1

Maximum subsequence

Problem

Given a sequence of integer numbers $x_0, x_1, \ldots, x_{N-1}$ (not necessarily positive) find a subsequence $x_i, \ldots, x_{j-1}$ such that the sum of numbers in it is maximum over all subsequences of consecutive elements.

We adopt the same notation used in *Programming in the 1990s* [1] and *Programming, The Derivation of Algorithms*[2]: The notation of function application is the "dot" notation with name of function, followed by arguments, each separated by a dot. The notation of quantified expressions has the operator followed by the bounded variables, then a colon followed by the range for the bounded variables and ended with a colon and the actual expression. So

$$(\sum k : i \leq k < j : x_k)$$

corresponds to the more classical mathematical notation $\sum_{k=i}^{j-1} x_k$.
For our derivation steps in predicate calculus we will use the following notation:

$$\begin{array}{cl} & A \\ = & \{\text{reason why A equals B}\} \\ & B \\ \leq & \{\text{reason why B is less than C}\} \\ & C \end{array}$$

If all the numbers are positive then the maximum sum is the sum of the whole initial sequence. If all the numbers are negative then the maximum sum is 0 (by definition 0 is the sum over an empty range). So the interesting case is a sequence with positive and negative numbers in it.

[1] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

[2] A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990

We hope to find an algorithm that visits every number in the sequence only once, so with runtime $O(n)$. Let's introduce some notation: Let's introduce some notation[3] :

$$f.n = (MAXi, j : 0 \le i \le j \le n : s.i.j)$$

with

$$s.i.j = (\sum k : i \le k < j : x_k).$$

We will use properties of quantified expressions as covered in Chapter 3 of *Programming in the 1990s*[4].

$f.N$
$= \ <$ definition of f $>$
$(MAXi, j : 0 \le i \le j \le N : s.i.j)$
$= \ <$ range nesting $>$
$(MAXj : 0 \le j \le N : (MAXi : 0 \le i \le j : s.i.j))$
$= \ <$ defining $p.j = (MAXi : 0 \le i \le j : s.i.j) >$
$(MAXj : 0 \le j \le N : p.j)$
$= \ <$ range split, 1-point rule $>$
$(MAXj : 0 \le j < N : p.j)\ max\ p.N$
$= \ <$ definition of f $>$
$f.(N-1)\ max\ p.N$

We now have a recursive expression for f, which still depends on a newly introduced function p. Let's see if we can get a recursive expression for p too:

$p.N$
$= \ <$ definition of p $>$
$(MAXi : 0 \le i \le N : s.i.N)$
$= \ <$ range split, 1-point rule $>$
$(MAXi : 0 \le i < N : s.i.N)\ max\ s.N.N$
$= \ <$ definition of s and s.N.N = 0 by definition of sum over empty range $>$
$(MAXi : 0 \le i < N : (\sum k : i \le k < N : x_k))\ max\ 0$
$= \ <$ range split in sum $>$
$(MAXi : 0 \le i < N : (\sum k : i \le k < N-1 : x_k) + x_{N-1})\ max\ 0$
$= \ <$ + distributes over max $>$
$(x_{N-1} + (MAXi : 0 \le i < N : (\sum k : i \le k < N-1 : x_k))\ max\ 0$
$= \ <$ definition of p $>$
$(x_{N-1} + p.(N-1))\ max\ 0$

So $f.N = f.(N-1)\ max\ p.N$ and $p.N = (x_{N-1} + p.(N-1))\ max\ 0$. The base cases are $f.0 = 0$ and $p.0 = 0$.

Armed with these recursive relations we can provide a Haskell program that solves the problem:

[3] Our problem can be stated as finding $f.N$ given $x_i \in \mathbb{Z}, 0 \le i < N, N \in \mathbb{N}$.

[4] Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990

Listing 1.1: Haskell code

```
maxSum :: [Int] -> (Int, Int)

maxSum (x:xs) = let (a, b) = maxSum xs
                    c = x + b
                in (max c (max a 0), max c 0)

maxSum [] = (0, 0)
```

The maxSum function calculates the tuple $(f.N, p.N)$.

Bibliography

Edward Cohen. *Programming in the 1990s, An Introduction to the Calculation of Programs*. Springer-Verlag, 1990.

A. Kaldewaij. *Programming, The Derivation of Algorithms*. Prentice Hall, 1990.