Programming in Haskell

Solutions to Exercises

Graham Hutton
University of Nottingham

Contents

Chapter 1 - Introduction	**1**
Chapter 2 - First steps	**3**
Chapter 3 - Types and classes	**4**
Chapter 4 - Defining functions	**5**
Chapter 5 - List comprehensions	**7**
Chapter 6 - Recursive functions	**9**
Chapter 7 - Higher-order functions	**13**
Chapter 8 - Functional parsers	**15**
Chapter 9 - Interactive programs	**19**
Chapter 10 - Declaring types and classes	**21**
Chapter 11 - The countdown problem	**23**
Chapter 12 - Lazy evaluation	**24**
Chapter 13 - Reasoning about programs	**26**

Chapter 1 - Introduction

Exercise 1

$$
\begin{array}{ll}
& double\ (double\ 2) \\
= & \quad \{\ \text{applying the inner } double\ \} \\
& double\ (2+2) \\
= & \quad \{\ \text{applying } double\ \} \\
& (2+2)+(2+2) \\
= & \quad \{\ \text{applying the first } +\ \} \\
& 4+(2+2) \\
= & \quad \{\ \text{applying the second } +\ \} \\
& 4+4 \\
= & \quad \{\ \text{applying } +\ \} \\
& 8
\end{array}
$$

or

$$
\begin{array}{ll}
& double\ (double\ 2) \\
= & \quad \{\ \text{applying the outer } double\ \} \\
& (double\ 2)+(double\ 2) \\
= & \quad \{\ \text{applying the second } double\ \} \\
& (double\ 2)+(2+2) \\
= & \quad \{\ \text{applying the second } +\ \} \\
& (double\ 2)+4 \\
= & \quad \{\ \text{applying } double\ \} \\
& (2+2)+4 \\
= & \quad \{\ \text{applying the first } +\ \} \\
& 4+4 \\
= & \quad \{\ \text{applying } +\ \} \\
& 8
\end{array}
$$

There are a number of other answers too.

Exercise 2

$$
\begin{array}{ll}
& sum\ [x] \\
= & \quad \{\ \text{applying } sum\ \} \\
& x + sum\ [\,] \\
= & \quad \{\ \text{applying } sum\ \} \\
& x+0 \\
= & \quad \{\ \text{applying } +\ \} \\
& x
\end{array}
$$

Exercise 3

(1)

$$
\begin{array}{lcl}
product\ [\,] & = & 1 \\
product\ (x : xs) & = & x * product\ xs
\end{array}
$$

(2)

$$
\begin{aligned}
& \mathit{product}\ [2,3,4] \\
= & \quad \{\text{ applying } \mathit{product} \ \} \\
& 2 * (\mathit{product}\ [3,4]) \\
= & \quad \{\text{ applying } \mathit{product} \ \} \\
& 2 * (3 * \mathit{product}\ [4]) \\
= & \quad \{\text{ applying } \mathit{product} \ \} \\
& 2 * (3 * (4 * \mathit{product}\ [\,])) \\
= & \quad \{\text{ applying } \mathit{product} \ \} \\
& 2 * (3 * (4 * 1)) \\
= & \quad \{\text{ applying } * \ \} \\
& 24
\end{aligned}
$$

Exercise 4

Replace the second equation by

$$\mathit{qsort}\ (x : \mathit{xs}) \quad = \quad \mathit{qsort}\ \mathit{larger} +\!\!+ [x] +\!\!+ \mathit{qsort}\ \mathit{smaller}$$

That is, just swap the occurrences of *smaller* and *larger*.

Exercise 5

Duplicate elements are removed from the sorted list. For example:

$$
\begin{aligned}
& \mathit{qsort}\ [2,2,3,1,1] \\
= & \quad \{\text{ applying } \mathit{qsort} \ \} \\
& \mathit{qsort}\ [1,1] +\!\!+ [2] +\!\!+ \mathit{qsort}\ [3] \\
= & \quad \{\text{ applying } \mathit{qsort} \ \} \\
& (\mathit{qsort}\ [\,] +\!\!+ [1] +\!\!+ \mathit{qsort}\ [\,]) +\!\!+ [2] +\!\!+ (\mathit{qsort}\ [\,] +\!\!+ [3] +\!\!+ \mathit{qsort}\ [\,]) \\
= & \quad \{\text{ applying } \mathit{qsort} \ \} \\
& ([\,] +\!\!+ [1] +\!\!+ [\,]) +\!\!+ [2] +\!\!+ ([\,] +\!\!+ [3] +\!\!+ [\,]) \\
= & \quad \{\text{ applying } +\!\!+ \ \} \\
& [1] +\!\!+ [2] +\!\!+ [3] \\
= & \quad \{\text{ applying } +\!\!+ \ \} \\
& [1,2,3]
\end{aligned}
$$

Chapter 2 - First steps

Exercise 1

$$(2 \uparrow 3) * 4$$
$$(2 * 3) + (4 * 5)$$
$$2 + (3 * (4 \uparrow 5))$$

Exercise 2

No solution required.

Exercise 3

$$\begin{array}{lcl} n & = & a \text{ `}\mathit{div}\text{` } \mathit{length}\ \mathit{xs} \\ & & \quad \textbf{where} \\ & & \qquad a = 10 \\ & & \qquad \mathit{xs} = [1, 2, 3, 4, 5] \end{array}$$

Exercise 4

$$\mathit{last}\ \mathit{xs} \quad = \quad \mathit{head}\ (\mathit{reverse}\ \mathit{xs})$$

or

$$\mathit{last}\ \mathit{xs} \quad = \quad \mathit{xs}\ !!\ (\mathit{length}\ \mathit{xs} - 1)$$

Exercise 5

$$\mathit{init}\ \mathit{xs} \quad = \quad \mathit{take}\ (\mathit{length}\ \mathit{xs} - 1)\ \mathit{xs}$$

or

$$\mathit{init}\ \mathit{xs} \quad = \quad \mathit{reverse}\ (\mathit{tail}\ (\mathit{reverse}\ \mathit{xs}))$$

Chapter 3 - Types and classes

Exercise 1

$$[\mathit{Char}]$$

$$(\mathit{Char}, \mathit{Char}, \mathit{Char})$$

$$[(\mathit{Bool}, \mathit{Char})]$$

$$([\mathit{Bool}], [\mathit{Char}])$$

$$[[a] \to [a]]$$

Exercise 2

$$[a] \to a$$

$$(a, b) \to (b, a)$$

$$a \to b \to (a, b)$$

$$\mathit{Num}\ a \Rightarrow a \to a$$

$$\mathit{Eq}\ a \Rightarrow [a] \to \mathit{Bool}$$

$$(a \to a) \to a \to a$$

Exercise 3

No solution required.

Exercise 4

In general, checking if two functions are equal requires enumerating all possible argument values, and checking if the functions give the same result for each of these values. For functions with a very large (or infinite) number of argument values, such as values of type *Int* or *Integer*, this is not feasible. However, for small numbers of argument values, such as values of type of type *Bool*, it is feasible.

Chapter 4 - Defining functions

Exercise 1

$$halve\ xs \quad = \quad splitAt\ (length\ xs\ `div`\ 2)\ xs$$

or

$$\begin{array}{lcl} halve\ xs & = & (take\ n\ xs, drop\ n\ xs) \\ & & \textbf{where} \\ & & \quad n = length\ xs\ `div`\ 2 \end{array}$$

Exercise 2

(a)

$$safetail\ xs \quad = \quad \textbf{if}\ null\ xs\ \textbf{then}\ [\,]\ \textbf{else}\ tail\ xs$$

(b)

$$\begin{array}{lcl} safetail\ xs \mid null\ xs & = & [\,] \\ \qquad\qquad \mid otherwise & = & tail\ xs \end{array}$$

(c)

$$\begin{array}{lcl} safetail\ [\,] & = & [\,] \\ safetail\ xs & = & tail\ xs \end{array}$$

or

$$\begin{array}{lcl} safetail\ [\,] & = & [\,] \\ safetail\ (\_ : xs) & = & xs \end{array}$$

Exercise 3

(1)

$$\begin{array}{lcl} False \vee False & = & False \\ False \vee True & = & True \\ True \vee False & = & True \\ True \vee True & = & True \end{array}$$

(2)

$$\begin{array}{lcl} False \vee False & = & False \\ \_ \vee \_ & = & True \end{array}$$

(3)

$$\begin{array}{lcl} False \vee b & = & b \\ True \vee \_ & = & True \end{array}$$

(4)

$$\begin{array}{lcl} b \vee c \mid b == c & = & b \\ \qquad \mid otherwise & = & True \end{array}$$

Exercise 4

$$\begin{array}{lll} a \wedge b & = & \textbf{if } a \textbf{ then} \\ & & \quad \textbf{if } b \textbf{ then } \mathit{True} \textbf{ else } \mathit{False} \\ & & \textbf{else} \\ & & \qquad \mathit{False} \end{array}$$

Exercise 5

$$a \wedge b \quad = \quad \textbf{if } a \textbf{ then } b \textbf{ else } \mathit{False}$$

Exercise 6

$$\mathit{mult} \quad = \quad \lambda x \rightarrow (\lambda y \rightarrow (\lambda z \rightarrow x * y * z))$$

Chapter 5 - List comprehensions

Exercise 1

$$sum\ [x \uparrow 2 \mid x \leftarrow [1 \mathinner{..} 100]]$$

Exercise 2

$$replicate\ n\ x \quad = \quad [x \mid \_ \leftarrow [1 \mathinner{..} n]]$$

Exercise 3

$$\begin{array}{lcl} pyths\ n & = & [(x, y, z) \mid x \leftarrow [1 \mathinner{..} n], \\ & & \qquad\qquad\quad y \leftarrow [1 \mathinner{..} n], \\ & & \qquad\qquad\quad z \leftarrow [1 \mathinner{..} n], \\ & & \qquad\qquad\quad x \uparrow 2 + y \uparrow 2 == z \uparrow 2] \end{array}$$

Exercise 4

$$perfects\ n \quad = \quad [x \mid x \leftarrow [1 \mathinner{..} n], sum\ (init\ (factors\ x)) == x]$$

Exercise 5

$$concat\ [[(x, y) \mid y \leftarrow [4, 5, 6]] \mid x \leftarrow [1, 2, 3]]$$

Exercise 6

$$\begin{array}{lcl} positions\ x\ xs & = & find\ x\ (zip\ xs\ [0 \mathinner{..} n]) \\ & & \textbf{where}\ n = length\ xs - 1 \end{array}$$

Exercise 7

$$scalarproduct\ xs\ ys \quad = \quad sum\ [x * y \mid (x, y) \leftarrow zip\ xs\ ys]$$

Exercise 8

$$\begin{array}{lcl} shift & :: & Int \rightarrow Char \rightarrow Char \\ shift\ n\ c \mid isLower\ c & = & int2low\ ((low2int\ c + n)\ `mod`\ 26) \\ \qquad\quad \mid isUpper\ c & = & int2upp\ ((upp2int\ c + n)\ `mod`\ 26) \\ \qquad\quad \mid otherwise & = & c \\ & & \\ freqs & :: & String \rightarrow [Float] \\ freqs\ xs & = & [percent\ (count\ x\ xs')\ n \mid x \leftarrow [\texttt{'a'} \mathinner{..} \texttt{'z'}]] \\ & & \textbf{where} \\ & & \quad xs' = map\ toLower\ xs \\ & & \quad n = letters\ xs \\ & & \\ low2int & :: & Char \rightarrow Int \\ low2int\ c & = & ord\ c - ord\ \texttt{'a'} \end{array}$$

$$
\begin{array}{lcl}
\mathit{int2low} & :: & \mathit{Int} \rightarrow \mathit{Char} \\
\mathit{int2low}\ n & = & \mathit{chr}\ (\mathit{ord}\ \texttt{'a'} + n)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{upp2int} & :: & \mathit{Char} \rightarrow \mathit{Int} \\
\mathit{upp2int}\ c & = & \mathit{ord}\ c - \mathit{ord}\ \texttt{'A'}
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{int2upp} & :: & \mathit{Int} \rightarrow \mathit{Char} \\
\mathit{int2upp}\ n & = & \mathit{chr}\ (\mathit{ord}\ \texttt{'A'} + n)
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{letters} & :: & \mathit{String} \rightarrow \mathit{Int} \\
\mathit{letters}\ \mathit{xs} & = & \mathit{length}\ [x \mid x \leftarrow \mathit{xs}, \mathit{isAlpha}\ x]
\end{array}
$$

Chapter 6 - Recursive functions

Exercise 1

(1)

$$
\begin{array}{lcl}
m \uparrow 0 & = & 1 \\
m \uparrow (n+1) & = & m * m \uparrow n
\end{array}
$$

(2)

$$
\begin{array}{ll}
 & 2 \uparrow 3 \\
= & \quad \{ \text{ applying } \uparrow \text{ } \} \\
 & 2 * (2 \uparrow 2) \\
= & \quad \{ \text{ applying } \uparrow \text{ } \} \\
 & 2 * (2 * (2 \uparrow 1)) \\
= & \quad \{ \text{ applying } \uparrow \text{ } \} \\
 & 2 * (2 * (2 * (2 \uparrow 0))) \\
= & \quad \{ \text{ applying } \uparrow \text{ } \} \\
 & 2 * (2 * (2 * 1)) \\
= & \quad \{ \text{ applying } * \text{ } \} \\
 & 8
\end{array}
$$

Exercise 2

(1)

$$
\begin{array}{ll}
 & \mathit{length}\ [1,2,3] \\
= & \quad \{ \text{ applying } \mathit{length} \text{ } \} \\
 & 1 + \mathit{length}\ [2,3] \\
= & \quad \{ \text{ applying } \mathit{length} \text{ } \} \\
 & 1 + (1 + \mathit{length}\ [3]) \\
= & \quad \{ \text{ applying } \mathit{length} \text{ } \} \\
 & 1 + (1 + (1 + \mathit{length}\ [\,])) \\
= & \quad \{ \text{ applying } \mathit{length} \text{ } \} \\
 & 1 + (1 + (1 + 0)) \\
= & \quad \{ \text{ applying } + \text{ } \} \\
 & 3
\end{array}
$$

(2)

$$
\begin{array}{ll}
 & \mathit{drop}\ 3\ [1,2,3,4,5] \\
= & \quad \{ \text{ applying } \mathit{drop} \text{ } \} \\
 & \mathit{drop}\ 2\ [2,3,4,5] \\
= & \quad \{ \text{ applying } \mathit{drop} \text{ } \} \\
 & \mathit{drop}\ 1\ [3,4,5] \\
= & \quad \{ \text{ applying } \mathit{drop} \text{ } \} \\
 & \mathit{drop}\ 0\ [4,5] \\
= & \quad \{ \text{ applying } \mathit{drop} \text{ } \} \\
 & [4,5]
\end{array}
$$

(3)

$$
\begin{array}{ll}
 & \mathit{init}\ [1,2,3] \\
= & \quad \{\ \text{applying } \mathit{init}\ \} \\
 & 1 : \mathit{init}\ [2,3] \\
= & \quad \{\ \text{applying } \mathit{init}\ \} \\
 & 1 : 2 : \mathit{init}\ [3] \\
= & \quad \{\ \text{applying } \mathit{init}\ \} \\
 & 1 : 2 : [\,] \\
= & \quad \{\ \text{list notation}\ \} \\
 & [1,2]
\end{array}
$$

Exercise 3

$$
\begin{array}{lll}
\mathit{and}\ [\,] & = & \mathit{True} \\
\mathit{and}\ (b : bs) & = & b \wedge \mathit{and}\ bs \\
\\
\mathit{concat}\ [\,] & = & [\,] \\
\mathit{concat}\ (xs : xss) & = & xs +\!\!+\ \mathit{concat}\ xss \\
\\
\mathit{replicate}\ 0\ \_ & = & [\,] \\
\mathit{replicate}\ (n+1)\ x & = & x : \mathit{replicate}\ n\ x \\
\\
(x : \_)\ !!\ 0 & = & x \\
(\_ : xs)\ !!\ (n+1) & = & xs\ !!\ n \\
\\
\mathit{elem}\ x\ [\,] & = & \mathit{False} \\
\mathit{elem}\ x\ (y : ys)\ \mid x == y & = & \mathit{True} \\
\qquad\qquad\qquad\ \mid \mathit{otherwise} & = & \mathit{elem}\ x\ ys
\end{array}
$$

Exercise 4

$$
\begin{array}{lll}
\mathit{merge}\ [\,]\ ys & = & ys \\
\mathit{merge}\ xs\ [\,] & = & xs \\
\mathit{merge}\ (x : xs)\ (y : ys) & = & \textbf{if}\ x \leq y\ \textbf{then} \\
 & & \quad x : \mathit{merge}\ xs\ (y : ys) \\
 & & \textbf{else} \\
 & & \quad\ y : \mathit{merge}\ (x : xs)\ ys
\end{array}
$$

Exercise 5

$$
\begin{array}{lll}
\mathit{halve}\ xs & = & \mathit{splitAt}\ (\mathit{length}\ xs\ \text{`}\mathit{div}\text{`}\ 2)\ xs \\
\\
\mathit{msort}\ [\,] & = & [\,] \\
\mathit{msort}\ [x] & = & [x] \\
\mathit{msort}\ xs & = & \mathit{merge}\ (\mathit{msort}\ ys)\ (\mathit{msort}\ zs) \\
 & & \textbf{where}\ (ys, zs) = \mathit{halve}\ xs
\end{array}
$$

Exercise 6.1

Step 1: define the type

$$sum \quad :: \quad [Int] \to Int$$

Step 2: enumerate the cases

$$\begin{array}{lll} sum\ [\,] & = & \\ sum\ (x : xs) & = & \end{array}$$

Step 3: define the simple cases

$$\begin{array}{lll} sum\ [\,] & = & 0 \\ sum\ (x : xs) & = & \end{array}$$

Step 4: define the other cases

$$\begin{array}{lll} sum\ [\,] & = & 0 \\ sum\ (x : xs) & = & x + sum\ xs \end{array}$$

Step 5: generalise and simplify

$$\begin{array}{lll} sum & :: & Num\ a \Rightarrow [a] \to a \\ sum & = & foldr\ (+)\ 0 \end{array}$$

Exercise 6.2

Step 1: define the type

$$take \quad :: \quad Int \to [a] \to [a]$$

Step 2: enumerate the cases

$$\begin{array}{lll} take\ 0\ [\,] & = & \\ take\ 0\ (x : xs) & = & \\ take\ (n+1)\ [\,] & = & \\ take\ (n+1)\ (x : xs) & = & \end{array}$$

Step 3: define the simple cases

$$\begin{array}{lll} take\ 0\ [\,] & = & [\,] \\ take\ 0\ (x : xs) & = & [\,] \\ take\ (n+1)\ [\,] & = & [\,] \\ take\ (n+1)\ (x : xs) & = & \end{array}$$

Step 4: define the other cases

$$\begin{array}{lll} take\ 0\ [\,] & = & [\,] \\ take\ 0\ (x : xs) & = & [\,] \\ take\ (n+1)\ [\,] & = & [\,] \\ take\ (n+1)\ (x : xs) & = & x : take\ n\ xs \end{array}$$

Step 5: generalise and simplify

$$\begin{array}{lll} take & :: & Int \to [a] \to [a] \\ take\ 0\ \_ & = & [\,] \\ take\ (n+1)\ [\,] & = & [\,] \\ take\ (n+1)\ (x : xs) & = & x : take\ n\ xs \end{array}$$

Exercise 6.3

Step 1: define the type

$$\mathit{last} \quad :: \quad [\,a\,] \rightarrow [\,a\,]$$

Step 2: enumerate the cases

$$\mathit{last}\ (x : \mathit{xs}) \quad =$$

Step 3: define the simple cases

$$\begin{array}{llll} \mathit{last}\ (x : \mathit{xs}) & \mid \mathit{null}\ \mathit{xs} & = & x \\ & \mid \mathit{otherwise} & = & \end{array}$$

Step 4: define the other cases

$$\begin{array}{llll} \mathit{last}\ (x : \mathit{xs}) & \mid \mathit{null}\ \mathit{xs} & = & x \\ & \mid \mathit{otherwise} & = & \mathit{last}\ \mathit{xs} \end{array}$$

Step 5: generalise and simplify

$$\begin{array}{lll} \mathit{last} & :: & [\,a\,] \rightarrow [\,a\,] \\ \mathit{last}\ [\,x\,] & = & x \\ \mathit{last}\ (\_ : \mathit{xs}) & = & \mathit{last}\ \mathit{xs} \end{array}$$

Chapter 7 - Higher-order functions

Exercise 1

$$map\ f\ (filter\ p\ xs)$$

Exercise 2

$$\begin{array}{lcl}
all\ p & = & and \circ map\ p \\[4pt]
any\ p & = & or \circ map\ p \\[4pt]
takeWhile\ _\ [\,] & = & [\,] \\
takeWhile\ p\ (x : xs) & & \\
\quad \mid p\ x & = & x : takeWhile\ p\ xs \\
\quad \mid otherwise & = & [\,] \\[4pt]
dropWhile\ _\ [\,] & = & [\,] \\
dropWhile\ p\ (x : xs) & & \\
\quad \mid p\ x & = & dropWhile\ p\ xs \\
\quad \mid otherwise & = & x : xs
\end{array}$$

Exercise 3

$$\begin{array}{lcl}
map\ f & = & foldr\ (\lambda x\ xs \rightarrow f\ x : xs)\ [\,] \\[4pt]
filter\ p & = & foldr\ (\lambda x\ xs \rightarrow \mathbf{if}\ p\ x\ \mathbf{then}\ x : xs\ \mathbf{else}\ xs)\ [\,]
\end{array}$$

Exercise 4

$$dec2nat \quad = \quad foldl\ (\lambda x\ y \rightarrow 10 * x + y)\ 0$$

Exercise 5

The functions being composed do not all have the same types. For example:

$$\begin{array}{lcl}
sum & :: & [Int] \rightarrow Int \\[4pt]
map\ (\uparrow 2) & :: & [Int] \rightarrow [Int] \\[4pt]
filter\ even & :: & [Int] \rightarrow [Int]
\end{array}$$

Exercise 6

$$\begin{array}{lcl}
curry & :: & ((a, b) \rightarrow c) \rightarrow (a \rightarrow b \rightarrow c) \\
curry\ f & = & \lambda x\ y \rightarrow f\ (x, y) \\[4pt]
uncurry & :: & (a \rightarrow b \rightarrow c) \rightarrow ((a, b) \rightarrow c) \\
uncurry\ f & = & \lambda (x, y) \rightarrow f\ x\ y
\end{array}$$

Exercise 7

$$
\begin{aligned}
\mathit{chop8} &= \mathit{unfold}\ \mathit{null}\ (\mathit{take}\ 8)\ (\mathit{drop}\ 8) \\
\mathit{map}\ f &= \mathit{unfold}\ \mathit{null}\ (f \circ \mathit{head})\ \mathit{tail} \\
\mathit{iterate}\ f &= \mathit{unfold}\ (\mathit{const}\ \mathit{False})\ \mathit{id}\ f
\end{aligned}
$$

Exercise 8

$$
\begin{array}{lll}
\mathit{encode} & :: & \mathit{String} \to [\mathit{Bit}] \\
\mathit{encode} & = & \mathit{concat} \circ \mathit{map}\ (\mathit{addparity} \circ \mathit{make8} \circ \mathit{int2bin} \circ \mathit{ord}) \\
\\
\mathit{decode} & :: & [\mathit{Bit}] \to \mathit{String} \\
\mathit{decode} & = & \mathit{map}\ (\mathit{chr} \circ \mathit{bin2int} \circ \mathit{checkparity}) \circ \mathit{chop9} \\
\\
\mathit{addparity} & :: & [\mathit{Bit}] \to [\mathit{Bit}] \\
\mathit{addparity}\ \mathit{bs} & = & (\mathit{parity}\ \mathit{bs}) : \mathit{bs} \\
\\
\mathit{parity} & :: & [\mathit{Bit}] \to \mathit{Bit} \\
\mathit{parity}\ \mathit{bs} \mid \mathit{odd}\ (\mathit{sum}\ \mathit{bs}) & = & 1 \\
\qquad\qquad \mid \mathit{otherwise} & = & 0 \\
\\
\mathit{chop9} & :: & [\mathit{Bit}] \to [[\mathit{Bit}]] \\
\mathit{chop9}\ [\,] & = & [\,] \\
\mathit{chop9}\ \mathit{bits} & = & \mathit{take}\ 9\ \mathit{bits} : \mathit{chop9}\ (\mathit{drop}\ 9\ \mathit{bits}) \\
\\
\mathit{checkparity} & :: & [\mathit{Bit}] \to [\mathit{Bit}] \\
\mathit{checkparity}\ (b : \mathit{bs}) & & \\
\quad \mid b == \mathit{parity}\ \mathit{bs} & = & \mathit{bs} \\
\quad \mid \mathit{otherwise} & = & \mathit{error}\ \texttt{"parity mismatch"}
\end{array}
$$

Exercise 9

No solution required.

Chapter 8 - Functional parsers

Exercise 1

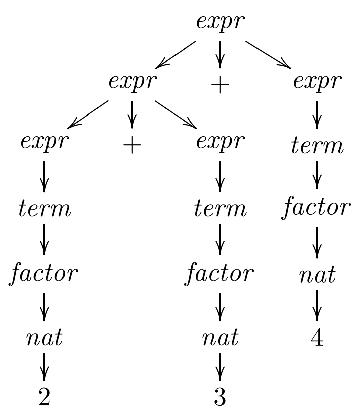
$$
\begin{array}{lcl}
\mathit{int} & = & \textbf{do}\ \mathit{char}\ \texttt{'-'} \\
 & & \quad n \leftarrow \mathit{nat} \\
 & & \quad \mathit{return}\ (-n) \\
 & & +\!\!+\!\!+\, \mathit{nat}
\end{array}
$$

Exercise 2

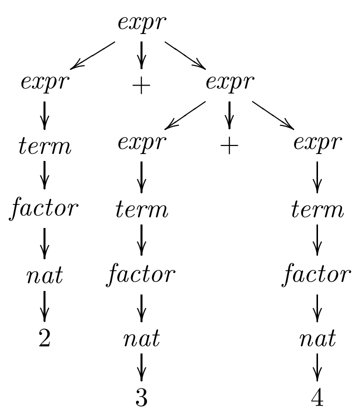
$$
\begin{array}{lcl}
\mathit{comment} & = & \textbf{do}\ \mathit{string}\ \texttt{"--"} \\
 & & \qquad \mathit{many}\ (\mathit{sat}\ (\neq \texttt{'\textbackslash n'})) \\
 & & \qquad \mathit{return}\ ()
\end{array}
$$

Exercise 3

(1)

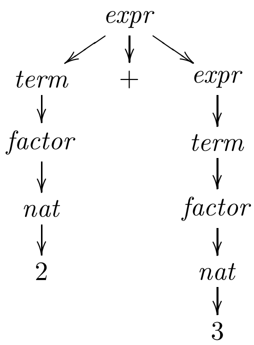


(2)

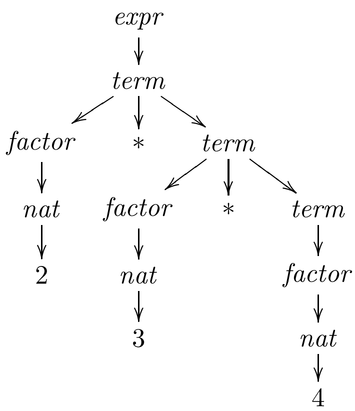


Exercise 4

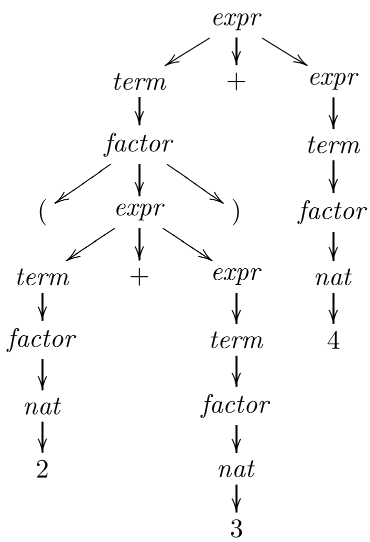
(1)



(2)



(3)



Exercise 5

Without left-factorising the grammar, the resulting parser would backtrack excessively and have exponential time complexity in the size of the expression. For example, a number would be parsed four times before being recognised as an expression.

Exercise 6

```
expr = do t ← term
          do symbol "+"
             e ← expr
             return (t + e)
           +++ do symbol "-"
                  e ← expr
                  return (t − e)
           +++ return t

term = do f ← factor
          do symbol "*"
             t ← term
             return (f ∗ t)
           +++ do symbol "/"
                  t ← term
                  return (f `div` t)
           +++ return f
```

Exercise 7

(1)

$$
\begin{aligned}
\mathit{factor} &::= \mathit{atom}\ (\uparrow \mathit{factor} \mid \mathit{epsilon}) \\
\mathit{atom} &::= (\mathit{expr}) \mid \mathit{nat}
\end{aligned}
$$

(2)

```
factor :: Parser Int
factor = do a ← atom
            do symbol "^"
               f ← factor
               return (a ↑ f)
             +++ return a

atom :: Parser Int
atom = do symbol "("
          e ← expr
          symbol ")"
          return e
        +++ natural
```

Exercise 8

(a)

$$\begin{array}{lcl} \mathit{expr} & ::= & \mathit{expr} - \mathit{nat} \mid \mathit{nat} \\ \mathit{nat} & ::= & 0 \mid 1 \mid 2 \mid \cdots \end{array}$$

(b)

$$\begin{array}{lcl} \mathit{expr} & = & \textbf{do}\ e \leftarrow \mathit{expr} \\ & & \quad\ \ \mathit{symbol}\ \texttt{"-"} \\ & & \quad\ \ n \leftarrow \mathit{natural} \\ & & \quad\ \ \mathit{return}\ (e - n) \\ & & \ \ +\!\!+\!\!+\ \mathit{natural} \end{array}$$

(c)

The parser loops forever without producing a result, because the first operation it performs is to call itself recursively.

(d)

$$\begin{array}{lcl} \mathit{expr} & = & \textbf{do}\ n \leftarrow \mathit{natural} \\ & & \quad\ \ \mathit{ns} \leftarrow \mathit{many}\ (\textbf{do}\ \mathit{symbol}\ \texttt{"-"} \\ & & \qquad\qquad\qquad\qquad\ \ \mathit{natural}) \\ & & \quad\ \ \mathit{return}\ (\mathit{foldl}\ (-)\ n\ \mathit{ns}) \end{array}$$

Chapter 9 - Interactive programs

Exercise 1

$$
\begin{array}{lll}
\mathit{readLine} & = & \mathit{get}\ \texttt{""} \\[1ex]
\mathit{get}\ \mathit{xs} & = & \textbf{do}\ x \leftarrow \mathit{getChar} \\
& & \quad \textbf{case}\ x\ \textbf{of} \\
& & \qquad \texttt{'\textbackslash n'} \rightarrow \mathit{return}\ \mathit{xs} \\
& & \qquad \texttt{'\textbackslash DEL'} \rightarrow \textbf{if}\ \mathit{null}\ \mathit{xs}\ \textbf{then} \\
& & \qquad\qquad\qquad\quad \mathit{get}\ \mathit{xs} \\
& & \qquad\qquad\qquad \textbf{else} \\
& & \qquad\qquad\qquad\quad \textbf{do}\ \mathit{putStr}\ \texttt{"\textbackslash ESC[1D \textbackslash ESC[1D"} \\
& & \qquad\qquad\qquad\qquad \mathit{get}\ (\mathit{init}\ \mathit{xs}) \\
& & \qquad \_ \rightarrow \mathit{get}\ (\mathit{xs} + [x])
\end{array}
$$

Exercise 2

No solution available.

Exercise 3

No solution available.

Exercise 4

No solution available.

Exercise 5

No solution available.

Exercise 6

$$
\begin{array}{lll}
\textbf{type}\ \mathit{Board} & = & [\mathit{Int}] \\[1ex]
\mathit{initial} & :: & \mathit{Board} \\
\mathit{initial} & = & [5,4,3,2,1] \\[1ex]
\mathit{finished} & :: & \mathit{Board} \rightarrow \mathit{Bool} \\
\mathit{finished}\ b & = & \mathit{all}\ (== 0)\ b \\[1ex]
\mathit{valid} & :: & \mathit{Board} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Bool} \\
\mathit{valid}\ b\ \mathit{row}\ \mathit{num} & = & b\ !!\ (\mathit{row} - 1) \geq \mathit{num} \\[1ex]
\mathit{move} & :: & \mathit{Board} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \rightarrow \mathit{Board} \\
\mathit{move}\ b\ \mathit{row}\ \mathit{num} & = & [\textbf{if}\ r == \mathit{row}\ \textbf{then}\ n - \mathit{num}\ \textbf{else}\ n \\
& & \quad \mid (r, n) \leftarrow \mathit{zip}\ [1\,..\,5]\ b] \\
\mathit{newline} & :: & \mathit{IO}\ () \\
\mathit{newline} & = & \mathit{putChar}\ \texttt{'\textbackslash n'}
\end{array}
$$

```
putBoard                :: Board → IO ()
putBoard [a, b, c, d, e] = do putRow 1 a
                              putRow 2 b
                              putRow 3 c
                              putRow 4 d
                              putRow 5 e

putRow                  :: Int → Int → IO ()
putRow row num          = do putStr (show row)
                             putStr ": "
                             putStrLn (stars num)

stars                   :: Int → String
stars n                 = concat (replicate n "* ")

getDigit                :: String → IO Int
getDigit prom           = do putStr prom
                             x ← getChar
                             newline
                             if isDigit x then
                               return (ord x − ord '0')
                              else
                                do putStrLn "ERROR: Invalid digit"
                                   getDigit prom

nim                     :: IO ()
nim                     = play initial 1

play                    :: Board → Int → IO ()
play board player       = do newline
                             putBoard board
                             if finished board then
                               do newline
                                  putStr "Player "
                                  putStr (show (next player))
                                  putStrLn " wins!!"
                              else
                                do newline
                                   putStr "Player "
                                   putStrLn (show player)
                                   r ← getDigit "Enter a row number: "
                                   n ← getDigit "Stars to remove : "
                                   if valid board r n then
                                     play (move board r n) (next player)
                                    else
                                      do newline
                                         putStrLn "ERROR: Invalid move"
                                         play board player

next                    :: Int → Int
next 1                  = 2
next 2                  = 1
```

Chapter 10 - Declaring types and classes

Exercise 1

$$\begin{array}{lcl}
mult\ m\ Zero & = & Zero \\
mult\ m\ (Succ\ n) & = & add\ m\ (mult\ m\ n)
\end{array}$$

Exercise 2

$$\begin{array}{lcl}
occurs\ m\ (Leaf\ n) & = & m == n \\
occurs\ m\ (Node\ l\ n\ r) & = & \textbf{case}\ compare\ m\ n\ \textbf{of} \\
 & & \quad LT \rightarrow occurs\ m\ l \\
 & & \quad EQ \rightarrow True \\
 & & \quad GT \rightarrow occurs\ m\ r
\end{array}$$

This version is more efficient because it only requires one comparison for each node, whereas the previous version may require two comparisons.

Exercise 3

$$\begin{array}{lcl}
leaves\ (Leaf\ _) & = & 1 \\
leaves\ (Node\ l\ r) & = & leaves\ l + leaves\ r
\end{array}$$

$$\begin{array}{lcl}
balanced\ (Leaf\ _) & = & True \\
balanced\ (Node\ l\ r) & = & abs\ (leaves\ l - leaves\ r) \leq 1 \\
 & & \wedge\ balanced\ l \wedge balanced\ r
\end{array}$$

Exercise 4

$$\begin{array}{lcl}
halve\ xs & = & splitAt\ (length\ xs\ \text{`}div\text{`}\ 2)\ xs
\end{array}$$

$$\begin{array}{lcl}
balance\ [x] & = & Leaf\ x \\
balance\ xs & = & Node\ (balance\ ys)\ (balance\ zs) \\
 & & \textbf{where}\ (ys, zs) = halve\ xs
\end{array}$$

Exercise 5

$$\begin{array}{lcl}
\textbf{data}\ Prop & = & \cdots \mid Or\ Prop\ Prop \mid Equiv\ Prop\ Prop
\end{array}$$

$$\begin{array}{lcl}
eval\ s\ (Or\ p\ q) & = & eval\ s\ p \vee eval\ s\ q \\
eval\ s\ (Equiv\ p\ q) & = & eval\ s\ p == eval\ s\ q
\end{array}$$

$$\begin{array}{lcl}
vars\ (Or\ p\ q) & = & vars\ p +\!\!+\ vars\ q \\
vars\ (Equiv\ p\ q) & = & vars\ p +\!\!+\ vars\ q
\end{array}$$

Exercise 6

No solution available.

Exercise 7

$$
\begin{array}{lcl}
\textbf{data}\ \mathit{Expr} & = & \mathit{Val}\ \mathit{Int} \mid \mathit{Add}\ \mathit{Expr}\ \mathit{Expr} \mid \mathit{Mult}\ \mathit{Expr}\ \mathit{Expr} \\
\textbf{type}\ \mathit{Cont} & = & [\mathit{Op}] \\
\textbf{data}\ \mathit{Op} & = & \mathit{EVALA}\ \mathit{Expr} \mid \mathit{ADD}\ \mathit{Int} \mid \mathit{EVALM}\ \mathit{Expr} \mid \mathit{MUL}\ \mathit{Int} \\
\mathit{eval} & :: & \mathit{Expr} \rightarrow \mathit{Cont} \rightarrow \mathit{Int} \\
\mathit{eval}\ (\mathit{Val}\ n)\ \mathit{ops} & = & \mathit{exec}\ \mathit{ops}\ n \\
\mathit{eval}\ (\mathit{Add}\ x\ y)\ \mathit{ops} & = & \mathit{eval}\ x\ (\mathit{EVALA}\ y : \mathit{ops}) \\
\mathit{eval}\ (\mathit{Mult}\ x\ y)\ \mathit{ops} & = & \mathit{eval}\ x\ (\mathit{EVALM}\ y : \mathit{ops}) \\
\mathit{exec} & :: & \mathit{Cont} \rightarrow \mathit{Int} \rightarrow \mathit{Int} \\
\mathit{exec}\ [\,]\ n & = & n \\
\mathit{exec}\ (\mathit{EVALA}\ y : \mathit{ops})\ n & = & \mathit{eval}\ y\ (\mathit{ADD}\ n : \mathit{ops}) \\
\mathit{exec}\ (\mathit{ADD}\ n : \mathit{ops})\ m & = & \mathit{exec}\ \mathit{ops}\ (n + m) \\
\mathit{exec}\ (\mathit{EVALM}\ y : \mathit{ops})\ n & = & \mathit{eval}\ y\ (\mathit{MUL}\ n : \mathit{ops}) \\
\mathit{exec}\ (\mathit{MUL}\ n : \mathit{ops})\ m & = & \mathit{exec}\ \mathit{ops}\ (n * m) \\
\mathit{value} & :: & \mathit{Expr} \rightarrow \mathit{Int} \\
\mathit{value}\ e & = & \mathit{eval}\ e\ [\,]
\end{array}
$$

Exercise 8

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Monad}\ \mathit{Maybe}\ \textbf{where} \\
\quad \begin{array}{lcl}
\mathit{return} & :: & a \rightarrow \mathit{Maybe}\ a \\
\mathit{return}\ x & = & \mathit{Just}\ x \\
(\gg\!=) & :: & \mathit{Maybe}\ a \rightarrow (a \rightarrow \mathit{Maybe}\ b) \rightarrow \mathit{Maybe}\ b \\
\mathit{Nothing} \gg\!= \_ & = & \mathit{Nothing} \\
(\mathit{Just}\ x) \gg\!= f & = & f\ x
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Monad}\ [\,]\ \textbf{where} \\
\quad \begin{array}{lcl}
\mathit{return} & :: & a \rightarrow [a] \\
\mathit{return}\ x & = & [x] \\
(\gg\!=) & :: & [a] \rightarrow (a \rightarrow [b]) \rightarrow [b] \\
\mathit{xs} \gg\!= f & = & \mathit{concat}\ (\mathit{map}\ f\ \mathit{xs})
\end{array}
\end{array}
$$

Chapter 11 - The countdown problem

Exercise 1

$$\mathit{choices}\ \mathit{xs} = [\mathit{zs} \mid \mathit{ys} \leftarrow \mathit{subs}\ \mathit{xs}, \mathit{zs} \leftarrow \mathit{perms}\ \mathit{ys}]$$

Exercise 2

$$\begin{array}{lcl}
\mathit{removeone}\ x\ [\,] & = & [\,] \\
\mathit{removeone}\ x\ (y : \mathit{ys}) & & \\
\quad \mid x == y & = & \mathit{ys} \\
\quad \mid \mathit{otherwise} & = & y : \mathit{removeone}\ x\ \mathit{ys}
\end{array}$$

$$\begin{array}{lcl}
\mathit{isChoice}\ [\,]\ _ & = & \mathit{True} \\
\mathit{isChoice}\ (x : \mathit{xs})\ [\,] & = & \mathit{False} \\
\mathit{isChoice}\ (x : \mathit{xs})\ \mathit{ys} & = & \mathit{elem}\ x\ \mathit{ys} \land \mathit{isChoice}\ \mathit{xs}\ (\mathit{removeone}\ x\ \mathit{ys})
\end{array}$$

Exercise 3

It would lead to non-termination, because recursive calls to *exprs* would no longer be guaranteed to reduce the length of the list.

Exercise 4

$> \mathit{length}\ [e \mid \mathit{ns}' \leftarrow \mathit{choices}\ [1, 3, 7, 10, 25, 50], e \leftarrow \mathit{exprs}\ \mathit{ns}]$
33665406

$> \mathit{length}\ [e \mid \mathit{ns}' \leftarrow \mathit{choices}\ [1, 3, 7, 10, 25, 50], e \leftarrow \mathit{exprs}\ \mathit{ns}, \mathit{eval}\ e \neq [\,]]$
4672540

Exercise 5

Modifying the definition of *valid* by

$$\begin{array}{lcl}
\mathit{valid}\ \mathit{Sub}\ x\ y & = & \mathit{True} \\
\mathit{valid}\ \mathit{Div}\ x\ y & = & y \neq 0 \land x\ \text{`}\mathit{mod}\text{`}\ y == 0
\end{array}$$

gives

$> \mathit{length}\ [e \mid \mathit{ns}' \leftarrow \mathit{choices}\ [1, 3, 7, 10, 25, 50], e \leftarrow \mathit{exprs}\ \mathit{ns}', \mathit{eval}\ e \neq [\,]]$
10839369

Exercise 6

No solution available.

Chapter 12 - Lazy evaluation

Exercise 1

(1)

$2 * 3$ is the only redex, and is both innermost and outermost.

(2)

$1 + 2$ and $2 + 3$ are redexes, with $1 + 2$ being innermost.

(3)

$1 + 2$, $2 + 3$ and $\mathit{fst}\ (1 + 2, 2 + 3)$ are redexes, with the first of these being innermost and the last being outermost.

(4)

$2 * 3$ and $(\lambda x \rightarrow 1 + x)\ (2 * 3)$ are redexes, with the first being innermost and the second being outermost.

Exercise 2

Outermost:

$$
\begin{array}{ll}
 & \mathit{fst}\ (1 + 2, 2 + 3) \\
= & \quad \{\ \text{applying } \mathit{fst}\ \} \\
 & 1 + 2 \\
= & \quad \{\ \text{applying } +\ \} \\
 & 3
\end{array}
$$

Innermost:

$$
\begin{array}{ll}
 & \mathit{fst}\ (1 + 2, 2 + 3) \\
= & \quad \{\ \text{applying the first } +\ \} \\
 & \mathit{fst}\ (3, 2 + 3) \\
= & \quad \{\ \text{applying } +\ \} \\
 & \mathit{fst}\ (3, 5) \\
= & \quad \{\ \text{applying } \mathit{fst}\ \} \\
 & 3
\end{array}
$$

Outermost evaluation is preferable because it avoids evaluation of the second argument, and hence takes one less reduction step.

Exercise 3

$$
\begin{array}{ll}
 & \mathit{mult}\ 3\ 4 \\
= & \quad \{\ \text{applying } \mathit{mult}\ \} \\
 & (\lambda x \rightarrow (\lambda y \rightarrow x * y))\ 3\ 4 \\
= & \quad \{\ \text{applying } \lambda x \rightarrow (\lambda y \rightarrow x * y)\ \} \\
 & (\lambda y \rightarrow 3 * y)\ 4 \\
= & \quad \{\ \text{applying } \lambda y \rightarrow 3 * y\ \} \\
 & 3 * 4 \\
= & \quad \{\ \text{applying } *\ \} \\
 & 12
\end{array}
$$

Exercise 4

$$fibs = 0 : 1 : [x + y \mid (x, y) \leftarrow zip\ fibs\ (tail\ fibs)]$$

Exercise 5

(1)

$$fib\ n = fibs\ !!\ n$$

(2)

$$head\ (dropWhile\ (\leq 1000)\ fibs)$$

Exercise 6

$$\begin{array}{lcl}
repeatTree & :: & a \rightarrow Tree\ a \\
repeatTree\ x & = & Node\ t\ x\ t \\
 & & \textbf{where}\ t = repeatTree\ x \\
takeTree & :: & Int \rightarrow Tree\ a \rightarrow Tree\ a \\
takeTree\ 0\ _ & = & Leaf \\
takeTree\ (n+1)\ Leaf & = & Leaf \\
takeTree\ (n+1)\ (Node\ l\ x\ r) & = & Node\ (takeTree\ n\ l)\ x\ (takeTree\ n\ r) \\
replicateTree & :: & Int \rightarrow a \rightarrow Tree\ a \\
replicateTree\ n & = & takeTree\ n \circ repeatTree
\end{array}$$

Chapter 13 - Reasoning about programs

Exercise 1

$$
\begin{array}{lcl}
\mathit{last} & :: & [a] \rightarrow a \\
\mathit{last}\ [x] & = & x \\
\mathit{last}\ (\_ : \mathit{xs}) & = & \mathit{last}\ \mathit{xs}
\end{array}
$$

or

$$
\begin{array}{lcl}
\mathit{init} & :: & [a] \rightarrow [a] \\
\mathit{init}\ [\_] & = & [\,] \\
\mathit{init}\ (x : \mathit{xs}) & = & x : \mathit{init}\ \mathit{xs}
\end{array}
$$

or

$$
\begin{array}{lcl}
\mathit{foldr1} & :: & (a \rightarrow a \rightarrow a) \rightarrow [a] \rightarrow a \\
\mathit{foldr1}\ \_\ [x] & = & x \\
\mathit{foldr1}\ f\ (x : \mathit{xs}) & = & f\ x\ (\mathit{foldr1}\ f\ \mathit{xs})
\end{array}
$$

There are a number of other answers too.

Exercise 2

Base case:

$$
\begin{array}{ll}
 & \mathit{add}\ \mathit{Zero}\ (\mathit{Succ}\ m) \\
= & \quad \{ \text{ applying } \mathit{add} \ \} \\
 & \mathit{Succ}\ m \\
= & \quad \{ \text{ unapplying } \mathit{add} \ \} \\
 & \mathit{Succ}\ (\mathit{add}\ \mathit{Zero}\ m)
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
 & \mathit{add}\ (\mathit{Succ}\ n)\ (\mathit{Succ}\ m) \\
= & \quad \{ \text{ applying } \mathit{add} \ \} \\
 & \mathit{Succ}\ (\mathit{add}\ n\ (\mathit{Succ}\ m)) \\
= & \quad \{ \text{ induction hypothesis } \} \\
 & \mathit{Succ}\ (\mathit{Succ}\ (\mathit{add}\ n\ m)) \\
= & \quad \{ \text{ unapplying } \mathit{add} \ \} \\
 & \mathit{Succ}\ (\mathit{add}\ (\mathit{Succ}\ n)\ m)
\end{array}
$$

Exercise 3

Base case:

$$
\begin{array}{ll}
 & \mathit{add}\ \mathit{Zero}\ m \\
= & \quad \{ \text{ applying } \mathit{add} \ \} \\
 & m \\
= & \quad \{ \text{ property of } \mathit{add} \ \} \\
 & \mathit{add}\ m\ \mathit{Zero}
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
 & add\ (Succ\ n)\ m \\
= & \quad \{\ \text{applying } add\ \} \\
 & Succ\ (add\ n\ m) \\
= & \quad \{\ \text{induction hypothesis}\ \} \\
 & Succ\ (add\ m\ n) \\
= & \quad \{\ \text{property of } add\ \} \\
 & add\ m\ (Succ\ n)
\end{array}
$$

Exercise 4

Base case:

$$
\begin{array}{ll}
 & all\ (== x)\ (replicate\ 0\ x) \\
= & \quad \{\ \text{applying } replicate\ \} \\
 & all\ (== x)\ [\,] \\
= & \quad \{\ \text{applying } all\ \} \\
 & True
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
 & all\ (== x)\ (replicate\ (n+1)\ x) \\
= & \quad \{\ \text{applying } replicate\ \} \\
 & all\ (== x)\ (x : replicate\ n\ x) \\
= & \quad \{\ \text{applying } all\ \} \\
 & x == x \wedge all\ (== x)\ (replicate\ n\ x) \\
= & \quad \{\ \text{applying } ==\ \} \\
 & True \wedge all\ (== x)\ (replicate\ n\ x) \\
= & \quad \{\ \text{applying } \wedge\ \} \\
 & all\ (== x)\ (replicate\ n\ x) \\
= & \quad \{\ \text{induction hypothesis}\ \} \\
 & True
\end{array}
$$

Exercise 5.1

Base case:

$$
\begin{array}{ll}
 & [\,] +\!\!+ [\,] \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
 & [\,]
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
 & (x : xs) +\!\!+ [\,] \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
 & x : (xs +\!\!+ [\,]) \\
= & \quad \{\ \text{induction hypothesis}\ \} \\
 & x : xs
\end{array}
$$

Exercise 5.2

Base case:

$$
\begin{array}{ll}
& [\,] +\!\!+ (ys +\!\!+ zs) \\
= & \quad \{ \text{ applying } +\!\!+ \} \\
& ys +\!\!+ zs \\
= & \quad \{ \text{ unapplying } +\!\!+ \} \\
& ([\,] +\!\!+ ys) +\!\!+ zs
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
& (x : xs) +\!\!+ (ys +\!\!+ zs) \\
= & \quad \{ \text{ applying } +\!\!+ \} \\
& x : (xs +\!\!+ (ys +\!\!+ zs)) \\
= & \quad \{ \text{ induction hypothesis } \} \\
& x : ((xs +\!\!+ ys) +\!\!+ zs) \\
= & \quad \{ \text{ unapplying } +\!\!+ \} \\
& (x : (xs +\!\!+ ys)) +\!\!+ zs \\
= & \quad \{ \text{ unapplying } +\!\!+ \} \\
& ((x : xs) +\!\!+ ys) +\!\!+ zs
\end{array}
$$

Exercise 6

The three auxiliary results are all general properties that may be useful in other contexts, whereas the single auxiliary result is specific to this application.

Exercise 7

Base case:

$$
\begin{array}{ll}
& \mathit{map}\ f\ (\mathit{map}\ g\ [\,]) \\
= & \quad \{ \text{ applying the inner } \mathit{map} \} \\
& \mathit{map}\ f\ [\,] \\
= & \quad \{ \text{ applying } \mathit{map} \} \\
& [\,] \\
= & \quad \{ \text{ unapplying } \mathit{map} \} \\
& \mathit{map}\ (f \circ g)\ [\,]
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
& \mathit{map}\ f\ (\mathit{map}\ g\ (x : xs)) \\
= & \quad \{ \text{ applying the inner } \mathit{map} \} \\
& \mathit{map}\ f\ (g\ x : \mathit{map}\ g\ xs) \\
= & \quad \{ \text{ applying the outer } \mathit{map} \} \\
& f\ (g\ x) : \mathit{map}\ f\ (\mathit{map}\ g\ xs) \\
= & \quad \{ \text{ induction hypothesis } \} \\
& f\ (g\ x) : \mathit{map}\ (f \circ g)\ xs \\
= & \quad \{ \text{ unapplying } \circ \} \\
& (f \circ g)\ x : \mathit{map}\ (f \circ g)\ xs \\
= & \quad \{ \text{ unappling } \mathit{map} \} \\
& \mathit{map}\ (f \circ g)\ (x : xs)
\end{array}
$$

Exercise 8

Base case:

$$\begin{array}{ll} & \mathit{take}\ 0\ \mathit{xs} +\!\!+ \mathit{drop}\ 0\ \mathit{xs} \\ = & \quad \{\ \text{applying } \mathit{take},\ \mathit{drop}\ \} \\ & [\,] +\!\!+ \mathit{xs} \\ = & \quad \{\ \text{applying } +\!\!+\ \} \\ & \mathit{xs} \end{array}$$

Base case:

$$\begin{array}{ll} & \mathit{take}\ (n+1)\ [\,] +\!\!+ \mathit{drop}\ (n+1)\ [\,] \\ = & \quad \{\ \text{applying } \mathit{take},\ \mathit{drop}\ \} \\ & [\,] +\!\!+ [\,] \\ = & \quad \{\ \text{applying } +\!\!+\ \} \\ & [\,] \end{array}$$

Inductive case:

$$\begin{array}{ll} & \mathit{take}\ (n+1)\ (x : \mathit{xs}) +\!\!+ \mathit{drop}\ (n+1)\ (x : \mathit{xs}) \\ = & \quad \{\ \text{applying } \mathit{take},\ \mathit{drop}\ \} \\ & (x : \mathit{take}\ n\ \mathit{xs}) +\!\!+ (\mathit{drop}\ n\ \mathit{xs}) \\ = & \quad \{\ \text{applying } +\!\!+\ \} \\ & x : (\mathit{take}\ n\ \mathit{xs} +\!\!+ \mathit{drop}\ n\ \mathit{xs}) \\ = & \quad \{\ \text{induction hypothesis}\ \} \\ & x : \mathit{xs} \end{array}$$

Exercise 9

Definitions:

$$\begin{array}{lcl} \mathit{leaves}\ (\mathit{Leaf}\ \_) & = & 1 \\ \mathit{leaves}\ (\mathit{Node}\ l\ r) & = & \mathit{leaves}\ l + \mathit{leaves}\ r \\[1ex] \mathit{nodes}\ (\mathit{Leaf}\ \_) & = & 0 \\ \mathit{nodes}\ (\mathit{Node}\ l\ r) & = & 1 + \mathit{nodes}\ l + \mathit{nodes}\ r \end{array}$$

Property:

$$\mathit{leaves}\ t = \mathit{nodes}\ t + 1$$

Base case:

$$\begin{array}{ll} & \mathit{nodes}\ (\mathit{Leaf}\ n) + 1 \\ = & \quad \{\ \text{applying } \mathit{nodes}\ \} \\ & 0 + 1 \\ = & \quad \{\ \text{applying } +\ \} \\ & 1 \\ = & \quad \{\ \text{unapplying } \mathit{leaves}\ \} \\ & \mathit{leaves}\ (\mathit{Leaf}\ n) \end{array}$$

Inductive case:

$$
\begin{array}{ll}
 & \mathit{nodes}\ (\mathit{Node}\ l\ r) + 1 \\
= & \quad \{\ \text{applying } \mathit{nodes}\ \} \\
 & 1 + \mathit{nodes}\ l + \mathit{nodes}\ r + 1 \\
= & \quad \{\ \text{arithmetic}\ \} \\
 & (\mathit{nodes}\ l + 1) + (\mathit{nodes}\ r + 1) \\
= & \quad \{\ \text{induction hypotheses}\ \} \\
 & \mathit{leaves}\ l + \mathit{leaves}\ r \\
= & \quad \{\ \text{unapplying } \mathit{leaves}\ \} \\
 & \mathit{leaves}\ (\mathit{Node}\ l\ r)
\end{array}
$$

Exercise 10

Base case:

$$
\begin{array}{ll}
 & \mathit{comp}'\ (\mathit{Val}\ n)\ c \\
= & \quad \{\ \text{applying } \mathit{comp}'\ \} \\
 & \mathit{comp}\ (\mathit{Val}\ n) +\!\!+\ c \\
= & \quad \{\ \text{applying } \mathit{comp}\ \} \\
 & [\mathit{PUSH}\ n] +\!\!+\ c \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
 & \mathit{PUSH}\ n : c
\end{array}
$$

Inductive case:

$$
\begin{array}{ll}
 & \mathit{comp}'\ (\mathit{Add}\ x\ y)\ c \\
= & \quad \{\ \text{applying } \mathit{comp}'\ \} \\
 & \mathit{comp}\ (\mathit{Add}\ x\ y) +\!\!+\ c \\
= & \quad \{\ \text{applying } \mathit{comp}\ \} \\
 & (\mathit{comp}\ x +\!\!+\ \mathit{comp}\ y +\!\!+\ [\mathit{ADD}]) +\!\!+\ c \\
= & \quad \{\ \text{associativity of } +\!\!+\ \} \\
 & \mathit{comp}\ x +\!\!+\ (\mathit{comp}\ y +\!\!+\ ([\mathit{ADD}] +\!\!+\ c)) \\
= & \quad \{\ \text{applying } +\!\!+\ \} \\
 & \mathit{comp}\ x +\!\!+\ (\mathit{comp}\ y +\!\!+\ (\mathit{ADD} : c)) \\
= & \quad \{\ \text{induction hypothesis for } y\ \} \\
 & \mathit{comp}\ x +\!\!+\ (\mathit{comp}'\ y\ (\mathit{ADD} : c)) \\
= & \quad \{\ \text{induction hypothesis for } x\ \} \\
 & \mathit{comp}'\ x\ (\mathit{comp}'\ y\ (\mathit{ADD} : c))
\end{array}
$$

In conclusion, we obtain:

$$
\begin{array}{lcl}
\mathit{comp}'\ (\mathit{Val}\ n)\ c & = & \mathit{PUSH}\ n : c \\
\mathit{comp}'\ (\mathit{Add}\ x\ y)\ c & = & \mathit{comp}'\ x\ (\mathit{comp}'\ y\ (\mathit{ADD} : c))
\end{array}
$$