Programming in Haskell

Solutions to Exercises

Graham Hutton
University of Nottingham

Contents

Chapter 1 - Introduction

Exercise 1

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

or

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

There are a number of other answers too.

Exercise 2

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

Exercise 3

(1)

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

(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

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

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

Exercise 5

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

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

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

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

Exercise 4

$$last\ xs \quad = \quad head\ (reverse\ xs)$$

or

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

Exercise 5

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

or

$$init\ xs \quad = \quad reverse\ (tail\ (reverse\ xs))$$

Chapter 3 - Types and classes

Exercise 1

$[Char]$

$(Char, Char, Char)$

$[(Bool, Char)]$

$([Bool], [Char])$

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

Exercise 2

$[a] \to a$

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

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

$Num\ a \Rightarrow a \to a$

$Eq\ a \Rightarrow [a] \to Bool$

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

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

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

or

$$\begin{aligned} \mathit{halve}\ \mathit{xs} &= (\mathit{take}\ n\ \mathit{xs}, \mathit{drop}\ n\ \mathit{xs}) \\ &\ \ \mathbf{where} \\ &\ \ \quad n = \mathit{length}\ \mathit{xs}\ \text{`}\mathit{div}\text{`}\ 2 \end{aligned}$$

Exercise 2

(a)

$$\mathit{safetail}\ \mathit{xs} = \mathbf{if}\ \mathit{null}\ \mathit{xs}\ \mathbf{then}\ [\,]\ \mathbf{else}\ \mathit{tail}\ \mathit{xs}$$

(b)

$$\begin{aligned} \mathit{safetail}\ \mathit{xs} \mid \mathit{null}\ \mathit{xs} &= [\,] \\ \mid \mathit{otherwise} &= \mathit{tail}\ \mathit{xs} \end{aligned}$$

(c)

$$\begin{aligned} \mathit{safetail}\ [\,] &= [\,] \\ \mathit{safetail}\ \mathit{xs} &= \mathit{tail}\ \mathit{xs} \end{aligned}$$

or

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

Exercise 3

(1)

$$\begin{aligned} \mathit{False} \lor \mathit{False} &= \mathit{False} \\ \mathit{False} \lor \mathit{True} &= \mathit{True} \\ \mathit{True} \lor \mathit{False} &= \mathit{True} \\ \mathit{True} \lor \mathit{True} &= \mathit{True} \end{aligned}$$

(2)

$$\begin{aligned} \mathit{False} \lor \mathit{False} &= \mathit{False} \\ \_ \lor \_ &= \mathit{True} \end{aligned}$$

(3)

$$\begin{aligned} \mathit{False} \lor b &= b \\ \mathit{True} \lor \_ &= \mathit{True} \end{aligned}$$

(4)

$$\begin{aligned} b \lor c \mid b == c &= b \\ \mid \mathit{otherwise} &= \mathit{True} \end{aligned}$$

Exercise 4

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

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

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

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

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

Exercise 7

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

Exercise 8

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

```
int2low      :: Int → Char
int2low n    =  chr (ord 'a' + n)

upp2int      :: Char → Int
upp2int c    =  ord c − ord 'A'

int2upp      :: Int → Char
int2upp n    =  chr (ord 'A' + n)

letters      :: String → Int
letters xs   =  length [x | x ← xs, isAlpha x]
```

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)

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

Exercise 2

(1)

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

(2)

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

(3)

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

Exercise 3

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

Exercise 4

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

Exercise 5

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

Exercise 6.1

Step 1: define the type

$$sum \; :: \; [Int] \rightarrow Int$$

Step 2: enumerate the cases

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

Step 3: define the simple cases

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

Step 4: define the other cases

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

Step 5: generalise and simplify

$$\begin{aligned} &sum &::&\; Num\; a \Rightarrow [a] \rightarrow a \\ &sum &=&\; foldr\; (+)\; 0 \end{aligned}$$

Exercise 6.2

Step 1: define the type

$$take \; :: \; Int \rightarrow [a] \rightarrow [a]$$

Step 2: enumerate the cases

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

Step 3: define the simple cases

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

Step 4: define the other cases

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

Step 5: generalise and simplify

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

Exercise 6.3

Step 1: define the type

$$last \quad :: \quad [a] \to [a]$$

Step 2: enumerate the cases

$$last\ (x : xs) \quad =$$

Step 3: define the simple cases

$$\begin{aligned} last\ (x : xs) \mid null\ xs &\quad = \quad x \\ \mid otherwise &\quad = \end{aligned}$$

Step 4: define the other cases

$$\begin{aligned} last\ (x : xs) \mid null\ xs &\quad = \quad x \\ \mid otherwise &\quad = \quad last\ xs \end{aligned}$$

Step 5: generalise and simplify

$$\begin{aligned} &last && :: \quad [a] \to [a] \\ &last\ [x] && = \quad x \\ &last\ (\_ : xs) && = \quad last\ xs \end{aligned}$$

Chapter 7 - Higher-order functions

Exercise 1

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

Exercise 2

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

Exercise 3

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

Exercise 4

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

Exercise 5

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

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

Exercise 6

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

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

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

Exercise 9

No solution required.

Chapter 8 - Functional parsers

Exercise 1

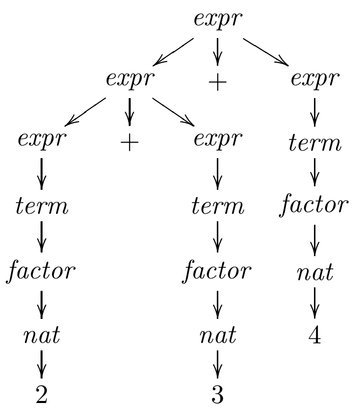
```
int = do char '-'
         n ← nat
         return (−n)
      +++ nat
```

Exercise 2

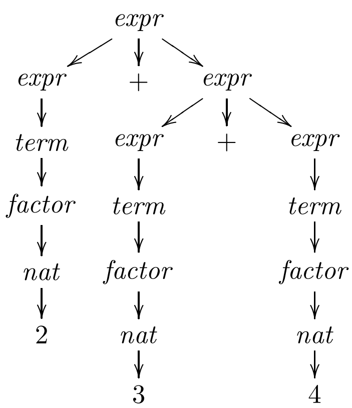
```
comment = do string "--"
             many (sat (≠ '\n'))
             return ()
```

Exercise 3

(1)

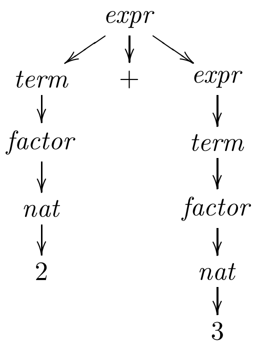


(2)

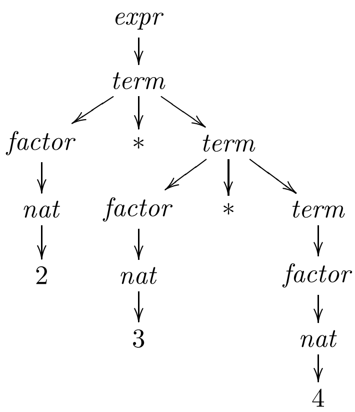


Exercise 4

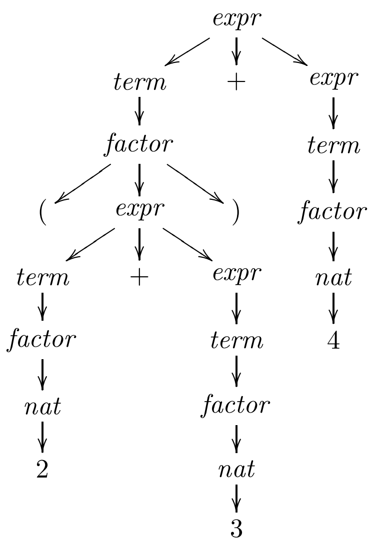
(1)



(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)

```
factor ::= atom (↑ factor | epsilon)

atom   ::= (expr) | nat
```

(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)

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

(b)

```
expr = do e ← expr
          symbol "-"
          n ← natural
          return (e − n)
        +++ natural
```

(c)

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

(d)

```
expr = do n ← natural
          ns ← many (do symbol "-"
                        natural)
          return (foldl (−) n ns)
```

Chapter 9 - Interactive programs

Exercise 1

```
readLine   = get ""

get xs     = do x ← getChar
                case x of
                  '\n' → return xs
                  '\DEL' → if null xs then
                               get xs
                           else
                               do putStr "\ESC[1D \ESC[1D"
                                  get (init xs)
                  _ → get (xs ++ [x])
```

Exercise 2

No solution available.

Exercise 3

No solution available.

Exercise 4

No solution available.

Exercise 5

No solution available.

Exercise 6

```
type Board       = [Int]

initial          :: Board
initial          = [5, 4, 3, 2, 1]

finished         :: Board → Bool
finished b       = all (== 0) b

valid            :: Board → Int → Int → Bool
valid b row num  = b !! (row − 1) ≥ num

move             :: Board → Int → Int → Board
move b row num   = [if r == row then n − num else n
                      | (r, n) ← zip [1 .. 5] b]
newline          :: IO ()
newline          = putChar '\n'
```

```
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

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

Exercise 2

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

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

Exercise 3

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

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

Exercise 4

$$\mathit{halve}\ \mathit{xs} = \mathit{splitAt}\ (\mathit{length}\ \mathit{xs}\ \text{`}\mathit{div}\text{`}\ 2)\ \mathit{xs}$$

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

Exercise 5

$$\textbf{data}\ \mathit{Prop} = \cdots \mid \mathit{Or}\ \mathit{Prop}\ \mathit{Prop} \mid \mathit{Equiv}\ \mathit{Prop}\ \mathit{Prop}$$

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

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

Exercise 6

No solution available.

Exercise 7

```
data Expr              = Val Int | Add Expr Expr | Mult Expr Expr

type Cont              = [Op]

data Op                = EVALA Expr | ADD Int | EVALM Expr | MUL Int

eval                   :: Expr → Cont → Int
eval (Val n) ops       = exec ops n
eval (Add x y) ops     = eval x (EVALA y : ops)
eval (Mult x y) ops    = eval x (EVALM y : ops)

exec                   :: Cont → Int → Int
exec [] n              = n
exec (EVALA y : ops) n = eval y (ADD n : ops)
exec (ADD n : ops) m   = exec ops (n + m)
exec (EVALM y : ops) n = eval y (MUL n : ops)
exec (MUL n : ops) m   = exec ops (n * m)

value                  :: Expr → Int
value e                = eval e []
```

Exercise 8

```
instance Monad Maybe where
  return           :: a → Maybe a
  return x         = Just x

  (≫=)             :: Maybe a → (a → Maybe b) → Maybe b
  Nothing ≫= _     = Nothing
  (Just x) ≫= f    = f x

instance Monad [] where
  return     :: a → [a]
  return x   = [x]

  (≫=)       :: [a] → (a → [b]) → [b]
  xs ≫= f    = concat (map f xs)
```

Chapter 11 - The countdown problem

Exercise 1

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

Exercise 2

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

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

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

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

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

Exercise 5

Modifying the definition of *valid* by

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

gives

$> length\ [e \mid ns' \leftarrow choices\ [1, 3, 7, 10, 25, 50], e \leftarrow exprs\ ns', eval\ e \neq [\,]]$
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 $(\lambda x \to 1 + x)\ (2 * 3)$ 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

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

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

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

Chapter 13 - Reasoning about programs

Exercise 1

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

or

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

or

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

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:

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

Inductive case:

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

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:

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

Inductive case:

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

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:

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

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

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:

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