19CSE205 – Program Reasoning

Jevitha KP

Lecture 11, 12, 13 –
Forward vs Backward Reasoning,
Weakest Preconditions – Conditionals

Credits

- Adapted from :
  - Dr. Bharat Jayaraman, University of Buffalo, CSE449-459 Software verification course, Spring 2020.

Sample Code

- x = 17;
- y = 42;
- z = x+y;
- Write the facts that are true at every point in the program

Assertion

- { true }
- x = 17;
- y = 42;
- z = x+y;

Assertion

- { true }
- x = 17;
- { x = 17 }
- y = 42;
- z = x+y;

Assertion

- { true }
- x = 17;
- { x = 17 }
- y = 42;
- { $x = 17 \land y = 42$ }
- z = x+y;

Assertion

- $\{ \text{true} \}$
- $x = 17;$
- $\{ x = 17 \}$
- $y = 42;$
- $\{ x = 17 \land y = 42 \}$
- $z = x+y;$
- $\{ x = 17 \land y = 42 \land z =59 \}$

Assertion

- An assertion is a logical formula inserted at some point in a program.
- It is presumed to hold true at that point in the program.

Precondition and Postcondition as Assertions

- A precondition is an assertion inserted prior to execution
- A postcondition is an assertion inserted after execution.
- {true} is the precondition
- $\{ x = 17 \wedge y = 42 \wedge z = 59 \}$ is the postcondition.
- All other assertions are called intermediate assertions.
- They serve as steps as you reason between precondition and postcondition, Eg: like the intermediate steps in a math problem to solution.

Forward reasoning

- Previous example - forward reasoning.
- Simulated the execution of the program, considering each statement in the order they would be executed.
- Disadvantage - assertions may accumulate a lot of irrelevant facts as you move through the program.
- Don't know which parts of the assertions will come in handy to prove something later and which parts won't.
- End up listing everything we know about the program.

Backward Reasoning

- When we write a block of code, we have a clear idea of what’s supposed to be true after it executes (ie) we know the postcondition already,
- To prove that the expected postcondition will indeed hold true given the appropriate precondition.
- For this reason, backward reasoning is often more useful than forward reasoning – Weakest precondition

Backward Reasoning Process

- Push the postcondition up through the statements to determine the precondition.
- Start by writing down the postcondition you want at the end of the block.
- Then look at the last statement in the block and ask, “For the postcondition to be true after this statement, what must be true before it?”

Backward Reasoning Process

- Keep going until you've reached the top of the statement list.
- Whatever must be true before the first statement is the precondition.
- It is guaranteed that if this precondition is satisfied before the block of code is executed, then the postcondition will be satisfied afterward.

Example

- x = y;
- x = x + 1;
- { x > 0 }

Example

- $x = y;$
- $\{ x + 1 > 0 \}$
- $x = x + 1;$
- $\{ x > 0 \}$

Example

- $\{ y + 1 > 0 \}$
- $x = y;$
- $\{ x + 1 > 0 \}$
- $x = x + 1;$
- $\{ x > 0 \}$

- False case:
- Eg: if y = -1 – if precondition is false
- x= -1 (x=y)
- x= 0 (x=x+1)
- x=0 – post condition (x>0) is also false

- True case:
- Eg: if y = 2 – if precondition is true
- x= 2 (x=y)
- x= 3 (x=x+1)
- x= 3 – post condition (x>0) is also true

Weakest Preconditions for conditional statements

Conditional Statements

- Statement : if (B) S1 else S2
- How do we define : WP(if (B) S1 else S2, O)
  - If-Part: _____________
  - Else-Part: _____________

Conditional Statements

- Statement : if (B) S1 else S2
- Defining WP : WP(if (B) S1 else S2, O)
  - If-Part: wp(S1,O)
  - Else-Part: wp(S2,O)

Conditional Statements

- Statement : if (B) S1 else S2
- Defining WP : WP(if (B) S1 else S2, O)
  - If-Part: wp(S1,O)
  - Else-Part: wp(S2,O)
- WP(if (B) S1 else S2, O) =

Conditional Statements

- Statement : if (B) S1 else S2
- Defining WP : WP(if (B) S1 else S2, O)
  - If-Part: wp(S1,O)
  - Else-Part: wp(S2,O)

- WP(if (B) S1 else S2, O) =

  B ==> wp(S1,O)

  &&

  not(B) ==> wp(S2,O)

Conditional Statements

- Statement : if (B) S1
- How do we define : WP(if (B) S1, O)
  - If block : ____________
  - Else block : ____________

Conditional Statements

- Statement : if (B) S1
- How do we define : WP(if (B) S1, O)
  - If block : wp(S1,O)
  - Else block : O

Conditional Statements

- Statement : if (B) S1
- How do we define : WP(if (B) S1, O)
  - If block : wp(S1,O)
  - Else block : O
- WP(if (B) S1 , O) =

  B ==> wp(S1,O)

  &&

  not(B) ==> O

Another definition of WP for Conditional Statements

- WP(if (B) S1 else S2, O) =

  B && wp(S1,O)

  ||

  not(B) && wp(S2,O)

- WP(if (B) S1 , O) =

  B && wp(S1,O)

  ||

  not(B) && O

Checking Equivalence

- We can check equivalence between:
- B && wp(S1, O) || not(B) && wp(S2, O) and
- (B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

- Abbreviate:
- wp(S1, O) ➔ P
- wp(S2, O) ➔ Q

Checking Equivalence

- We can check equivalence between:
- B && wp(S1, O) || not(B) && wp(S2, O) and
- (B ==>wp(S1, O)) && (not(B) ==> wp(S2, O))

- Using Alt-Ergo, we can quickly check equivalence :

(B && P) || (not(B) && Q) <==>

(B ==>P) && (not(B) ==> Q)

Checking Equivalence

```
logic p,q,b: prop

goal a:
    (b and p) or  (not(b) and q) <->
    (b -> p) and (not(b) -> q)
```

```
File "try-alt-ergo-file", line 4, characters 5-68: Valid (0.1060) (7 steps)
(goal a)
```

Example - 1

- Output condition : x >= 0
- Input condition : i=0
- Program:

If(i >=0) then

x= i;

else

 x = -i;

Find WP and check I=>WP.

Example - 1

- O: x>=0 ; B : i>=0; S1: x= i; S2: x= -i;
- WP(if(B) S1 else S2, O)
- b ==> wp(S1,O) && not(b) -> wp(S2,O)
- WP(S1,O) = (x>=0){x=i} = (i>=0)
- WP(S2,O) = (x>=0) {x=-i} = (-i>=0)=(i<=0)
- (i>=0) ==> (i>=0) && not(i>=0) ==> (i<=0)
- (i>=0) ==> (i>=0) && (i<0) ==> (i<=0)
- Alternative WP:
- (i>=0) && (i>=0) || (i<0)&&(i<=0)

Example - 1

- WP Ans:
- i>=0 => (i>=0) && (i<0) => (i<=0)

I==> WP

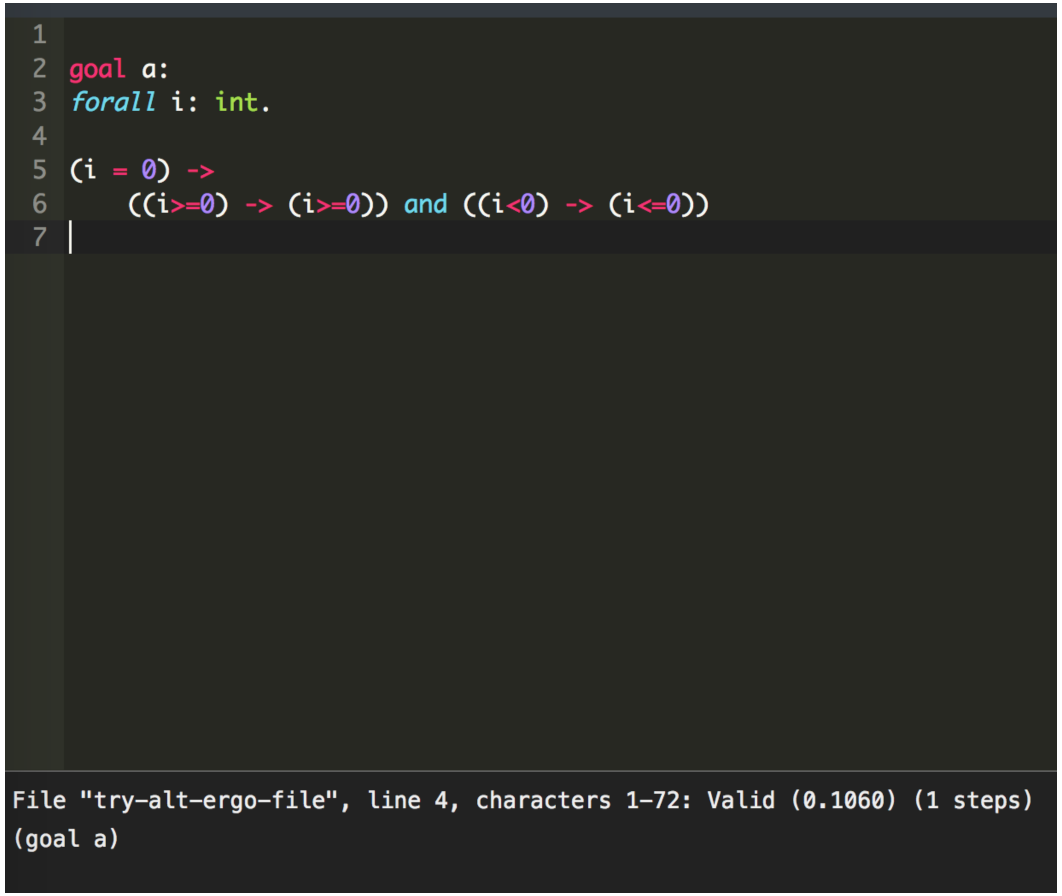
- i=0 ==> (i>=0) ==> (i>=0) && (i<0) ==> (i<=0)

Alternative I==> WP

- i=0 ==> (i>=0) && (i>=0) || (i<0)&&(i<=0)

Alt-ergo

```
goal a:
forall i: int.
(i = 0) ->
  ( i>=0 -> i>=0 )
     and
  ( i<0 -> i<=0 )
```



Alt-ergo – Alternative WP

goal a:

forall i: int.

(i = 0) ->

(i>=0 and i>=0)

or

(i<0) and i<=0)

```
goal a:
forall i: int.

(i = 0)
    ->
    ((i>=0) and (i>=0))
     or
    ((i<0) and (i<=0))
```

```
File "try-alt-ergo-file", line 2, characters 1-89: Valid (0.0960) (1 steps)
(goal a)
```

Example - 2

- If(x<5)
- x=x*x
- else
- x=x+1
- O : x>=9

- B => wp(S1,O) && not(B) => wp(S2,O)
- B: (x<5)
- WP(S1,O) = x>=9 {x=x*x} = x*x >= 9
- WP(S2,O) = x>=9{x=x+1} = x+1>=9 = x>=8
- (x<5) => (x*x>=9) && not(x<5) => (x>=8)
- (x<5)=> (x*x>=9) && (x>=5) => (x>=8)

Example 2 – Alt-Ergo

- WP : (x<5)=> (x*x>=9) && (x>=5) => (x>=8)
- I : x =3
- I => WP
- (x=3) => (x<5)=> (x*x>=9) && (x>=5) => (x>=8)

- Alt-ergo:
- logic x:int
- goal a:
- (x=3) ->
-     (x<5 -> x*x>=9)
-     and
-     (x>=5 -> x>=8)

```
1 logic x:int
2
3 goal a:
4 (x=3) ->
5         (x<5) -> (x*x>=9)
6         and
7         (x>=5) -> (x>=8)
```

```
File "try-alt-ergo-file", line 4, characters 1-69: Valid (0.1230) (2 steps)
(goal a)
```

Example 2 – Alt-Ergo

- Alt-ergo: Invalid
- logic x:int
- goal a:
- (x=2) ->
-     (x<5 -> x*x>=9)
-     and
-     (x>=5 -> x>=8)

```
1 logic x:int
2
3 goal a:
4     (x=2) ->
5         (x<5  -> x*x>=9)
6         and
7         (x>=5 -> x>=8)
```

```
File "try-alt-ergo-file", line 4, characters 5-81: I don't know (0.1270) (3
steps) (goal a)
```

Example - 3

if(x!=0)

  z=x

else

z=x+1

- O : z>0

- B=> WP(s1,o) && not(B) => wp(s2,o)
- Wp(s1,o) = z>0 {z=x} = x>0
- Wp(s2,o) = z>0 {z=x+1} = x+1>0
- (x!=0)=> x>0 && not(x!=0) => x+1>0
- WP: (x!=0) => x>0 && (x=0) => x+1>0
- WP: (x!= 0) && x>0 || x=0 && x>-1

Example 3 – Alt-ergo

- WP: (x!=0) => x>0 && (x=0) => x+1>0
- I : x= 2
- Alt-ergo : (!= should be written as <> in Alt-Ergo)
- x=2 ->
- (x<>0 -> x>0)
- And
- (x=0 -> x+1>0)

```
1 logic x:int
2
3 goal a:
4     (x=2) ->
5             (x<>0 -> x>0)
6             and
7             (x=0 -> x+1>0)
```

```
File "try-alt-ergo-file", line 4, characters 5-88: Valid (0.1180) (2 steps)
(goal a)
```

Example 3 – Alt-ergo

- WP: (x!=0) => x>0 && (x=0) => x+1>0
- I : x= -1
- Alt-ergo : (!= should be written as <> in Alt-Ergo)
- x=-1 ->
- (x<>0 -> x>0)
- And
- (x=0 -> x+1>0)

```
1 logic x:int
2
3 goal a:
4 x=-1 ->
5         (x<>0 -> x>0)
6         and
7         (x=0 -> x+1>0)
```

```
File "try-alt-ergo-file", line 4, characters 1-63: I don't know (0.1160) (3
steps) (goal a)
```

Example 4

if(a==b)

b = 2*a+1

else

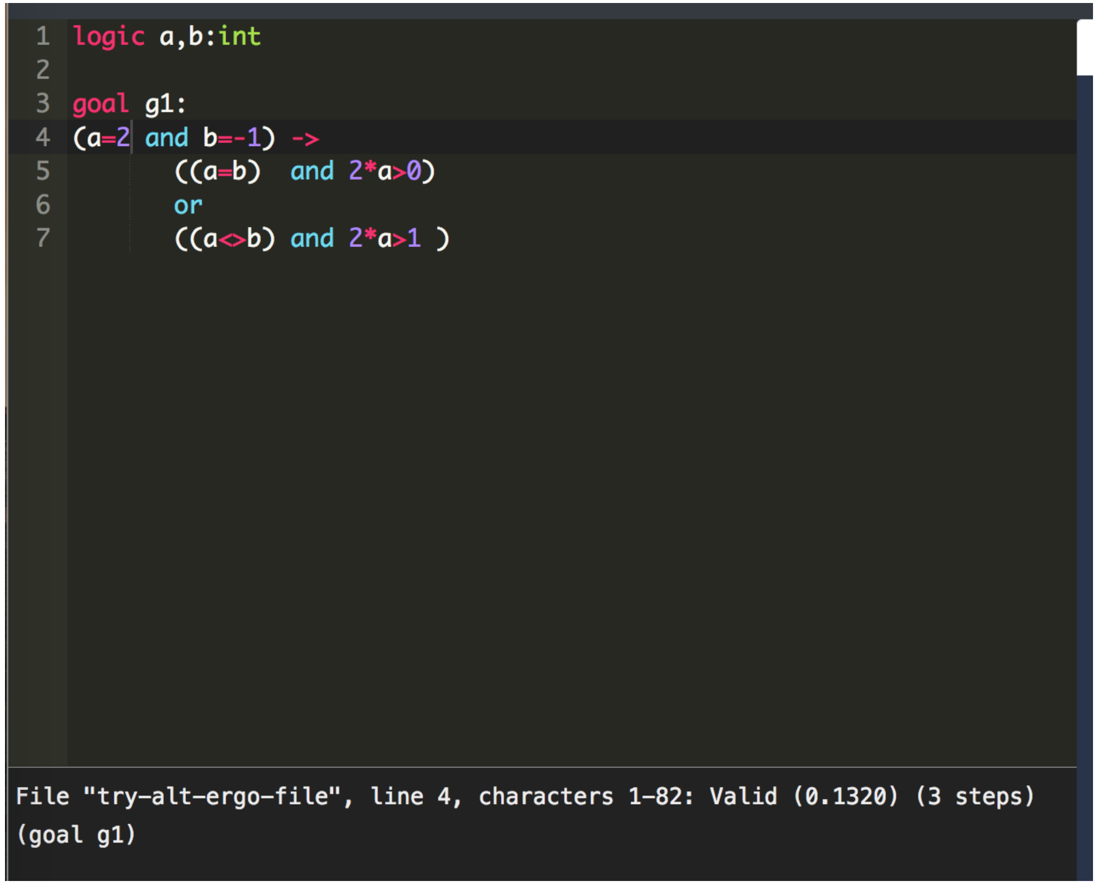
b = 2*a

O : b>1

- B=> WP(s1,o) && not(B) => wp(s2,o)
- Wp(s1,o) = b>1 {b=2*a+1} = 2*a+1>1
- Wp(s2,o) = b>1 {b=2*a} = 2*a>1
- (a=b)=> 2*a+1>1 && not(a=b) => 2*a>1
- WP: (a=b)=> 2*a>0 && not(a=b) => 2*a>1
- WP: (a=b) && 2*a>0 || not(a=b) && 2*a>1

Example 4 – Alt-Ergo

- I : a = 2 and b=-1
- (a=2 and b=-1) => ((a=b) && 2*a>0) || (not(a=b) && 2*a>1)
- Alt-ergo:
- logic a,b:int
- goal g1:
- (a=2 and b=-1) ->
- ((a=b) and 2*a>0)
- or
- ((a<>b) and 2*a>1)



Example 4 – Alt-Ergo

- I : a = -1 and b=2 - Invalid
- (a=-1 and b=2) => (a=b && 2*a>0) || (not(a=b) && 2*a>1)
- Alt-ergo:
- logic a,b:int
- goal g1:
- (a=2 and b=-1) ->
- ((a=b) and 2*a>0)
- or
- ((a<>b) and 2*a>1)

```
1 logic a,b:int
2
3 goal g1:
4 (a=-1 and b=2) ->
5         ((a=b)  and 2*a>0)
6         or
7         ((a<>b) and 2*a>1 )
```

```
File "try-alt-ergo-file", line 4, characters 1-82: I don't know (0.1210) (3
steps) (goal g1)
```

Comparison of Multiple-if vs Else-if

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

```
@requires marks = 75
@ensures  grade = B
@program {
  if (marks > 90)
        grade = A;
  else  if (marks > 70)
            grade = B;
        else if (marks > 50)
                grade = C;
             else grade = F;
}
.
```

Which one is easy to verify???

Multiple-if

```
If (a == b)
    S1;
If (b == c)
   S2;
If( c == a)
   S3;
```

How many execution paths??

Multiple-if

```
If (a == b)
    S1;
If (b == c)
    S2;
If( c == a)
    S3;
```

Work out here...

A=1,b=2,c=3 --> None

A=1,b=1,c=3 -->S1

None

S1

S2

S3

All

Multiple-if

```
If (a)
    S1;
If (b)
    S2;
If(c)
    S3;
```

None

S1

S2

S3

S1,S2

S1,S3

S2,S3

S1,S2,S3

Multiple-if

```
If (a == b)
     S1;
If (b == c)
    S2;
If( c == a)
    S3;
```

$2 \wedge n$

Multiple-if

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

- How many execution paths??

Multiple-if

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

- How many execution paths??
- S1
- S1,S2
- S1,S2,S3
- S1,S2,S3,S4

Multiple-if

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

- How many execution paths??
- S1
- S1,S2
- S1,S2,S3
- S1,S2,S3,S4

Multiple-if

```
if (B1)
  s1;

if (B2)
  s2;

{O}
```

Try to derive the weakest precondition for this program

Multiple-if

```
if (B1)
  s1;
{(B2 && wp(s2,O)) || (not B2 && O)}
if (B2)
  s2;
{O}
```

Multiple-if

{(B1 && wp(s1,P)) || (not B1 && P)}

```
if (B1)
  s1;
```

P= {(B2 && wp(s2,O)) || (not B2 && O)}

```
if (B2)
  s2;
```

{O}

Multiple-if

{(B1 && wp(s1,{
(B2 && wp(s2,O))
|| (not B2 && O)
}))
||
(not B1 && {
(B2 && wp(s2,O)) || (not B2 && O)})
}

- Larger the number of execution paths, greater is the size of verification conditions.
- The size of verification condition generated increases exponentially (eg. For n cases, its 2^n

Else-if

```
If(B1)
  S1;
else if(B2)
  S2;
..
..
Sn;
```

- How many execution paths for n cases ?

Else-if

```
If(B1)
  S1;
else if(B2)
  S2;
..
..
Sn;
```

- How many execution paths for n cases ?

ONLY n paths!

s1 or s2 or s3 or …..sn

Else-if

```
If(B1)
  S1;
else if(B2)
  S2;
else if(B3)
  S3;
..
else
Sn;
```

B1 && wp(S1,O)
|| not(B1) && B2 && wp(S2,O)
|| not(B1) && not(B2) && B3 && wp(S3,O)
..
..
..
not(B1 || B2 || … || Bn-1) && wp(Sn,O)

Increase in size of VC generated is linear

Comparison

```
@requires marks = 75
@ensures  grade = B
@program {

  grade = F;

  if (marks > 50) grade = C;
  if (marks > 70) grade = B;
  if (marks > 90) grade = A;
}
.
```

```
@requires marks = 75
@ensures  grade = B
@program {
  if (marks > 90)
        grade = A;
  else  if (marks > 70)
            grade = B;
        else if (marks > 50)
                 grade = C;
             else grade = F;
}
.
```

Else-if is easier to verify: smaller VC and less complex (grade is assigned exactly once)