CSC240

Lecture 13 week 9

Author: Joseph Siu

Email: joseph.siu@mail.utoronto.ca

Date: March 13, 2024

MERGESORT(A$[1..n], n$)
if $n > 1$ then
    $m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$
    A$' \leftarrow$ A$[1..m]$
    A$'' \leftarrow$ A$[m+1..n]$
    MERGESORT(A', m)
    MERGESORT$(A'', n-m)$
    A $\leftarrow$ MERGE(A',m,A",n-m)

1 Correctness of Algorithms

An algorithm is correct if it satisfies its specifications

Specifications are often written using

1. **Precondition**
    - certain facts must be true before an execution of the algorithm begins
    - it can describe what inputs are allowed
2. **Postcondition**
    - certain facts must be true when an execution of the algorithm ends
    - often it describes the correct ouput or possible correct outputs for a given input
3. **Termination**
    - the algorithm halts when the preconditions are true

Example 1. Search array A for value k.
Precondition: The elements of A$[1..n]$ and k are from the same domain (so we can compare them) (i.e. we can't compare int with str)
Postcondition: Return an integer i such that $1 \leq i \leq n$ and A$[i] = k$ or 0 if no such index i exists.
Specification does not say anything about the algorithm just tells us the relation between the inputs and the outputs.

For this example, consider the algorithm:
    A$[1] \leftarrow k$
    return 1
or
    $k \leftarrow$ A$[1]$
    return 1
This works but it is not what we intended.
So,
Postcondition: A is not changed, k is not changed.

Example 2. Specifications for Binary Search Algorithm
Precondition: $A[1..nn]$ is sorted in non decreasing order

$$\forall i \in \mathbb{Z}^+ . \forall j \in \mathbb{Z}^+ . [(i < j \leq n) \text{ IMPLIES } (A[i] \leq A[j])]$$

[a]

Postcondition: same as for searching an array

[a]elements of $A[1..n]$ and k must be from a totally ordered domain.

2 Sorting an array

Precondition: the elements in A$[1..n]$ are from a totally ordered domain

Postcondition:

- The multiset of elements in A$[1..n]$ is not changed
- The elements in A$[1..n]$ afterwrads are a permutation of the elements in A$[1..n]$ before the algorithm was in A$[1..n]$ before the algorithm has executed
- The elements of A are in nondecreasing order.

3 Merging two arrays

Precondition: A$[1..m]$ and B$[1..n]$ are sorted in nondecreasing order. The elements in A$[1..m]$ and B$[1..n]$ are from the same totally ordered domain.

Postcondition:

- Outputs an array C$[1..m+n]$ such that the multiset of elements in C$[1..m+n]$ is the union of the multisets of elements in A$[1..m]$ and B$[1..n]$.
- A and B are not changed.
- C is sorted in nondecreasing order.

4 Proving Correctness of Recursive Algorithms

Usually use induction

Example 3. Correctness of MERGESORT.
Assuming the correctness of MERGE.
For $n \in \mathbb{N}$, let $P(n) =$ "for all array A$[1..n]$ with elements from a totaly ordered set.
If MERGESORT(A$[1..n]$) is performed, then it eventually halts / returns, at which time A is sorted in nondecreasing order and the multiset of element A is unchanged."

Proof.
Let $n \in \mathbb{N}$ be arbitrary.
Let A$[1..n]$ be an arbitrary array with elements from a totally ordered set.
Consider MERGESORT(A$[1..n]$).
Base Case $n = 0$ and $n = 1$
the test on line 1 fails, algorithm terminates immediately, so A is unchanged. Trivially A is sorted in nondecreasing order.
Induction Case $n > 1$:
test on line 1 is true and $m = \left\lfloor \frac{n}{2} \right\rfloor$ from line 2.
$m, n - m < n$.
$\text{A} = \text{A}' \cup \text{A}''$.

By the induction hypothesis after lines $5+6$, A$'$ and A$''$ are sorted in nondecreasing order and the multisets of elements in A$'$ and A$''$ are unchanged.
Preconditions of MERGE are satisfied, so after line 7, A is sorted in nondecreasing after, and the multiset of elements in A is the union of the multiset of elements in A$' \cup$ A$''$, which is the original multiset of elements in A.
So, by generalization, $P(n)$ is true.
Since n was arbitrary, by induction $\forall n \in \mathbb{N}.P(n)$.
Hence MERGESORT is correct.

Quod Erat Dem■

5 Correctness of Iterative Algorithms

Partial Correctness: <u>if</u> the preconditions hold, the algorithm is executed, and it terminates, <u>then</u> the postconditions hold.

<u>Termination</u>: if the preconditions hold, and the algorithm is executed, then it eventually terminates.

Total correctness = partial correctness and termination.

For iterative algorithms, they are typically proved separately.

```
M(m, n)
1 z ← 0
2 w ← m
3 while w ≠ 0 do
4     z ← z + n
5     w ← w − 1
6 return z
```

It performs multiplication by repeated addition

Precondition: $m \in \mathbb{N}, n \in \mathbb{C}$

Postcondition: $z = m \times n$, m and n are unchanged. [1]

~~Immediately after the i^{th} iteration of the while loop, $w = m - i$ and $z = n \times i$.~~

Correction: immediately after the 0^{th} iteration means immediately before the 1^{st} iteration

Let $P(i) =$ " if the loop is executed at least i times, the immediately after the i^{th} iteration $w = m - i$ and $z = n \times i$."

Lemma 1

Let $M \in \mathbb{Z}, n \in \mathbb{C}$,

$$\forall i \in \mathbb{N}.P(i)$$

proof of Lemma 1.
Let w_i and z_i denote the values of w and z immediately after the i^{th} iteration
Base Case:
Initially $w_0 = m = m - 0$ by line 2,

[1] we can see there are no assert to m or n, thus they are trivially unchanged.

$z_0 = 0 = n \times 0$ by line 1,
so $P(0)$ is true.
Induction Case:
Let $i \in \mathbb{N}$ be arbitrary and assume $P(i)$ is true.
Assume the loop is executed at least $i+1$ times. Then $w_i = m - i$ and $z_i = n \times i$.
From lines 4 and 5, we have $z_{i+1} = z_i + n = n \times i + n = n \times (i+1)$ and $w_{i+1} = w_i - 1 = m - i - 1 = m - (i+1)$.
Hence $P(i+1)$ is true.
Hence by induction $\forall n \in \mathbb{N}.P(n)$.

QUOD ERAT DEM■

To show total correctness, we first show the partial correctness (we are going to prove it in 2 ways).

Corollary 1 – Partial Correctness

Let $m \in \mathbb{Z}$ and $n \in \mathbb{C}$, if $\mathrm{M}(m, n)$ is non of it halts then it returns $z = n \times m$.

Proof of Corollary 1.
Suppose the loop halts imemdiately after the i^{th} iteration of the loop
From the termination condition of the loop (line 3) $w_i = 0$
By the lemma, $w_i = m - i$ and $z_i = n \times i$, so $i = m$ and $z_i = n \times m$
Hence $z_i = m \times n$ is returned.

QUOD ERAT DEM■

A **loop invariant** is a predicate that is true each time a particular place in the loop is reached.

Often we consider the beginning / end of iterations of the loop.

Assume this, unless specified otherwise.

Lemma 2

$z = n \times (m - w)$ is a loop invariant. [a]

[a] does not contain i where i is the number of iterations

Proof of Lemma 2.
Initially from lines 1 and 2, $z = 0$ and $w = m$ so $n \times (m - w) = 0 = z$.
Consider an arbitrary iteration of the loop
Let w' and z' be the values of w and z at the beginning of the iteration and let w'' and z'' be the values of w and z at the end of the iteration.
Suppose the claim is true at the beginning of the iteration. Then $z' = n \times (m - w')$.
From lines 4 and 5 the code $w'' = w' - 1$ and $z'' = z' + n$, so

$$\begin{aligned} n \times (m - w'') &= n \times (m - (w' - 1)) \\ &= n \times (m - w') + n \\ &= z' + n \\ &= z'' \end{aligned}$$

Hence the claim is true at the end of the iteraion.

By induction, $z = n \times (m - w)$ after every iteration.

Quod Erat Dem■

Another Proof of Corollary 1.
From the termination condition of the loop (line 3) $w = 0$
Since $z = n \times (m - w)$ is a loop invariant
$z = n \times (m - w) = n \times m$ when the loop terminates.

Quod Erat Dem■

Now, we show the termination. From there we can conclude total correctness.

Lemma 3 – Termination

If $n \in \mathbb{C}$ and $m \in \mathbb{N}$ and $\mathrm{M}(m, n)$ is run, then it eventually halts.
Namely,

$$\forall n \in \mathbb{C}.\forall m \in \mathbb{N}.(\mathrm{M}(m, n) \text{ eventually halts}).$$

Informal Proof of Lemma 3.
Before the loop is executed w is set to $m \in \mathbb{N}$.
Each iteration, w is decreased by 1, so it is a smaller natural number.
Hence w must eventually reach 0. This is the exit condition of the loop. Therefore the loop terminates and the algorithm returns.

Quod Erat Dem■

More formal Proof of Lemma 3.
Suppose the loop does not terminate.
 Let n, m be arbitrary.
 Let w_i be the value of w immediatley after the i^{th} iteration of the loop.
 From line 5, we know $w_{i+1} = w_i - 1$
 For all $i \in \mathbb{N}$, let $Q(i) =$ "$w_i \in \mathbb{N}$"
 Base Case:
  Since $w_0 = m \in \mathbb{N}$ by assumption, $Q(0)$ is true.
 Induction Case:
  Let $i \geq 0$ be arbitrary and assume $Q(i)$ is true
  Since the loop does not terminate $w_i \neq 0$, then $w_i \in \mathbb{Z}^+$. Since $w_{i+1} = w_i - 1$, it follows that $w_{i+1} \in \mathbb{N}$. Hence $Q(i + 1)$ is true.
 By induction $\forall i \in \mathbb{N}.Q(i)$.
 Then w_0, w_1, w_2 is a sequence of natural numbers such that w_{i+1}, w_i for all $i \in \mathbb{N}$.
 By the well ordering principle, this sequence has a smallest element w_k.
 But $w_{k+1} < w_k$
 This contradicts the definition of w_k,
 Thus the loop (and algorithm M) eventually terminates.
Since n, m were arbitrary. By generalization, the lemma is true.

Quod Erat Dem■

Therefore, we conclude that algorithm M is totally correct.