Appendix A

Appendix

A.1 Haskell support library

$\textbf{infix}\ 5\ \times$
$\textbf{infix}\ 4\ +$

Products

$\langle \cdot, \cdot \rangle :: (a \to b) \to (a \to c) \to a \to (b, c)$
$\langle f, g \rangle\ x = (f\ x, g\ x)$

$(\times) :: (a \to b) \to (c \to d) \to (a, c) \to (b, d)$
$f \times g = \langle f \cdot \pi_1, g \cdot \pi_2 \rangle$

The 0-adic split is the unique function of its type

$(!) :: a \to ()$
$(!) = \underline{()}$

Renamings:

$\pi_1 = \mathsf{fst}$
$\pi_2 = \mathsf{snd}$

Coproduct

Renamings:

$$i_1 = i_1$$
$$i_2 = i_2$$

Either is predefined:

$$(+) :: (a \to b) \to (c \to d) \to a + c \to b + d$$
$$f + g = [i_1 \cdot f, i_2 \cdot g]$$

McCarthy's conditional:

$$p \to f, g = [f, g] \cdot p?$$

Exponentiation

Curry is predefined.

$$ap :: (a \to b, a) \to b$$
$$ap = \widehat{(\$)}$$

Functor:

$$\cdot^{\cdot} :: (b \to c) \to (a \to b) \to a \to c$$
$$f^{\cdot} = \overline{f \cdot ap}$$

Pair to predicate isomorphism (2.99):

$$p2p :: (b, b) \to \mathbb{B} \to b$$
$$p2p\ p\ b = \textbf{if}\ b\ \textbf{then}\ (\mathsf{snd}\ p)\ \textbf{else}\ (\mathsf{fst}\ p)$$

The exponentiation functor is $(a \to)$ predefined:

$$\textbf{instance}\ \mathit{Functor}\ ((\to)\ s)\ \textbf{where}$$
$$\quad \mathit{fmap}\ f\ g = f \cdot g$$

Others

$\underline{\cdot} :: a \to b \to a$ such that $\underline{a}\ x = a$ is predefined. Guards:

$$\cdot? :: (a \to \mathbb{B}) \to a \to a + a$$
$$p?\ x = \textbf{if}\ p\ x\ \textbf{then}\ i_1\ x\ \textbf{else}\ i_2\ x$$

Natural isomorphisms

$\mathsf{swap} :: (a, b) \to (b, a)$
$\mathsf{swap} = \langle \pi_2, \pi_1 \rangle$

$\mathsf{assocr} :: ((a, b), c) \to (a, (b, c))$
$\mathsf{assocr} = \langle \pi_1 \cdot \pi_1, \mathsf{snd} \times id \rangle$

$\mathsf{assocl} :: (a, (b, c)) \to ((a, b), c)$
$\mathsf{assocl} = \langle id \times \pi_1, \pi_2 \cdot \pi_2 \rangle$

$\mathsf{undistr} :: (a, b) + (a, c) \to (a, b + c)$
$\mathsf{undistr} = [id \times i_1, id \times i_2]$

$\mathsf{undistl} :: (b, c) + (a, c) \to (b + a, c)$
$\mathsf{undistl} = [i_1 \times id, i_2 \times id]$

$\mathsf{coswap} :: a + b \to b + a$
$\mathsf{coswap} = [i_2, i_1]$

$\mathsf{coassocr} :: (a + b) + c \to a + (b + c)$
$\mathsf{coassocr} = [id + i_1, i_2 \cdot i_2]$

$\mathsf{coassocl} :: b + (a + c) \to (b + a) + c$
$\mathsf{coassocl} = [i_1 \cdot i_1, i_2 + id]$

$\mathsf{distl} :: (c + a, b) \to (c, b) + (a, b)$
$\mathsf{distl} = \widehat{[\overline{i_1}, \overline{i_2}]}$

$\mathsf{distr} :: (b, c + a) \to (b, c) + (b, a)$
$\mathsf{distr} = (\mathsf{swap} + \mathsf{swap}) \cdot \mathsf{distl} \cdot \mathsf{swap}$

$\mathit{flatr} :: (a, (b, c)) \to (a, b, c)$
$\mathit{flatr}\ (a, (b, c)) = (a, b, c)$

$\mathit{flatl} :: ((a, b), c) \to (a, b, c)$
$\mathit{flatl}\ ((b, c), d) = (b, c, d)$

$br = \langle id, ! \rangle$

$bl = \mathsf{swap} \cdot br$

Class bifunctor

$\textbf{class}\ \mathit{BiFunctor}\ f\ \textbf{where}$
$\quad \mathit{bmap} :: (a \to b) \to (c \to d) \to (f\ \ a\ \ c \to f\ \ b\ \ d)$

$\textbf{instance}\ \mathit{BiFunctor}\ \ \cdot + \cdot\ \ \textbf{where}$
$\quad \mathit{bmap}\ f\ \ g = f + g$

$$\textbf{instance}\ \mathit{BiFunctor}\ (,)\ \textbf{where}$$
$$\quad \mathit{bmap}\ f\ g = f \times g$$

Monads

Kleisli monadic composition:

$$\textbf{infix}\ 4 \bullet$$
$$(\bullet) :: \mathit{Monad}\ a \Rightarrow (b \to a\ c) \to (d \to a\ b) \to d \to a\ c$$
$$(f \bullet g)\ a = (g\ a) \gg\!= f$$

Multiplication, also known as join:

$$\mathit{mult} :: (\mathit{Monad}\ m) \Rightarrow m\ (m\ b) \to m\ b$$
$$\mathit{mult} = (\gg\!= \mathit{id})$$

Monadic binding:

$$\mathit{ap}' :: (\mathit{Monad}\ m) \Rightarrow (a \to m\ b, m\ a) \to m\ b$$
$$\mathit{ap}' = \widehat{\mathit{flip}\ (\gg\!=)}$$

List monad:

$$\mathit{singl} :: a \to [\,a\,]$$
$$\mathit{singl} = \mathsf{return}$$

Strong monads:

$$\textbf{class}\ (\mathit{Functor}\ f, \mathit{Monad}\ f) \Rightarrow \mathit{Strong}\ f\ \textbf{where}$$
$$\quad \mathit{rstr} :: (f\ a, b) \to f\ (a, b)$$
$$\quad \mathit{rstr}\ (x, b) = \textbf{do}\ a \leftarrow x; \mathsf{return}\ (a, b)$$
$$\quad \mathit{lstr} :: (b, f\ a) \to f\ (b, a)$$
$$\quad \mathit{lstr}\ (b, x) = \textbf{do}\ a \leftarrow x; \mathsf{return}\ (b, a)$$
$$\textbf{instance}\ \mathit{Strong}\ \mathsf{IO}$$
$$\textbf{instance}\ \mathit{Strong}\ [\,]$$
$$\textbf{instance}\ \mathit{Strong}\ \mathit{Maybe}$$

Double strength:

$$\mathit{dstr} :: \mathit{Strong}\ m \Rightarrow (m\ a, m\ b) \to m\ (a, b)$$
$$\mathit{dstr} = \mathit{rstr} \bullet \mathit{lstr}$$

Exercise 4.8.13 in Jacobs' "Introduction to Coalgebra" [20]:

$$splitm :: Strong\ \mathsf{F} \Rightarrow \mathsf{F}\ (a \to b) \to a \to \mathsf{F}\ b$$
$$splitm = \overline{fmap\ ap \cdot rstr}$$

Monad transformers:

$$\textbf{class}\ (Monad\ m, Monad\ (t\ m)) \Rightarrow MT\ t\ m\ \textbf{where} \quad \text{-- monad transformer class}$$
$$\quad lift :: m\ a \to t\ m\ a$$

Nested lifting:

$$dlift :: (MT\ t\ (t1\ m), MT\ t1\ m) \Rightarrow m\ a \to t\ (t1\ m)\ a$$
$$dlift = lift \cdot lift$$

Basic functions, abbreviations

$$\mathsf{zero} = \underline{0}$$
$$\mathsf{one} = \underline{1}$$
$$\mathsf{nil} = \underline{[\,]}$$
$$\mathsf{cons} = \widehat{:}$$
$$\mathsf{add} = \widehat{+}$$
$$\mathsf{mul} = \widehat{*}$$
$$\mathsf{conc} = \widehat{++}$$
$$inMaybe :: () + a \to Maybe\ a$$
$$inMaybe = [\underline{\mathsf{Nothing}}, \mathsf{Just}]$$

More advanced

$$\textbf{class}\ (Functor\ f) \Rightarrow Unzipable\ f\ \textbf{where}$$
$$\quad unzp :: f\ (a, b) \to (f\ a, f\ b)$$
$$\quad unzp = \langle fmap\ \pi_1, fmap\ \pi_2 \rangle$$
$$\textbf{class}\ Functor\ g \Rightarrow DistL\ g\ \textbf{where}$$
$$\quad \lambda :: Monad\ m \Rightarrow g\ (m\ a) \to m\ (g\ a)$$
$$\textbf{instance}\ DistL\ [\,]\ \textbf{where}\ \lambda = \mathsf{sequence}$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{DistL}\ \mathit{Maybe}\ \textbf{where} \\
\quad \lambda\ \mathsf{Nothing} = \mathsf{return}\ \mathsf{Nothing} \\
\quad \lambda\ (\mathsf{Just}\ a) = mp\ \mathsf{Just}\ a\ \textbf{where}\ mp\ f = (\mathsf{return} \cdot f) \bullet id
\end{array}
$$

Convert Monad into Applicative:

$$
\begin{array}{l}
aap :: Monad\ m \Rightarrow m\ (a \to b) \to m\ a \to m\ b \\
aap\ mf\ mx = \textbf{do}\ \{f \leftarrow mf; x \leftarrow mx; \mathsf{return}\ (f\ x)\}
\end{array}
$$

A.2 Alloy support library

not given in the current version of this textbook

Bibliography

[1] C. Aarts, R.C. Backhouse, P. Hoogendijk, E.Voermans, and J. van der Woude. A relational theory of datatypes, December 1992. Available from `www.cs.nott.ac.uk/~rcb`.

[2] R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

[3] J. Backus. Can programming be liberated from the von Neumann style? a functional style and its algebra of programs. *CACM*, 21(8):613–639, August 1978.

[4] L.S. Barbosa. *Components as Coalgebras*. University of Minho, December 2001. Ph. D. thesis.

[5] R. Bird. Introduction to Functional Programming. Series in Computer Science. Prentice-Hall International, 2nd edition, 1998. C.A.R. Hoare, series editor.

[6] R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall, 1997.

[7] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *JACM*, 24(1):44–67, January 1977.

[8] M. Erwig and S. Kollmannsberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16:21–34, January 2006.

[9] R.W. Floyd. Assigning meanings to programs. In J.T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19, pages 19–32. American Mathematical Society, 1967. Proc. Symposia in Applied Mathematics.

[10] M.M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, University of Twente, Dept INF, Enschede, The Netherlands, 1992.

[11] J. Gibbons. Kernels, in a nut shell. *JLAMP*, 85(5, Part 2):921–930, 2016.

[12] J. Gibbons and R. Hinze. Just do it: simple monadic equational reasoning. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, ICFP'11, pages 2–14, New York, NY, USA, 2011. ACM.

[13] Jeremy Gibbons, Graham Hutton, and Thorsten Altenkirch. When is a function a fold or an unfold?, 2001. WGP, July 2001 (slides).

[14] A.S. Green, P.L. Lumsdaine, N.J. Ross, P. Selinger, and B. Valiron. An introduction to quantum programming in Quipper. *CoRR*, cs.PL(arXiv:1304.5485v1), 2013.

[15] Ralf Hinze. Adjoint folds and unfolds — an extended study. *Science of Computer Programming*, 78(11):2108–2159, 2013.

[16] Ralf Hinze. Adjoint folds and unfolds — an extended study. *Science of Computer Programming*, 78(11):2108–2159, 2013.

[17] Ralf Hinze, Nicolas Wu, and Jeremy Gibbons. Conjugate hylomorphisms – or: The mother of all structured recursion schemes. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 527–538, New York, NY, USA, 2015. ACM.

[18] P. Hudak. The Haskell School of Expression - Learning Functional Programming Through Multimedia. Cambridge University Press, 1st edition, 2000. ISBN 0-521-64408-9.

[19] Graham Hutton and Erik Meijer. Monadic parsing in Haskell. *Journal of Functional Programming*, 8(4), 1993.

[20] B. Jacobs. *Introduction to Coalgebra. Towards Mathematics of States and Observations*. Cambridge University Press, 2016.

[21] P. Jansson and J. Jeuring. Polylib — a library of polytypic functions. In *Workshop on Generic Programming (*WGP'98*), Marstrand, Sweden*, 1998.

[22] J. Jeuring and P. Jansson. Polytypic programming. In *Advanced Functional Programming*, number 1129 in LNCS, pages 68–114. Springer-Verlag, 1996.

[23] S.L. Peyton Jones. *Haskell 98 Language and Libraries*. Cambridge University Press, Cambridge, UK, 2003. Also published as a Special Issue of the Journal of Functional Programming, 13(1) Jan. 2003.

[24] R. Lämmel and J. Visser. A Strafunski Application Letter. In V. Dahl and P.L. Wadler, editors, *Proc. of Practical Aspects of Declarative Programming (PADL'03)*, volume 2562 of *LNCS*, pages 357–375. Springer-Verlag, January 2003.

[25] S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[26] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14:255–279, 1990.

[27] E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 1986. D. Gries, series editor.

[28] J. McCarthy. Towards a mathematical science of computation. In C.M. Popplewell, editor, *Proc. of* IFIP *62*, pages 21–28, Amsterdam-London, 1963. North-Holland Pub. Company.

[29] E. Meijer and G. Hutton. Bananas in space: Extending fold and unfold to exponential types. In S. Peyton Jones, editor, *Proceedings of Functional Programming Languages and Computer Architecture (FPCA95)*, 1995.

[30] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 14–23. IEEE Computer Society Press, Washington, DC, 1989.

[31] S-C. Mu, Z. Hu, and M. Takeichi. An injective language for reversible computation. In *MPC 2004*, pages 289–313, 2004.

[32] S.C. Mu and R. Bird. Quantum functional programming, 2001. 2nd Asian Workshop on Programming Languages and Systems, KAIST, Dajeaon, Korea, December 17-18, 2001.

[33] D. Murta and J.N. Oliveira. A study of risk-aware program transformation. *SCP*, 110:51–77, 2015.

[34] P. Naur and B. Randell, editors. *Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7th to 11th October 1968*. Scientific Affairs Division, NATO, 1969.

[35] A. Oettinger. The hardware-software complementarity. *Commun. ACM*, 10:604–606, October 1967.

[36] J.N. Oliveira. Towards a linear algebra of programming. *FAoC*, 24(4-6):433–458, 2012.

[37] J.N. Oliveira. Lecture notes on relational methods in software design, 2015. Available from ResearchGate:
`https://www.researchgate.net/profile/Jose_Oliveira34`.

[38] J.N. Oliveira and V.C. Miraldo. “Keep definition, change category” — a practical approach to state-based system calculi. *JLAMP*, 85(4):449–474, 2016.

[39] M.S. Paterson and C.E. Hewitt. Comparative schematology. In *Project MAC Conference on Concurrent Systems and Parallel Computation*, pages 119–127, August 1970.

[40] G. Villavicencio and J.N. Oliveira. *Reverse Program Calculation Supported by Code Slicing* . In *Proceedings of the Eighth Working Conference on Reverse Engineering (*WCRE 2001*) 2-5 October 2001, Stuttgart, Germany*, pages 35–46. IEEE Computer Society, 2001.

[41] P.L. Wadler. Theorems for free! In *4th International Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359, London, Sep. 1989. ACM.