SMALL PROGRAMMING EXERCISES 24

M. REM

Department of Mathematics and Computing Science, Eindhoven University of Technology,
Eindhoven, Netherlands

A function \( f \) is called convex if \( f(i) \) does not exceed the average of \( f(i-1) \) and \( f(i+1) \). Examples of convex functions are \( f(x) = 0 \) and \( f(x) = x^2 \). Exercise 55 deals with an array that represents a convex function. We have to check whether the array contains equal consecutive elements. This problem may be solved in \( O(\log N) \) time for an array of size \( N \). I owe this problem to Jan L.A. van de Snepscheut.

A subsequence of a sequence is one that can be obtained by deleting zero or more elements from the original sequence. Exercise 56 asks to write a program that determines the lexicographically greatest subsequence of a given sequence. It allows a solution with an execution time that is linear in the length of the given sequence. This problem is due to Anne Kaldewaij.

Exercise 55: Convex neighbour equality

Find a statement list \( S \) such that

\[
\begin{align*}
\{ & N: \text{int}; \{ N \geq 1 \} \\
& X(i): 0 \leq i \leq N; \text{array of int}; \\
& \{ \{ A: 0 < i < N: 2 \times X(i) \leq X(i - 1) + X(i + 1) \} \} \\
& \{ \{ b: \text{bool}; \\
& S \\
& \{ b \equiv (E: 0 < i < N: X(i) = X(i + 1)) \} \} \\
& \} \\
\end{align*}
\]

Exercise 56: Lexicographically greatest subsequence

We first define the terms subsequence and lexicographic order. Let \( \varepsilon \) denote the empty sequence and let \( a; u \) denote the nonempty sequence consisting of element \( a \) followed by sequence \( u \). For sequences \( s \) and \( t \) we define \( s \) to be a subsequence of \( t \), notation \( s \subseteq t \), by

\[
\begin{align*}
(0) \quad & \varepsilon \subseteq t \\
(1) \quad & \neg(a; u \subseteq \varepsilon) \\
(2) \quad & a; u \subseteq b; v \equiv a; u \subseteq v \lor (a = b \land u \subseteq v)
\end{align*}
\]

0167-6423/90/$03.50 \copyright 1990—Elsevier Science Publishers B.V. (North-Holland)
The lexicographic order for sequences, notation \( s \subseteq t \), is defined by

(0) \( \varepsilon \subseteq t \)
(1) \( \neg(a; u \subseteq v) \)
(2) \( a; u \subseteq b; v \equiv a < b \lor (a = b \land u \subseteq v) \)

Using these notations our exercise may be formulated as

\[
[[ N: \text{int}; \{N \geq 1\} \]
X(i: 0 \leq i < N): \text{array of int};
[[ m: \text{int}; c(i: 0 \leq i < N): \text{array of int};
S \
\{c(i: 0 < i < m) \subseteq X \land (A_s: s \subseteq X: s \subseteq c(i: 0 < i < m))\}
]]
]

**Solution of Exercise 53 (idempotence swap)**

We have to find a solution for \( S \), whose only operations on \( X \) are swaps, in

\[
[[ N: \text{int}; \{N \geq 0\} \]
[[ X(i: 0 \leq i < N): \text{array of int};
\{ (A_i: 0 \leq i < N: 0 \leq X(i) < N) \}
S \
\{ (A_i: 0 < i < N: X(X(i)) = X(i)) \}
]]
]

The standard way of finding an invariant is to replace a constant in the postcondition by a variable. Doing so, we obtain:

\[
P: \quad 0 \leq n \leq N \land (A_i: 0 \leq i < n: X(X(i)) = X(i))
\]

Invariant \( P \) can be established by initializing \( n \) at 0. The guard of the repetition is then \( n \neq N \). Per step of the repetition \( n \) is increased by 1. This increment of \( n \) maintains \( P \) provided \( X(X(n)) = X(n) \). If this equality does not hold the array element at position \( n \) must be swapped with another element, since swapping is the only array operation allowed. We consider to swap it with the element at position \( X(n) \). This operation satisfies

\[
\{ X(n) = A \} \quad X: \text{swap}(X(n), n) \quad \{ X(A) = A \} \quad (0)
\]

The above swap can violate \( X(X(i)) = X(i) \) for \( 0 \leq i < n \) only for values \( i \) that satisfy \( i = A \) or \( X(i) = A \). However, for both values of \( i \) we can conclude \( X(X(i)) = X(i) \) from \( X(A) = A \). Hence, the above swap maintains \( P \). We, therefore, propose
the following program:

\[
S: \quad \{n: \text{int}, n := 0 \} \{P\}
\]
\[
; \quad \text{do } n \neq N
\]
\[
\quad \rightarrow \text{do } X(X(n)) \neq X(n)
\]
\[
\quad \rightarrow X: \text{swap}(X(n), n)
\]
\[
\quad \text{od}
\]
\[
\{ P \land X(X(n)) = X(n) \}
\]
\[
; \quad n := n + 1 \{P\}
\]
\[
\text{od}
\]
\[
\{ P \land n = N, \text{ hence the postcondition holds} \}
\]

The only remaining proof obligation is the termination of the inner repetition. Since the swap in (0) establishes \(X(A) = A\), we choose

\[
(N_i: 0 \leq i < N: \ X(X(i)) \neq X(i))
\]
as our bound function. Notice that the swap in (0) is applied only if \(X(X(n)) \neq X(n)\), i.e. if \(X(A) \neq A\) and \(X(n) \neq n\). Since it establishes \(X(A) = A\) and affects the elements at positions \(A\) and \(n\) only, it decreases the bound function by at least 1. We not only conclude that the inner repetition terminates, but also that the execution time of the whole program is linear in \(N\).

As a side remark we mention that changing the postcondition to \(X(X(i)) \leq X(i)\) would only require to change the guard of the inner repetition to \(X(X(i)) > X(i)\).

**Solution of Exercise 54 (multiple-arcs reduction)**

The representation of directed graph \(G\) may contain multiple arcs. It is the purpose of this exercise to transform the representation of \(G\) into the standard form, which does not have multiple arcs:

\[
[[N,M: \text{int}; \{N \geq 1 \land M \geq 0\}]
\]
\[
\quad b(j: 0 \leq j \leq N), e(i: 0 \leq i \leq M): \text{array of int} ;
\]
\[
\quad \{MSUC(G,b,e)\}
\]
\[
[[b0(j: 0 \leq j \leq N), e0(i: 0 \leq i \leq M): \text{array of int} ;
\]
\[
\quad S
\]
\[
\quad \{SUC(G,b0,e0(i: 0 \leq i < b0(N)))\}
\]
\[
\]}

The postcondition may be written as

\[
(Aj: 0 \leq j < N: Q(j))
\]

where

\[
Q(j) = (\{e0(i) | b0(j) \leq i < b0(j + 1)\} = \{e(i) | b(j) \leq i < b(j + 1)\})
\]
\[
\land (Ah,i: b0(j) \leq h < i < b0(j + 1): e0(h) \neq e0(i))
\]
Again we obtain the invariant by replacing constant $N$ by a variable $n$:

$$P0: \quad 0 \leq n \leq N \land (\forall j: 0 \leq j < n: Q(j))$$

It may be initialized with $n = 0$. The guard of the repetition is $n \neq N$.

We next consider the statement list of the repetition. It should establish $Q(n)$. We are, consequently, interested in set

$$\{e(i) | b(n) \leq i < b(n + 1)\}$$

To record this set we introduce boolean array $c(k: 0 \leq k < N)$ and establish

$$(Ak: 0 \leq k < N: c(k) = k \in \{e(i) | b(n) \leq i < b(n + 1)\})$$

(1)

Then $Q(n)$ can be established by recording the elements of set $\{k | c(k)\}$ in array $e0$, at the positions from $b0(n)$ upwards.

Our goal is to make the execution of the step of the repetition proportional to $h(n + 1) - b(n)$. The execution time of the whole program will then be $O(M + N)$. This excludes the possibility to initialize $c$ at false within the repetition. We, therefore, extend the invariant to

$$P1: \quad P0 \land (\forall k: 0 \leq k < N: \neg c(k))$$

The above analysis would lead to a program of the following structure:

```
c := false; n := 0
; do n := N
    \rightarrow \text{ESTABLISH (1)}
    ; \text{ESTABLISH } Q(n)
    ; \text{MAKE } c \text{ FALSE}
    ; n := n + 1
od
```

The part "MAKE $c$ FALSE" can easily be coded efficiently:

```
[[i: int; i := b0(n)]
 ; do i := b0(n + 1)
    \rightarrow c: (e0(i)) = false; i := i + 1
od
]
```

There is, however, a problem with "ESTABLISH $Q(n)$". This would require $N$ steps, making the whole program $O(N^2)$. We can remedy this by merging it with "ESTABLISH (1)": whenever in "ESTABLISH (1)" an array element $c(k)$ changes from false to true, vertex $k$ is added to array $e0$. We thus arrive at the following
solution:

\[ S: \quad \begin{array}{l}
    \forall n: \text{int}; \ c(k: 0 \leq k < N): \text{array of bool}; \\
    n := 0; \quad \text{do} \ n \neq N \rightarrow c: (n) = \text{false}; \ n := n + 1 \ \text{od} \\
    ; \ b0: (0) = b(0); \ n := 0 \\
    ; \ \text{do} \ n \neq N \\
    \rightarrow \begin{array}{l}
    o \ i, i0: \text{int}; \ i, i0 := b(n), b0(n) \\
    ; \ \text{do} \ i \neq b(n + 1) \\
    \rightarrow \begin{array}{l}
    o k: \text{int}; \ k := e(i) \\
    ; \ \text{if} \ c(k) \rightarrow \text{skip} \\
    \quad \neg c(k) \rightarrow e0: (i0) = k; \ c: (k) = \text{true}; \ i0 := i0 + 1 \\
    \text{fi} \\
    \end{array} \\
    ; \ i := i + 1 \\
    \text{od} \\
    ; \ b0: (n + 1) = i0 \\
    \end{array} \\
    \end{array} \\
    \begin{array}{l}
    ; \ i := b0(n) \\
    ; \ \text{do} \ i \neq b0(n + 1) \rightarrow c: (e0(i)) = \text{false}; \ i := i + 1 \ \text{od} \\
    ; \ n := n + 1 \\
    \text{od} \\
    \end{array} \\
\]

The above is indeed an \( O(M + N) \) algorithm.