SLAM
Motivation & Example

Verification & Testing

Roderick Bloem

This week: SLAM

Automatically verify properties of drivers

Part of MS Windows Driver Kit (called the *Static Driver Verifier*)

Key: automatic abstraction

Based on: Ball & Rajamani, Automatically Validating Temporal Safety Properties of Interfaces, SPIN Workshop on Software Model Checking, 2001

Why Drivers?

Drivers are critical
- Often run in kernel space, can wreak havoc

Drivers are not under MS control
- Developed by hardware companies
- Cannot verify correctness, cannot impose coding standards, cannot educate designers, hardware companies do not have the same understanding of windows

Drivers are simple
- Important properties are things like locking protocol
- The correctness of such properties usually does not depend on details of driver implementation or hardware
- Drivers are relatively small

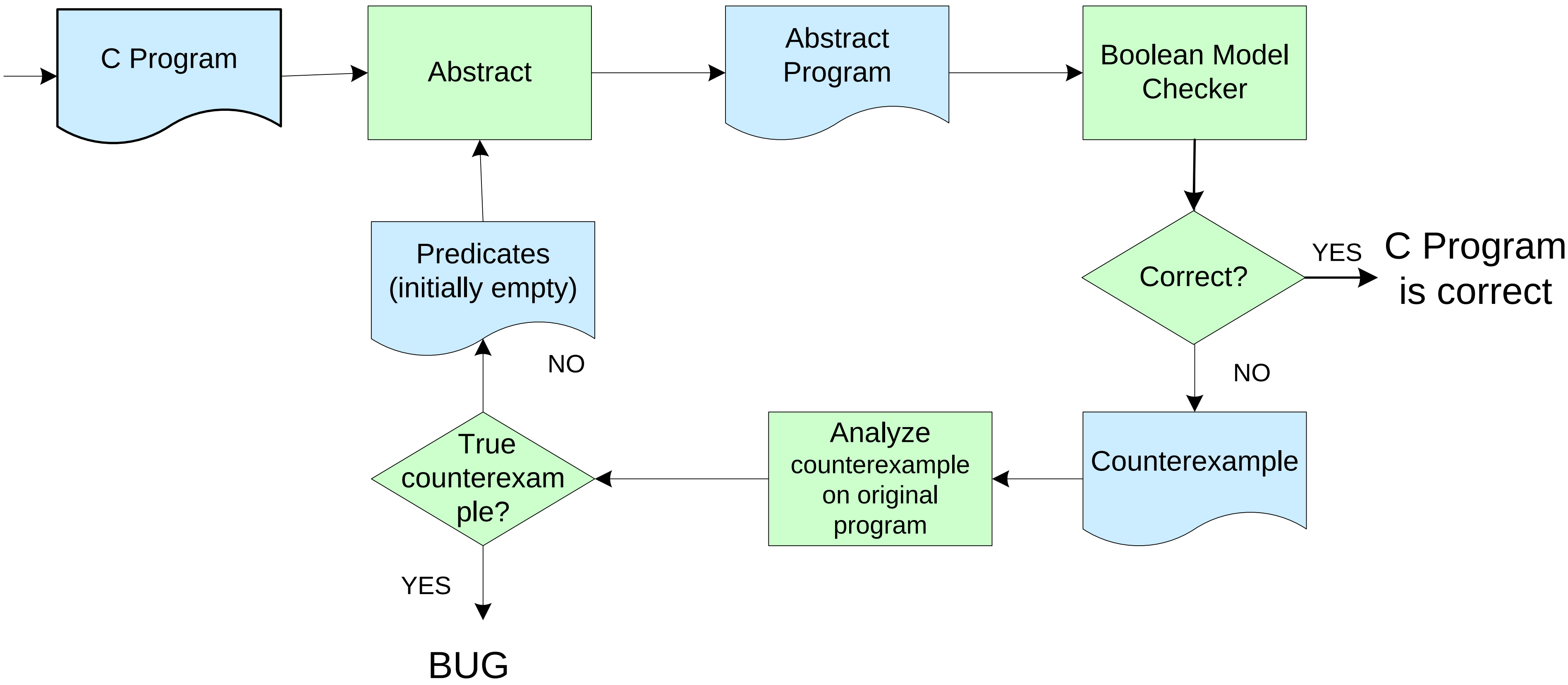
Choose

SLAM papers use choose function:
choose(f, g) is the same as
f ? true : (g? false: *);

i.e.,

- If f is true then true,
- if g is true then false,
- if neither are true then nondeterministic
- Both are true should be impossible

The Approach



The Approach

- Construct abstraction of program.
  - Remember: Abstraction adds behavior
- On abstraction, we may find counterexample that is impossible on real program
- Make abstraction more precise until
  - The abstraction contains no counterexamples
  - We find a real bug
- We use predicate abstraction
- The result of the abstraction is a Boolean Program

The Approach

1. The first set of predicates is empty
2. The abstraction engine uses the predicates to construct a Boolean Program
   – First approximation has only data flow, no variables
3. Run model checker for Boolean programs
   – No counterexample? C program is correct! Stop.
4. Analyze counterexample
   – If it is real, we have found a real bug. Stop
   – If not, add predicates to make abstraction more precise. Start from 2.

There are several reasons that this may not work (undecidability!).

Boolean Programs

- Functions with parameters, recursion
- Global and local Variables

- No mallocs and frees
- Only Boolean (one-bit) variables; no integers
- Nondeterminism: *
- assert , assume

Theory: Boolean programs are pushdown automata

- Checking Boolean programs is not hard (algorithm next week)

Some Syntax

- `a ? b : c` means if a then b else c. Example: `b = c ? 2 : 3`
- `assert(e)` means dump core unless `e` is true
- `assume(e)` means executions with `e` false are irrelevant.
- `*` is a function that may evaluate to 0 or 1 nondeterministically
  - `if(*) b = 1; else b=2;` evaluates to `b=1` or `b=2`, but never to `b=3`.
  - `*` is a function, not a value (If you set `b = *,` then afterwards `b` equals `0` or `1`, but b cannot equal `*`)

Example: Specification

```
int isLocked = 0;

void lock(){
  assert(!isLocked);
  isLocked = true;
}

void release(){
  assert(isLocked);
  isLocked = false;
}
```

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();
}
```

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

Which Predicate can Prove Counterexample Infeasible?

{nPackets == nPacketsOld}

Second Boolean Program

```
void example(){
  do{
    lock();
    b = true;
    skip;
    if(*){
      skip;
      release();
      skip;
      if(*){
        skip;
        skip;
      }  else {
        skip;
        skip;
      }
      skip;
      skip;
      b = b ? false: *;
    }// if
  } while(!b);
  release();
}
```

Predicate b:

`{nPackets==nPacketsOld}`

Second Boolean Program

Is correct – we are done.