Wednesday, 6 April 2022 11:31

C12.3 Prove that the following maximum-finding algorithm is totally correct with respect to precondition P and postcondition Q, where:

P: *$n \geq 1$ is an integer and $A[1:n]$ is an integer array[1] with n elements.*

Q: $a = \max\{A[i] \mid i = 1, \ldots, n\}$.

S {
```
k ← 1;
a ← A[1];
while k < n do
    k ← k + 1;
    if a < A[k] then a ← A[k]
od
```
}

(*Hint:* For weak correctness, first find an appropriate loop invariant.)

Precondition $P(w)$: $w = \langle n=k, A \rangle$
where $k \geq 1$ A is a list with n elements

Post-condition $Q(w')$: $w' = \langle a = \max\{A[i]\} \rangle$

Total correctness: the algorithm S transforms $\}$ weak corr.
w into w'. and that it halts.
strong correctness

Weak correctness:

Set the loop invariant t: be
the predicate $\} = \{ a = \max\{A[1, \_, k]\} : 1 \leq k \leq n \}$

Base case: by definition, since a list of length 1
only has a maximal element.
Induction ass: assume holds for k^{th} iteration. Then
on $k+1^{st}$ step we look into the next element
of our list, $A[k+1]$. Then either

(i) $A[k+1] \leq a$, then we don't assign
a new value, and by induction assumption then a remembers the maximal element.

(ii) $A[k+1] > a$, and in that case a gets assigned the maximal value
of $A[1, \_, k+1]$ and $a = \max\{A[1, \_, k+1]\}$.

So a is a loop invariant and we get weak correctness.

Strong correctness:

Let our ranking function
$r(A, n, k) = n - k$.
For any iteration of the
while-loop the value
of n decreases by 1, sin
the assignment
$k \leftarrow k + 1$
increases the value of k
Therefore S should halt
precisely after n iteration

Since the algorithm satisfies weak and s
correctness, it is also totally correct.

12.7 Establishing strong correctness

- The remaining part of the story is establishing the strong correctness of a program S that has already been validated as weakly correct w.r.t. a specification $\{P\}S\{Q\}$.
- This entails proving that all the loops in program S terminate, given an initial state ω that satisfies precondition $P(\omega)$.
- This is the theoretically most challenging part of the verification task, because it is essentially Turing's Halting Problem. In practice it is however seldom difficult.
- The usual approach is to determine, for each loop in the program, a *ranking function* $r(\omega)$ that maps admissible (given the precondition) states of the program to the nonnegative integers or some other well-founded ordered set,[4] and prove that in each iteration of the loop the value of $r(\omega)$ decreases.

[4] A partially-ordered set is *well-founded* if the ordering has no infinite descending sequences.

Exponentiation:

```
{n ≥ 0}
m ← 1; i ← n;
while i > 0 do
    m ← 2 * m;
    i ← i − 1
od
{m = 2^n}
```

In the case of the exponentiation program, one may clearly choose rank function $r(n, m, i) = i$.

Euclid's algorithm:

```
{m0 ≥ 0, n0 ≥ 0}
m ← m0; n ← n0;
if m < n then m ↔ n;
while n > 0 do
    m ← m − n;
    if m < n then m ↔ n;
od
{m = gcd(m0, n0)}
```

For Euclid's algorithm, one can choose rank function $r(m_0, n_0, m, n) = m + n$.

Another option would be to rank-order the pairs (m, n) according to: $(m, n) \prec (m', n')$ if $m < n$ or $(m = n$ & $n < n')$.

s

ue

by 1..

-

·l .

strong