Reinventing Haskell Backtracking

Sebastian Fischer

Christian-Albrechts University of Kiel

KPS 2009

nondeterministic search

$$
\begin{aligned}
&\mathit{anyof} :: [a] \rightarrow \mathit{Search}\ a \\
&\mathit{anyof}\ [\,] \qquad\quad = \emptyset \\
&\mathit{anyof}\ (x : xs) = \mathit{anyof}\ xs \oplus \mathit{return}\ x
\end{aligned}
$$

interface

Failure

$$\emptyset :: \mathit{Search}\ a$$

Success

$$\mathit{return} :: a \rightarrow \mathit{Search}\ a$$

Choice

$$(\oplus) :: \mathit{Search}\ a \rightarrow \mathit{Search}\ a \rightarrow \mathit{Search}\ a$$

lazy lists backtrack

$$\emptyset :: [a]$$
$$\emptyset = [\,]$$

$$\mathit{return} :: a \rightarrow [a]$$
$$\mathit{return}\ x = [x]$$

$$(\oplus) :: [a] \rightarrow [a] \rightarrow [a]$$
$$[\,] \qquad \oplus\ ys = ys$$
$$(x : xs) \oplus ys = x : (xs \oplus ys) \qquad \text{-- lazy}$$

example

$> \mathit{anyof}\ [1\mathinner{..}10] :: [\mathit{Int}]$

example

$> \mathit{anyof}\ [1\mathinner{.\,.}10] :: [\mathit{Int}]$
$[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

example

$> \mathit{anyof}\ [1\mathinner{\ldotp\ldotp}10] :: [\mathit{Int}]$
$[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

$> \mathit{anyof}\ [1\mathinner{\ldotp\ldotp}1000000] :: [\mathit{Int}]$
takes very long

quadratic run time

$$\begin{aligned}
&\mathit{anyof} :: [a] \longrightarrow \mathit{Search}\ a \\
&\mathit{anyof}\ [\,] \qquad\quad = \emptyset \\
&\mathit{anyof}\ (x : xs) = \mathit{anyof}\ xs \oplus \mathit{return}\ x
\end{aligned}$$

quadratic run time

$$
\begin{aligned}
&\mathit{anyof} :: [a] \rightarrow \mathit{Search}\ a \\
&\mathit{anyof}\ [\,] \qquad\quad = \varnothing \\
&\mathit{anyof}\ (x : xs) = \mathit{anyof}\ xs \oplus \mathit{return}\ x
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{reverse} :: [a] \rightarrow [a] \\
&\mathit{reverse}\ [\,] \qquad\quad = [\,] \\
&\mathit{reverse}\ (x : xs) = \mathit{reverse}\ xs \mathbin{+\!\!+} [x]
\end{aligned}
$$

accumulator

$$reverse :: [a] \to [a]$$
$$reverse\ xs = rev\ xs\ [\,]$$

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

accumulator

$$
\begin{aligned}
&\mathit{reverse} :: [a] \to [a] \\
&\mathit{reverse}\ \mathit{xs} = \mathit{rev}\ \mathit{xs}\ [\,]
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathit{rev} :: [a] \to [a] \to [a] \\
&\mathit{rev}\ [\,] \qquad\quad = \lambda \mathit{ys} \to \mathit{ys} \\
&\mathit{rev}\ (x : \mathit{xs}) = \lambda \mathit{ys} \to \mathit{rev}\ \mathit{xs}\ ((\lambda \mathit{zs} \to x : \mathit{zs})\ \mathit{ys})
\end{aligned}
$$

difference lists

$$\textbf{type}\ \mathit{DiffList}\ a = [a] \rightarrow [a]$$

$$\mathit{toList} :: \mathit{DiffList}\ a \rightarrow [a]$$
$$\mathit{toList}\ a = a\ [\,]$$

interface

$$empty :: DiffList\ a$$
$$empty = \lambda xs \rightarrow xs$$

$$singleton :: a \rightarrow DiffList\ a$$
$$singleton\ x = \lambda xs \rightarrow x : xs$$

$$append :: DiffList\ a \rightarrow DiffList\ a \rightarrow DiffList\ a$$
$$append\ a\ b = \lambda xs \rightarrow a\ (b\ xs)$$

an old friend

$$\varnothing :: \mathit{Search}\ a$$
$$\varnothing = \mathit{empty}$$

$$\mathit{return} :: a \rightarrow \mathit{Search}\ a$$
$$\mathit{return} = \mathit{singleton}$$

$$(\oplus) :: \mathit{Search}\ a \rightarrow \mathit{Search}\ a \rightarrow \mathit{Search}\ a$$
$$(\oplus) = \mathit{append}$$

an old friend

$$\varnothing :: Search\ a$$
$$\varnothing = empty$$

$$return :: a \to Search\ a$$
$$return = singleton$$

$$(\oplus) :: Search\ a \to Search\ a \to Search\ a$$
$$(\oplus) = append$$

Nondeterministic application

$$flatMap :: (a \to Search\ b) \to Search\ a \to Search\ b$$
$$flatMap = ???$$

continuation-based search

$$\textbf{type}\ \mathit{CSearch}\ a = \forall b.(a \rightarrow \mathit{Search}\ b) \rightarrow \mathit{Search}\ b$$

$$\begin{aligned}
&\mathit{search} :: \mathit{CSearch}\ a \rightarrow \mathit{Search}\ a\\
&\mathit{search}\ a = a\ (\mathit{return} :: a \rightarrow \mathit{Search}\ a)
\end{aligned}$$

the missing piece

$$\varnothing :: CSearch\ a$$
$$\varnothing = \lambda\_ \rightarrow (\varnothing :: Search\ a)$$

$$return :: a \rightarrow CSearch\ a$$
$$return\ x = \lambda c \rightarrow c\ x$$

$$(\oplus) :: CSearch\ a \rightarrow CSearch\ a \rightarrow CSearch\ a$$
$$a \oplus b = \lambda c \rightarrow (a\ c \oplus b\ c :: Search\ a)$$

$$flatMap :: (a \rightarrow CSearch\ b) \rightarrow CSearch\ a \rightarrow CSearch\ b$$
$$flatMap\ f\ a = \lambda c \rightarrow a\ (\lambda x \rightarrow f\ x\ c)$$

example

$> \mathit{toList}\ (\mathit{search}\ (\mathit{anyof}\ [1\mathinner{\ldotp\ldotp}10]))$
$[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

example

$> toList\ (search\ (anyof\ [1\,..\,10]))$
$[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

$> toList\ (search\ (anyof\ [1\,..\,1000000]))$
$[1000000, 999999, 999998, 999997, \ldots$

difference lists + continuations

$$\emptyset = \lambda succ\ fail \rightarrow fail$$

$$return\ x = \lambda succ\ fail \rightarrow succ\ x\ fail$$

$$a \oplus b = \lambda succ\ fail \rightarrow a\ succ\ (b\ succ\ fail)$$

$$flatMap\ f\ a = \lambda succ\ fail \rightarrow a\ (\lambda x\ fail' \rightarrow f\ x\ succ\ fail')\ fail$$

nondeterminism monad

$$
\begin{aligned}
&\mathit{pytriple}_{\leqslant} :: \mathit{Int} \rightarrow \mathit{CSearch}\ (\mathit{Int}, \mathit{Int}, \mathit{Int}) \\
&\mathit{pytriple}_{\leqslant}\ n = \mathbf{do}\ a \leftarrow \mathit{anyof}\ [1 \mathinner{.\,.} n] \\
&\qquad\qquad\qquad\quad\ b \leftarrow \mathit{anyof}\ [a \mathinner{.\,.} n] \\
&\qquad\qquad\qquad\quad\ c \leftarrow \mathit{anyof}\ [b \mathinner{.\,.} n] \\
&\qquad\qquad\qquad\quad\ \mathit{guard}\ (a * a + b * b \equiv c * c) \\
&\qquad\qquad\qquad\quad\ \mathit{return}\ (a, b, c)
\end{aligned}
$$

$$
\begin{aligned}
&> \mathit{toList}\ (\mathit{search}\ (\mathit{pytriple}_{\leqslant}\ 10)) \\
&[(6, 8, 10), (3, 4, 5)]
\end{aligned}
$$

summary

efficient backtracking can be factored into two parts

- difference lists
- continuation passing

continuations provide *flatMap* **for free**

continuation passing can be reused for other strategies

`http://hackage.haskell.org/package/level-monad`

question menu

1. What other search strategies can be implemented like this?
2. How efficient are they?
3. Why do I need different strategies at all?
4. How can I decide *which* strategy to use *when*?
5. Why are nondeterminism monads useful?
6. Does *CSearch* satisfy the monad laws?
7. What are monad laws, anyway?

no upper bound

$$
\begin{aligned}
&\mathit{pytriple} :: \mathit{CSearch}\ (\mathit{Int}, \mathit{Int}, \mathit{Int}) \\
&\mathit{pytriple} = \mathbf{do}\ a \leftarrow \mathit{anyof}\ [1\,..] \\
&\qquad\qquad\quad\ b \leftarrow \mathit{anyof}\ [a\,..] \\
&\qquad\qquad\quad\ c \leftarrow \mathit{anyof}\ [b\,..] \\
&\qquad\qquad\quad\ \mathit{guard}\ (a * a + b * b \equiv c * c) \\
&\qquad\qquad\quad\ \mathit{return}\ (a, b, c)
\end{aligned}
$$

$> \mathit{take}\ 5\ (\mathit{toList}\ (\mathit{search}\ \mathit{pytriple}))$

diverges

level-wise search

$$\textbf{type}\ \mathit{Levels}\ a = [[a]]$$

$$\begin{aligned}
&\varnothing :: \mathit{Levels}\ a\\
&\varnothing = [\,]
\end{aligned}$$

$$\begin{aligned}
&\mathit{return} :: a \rightarrow \mathit{Levels}\ a\\
&\mathit{return}\ x = [[x]]
\end{aligned}$$

$$\begin{aligned}
&(\oplus) :: \mathit{Levels}\ a \rightarrow \mathit{Levels}\ a \rightarrow \mathit{Levels}\ a\\
&a \oplus b = [\,] : \mathit{merge}\ a\ b
\end{aligned}$$

$$\begin{array}{llll}
\mathit{merge}\ [\,] & \mathit{ys} & = & \mathit{ys}\\
\mathit{merge}\ \mathit{xs} & [\,] & = & \mathit{xs}\\
\mathit{merge}\ (x : \mathit{xs}) & (y : \mathit{ys}) & = & (x \mathbin{+\!\!+} y) : \mathit{merge}\ \mathit{xs}\ \mathit{ys}
\end{array}$$

limited-depth search

$$\textbf{type}\ \mathit{Limited}\ a = \mathit{Int} \rightarrow [a]$$

$$\varnothing :: \mathit{Limited}\ a$$
$$\varnothing = \lambda\_ \rightarrow [\,]$$

$$\mathit{return} :: a \rightarrow \mathit{Limited}\ a$$
$$\mathit{return}\ x = \lambda d \rightarrow \textbf{if}\ d \equiv 0\ \textbf{then}\ [x]\ \textbf{else}\ [\,]$$

$$(\oplus) :: \mathit{Limited}\ a \rightarrow \mathit{Limited}\ a \rightarrow \mathit{Limited}\ a$$
$$a \oplus b = \lambda d \rightarrow \textbf{if}\ d \equiv 0\ \textbf{then}\ [\,]\ \textbf{else}\ a\ (d-1) +\!\!+\ b\ (d-1)$$

fair search

$>$ *take* 5 (*concat* (*search pytriple*))
$[(3,4,5),(6,8,10),(5,12,13),(9,12,15),(8,15,17)]$

500 triples $\approx$ 20 seconds, 1 GB

$>$ *take* 5 (*iterDepth pytriple*) -- iteratively increasing limit
$[(3,4,5),(6,8,10),(5,12,13),(9,12,15),(8,15,17)]$

500 triples $\approx$ 40 seconds, 2 MB