SLAM IV
Boolean Model Checking

Verification & Testing

Anja Karl
anja.karl@iaik.tugraz.at

SLAM thus far

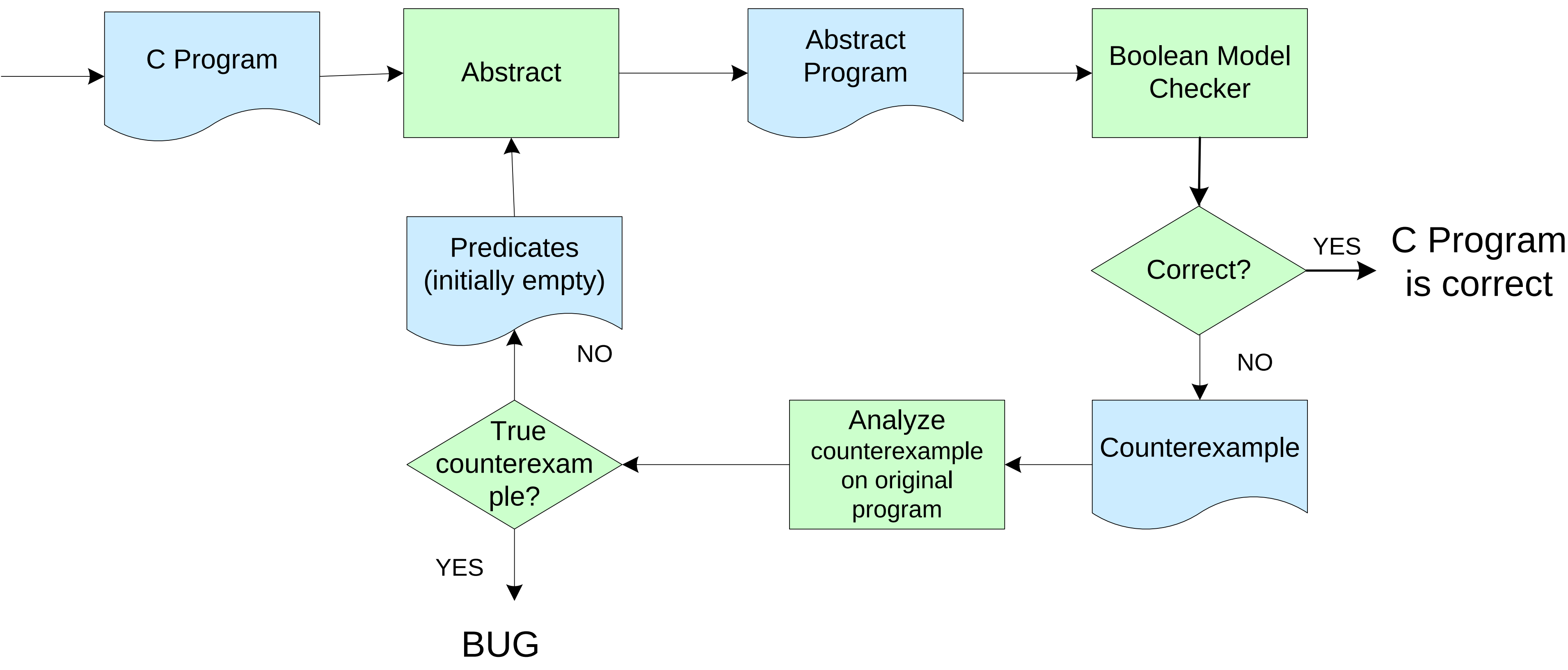
Automatic model checking of C programs

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!

Dealing with asserts

Consider this program

```
1. x = 4;
2. if(x == 4){
3.  x = x + 1;
4. }
5. assert(x==5);
```

and the predicate `b:{x==5}`

The abstract program is:

```
1. b = FALSE;
2. if(b? false : *){
3.  b = b? FALSE : *;
5. }
6. assert(b);
```

A counterexample:

1, 2, 3, 4, 5:

```
1. x = 4;
2. assume(x == 4)
3. x = x + 1;
4.
5. assume(x!=5);
```

From here you can compute new predicate.

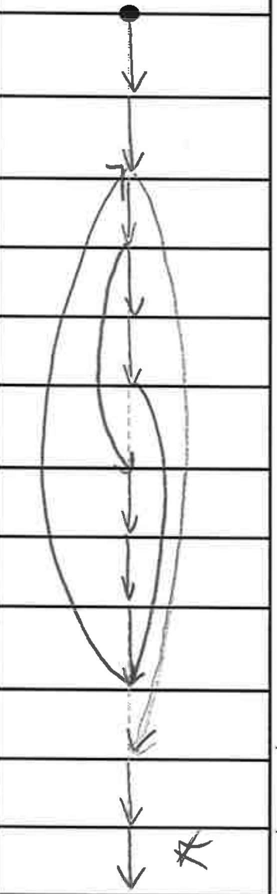
IMPRORTANT: The broken assertion becomes an assumption that the condition is false

Model Checking Boolean Programs

The question: can the program make nondeterministic decisions such that assertion is violated?

Program	Abstraction No predicate	Boolean MC	
1. a = 8			
2. b = -2			
3. while (a > 6){			
4. if (b == 0) {			
5. a = 6			
6. } else {			
7. a = a + 1			
8. b = b + 2			
9. }			
10. }			
11. b = b * (-1)			
12. assert(b == 0)			

Program	Abstraction No predicate	Boolean MC
1. a = 8	skip	
2. b = -2	skip	
3. while (a > 6){	while(*){	
4. if (b == 0) {	if (*) {	
5. a = 6	skip	
6. } else {	} else {	
7. a = a + 1	skip	
8. b = b + 2	skip	
9. }	}	
10. }	}	
11. b = b * (-1)	skip	
12. assert(b == 0)	assert(*)	*



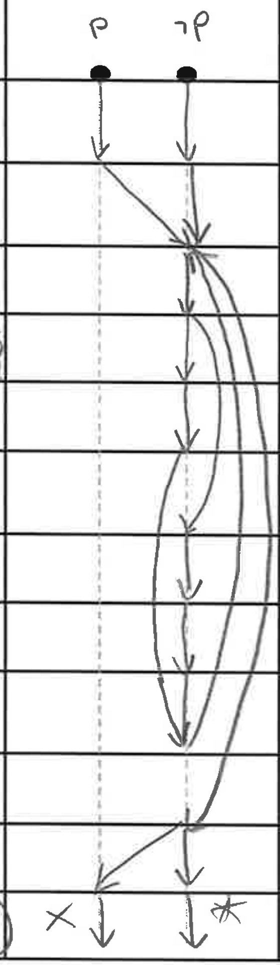
Counterexample:
1, 2, 3, 11, 12

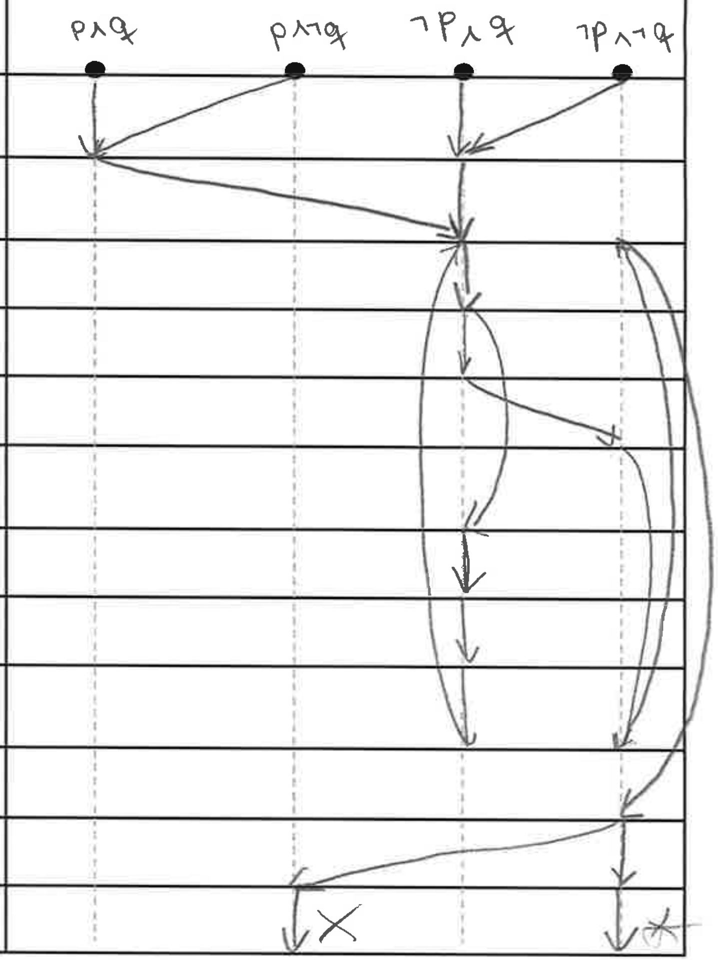
{8 ≤ 6} → False
1: a=8
{2 ≠ 0 ∧ a ≤ 6}
2: b=-2
{-b ≠ 0 ∧ a ≤ 6}
3: assume(a ≤ 6)
{-b ≠ 0}
11: b = b * (-1)
{b ≠ 0}
12: assume(b ≠ 0)

Learned Predicate:
a ≤ 6

Program	Abstraction p: b<-2	Boolean MC	Abstraction p: b<-2 q:	Boolean MC
1. a = 8				
2. b = -2				
3. while (a > 6){				
4. if (b == 0) {				
5. a = 6				
6. } else {				
7. a = a + 1				
8. b = b + 2				
9. }				
10. }				
11. b = b * (-1)				
12. assert(b == 0)				

Program	Abstraction p: b<-2	Boolean MC p ¬p	Abstraction p: b<-2 q: a > 6	Boolean MC p∧q p∧¬q ¬p∧q ¬p∧¬q
1. a = 8	skip		q = True	
2. b = -2	p = False		p = False	
3. while (a > 6){	while(*){		while (q) {	
4. if (b == 0) {	if(p ? F : *){		if (p ? F : *) {	
5. a = 6	skip		q = False	
6. } else {	} else {		} else {	
7. a = a + 1	skip		q = q ? T : *	
8. b = b + 2	p = p ? * : F		p = p ? * : F	
9. }	}		}	
10. }	}		}	
11. b = b * (-1)	p = p ? F : *		p = p ? F : *	
12. assert(b == 0)	assert(p ? F : *)	× *	assert(p ? F : *);	× *





Counterexample 1:

1, 2, 3, 11, 12

$\{8 \leq 6\}$ → False!

1. $a = 8$

$\{2 \neq 0 \wedge \underline{a \leq 6}\}$

2. $b = -2$

$\{b \neq 0 \wedge \underline{a \leq 6}\}$

3. assume$(\underline{a \leq 6})$

$\{b \neq 0\}$

11. $b = b + (-1)$

$\{b \neq 0\}$

12. assume$(b \neq 0)$

Learned Predicate: $a \leq 6$

(you can also use $a > 6$)

Counterexample 2:

1, 2, 3, 4, 5, 3, 11, 12

$\{\underline{-b \neq 0} \wedge \underline{b = 0}\}$ → False!

4: assume$(\underline{b = 0})$

$\{\underline{-b \neq 0} \wedge a \leq 6\}$

5: $a = 6$

$\{\underline{-b \neq 0} \wedge a \leq 6\}$

3: assume$(a \leq 6)$

$\{\underline{-b \neq 0}\}$

11: $b = b + (-1)$

$\{\underline{b \neq 0}\}$

12: assume$(b \neq 0)$

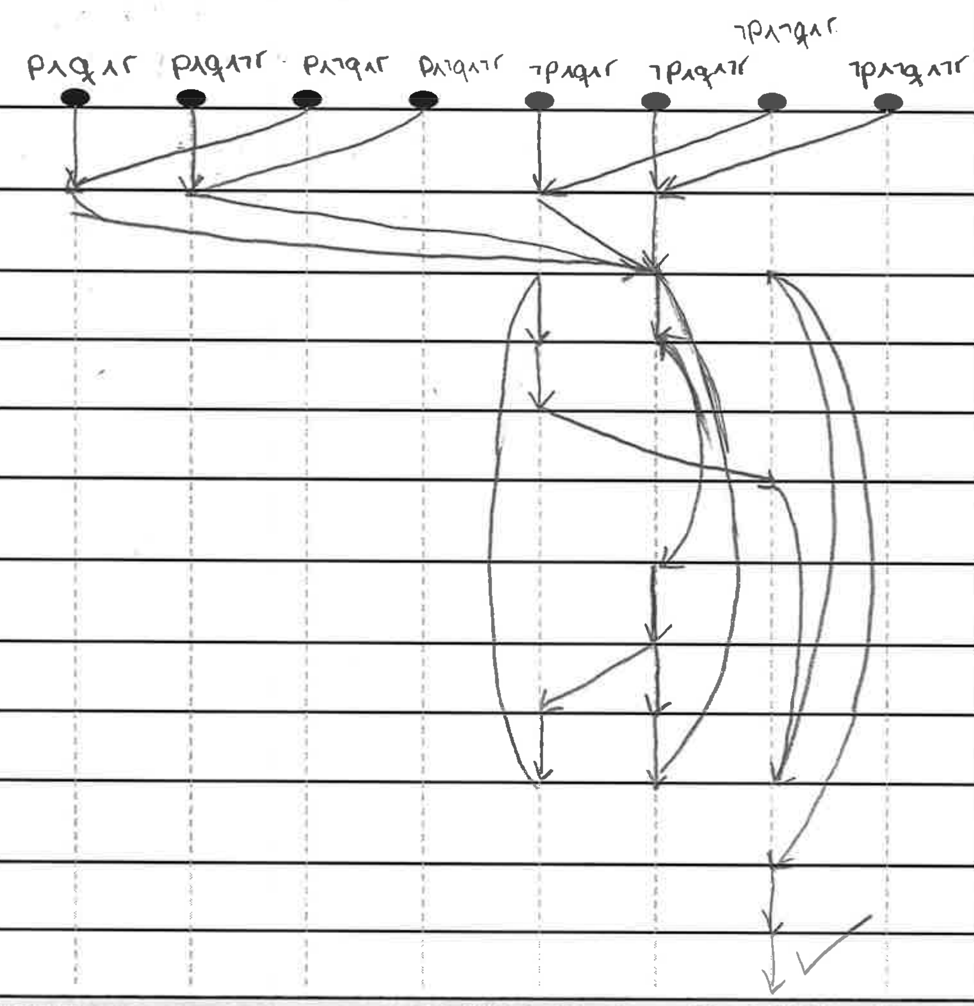
learned predicate: $b = 0$

(you can also use $b \neq 0$ or $-b \neq 0$)

Program	**Abstraction** p: b<-2 q: r:	**Boolean MC**
1. a = 8		
2. b = -2		
3. while (a > 6){		
4. if (b == 0) {		
5. a = 6		
6. } else {		
7. a = a + 1		
8. b = b + 2		
9. }		
10. }		
11. b = b * (-1)		
12. assert(b == 0)		

Program	Abstraction p: b<-2 q: a > 6 r: b = 0
1. a = 8	q = True
2. b = -2	p = False; r = False;
3. while (a > 6){	while (q) {
4. if (b == 0) {	if (r) {
5. a = 6	q = False
6. } else {	} else {
7. a = a + 1	q = q ? T : *
8. b = b + 2	p = p ? * : F r = p ? F : (r ? F : *)
9. }	}
10. }	}
11. b = b * (-1)	r = r ? T : F p = p ? F : (r ? F : *)
12. assert(b == 0)	assert (r)

Boolean MC



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

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

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)

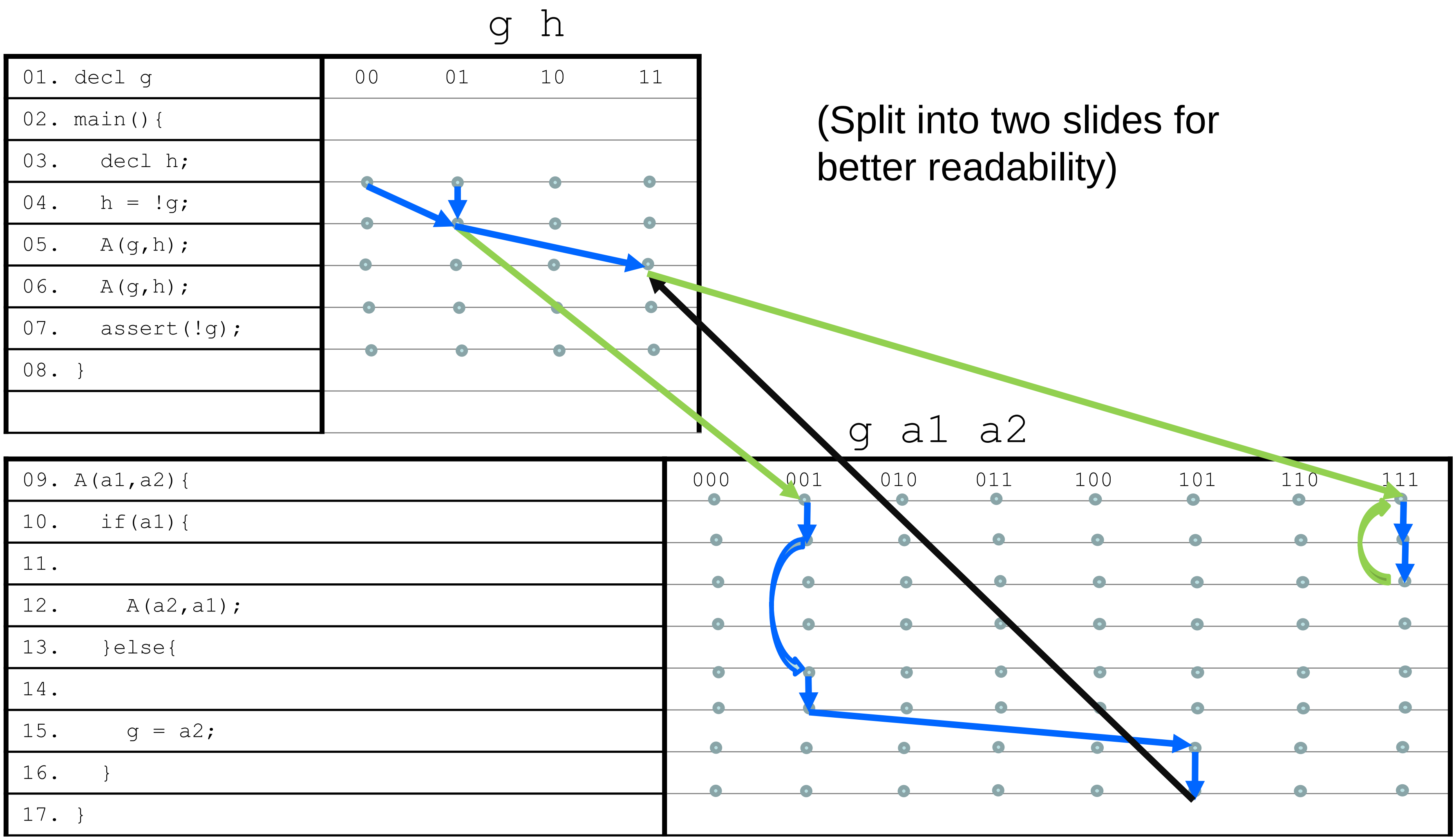
When calling a function,

- 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

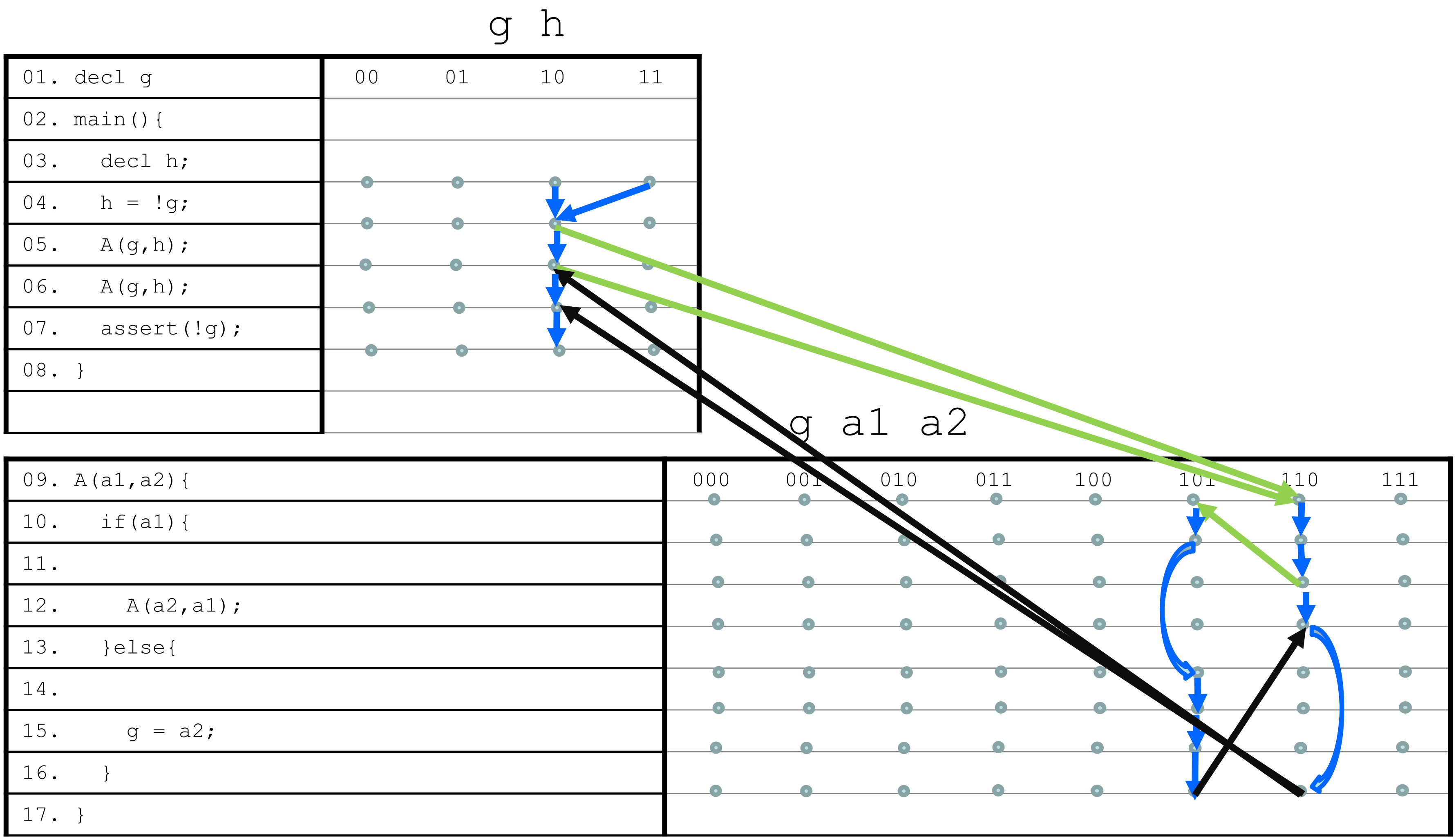
- Value of globals in caller after call = value of globals in callee at end of function
- Value of locals in caller after call = value of locals in caller before call

Example



(Split into two slides for better readability)

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, 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. }`	

Note:

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

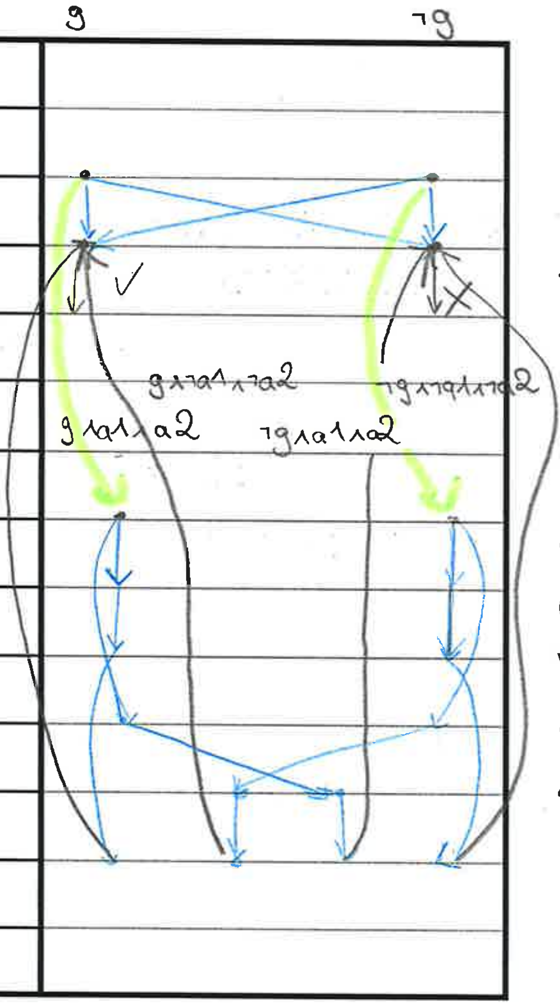
For instance:

- 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.

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) $\underline{p}$ (0)	**Abstraction** p: y < 44 q:	**Boolean MC** $\underline{p}\underline{q}$ (00) $\underline{p}q$ (01) $p\underline{q}$ (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)				

	Program	Abstraction p: y < 44	Boolean MC p 1 p 0	Abstraction p: y < 44 q: x ≤ 0	Boolean MC pq 00 pq 01 pq 10 pq 11
1	y = 22	p = True		p = True	
2	x = 12	skip		q = False	
3	z = x*x+1	skip		skip	
4	if (x <= 0) {	if (*) {		if (q) {	
5	if (y>42) {	if (p ? * : T) {		if (p ? * : T) {	
6	y = y - x	p = *		p = (p ∧ ¬q) ? T : ((¬p ∧ q) ? F : *)	
7	} else {	} else {		} else {	
8	y = 42	p = True		p = True	
9	}	}		}	
10	}	}		}	
11	assert (y<44)	assert (p)	✓ ✗	assert (p)	✓

Counterexample 1:

1, 2, 3, 4, 5, 6, 11

$\{y - 12 \geq 44 \wedge y > 42 \wedge \underline{12 \leq 0}\}$

2 $x = 12$

$\{y - x \geq 44 \wedge y > 42 \wedge \underline{x \leq 0}\}$

3 $z = x * x + 1$

$\{y - x \geq 44 \wedge y > 42 \wedge \underline{x \leq 0}\}$

4 assume$\underline{(x \leq 0)}$

$\{y - x \geq 44 \wedge y > 42\}$

5 assume$(y > 42)$

$\{y - x \geq 44\}$

6 $y = y - x$

$\{y \geq 44\}$

11 assume$(y \geq 44)$

learned predicate:

$x \leq 0$

(could also learn $x > 0$)