CSC 240 LECTURE 9

CORRECTNESS OF ALGORITHMS

Consider the following algorithm:

```
M(m,n)
z ← 0
w ← m
while w ≠ 0 do
   z ← z + n
   w ← w - 1
return(z)
```

What does it mean that an algorithm is correct?
That it satisfies its specifications.

Specifications are often written using preconditions and postconditions.

A precondition is a statement (an assertion) involving the variables used in the algorithm.

It says that certain facts must be true before an execution of the algorithm begins.

It can describe which inputs are allowable.

A postcondition is also a statement about variables used in the algorithm.
It says that certain facts must be true when an execution of the algorithm ends.
Often a postcondition describes the correct output (or possible correct outputs) for a given input.

We are also concerned about termination of an algorithm:
it halts (provided the preconditions hold).

partial correctness:
if the preconditions hold, the algorithm is executed, and it eventually halts, then the postconditions hold.
termination:
if the preconditions hold, and the algorithm is executed, then
it eventually halts.

total correctness = partial correctness + termination

Algorithm M performs multiplication by repeated addition.
It starts z at 0 and adds n to z a total of m times.

```
M(m,n)
z ← 0
w ← m
while w ≠ 0 do
   z ← z + n
   w ← w - 1
return(z)
```

Precondition: ~~m,n ∈ ℤ~~ $n \in \mathbb{Z}$ and $m \in \mathbb{N}$
Postcondition: z = m x n

Is M correct with respect to these specifications?
NO. If m < 0, then M doesn't halt.

Example 1
specifications for search of an array A for a value/key k

Precondition:
The elements of A[1..n] and k are from the same domain.
Postcondition:
return an integer i such that 1 ≤ i ≤ length(A)
and A[i] = k, if such an integer i exists; otherwise return 0.
A and k must not be changed.

Note: the specifications don't say how the search is to be performed.
It just gives the relationship between inputs and outputs.

Consider the algorithm

```
A[1] ← k
return(1)
```

It satisfies these specifications.

So does

k ← A[1]
return(1)

Example 2
specifications for binary search algorithm
Precondition: A[1..n] is sorted in non-decreasing order:
$\forall i \in \mathbb{Z}^+. \forall j \in \mathbb{Z}^+.$ [(i < j ≤ length(A)) IMPLIES (A[i] ≤ A[j])].
Postcondition: return an integer i such that 1 ≤ i ≤ length(A)
and A[i] = k, if such an integer i exists; otherwise return 0.
A and k are not changed.

If the precondition isn't satisfied, an algorithm may return
an incorrect answer, it may run forever, or it may perform
an illegal operation such as dividing by 0.

Example 3
specifications for sorting an array A
Precondition: the elements in A are from a totally ordered domain,
i.e. comparable to one another
Postcondition: the elements of A are in nondecreasing order
and the elements in A after the algorithm has executed
are a permutation of the elements in A before the algorithm
executed.

Example 4
specifications for merging two arrays A and B
Preconditions:
A and B are in nondecreasing order
The elements in A and B are from a totally ordered domain
Postconditions:
C is in nondecreasing order
the multiset of elements in C is equal to the union of
the multiset of elements in A and the multiset of elements in B.
A and B are unchanged.

Proving Correctness of Recursive Algorithms
usually use induction

Example 5

```
MERGESORT(A,n)
1. if n > 1 then
   2. m ← ⌊n/2⌋
   3. U ← A[1..m]
   4. V ← A[m+1..n]
   5. MERGESORT(U,m)
   6. MERGESORT(V,n-m)
   7.  A ← MERGE(U,V)
8. return
```

Proof of correctness, by induction:

For $n \in \mathbb{N}$, let P(n) = "for all arrays A[1..n]
with elements from a totally ordered domain,
if MERGESORT(A,n) is performed,
then it eventually halts, at which time A is sorted in nondecreasing order
and the multiset of elements in A is unchanged."

Let $n \in \mathbb{N}$ be arbitrary. Let A[1..n] be an arbitrary array with elements from a totally ordered domain.

Consider MERGESORT(A,n).

Base cases: n = 0,1.
The test on line 1 fails and A is unchanged.
Also note that A is sorted in non-decreasing order.
By generalization, this holds for all arrays A[1..n].
Hence P(n) is true.

Induction cases: n > 1.
Suppose P(n') is true for all natural numbers n' < n.
The test on line 1 succeeds and m = ⌊n/2⌋ from line 2.

Then m, n-m < n.
By the induction hypothesis,
after MERGESORT(U,m) and MERGESORT(V,n-m) are performed,
U and V are each sorted in nondecreasing order
the multiset of elements in U is the multiset of elements in A[1..m],
and the multiset of elements in V is the multiset of elements in A[m+1..n].

Note that the union of the multiset of elements in U and V is the set of elements originally in A.
It follows from the correctness of MERGE that,
after A ← MERGE(U,V) is performed,
the elements in A are sorted in nondecreasing order
and the multiset of elements in A is the union of
the multiset of elements in U and V.
Hence the multiset of elements in A is unchanged.

It follows by generalization that P(n) is true
and, by induction, that $\forall n \in \mathbb{N}$. P(n) is true.
Hence MERGESORT is correct.

Proving Correctness of Iterative Algorithms

termination and partial correctness are often proved separately.

Example 6


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

The hard part is dealing with loops.
What can we say about the value of the variables immediately
after the i'th iteration of the loop?

w = m - i
z = n x i

Convention: immediately after the 0'th iteration
is equivalent to immediately before the first iteration.

We can prove these statements by induction on i:
Let P(i) = "if the loop is executed at least i times, then,
immediately after the i'th iteration w = m - i and z = n x i.

LEMMA 1 Let $m \in \mathbb{N}$ and $n \in \mathbb{Z}$. For all $i \in \mathbb{N}$, P(i) is true.
Proof:
Let w_i and z_i denote the values of w and z immediately after the i'th iteration.

Initially, $w_0 = m = m - 0$ and $z_0 = 0 = n \times 0$,
from lines 1 and 2, so P(0) is true.

Let $i \geq 0$ and assume P(i) is true.
Then $w_i = m - i$ and $z_i = n \times i$.
Assume the loop is executed at least i+1 times.
From lines 4 and 5, $z_{i+1} = z_i + n = n \times i + n = n \times (i+1)$ and
$w_{i+1} = w_i - 1 = m - i - 1 = m - (i+1)$. Hence P(i+1) is true.

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

COROLLARY 2 (Partial correctness) Let $m \in \mathbb{N}$ and $n \in \mathbb{Z}$.
If MULTbyAdd(m,n) is run and it halts, then, when it halts, $z = n \times m$.

Proof:
Suppose the loop halts immediately after the i'th iteration.
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 $z_i = n \times m$.

Another proof of partial correctness:

Combine $w = m - i$ and $z = n \times i$ to get $z = n \times (m-w)$.
This is nice because it doesn't involve i.
It is the same immediately after every iteration of the loop.

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 the loop,
but, sometimes, it is easier to consider other places.
If we don 't specify the location, we mean it is true
at the beginning/end of every iteration of the loop.

LEMMA 3 $z = n \times (m - w)$ is a loop invariant.

Proof: Initially, from lines 1 and 2,
z = 0 and w = m so n x (m - w) = 0 = z.

Consider an arbitrary iteration of the loop.
Let w' and z' denote the values of w and z at the beginning
of the iteration and
let w" and z" denote 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 x (m - w').

From lines 4 and 5 of the code, w" = w' - 1 and z" = z' + n,
so n x (m - w") = n x (m - (w' - 1)) = n x (m - w') + n = z' + n = z".
Thus the claim is true at the end of the iteration.

Hence, by induction, z = n x (m - w) after every iteration.

COROLLARY 4 (Partial correctness) Let $m \in \mathbb{N}$ and $n \in \mathbb{Z}$.
If the algorithm is run and it halts, then, when it halts, z = n x m.

Proof: From the termination condition of the loop (line 3), w = 0.
By Lemma 3, z = n x (m - w) = n x m.

Termination

Proving termination is easy if no while loops, repeat until loops,
or recursion, assuming any subprograms that are called terminate.
For while loops and repeat until loops, to show termination, it suffices to
show that some non-negative integral quantity decreases each time
through the loop.

MULTbyAdd(m,n)
1. z ← 0
2. w ← m
3. while w ≠0 do
    4. z ← z + n
    5. w ← w - 1
6. return(z)

LEMMA 5 (Termination) If $n \in \mathbb{Z}$, $m \in \mathbb{N}$, and MULTbyAdd(m,n) is run, it eventually halts.

Proof:
Before the loop is executed, w is set to the natural number m.
Each iteration of the loop 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 halts.

Somewhat more formally:

Suppose the loop doesn't terminate.
Let w_i be the value of w immediately before the i'th iteration of the loop.
From line 5, $w_{i+1} = w_i - 1$.

Let Q(i) = "$w_i \in \mathbb{N}$".
Since $w_1 = m \in \mathbb{N}$, Q(1) is true.

Let $i \geq 1$ be arbitrary and suppose Q(i) is true.
Since the loop doesn't terminate, $w_i \neq 0$. Thus $w_i \in \mathbb{Z}^+$.
Since $w_{i+1} = w_i - 1$, it follows that $w_{i+1} \in \mathbb{N}$.

Hence Q(i+1).
By induction, $\forall i \in \mathbb{Z}^+$. Q(i).

Thus w_1, w_2 ... is a sequence of natural numbers.
By the well-ordering principle, the set of elements in this sequence has a smallest element, w_i.
But $w_{i+1} < w_i$.
This contradicts the definition of w_i.
Thus, the loop eventually terminates.

Example 7

MULTbyShift&Add(m,n)
1. w ← m
2. y ← n
3. z ← 0
4. while w ≠ 0 do

5. if w mod 2 = 1 then z ← z + y
6. w ← w div 2 %quotient when w is divided by 2
7. y ← 2 x y %or y ← y + y
8. return(z)

Note that division and multiplication by 2 are implemented using shift.

Preconditions: $m \in \mathbb{N}$, $n \in \mathbb{Z}$
Postconditions: z = n x m

How to come up with loop invariants?

1. look at examples:
m = 11, which is 1011 in base 2
n = 20, which is 10100 in base 2

Let w_i, y_i and z_i denote the values of w, y, and z immediately after the i'th iteration of the while loop.

i	w_i	binary w_i	mod 2	y_i	z_i
0	11	1011	1	20	0
1	5	101	1	40	$20 = n = 1\ y_0$
2	2	10	0	80	$60 = 2n + n = 1\ y_1 + 1\ y_0$
3	1	1	1	160	$60 = 2n + n = 0\ y_2 + 1\ y_1 + 1\ y_0$
4	0	0	0	320	$220 = 8n + 2n + n = 1\ y_3 + 0\ y_2 + 1\ y_1 + 1\ y_0$

$w_i = \lfloor m / 2^i \rfloor$

$y_i = n\ 2^i$
z_i = ?

2. use your understanding of how the algorithm is supposed to work

```
    10100
    x1011
------------
    10100
   10100
  00000
10100
-------------
11011100
```

Let m[k-1] ... m[1] m[0] denote the binary representation of m, so
$m = \sum \{m[a]\ 2^a \mid 0 \le a \le k-1\}$
where $m[a] \in \{0,1\}$ for $0 \le a \le k-1$.

For example, $m = 11_{10} = 1011_2$, $k = 4$, $m[3] = 1$, $m[2] = 0$, $m[1] = 1$, $m[0] = 1$.

w_i is obtained from m by deleting the i least significant bits
$w_i = \sum \{m[a]\ 2^{\{a-i\}} \mid i \le a \le k-1\}$
$w_i\ 2^i = \sum \{m[a]\ 2^a \mid i \le a \le k-1\}$

Since $m = \sum \{m[a]\ 2^a \mid 0 \le a \le k-1\}$ and
$w_{i+1}\ 2^{\{i+1\}} = \sum \{m[a]\ 2^a \mid i+1 \le a \le k-1\}$ it follows that
$m - w_{i+1}\ 2^{\{i+1\}} = \sum \{m[a]\ 2^a \mid 0 \le a \le i\}$.

The least significant bit of w_i is m[i], so $w_i \bmod 2 = m[i]$.

Initially, y = n. Each time through the loop, the value of y is doubled.
Thus $y_i = n\ 2^i$.

$z_0 = 0$ from its initialization on line 3
$z_{i+1} = z_i + m[i]\ y_i$ since $z \leftarrow z + y$ if $w \bmod 2 = 1$
$= z_i + m[i]\ n\ 2^i$

using repeated substitution
$z_{i+1} = m[0]\ n\ 2^0 + m[1]\ n\ 2^1 + \ldots + m[i]\ n\ 2^i$
$= n \sum \{m[a]2^a \mid 0 \le a \le i\}$
$= n\ (\ m - w_{i+1}\ 2^{\{i+1\}}\ )$
$= nm - w_{i+1}\ x\ y_{i+1}$

MULTbyShift&Add(m,n)
1. w ← m
2. y ← n
3. z ← 0
4. while w ≠ 0 do
   5. if w mod 2 = 1 then z ← z + y
   6. w ← w div 2 %quotient when w is divided by 2
   7. y ← 2 x y %or y ← y + y
8. return(z)

LEMMA 6 If $m \in \mathbb{N}$ and $n \in \mathbb{Z}$, then $z = n \times m - w \times y$ and $w \in \mathbb{N}$ are loop invariants.

Proof:
Note that m and n aren't changed by the algorithm, since there are no assignments to m and n.

Initially, $z = 0$, $w = m \in \mathbb{N}$ and $y = n$ by lines 1,2, and 3
so $n \times m - w \times y = 0 = z$.

Consider an arbitrary iteration of the loop.
Let w',y' and z' denote the values of w, y, and z at the beginning of the iteration and
let w", y", and z" denote the values of w, y, and z at the end of the iteration.

Suppose the claim is true at the beginning of the iteration.
Then $z' = n \times m - w' \times y'$ and $w' \in \mathbb{N}$.

From line 7 of the code $y" = 2y'$.

Case 1: w' is odd.
Then, from lines 5 and 6 in the code, $z" = z' + y'$ and $w" = (w'-1)/2 \in \mathbb{N}$, so

$$\begin{aligned} n \times m - w" \times y" &= n \times m - (w'-1)/2 \times 2y' \\ &= n \times m - (w'-1) \times y' \\ &= n \times m - w' \times y' + y' \\ &= z' + y' \\ &= z". \end{aligned}$$

Case 2: w' is even.
Then, from lines 5 and 6 in the code, $z" = z'$ and $w" = w'/2 \in \mathbb{N}$, so

$$\begin{aligned} n \times m - w" \times y" &= n \times m - w'/2 \times 2y' \\ &= n \times m - w' \times y' \\ &= z' \\ &= z". \end{aligned}$$

Since w' is either even or odd, the claim is true at the end of the iteration.

Hence, by induction, $z = n \times m - w \times y$ is a loop invariant.

Where do we use the assumption that m ∈ ℕ?
In the statement w' is either even or odd.

LEMMA 6 (corrected) If m ∈ ℕ and n ∈ ℤ, then and z = n x m - w x y are loop invariants.

COROLLARY 7 (partial correctness) Let m ∈ ℕ and n ∈ ℤ.
If MULTbyShift&Add(m,n) is run and it halts, then, when it halts, z = n x m.

Proof:
Assume MULTbyShift&Add(m,n) is run and it halts.
From the termination condition of the loop (while w ≠ 0 do)
w= 0 when the loop terminates.
By Lemma 6, z = n x m - w x y. Hence z = n x m.

Q: Why does MULTbyShift&Add(m,n) terminate?
w gets smaller every iteration, provided w ≥ 1.