Journal of Functional Programming 1 (1): 121–124, January 1991

FUNCTIONAL PEARLS

The *Minout* problem

RICHARD S. BIRD
Programming Research Group, Oxford University

1 Introduction

The problem of computing the smallest natural number not contained in a given set of natural numbers has a number of practical applications. Typically, the given set represents the indices of a class of objects 'in use' and it is required to find a 'free' object with smallest index. Our purpose in this article is to derive a linear-time functional program for the problem. There is an easy solution if arrays capable of being accessed and updated in constant time are available, but we aim for an algorithm that employs only standard lists. Noteworthy is the fact that, although an algorithm using lists is the result, the derivation is carried out almost entirely in the world of sets.

2 Specification

For the specification, let $(-)$ denote set difference. Then we have

$$minout\ x = \sqcap / (nats - x), \tag{1}$$

where *nats* denotes the set of natural numbers, $\sqcap$ is a binary operator returning the smaller of its two arguments, and $\sqcap /$ (pronounced 'min reduce' or 'smallest') applied to a set $\{a_1, a_2, \ldots, a_n\}$ returns

$$a_1 \sqcap a_2 \sqcap \ldots \sqcap a_n.$$

Since the numeric ordering on naturals is well founded, the value of $\sqcap / x$, for an infinite subset x of the naturals, is well defined.

Given a lazy functional language, definition (1) can easily be rendered as an executable function. The essential idea is to represent *nats* as the infinite list [0 ..] and replace $\sqcap /$ by a function that returns the first element of a list. The result is a program that takes $O(n^2)$ steps, where $n = \# x$. If we sort x into increasing order, then this time can be reduced to $O(n \log n)$ steps. Our target, however, is a linear-time algorithm.

3 Derivation

The right-hand side of (1) refers to two sets, only one of which is named as an argument to *minout*. It is reasonable to avoid commitment as to which of these sets will turn out to be the more important for developing an algorithm, so we are led to replace (1) with the more general specification

$$minout\ x \sqcap y = \sqcap / (x - y). \tag{2}$$

https://doi.org/10.1017/S0956796800000083 Published online by Cambridge University Press

(For those in the know, the generalisation can be motivated in another way: the first version of *minout* is not a homomorphism on sets while the second version is.)

In the specialization $x = nats$ of (2), y is a finite, proper subset of x. It seems reasonable to restrict (2) to arguments for which this property is maintained. In particular, it follows that we do not have to introduce a fictitious identity element ∞ of $\sqcap$ for the value of $\sqcap/\{\}$. Hence we shall qualify (2) by requiring

$$y \subset x \tag{3}$$

(where $\subset$ means strict set inclusion) as an invariant on the arguments of *minout*.

Since

$$(x1 \cup x2) - y = (x1 - y) \cup (x2 - y)$$

we have by straightforward calculation that

$$minout\,(x1 \cup x2)\,y = minout\;x1\;y \sqcap minout\;x2\;y. \tag{4}$$

In order to maintain our invariant (3) for this decomposition, we first use the fact that

$$x - y = x - (x \cap y)$$

to rewrite (4) in the form

$$\begin{aligned} minout\,(x1 \cup x2)\,y &= minout\;x1\;y1 \sqcap minout\;x2\;y2 \\ &\quad \text{where } (y1, y2) = (x1 \cap y, x2 \cap y). \end{aligned}$$

Clearly, $y1 \subseteq x1$ and $y2 \subseteq x2$. Now we claim that

$$y \subset x1 \cup x2 \Rightarrow y1 \subset x1 \vee y2 \subset x2 \tag{5}$$

The proof is by a contrapositive argument:

$$\begin{array}{ll} & \neg(y1 \subset x1 \vee y2 \subset x2) \\ = & \qquad \{\text{de Morgan}, y1 \subseteq x1, \text{and}\; y2 \subseteq x2\} \\ & y1 = x1 \wedge y2 = x2 \\ = & \qquad \{\text{definition of}\; y1 \;\text{and}\; y2\} \\ & x1 \cap y = x1 \wedge x2 \cap y = x2 \\ = & \qquad \{\text{set theory}\} \\ & x1 \subseteq y \wedge x2 \subseteq y. \\ = & \qquad \{\text{set theory}\} \\ & x1 \cup x2 \subseteq y. \\ \Rightarrow & \qquad \{\text{set theory}\} \\ & \neg y \subset x1 \cup x2. \end{array}$$

Condition (5) and $y \subset x1 \cup x2$ ensures the invariant (3) holds on one of the argument pairs in *minout* $x1\,y1$ and *minout* $x2\,y2$. In order to eliminate that pair for which (3) may not hold, we need to impose conditions on $x1$ and $x2$, which so far have been completely arbitrary. We shall take $x1$ and $x2$ to be disjoint, nonempty sets, with $x1$ finite and preceding $x2$:

$$\sqcup / x1 < \sqcap / x2. \tag{6}$$

https://doi.org/10.1017/S0956796800000083 Published online by Cambridge University Press

Here, $\sqcup$ takes the greater of its two arguments, so $\sqcap / x$ returns the largest element of the finite set x.

If we now appeal to the simple, but important, linear search theorem, which says that the smallest value existing is the *first* value encountered during a search in increasing order, we can replace the operator $\sqcap$ in the equation for *minout* by a case analysis:

$$\begin{aligned} \mathit{minout}\,(x1 \cup x2)\,y &= \mathit{minout}\,x1\,y1, \quad \text{if } y1 \subset x1 \\ &= \mathit{minout}\,x2\,y2, \quad \text{if } y1 = x1 \\ &\quad \text{where } (y1, y2) = (x1 \cap y, x2 \cap y). \end{aligned}$$

With this step, property (5) guarantees invariant (3) is maintained.

If the above equation for *minout* is to be used as the recursive step of an efficient computation, we have the obligation of providing a base case, together with a proof that the recursion makes progress toward termination.

To determine an appropriate choice for $x1$ and $x2$, we need the fact that $(y1, y2)$ is a partition of y:

$$y1 \cup y2 = y \wedge y1 \cap y2 = \{\,\}. \tag{7}$$

This assertion is an obvious consequence of $y \subset x1 \cup x2$. If we now define $x1$ by the condition

$$\#x1 = \lceil \#y \div 2 \rceil, \tag{8}$$

where $\#x$ denotes the size of the finite set x, we have, in the case $y1 \subset x1$, that

$$\#y1 \leqslant \#x1 - 1 \leqslant \#y \operatorname{div} 2$$

and, in the case $y1 = x1$, that

$$\#y2 = \#y - \#y1 = \#y - \#x1 \leqslant \#y \operatorname{div} 2.$$

In either case, the size of the second argument to *minout* is decreased by a half if $\#y > 1$, and reduced to zero if $\#y = 1$. To guarantee termination it is therefore sufficient to take as base case:

$$\mathit{minout}\,x\,\{\,\} = \sqcap / x.$$

4 Implementation

So far, we have developed a set-theoretic algorithm. To implement it in a functional language we have to choose suitable representations for the two arguments x and y of *minout* $x\ y$. We suppose that y is given as a list with no duplicated elements (so that the length of the list is the size of the set). It also seems reasonable to represent x as a list. However, there is a simpler representation given that, initially, $x = \mathit{nats}$. From (6) it follows that the first argument of *minout* is always a contiguous interval of natural numbers. It is sufficient to represent this interval by its first element. If $\sqcap / x = a$ and

$$b = a + \lceil \#y \div 2 \rceil$$

we have, by (6) and (8), that $\sqcap / x1 = a$ and $\sqcap / x2 = b$. Furthermore, using $\lhd$ to denote the filter operation, we have

$$(y1, y2) = ((<b) \lhd y, (\geqslant b) \lhd y).$$

Using these facts, the final algorithm is:

$$\begin{aligned} minout\, a\, y &= a, && \text{if } y = [\,] \\ &= minout\, a\, y1, && \text{if } \# y1 < b - a \\ &= minout\, b\, y2, && \text{if } \# y1 = b - a \\ &\quad \text{where } (y1, y2) = ((<b) \lhd y, (\geqslant b) \lhd y), \\ &\qquad\qquad\quad b = a + \lceil \# y \div 2 \rceil. \end{aligned}$$

We omit a final optimization that avoids recomputation of the size of the second argument. If $T(n)$ denotes the time to evaluate *minout a y* for a list y of size n, then

$$T(n) = T(n \operatorname{div} 2) + O(n)$$

for $n > 0$, leading to an $O(n)$ algorithm.

5 Postscript

I posed the problem of deriving a linear-time functional program for *minout* at an international workshop on program transformation in the Netherlands in February 1988. Among the audience were authors of transformation systems from Holland, Germany, the USA and Great Britain. I challenged them to use their systems to derive the algorithm, and said I would collate replies. To date I have received just one reply, a far more complicated algorithm than the one given above, and – like the present one – not based on mechanized assistance. Leaving aside the question of my powers of exhortation, the only other conclusion is that existing transformation systems are still quite inadequate for providing reasonable help in the derivation of algorithms.