SLAM & Static Driver Verifier:
Technology Transfer of Formal Methods in Microsoft

Thomas Ball

Microsoft Research

Joint work with Sriram Rajamani, Byron Cook and Vladimir Levin

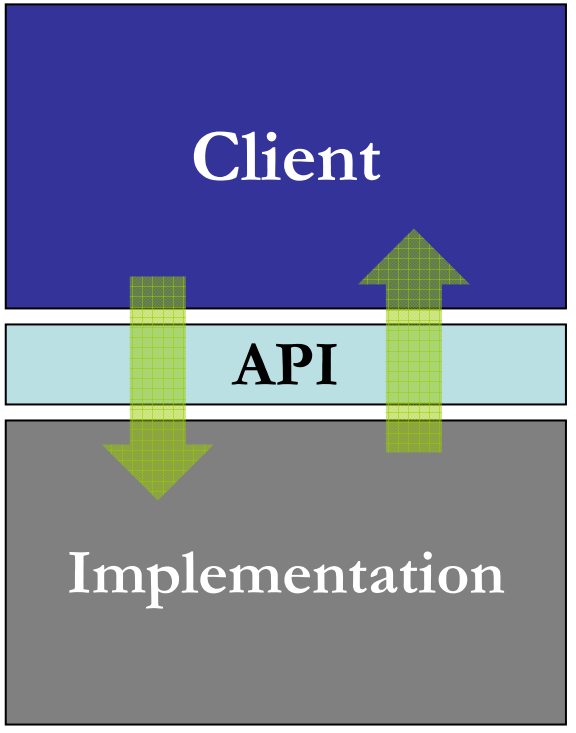
Overview

- Interface contracts

- SLAM analysis engine
  - technical overview

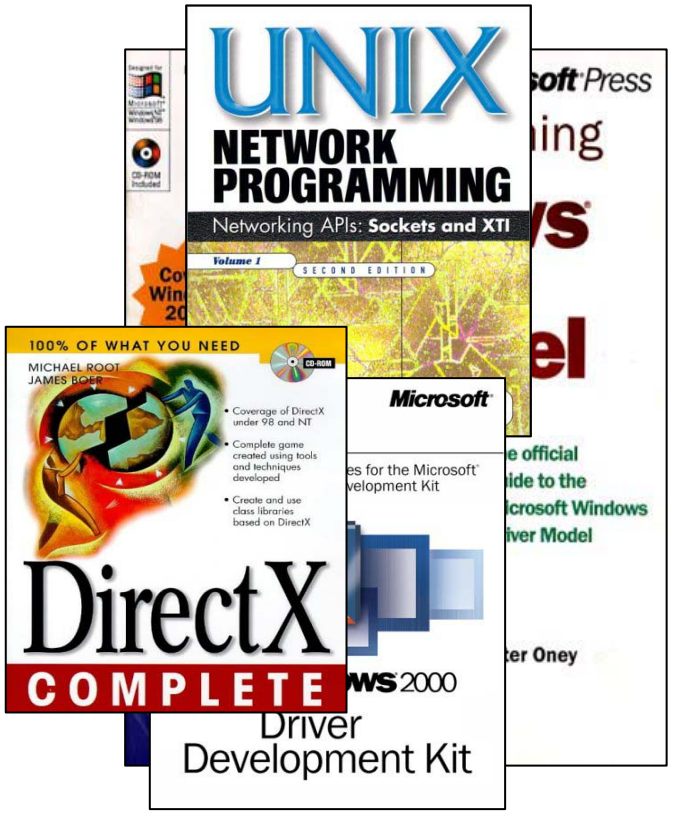
- Static Driver Verifier
  - transfer of technology to Windows

Platform Interfaces Everywhere!



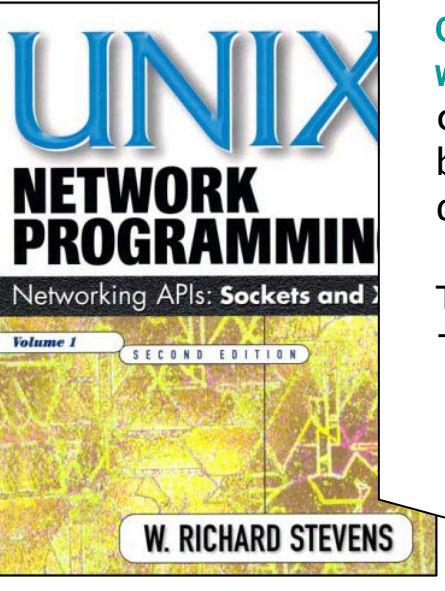
But no contracts!

Interface Contracts



- 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

Informal Contract: Sockets



the "communication domain" in which communication is to take place; see protocols(5).

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a connected state before any data may be sent or received on it. A connection to another socket is created with a connect(2) call. Once connected, data may be transferred using read(2V) and write(2V) calls or some variant of the send(2) and recv(2) calls. When a session has been completed a close(2V), may be performed. Out-of-band data may also be transmitted as described in send(2) and received as described in recv(2).

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of

Formalizing Contracts

- Pre/post conditions
  - Hoare logic
  - Eiffel: “design by contract”, integrated into language
  - JML: pre/post language
- Monitors
  - security automata
  - SLIC - SLAM’s API rule language
- Models
  - ASML: separate modeling language

Why are Contracts Useful?

- Precision in specification & design
- Separation of concerns
- Documentation
- Checking/Testing
  - dynamic (run-time)
  - static (compile-time)
- Responsibility, enforceability, liability, …

Contract Checking

- Precisely specify contracts
  - partial specifications for interfaces
- Client code is automatically checked against contracts
- Different from proving program correctness
  - contracts are not complete

Does a given contract 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 contract checking: A Study of Tradeoffs

- Soundness vs. completeness
  - false positives
  - false negatives
- Annotation burden on the programmer
- Complexity of the analysis
  - local vs. global
  - precision vs. efficiency
  - space vs. time

Broad classification

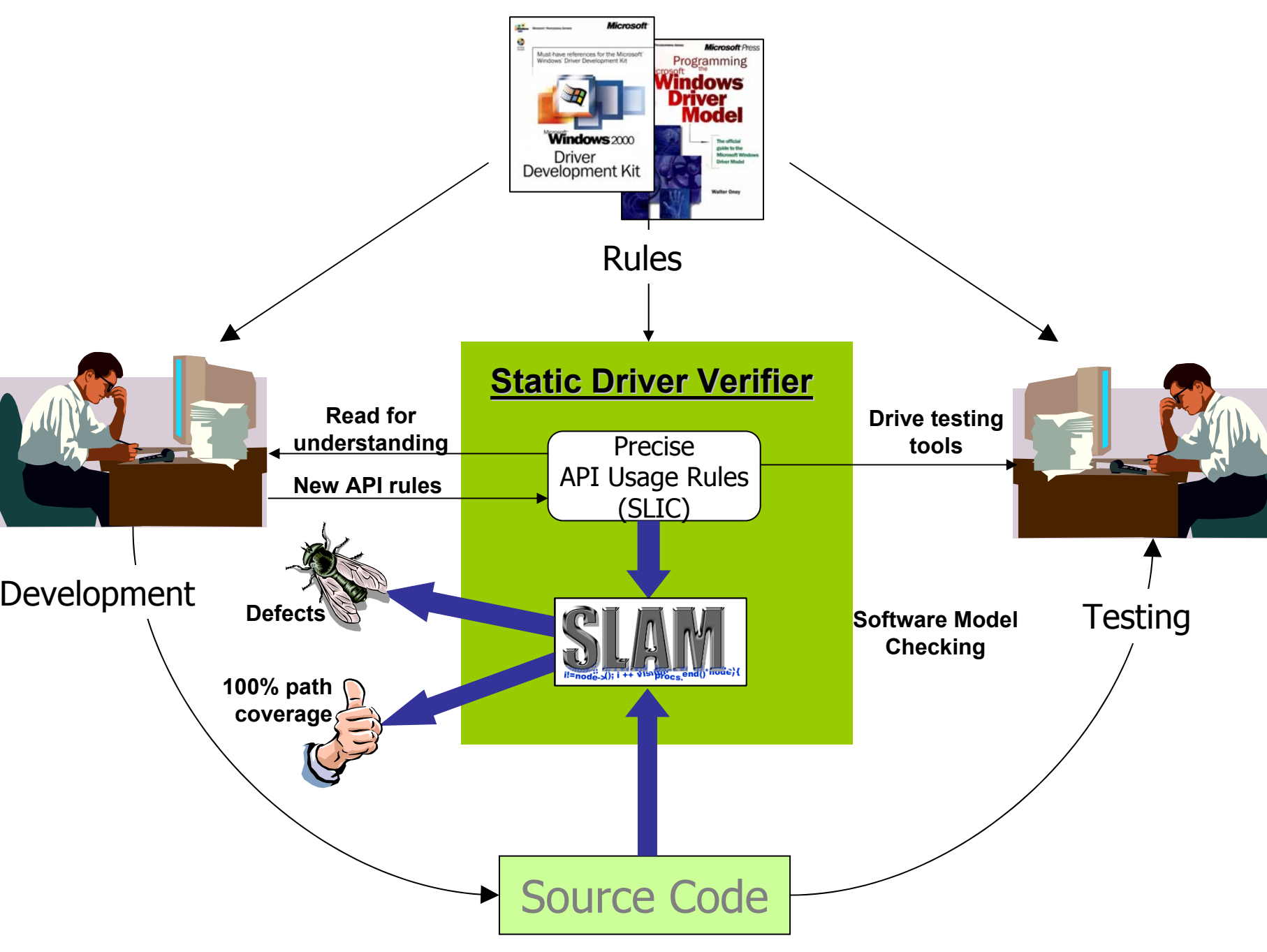
- Underapproximations
  - testing
    - after passing testing, a program may still violate a given property
- Overapproximations
  - type checking
    - even if a program satisfies a property, the type checker for the property could still reject it

Contract Checking

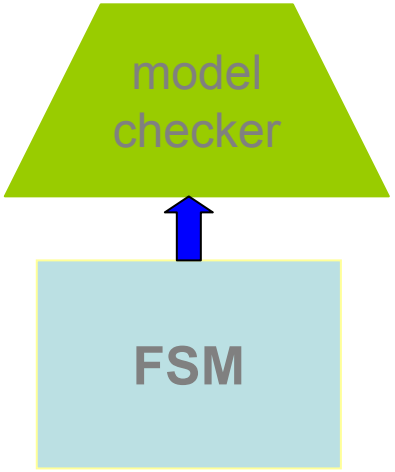
- Confluence of techniques from different fields:
  - model checking
  - automatic theorem proving
  - program analysis
- Significant emphasis on practicality
- New projects in industry and academia
  - SLAM, Feaver, BLAST, Magic, Metal, Mops, …

Overview

- Interface contracts
- SLAM analysis engine
  - technical overview
- Static Driver Verifier
  - transfer of technology to Windows


Microsoft
Windows 2000
Driver
Development Kit
Microsoft Press
Programming the
Windows
Driver
Model
Rules
Static Driver Verifier
Read for
understanding
New API rules
Precise
API Usage Rules
(SLIC)
Drive testing
tools
Development
Defects
SLAM
Software Model
Checking
Testing
100% path
coverage
Source Code

Traditional approach

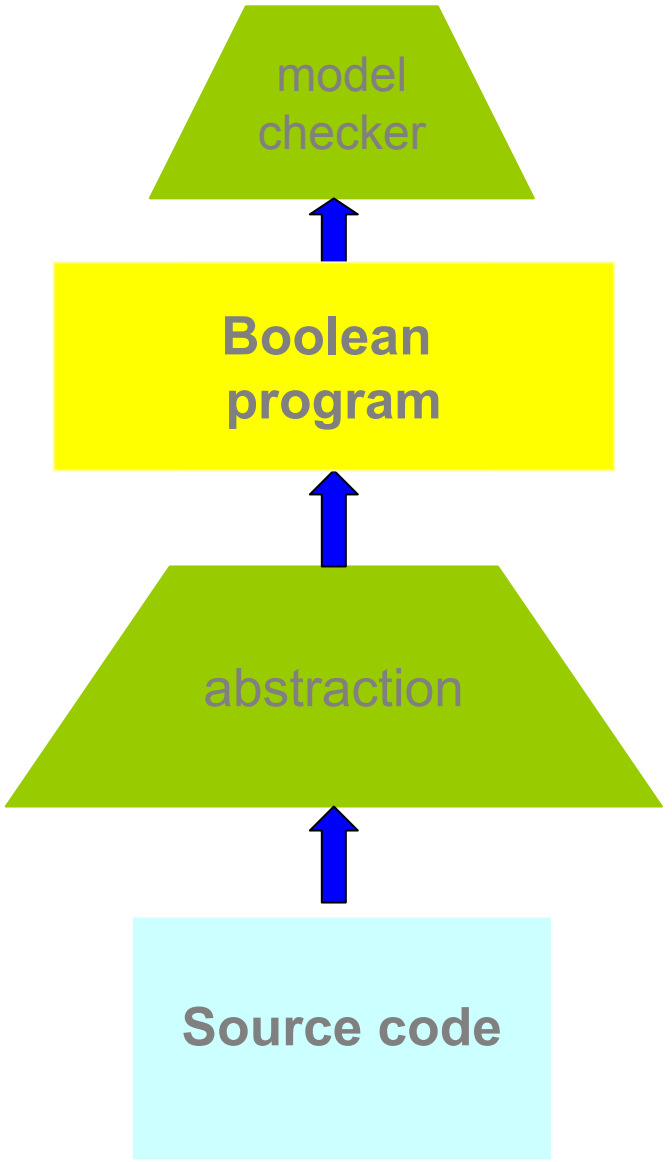


Finite state machines

Sequential C program

Automatic abstraction

SLAM

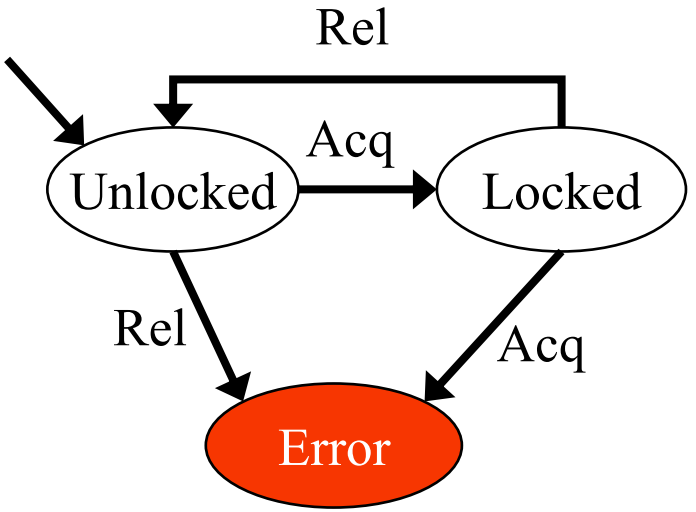


Finite state machines

C data structures, pointers, procedure calls, parameter passing, scoping,control flow

Sequential C program

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;
}
```

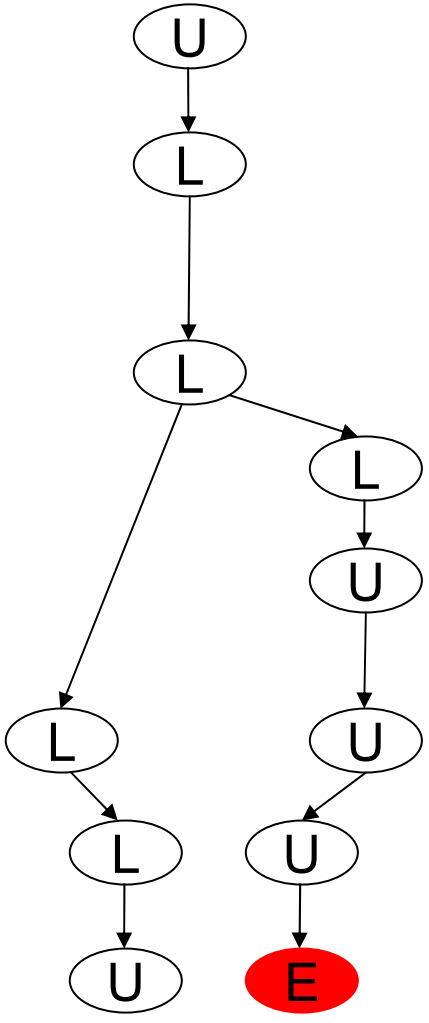
Example

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)

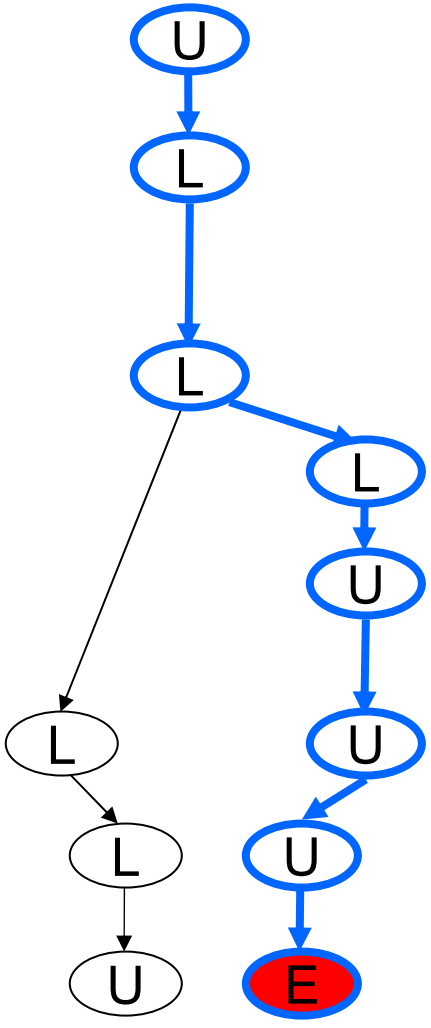



```
do {
   KeAcquireSpinLock();



   if(*){

       KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Is error path feasible in C program? (newton)

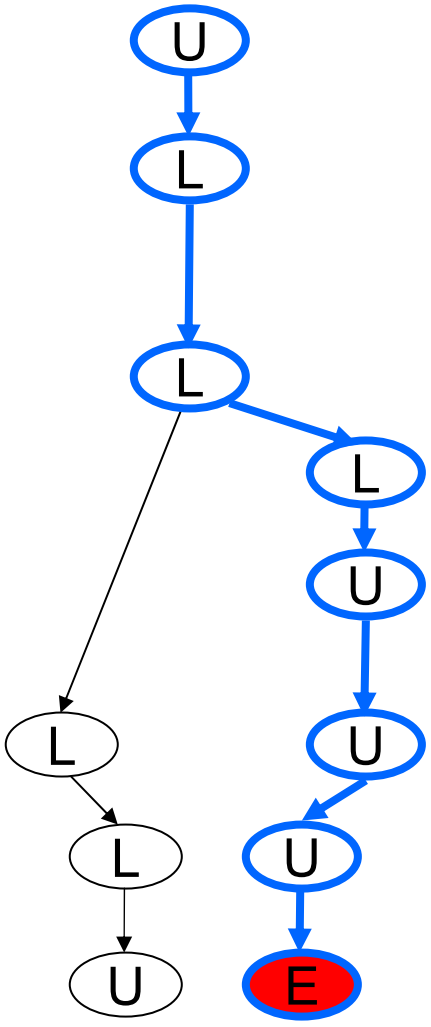



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Add new predicate to boolean program (c2bp)

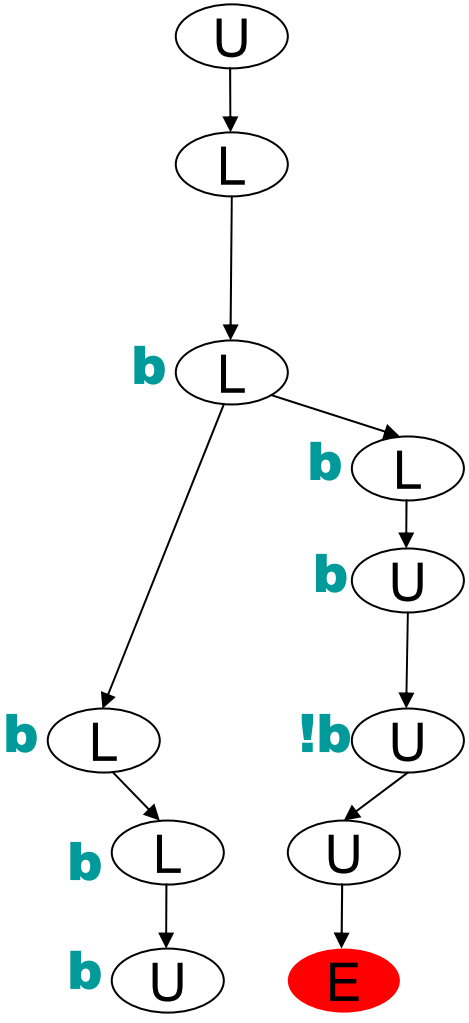



```
do {
   KeAcquireSpinLock();

   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)

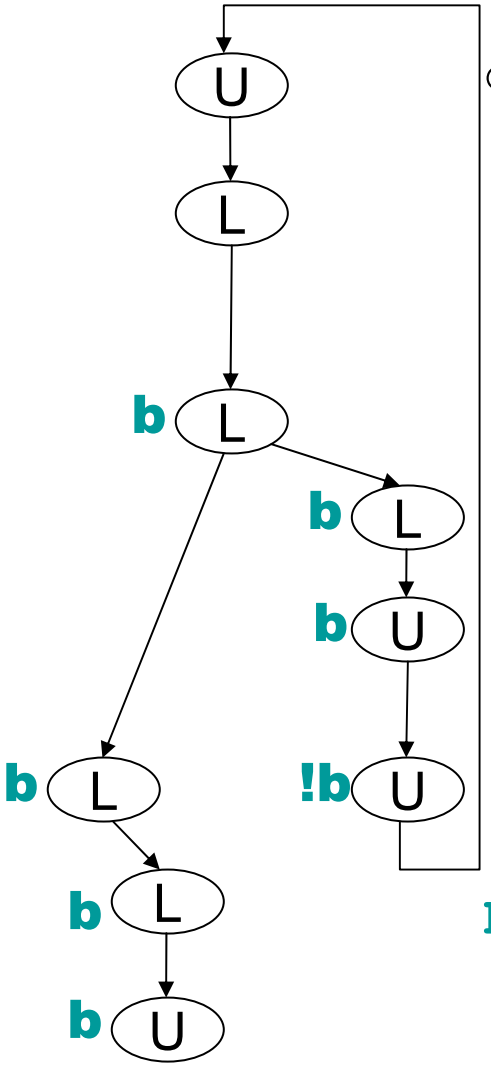



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Model checking refined boolean program (bebop)

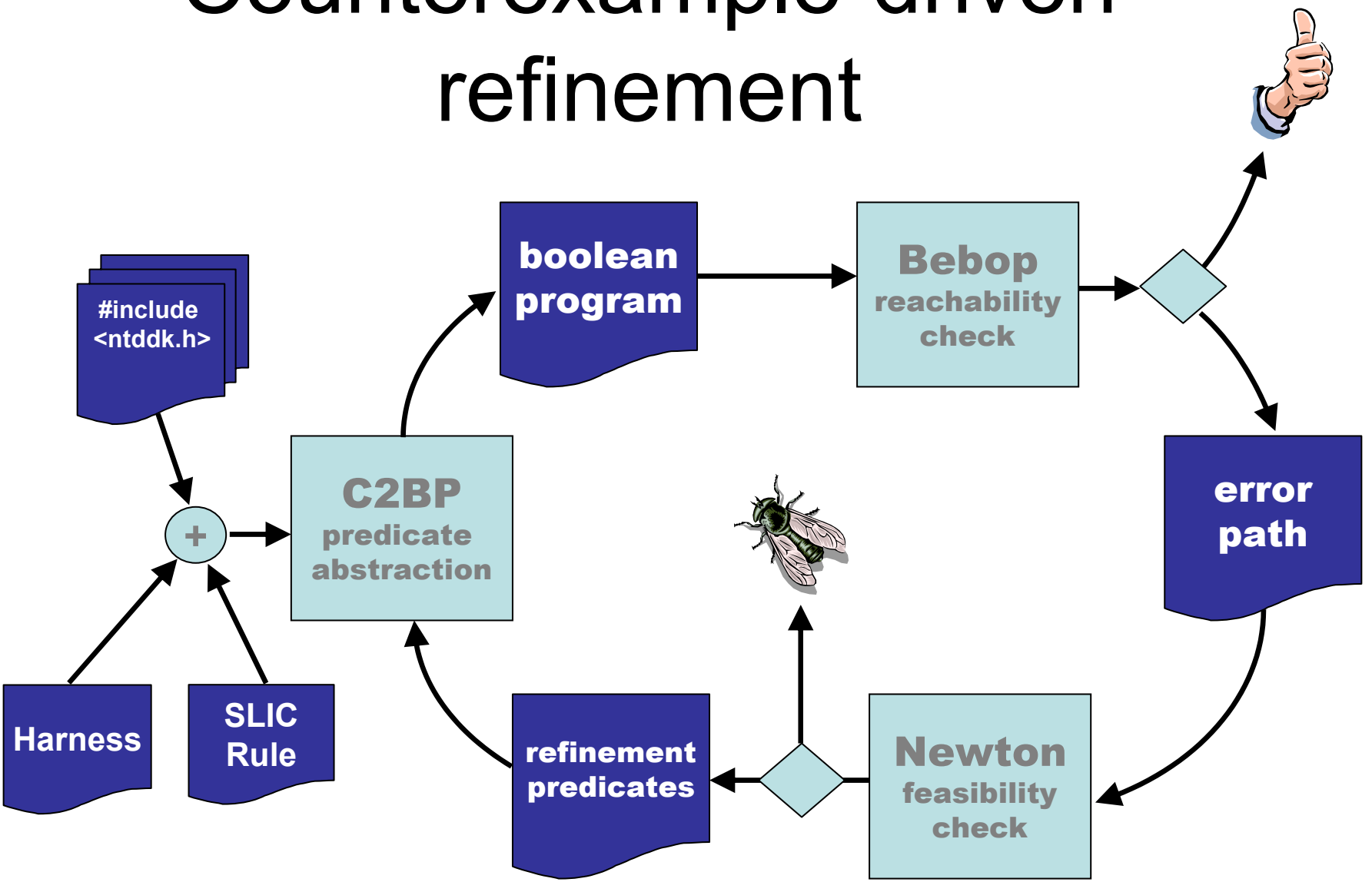



```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

Inferred Invariant

"The lock is held at the end of the loop if and only if nPackets == nPacketsOld"

Counterexample-driven refinement



Kurshan '93, Ball/Rajamani '00, Clarke et al '00

Observations about SLAM

- Automatic discovery of invariants
  - driven by property and false negatives
  - predicates are ***not*** invariants, but *observations*
  - abstraction + model checking computes invariants

- A new form of program slicing
  - code and data not relevant to property omitted
  - non-determinism allows slices to have more behaviors

Overview

- Interface contracts
- SLAM analysis engine
  - technical overview
- Static Driver Verifier
  - transfer of technology to Windows

SLAM/SDV History

- **2000-2001**
  - foundations, algorithms, prototyping
  - papers in CAV, PLDI, POPL, SPIN, TACAS
- **March 2002**
  - Bill Gates review
- **May 2002**
  - Windows committed to hire two Ph.D.s in model checking to support Static Driver Verifier
- **July 2002**
  - running SLAM on 100+ drivers, 20+ properties
- **September 3, 2002**
  - made initial release of SDV to Windows (friends and family)
- **April 1, 2003**
  - made wide release of SDV to Windows (any internal driver developer)
- **September, 2003**
  - team of six in Windows working on SDV
  - researchers moving into "consultant" role
- **November, 2003**
  - demonstration at Driver Developer Conference

December, 2004
ready to ship!!

Successes

- Static Driver Verifier is now deployed within Microsoft:
  - "This bug would be a really hard bug to find other than with a tool like SDV. There are just too many details to keep track of to have a good chance of finding it."
  - "These are all real, difficult to discover bugs. Good work!"
  - "This bug would have been very difficult to find by inspection and it was one of those bugs that would be near-impossible to reproduce…"
  - "Fixing this bug will definitely stop some unexplainable and hard to debug random system crashes in the future."

Successes

- Static analysis tools (such as SDV) are now a part of the standard suite of tools used within Microsoft
- These tools are becoming available to Microsoft's customers
- These tools are encouraging specification and more modular design

Some Lessons Learned

- People power
- Focus on problems not solutions
- Exploit synergies and shoulders
- Plan carefully
- Cultivate champions
- Embedded verification experts
- “Push button” technology is not simple

People Power

Software Productivity Tools group members

- Sriram Rajamani, Manuvir Das, Rob DeLine, Jim Larus, Manuel Fahndrich, Rustan Leino, Jakob Rehof, Shaz Qadeer

SLAM summer interns

- Sagar Chaki, Todd Millstein, Rupak Majumdar (2000)
- Satyaki Das, Wes Weimer, Robby (2001)
- Jakob Lichtenberg, Mayur Naik (2002)
- Jakob Lichtenberg, Shuvendu Lahiri, Georg Weissenbacher, Fei Xie (2003)

SLAM Visitors

- Giorgio Delzanno, Andreas Podelski, Stefan Schwoon

Static Driver Verifier: Windows Partners

- Byron Cook, John Henry, Vladimir Levin, Con McGarvey, Bohus Ondrusek, Abdullah Ustuner
- Neill Clift, Nar Ganapathy, Adrian Oney, Johan Marien, Bob Rinne, Rob Short, Peter Wieland

Focus on Problems not Solutions

- Device driver problem
  - important to Microsoft
  - testing insufficient to ensure quality
  - many complexities but code of reasonable size
- Problem space guides search for solution
  - control-dominated properties $\Rightarrow$ boolean programs
  - no annotations $\Rightarrow$ counterexample-driven refinement

Exploit Synergies and Shoulders

- Diverse backgrounds of investigators
- SLAM built on strong foundations
  - program analysis
  - model checking
  - automated deduction
- Infrastructure
  - MS compiler front-end and alias analysis
  - CUDD BDD library
  - Simplify theorem prover
  - OCaml programming language

Plan Carefully

- Creativity = 10% inspiration + 90% perspiration
- Initial technical report
  - laid out plan, left open problems
  - recruiting/preparing interns
- Demo milestones
- Software process
  - open software architecture
  - code ownership, code reviews, code refactoring and cleanup
  - regression test suite

Cultivate Champions

- Device driver experts
  - Adrian Oney, Peter Wieland
  - explained subtleties of kernel
  - reviewed rules and error traces

- Management champions
  - Bob Rinne, Base OS
  - Amitabh Srivastava, PPRC

Embedded Verification Experts

- Windows committed to hire two Ph.D.s with verification expertise
  - Byron Cook and Vladimir Levin
  - offices in both development and research
- Virtual team worked closely together for 1.5 years
- Product team now has 6 people full-time
- High bandwidth channel between groups

Making It “Push Button”

- Without rules, SLAM does nothing
  - developing rules is an error-prone process, especially for legacy APIs
- Environment model costly to build as well
  - for drivers, environment is the Windows kernel
- Rule designer needs to know a lot about SLAM to get efficiency

Conclusions

- The technology now exists for enforcing simple API contracts
- Rollout/adoption
  - first as out-of-band tools (i.e., SLAM/SDV)
  - next as in-band tools (part of language/compiler)
- Many variables in equation of technology transfer
  - keep your eyes wide open!

Further Reading

See papers, slides from:

http://research.microsoft.com/slam

http://research.microsoft.com/~tball