More on Merging and Selection

Jeremy Gibbons
School of Computing and Mathematical Sciences,
Oxford Brookes University
Gipsy Lane, Headington,
Oxford OX3 0BP, UK.
Email: `jgibbons@brookes.ac.uk`

1st September 1997

Abstract

In his paper *On Merging and Selection* (*Journal of Functional Programming* 7(3), 1997), Bird considers the problem of computing the nth element of the list resulting from merging the two sorted lists x and y. Representing x and y by trees, Bird derives an algorithm for the problem taking time proportional to the sum of their depths. Bird's derivation is more complicated than necessary. By the simple tactic of delaying a design decision (in this case, the decision to represent the lists as trees) as long as possible, we obtain a much simpler solution.

1 Introduction

Richard Bird [2] considers the problem of computing

$$\mathit{merge}\,(x, y) \mathbin{!!} n$$

—that is, of computing the nth element of the list resulting from merging the two sorted lists x and y—and the related problem when x and y are represented by trees. For the former problem, $\Theta(n)$ steps are required, even in a lazy functional language; for the latter problem Bird derives a solution taking time proportional to the sum of the depths of the two trees—despite the fact that there is no known way of computing just the *merge* that quickly.

However, Bird's derivation can be simplified in at least two ways. One simplification is to delay the introduction of the tree representation as long as possible, and to perform part of the derivation in terms of lists instead. This heuristic of delaying design decisions as long as possible is standard in program construction; it has the direct benefit here of simplifying the calculations, but leads to an indirect benefit too. A crucial part of Bird's derivation is a 'none too obvious' (Bird's own words) relationship between merging and indexing, pulled from thin air and requiring a 'quite tricky' two-page proof. Using the standard heuristic of delaying design decisions, we derive this relationship by calculation, simultaneously motivating its introduction and greatly simplifying its proof.

2 Preliminaries

We write '$x \sqsubseteq y$' for lists x and y to mean '$a \leq b$ for all elements a of x and b of y'. The notation '$x \mathbin{!!} n$' denotes the nth element of list x (the first element of x being $x \mathbin{!!} 0$), '$\#x$' denotes the length of list x and '$x +\!\!+\, y$' the concatenation of lists x and y.

We use without proof the following two properties of $+\!\!+$, $\sqsubseteq$ and *merge*.

Fact 1 *If lists $w +\!\!+\, y$ and $x +\!\!+\, z$ are sorted, then either $w \sqsubseteq z$ or $x \sqsubseteq y$.*

Fact 2 *If w, x, y, z are sorted, $w \sqsubseteq z$ and $x \sqsubseteq y$, then*

$$\mathit{merge}\,(w +\!\!+\, y, x +\!\!+\, z) = \mathit{merge}\,(w, x) +\!\!+\, \mathit{merge}\,(y, z)$$

Fact 2 is a kind of 'abiding property' [1].

We also use, again without proof, the following property of indexing.

Fact 3

$$\begin{array}{rcll} (x +\!\!+\, y) \mathbin{!!} n & = & x \mathbin{!!} n, & \text{if } n < \#x \\ & = & y \mathbin{!!} (n - \#x), & \text{if } n \geq \#x \end{array}$$

3 Derivation

We now proceed with the calculation. Suppose that $w +\!\!+\, y$ and $x +\!\!+\, z$ are both sorted, and assume without loss of generality that $w \sqsubseteq z$. We have

$$\begin{array}{cl} & \mathit{merge}\,(w +\!\!+\, y, x +\!\!+\, z) \mathbin{!!} n \\ = & \quad \{ \text{ choose } y_1, y_2 \text{ such that } y = y_1 +\!\!+\, y_2,\ w +\!\!+\, y_1 \sqsubseteq z,\ x \sqsubseteq y_2 \ \} \\ & \mathit{merge}\,(w +\!\!+\, y_1 +\!\!+\, y_2, x +\!\!+\, z) \mathbin{!!} n \\ = & \quad \{ \text{ Fact 2 } \} \\ & (\mathit{merge}\,(w +\!\!+\, y_1, x) +\!\!+\, \mathit{merge}\,(y_2, z)) \mathbin{!!} n \end{array}$$

$$
\begin{array}{ll}
= & \quad \{ \text{ assume } n < \#w + \#y_1 + \#x; \text{ Fact 3 } \} \\
& \mathit{merge}\,(w \mathbin{+\!\!\!+} y_1, x) \mathbin{!!} n \\
= & \quad \{ \text{ Fact 3 again } \} \\
& (\mathit{merge}\,(w \mathbin{+\!\!\!+} y_1, x) \mathbin{+\!\!\!+} \mathit{merge}\,(y_2, [\,])) \mathbin{!!} n \\
= & \quad \{ \text{ Fact 2 } \} \\
& \mathit{merge}\,(w \mathbin{+\!\!\!+} y, x) \mathbin{!!} n
\end{array}
$$

Still considering the case that $w \sqsubseteq z$, a symmetric argument gives us

$$
\begin{array}{ll}
& \mathit{merge}\,(w \mathbin{+\!\!\!+} y, x \mathbin{+\!\!\!+} z) \mathbin{!!} n \\
= & \quad \{ \text{ choose } x_1, x_2 \text{ such that } x = x_1 \mathbin{+\!\!\!+} x_2,\ x_1 \sqsubseteq y,\ w \sqsubseteq x_2 \mathbin{+\!\!\!+} z \ \} \\
& \mathit{merge}\,(w \mathbin{+\!\!\!+} y, x_1 \mathbin{+\!\!\!+} x_2 \mathbin{+\!\!\!+} z) \mathbin{!!} n \\
= & \quad \{ \text{ Fact 2 } \} \\
& (\mathit{merge}\,(w, x_1) \mathbin{+\!\!\!+} \mathit{merge}\,(y, x_2 \mathbin{+\!\!\!+} z)) \mathbin{!!} n \\
= & \quad \{ \text{ assume } n \geq \#w + \#x_1; \text{ Fact 3, twice } \} \\
& (\mathit{merge}\,([\,], x_1) \mathbin{+\!\!\!+} \mathit{merge}\,(y, x_2 \mathbin{+\!\!\!+} z)) \mathbin{!!} (n - \#w) \\
= & \quad \{ \text{ Fact 2 } \} \\
& \mathit{merge}\,(y, x \mathbin{+\!\!\!+} z) \mathbin{!!} (n - \#w)
\end{array}
$$

Thus we have, when $w \sqsubseteq z$,

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{merge}\,(w \mathbin{+\!\!\!+} y, x \mathbin{+\!\!\!+} z) \mathbin{!!} n} \\
\qquad = & \mathit{merge}\,(w \mathbin{+\!\!\!+} y, x) \mathbin{!!} n, & \text{if } n < \#w + \#y_1 + \#x \\
\qquad = & \mathit{merge}\,(y, x \mathbin{+\!\!\!+} z) \mathbin{!!} (n - \#w), & \text{if } n \geq \#w + \#x_1
\end{array}
$$

These guards overlap; they can be strengthened to

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{merge}\,(w \mathbin{+\!\!\!+} y, x \mathbin{+\!\!\!+} z) \mathbin{!!} n} \\
\qquad = & \mathit{merge}\,(w \mathbin{+\!\!\!+} y, x) \mathbin{!!} n, & \text{if } n < \#w + \#x \\
\qquad = & \mathit{merge}\,(y, x \mathbin{+\!\!\!+} z) \mathbin{!!} (n - \#w), & \text{if } n \geq \#w + \#x
\end{array}
$$

If it is not the case that $w \sqsubseteq z$, then $x \sqsubseteq y$ by Fact 1, and dually we get

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{merge}\,(w \mathbin{+\!\!\!+} y, x \mathbin{+\!\!\!+} z) \mathbin{!!} n} \\
\qquad = & \mathit{merge}\,(w, x \mathbin{+\!\!\!+} z) \mathbin{!!} n, & \text{if } n < \#w + \#x \\
\qquad = & \mathit{merge}\,(w \mathbin{+\!\!\!+} y, z) \mathbin{!!} (n - \#x), & \text{if } n \geq \#w + \#x
\end{array}
$$

These equations embody Bird's 'none too obvious' property, but now they are perfectly obvious equations and need no tricky proof.

4 Data refinement

It is only at this point that we need to bring trees into the picture. We will need to be able to find quickly the size of the left subtree of a tree, so we use the datatype definition

$$\textbf{data}\ \mathit{Tree}\ a \quad = \quad \mathit{Empty} \mid \mathit{Bin}\,(\mathit{Tree}\ a)\ \mathit{Int}\ a\,(\mathit{Tree}\ a)$$

with the datatype invariants that for every tree t, the list $\mathit{flatten}\ t$ is sorted—where the function $\mathit{flatten}$ is defined by

$$\begin{array}{rcl} \mathit{flatten}\ \mathit{Empty} & = & [\,] \\ \mathit{flatten}\,(\mathit{Bin}\ w\ p\ a\ x) & = & \mathit{flatten}\ w +\!\!+ [a] +\!\!+ \mathit{flatten}\ x \end{array}$$

—and that for every non-empty tree $\mathit{Bin}\ w\ p\ a\ x$, we have $p = \#(\mathit{flatten}\ w)$.

We now introduce the function select, defined by

$$\mathit{select}\,(t, u)\ n = \mathit{merge}\,(\mathit{flatten}\ t, \mathit{flatten}\ u)\ !!\ n$$

The problem is to synthesize a more efficient characterization of select.

We consider the first of the four cases for non-empty trees. We assume in the following that $a \leq b$ (so that $\mathit{flatten}\ w +\!\!+ [a] \sqsubseteq [b] +\!\!+ \mathit{flatten}\ z$), and that $n < \#\mathit{flatten}\ w + 1 + \#\mathit{flatten}\ x$.

$$\begin{array}{cl} & \mathit{select}\,(\mathit{Bin}\ w\ p\ a\ y, \mathit{Bin}\ x\ q\ b\ z)\ n \\ = & \quad \{\ \mathit{select}\ \} \\ & \mathit{merge}\,(\mathit{flatten}\,(\mathit{Bin}\ w\ p\ a\ y), \mathit{flatten}\,(\mathit{Bin}\ x\ q\ b\ z))\ !!\ n \\ = & \quad \{\ \mathit{flatten}\ \} \\ & \mathit{merge}\,(\mathit{flatten}\ w +\!\!+ [a] +\!\!+ \mathit{flatten}\ y, \mathit{flatten}\ x +\!\!+ [b] +\!\!+ \mathit{flatten}\ z)\ !!\ n \\ = & \quad \{\ \text{case hypotheses; property from Section 3}\ \} \\ & \mathit{merge}\,(\mathit{flatten}\ w +\!\!+ [a] +\!\!+ \mathit{flatten}\ y, \mathit{flatten}\ x)\ !!\ n \\ = & \quad \{\ \mathit{flatten}\ \text{and}\ \mathit{select}\ \text{again}\ \} \\ & \mathit{select}\,(\mathit{Bin}\ w\ p\ a\ y, x)\ n \end{array}$$

The other three cases are equally straightforward, and we arrive at the following program:

$$\begin{array}{rcll} \mathit{select}\,(t, u)\ n & = & \mathit{select}\,(t, x)\ n, & \text{if } a \leq b \text{ and } n < p + 1 + q \\ & = & \mathit{select}\,(y, u)\,(n - p - 1), & \text{if } a \leq b \text{ and } n \geq p + 1 + q \\ & = & \mathit{select}\,(w, u)\ n, & \text{if } a \geq b \text{ and } n < p + 1 + q \\ & = & \mathit{select}\,(t, z)\,(n - q - 1), & \text{if } a \geq b \text{ and } n \geq p + 1 + q \\ & & \text{where}\ \begin{array}{rcl} \mathit{Bin}\ w\ p\ a\ y & = & t \\ \mathit{Bin}\ x\ q\ b\ z & = & u \end{array} & \end{array}$$

When u is empty, we have

$$
\begin{array}{cl}
 & \mathit{select}\,(t, \mathit{Empty})\,n \\
= & \quad \{\ \mathit{select}\ \} \\
 & \mathit{merge}\,(\mathit{flatten}\,t, [\,])\,!!\,n \\
= & \quad \{\ \mathit{merge}\ \} \\
 & \mathit{flatten}\,t\,!!\,n \\
= & \quad \{\ \text{define } \mathit{index}\,t\,n = \mathit{flatten}\,t\,!!\,n\ \} \\
 & \mathit{index}\,t\,n
\end{array}
$$

Similarly, we have

$$\mathit{select}\,(\mathit{Empty}, u)\,n = \mathit{index}\,u\,n$$

It is straightforward to derive the characterization

$$
\begin{array}{rcll}
\mathit{index}\,(\mathit{Bin}\,w\,p\,a\,x)\,n & = & \mathit{index}\,w\,n, & \text{if } n < p \\
 & = & a, & \text{if } n = p \\
 & = & \mathit{index}\,x\,(n - p - 1), & \text{if } n > p
\end{array}
$$

of *index*. This completes the derivation: every equation for *select* and *index* reduces the depth of one of the two trees by one, and so the whole program runs in time proportional to the sum of the depths of the two trees.

References

[1] Richard Bird. Lectures on Constructive Functional Programming. In *Constructive Methods in Computer Science*, ed. M. Broy, NATO ASI Series F Volume 55, 1988, Springer-Verlag.

[2] Richard Bird. On Merging and Selection. *Journal of Functional Programming*, **7** (3) p. 349–354, 1997.