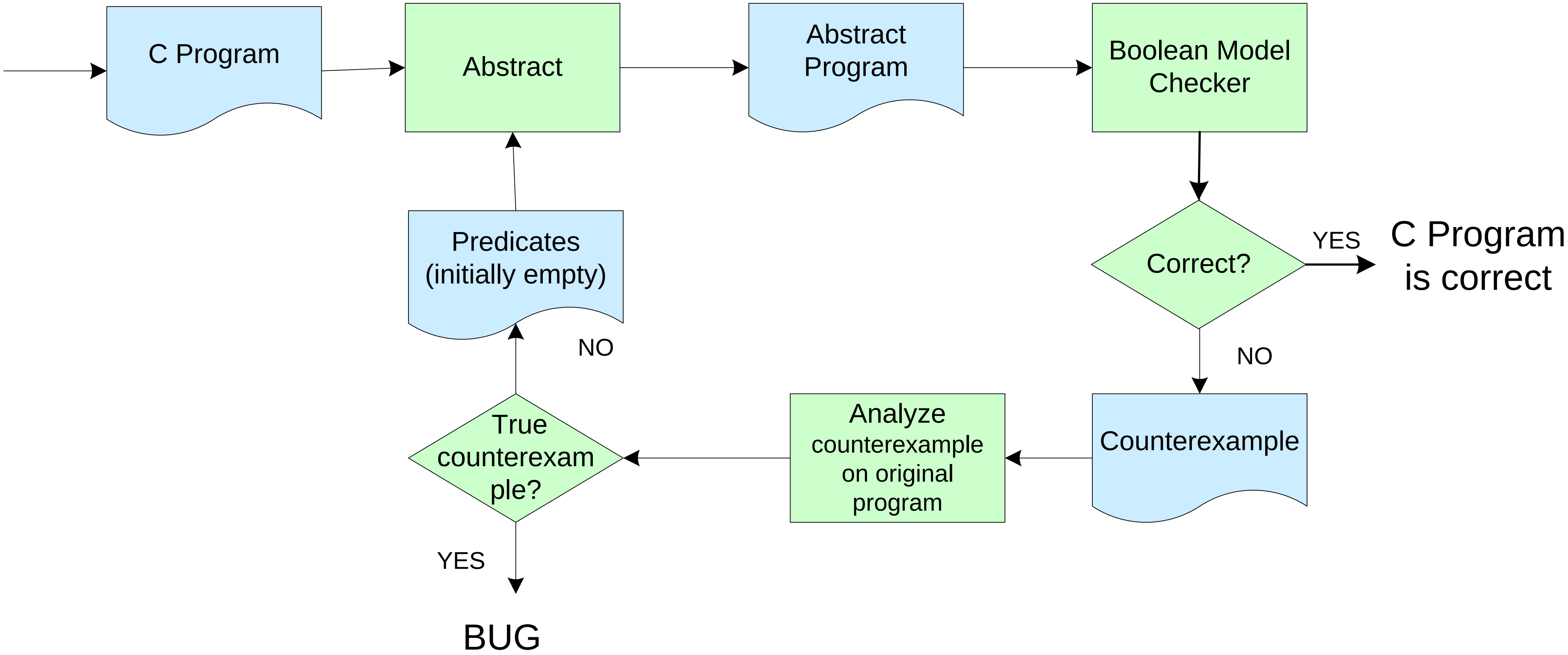
SLAM III: Discovering Predicates

V&T

Roderick Bloem

The Approach



Today's Problem

- Figuring out that a counterexample is infeasible
- Finding new predicates

First, recapitulate example from last week

Program

```
void example(){
  do{
    lock();
    nPacketsOld = nPackets;
    req = devExt->WLHV;
    if(req && req-> status){
      devExt->WLHV = req->next;
      release();
      irp = req->irp;
      if(req->status > 0){
        irp->IoS.status = SUCCESS;
        irp->IoS.Info = req->Stat;
      } else {
        irp->IoS.status = FAIL;
        irp->IoS.Info = req->Stat;
      }
      smartDevFreeBlock(req);
      IoCompleteRequest(irp);
      nPackets++;
    }// if req
  } while(nPackets != nPacketsOld);
  release();
}
```

First Boolean Program

```
void example(){
  do{
    lock();
    skip;
    skip;
    if(*){
      skip;
      release();
      skip;
      if(*){
        skip;
        skip;
      } else {
        skip;
        skip;
      }
      skip;
      skip;
      skip;
    }// if
  } while(*);
  release();
}
```

Boolean Counterexample

```
void example(){
  {
    lock();
    skip;
    skip;
    {
      skip;
      release();
      skip;
      {
        skip;
        skip;
      }



    skip;
    skip;
    skip;
    }
  }
  release();
}
```

```
void example(){
  do{
    lock();
    nPacketsOld = nPackets;
    req = devExt->WLHV;
    if(req && req-> status){
      devExt->WLHV = req->next;
      release();
      irp = req->irp;
      if(req->status > 0){
        irp->IoS.status = SUCCESS;
        irp->IoS.Info = req->Stat;
      } else {
        irp->IoS.status = FAIL;
        irp->IoS.Info = req->Stat;
      }
      smartDevFreeBlock(req);
      IoCompleteRequest(irp);
      nPackets++;
    }// if req
  } while(nPackets != nPacketsOld);
  release();
}
```

Counterexample in C

```
void example(){
  {
    lock();
    nPacketsOld = nPackets;
    req = devExt->WLHV;
    {assume(req && req->status);
      devExt->WLHV = req->next;
      release();
      irp = req->irp;
     {assume(req->status > 0)
        irp->IoS.status = SUCCESS;
        irp->IoS.Info = req->Stat;
      }
      smartDevFreeBlock(req);
      IoCompleteRequest(irp);
      nPackets++;
    }
  } assume(nPackets == nPacketsOld);
  release();
}
```

The assume statements show the knowledge that we have because we know whether the if-condition was true

How do we show that this counterexample is infeasible?

Reminder: Hoare Axiom for Assignment

An example:

`{b+c = 5}`

`a = b + c`

`{a = 5}`

In general:

`{Q[x → e]}`

`x = e;`

`{Q}`

`[x → e]` means that `x` is replaced by `e`

This is the *weakest precondition*:

`Q[x → e]` holds before `x = e`

if and only if

`Q` holds afterwards

Discovering Infeasible Paths

```
lock();
% FALSE && devExt->WLHV->status>0
% && devExt->WLHV && devExt->WLHV->status
nPacketsOld = nPackets;
% nPackets+1==nPacketsOld && devExt->WLHV-status>0
% && devExt->WLHV && devExt->WLHV->status
req = devExt->WLHV;
% nPackets+1==nPacketsOld && req->status>0 &&
% req && req->status
assume(req && req->status)
% nPackets+1==nPacketsOld && req->status>0
devExt->WLHV = req->next;
% nPackets+1==nPacketsOld && req->status>0
release();
% nPackets+1==nPacketsOld && req->status>0
irp = req->irp;
% nPackets+1==nPacketsOld && req->status>0
assume(req->status > 0)
% nPackets+1==nPacketsOld
irp->IoS.status = SUCCESS;
% nPackets+1==nPacketsOld
irp->IoS.Info = req->Stat;
% nPackets+1==nPacketsOld
smartDevFreeBlock(req);
% nPackets+1==nPacketsOld
IoCompleteRequest(irp);
% nPackets+1==nPacketsOld
nPackets = nPackets + 1;
% nPackets==nPacketsOld
assume(nPackets == nPacketsOld)
%TRUE
release();
% TRUE
```

Computed from bottom to top
Note the use of colors!
Assumption: function calls do not affect predicates

Adding Predicates

if we add

- `{nPackets==nPacketsOld}` and
- `{nPackets+1==nPacketsOld}`

then at any line we know that the red predicate is false. Thus, this path is no longer possible in the abstract program.

Note:

- Infeasibility is computed bottom-up,
- Functions (`smartDevFreeBlock`, etc.) must inlined or analyzed
- A path consists of
  - assignments (change the predicates)
  - assumes (derive from if, while, add a predicate)

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. (see next slide)

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

```
x = 4;
{x!=4 && x==4} = FALSE
assume(x == 4)
{x != 4}
x = x + 1;
{x != 5}
assume(x!=5);
```

New predicate: `x != 4.`

Computing Infeasible Paths

```
In: a path P with l lines.
lp := l + 1                        // line pointer
condition[lp] := true;             // condition for line lp, a condition is a
                                      set of predicates
while(lp ≠ 0 && condition[lp] ≠ false){
  lp--
  if(statement(lp) = “assume(p)”){
    condition[lp] := “condition[lp+1] && p”, use new color for p
  else if(statement(lp) = “x := e”){
    condition[lp] := condition[lp+1][x → e]
  }
  simplify condition
}

if(condition[lp]= … &&false&&…){
  report INFEASIBLE!
  add all predicates from all condition[i] with same color as false
} else {
  report FEASIBLE: BUG FOUND
}
```

Excluding paths

Guarantee: New predicates exclude the infeasible path
Hope: They also exclude other paths

How do you simplify?

- SLAM pretends that C's `int`s are integers and `float`s are real numbers and uses a theorem prover. This is
  - Not precise (overflows, limited precision for floats)
  - Theorems over integers are undecidable
  - Theorem provers are relatively fast.
- Better: ints are bit vectors
  - Modern theorem provers are fast for bit vectors
  - Decidable, precise

Undecidability

- Equality of integer formulas including multiplication is undecidable!
  - There is no algorithm to decide formulas like
  - Are there x, y, z such that x*x*x + y*y*y = z*z*z?
  - Are there x, y, z, n>2 such that $x^n + y^n = z^n$? (Fermat's theorem)
- Without multiplication everything is OK.
- Use a theorem prover to prove equivalence, theorem prover may never return an answer, or say "I don't know"

Hoare's Axiom of Assignment & Pointers

What is the weakest precondition of `*p > n` for statement `x = 3`?

Wrong answer:

`*p > n`

Consider

```
x = 1;
n = 2;
p = &x;
x = 3;
```

We get

```
{false}
x=1
{x > 2}
n = 2
{*&x > n} = {x > n}
p = &x;
{*p > n}
x = 3
{*p > n}
```

This is incorrect. The update of x is not considered

Pointers

What is the weakest precondition of `*p > n` for statement `x = 3`?

Wrong answer:

`*p > n`

Correct answer:

$(p \neq \&x \wedge *p>n) \vee (p = \&x \wedge 3>n)$

Easily generalized to more variables.

```
{true}
{3 > 2}
n = 2
{(3 > n)}
{{(&x != &x &&*p > n) ||
(&x=&x && 3 > n)}
p = &x;
{(p != &x &&*p > n) || (p=&x
&& 3 > n)}
x = 3
{*p > n}
```

Conclusion: the standard axiom of assignment is not correct when there is aliasing. Aliases occur as pointers (thus also when references are used)

Conclusion

- You can find infeasible paths by repeated application of Hoare's axiom for assignment
- You color each predicate that you find, if one predicate is transformed to false, add all predicates of that color.