Prog & Alg I (COMS10002)
Week 6 - Intro to Program Correctness

Dr. Oliver Ray
Department of Computer Science
University of Bristol

Wednesday 5th November, 2014

Reasoning about Loops

- Reasoning about programs with loops is complicated by the fact that variables change value during a loop
- We can use the notation x_i to denote the value of variable x on the i'th iteration of the loop
- We could do proof by induction on the number of loop iterations, but it is usually easier to use ***invariants***
- Informally, an invariant is a logical property that is
  - set-up by the code preceding the loop
  - re-established on any iteration of the loop
  - ensures correctness of the code following the loop

(these correspond to the initialisation, maintenance and termination properties from Kerstin's slides)

Pre, Post and Mid conditions

- To reason about a function, we first need to specify its
  - ***pre-conditions***: the logical properties we assume to be given when the function is called
  - ***post-conditions***: the logical properties we require to hold of the returned value
- If the program is correct then we should be able to prove the pre-conditions ***logically entail*** the post-conditions
- Typically this is done by annotating the program with
  - ***mid-conditions***: the logical properties we know to be true at that point in program execution
- We must show each mid-condition implies the next by a combination of ***forward*** and ***backward*** reasoning

Example: Exponentiation

```
r(x,n) {
    // given n>=0 & ¬( x=0 & n=0 )
    int a=1;
    // ???
    while (n!=0) {n--; a*=x;}
    // ???
    return a;
    // return x^n
}
```

PRE CONDITION

MID CONDITIONS

POST CONDITION

Reasoning Forward

```
r(x,n) {

    // given n>=0 & ¬( x=0 & n=0 )

    int a=1;

    // a=1 & n>=0 & ¬( x=0 & n=0 )

    while (n!=0) {n--; a*=x;}

    // ???

    return a;

    // return x^n

}
```

reasoning forwards: easy step

PRE CONDITION

MID CONDITIONS

POST CONDITION

Reasoning Backward

```
r(x,n) {

    // given n>=0 & ¬( x=0 & n=0 )

    int a=1;

    // a=1 & n>=0 & ¬( x=0 & n=0 )

    while (n!=0) {n--; a*=x;}

    // a=x^n

    return a;

    // return x^n

}
```

PRE CONDITION

MID CONDITIONS

POST CONDITION

reasoning forwards: easy step

reasoning backwards: easy step

Linking the proof up

reasoning forwards: easy step

???

reasoning backwards: easy step

```
r(x,n) {

    // given n>=0 & ¬( x=0 & n=0 )

    int a=1;

    // a=1 & n>=0 & ¬( x=0 & n=0 )

    while (n!=0) {n--; a*=x;}

    // a=x^n

    return a;

    // return x^n

}
```

PRE CONDITION

MID CONDITIONS

POST CONDITION

Can we find an invariant to complete the proof?

But, there is a Problem!

```
s(x,n) {

    // given n>=0 & ¬( x=0 & n=0 )

    int a=1;

    // a=1 & n>=0 & ¬( x=0 & n=0 )

    x=n=1;

    // a=x^n

    return a;

    // return x^n

}
```

PRE CONDITION

MID CONDITIONS

POST CONDITION

this spec is trivially satisfied by code that does NOT perform exponentiation!

We really need to remember the initial variable values!

Example: Corrected

reasoning forwards: easy step

???

reasoning backwards: easy step

```
r(x,n) {

    // given n>=0 & ¬( x=0 & n=0 ) & x=x0 & n=n0

    int a=1;

    // a=1 & n>=0 & ¬( x=0 & n=0 ) & x=x0 & n=n0

    while (n!=0) {n--; a*=x;}

    // a=x0^n0

    return a;

    // return x0^n0

}
```

So, now can we find an invariant to complete the proof?

General Properties of Loop Invariants

```
// Pre

...

// Inv

while (b) {

    // Inv & b

    ...

    // Inv
}

// Inv & ¬b

...

// Post
```

<u>Initialisation</u>
if Preconditions are true initially then Invariant must be set-up before the loop runs

<u>Maintenance</u>
if Invariant and loop Condition are both true then Invariant must be re-established after the loop Body runs

<u>Termination</u>
on exiting the loop the Invariant and negation of the loop Condition are both true and these are sufficient to prove the post-condition

Requirements for our Loop Invariant

```
// a=1 & n>=0 & ¬( x=0 & n=0 ) & x=x0 & n=n0

// Inv

while (n!=0) {

    // Inv & n!=0

    n--; a*=x;

    // Inv

}

// Inv & n=0

// a=x0^n0
```

We need an Inv that allows us to prove the above properties

Guessing our Loop Invariant

- It is always sensible to choose some example inputs and consider the variable values on each iteration

#	x	n	a
0	3	5	1
1	3	4	3
2	3	3	9
3	3	2	27
4	3	1	81
5	3	0	243

```
x=x0
n>=0
n<=n0
a=x0^(n0-n)
...
```

intuitively these seem to be the most useful

- An invariant should refer to the current and initial values only and it should be true on all iterations

Proving our Loop Invariant

reasoning forwards

INIT

MAINT

n->(n-1)

a->(a*x)

TERM

reasoning backwards

```
// given n>=0 & ¬( x=0 & n=0 ) & x=x0 & n=n0
int a=1;
// a=1 & n>=0 & ¬( x=0 & n=0 ) & x=x0 & n=n0
// a=x0^(n0-n) & x=x0
while (n!=0) {
    // a=x0^(n0-n) & x=x0 & n!=0
    // (a*x)=x0^(n0-(n-1)) & x=x0
    n--;
    // (a*x)=x0^(n0-n) & x=x0
    a*=x;
    // a=x0^(n0-n) & x=x0
}
// a=x0^(n0-n) & x=x0 & ¬(n!=0)
// a=x0^n0
return a;
// return x0^n0
```

Can we show INIT, MAINT and TERM?

INIT

$a=1\ \&\ n>=0\ \&\ \neg(\ x=0\ \&\ n=0\ )\ \&\ x=x_0\ \&\ n=n_0$

logically entails (|=)

$a=x_0\text{^}(n_0-n)\ \&\ x=x_0$

because

from $n=n_0$ we have $n-n_0=0$ (subtracting n_0 from both sides)

from $n-n_0=0$ we have $x_0\text{^}(n_0-n)=1$ (by defn. of exponents)

from $a=1$ we have $a=x_0\text{^}(n_0-n)$ (by transitivity of equality)

and from $x=x_0$ we have $x=x_0$ (trivially)

MAINT

$a=x_0\text{^}(n_0-n)$ & $x=x_0$ & $n!=0$

logically entails (|=)

$(a*x)=x_0\text{^}(n_0-(n-1))$ & $x=x_0$

because

from $a=x_0\text{^}(n_0-n)$

we have $(a*x)=x*x_0\text{^}(n_0-n)$ (multiplying both sides by x)

so $(a*x)=x_0*x_0\text{^}(n_0-n)$ (using the fact $x=x_0$)

so $(a*x)=x_0\text{^}(1+(n_0-n))$ (by properties of exponents)

so $(a*x)=x_0\text{^}(n_0-(n-1))$ (by simple algebra)

and from $x=x_0$ we trivially have $x=x_0$

Note that we need to use the fact $x=x_0$ in this proof!

TERM

$a=x_0$^(n_0-n) & $x=x_0$ & ¬(n!=0)

logically entails (|=)

$a=x_0$^n_0

because

from ¬(n!=0)

we have n=0 (by double negation elimination)

so -n=-0 (by negating both sides)

so -n=0 (by properties of 0)

so $n_0-n=n_0$ (by adding n_0 to both sides)

so x_0^$(n_0-n)=x_0$^n_0 (by simple algebra)

and $a=x_0$^n_0 (using the fact $a=x_0$^(n_0-n))

So the proof is complete!

- Hooray! But there a couple of points to note:
- Strictly these proofs assume we take $0 \wedge 0 = 1$ (since expressions like $x_0 \wedge (n_0 - n)$ reduce to $0 \wedge 0$ even if x and n are not both initially zero)
- This is not a problem but it would be easier to just drop the precondition $\neg(x=0 \,\&\, n=0)$ which is not actually required for correctness or used in the proof
- Alternatively we could change the invariant to

  $$(n=n_0 \rightarrow a=1)\,\&\,(n \neq n_0 \rightarrow a=x_0 \wedge (n_0-n))\,\&\,(x=x_0)$$

- Exercise: Rework the proof using the above invariant.