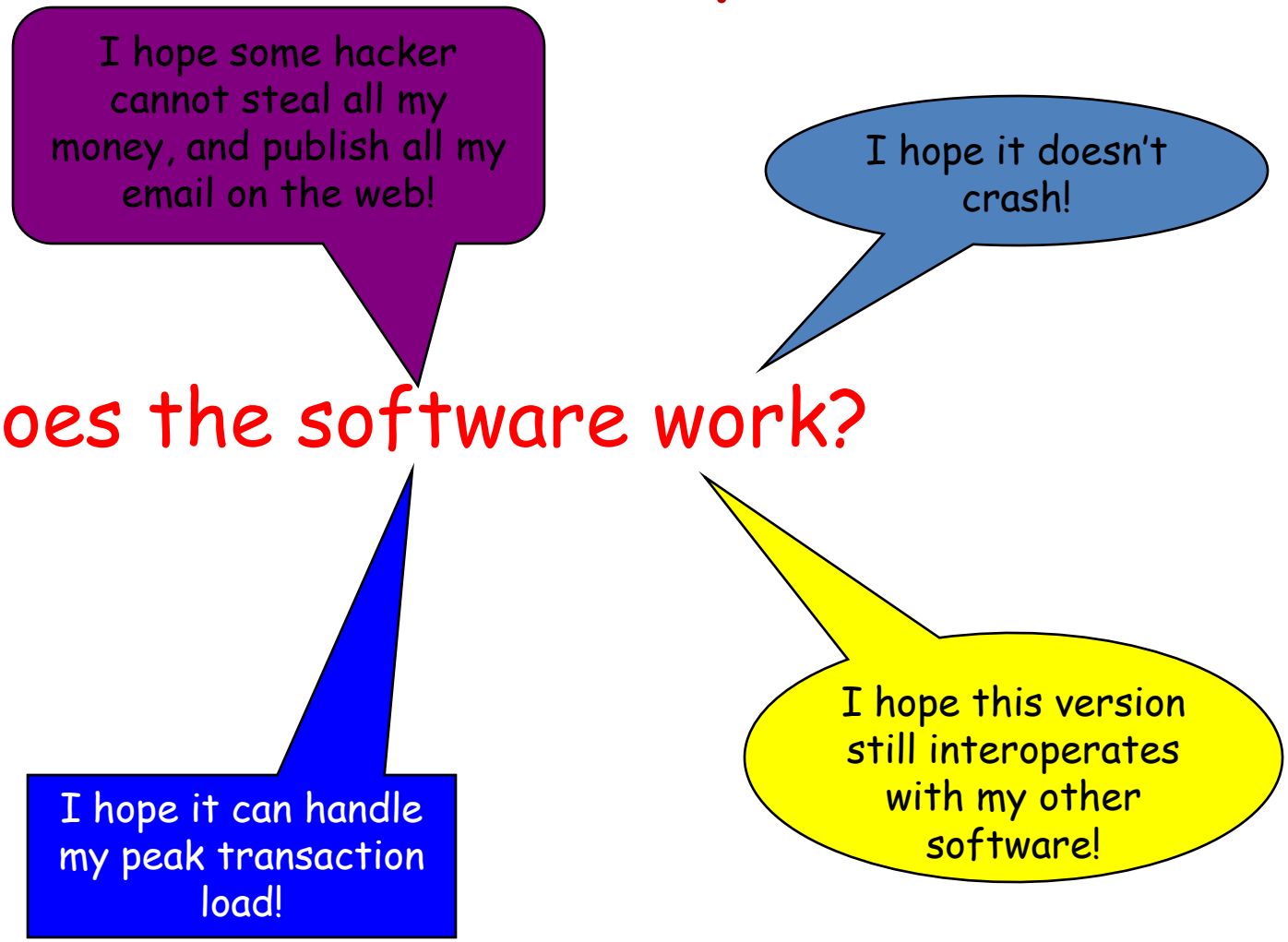
Software Model Checking

Aditya V. Nori
Microsoft Research India

Thanks to Tom Ball & Sriram Rajamani for material from their lectures

PROBLEM

Software validation problem



How do we do software validation?

Testing:

- The "old-fashioned" way
- Run it and see if it works
- Fix it if it doesn't work
- Ship it if it doesn't crash!

What's wrong with testing?

program correctness. Today a usual technique is to make a program and then to test it. But: program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence. The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness. But one should not first make

What's wrong with testing?

ACM Turing Lecture 1972 EWD340 - 0

The Humble Programmer.

by

Edsger W.Dijkstra

As a result of a long sequence of coincidences I entered the programming profession officially on the first spring morning of 1952 and as far as I have been able to trace, I was the first Dutchman to do so in my country. In retrospect the most amazing thing was the slowness with which, at least in my part of the world, the programming profession emerged, a slowness which is now hard to believe. But I am grateful for two vivid recollections from that period that establish that slowness beyond any doubt.

Program Verification

The algorithmic discovery of properties of a program by inspection of the source text

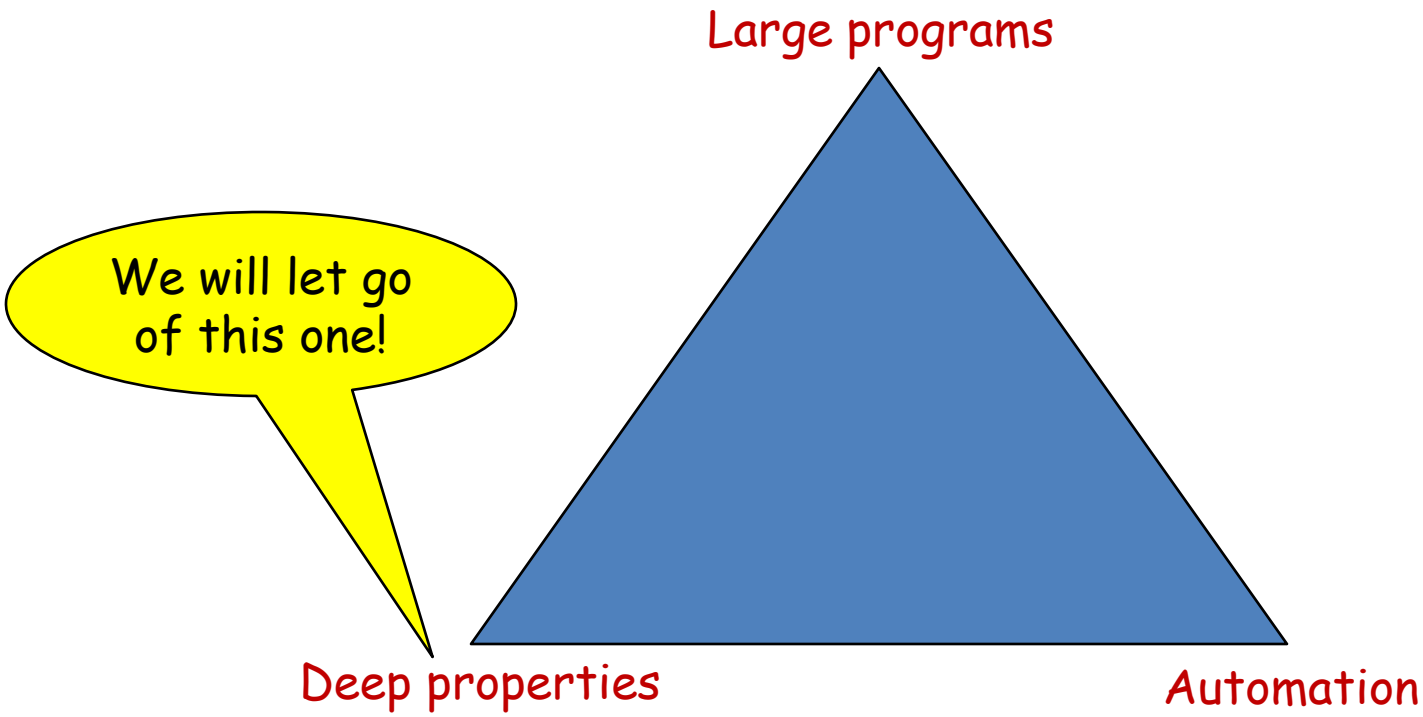
– *Manna and Pnueli, "Algorithmic Verification"*

Also known as: static analysis, static program analysis, formal methods, ….

Difficulties in program verification

- What will you prove?
  - Specification of a complex software is as complex as the software itself
- "Deep" specifications of software are hard to prove
  - State-of-art in tools and automation not good enough

Elusive triangle



Example properties

- Type safety
- Memory safety (absence of buffer overruns)
- Protocol conformance for APIs
- Race freedom

New generation of software tools

- SLAM/SDV (Windows Device Drivers)
- SAL+PREfast (Buffer overflow checking for C/C++)
- Spec# & Boogie (.NET)
- ASTREE (C, avionics software)
- FindBugs (Java, bug finder)
- Saturn (C, null deref bug finder)

and many more! …

Other routes to reliability

- Test
- Don't program in C ☺
- Debug
- Code inspection
- Modern languages (Java, C#, ML, …)
- Runtime checking

Outline

- SLAM: Software model checking via abstraction refinement
  - c2bp
  - bebop
  - newton
- Synergy: Property checking by combining static analysis and testing

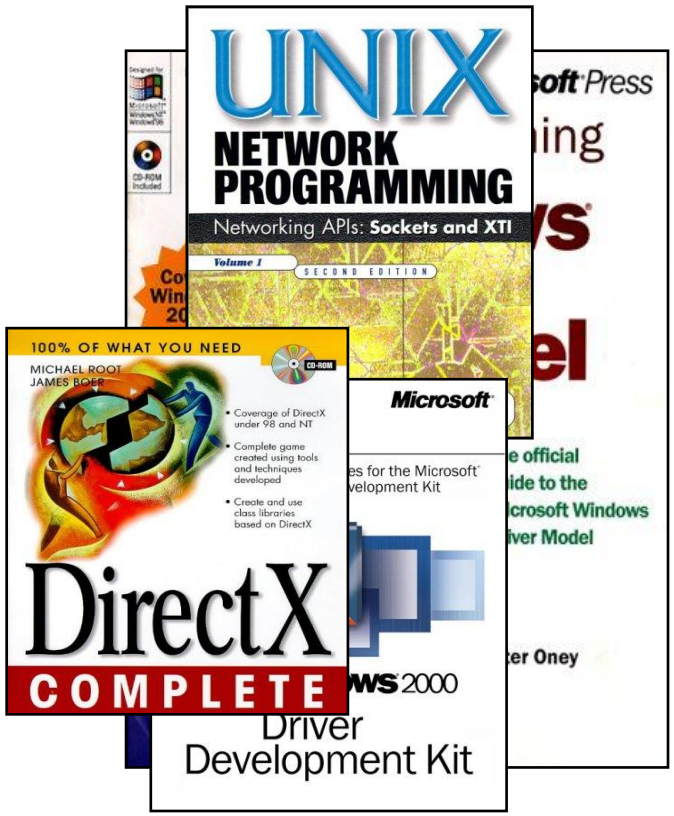
Software Validation

- Large scale reliable software is hard to build and test.
- Different groups of programmers write different components.
- Integration testing is a nightmare.

Property Checking

- Programmer provides redundant partial specifications
- Code is automatically checked for consistency
- Different from proving whole program correctness
  - Specifications are not complete

Interface Usage Rules



- Rules in documentation
  - Incomplete, unenforced, wordy
  - Order of operations & data access
  - Resource management
- Disobeying rules causes bad behavior
  - System crash or deadlock
  - Unexpected exceptions
  - Failed runtime checks

Does a given usage rule hold?

- Checking this is computationally impossible!
- Equivalent to solving Turing's halting problem (undecidable)
- Even restricted computable versions of the problem (finite state programs) are prohibitively expensive

Why bother?

Just because a problem is undecidable, it doesn't go away!

Automatic property checking = Study of tradeoffs

- Soundness vs completeness
  - Missing errors vs reporting false alarms
- Annotation burden on the programmer
- Complexity of the analysis
  - Local vs Global
  - Precision vs Efficiency
  - Space vs Time

Broad classification

- Underapproximations
  - Testing
    - After passing testing, a program may still violate a given property
- Overapproximations
  - Type checking
    - Even if a program satisfies a property, the type checker for the property could still reject it

Current trend

- Confluence of techniques from different fields:
  - Model checking
  - Automatic theorem proving
  - Program analysis
- Significant emphasis on practicality
- Several new projects in academia and industry

Software Model Checking via Abstraction Refinement

- Model checking = exhaustive exploration of state space
- Challenge: realistic software has a huge state space?
- Approach: Abstraction-refinement
  - Construct an abstraction
    - a "simpler model" of the software that only contains the variables and relationships that are important to the property being checked
  - Model check the abstraction
    - easier because state space of the abstraction is smaller
  - Refine the abstraction
    - to reduce false errors

SLAM – Software Model Checking

SLAM models

- boolean programs: a new model for software

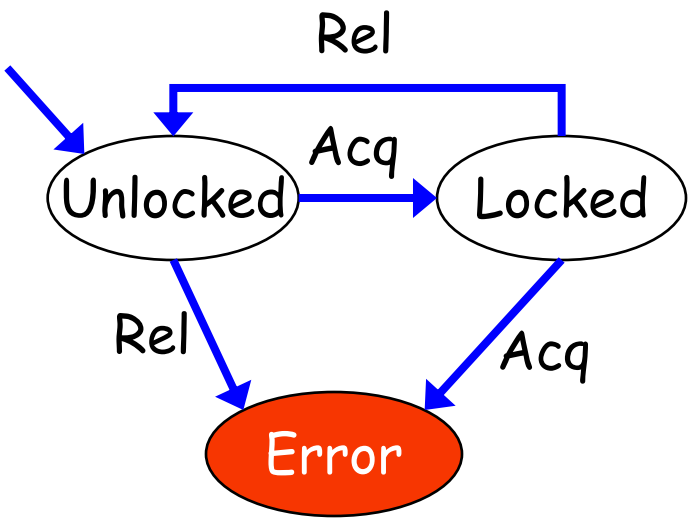
SLAM components

- model creation (c2bp)
- model checking (bebop)
- model refinement (newton)

SLIC

- Finite state language for stating rules
  - monitors behavior of C code
  - temporal safety properties (security automata)
  - familiar C syntax
- Suitable for expressing control-dominated properties
  - e.g. proper sequence of events
  - can encode data values inside state

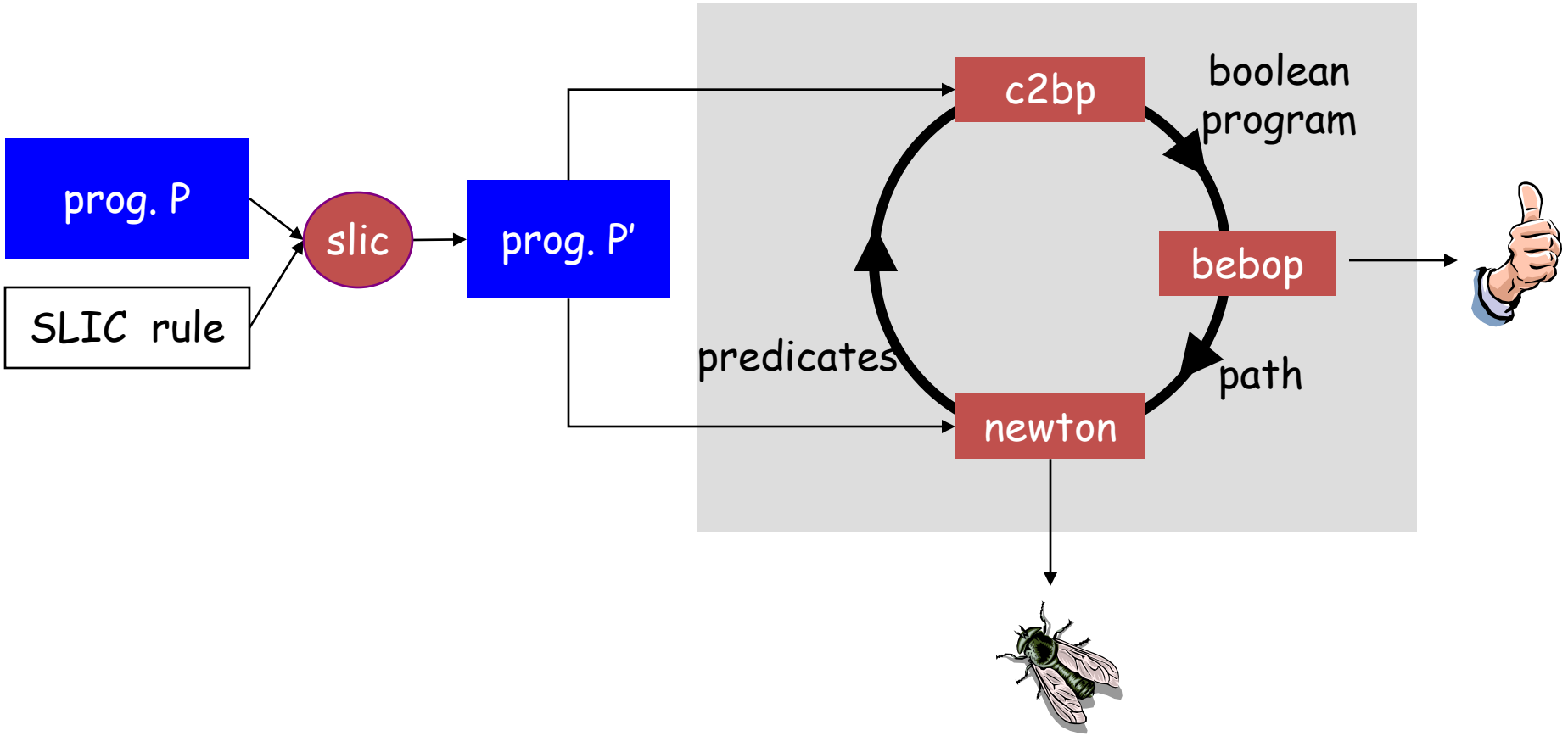
State Machine for Locking



Locking Rule in SLIC

```
state {
  enum {Locked,Unlocked}
        s = Unlocked;
}

KeAcquireSpinLock.entry {
  if (s==Locked) abort;
  else s = Locked;
}

KeReleaseSpinLock.entry {
  if (s==Unlocked) abort;
  else s = Unlocked;
}
```

The SLAM Process



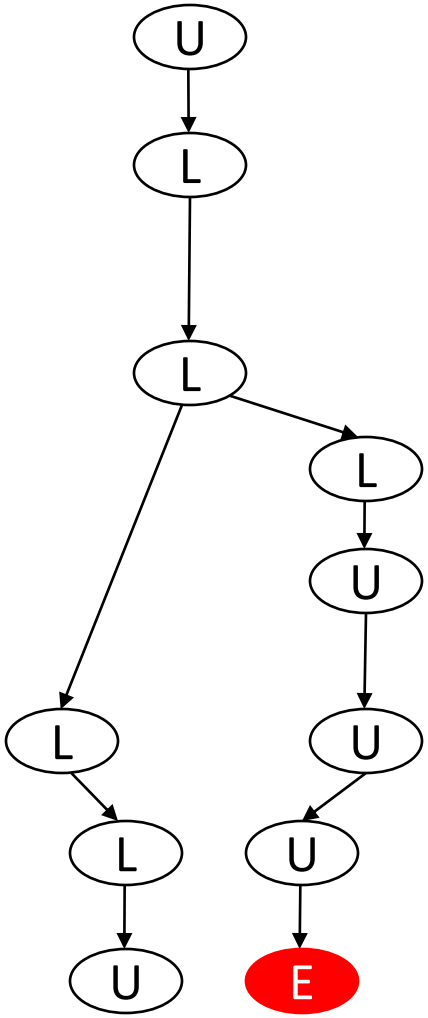
Example

Does this code obey the locking rule?

```
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

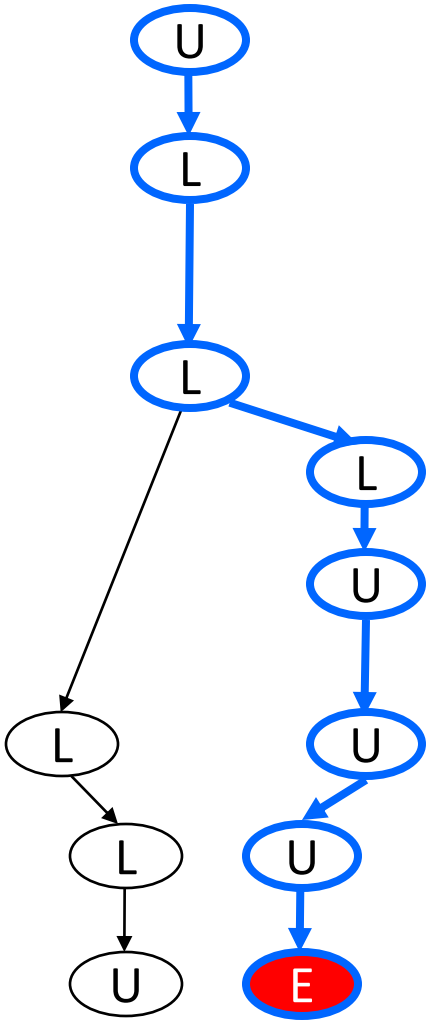
Example

Model checking boolean program (bebop)




```
do {
    KeAcquireSpinLock();

    if(*){
        KeReleaseSpinLock();
    }
} while (*);

KeReleaseSpinLock();
```

Example



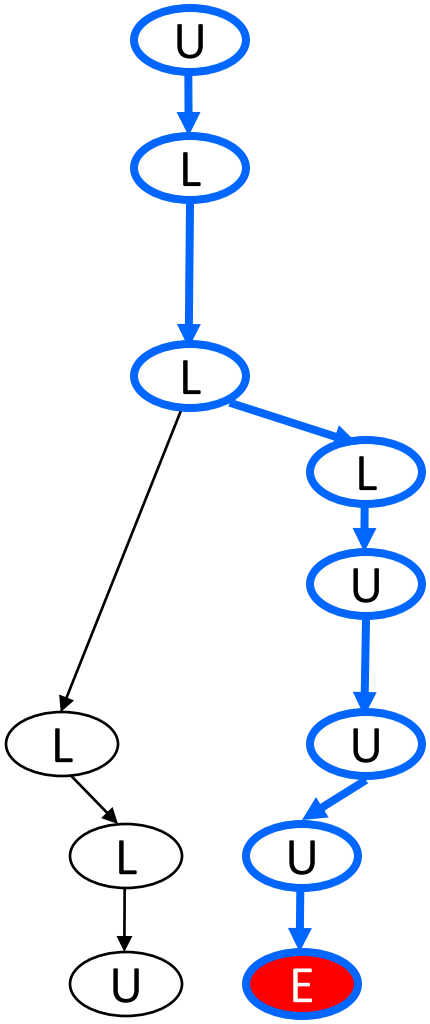

```
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
```

Is error path feasible in C program? (newton)

Example

b : (nPacketsOld == nPackets)

Add new predicate to boolean program (c2bp)

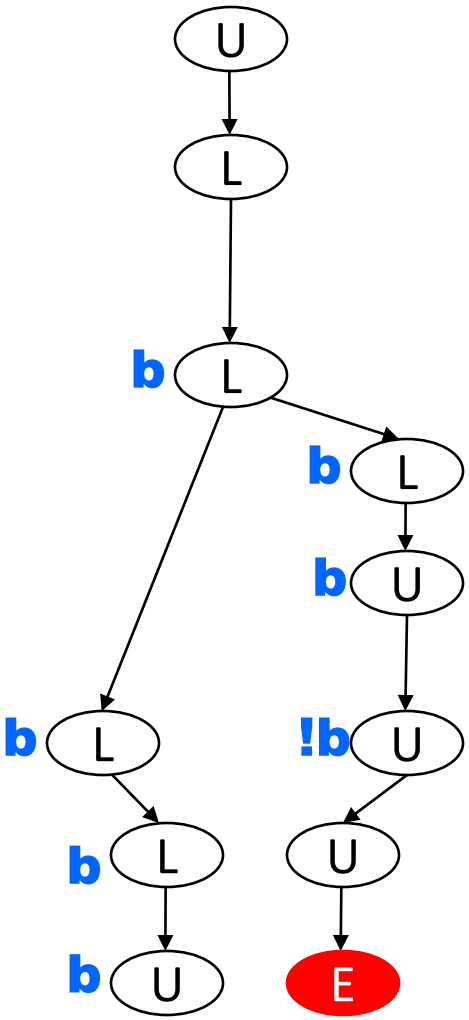



```
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets; b = true;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++; b = b ? false : *;
    }
} while (nPackets != nPacketsOld);  !b

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Model checking refined boolean program (bebop)

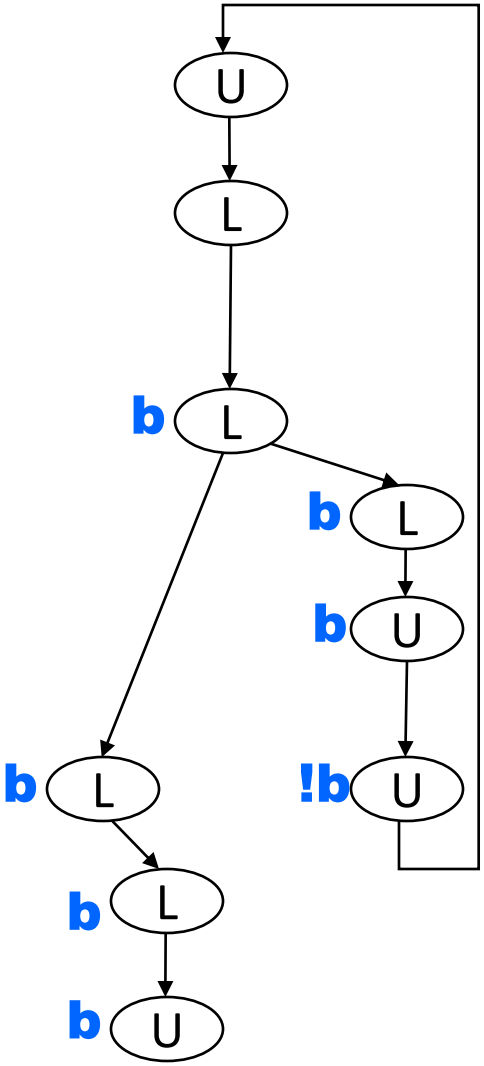



```
do {
    KeAcquireSpinLock();

    b = true;

    if(*){

        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while ( !b  );

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Model checking refined boolean program (bebop)




```
do {
    KeAcquireSpinLock();

    b = true;

    if(*){

        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while ( !b  );

KeReleaseSpinLock();
```

Observations about SLAM

- Automatic discovery of invariants
  - driven by property and a finite set of (false) execution paths
  - predicates are ***not*** invariants, but observations
  - abstraction + model checking computes inductive invariants (boolean combinations of observations)
- A hybrid dynamic/static analysis
  - newton executes path through C code symbolically
  - c2bp+bebop explore all paths through abstraction
- A new form of program slicing
  - program code and data not relevant to property are dropped
  - non-determinism allows slices to have more behaviors