SLAM IV
Boolean Model Checking

Verification & Testing

SLAM thus far

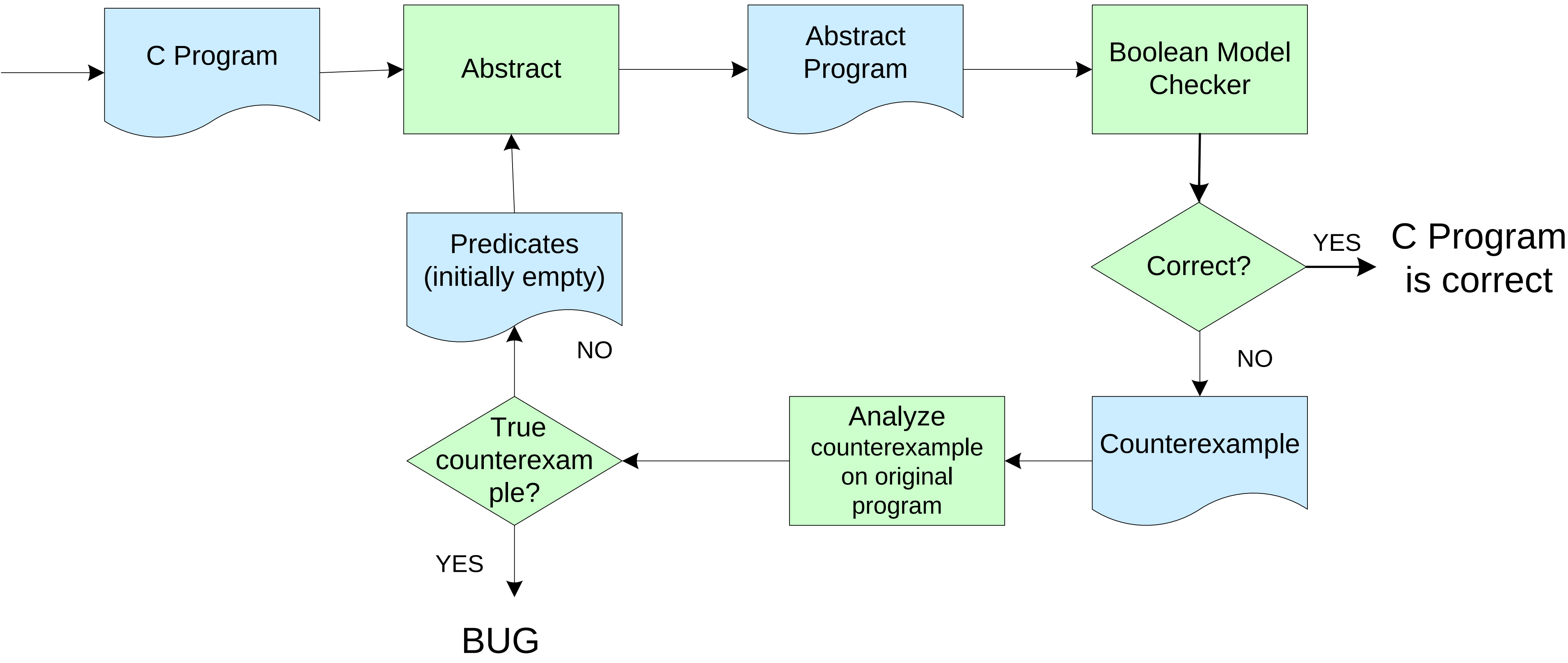
Automatic model checking of C programs

Abstraction/Refinement loop

- Predicate abstractions
- Initial abstraction: no predicates, only control flow
- When abstract program correct, so is concrete program
- When bug found in abstract program, check on concrete program
- If bug is real, Stop.
- If bug is not real, add predicates to prove impossibility of path, create new abstraction, and redo

This week: Model checking Boolean Programs

The Approach



Boolean Programs

All variables are Boolean. We have

- Global and local variables
- nondeterminism (*)
- Functions with parameters
- Function calls, recursion
- skip
- return
- if
- while
- assume
- assert

We do not have: integers, malloc, free

Assert & Assume

assert b

Check if b is true.

Is b true?
yes? continue
no? Found failure!

assume b

assume that b is true

Is b true?
yes? continue
no? disregard this execution

Model check this program!

Model Checking Boolean Programs

Question: can Booleanp rogram make nondeterministic decisions such that assertion is violated?

Example

```
decl g
main(){
  decl h;
  h = !g;
  A(g,h);
  A(g,h);
  assert(!g);
}

A(a1,a2){
  if(a1){
    A(a2,a1);
  }else{
    g = a2;
  }
}
```

Example

g h

00 01 10 11

```
decl g
main(){
  decl h;
  h = !g;
  A(g,h);
  A(g,h);
  assert(!g);
}
```

g a1 a2

000 001 010 011 100 101 110 111

```
A(a1,a2){
  if(a1){

    A(a2,a1);
  }else{

    g = a2;
  }
}
```

Some Definitions

A *valuation* gives a value to a set of variables.

The *visible variables* are the global variables plus the local variables that are in scope

For function calls,

- The *caller* is the calling function
- The *callee* is the called function

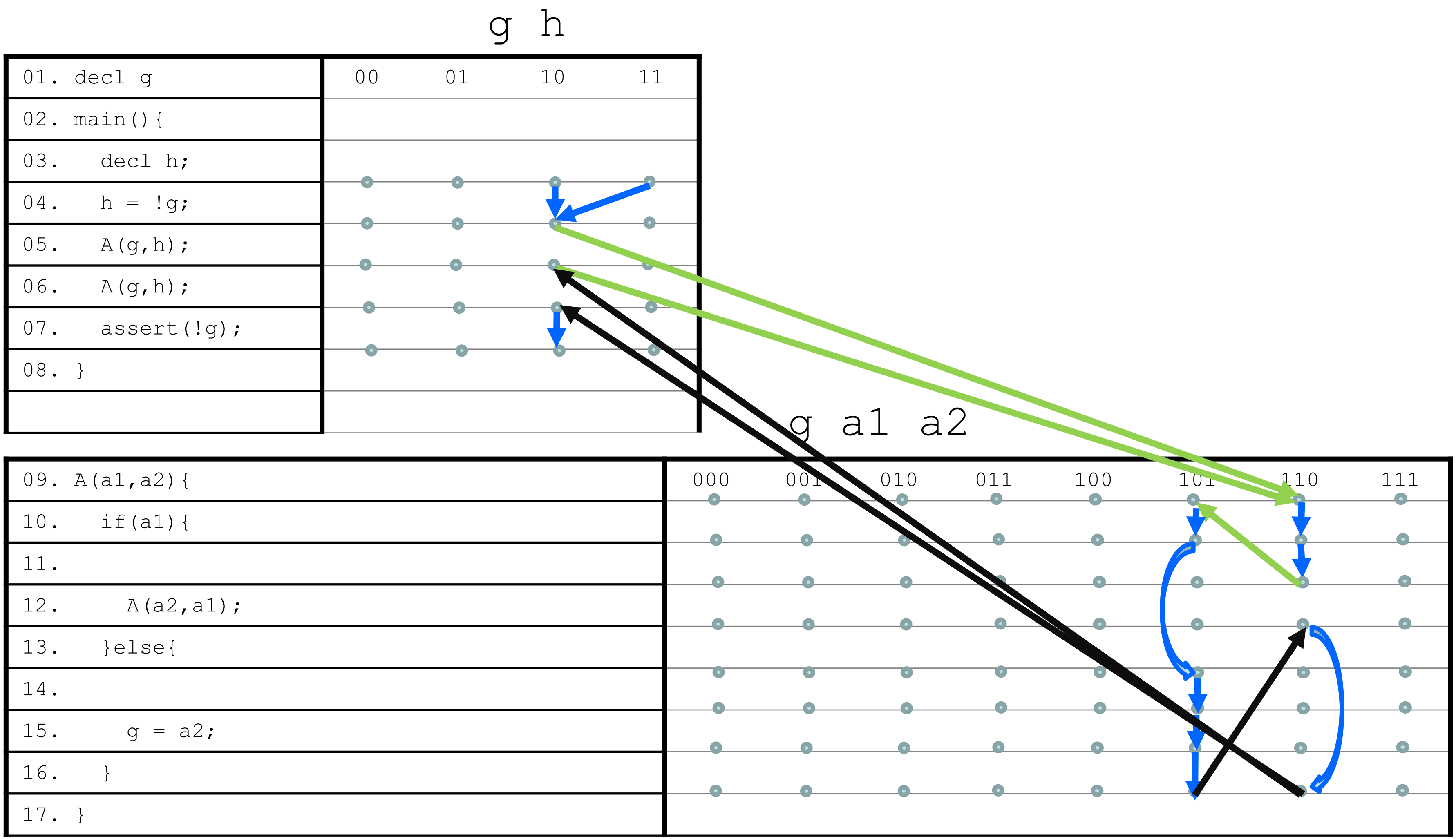
We add *points* to every line

- A point is labeled with a valuation of the visible variables (the valuation after execution of the line)
- A point is marked “done” or “not done”

We add arrows

- blue arrows for control flow
- green arrows for function calls
- black arrows for returns

Example


g h
01. decl g
02. main(){
03. decl h;
04. h = !g;
05. A(g,h);
06. A(g,h);
07. assert(!g);
08. }
00
01
10
11
g a1 a2
09. A(a1,a2){
10. if(a1){
11.
12. A(a2,a1);
13. }else{
14.
15. g = a2;
16. }
17. }
000
001
010
011
100
101
110
111

Example

```
01. decl g
02. main(){
03.    decl h;
04.    h = !g;
05.    A(g,h);
06.    A(g,h);
07.    assert(!g);
08. }

09. A(a1,a2){
10.    if(a1){
11.
12.       A(a2,a1);
13.    }else{
14.
15.       g = a2;
16.    }
17. }
```

Bug:

```
03. g=1, h=0
04. g=1, h=0
  09. g=1, a1=1, a2=0
  11. g=1, a1=1, a2=0
    09. g=1, a1=0, a2=1
    14. g=1, a1=0, a2=1
    15. g=1, a1=0, a2=1
  12. g=1, a1=1, a2=0
  16. g=1, a1=1, a2=0
05. g=1, h=0
09. g=1, a1=1, a2=0
  11. g=1, a1=1, a2=0
    09. g=1, a1=0, a2=1
    14. g=1, a1=0, a2=1
    15. g=1, a1=0, a2=1
  12. g=1, a1=1, a2=0
  16. g=1, a1=1, a2=0
06. g=1, h=0
07. assert(false)!
```

Example

```
decl g
main(){
  decl h;
  h = !g;
  A(g,h);
  A(g,h);
  assert(!g);
}

A(a1,a2){
  if(a1){

    A(a2,a1);
  }else{

    g = a2;
  }
}
```

Note:

Example is deterministic (no *)

Example has an infinite loop.

- This is not a bug
- The model checker should still finish

Model Checking

We perform forward analysis and build graph. Nodes: combination of line number and valuation of variables. Arrows: blue (normal execution) and green (function calls).

At beginning of main, add point for every valuation

For every point p not marked *done*:

- If next statement is
  - **assignment:** compute new valuations, add point q to next line, label with each valuation. (Nondeterminism can cause multiple valuations)
  - **if:** Add point q with same valuation to beginning of then or else branch. (or both if condition is *)
  - **while statement:** Like if
  - **end of function f:** For all p' with green arrow to the start of f and path of blue arrows from start of f to p (calls to f that end in p), compute new valuation of caller and add point q with this valuation.
  - **assert:** Condition false? Bug! Otherwise, create q with same valuation after assert.
- Mark p done. If not at end of function, add blue arrow from p to q
- If next statement is **function call:** compute valuation local to function, add point q to start of callee, add green arrow from p to q

All points marked done and no bug found? program is correct!

Function Calls

Function calls are call-by-value (like in C)

When calling a function,

- Value of globals in callee = value of globals in caller before call
- Value of formal parameters in callee = value of actual parameters in caller before call

When returning

- Value of globals in caller after call = value of globals in callee at end of function
- Value of locals in caller after call = value of locals in caller before call

Example, Notes

For a given function and valuation there may be

- No call with that valuation: ignored
- A call but no returns: infinite loop
- A call and one return: deterministic
- A call and multiple returns.
- The last case happens if there is nondetermism in the function. Every return is propagated to caller. Try replacing if(a1) by if(*) in example.

There may be multiple callers for every valuation

We avoid infinite loops by keeping track of valuations we have seen before.

Another Example: nondeterminism

01. decl g	
02. main(){	
03. A(g,g)	
04. assert(g);	
05. }	
06. A(a1,a2){	
07. if(*){	
08. g = a1;	
09. } else {	
10. g = !a1	
11. }	
13. }	

Another Example: nondeterminism

`01. decl g`	
`02. main(){`	
`03.   A(g,g)`	
`04.   assert(g);`	
`05. }`	
`06. A(a1,a2){`	
`07.   if(*){`	
`08.     g = a1;`	
`09.   } else {`	
`10.     g = !a1`	
`11.   }`	
`13. }`	

Note:

Nondeterminism causes two outgoing transitions for each point on line 7 and line 3.

For instance:

- In line 7 with (g,a1,a2)=(0,0,0), we can go to line 8 with (0,0,0) or line 10 with (0,0,0).
- In line 3 with g = 0 we can go to line 4 with g = 0 or line 4 with g = 1.

Concluding

Model checking a Boolean program

It's simple, just keep track of what you've done

Now practice

Program	Abstraction p: y < 44	Boolean MC p 1, p 0	Abstraction p: y < 44 q:	Boolean MC pq 00, pq 01, pq 10, pq 11
`y = 22`				
`x = 12`				
`z = x*x+1`				
`if (x <= 0) {`				
`  if (y>42) {`				
`    y = y - x`				
`  } else {`				
`    y = 42`				
`  }`				
`}`				
`assert (y<44)`				