*J. Functional Programming* 3 (1): 117–122, January 1993  

# FUNCTIONAL PEARLS

## *The last tail*

R. S. BIRD

*Programming Research Group, Oxford University, 11 Keble Rd, Oxford OX1 3QD*

### 1 Introduction

Suppose the tail segments of a given list are sorted into dictionary order. What segment comes last? For example, the last tail of 'testing' is 'ting' and the last tail of 'redared' is 'redared' itself since 'red' precedes 'redared' in dictionary order.

The function *lt* for returning the last tail is specified by

$$lt = \sqcup/ \cdot tails, \tag{1}$$

where *tails* returns the set of tail segments of a list and $\sqcup/$ distributes $\sqcup$ over a set. The binary operator $\sqcup$ returns the larger of its arguments under the lexical ordering $\sqsubseteq$ (defined later). Given a suitable implementation of $\sqsubseteq$, equation (1) can be used to compute *lt* but the result is a quadratic time algorithm even with lazy evaluation. Our purpose is to derive a linear time functional algorithm. Given such a simply stated problem, this turns out to be surprisingly difficult.

### 2 First steps

Our strategy is to head for an inductive characterisation of *lt*. Since $tails\ [\,] = \{[\,]\}$, we get $lt\ [\,] = [\,]$. For the general case we can try to express either $lt\ ([a] \mathbin{+\!\!+} x)$ or $lt\ (x \mathbin{+\!\!+} [a])$ in terms of $a$ and $lt\ x$. But since, for example, the largest tail of 'zebra' (namely, 'zebra' itself) cannot be expressed in terms of 'z' and the largest tail of 'ebra' (namely, 'ra'), it is apparent that the first method cannot work. So we will look for an $\oplus$ such that

$$lt\ (x \mathbin{+\!\!+} [a]) \quad = \quad lt\ x \oplus a. \tag{2}$$

To this end, let $lt\ x = y$ and $lt\ (x \mathbin{+\!\!+} [a]) = z \mathbin{+\!\!+} [a]$. For (2) to be satisfied we need $z \in tails\ y$. We reason

$$\begin{array}{ll} & lt\ x = y \ \wedge\ lt\ (x \mathbin{+\!\!+} [a]) = z \mathbin{+\!\!+} [a] \\ \Rightarrow & \quad \{(1), \text{ since } y \text{ and } z \text{ are both tails of } x\} \\ & z \sqsubseteq y \ \wedge\ y \mathbin{+\!\!+} [a] \sqsubseteq z \mathbin{+\!\!+} [a]. \end{array}$$

To proceed we need the definition of the lexical ordering $\sqsubseteq$:

$$\begin{array}{lcl} z \sqsubseteq y & = & z \in inits\ y \ \vee\ z \prec y \\ z \prec y & = & (\exists k : 0 \leqslant k < \min\{\#z, \#y\} : z \uparrow k = y \uparrow k \wedge z!k < y!k). \end{array}$$

https://doi.org/10.1017/S0956796800000630 Published online by Cambridge University Press

Here, $x \uparrow k$ is the initial segment of $x$ of length $k$ and $x!k$ is the element of $x$ at position $k$ (counting from 0). The function *inits* returns the set of initial segments of a list. The following two properties of $\sqsubseteq$ are easily proved from the definition and we omit details:

$$z \in inits\, y \ \wedge\ z \neq y \quad \Rightarrow \quad (a \leqslant hd\,(y \rightharpoondown z) \quad \equiv \quad z \mathbin{+\!\!+} [a] \sqsubseteq y \mathbin{+\!\!+} [a]) \tag{3}$$

$$z \notin inits\, y \ \wedge\ z \sqsubset y \quad \Rightarrow \quad z \mathbin{+\!\!+} [a] \sqsubset y \mathbin{+\!\!+} [a]. \tag{4}$$

Here, $y \rightharpoondown z$ (pronounced $y$ 'drop' $z$) is what remains when initial segment $z$ of $y$ is removed from $y$, and $hd\, x$ returns the first element of the nonempty sequence $x$. In (4) the strict lexical order $\sqsubset$ is defined by $z \sqsubset y = z \sqsubseteq y \ \wedge\ z \neq y$. The conclusion of (4) can be strengthened to read: $z \mathbin{+\!\!+} u \sqsubset y \mathbin{+\!\!+} v$ for all $u$ and $v$.

Now we can continue:

$$\begin{array}{ll}
 & z \sqsubseteq y \ \wedge\ y \mathbin{+\!\!+} [a] \sqsubseteq z \mathbin{+\!\!+} [a] \\
\Rightarrow & \quad \{\text{contrapositive of (4)}\} \\
 & z \sqsubseteq y \ \wedge\ (y \sqsubseteq z \ \vee\ z \in inits\, y) \\
\Rightarrow & \quad \{\text{since } z \sqsubseteq y \wedge y \sqsubset z \Rightarrow z = y\} \\
 & z \in inits\, y \\
\Rightarrow & \quad \{\text{since one of } y \text{ and } z \text{ is a tail of the other}\} \\
 & z \in tails\, y.
\end{array}$$

Hence we get the desired result. In fact we have that $z \in \{y\} \cup rims\, y$, where we define

$$rims\, y \quad = \quad (inits\, y \cap tails\, y) - \{y\}.$$

To determine $\oplus$, suppose $lt\, x = y \neq [\,]$ and let $rims\, y = [z_0, z_1, \ldots, z_n]$ be arranged in order of decreasing length. We have $rims\, z_i = [z_{i+1}, \ldots, z_n]$ for $0 \leqslant i < n$. In particular, if we define $rim\, w$ to be the longest element of $rims\, w$, then $z_{i+1} = rim\, z_i$ for $0 \leqslant i < n$.

We now claim that if $y = lt\, x$, then

$$0 \leqslant i < j \leqslant n \quad \Rightarrow \quad hd\,(y \rightharpoondown z_i) \leqslant hd\,(y \rightharpoondown z_j). \tag{5}$$

To prove (5), observe that both $z_i$ and $z_j$ are initial segments of $y$, so $y = z_i \mathbin{+\!\!+} [a] \mathbin{+\!\!+} u$ and $y = z_j \mathbin{+\!\!+} [b] \mathbin{+\!\!+} v$ for some $u$ and $v$, where $a = hd\,(y \rightharpoondown z_i)$ and $b = hd\,(y \rightharpoondown z_j)$. Since $z_i$ and $z_j$ are also both tails of $y$, and $z_j$ is shorter than $z_i$, we have that $z_j$ is a tail of $z_i$. Hence $z_j \mathbin{+\!\!+} [a] \mathbin{+\!\!+} u$ is a tail of $y$. But since $y$ is a largest tail, $z_j \mathbin{+\!\!+} [a] \mathbin{+\!\!+} u \sqsubset y = z_j \mathbin{+\!\!+} [b] \mathbin{+\!\!+} v$, and so $a \leqslant b$ by definition of $\sqsubseteq$.

We can exploit (5) in conjunction with (3) to get, in the case $y \neq [\,]$,

$$a \leqslant hd\,(y \rightharpoondown rim\, y) \ \wedge\ z \in rims\, y \quad \Rightarrow \quad z \mathbin{+\!\!+} [a] \sqsubseteq y \mathbin{+\!\!+} [a],$$

Hence we have

$$lt\,(x \mathbin{+\!\!+} [a]) \quad = \quad \begin{cases} [a], & \text{if } y = [\,] \\ y \mathbin{+\!\!+} [a], & \text{if } a \leqslant hd\,(y \rightharpoondown rim\, y) \\ lt\,(rim\, y \mathbin{+\!\!+} [a]), & \text{if } a > hd\,(y \rightharpoondown rim\, y) \end{cases}$$

Thus we can define an $\oplus$ satisfying (2) by taking $[\,] \oplus a = [a]$ and for $y \neq [\,]$

$$y \oplus a \quad = \quad \begin{cases} y +\!\!+ [a], & \text{if } a \leqslant hd\ (y \rightharpoondown z) \\ lt\ (z +\!\!+ [a]), & \text{otherwise} \\ \text{where } z = rim\ y \end{cases} \tag{6}$$

There is an important optimisation we can make in the case $a > hd\ (y \rightharpoondown z)$ of (6). Suppose $y = z +\!\!+ w$ (so $w$ is a tail of $y$ and $y \rightharpoondown z = w$). We claim that

$$\#w \leqslant \#z \quad \Rightarrow \quad lt\ (z +\!\!+ [a]) = lt\ (tail\ w +\!\!+ [a]),$$

where $tail\ ([b] +\!\!+ w) = w$. For the proof, note that $\#w \leqslant \#z$ implies $w \in tails\ z$ since both $z$ and $w$ are tails of $y$. By assumption $hd\ w < a$, so $u +\!\!+ w +\!\!+ [a] \sqsubset u +\!\!+ [a]$ for any $u$ and, in particular, for any $u$ such that $u +\!\!+ w \in tails\ z$. Hence $lt\ (z +\!\!+ [a])$ is no longer than $tail\ w +\!\!+ [a]$.

It follows that we can replace (6) by

$$y \oplus a \quad = \quad \begin{cases} y +\!\!+ [a], & \text{if } a \leqslant hd\ (y \rightharpoondown z) \\ lt\ (u +\!\!+ [a]), & \text{otherwise} \\ \text{where } z = rim\ y \text{ and } u = z \sqcap_{\#} tail\ (y \rightharpoondown z) \end{cases} \tag{7}$$

where $\sqcap_{\#}$ returns the shorter of its two arguments.

## 3 An iterative algorithm

In order to time the program for $lt$ it is convenient to turn it into an iterative algorithm. Define $lti$ for $x \neq [\,]$ by

$$lt\ (x +\!\!+ t) \quad = \quad lti\ (lt\ x, t).$$

In particular, $lt\ ([a] +\!\!+ t) = lti\ ([a], t)$. With $lt\ x = y$ we have

$$\begin{array}{ll} & lti\ (y, [a] +\!\!+ t) \\ = & \quad \{\text{specification of } lti\} \\ & lt\ (x +\!\!+ [a] +\!\!+ t) \\ = & \quad \{\text{specification of } lti\} \\ & lti\ (lt\ (x +\!\!+ [a]), t) \\ = & \quad \{\text{given } y = lt\ x \text{ and specification of } \oplus\} \\ & lti\ (y \oplus a, t). \end{array}$$

We now install the program (7) for $\oplus$. The interesting case is when $a > hd\ (y \rightharpoondown z)$, where $z = rim\ y$, when we get

$$\begin{array}{ll} & lti\ (y \oplus a, t) \\ = & \quad \{\text{case assumption}\} \\ & lti\ (lt\ (u +\!\!+ [a]), t) \\ = & \quad \{\text{specification of } lti\} \\ & lt\ (u +\!\!+ [a] +\!\!+ t). \end{array}$$

https://doi.org/10.1017/S0956796800000630 Published online by Cambridge University Press

Hence we obtain the following program for *lt*:

$$
\begin{array}{rcl}
lt\ [\,] &=& [\,] \\
lt\ ([a] \mathbin{+\!\!\!+} t) &=& lti\ ([a], t) \\
lti\ (y, [\,]) &=& y \\
lti\ (y, [a] \mathbin{+\!\!\!+} t) &=& \begin{cases} lti\ (y \mathbin{+\!\!\!+} [a], t), & \text{if } a \leqslant hd\ (y \rightharpoondown z) \\ lt\ (u \mathbin{+\!\!\!+} [a] \mathbin{+\!\!\!+} t), & \text{otherwise} \\ \text{where } z = rim\ y \text{ and } u = z \sqcap_{\#} tail\ (y \rightharpoondown z) \end{cases}
\end{array}
$$

Ignoring the cost of computing $\mathbin{+\!\!\!+}$, $\rightharpoondown$, *rim* and $\sqcap_{\#}$ we can now show that this program for computing $lt\ ([a] \mathbin{+\!\!\!+} t)$ is linear in $\#t$. We claim that there are at most $2\#t + 1$ calls to *lti* during the computation. To prove the claim we show that the value $\#y + 2\#t$ decreases by at least one at each call of *lti*. The only non–trivial case is the last one. Suppose firstly that $u = z$ so $\#z \leqslant (\#y - \#z - 1)$. Then since $lt\ (u \mathbin{+\!\!\!+} [a] \mathbin{+\!\!\!+} t)$ rewrites to $lti\ ([b], v)$, where $u \mathbin{+\!\!\!+} [a] \mathbin{+\!\!\!+} t = [b] \mathbin{+\!\!\!+} v$, we have

$$1 + 2(\#(u \mathbin{+\!\!\!+} [a] \mathbin{+\!\!\!+} t) - 1) = 1 + 2\#z + 2\#t < \#y + 2\#([a] \mathbin{+\!\!\!+} t).$$

On the other hand, if $u = tail\ (y \rightharpoondown z)$, so $\#y - \#z - 1 \leqslant \#z$, we have

$$1 + 2(\#(u \mathbin{+\!\!\!+} [a] \mathbin{+\!\!\!+} t) - 1) = 1 + 2(\#y - \#z) + 2\#t < \#y + 2\#([a] \mathbin{+\!\!\!+} t).$$

## 4 Computing *rim*

The function *rims* can also be computed inductively. We have $rims\ [a] = \{[\,]\}$ and for $y \neq [\,]$

$$rims\ (y \mathbin{+\!\!\!+} [a]) \;=\; \{z \mathbin{+\!\!\!+} [a] \mid z \in rims\ y \wedge hd\ (y \rightharpoondown z) = a\} \cup \{[\,]\}.$$

The proof of this claim is straightforward and we omit details. Hence we obtain $rim\ [a] = [\,]$ and for $y \neq [\,]$

$$
rim\ (y \mathbin{+\!\!\!+} [a]) \;=\; \begin{cases} z \mathbin{+\!\!\!+} [a], & \text{if } hd\ (y \rightharpoondown z) = a \\ rim\ (z \mathbin{+\!\!\!+} [a]), & \text{otherwise} \\ \text{where } z = rim\ y \end{cases}
$$

Since $rim\ y$ is needed only for $y$ satisfying $y = lt\ y$, we can appeal to (5) to optimise the computation, obtaining

$$
rim\ (y \mathbin{+\!\!\!+} [a]) \;=\; \begin{cases} [\,], & \text{if } a < hd\ (y \rightharpoondown z) \\ z \mathbin{+\!\!\!+} [a], & \text{if } a = hd\ (y \rightharpoondown z) \\ rim\ (z \mathbin{+\!\!\!+} [a]), & \text{if } a > hd\ (y \rightharpoondown z) \\ \text{where } z = rim\ y. \end{cases}
$$

## 5 Combining computations

The computations of *lt* and *rim* have turned out to be very similar, and the next step is to combine them into one. We also take the opportunity to eliminate the expensive operation $hd\ (y \rightharpoondown z)$. For $x \neq [\,]$ define

$$le\ x \;=\; (lt\ x, rim\ (lt\ x), lt\ x \rightharpoondown rim\ (lt\ x)).$$

https://doi.org/10.1017/S0956796800000630 Published online by Cambridge University Press

In particular, $le\,[a] = ([a], [\,], [a])$. We omit details of the routine derivation that establishes for $x \neq [\,]$

$$le\,(x \mathbin{+\!\!+} [a]) \quad = \quad le\,x \otimes a, \tag{8}$$

where

$$(y, z, [b] \mathbin{+\!\!+} w) \otimes a \quad = \quad \begin{cases} (y \mathbin{+\!\!+} [a], [\,], y \mathbin{+\!\!+} [a]), & \text{if } a < b \\ (y \mathbin{+\!\!+} [a], z \mathbin{+\!\!+} [a], w \mathbin{+\!\!+} [a]), & \text{if } a = b \\ le\,((z \sqcap_{\#} w) \mathbin{+\!\!+} [a]), & \text{if } a > b \end{cases}$$

The value $lt\,x$ can be recovered as the first component of $le\,x$. In the same way as we did for $lt$ we can rewrite the program for $le$ as an iterative algorithm. Define $lti$ for $x \neq [\,]$ by

$$lt\,(x \mathbin{+\!\!+} t) \quad = \quad lti\,(le\,x, t)$$

It is routine to obtain the following program for $lt$:

$$\begin{aligned} lt\,[\,] \quad &= \quad [\,] \\ lt\,([a] \mathbin{+\!\!+} t) \quad &= \quad lti\,([a], [\,], [a], t) \\ lti\,(y, z, w, [\,]) \quad &= \quad y \\ lti\,(y, z, [b] \mathbin{+\!\!+} w, [a] \mathbin{+\!\!+} t) \quad &= \quad \begin{cases} lti\,(y \mathbin{+\!\!+} [a], [\,], y \mathbin{+\!\!+} [a], t), & \text{if } a < b \\ lti\,(y \mathbin{+\!\!+} [a], z \mathbin{+\!\!+} [a], w \mathbin{+\!\!+} [a], t), & \text{if } a = b \\ lt\,((z \sqcap_{\#} w) \mathbin{+\!\!+} [a] \mathbin{+\!\!+} t), & \text{if } a > b \end{cases} \end{aligned}$$

This program is linear for the same reason as before.

## 6 Final optimisations

The remaining tasks are to eliminate the operations $\mathbin{+\!\!+}$ and $\sqcap_{\#}$ since we cannot suppose $\mathbin{+\!\!+}$ takes constant time in a standard functional language. We can compute $\sqcap_{\#}$ in constant time by adding the lengths of the various arguments as additional parameters (giving a total of no fewer than eight arguments!). To eliminate $\mathbin{+\!\!+}$ define $ltj$ by the equation

$$lti\,(y, z, w, t) \quad = \quad ltj\,(y \mathbin{+\!\!+} t, z \mathbin{+\!\!+} t, w \mathbin{+\!\!+} t, t).$$

Using the program for $lti$ we obtain

$$\begin{aligned} lt\,[\,] \quad &= \quad [\,] \\ lt\,([a] \mathbin{+\!\!+} t) \quad &= \quad ltj\,([a] \mathbin{+\!\!+} t, t, [a] \mathbin{+\!\!+} t, t) \\ ltj\,(y, z, w, [\,]) \quad &= \quad y \\ ltj\,(y, z, [b] \mathbin{+\!\!+} w, [a] \mathbin{+\!\!+} t) \quad &= \quad \begin{cases} ltj\,(y, t, y, t), & \text{if } a < b \\ ltj\,(y, z, w, t), & \text{if } a = b \\ lt\,(z \sqcap_{\#} w), & \text{if } a > b \end{cases} \end{aligned}$$

**Comments and acknowledgements**

My derivation of the last tail problem has been under revision for a year or more. One earlier version created a cyclic structure to achieve the required efficiency.

Comments and criticisms on earlier versions by Wim Feijen (who has derived a similar algorithm for the problem of finding the lexically least rotation of a sequence), Rob Hoogerwoord and Berry Schoenmakers have proved invaluable. My only regret is that the final program has such an imperative flavour. A good challenge is to find a decent functional program for the problem.

https://doi.org/10.1017/S0956796800000630 Published online by Cambridge University Press