
$count\ (y \# l)\ x\ = \textbf{if}\ x{=}y\ \textbf{then}\ 1 + count\ l\ x\ \textbf{else}\ count\ l\ x$

$count\ [2, 2, 1, 2]\ 2$
$= 1 + count\ [2, 1, 2]\ 2$
$= 1 + 1 + count\ [1, 2]\ 2$
$= 1 + 1 + count\ [2]\ 2$
$= 1 + 1 + 1 + count\ []\ 2$
$= 1 + 1 + 1 + 0$
$= 3$

Sortedness

Is a list sorted? E.g. $\textit{sorted}\ [1,\ 2,\ 4] = \textit{True}$, $\textit{sorted}\ [1,\ 4,\ 3] = \textit{False}$

Sortedness

Is a list sorted? E.g. $\mathit{sorted}\ [1, 2, 4] = \mathit{True}$, $\mathit{sorted}\ [1, 4, 3] = \mathit{False}$

$\mathit{sorted}\ [\,] \ =$

Sortedness

Is a list sorted? E.g. $\textit{sorted}\ [1, 2, 4] = \textit{True}$, $\textit{sorted}\ [1, 4, 3] = \textit{False}$

$\textit{sorted}\ [\,]\ = \textit{True}$

Sortedness

Is a list sorted? E.g. $\mathit{sorted}\ [1, 2, 4] = \mathit{True}$, $\mathit{sorted}\ [1, 4, 3] = \mathit{False}$

$\mathit{sorted}\ [] \ = \mathit{True}$
$\mathit{sorted}\ [x] \ =$

Sortedness

Is a list sorted? E.g. $\mathit{sorted}\ [1, 2, 4] = \mathit{True}$, $\mathit{sorted}\ [1, 4, 3] = \mathit{False}$

$\mathit{sorted}\ []\ = \mathit{True}$
$\mathit{sorted}\ [x]\ = \mathit{True}$

Sortedness

Is a list sorted? E.g. $\mathit{sorted}\ [1, 2, 4] = \mathit{True}$, $\mathit{sorted}\ [1, 4, 3] = \mathit{False}$

$\mathit{sorted}\ [] \ = \mathit{True}$
$\mathit{sorted}\ [x] \ = \mathit{True}$
$\mathit{sorted}\ (x \# y \# l) \ =$

Sortedness

Is a list sorted? E.g. $\textit{sorted}\ [1, 2, 4] = \textit{True}$, $\textit{sorted}\ [1, 4, 3] = \textit{False}$

$\textit{sorted}\ [\,]\ = \textit{True}$
$\textit{sorted}\ [x]\ = \textit{True}$
$\textit{sorted}\ (x \# y \# l)\ = x \le y \wedge \textit{sorted}\ (y \# l)$

Sortedness

Is a list sorted? E.g. $\textit{sorted}\ [1, 2, 4] = \textit{True}$, $\textit{sorted}\ [1, 4, 3] = \textit{False}$

$$\begin{aligned}
&\textit{sorted}\ [\,] = \textit{True}\\
&\textit{sorted}\ [x] = \textit{True}\\
&\textit{sorted}\ (x \mathbin{\#} y \mathbin{\#} l) = x \le y \wedge \textit{sorted}\ (y \mathbin{\#} l)
\end{aligned}$$

Note $\wedge$ means ”and”.

`Demo.thy`

Count and Sortedness

Quicksort

Algorithm to sort a list

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$qs\ [3, 2, 5, 4, 7]$
$=$

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$qs\ [3, 2, 5, 4, 7]$
$= qs\ [2]\ @\ [3]\ @\ qs\ [5, 4, 7]$
$=$

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$qs\ [3, 2, 5, 4, 7]$
$= qs\ [2]\ @\ [3]\ @\ qs\ [5, 4, 7]$
$= [2]\ @\ [3]\ @\ qs\ [4]\ @\ [5]\ @\ qs\ [7]$
$=$

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$qs\ [3, 2, 5, 4, 7]$
$= qs\ [2]\ @\ [3]\ @\ qs\ [5, 4, 7]$
$= [2]\ @\ [3]\ @\ qs\ [4]\ @\ [5]\ @\ qs\ [7]$
$= [2]\ @\ [3]\ @\ [4]\ @\ [5]\ @\ [7]$
$=$

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$qs\ [3, 2, 5, 4, 7]$
$= qs\ [2]\ @\ [3]\ @\ qs\ [5, 4, 7]$
$= [2]\ @\ [3]\ @\ qs\ [4]\ @\ [5]\ @\ qs\ [7]$
$= [2]\ @\ [3]\ @\ [4]\ @\ [5]\ @\ [7]$
$= [2, 3, 4, 5, 7]$

Quicksort

Algorithm to sort a list

Idea:

1. pick pivot element p (e.g. first element of list)
2. partition list into elements $\leq p$ and $> p$
3. recursively sort partitions

$$
\begin{aligned}
& qs\ [3, 2, 5, 4, 7] \\
&= qs\ [2] \ @\ [3] \ @\ qs\ [5, 4, 7] \\
&= [2] \ @\ [3] \ @\ qs\ [4] \ @\ [5] \ @\ qs\ [7] \\
&= [2] \ @\ [3] \ @\ [4] \ @\ [5] \ @\ [7] \\
&= [2, 3, 4, 5, 7]
\end{aligned}
$$

$$
\begin{aligned}
& qs\ [] = [] \\
& qs\ (p \# l) = qs\ (filter\ (\lambda x.\ x \leq p)\ l) \ @\ [p] \ @\ qs\ (filter\ (\lambda x.\ x > p)\ l)
\end{aligned}
$$

`Demo.thy`

Quicksort

Correct Sorting

What does it mean that quicksort is correct?

Correct Sorting

What does it mean that quicksort is correct?

1. The resulting list must be sorted $\mathit{sorted}\ (\mathit{qs}\ l)$

Correct Sorting

What does it mean that quicksort is correct?

1. The resulting list must be sorted $\mathit{sorted}\ (\mathit{qs}\ l)$
2. and must contain same elements $\forall x.\ \mathit{count}\ (\mathit{qs}\ l)\ x = \mathit{count}\ l\ x$

   $\forall x. \ldots$ means "for all x"

`Demo.thy`

Correctness of Sorting

Useful Properties

$count\ (l_1\ @\ l_2)\ x =$

Useful Properties

$$\mathit{count}\ (l_1 \ @\ l_2)\ x = \mathit{count}\ l_1\ x + \mathit{count}\ l_2\ x$$

Useful Properties

$$count\ (l_1\ @\ l_2)\ x = count\ l_1\ x + count\ l_2\ x$$

$$count\ (filter\ P\ l)\ x\ =$$

Useful Properties

$$count\ (l_1\ @\ l_2)\ x = count\ l_1\ x + count\ l_2\ x$$

$$count\ (filter\ P\ l)\ x\ = if\ P\ x\ then\ count\ l\ x\ else\ 0$$

Useful Properties

$$count\ (l_1 \ @\ l_2)\ x = count\ l_1\ x + count\ l_2\ x$$

$$count\ (filter\ P\ l)\ x\ = if\ P\ x\ then\ count\ l\ x\ else\ 0$$

$$count\ (filter\ (\lambda x.\ x \leq p)\ l)\ x + count\ (filter\ (\lambda x.\ x > p)\ l)\ x\ =$$

Useful Properties

$$count\ (l_1\ @\ l_2)\ x = count\ l_1\ x + count\ l_2\ x$$

$$count\ (filter\ P\ l)\ x\ = if\ P\ x\ then\ count\ l\ x\ else\ 0$$

$$count\ (filter\ (\lambda x.\ x \leq p)\ l)\ x + count\ (filter\ (\lambda x.\ x > p)\ l)\ x\ = count\ l\ x$$