Haskell Quick Reference

COSC 8—Fall 2007

This document summarizes a number of common Haskell functions defined in the standard Prelude. This information is based on Chapter 23 of "The Haskell School of Expression."

Simple List Selector Functions

The functions *head* and *tail* extract the first element and remaining elements, respectively, from a list, which must be nonempty. The functions *last* and *init* are the dual functions that work from the end of a list, rather than from the beginning. The *null* function tests to see if a list is empty.

$$
\begin{aligned}
&\mathit{head} :: [a] \to a\\
&\mathit{head}\ (x : \_) = x\\
&\mathit{head}\ \_ = \mathit{error}\ \texttt{"head: empty list"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{last} :: [a] \to a\\
&\mathit{last}\ [x] = x\\
&\mathit{last}\ (\_ : xs) = \mathit{last}\ xs\\
&\mathit{last}\ [\,] = \mathit{error}\ \texttt{"last: empty list"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{tail} :: [a] \to [a]\\
&\mathit{tail}\ (\_ : xs) = xs\\
&\mathit{tail}\ [\,] = \mathit{error}\ \texttt{"tail: empty list"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{init} :: [a] \to [a]\\
&\mathit{init}\ [x] = [\,]\\
&\mathit{init}\ (x : xs) = x : \mathit{init}\ xs\\
&\mathit{init}\ [\,] = \mathit{error}\ \texttt{"init: empty list"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{null} :: [a] \to \mathit{Bool}\\
&\mathit{null}\ [\,] = \mathit{True}\\
&\mathit{null}\ (\_ : \_) = \mathit{False}
\end{aligned}
$$

Index-Based Selector Functions

To select the mth element from a list, with the first element being the 0th element, we can use the indexing function (!!). The value $\mathit{take}\ n\ xs$ is the prefix of xs of length n, or xs itself if $n > \mathit{length}\ xs$. Similarly, $\mathit{drop}\ n\ xs$ is the suffix of xs after the first n elements, or $[\,]$ if $n > \mathit{length}\ xs$. Finally, $\mathit{splitAt}\ n\ xs$ is equivalent to $(\mathit{take}\ n\ xs, \mathit{drop}\ n\ xs)$.

$$
\begin{aligned}
&(!!) :: [a] \to \mathit{Int} \to a\\
&(x : \_)\ !!\ 0 = x\\
&(\_ : xs)\ !!\ n \mid n > 0 = xs\ !!\ (n-1)\\
&(\_ : \_)\ !!\ \_ = \mathit{error}\ \texttt{"!!: negative index"}\\
&[\,]\ !!\ \_ = \mathit{error}\ \texttt{"!!: index too large"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{take} :: \mathit{Int} \to [a] \to [a]\\
&\mathit{take}\ 0\ \_ = [\,]\\
&\mathit{take}\ \_\ [\,] = [\,]\\
&\mathit{take}\ n\ (x : xs) \mid n > 0 = x : \mathit{take}\ (n-1)\ xs\\
&\mathit{take}\ \_\ \_ = \mathit{error}\ \texttt{"take: negative argument"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{drop} :: \mathit{Int} \to [a] \to [a]\\
&\mathit{drop}\ 0\ xs = xs\\
&\mathit{drop}\ \_\ [\,] = [\,]\\
&\mathit{drop}\ n\ (\_ : xs) \mid n > 0 = \mathit{drop}\ (n-1)\ xs\\
&\mathit{drop}\ \_\ \_ = \mathit{error}\ \texttt{"drop: negative argument"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{splitAt} :: \mathit{Int} \to [a] \to ([a], [a])\\
&\mathit{splitAt}\ 0\ xs = ([\,], xs)\\
&\mathit{splitAt}\ \_\ [\,] = ([\,], [\,])\\
&\mathit{splitAt}\ n\ (x : xs) \mid n > 0 = (x : xs', xs'')\\
&\quad \textbf{where}\ (xs', xs'') = \mathit{splitAt}\ (n-1)\ xs\\
&\mathit{splitAt}\ \_\ \_ = \mathit{error}\ \texttt{"splitAt: negative argument"}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{length} :: [a] \to \mathit{Int}\\
&\mathit{length}\ [\,] = 0\\
&\mathit{length}\ (\_ : xs) = 1 + \mathit{length}\ xs
\end{aligned}
$$

Predicate-Based Selector Functions

The value $\mathit{takeWhile}\ p\ xs$ is the longest (possibly empty) prefix of xs, all of whose elements satisfy the predicate p. The value $\mathit{dropWhile}\ p\ xs$ is the remaining suffix.

$$
\begin{aligned}
&\mathit{takeWhile} :: (a \to \mathit{Bool}) \to [a] \to [a]\\
&\mathit{takeWhile}\ p\ [\,] = [\,]\\
&\mathit{takeWhile}\ p\ (x : xs)\\
&\quad \mid p\ x = x : \mathit{takeWhile}\ p\ xs\\
&\quad \mid \mathit{otherwise} = [\,]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{dropWhile} :: (a \to \mathit{Bool}) \to [a] \to [a]\\
&\mathit{dropWhile}\ p\ [\,] = [\,]\\
&\mathit{dropWhile}\ p\ xs@(x : xs')\\
&\quad \mid p\ x = \mathit{dropWhile}\ p\ xs'\\
&\quad \mid \mathit{otherwise} = xs
\end{aligned}
$$

The function $span\ p\ xs$ is equivalent to $(takeWhile\ p\ xs, dropWhile\ p\ xs)$, while $break\ p$ uses the negation of p. The function $filter$ removes all elements of a list not satisfying a predicate.

$$\begin{array}{l}
span, break :: (a \to Bool) \to [a] \to ([a],[a]) \\
span\ p\ [\,] \quad = ([\,],[\,]) \\
span\ p\ xs@(x : xs') \\
\quad \mid p\ x = \mathbf{let}\ (xr, xt) = span\ p\ xs'\ \mathbf{in}\ (x : xr, xt) \\
\quad \mid otherwise = ([\,], xs)
\end{array}$$

$$break\ p = span\ (not \circ p)$$

$$\begin{array}{l}
filter \qquad :: (a \to Bool) \to [a] \to [a] \\
filter\ p\ [\,] \quad = [\,] \\
filter\ p\ (x : xs) \\
\quad \mid p\ x \qquad = x : filter\ p\ xs \\
\quad \mid otherwise = filter\ p\ xs
\end{array}$$

Fold-like Functions

The functions $foldl1$ and $foldr1$ are variants of $foldl$ and $foldr$ that have no starting value argument, and thus must be applied to nonempty lists.

$$\begin{array}{l}
foldl \qquad :: (a \to b \to a) \to a \to [b] \to a \\
foldl\ f\ z\ [\,] = z \\
foldl\ f\ z\ (x : xs) = foldl\ f\ (f\ z\ x)\ xs
\end{array}$$

$$\begin{array}{l}
foldl1 \qquad :: (a \to a \to a) \to [a] \to a \\
foldl1\ f\ (x : xs) = foldl\ f\ x\ xs \\
foldl1\ _\ [\,] \qquad = error\ \texttt{"foldl1: empty list"}
\end{array}$$

$$\begin{array}{l}
foldr \qquad :: (a \to b \to b) \to b \to [a] \to b \\
foldr\ f\ z\ [\,] = z \\
foldr\ f\ z\ (x : xs) = f\ x\ (foldr\ f\ z\ xs)
\end{array}$$

$$\begin{array}{l}
foldr1 \qquad :: (a \to a \to a) \to [a] \to a \\
foldr1\ f\ [x] \qquad = x \\
foldr1\ f\ (x : xs) = f\ x\ (foldr1\ f\ xs) \\
foldr1\ _\ [\,] \qquad = error\ \texttt{"foldr1: empty list"}
\end{array}$$

The function $scanl$ is similar to $foldl$, but returns a list of successive reduced values from the left. The function $scanr$ is the analogue for $foldr$.

$$\begin{array}{l}
scanl \qquad :: (a \to b \to a) \to a \to [b] \to [a] \\
scanl\ f\ q\ xs = q : (\mathbf{case}\ xs\ \mathbf{of} \\
\qquad\qquad\quad [\,] \to [\,] \\
\qquad\qquad\quad (x : xs') \to scanl\ f\ (f\ q\ x)\ xs')
\end{array}$$

$$\begin{array}{l}
scanl1 \qquad :: (a \to a \to a) \to [a] \to [a] \\
scanl1\ f\ (x : xs) = scanl\ f\ x\ xs \\
scanl1\ _\ [\,] \qquad = error\ \texttt{"scanl1: empty list"}
\end{array}$$

$$scanr \qquad :: (a \to b \to b) \to b \to [a] \to [b]$$

$$\begin{array}{l}
scanr\ f\ z\ [\,] = [z] \\
scanr\ f\ z\ (x : xs) = f\ x\ q : qs \\
\quad \mathbf{where}\ qs@(q : _) = scanr\ f\ z\ xs
\end{array}$$

$$\begin{array}{l}
scanr1 \qquad :: (a \to a \to a) \to [a] \to [a] \\
scanr1\ f\ [x] = [x] \\
scanr1\ f\ (x : xs) = f\ x\ q : qs \\
\quad \mathbf{where}\ qs@(q : _) = scanr1\ f\ xs \\
scanr1\ _\ [\,] \quad = error\ \texttt{"scanr1: empty list"}
\end{array}$$

List Generators

$$\begin{array}{l}
iterate \qquad :: (a \to a) \to a \to [a] \\
iterate\ f\ x = x : iterate\ f\ (f\ x)
\end{array}$$

$$\begin{array}{l}
repeat \quad :: a \to [a] \\
repeat\ x = xs\ \mathbf{where}\ xs = x : xs
\end{array}$$

$$\begin{array}{l}
replicate \qquad :: Int \to a \to [a] \\
replicate\ n\ x = take\ n\ (repeat\ x)
\end{array}$$

$$\begin{array}{l}
cycle \quad :: [a] \to [a] \\
cycle\ [\,] = error\ \texttt{"cycle: empty list"} \\
cycle\ xs = xs'\ \mathbf{where}\ xs' = xs +\!\!+\ xs'
\end{array}$$

String-Based Functions

$$\begin{array}{l}
lines \qquad :: String \to [String] \\
lines\ \texttt{""} = [\,] \\
lines\ s \quad = \mathbf{let}\ (l, s') = break\ (\equiv \texttt{'\char`\\n'})\ s \\
\qquad\qquad \mathbf{in}\ l : \mathbf{case}\ s'\ \mathbf{of} \\
\qquad\qquad\qquad [\,] \to [\,] \\
\qquad\qquad\qquad (_ : s'') \to lines\ s''
\end{array}$$

$$\begin{array}{l}
words \quad :: String \to [String] \\
words\ s = \mathbf{case}\ dropWhile\ Char.isSpace\ s\ \mathbf{of} \\
\qquad\qquad \texttt{""} \to [\,] \\
\qquad\qquad s' \to w : words\ s'' \\
\qquad\qquad\quad \mathbf{where}\ (w, s'') = break\ Char.isSpace\ s'
\end{array}$$

$$
\begin{aligned}
&\mathit{unlines} :: [\mathit{String}] \to \mathit{String}\\
&\mathit{unlines} = \mathit{concatMap}\ (+\!\!+\texttt{"\textbackslash n"})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{unwords} && :: [\mathit{String}] \to \mathit{String}\\
&\mathit{unwords}\ [\,] && = \texttt{""}\\
&\mathit{unwords}\ \mathit{ws} && = \mathit{foldr1}\ (\lambda w\ s \to w +\!\!+ \texttt{' '} : s)\ \mathit{ws}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{reverse} :: [a] \to [a]\\
&\mathit{reverse} = \mathit{foldl}\ (\mathit{flip}\ (:))\ [\,]
\end{aligned}
$$

Boolean List Functions

$$
\begin{aligned}
&\mathit{and}, \mathit{or} :: [\mathit{Bool}] \to \mathit{Bool}\\
&\mathit{and} && = \mathit{foldr}\ (\wedge)\ \mathit{True}\\
&\mathit{or} && = \mathit{foldr}\ (\vee)\ \mathit{False}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{any}, \mathit{all} :: (a \to \mathit{Bool}) \to [a] \to \mathit{Bool}\\
&\mathit{any}\ p && = \mathit{or} \circ \mathit{map}\ p\\
&\mathit{all}\ p && = \mathit{and} \circ \mathit{map}\ p
\end{aligned}
$$

List Membership Functions

$$
\begin{aligned}
&\mathit{elem}, \mathit{notElem} :: (\mathit{Eq}\ a) \Rightarrow a \to [a] \to \mathit{Bool}\\
&\mathit{elem}\ x && = \mathit{any}\ (\equiv x)\\
&\mathit{notElem}\ x && = \mathit{all}\ (\not\equiv x)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{lookup} && :: (\mathit{Eq}\ a) \Rightarrow a \to [(a, b)] \to \mathit{Maybe}\ b\\
&\mathit{lookup}\ \mathit{key}\ [\,] && = \mathit{Nothing}\\
&\mathit{lookup}\ \mathit{key}\ ((x, y) : \mathit{xys}) &&\\
&\quad \mid \mathit{key} \equiv x && = \mathit{Just}\ y\\
&\quad \mid \mathit{otherwise} && = \mathit{lookup}\ \mathit{key}\ \mathit{xys}
\end{aligned}
$$

Arithmetic on Lists

$$
\begin{aligned}
&\mathit{sum}, \mathit{product} :: (\mathit{Num}\ a) \Rightarrow [a] \to a\\
&\mathit{sum} && = \mathit{foldl}\ (+)\ 0\\
&\mathit{product} && = \mathit{foldl}\ (*)\ 1
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{maximum}, \mathit{minimum} :: (\mathit{Ord}\ a) \Rightarrow [a] \to a\\
&\mathit{maximum}\ [\,] && = \mathit{error}\ \texttt{"maximum: empty list"}\\
&\mathit{maximum}\ \mathit{xs} && = \mathit{foldl1}\ \mathit{max}\ \mathit{xs}\\
&\mathit{minimum}\ [\,] && = \mathit{error}\ \texttt{"minimum: empty list"}\\
&\mathit{minimum}\ \mathit{xs} && = \mathit{foldl1}\ \mathit{min}\ \mathit{xs}
\end{aligned}
$$

List Combining Functions

$$
\begin{aligned}
&\mathit{map} :: (a \to b) \to [a] \to [b]\\
&\mathit{map}\ f\ [\,] = [\,]\\
&\mathit{map}\ f\ (x : \mathit{xs}) = f\ x : \mathit{map}\ f\ \mathit{xs}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zipWith} && :: (a \to b \to c) \to [a] \to [b] \to [c]\\
&\mathit{zipWith}\ z\ (a : \mathit{as})\ (b : \mathit{bs}) &&\\
& && = z\ a\ b : \mathit{zipWith}\ z\ \mathit{as}\ \mathit{bs}\\
&\mathit{zipWith}\ \_\ \_\ \_ = [\,] &&
\end{aligned}
$$

$$
\begin{aligned}
&(+\!\!+) :: [a] \to [a] \to [a]\\
&[\,] +\!\!+ \mathit{ys} = \mathit{ys}\\
&(x : \mathit{xs}) +\!\!+ \mathit{ys} = x : (\mathit{xs} +\!\!+ \mathit{ys})
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zipWith3} :: (a \to b \to c \to d) \to [a] \to [b] \to [c] \to [d]\\
&\mathit{zipWith3}\ z\ (a : \mathit{as})\ (b : \mathit{bs})\ (c : \mathit{cs})\\
&\qquad = \qquad z\ a\ b\ c : \mathit{zipWith3}\ z\ \mathit{as}\ \mathit{bs}\ \mathit{cs}\\
&\mathit{zipWith3}\ \_\ \_\ \_\ \_ = [\,]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{concat} :: [[a]] \to [a]\\
&\mathit{concat}\ \mathit{xss} = \mathit{foldr}\ (+\!\!+)\ [\,]\ \mathit{xss}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{concatMap} && :: (a \to [b]) \to [a] \to [b]\\
&\mathit{concatMap}\ f && = \mathit{concat} \circ \mathit{map}\ f
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{unzip} && :: [(a, b)] \to ([a], [b])\\
&\mathit{unzip} && = \mathit{foldr}\ (\lambda(a, b){\sim}(\mathit{as}, \mathit{bs}) \to (a : \mathit{as}, b : \mathit{bs}))\ ([\,], [\,])
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zip} :: [a] \to [b] \to [(a, b)]\\
&\mathit{zip} = \mathit{zipWith}\ (,)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{unzip3} :: [(a, b, c)] \to ([a], [b], [c])\\
&\mathit{unzip3} =\\
&\quad \mathit{foldr}\ (\lambda(a, b, c){\sim}(\mathit{as}, \mathit{bs}, \mathit{cs}) \to (a : \mathit{as}, b : \mathit{bs}, c : \mathit{cs}))\\
&\qquad ([\,], [\,], [\,])
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{zip3} :: [a] \to [b] \to [c] \to [(a, b, c)]\\
&\mathit{zip3} = \mathit{zipWith3}\ (,,)
\end{aligned}
$$