Journal of Functional Programming 1 (2): 235–43, April 1991

FUNCTIONAL PEARLS

On removing duplicates

RICHARD S. BIRD
Programming Research Group, Oxford University, UK

1 Introduction

The function *remdup* (also called *mkset* in some functional languages) removes duplicates from a given list. The following definition of *remdup* is standard and leads to a quadratic time algorithm

$$\begin{aligned} \mathit{remdup}\,[\,] &= [\,] \\ \mathit{remdup}\,([a] \mathbin{+\!\!+} x) &= [a] \mathbin{+\!\!+} \mathit{remdup}\,(x - [a]). \end{aligned}$$

The operator $(-)$ used in the last expression subtracts one list from another; its definition is

$$x - y = [a \mid a \leftarrow x; a \notin y].$$

Defined in this way *remdup* x returns the list of distinct elements of x in the order in which they first appear in x. In other words, the position of *remdup* x as a subsequence of x is lexically the smallest among all possible solutions.

Now let us change the problem and ask that *remdup* simply return the lexically least solution. Note the subtle difference between this version and the previous one: before, it was the position of the subsequence that was lexically the least, now it is the subsequence itself. To make the distinction clear, consider $x = [1,4,2,4,3]$. With the original definition *remdup* x returns $[1,4,2,3]$; the lexically least subsequence, however, is $[1,2,4,3]$.

The question we are interested in is this: can we also find a quadratic time algorithm for the new problem? The answer turns out to be yes, but justifying it requires a bit of work.

2 Specification

Formally, *remdup* x is the lexically least sequence y satisfying the following properties

1. y is a subsequence of x; in symbols, $y \in \mathit{subs}\,x$.
2. y contains all the elements of x; in symbols, $\mathit{set}\ y = \mathit{set}\ x$, where $\mathit{set}\ z$ is the set of elements in the list z.
3. y does not contain duplicates.

To express these conditions in a form suitable for manipulation, let us define the predicate *ok* by

$$\mathit{ok}\,x\,y = (\mathit{set}\,x = \mathit{set}\,y) \wedge \mathit{nodup}\,y,$$

where *nodup* y is the condition that y does not contain duplicates. Also, let $\sqcap$ denote the operation that selects the lexically smaller of its two arguments. Then we have

$$remdup\ x = \sqcap / ok\ x \lhd subs\ x, \tag{1}$$

where $\sqcap /$ distributes $\sqcap$ over the elements in a list

$$\sqcap / [x_1, x_2, \ldots, x_n] = x_1 \sqcap x_2 \ldots \sqcap x_n,$$

and $p \lhd x$ is an abbreviation for *filter* $p\,x$. Thus (1) reads: the lexically smallest subsequence of x satisfying the predicate *ok* x.

The above notations for reduce and filter, together with the abbreviation $f * x$ for *map* $f\,x$, are introduced elsewhere (Bird, 1987, 1989). They satisfy a number of useful algebraic laws, and are the basic components of a calculus for deriving programs. It would take up too much space to discuss all the laws here, and the reader is referred to the cited references for full details. We shall try and get away with just using the ones we need, appending a brief explanation when necessary.

3 First derivation

The specification of *remdup* given in (1) is incomplete because definitions of the functions *set*, *nodup*, *subs*, and $\sqcap$ are missing. These functions can reasonably be defined in a number of ways, and the best method depends in part on the kind of result we are looking for. For our first derivation we shall head for a standard decomposition of the form

$$\begin{aligned} remdup\,[\,] &= \ldots \\ remdup\,([a] \mathbin{+\!\!+} x) &= \ldots, \end{aligned}$$

so a similar style of definition is appropriate for the missing functions

$$\begin{aligned} subs\,[\,] &= \{[\,]\} \\ subs\,([a] \mathbin{+\!\!+} x) &= subs\ x \cup ([a] \mathbin{+\!\!+}) * subs\ x \\[1ex] set\,[\,] &= \{\,\} \\ set\,([a] \mathbin{+\!\!+} x) &= \{a\} \cup set\ x \\[1ex] nodup\,[\,] &= True \\ nodup\,([a] \mathbin{+\!\!+} x) &= nodup\ x \wedge a \notin x. \end{aligned}$$

In order to define the lexicographic ordering (which we denote by $\leqslant$), it is convenient to introduce first the partial order $\sqsubset$ defined by

$$x \sqsubset y = (\exists k : take\ k\ x = take\ k\ y \wedge x.k < y.k), \tag{2}$$

where $x.k$ denotes the kth element of x (counting from 0). Then we have

$$x \leqslant y = x \in inits\ y \vee x \sqsubset y,$$

where *inits* y denotes the set of initial segments of y. The operator $\sqcap$ is now defined to return the smaller under $\leqslant$ of its arguments.

An important property of the lexicographic ordering is that

$$y \leqslant z \Rightarrow x +\!\!+ y \leqslant x +\!\!+ z,$$

for all x, y and z. In fact, this is the only property of $\leqslant$ we use in the first derivation. It is equivalent to the assertion that $+\!\!+$ distributes forward through $\sqcap$

$$(x +\!\!+ y) \sqcap (x +\!\!+ z) = x +\!\!+ (y \sqcap z). \tag{3}$$

The same law can also be phrased as a property of $\sqcap /$

$$\sqcap /(x +\!\!+)*xs = x +\!\!+ \sqcap /xs,$$

for all x and set of sequences xs, provided xs is not empty. (The restriction that xs be non-empty can be lifted provided we introduce a fictitious identity element $\omega = \sqcap /\{\}$ of $\sqcap$, and make ω the zero element of $+\!\!+$.)

Although $x \leqslant y$ does not imply $x +\!\!+ z \leqslant y +\!\!+ z$, we do have

$$x \sqsubset y \Rightarrow x +\!\!+ z_1 < y +\!\!+ z_2,$$

for all z_1, z_2, and this is why the subordinate partial order $\sqsubset$ is useful in its own right.

Now for the derivation. It is carried out in some detail to show the kind of manipulations that arise in our functional calculus. There are two cases to consider, the first being

$$\begin{array}{ll}
 & \mathit{remdup}\,[\,] \\
= & \{(1)\} \\
 & \sqcap /\mathit{ok}[\,] \lhd \mathit{subs}\,[\,] \\
= & \{\text{given characterisation of } \mathit{subs}\} \\
 & \sqcap /\mathit{ok}\,[\,] \lhd \{[\,]\} \\
= & \{\text{claim: } \mathit{ok}\,[\,]\,[\,] = \mathit{True}\} \\
 & \sqcap /\{[\,]\} \\
= & \{\text{one-point rule: } \oplus/\{a\} = a\} \\
 & [\,].
\end{array}$$

Hence, we have our first equation

$$\mathit{remdup}\,[\,] = [\,].$$

For the second case we argue

$$\begin{array}{ll}
 & \mathit{remdup}\,([a] +\!\!+ x) \\
= & \{(1)\} \\
 & \sqcap /\mathit{ok}\,([a] +\!\!+ x) \lhd \mathit{subs}\,([a] +\!\!+ x) \\
= & \{\text{characterisation of } \mathit{subs}\} \\
 & \sqcap /\mathit{ok}\,([a] +\!\!+ x) \lhd (\mathit{subs}\,x \cup ([a] +\!\!+) * \mathit{subs}\,x) \\
= & \{\lhd \text{ distributes over } \cup\} \\
 & \sqcap /(\mathit{ok}\,([a] +\!\!+ x) \lhd \mathit{subs}\,x \cup \mathit{ok}\,([a] +\!\!+ x) \lhd ([a] +\!\!+) * \mathit{subs}\,x) \\
= & \{\text{promoting } \sqcap / \text{ over } \cup\} \\
 & (\sqcap /\mathit{ok}\,([a] +\!\!+ x) \lhd \mathit{subs}\,x) \sqcap (\sqcap /\mathit{ok}\,([a] +\!\!+ x) \lhd ([a] +\!\!+) * \mathit{subs}\,x)
\end{array}$$

The two terms in the last expression are now simplified separately. We need the following facts about the predicate *ok*; provided $y \in subs\, x$ we have

$$ok\,([a] +\!\!+ x)\,y = (a \in x \rightarrow ok\,x\,y,\quad False), \tag{4}$$

and

$$ok\,([a] +\!\!+ x)\,([a] +\!\!+ y) = (a \in x \rightarrow ok\,(x - [a])\,y \wedge a \notin y,\quad ok\,x\,y). \tag{5}$$

Here use is made of the McCarthy conditional form $(p \rightarrow a, b)$. Proofs of (4) and (5) are straightforward given the definitions of *set* and *nodup*, and are omitted.

Now we argue

$$\begin{array}{ll}
& \sqcap/ok\,([a] +\!\!+ x) \lhd subs\,x \\
= & \{(4)\} \\
& \sqcap/(a \in x \rightarrow ok\,x \lhd subs\,x,\quad \{\,\}) \\
= & \{\text{conditionals}\} \\
& (a \in x \rightarrow \sqcap/ok\,x \lhd subs\,x,\quad \sqcap/\{\,\}) \\
= & \{(1)\text{ and introducing } \omega = \sqcap/\{\,\}\} \\
& (a \in x \rightarrow remdup\,x,\ \omega).
\end{array}$$

The step labelled 'conditionals' above refers to the law

$$f(p \rightarrow a, b) = (p \rightarrow f\,a, f\,b),$$

which is valid provided p is a total predicate.

The second term is simplified as follows

$$\begin{array}{ll}
& \sqcap/ok\,([a] +\!\!+ x) \lhd ([a] +\!\!+) * subs\,x \\
= & \{\text{law}: p \lhd f * xs = f * (p \cdot f) \lhd xs\} \\
& \sqcap/([a] +\!\!+) * (ok\,([a] +\!\!+ x) \cdot ([a] +\!\!+)) \lhd subs\,x \\
= & \{(5)\text{ and law: } (p \wedge q) \lhd xs = p \lhd q \lhd xs\} \\
& \sqcap/([a] +\!\!+) * (a \in x \rightarrow ok\,(x - [a]) \lhd (a \notin) \lhd subs\,x,\quad ok\,x \lhd subs\,x) \\
= & \{\text{claim: see below}\} \\
& \sqcap/([a] +\!\!+) * (a \in x \rightarrow ok\,(x - [a]) \lhd subs\,(x - [a]),\quad ok\,x \lhd subs\,x) \\
= & \{(3)\} \\
& [a] +\!\!+ \sqcap/(a \in x \rightarrow ok\,(x - [a]) \lhd subs\,(x - [a]),\quad ok\,x \lhd subs\,x) \\
= & \{\text{conditionals}\} \\
& (a \in x \rightarrow [a] +\!\!+ \sqcap/ok\,(x - [a]) \lhd subs\,(x - [a]), [a] +\!\!+ \sqcap/ok\,x \lhd subs\,x) \\
= & \{(1)\} \\
& (a \in x \rightarrow [a] +\!\!+ remdup\,(x - [a]),\quad [a] +\!\!+ remdup\,x).
\end{array}$$

The claim referred to in this calculation is the identity

$$(a \notin) \lhd subs\,x = subs\,(x - [a]),$$

The equation is an instance of a more general law

$$all\,p \lhd \cdot subs = subs \cdot p \lhd, \tag{6}$$

in which

$$all\,p = \wedge/\cdot p *.$$

Thus, $all\,p\,x$ is the condition that every element of x satisfies the predicate p. The claim is a special case of (6), since $(a \notin) = all\,(a \neq)$ and $x - [a] = (a \neq) \lhd x$.

Putting the two pieces of the derivation together, we have derived the following program for *remdup*

$$
\begin{aligned}
\mathit{remdup}\,[\,] &= [\,]\\
\mathit{remdup}\,([a] +\!\!+\, x) &= \begin{cases} \mathit{remdup}\,x \sqcap [a] +\!\!+\, \mathit{remdup}\,(x-[a]), & \text{if} \quad a \in x\\ [a] +\!\!+\, \mathit{remdup}\,x, & \text{otherwise.}\end{cases}
\end{aligned} \tag{7}
$$

Though it appears somewhat long, the above derivation can fairly be called routine. Most of the laws used are standard, and much of the derivation is mechanical. The only non-obvious part perhaps resides in the two equations (4) and (5) for *ok*. However, it is here that the problem-specific information really enters the picture, so it is not surprising that some work has to be done to see what is required.

It appears quite difficult to analyse the running time of (7). The reason is that evaluation of $x \sqcap y$ does not require complete evaluation of both x and y unless $x = y$. It can be shown, however, that algorithm (7) takes exponential time in the worst case. Nevertheless, our work has not been wasted, since (7) is used as the starting point for a second derivation, one that achieves the goal of a quadratic time algorithm.

4 Second derivation

In order to reach our goal, let us take a closer look at the computation of (7). Suppose that $a, b \in x$ with $a < b$. Unfolding the computation of $\mathit{remdup}\,([a,b] +\!\!+\, x)$, we obtain

$$
\begin{aligned}
\mathit{remdup}\,([a,b] +\!\!+\, x) = {} & \mathit{remdup}\,x\\
& \sqcap [b] +\!\!+\, \mathit{remdup}\,(x-[b])\\
& \sqcap [a] +\!\!+\, \mathit{remdup}\,(x-[a])\\
& \sqcap [a,b] +\!\!+\, \mathit{remdup}\,(x-[a,b]).
\end{aligned}
$$

Since $a < b$, the second term is lexically larger than the third and can be omitted. Thus, we can write the above expression as follows

$$
\begin{aligned}
\mathit{remdup}\,([a,b] +\!\!+\, x) = {} & [\,] +\!\!+\, \mathit{remdup}\,(x-[\,])\\
& \sqcap [a] +\!\!+\, \mathit{remdup}\,(x-[a])\\
& \sqcap [a,b] +\!\!+\, \mathit{remdup}\,(x-[a,b]).
\end{aligned}
$$

The form of this expression suggests an appropriate generalization: for a strictly increasing sequence w such that $\mathit{set}\,w \subseteq \mathit{set}\,x$, define

$$f\,w\,x = \sqcap / g\,x * \mathit{inits}\,w, \tag{8}$$

where

$$g\,x\,y = y +\!\!+\, \mathit{remdup}\,(x-y). \tag{9}$$

This is a large generalization to take in at one go, but the way it is discovered is standard practice: symbolically unfold the computation to see what subcomputations are required and how they can be combined. That f is a generalization of *remdup* follows from

$$\begin{aligned}
& \mathit{remdup}\, x \\
= \;& \{(9)\} \\
& g\, x\, [\,] \\
= \;& \{\text{definition of } \sqcap/ \text{on singletons}\} \\
& \sqcap/[g\, x\, [\,]] \\
= \;& \{\text{definition of the map operator } *\} \\
& \sqcap/g\, x * [[\,]] \\
= \;& \{\text{definition of } \mathit{inits}\, [\,] \text{ and } (8)\} \\
& f\, [\,]\, x.
\end{aligned}$$

Having found our generalization, we now need to develop an efficient algorithm for f. Again, we head for a standard recursion of the form

$$\begin{aligned}
f\, w\, [\,] &= \ldots \\
f\, w\, ([a] \mathbin{+\!\!+} x) &= \ldots.
\end{aligned}$$

The first step, namely to derive $f\, w\, [\,] = [\,]$, is straightforward and details are omitted. For the second step we have by instantiation in (8) that

$$f\, w\, ([a] \mathbin{+\!\!+} x) = \sqcap/g\, ([a] \mathbin{+\!\!+} x) * \mathit{inits}\, w.$$

To simplify this expression we need to consider the value of $g\, ([a] \mathbin{+\!\!+} x)\, y$. After some routine manipulations using (9) and (7) we obtain

$$g\, ([a] \mathbin{+\!\!+} x)\, y = \begin{cases} g\, x\, y & \text{if } \; a \in y \\ g\, x\, y \sqcap g\, x (y \mathbin{+\!\!+} [a]), & \text{if } \; a \notin y \wedge a \in x \\ g\, x\, (y \mathbin{+\!\!+} [a]), & \text{if } \; a \notin y \wedge a \notin x. \end{cases} \tag{10}$$

Since $g\, ([a] \mathbin{+\!\!+} x)\, y$ is required only for $y \in \mathit{inits}\, w$, and the above expression divides into cases depending on whether $a \in y$, we are led to consider those initial segments of w which might contain a and those that do not. Suppose we split w into two parts $w = u \mathbin{+\!\!+} v$, where

$$(u, v) = (\mathit{takewhile}\, (\leqslant a)\, w, \mathit{dropwhile}\, (\leqslant a)\, w).$$

Note that if $a \in w$, then a will be the last element of u, since w is assumed to be a sequence of values in strictly increasing order.

Next we use the fact that

$$\mathit{inits}\, (u \mathbin{+\!\!+} v) = \mathit{inits}^{-} u \mathbin{+\!\!+} (u \mathbin{+\!\!+}) * \mathit{inits}\, v,$$

where $\mathit{inits}^{-} u$ is the list of initial segments of u excluding u itself, to rewrite $f w\, ([a] \mathbin{+\!\!+} x)$ as the minimum of two terms

$$\begin{aligned}
f\, w\, ([a] \mathbin{+\!\!+} x) = \; & \sqcap/g\, ([a] \mathbin{+\!\!+} x) * \mathit{inits}^{-} u \sqcap \\
& \sqcap/g\, ([a] \mathbin{+\!\!+} x) * (u \mathbin{+\!\!+}) * \mathit{inits}\, v.
\end{aligned} \tag{11}$$

The remainder of the derivation is devoted to simplifying (11). To start with, consider the first term on the right of (11). Since if a appears at all in u, it appears only as the last element of u, we have that no sequence in $\mathit{inits}^{-} u$ contains a. Hence, using (10) and simplifying, we have

$$\sqcap/g\, ([a] \mathbin{+\!\!+} x) * \mathit{inits}^{-} u = \begin{cases} t1 \sqcap t2, & \text{if } \; a \in x \\ t2, & \text{if } \; a \notin x, \end{cases}$$

where

$$\begin{aligned} t1 &= \sqcap / g\,x * inits^{-}u \\ t2 &= \sqcap / g\,x * (\mathbin{+\!\!+} [a]) * inits^{-}u. \end{aligned}$$

A similar simplification yields

$$\sqcap / g\,([a] \mathbin{+\!\!+} x) * (u \mathbin{+\!\!+}) * inits\, v = \begin{cases} t3, & \text{if} \quad a \in u \\ t3 \sqcap t4, & \text{if} \quad a \notin u \wedge a \in x \\ t4, & \text{if} \quad a \notin u \wedge a \notin x, \end{cases}$$

where

$$\begin{aligned} t3 &= \sqcap / g\,x * (u \mathbin{+\!\!+}) * inits\, v \\ t4 &= \sqcap / g\,x * (\mathbin{+\!\!+} [a]) * (u \mathbin{+\!\!+}) * inits\, v. \end{aligned}$$

Putting these results together, we have

$$f\,w\,([a] \mathbin{+\!\!+} x) = \begin{cases} t1 \sqcap t2 \sqcap t3, & \text{if} \quad a \in w \wedge a \in x \\ t2 \sqcap t3, & \text{if} \quad a \in w \wedge a \notin x \\ t1 \sqcap t2 \sqcap t3 \sqcap t4, & \text{if} \quad a \notin w \wedge a \in x \\ t2 \sqcap t4, & \text{if} \quad a \notin w \wedge a \notin x. \end{cases} \tag{12}$$

To simplify the right-hand side of (12) we need the following properties of g (the proofs are left to the reader)

$$y \sqsubseteq z \Rightarrow g\,x\,y \leqslant g\,x\,z \tag{13}$$

$$g\,x\,(u \mathbin{+\!\!+} v) = u \mathbin{+\!\!+} g\,(x - u)\,v. \tag{14}$$

Implications (13) refers to the partial order $\sqsubset$ defined by (2). For ease of reference, here are the relevant terms $t1, \ldots, t4$ again

$$\begin{aligned} t1 &= \sqcap / g\,x * inits^{-}u \\ t2 &= \sqcap / g\,x * (\mathbin{+\!\!+} [a]) * inits^{-}u \\ t3 &= \sqcap / g\,x * (u \mathbin{+\!\!+}) * inits\, v \\ t4 &= \sqcap / g\,x * (\mathbin{+\!\!+} [a]) * (u \mathbin{+\!\!+}) * inits\, v. \end{aligned}$$

To begin with, suppose $t2 = gx(u' \mathbin{+\!\!+} [a])$, for some proper prefix u' of u. Thus $u = u' \mathbin{+\!\!+} [b] \mathbin{+\!\!+} u''$, where $b \leqslant a$ by definition of u. Hence, $u \sqsubseteq u' \mathbin{+\!\!+} [a]$, and so by (13) we have

$$g\,x\,u \leqslant g\,x\,(u' \mathbin{+\!\!+} [a]).$$

But, by definition of $t3$ we have $t3 \leqslant gxu$, and so it follows that

$$t2 \sqcap t3 = t3.$$

Equation (12) can therefore be simplified to read

$$f\,w\,([a] \mathbin{+\!\!+} x) = \begin{cases} t1 \sqcap t3, & \text{if} \quad a \in w \wedge a \in x \\ t3, & \text{if} \quad a \in w \wedge a \notin x \\ t1 \sqcap t3 \sqcap t4, & \text{if} \quad a \notin w \wedge a \in x \\ t2 \sqcap t4, & \text{if} \quad a \notin w \wedge a \notin x. \end{cases} \tag{15}$$

Now we simplify each term. First

$$\begin{aligned}
& t1 \sqcap t3 \\
= & \quad \{\text{definition of } t1 \text{ and } t3\} \\
& \sqcap / g\,x * \mathit{inits}^{-} u \sqcap \sqcap / g\,x * (u +\!\!+) * \mathit{inits}\, v \\
= & \quad \{\text{decomposition of } \mathit{inits}\} \\
& \sqcap / g\,x * \mathit{inits}\,(u +\!\!+ v) \\
= & \quad \{(8) \text{ and } w = u +\!\!+ v\} \\
& f\,w\,x.
\end{aligned}$$

Second

$$\begin{aligned}
& t3 \\
= & \quad \{\text{definition of } t3\} \\
& \sqcap / g\,x * (u +\!\!+) * \mathit{inits}\, v \\
= & \quad \{\text{distributive law};:\ f * . g * = (f.g) *, \text{ and } (14)\} \\
& \sqcap / (u +\!\!+) * g\,(x - y) * \mathit{inits}\, v \\
= & \quad \{(3)\} \\
& u +\!\!+ \sqcap / g\,(x - u) * \mathit{inits}\, v \\
= & \quad \{(8)\} \\
& u +\!\!+ f\,v\,(x - u).
\end{aligned}$$

To simplify the third term $t1 \sqcap t3 \sqcap t4$ of (15), observe that the first element of v, if it exists, is greater than a. Hence

$$u +\!\!+ [a] \sqsubseteq u +\!\!+ v',$$

for any non-empty initial segment v' of v. Using (13) it follows that

$$t3 \sqcap t4 = g\,x\,u \sqcap g\,x\,(u +\!\!+ [a]).$$

Thus

$$\begin{aligned}
& t1 \sqcap t3 \sqcap t4 \\
= & \quad \{\text{above}\} \\
& t1 \sqcap g\,x\,u \sqcap g\,x\,(u +\!\!+ [a]) \\
= & \quad \{\text{definition of } t1 \text{ and } \mathit{inits}\} \\
& \sqcap / g\,x * \mathit{inits}\,(u +\!\!+ [a]) \\
= & \quad \{(8)\} \\
& f(u +\!\!+ [a])\,x.
\end{aligned}$$

To simplify the final term $t2 \sqcap t4$, assume $a \notin w \wedge a \notin x$, so *last* $u < a$. It follows that $u +\!\!+ [a] \sqsubset u' +\!\!+ [a]$ for any proper prefix u' of u, and so

$$\begin{aligned}
& t2 \sqcap t4, \\
= & \quad \{\text{above and } (13)\} \\
& g\,x\,(u +\!\!+ [a]) \\
= & \quad \{(9) \text{ and } a \notin x\} \\
& u +\!\!+ [a] +\!\!+ \mathit{remdup}\,(x - u) \\
= & \quad \{(8)\} \\
& u +\!\!+ [a] +\!\!+ f\,[\,]\,(x - u).
\end{aligned}$$

5 The result

Putting the above pieces together, we have that *remdup* $= f[\,]$, where

$$\begin{aligned} f\,w\,[\,] &= [\,] \\ f\,w\,([a] +\!\!+\, x) &= \begin{cases} f\,w\,x, & \text{if } a \in x \wedge a \in w \\ u +\!\!+ f\,v\,(x - u), & \text{if } a \notin x \wedge a \in w \\ f\,(u +\!\!+ [a])\,x, & \text{if } a \in x \wedge a \notin w \\ u +\!\!+ [a] +\!\!+ f\,[\,]\,(x-u), & \text{if } a \notin x \wedge a \notin w \\ \text{where } (u, v) = \mathit{split}\,a\,w, & \end{cases} \end{aligned}$$

and

$$\mathit{split}\,a\,w = (\mathit{takewhile}\,(\leqslant a)\,w,\ \mathit{dropwhile}\,(\leqslant a)\,w).$$

This program takes $O(n^2)$ steps for a sequence of length n. If we ignore the time spent evaluating $x - u$, then $O(n)$ steps are performed at each iteration since $\# \, w \leqslant n$. This gives $O(n^2)$ steps. However, the total time required for evaluating expressions of the form $x - u$ is also $O(n^2)$ steps, since the sum of the lengths of the us for which this operation is performed is, at most, n. Hence, the total running time is $O(n^2)$ steps.

Well, it was a lot of work to achieve the above result. Is there a simpler solution?

References

Bird, R. S., 1987. An introduction to the theory of lists. In: M. Broy (editor), *Logic of Programming and Calculi of Discrete Design*, NATO Series F, vol 36. Springer-Verlag.

Bird, R. S., 1989. Lectures on constructive functional programming. In: M. Broy (editor), *Constructive Methods in Computing Science*, NATO Series F, vol 52. Springer-Verlag.