OTE: Ohjelmointitekniikka
Programming Techniques

course homepage: `http://www.cs.uku.fi/~mnykanen/OTE/`

Week 03/2009

Exercise 1.

(a) Write the well-known Fibonacci function

$$\mathit{fib}(0) = 0 \tag{1}$$
$$\mathit{fib}(1) = 1 \tag{2}$$
$$\mathit{fib}(m+2) = \mathit{fib}(m+1) + \mathit{fib}(m) \tag{3}$$

as a recursive GCL procedure.

Solution sketch. Follow its definition:

$\{$ pre: $m \in \mathbb{N}$
post: $n = \mathit{fib}(m)$
bound: m $\}$
proc fibo(**value** m:$\mathbb{N}$; **result** n:$\mathbb{N}$);
if $m < 2 \rightarrow$
$n := m$
$[\!]$ $m \geq 2 \rightarrow$
$\mathit{fibo}(m-1, x)$;
$\mathit{fibo}(m-2, y)$;
$n := x + y$
od

(b) Prove your procedure correct.

Solution sketch. The nonrecursive branch is straightforward and omitted.

The guard of the recursive branch ensures that the bound $m > 0$, so this call is permitted to make other calls. The guard also ensures that $m - 1 \in \mathbb{N}$, so the precondition of the first call is satisfied. Moreover, $m - 1 < m$, so the first call also decreases the bound. Hence it is permitted. Similarly, the second call is also permitted. Hence we can apply Theorem 16 to both of them to get:

$\{$ pre $\wedge$ guard $\}$
$\mathit{fibo}(m-1, x)$;
$\{ x = \mathit{fib}(m-1) \wedge$ pre $\wedge$ guard $\}$
$\mathit{fibo}(m-2, y)$;
$\{ y = \mathit{fib}(m-2) \wedge x = \mathit{fib}(m-1) \wedge$ pre $\wedge$ guard $\}$
$n := x + y$
$\{ n = x + y \wedge y = \mathit{fib}(m-2) \wedge x = \mathit{fib}(m-1) \wedge$ pre $\wedge$ guard $\}$

This last assertion implies the *fibo* postcondition, so we are done.

Exercise 2. Let a be an array containing both positive and negative numbers. We want to find its indices l and r such that the sum

$$\sum_{i=l}^{r} a[i]$$

of the slice $a[l \ldots r]$ is as large as possible.

(a) Express this pre- and postcondition formally.

(b) Manipulate the postcondition into the initialization, invariant and guard of the search loop.

(c) Write the corresponding GCL program.

Solution sketch. The precondition is that a is any array of numbers, which needs no formalization. We omit also the background information that a must not be modified. The postcondition can be formalized as

$$\forall \operatorname{lower}(a) \leq p \leq \operatorname{upper}(a) : \forall \operatorname{lower}(a) \leq q \leq \operatorname{upper}(a) : \mathit{interval}(p, q) \leq \mathit{interval}(l, r)$$

where

$$\mathit{interval}(p, q) = \sum_{i=p}^{q} a[i]$$

is the sum for interval $a[p \ldots q]$. This sum can be rewritten as

$$= \mathit{prefix}(q) - \mathit{prefix}(p - 1)$$

where

$$\mathit{prefix}(t) = \sum_{i=\operatorname{lower}(a)}^{t} a[i]$$

is the interval sum for the prefix $a[\operatorname{lower}(a) \ldots t]$ from the beginning of the array a up to index t. Hence when we want to maximize $\mathit{interval}(p, q)$ for the given index q, we only need to know the index $t \leq q$ whose $\mathit{prefix}(t)$ is the smallest possible, and choose $p = t + 1$. Admittedly, it would be cumbersome to carry out this reasoning with formal logic, and we skip it.

By this reasoning, it suffices to maintain the following quantities as our loop invariant:

1. The current index q and the value $Q = \mathit{prefix}(q)$ for it.
2. An index $t \leq q$ where the value $T = \mathit{prefix}(t)$ is as small as possible. Note that they can be computed given the Q above.
3. Indices l and $r \leq q$ where the value $S = \mathit{sum}(l, r)$ is as large as possible. Note that they can be computed given the Q and T above.

We reflect this three-step invariant in the code too:

```
q , Q := lower(a) − 1 , 0;
t , T := q , Q;
l , r , S := q , q , 0;
do q < upper(a) →
    q , Q := q + 1 , Q + a[q + 1];
    if Q < T →
        t , T := q , Q
     [] Q ≥ T →
        skip
    if;
    if Q − T > S →
        l , r , S := t + 1 , q , Q − T
     [] Q − T ≤ S →
        skip
    if
od
```

Exercise 3. In the *saddleback search* problem, we are given a rectangular matrix a with rows $0, 1, 2, \ldots, M-1$ and columns $0, 1, 2, \ldots, N-1$ and an element x which is guaranteed to be in a. Moreover, we also know that the rows and colums of a are ordered: always $a[p][q] \leq a[p][q+1]$ and $a[p][q] \leq a[p+1][q]$. We must find indices i and j such that $a[i][j] = x$.

(a) Express this pre- and and postcondition formally.

(b) Manipulate the postcondition into the initialization, invariant and guard of the search loop.

(c) Write the corresponding GCL program.

(d) How did the ordering help, compared with the general matrix search in the lectures (Figure 12)?

Exercise 4. Redo the three parts (a)–(c) of Exercise 3, but this time x is not guaranteed to be in a, so that the search may also fail.

Solution sketch. Informally, the precondition is that every row r and column c of a is ordered; we omit its straightforward logical formalization. The postcondition is

$$(\exists 0 \leq r < M : \exists 0 \leq c < N : a[r][c] = x) \implies a[i][j] = x$$

or "if x occurrs anywhere in a then $a[i][j]$ is one such occurrence". There is also a background assumption that the algorithm is not allowed to modify a — otherwise it might just first assign $a[0][0] := x$ and then reply $i = 0, j = 0 \ldots$

It is logically equivalent to

$$(\forall 0 \leq r < M : \forall 0 \leq c < N : a[r][c] \neq x) \,\textbf{cor}\, a[i][j] = x.$$

We reformulate it as

$$(i = M \vee j = -1) \wedge (\forall 0 \leq r < M : \forall 0 \leq c < N : r < i \vee c > j \implies a[r][c] \neq x) \,\textbf{cor}\, a[i][j] = x.$$

or "either x is not in any row $r < i$ or column $c > j$ **or**…" to get our final postcondition. Note how their chosen values ensures that this reformulation is equivalent.

Then we initialize i and j to make the '$\forall$'-part TRUE, so it becomes our loop invariant, whereas the negation of the other parts become the guard:

$\{$ inv: $\forall 0 \le r < M : \forall 0 \le c < N : r < i \vee c > j \Longrightarrow a[r][c] \neq x$
bound: i is incremented or j decremented $\}$
$i\,,\ j\ :=\ 0\,,\ N-1\,;$
do $i \neq M \wedge j \neq -1$ **cand** $a[i][j] \neq x \rightarrow$
  if i can be incremented $\rightarrow$
    $i\ :=\ i+1$
  $[\!]$ j can be decremented $\rightarrow$
    $j\ :=\ j-1$
  fi
od

Calculating the condition $\mathrm{wp}(i := i+1, \text{inv})$ which permits incrementing i reveals that we must have $a[i][0 \ldots j-1] < x$. Since row i is ordered by the precondition, $a[i][j] < x$ is enough to guarantee it.

Similarly $\mathrm{wp}(j := j-1, \text{inv})$ yields the guard $a[i][j] > x$ for that branch. Hence the final answer is

$i\,,\ j\ :=\ 0\,,\ N-1\,;$
do $i \neq M \wedge j \neq -1$ **cand** $a[i][j] \neq x \rightarrow$
  if $a[i][j] < x \rightarrow$
    $i\ :=\ i+1$
  $[\!]$ $a[i][j] > x \rightarrow$
    $j\ :=\ j-1$
  fi
od

This loop runs for just $\mathcal{O}(M+N)$ steps, whereas general matrix search would have taken $\mathcal{O}(M \cdot N)$ steps instead.

Exercise 5. Consider the subroutine

$\{$ pre: TRUE
post: $x = y + z$ $\}$
proc sum(**result** $x : \mathbb{R}$; **value** $y, z : \mathbb{R}$);
$\mathcal{B}$

Verify that the new value of p after the call $sum(p, p, p)$ is twice its old value before the call.

Solution sketch. Note that we cannot use the given sum specification as it, because Theorem 17 allows **value** parameters (here x and y) in the postcondition only if their values in the call (here p and p) do not mention the values (here also p) of the **result** parameters (here x). Hence we must revert to using their ghosts:

$\{$ pre: $Y = y \wedge Z = z$
post: $x = Y + Z$ $\}$
proc sum(**result** $x : \mathbb{R}$; **value** $y, z : \mathbb{R}$);
$\mathcal{B}$

Let also P be the ghost for the initial value p before the call. Theorem 16 gives

$\{$ pre$[y \leftarrow p, z \leftarrow p] \wedge \iota \quad = \quad (Y = p \wedge Z = p) \wedge \iota$ $\}$
$sum(p, p, p)$
$\{$ post$[x \leftarrow p] \wedge \iota \quad = \quad (p = Y + Z) \wedge \iota$ $\}$

for any ι which does not mention p. We can get the desired postcondition

$$\begin{aligned} p &= 2 \cdot P \\ &= P + P \end{aligned}$$

from this postcondition with

$$\iota = (Y = P \wedge Z = P).$$

This precondition in turn ensures this ι by adding the conjunct $p = P$, the definition of P. Hence the call has been verified.