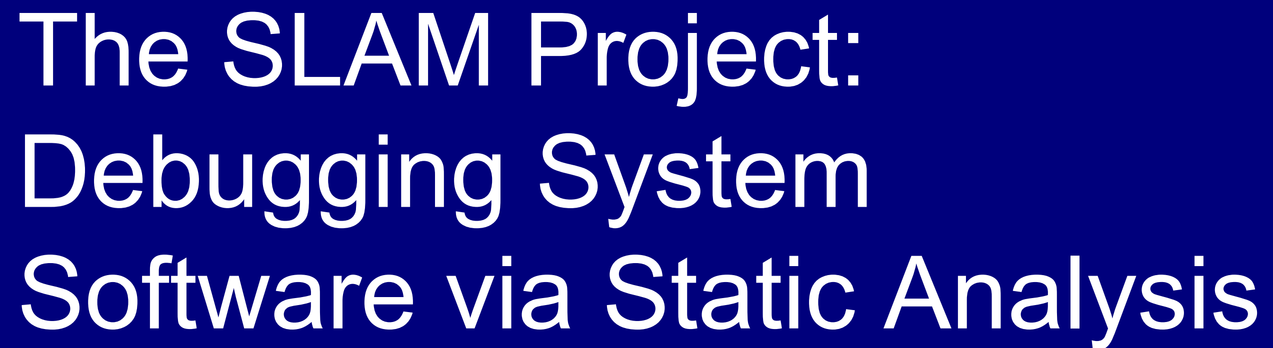


The SLAM Project: Debugging System Software via Static Analysis

Thomas Ball
Sriram K. Rajamani

Microsoft Research
http://research.microsoft.com/slam/

Agenda

- Overview
- Demo
- Under the hood
- Technology transfer
- Questions

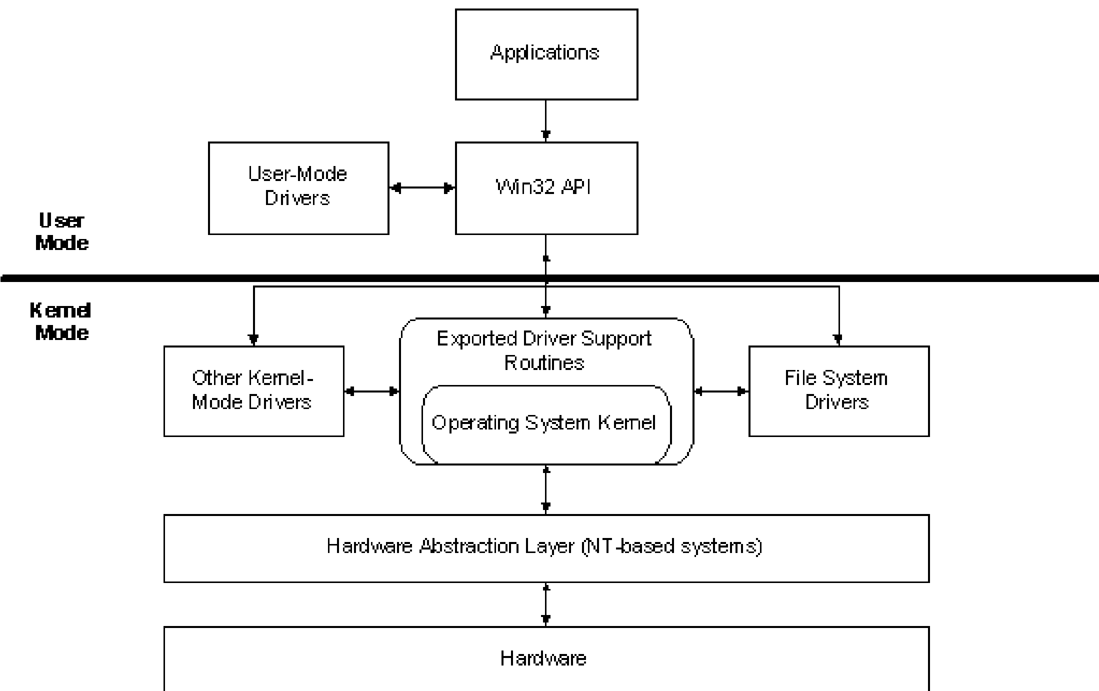
Specifying and Checking Properties of Programs

- Goals
  - defect detection
  - partial validation
- Properties
  - memory safety
  - temporal safety
  - security
  - …
- Many approaches
  - automated deduction
  - program analysis
  - type checking
  - model checking
- Many projects
  - Bandera, BLAST, CANVAS, ESC-Java, ESP, FeaVer, FLAVERS, JPF, PolyScope, PREfix, Programatica, rccjava, Splint, TVLA, Verisoft, Vault, xgcc, …

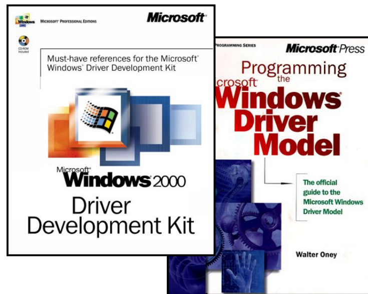
Software Productivity Tools Research http://research.microsoft.com/spt/

- New language design
  - Vault
    - safe systems programming language
  - Behave!
    - message passing + behavioral types + model checking
- Program analysis of legacy (C) code
  - ESP
    - scalable analysis
  - SLAM
    - precise analysis based on software model checking

Focus: Windows Device Drivers



- Kernel presents a very complex interface to driver
  - stack of drivers
  - NT kernel multi-threaded
  - IRP completion, IRQL, plug-n-play, power management, …
- Correct API usage described by finite state protocols
- Automatically check that clients respect these protocols



Rules

Development

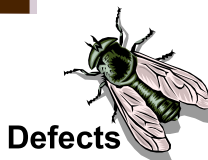
Read for understanding

New API rules

Unambiguous API Usage Rules (SLIC)

Drive testing tools

Testing

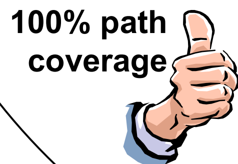


Defects

SLAM

Software Model Checking

100% path coverage



Source Code

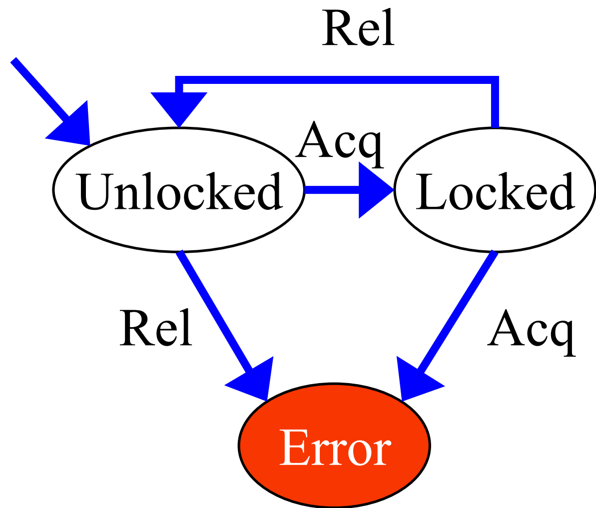
The SLAM Thesis

- We can *soundly* and *precisely* check an *unannotated* program against temporal safety properties by
  - *creating* a program abstraction
  - *exploring* the abstraction's state space
  - *refining* the abstraction
- We can scale such an approach to many 100kloc via
  - modular analysis
  - model checking

Results on Drivers

- Automatically analyzed 30+ drivers in the Windows DDK against 4 properties
  - 2-30Kloc (200Kloc total)
  - 3 hours to run (4-proc, 700Mhz, 2Gb server)
  - over 20 real errors found (and counting)
- Successful application of fully automated software model checking

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;
}
```

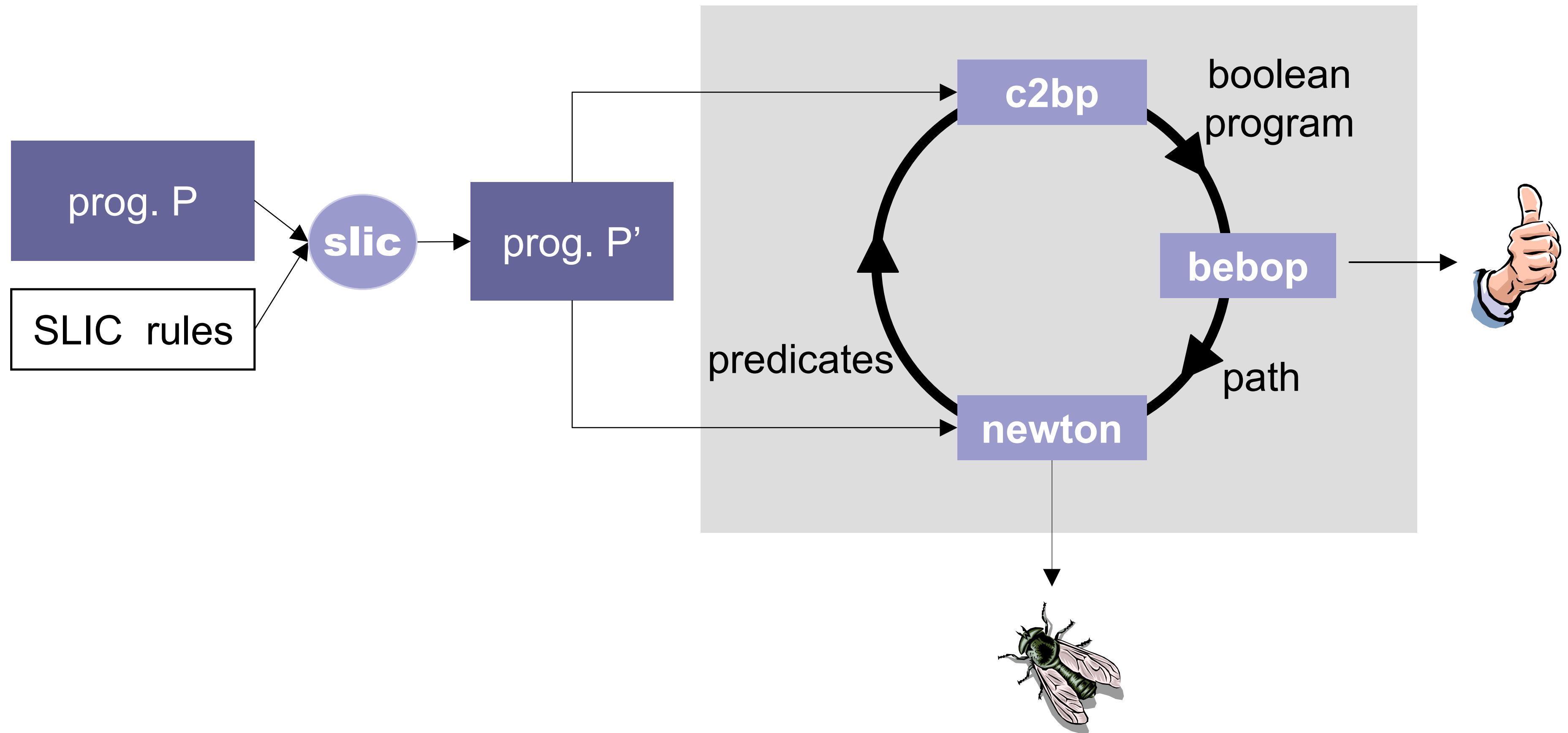
start NP
SKIP1
CallDriver
SKIP2
return child status
Skip
CallDriver
IPC
MPR3
synch
NP
CallDriver
not pending returned
prop completion
PPC
MPR completion
Complete request
MPR2
CallDriver
MPR1
DC
return not Pend
synch
no prop completion
CallDriver
IRP accessible
N/A
N/A

start P
Mark Pending
SKIP1
CallDriver
SKIP2
Skip
CallDriver
IPC
MPR3
synch
NP
CallDriver
return Pending
not pending returned
prop completion
PPC
MPR completion
Complete request
MPR2
CallDriver
MPR1
DC
no prop completion
CallDriver
N/A

Demo

- Example
- Some technical details

The SLAM Process



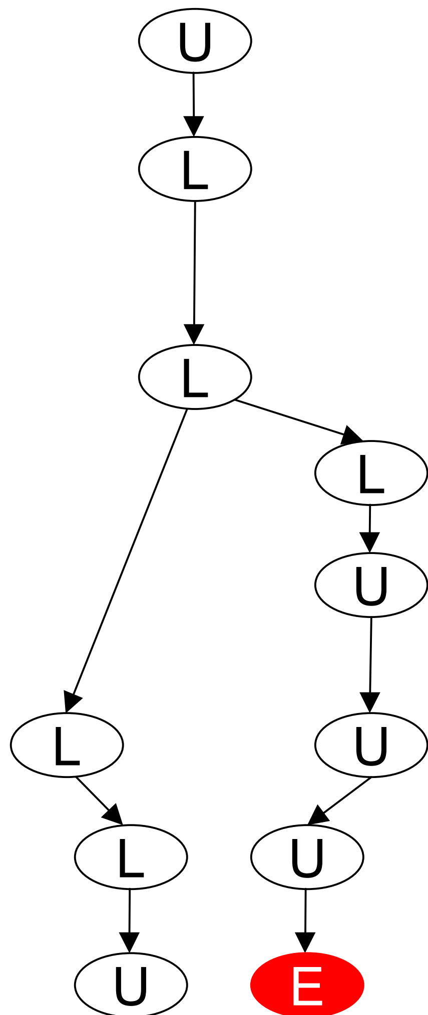
Example

Does this code obey the locking rule?

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

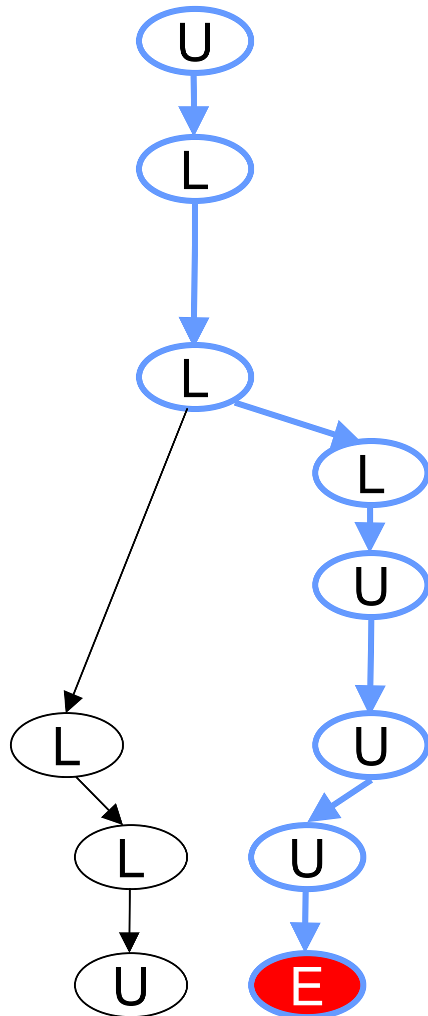
Model checking boolean program (bebop)




```
do {
   KeAcquireSpinLock();

   if(*){

      KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Example

Is error path feasible in C program? (newton)

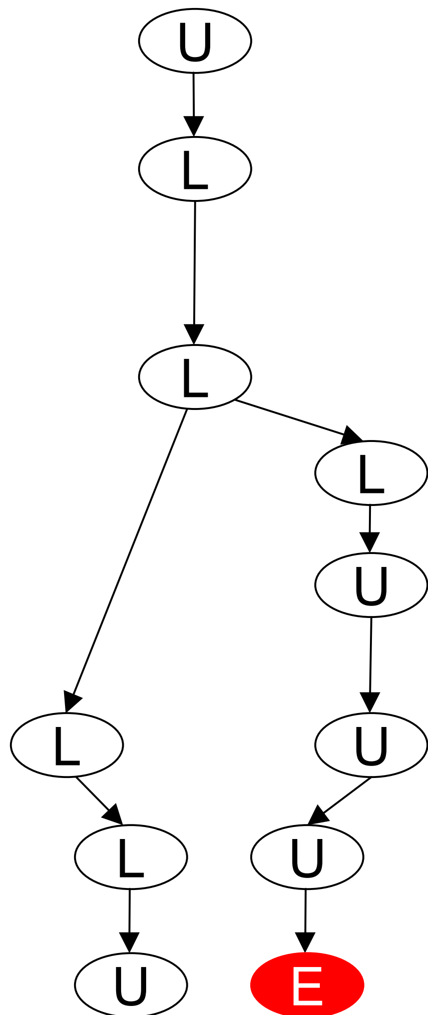



```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

b : (nPacketsOld == nPackets)

Add new predicate to boolean program (c2bp)

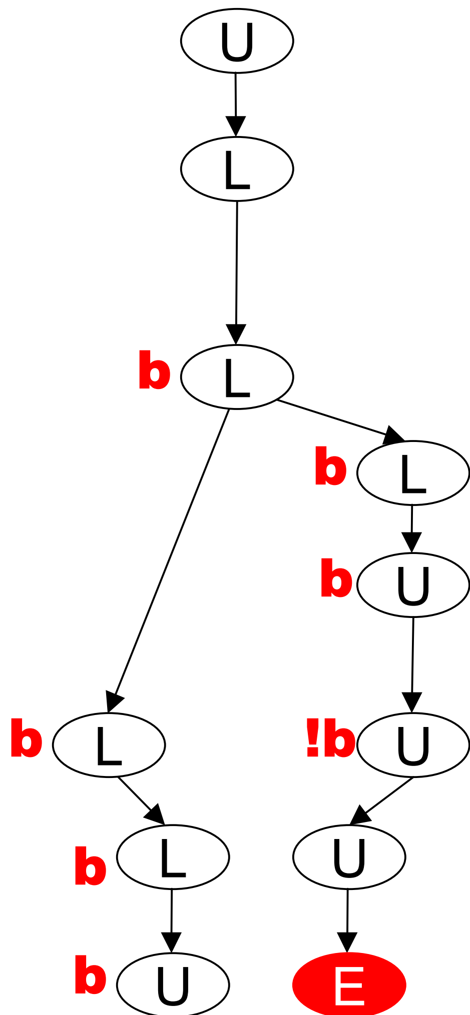



```
do {
   KeAcquireSpinLock();

   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)

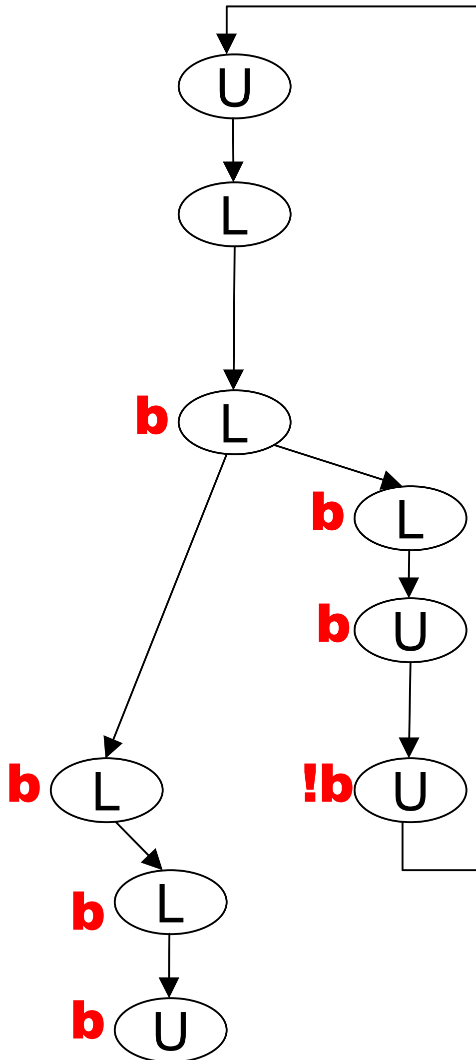



```
do {
   KeAcquireSpinLock();

   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();
```

SLAM Tools

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

Model Creation (c2bp)

- Input
  - a C program P
  - set of predicates E
- Goal
  - Produce a boolean program (abstraction) of C program
  - One boolean variable per predicate
- Output: a *boolean program* that is
  - a sound abstraction of P
  - a precise abstraction of P
- Main result
  - modular predicate abstraction of C in presence of pointers and procedures

Assignment Example

Statement in P: `y:=y+1;`

Predicates in E: $\{ y=2, y<5 \}$

Weakest Precondition:

pre(`y:=y+1`, $y<5$) = $(y<4)$

Strengthenings:

$S(y<4) = y=2$

$S(y>4) = !(y<5)$

Abstraction of s in B:

```
[y<5] := [y=2] ? true
                : (![y<5] ? false : *);
```

Model Checking (bebop)

- Interprocedural dataflow analysis via BDDs
  - *Explicit* representation of control flow graph
  - *Implicit* representation of reachable states via BDDs
    - boolean program reachability
    - *set* of bit-vectors per statement, represented by a BDD
- Worst-case complexity is $O(P(GL)^3)$
  - P = program size
  - G = number global states in state machine
  - L = number of local states in procedure
  - exploits locality of scoping from procedures

Model Refinement (newton)

- Symbolically execute path in C program
  - VCgen along a single path
- Check for path infeasibility using automatic decision procedures
  - **simplify** theorem prover
- If infeasibility detected
  - generate new predicates or “explanation”
  - new predicates rule out infeasible path in subsequent iterations

Newton: Path Simulation

Store:

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β
- β ->WLHeadVa: γ (3)

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β
- β ->WLHeadVa: γ (3)
- request: γ (3,4)

Conditions:

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β
- β ->WLHeadVa: γ (3)
- request: γ (3,4)

Conditions:

$!\gamma$ (5)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β
- β ->WLHeadVa: γ (3)
- request: γ (3,4)

Conditions:

$! \gamma$ (5)

$\alpha \mathrel{!=} \alpha$ (1,2)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)
- devExt: β
- β ->WLHeadVa: γ (3)
- request: γ (3,4)

Conditions:

α != α (1,2)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Store:

- nPacketsOld: α
- nPackets: α (1)

Conditions:

α != α (1,2)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Predicates:

(nPacketsOld == α)

(nPackets == α)

(α != α)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Predicates:

(nPacketsOld != nPackets)

Newton

```
nPackets = nPacketsOld;

request = devExt->WLHeadVa;

assume(!request);

assume(nPackets != nPacketsOld);
```

Predicates:

(nPacketsOld == nPackets)

SLAM's Path to Tech. Transfer

- 2000-2001
  - innovating on top of a large body of analysis research
  - designing, writing, implementing, giving talks…
  - static analysis at Microsoft
  - acquiring device driver domain expertise and dev friends
  - last but not least… **interns and visitors**
- Recent checkpoints
  - February 2002: MSR TechFest
  - March 5 2002: **billg review**
- Next steps
  - stability
  - internal tool release

SLAM Chronology

- Spring 2000
  - process and algorithms
  - **bebop**
- Summer 2000
  - **c2bp**
  - **checked a property of a driver**
- Autumn 2000
  - formalize precision of c2bp
  - **newton**
  - **checked properties of a few drivers from DDK**
- Winter 2000
  - **SLIC** specification language and instrumentation took
- Spring 2001
  - **found first real error in a driver**
    - total automation
- Summer 2001
  - **running on a handful of drivers**
  - relative completeness result
  - scale, scale, scale ...
- Autumn 2001
  - many more process optimizations
  - development of more properties
  - integration with DDK build
- Winter 2001
  - **running on over 30+ drivers in DDK on various properties**

SLAMming on the shoulders of …

- Model checking
  - predicate abstraction
  - counterexample-driven refinement
  - BDDs and symbolic model checking
- Program analysis
  - abstract interpretation
  - points-to analysis
  - dataflow via CFL-reachability
- Automated deduction
  - weakest preconditions
  - theorem proving
- Software
  - AST toolkit
  - Das's points-to analysis
  - CU and CMU BDD libs
  - SRC's simplify
  - OCAML

- About 3 years of experience with static analysis for defect detection
  - PREfix
  - PREfast

- Developers
  - aware of benefits and pitfalls of static analysis
  - willing to provide information to tools

Domain Expertise

- Learning about drivers
  - slow, painful process
  - documents imprecise
- Adrian Oney, our champion in Windows
  - wrote “Driver Verifier” test tool (part of Windows XP)
  - interested in techniques to improve driver quality
  - help us to understand drivers and refine properties
  - positive dev presence at billg review

A Little Help from Our Friends…

- Visitors
  - Giorgio Delzanno
  - Andreas Podelski
  - Stefan Schwoon
- 2000 Interns
  - Sagar Chaki
  - Rupak Majumdar
  - Todd Millstein
- 2001 Interns
  - Sagar Chaki
  - Satayaki Das
  - Robby
  - Wes Weimer
- 2002 Interns
  - Jakob Lichtenberg

Software Tool Pipeline

- Stages in a software tool project
  - Inception
  - Prototyping
  - Proof of concept on real code
  - Stability
  - Tool used by internal customers
  - Tool used by external customers

Stability

- Stable set of rules
- Stable set of driver models
- Stable SLAM: given any one of approx. 700 kernel-mode drivers in XP
  - no SLAM errors
  - run in a reasonable amount of time
  - produce meaningful error messages

Internal Use

- Technical
  - move focus from analysis engine to user experience
  - PREfix lesson
    - analysis engine 10% of code
    - remaining 90% of code devoted to build integration, data storage, querying/filtering, user interface, scripting, …
- Educational
  - how to write API rules, interpret output
  - documentation
- Political
  - many decisions not directly under our control
  - who will support this tool?...

Conclusions

- SLAM: two years from conception to demonstration on real code
  - focus on device drivers
  - innovate on established body of work
  - aggressive schedule
- Now the real fun begins…
  - shifting focus from analysis to users
  - can we get Windows to develop maintain rules and models?