Boolean and Cartesian Abstraction for Model Checking C Programs

Thomas Ball, Andreas Podelski, Sriram K. Rajamani @ MSR

Werner A. König

Prof. Dr. Stefan Leue
Chair for Software Engineering

01/25/2005
University of Konstanz

Outline

- Overview
- Project Components
- History
- Challenges
- Boolean Programs and Rules
- Example
  - Abstraction
  - Model Checking
  - Refinement
- Complexity
- Conclusion

What is SLAM?

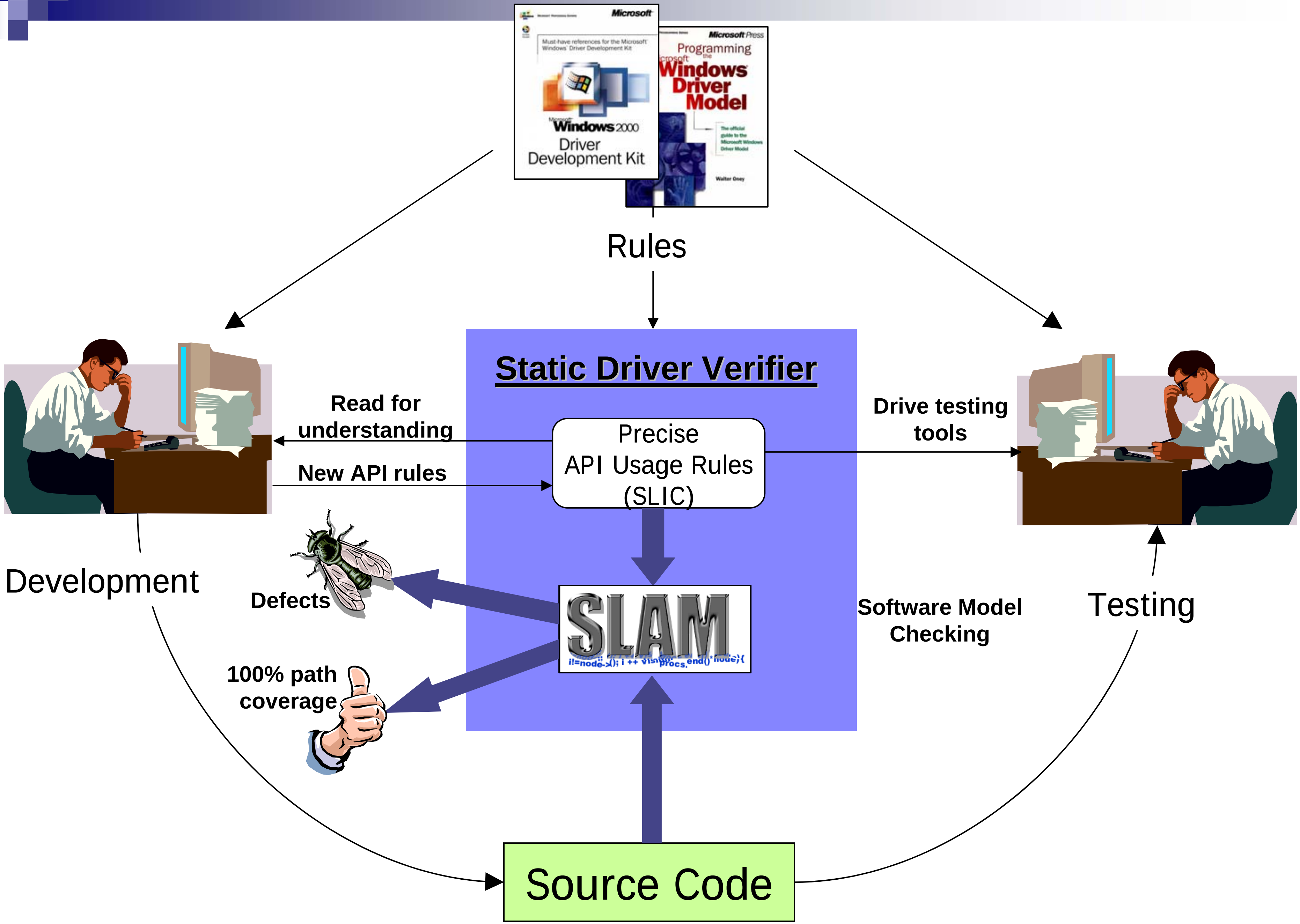
- SLAM is a software model checking project
  @ Microsoft Research MSR

- Goal:
  - Check automatically C programs (system software) against safety properties using model checking.
  - Present property violations as error traces in the source code.

- Application domain: device drivers

Used internally inside Windows for driver verification.

Planned to be released for third-party driver developer.



Rules

Static Driver Verifier

Read for understanding

New API rules

Precise API Usage Rules (SLIC)

Drive testing tools

Development

Defects

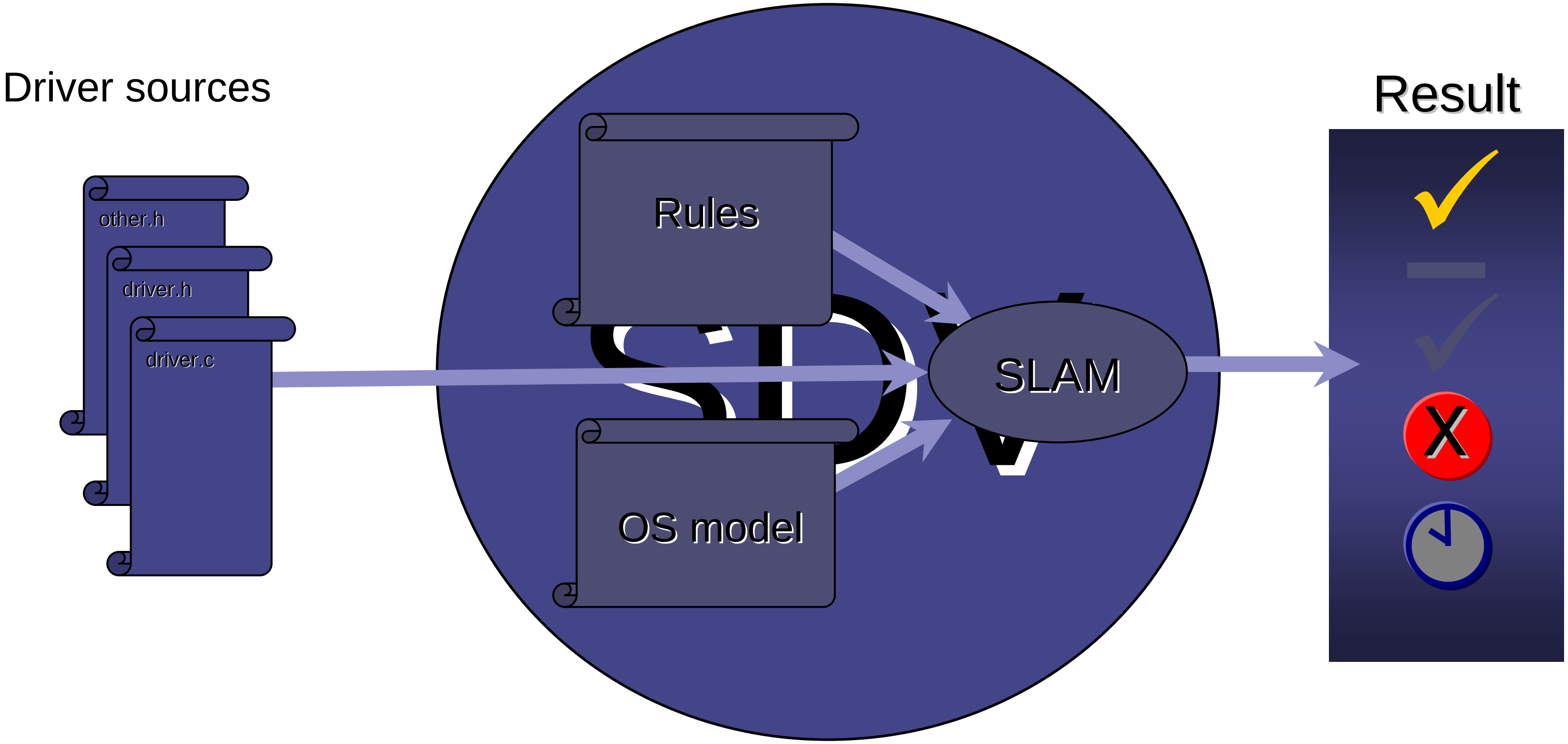
100% path coverage

Software Model Checking

Testing

Source Code

Static Driver Verifier



Driver sources

other.h

driver.h

driver.c

Rules

OS model

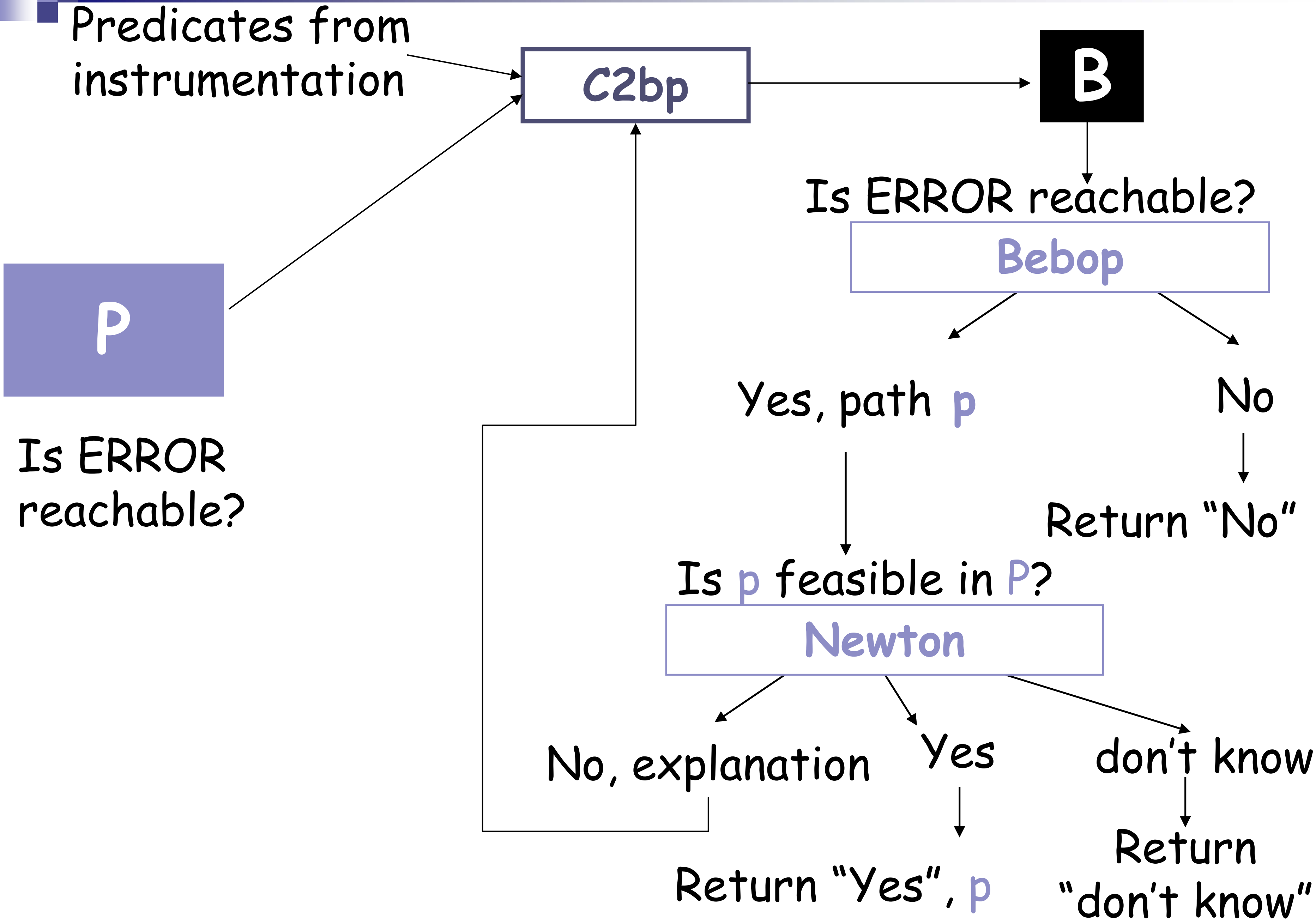
SLAM

Result

SLAM – Software Model Checking

Given a safety property to check on a C program P, the SLAM process iteratively refines a boolean program abstraction of P using three tools:

- C2bp: predicate abstraction, abstracts P into boolean program BP(P,E) with respect of predicates E over P
- Bebop: tool for model checking boolean programs, determine if ERROR is reachable in BP(P,E)
- Newton: discovers additional predicates to refine boolean program by analyzing feasibility of paths in P



Predicates from instrumentation

C2bp

B

P

Is ERROR reachable?

Is ERROR reachable?

Bebop

Yes, path p

No

Return “No”

Is p feasible in P?

Newton

No, explanation

Yes

don’t know

Return “Yes”, p

Return “don’t know”

Why Driver Domain?

- Most drivers run within the Windows kernel
- Can cause the kernel to crash or hang
- Very complex and unpredictable environment
- Drivers are mostly written by third-party developer
- Driver failures are perceived by the end-user as a windows failure

Automated analysis of drivers

- Relatively small (< 100K LOC)
- WDM usage rules could be applied for all drivers

SLAM - History

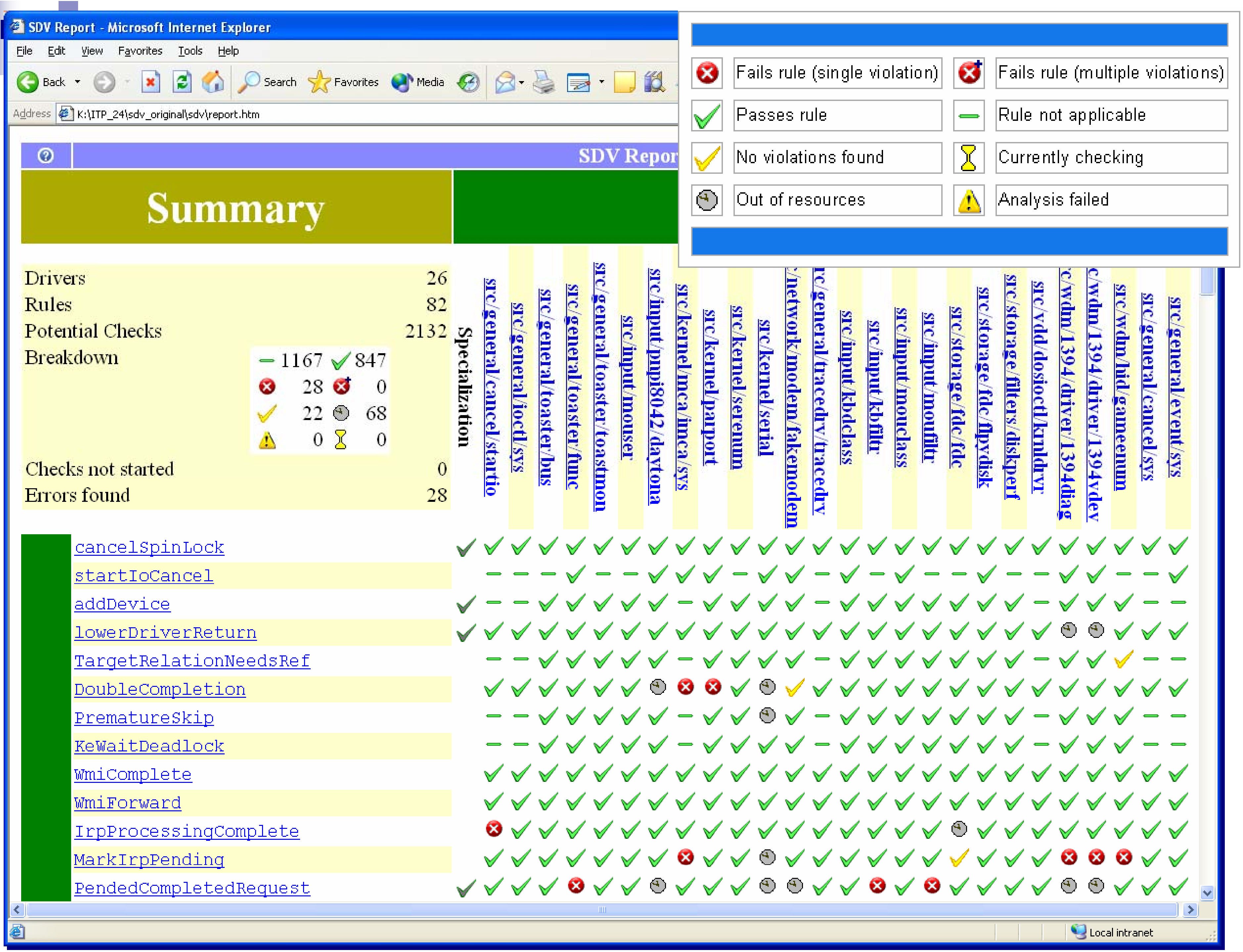
- Initial discussions: Thomas Ball, Sriram K. Rajamani @ Microsoft Research MSR
- First technical report January 2000
- Spring 2000
  - Bebop model checker
- Summer 2000
  - Initial c2bp implementation
  - Model checked a safety property of an NT driver
  - Hand instrumented code/predicates discovered by hand
- Autumn 2000
  - Predicate discovery (Newton)
  - Checked properties of drivers from DDK
  - Hand instrumented code/automatic discovery of predicates
- Winter 2000
  - SLIC specification language
- Spring 2001
  - Found first real error in production code
  - Total automation (manual model specifications)
- TACAS 2001: Boolean and Cartesian Abstractions for Model Checking C Programs, April 2001, Genoa, Italy.

Static Driver Verifier - History

- Research Prototype SLAM → Production Tool SDV

- March 2002
  - Bill Gates Review
- Mai 2002
  - Windows committed to hire two Ph.D.s in model checking to support Static Driver Verifier
- Autumn 2002
  - Joint Project between MSR and Windows
  - Initial release of SDV 1.0 to Windows (friends and family)
(SLAM engine, interface usage rules, kernel model, GUI and scripts)
- April 2003
  - wide release of SDV 1.2 to Windows
(any internal driver developer)
- November 2003
  - SDV 1.3 released at Driver Developer Conference (better integration, more rules, better models)

- Since 2004 SDV fully transferred to Windows (6 full-time positions)
- Public release planned for end of 2004 ???



✗	Fails rule (single violation)	✗⁺	Fails rule (multiple violations)
✓	Passes rule	–	Rule not applicable
✔	No violations found	⧗	Currently checking
◷	Out of resources	⚠	Analysis failed

SDV Repor

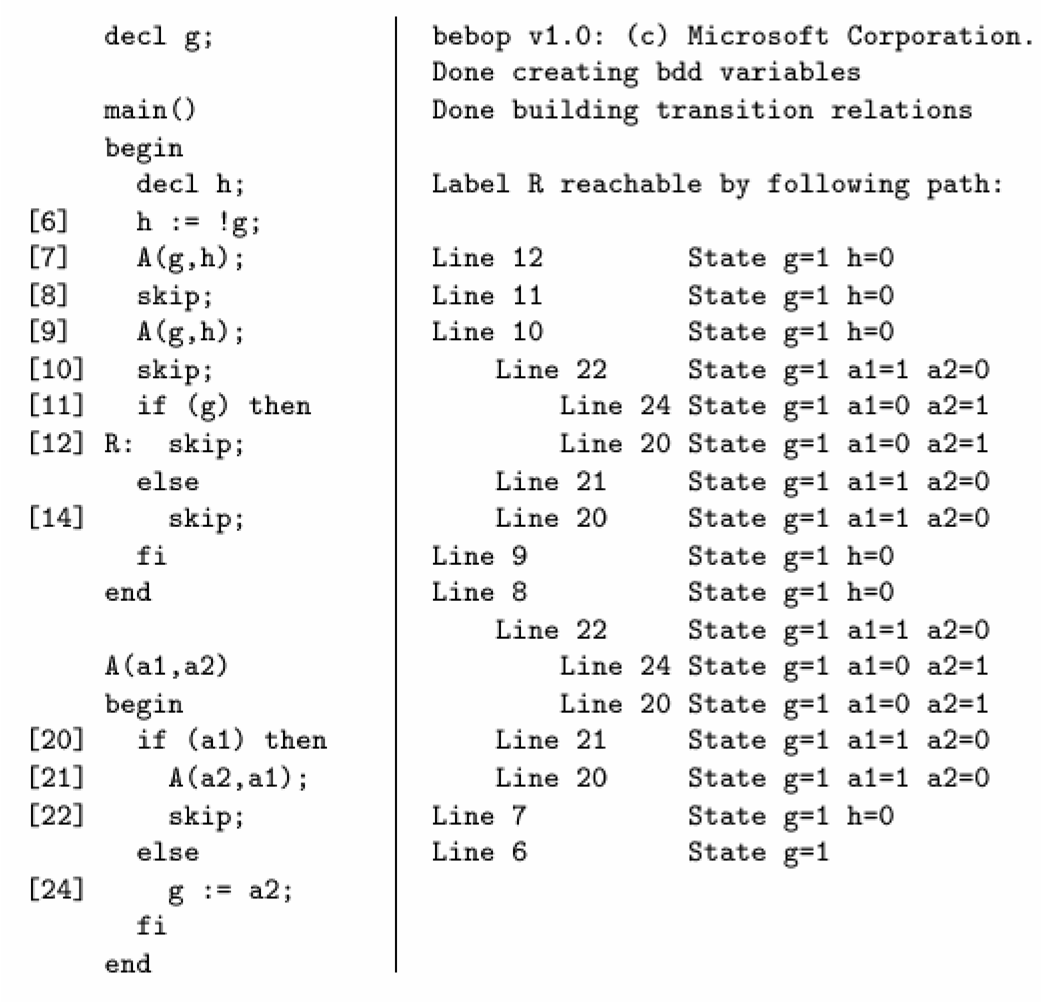
Summary

Drivers	26
Rules	82
Potential Checks	2132
Breakdown	– 1167 ✓ 847 ✗ 28 ✗⁺ 0 ✔ 22 ◷ 68 ⚠ 0 ⧗ 0
Checks not started	0
Errors found	28

	Specialization	src/general/cancel/startio	src/general/ioctl/sys	src/general/toaster/bus	src/general/toaster/func	src/general/toaster/toastmon	src/input/mouser	src/input/pnpi8042/daytona	src/kernel/mca/imca/sys	src/kernel/parport	src/kernel/serenum	src/kernel/serial	/network/modem/fakemodem
cancelSpinLock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
startIoCancel		–	–	–	✓	–	–	✓	✓	✓	–	✓	✓
addDevice	✓	–	–	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
lowerDriverReturn	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
TargetRelationNeedsRef		–	–	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
DoubleCompletion		✓	✓	✓	✓	✓	✓	◷	✗	✗	✓	◷	✔
PrematureSkip		–	–	✓	✓	✓	✓	✓	–	✓	✓	◷	✓
KeWaitDeadlock		–	–	✓	✓	✓	✓	✓	–	✓	✓	✓	✓
WmiComplete		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
WmiForward		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IrpProcessingComplete		✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MarkIrpPending		✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	◷	✓
PendedCompletedRequest	✓	✓	✓	✓	✗	✓	✓	◷	✓	✓	✓	◷	◷

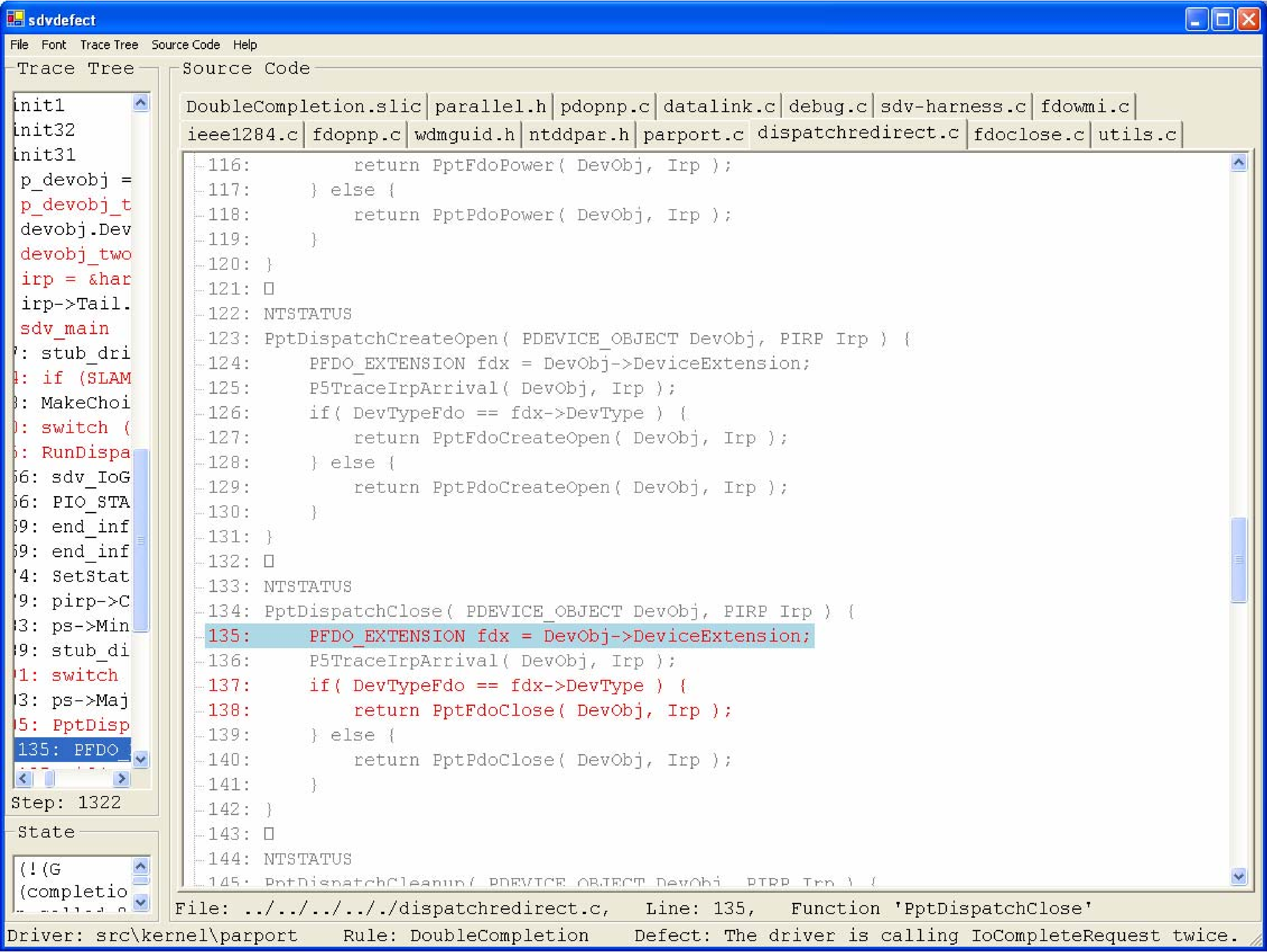
	rc/general/tracedrv/tracedrv	src/input/kbdclass	src/input/kbfiltr	src/input/mouclass	src/input/moufiltr	src/storage/fdc/fdc	src/storage/fdc/flpydisk	src/storage/filters/diskperf	src/vdd/dosioctl/krnldrv	c/wdm/1394/driver/1394diag	c/wdm/1394/driver/1394vdev	src/wdm/hid/gameenum	src/general/cancel/sys	src/general/event/sys
cancelSpinLock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
startIoCancel	–	✓	–	✓	–	–	✓	–	–	✓	✓	–	–	✓
addDevice	–	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	–	–
lowerDriverReturn	✓	✓	✓	✓	✓	✓	✓	✓	✓	◷	◷	✓	✓	✓
TargetRelationNeedsRef	–	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✔	–	–
DoubleCompletion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PrematureSkip	–	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	–	–
KeWaitDeadlock	–	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	✓	–	–
WmiComplete	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
WmiForward	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IrpProcessingComplete	✓	✓	✓	✓	✓	◷	✓	✓	✓	✓	✓	✓	✓	✓
MarkIrpPending	✓	✓	✓	✓	✓	✔	✓	✓	✓	✗	✗	✗	✓	✓
PendedCompletedRequest	✓	✓	✗	✓	✗	✓	✓	✓	✓	◷	◷	✓	✓	✓

Bebop Model Checker



```
     decl g;

     main()
     begin
       decl h;
[6]    h := !g;
[7]    A(g,h);
[8]    skip;
[9]    A(g,h);
[10]   skip;
[11]   if (g) then
[12] R:  skip;
       else
[14]     skip;
       fi
     end

     A(a1,a2)
     begin
[20]   if (a1) then
[21]     A(a2,a1);
[22]     skip;
       else
[24]     g := a2;
       fi
     end
```

```
bebop v1.0: (c) Microsoft Corporation.
Done creating bdd variables
Done building transition relations

Label R reachable by following path:

Line 12         State g=1 h=0
Line 11         State g=1 h=0
Line 10         State g=1 h=0
    Line 22     State g=1 a1=1 a2=0
        Line 24 State g=1 a1=0 a2=1
        Line 20 State g=1 a1=0 a2=1
    Line 21     State g=1 a1=1 a2=0
    Line 20     State g=1 a1=1 a2=0
Line 9          State g=1 h=0
Line 8          State g=1 h=0
    Line 22     State g=1 a1=1 a2=0
        Line 24 State g=1 a1=0 a2=1
        Line 20 State g=1 a1=0 a2=1
    Line 21     State g=1 a1=1 a2=0
    Line 20     State g=1 a1=1 a2=0
Line 7          State g=1 h=0
Line 6          State g=1
```



```
init1
init32
init31
p_devobj =
p_devobj_t
devobj.Dev
devobj_two
irp = &har
irp->Tail.
sdv_main
: stub_dri
: if (SLAM
: MakeChoi
: switch (
: RunDispa
6: sdv_IoG
6: PIO_STA
9: end_inf
9: end_inf
4: SetStat
9: pirp->C
3: ps->Min
9: stub_di
1: switch
3: ps->Maj
5: PptDisp
135: PFDO_
```

Step: 1322

```
(!(G
(completio
```

```
116:             return PptFdoPower( DevObj, Irp );
117:         } else {
118:             return PptPdoPower( DevObj, Irp );
119:         }
120: }
121: □
122: NTSTATUS
123: PptDispatchCreateOpen( PDEVICE_OBJECT DevObj, PIRP Irp ) {
124:     PFDO_EXTENSION fdx = DevObj->DeviceExtension;
125:     P5TraceIrpArrival( DevObj, Irp );
126:     if( DevTypeFdo == fdx->DevType ) {
127:         return PptFdoCreateOpen( DevObj, Irp );
128:     } else {
129:         return PptPdoCreateOpen( DevObj, Irp );
130:     }
131: }
132: □
133: NTSTATUS
134: PptDispatchClose( PDEVICE_OBJECT DevObj, PIRP Irp ) {
135:     PFDO_EXTENSION fdx = DevObj->DeviceExtension;
136:     P5TraceIrpArrival( DevObj, Irp );
137:     if( DevTypeFdo == fdx->DevType ) {
138:         return PptFdoClose( DevObj, Irp );
139:     } else {
140:         return PptPdoClose( DevObj, Irp );
141:     }
142: }
143: □
144: NTSTATUS
145: PptDispatchCleanup( PDEVICE_OBJECT DevObj, PIRP Irp ) {
```

File: ../../../../dispatchredirect.c,   Line: 135,   Function 'PptDispatchClose'

Driver: src\kernel\parport   Rule: DoubleCompletion   Defect: The driver is calling IoCompleteRequest twice.

Model Checking Challenges

- Thousands of lines of code
- Recursive procedures
- Infinite Control
- Infinite Data

- SLAM Solutions:
  - → Boolean transformation
  - → Predicate Abstraction

Boolean Programs

- C program, but only boolean variables
- Boolean variables represent finite sets of dataflow facts
- All C control-flow primitives (condition, loops, …)
- No pointers
- Procedures with call-by-value parameter passing

- Non-deterministic choice operator *

```
do {
   b = true;

   if (*){
       b = b ? false : *;
   }
} while (!b);
```

```
do {
   deviceNoOld = deviceNo;
   more = devices->Next;
   if (more){
       deviceNo++;
   }
} while (deviceNoOld != deviceNo);
```

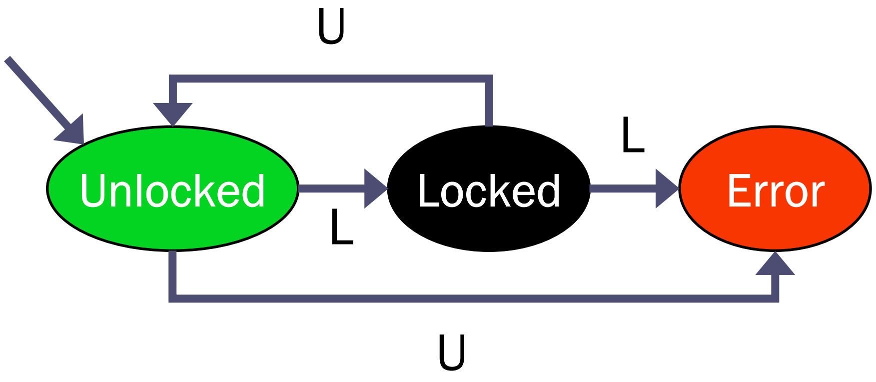
SLIC

- Low-level specification language
- Specifies temporal safety properties/rules
- Defines state machine, that monitors behavior of a C program
- Atomic propositions of a SLIC specification are boolean functions

- Suitable for expressing control-dominated properties
  - e.g. proper sequence of events
  - can encode data values inside state

Usage Rule for Locking

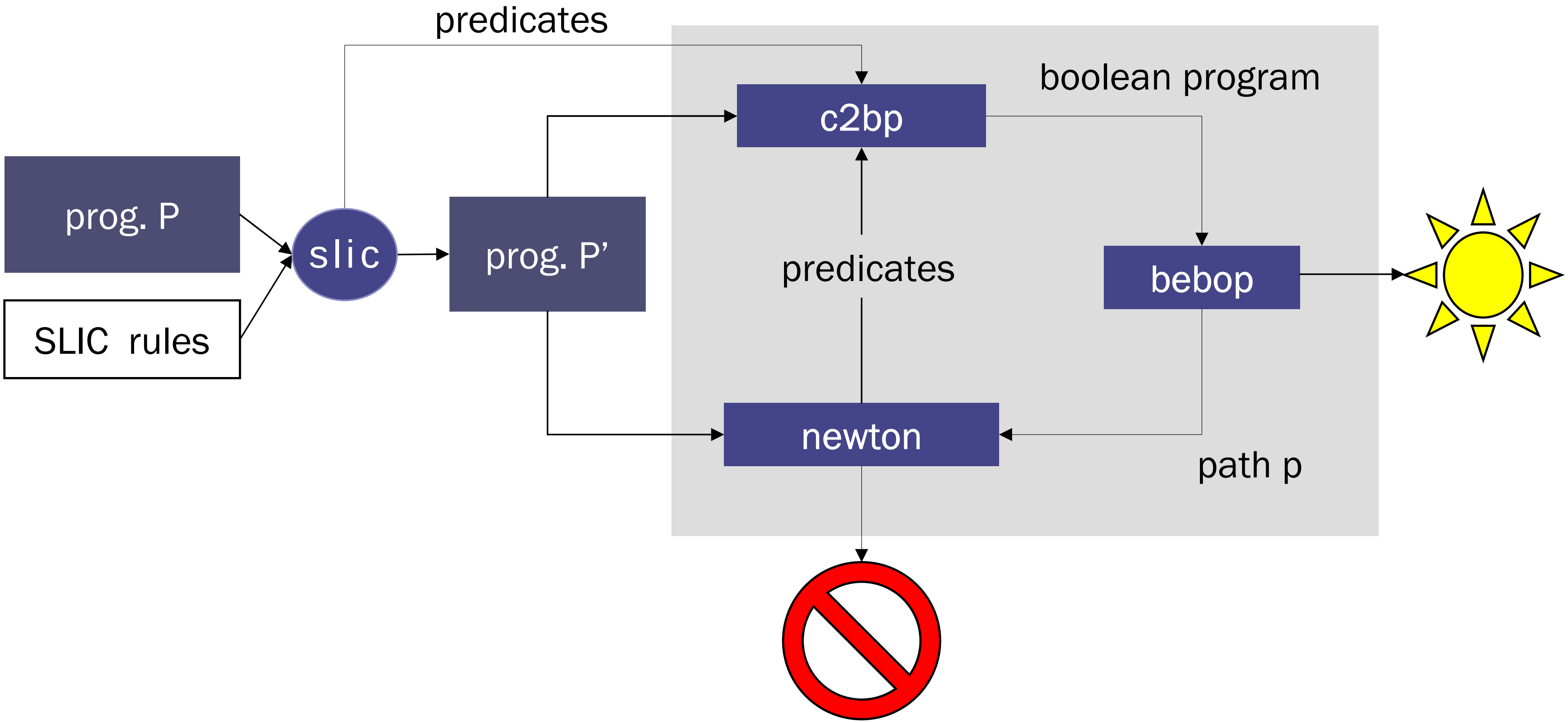
State Machine



SLIC

```
int locked = 0;

AcquireLock.call {
    if (locked==1) {
        abort;
    } else {
        locked=1;
    }
}

ReleaseLock.call {
    if (locked==0) {
        abort;
    } else {
        locked=0;
    }
}
```

The SLAM Process



Example Device Driver

API usage

Invoke SLIC rules

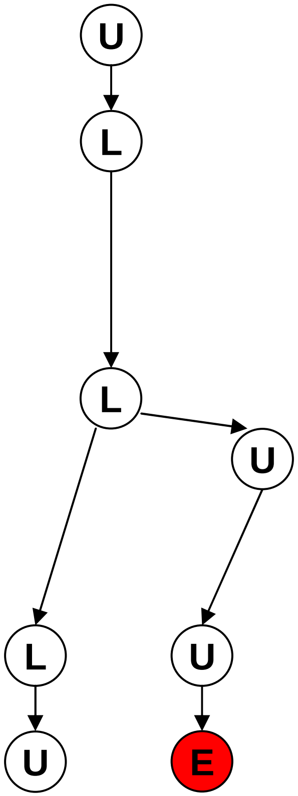
```
do {

   AcquireLock(&devices>writeListLock);
   AcquireLock.call();
   deviceNoOld = deviceNo;
   more = devices->Next;

   if (more){
       ReleaseLock(&devices->writeListLock);
       ReleaseLock.call();
       deviceNo++;
   }
} while (deviceNoOld != deviceNo);

ReleaseLock(&devices->writeListLock);
ReleaseLock.call();
```

Does this device driver violate the locking rules ?

C2bp: Boolean Abstraction

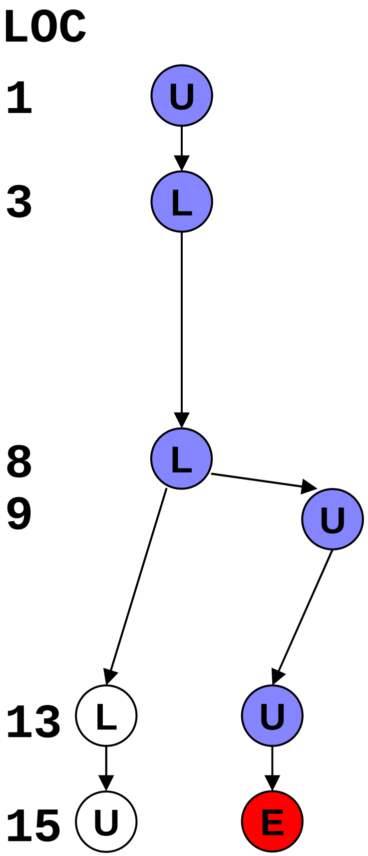
```
do {

   AcquireLock(&devices>writeListLock);

                    deviceNoOld = deviceNo;
                    more = devices->Next;

   if ( * ){        if (more){
       ReleaseLock(&devices->writeListLock);
                        ...
                        deviceNo++;
   }
} while ( * );    while (deviceNoOld != deviceNo);

ReleaseLock(&devices->writeListLock);
```

Bebop: Model Checking on P’ = BP(P,E)



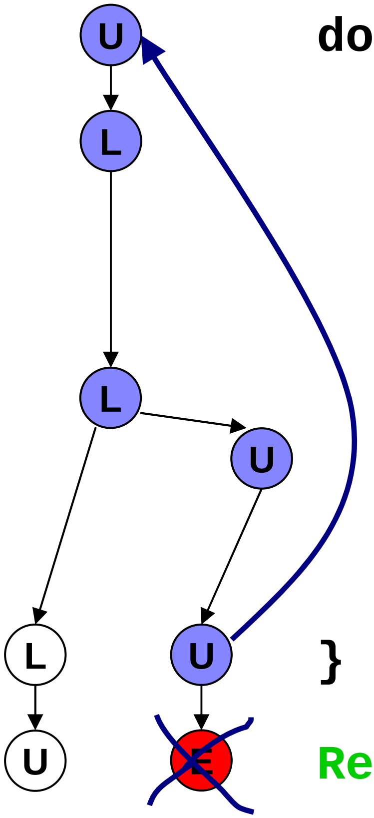
```
do {

   AcquireLock(&devices>writeListLock);




   if ( * ){
       ReleaseLock(&devices->writeListLock);


   }
} while ( * );

ReleaseLock(&devices->writeListLock);
```

Bebop: Model Checking on P’ = BP(P,E)



```
do {
   AcquireLock(&devices>writeListLock);




   if ( * ){
       ReleaseLock(&devices->writeListLock);



   }
} while ( * );

ReleaseLock(&devices->writeListLock);
```

Newton: Path Feasibility



```
do {

   AcquireLock(&devices>writeListLock);

   deviceNoOld = deviceNo;
   more = devices->Next;

   if (more){
       ReleaseLock(&devices->writeListLock);
       ...
       deviceNo++;
   }
} while (deviceNoOld != deviceNo);

ReleaseLock(&devices->writeListLock);
```

Newton: New Predicate b for P’

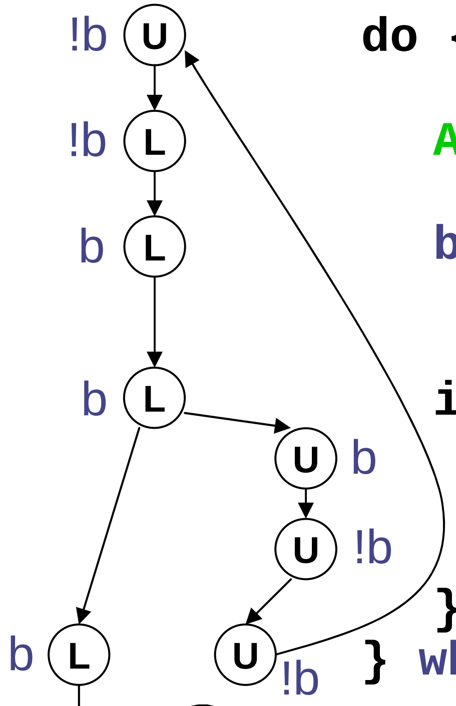
b : (deviceNoOld == deviceNo)

```
do {

   AcquireLock(&devices>writeListLock);

   deviceNoOld = deviceNo; b = true;
   more = devices->Next;

   if (more){
        ReleaseLock(&devices->writeListLock);
        ...
        deviceNo++; b = b ? false : *;
   }
} while (deviceNoOld != deviceNo); while(!b);

ReleaseLock(&devices->writeListLock);
```

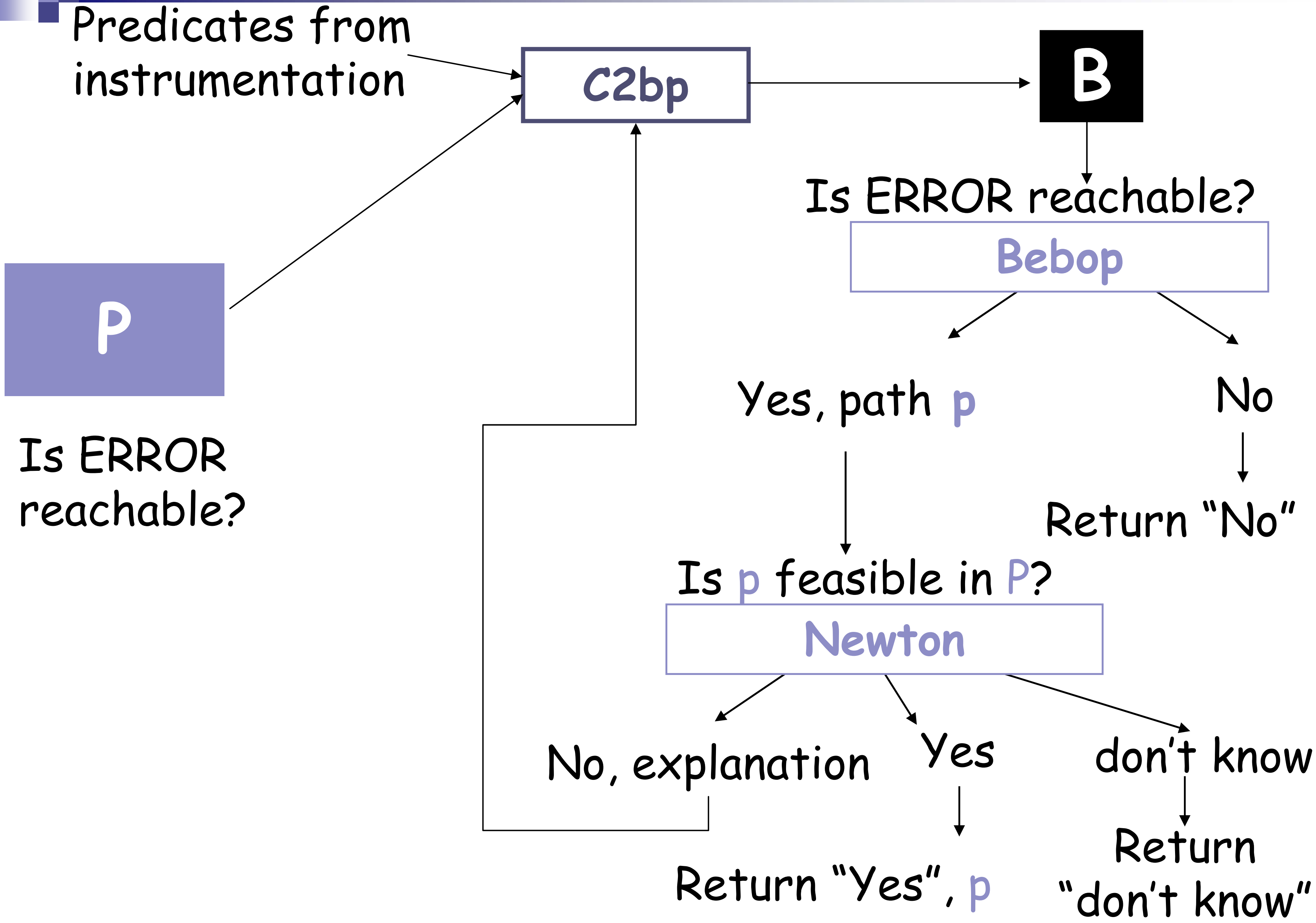
C2bp: Refined Boolean Program

```
do {

   AcquireLock(&devices>writeListLock);

   b = true;


   if (*){
       ReleaseLock(&devices->writeListLock);

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

ReleaseLock(&devices->writeListLock);
```

Bebop: Model Checking refined Program



```
do {

   AcquireLock(&devices>writeListLock);

   b = true;


   if (*){
        ReleaseLock(&devices->writeListLock);

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

ReleaseLock(&devices->writeListLock);
```



Predicates from instrumentation

C2bp

B

P

Is ERROR reachable?

Is ERROR reachable?

Bebop

Yes, path p

No

Return “No”

Is p feasible in P?

Newton

No, explanation

Yes

don’t know

Return “Yes”, p

Return “don’t know”

Complexity

- Worst-case run-time complexity of Bebop and C2bp is linear in the size of the program's control flow graph, and exponential in the number of predicates used in the abstraction.

  $O(E \times 2^{g+l})$

  E: # Edges in control flow graph

  g+l: Maximal number of global and local variables in scope

- Newton scales linearly with path length.

  O(|p|), |p|: length of path

Conclusion

- Hard to specify OS model and SLIC specifications
  - → Paper “Automatic Creation of Environment Models via Training”
- Current versions of SDV:
  - ☐ about 70% true driver errors, 30% warnings or informational errors (noise).
- >200K LOC → ~ have hour
- Full automatically
- Integrated as standard tool @ Microsoft

→ **Improves SW Quality**

Developer Comments

- “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.”

References

- **Automatic Predicate Abstraction of C Programs**, Thomas Ball, Rupak Majumdar, Todd Millstein, Sriram K. Rajamani, PLDI 2001, SIGPLAN Notices 36(5), pp. 203-213.
- **Automatic Predicate Abstraction of C Programs** (Presentation PLDI 2001)
- **Automatically Validating Temporal Safety Properties of Interfaces**, Thomas Ball, Sriram K. Rajamani, SPIN 2001, Workshop on Model Checking of Software, LNCS 2057, May 2001, pp. 103-122.
- **Automatically Validating Temporal Safety Properties of Interfaces** (Presentation SPIN 2001)
- **Bebop: A Path-sensitive Interprocedural Dataflow Engine**, Thomas Ball, Sriram K. Rajamani, PASTE 2001.
- **Bebop: A Path-sensitive Interprocedural Dataflow Engine** (Presentation PASTE 2001)
- **Bebop: A Symbolic Model Checker for Boolean Programs**, Thomas Ball, Sriram K. Rajamani, SPIN 2000 Workshop on Model Checking of Software LNCS 1885, August/September 2000, pp. 113-130.
- **Bebop: A Symbolic Model Checker for Boolean Programs** (Presentation SPIN 2000)
- **Boolean and Cartesian Abstractions for Model Checking C Programs,** Thomas Ball, Andreas Podelski, Sriram K. Rajamani, TACAS 2001, LNCS 2031, April 2001, pp. 268-283.
- **Boolean and Cartesian Abstraction for Model Checking C Programs** (Presentation TACAS 2001)
- **Boolean Programs: A Model and Process for Software Analysis,** Thomas Ball, Sriram K. Rajamani, MSR Technical Report 2000-14.
- **Checking Temporal Properties of Software with Boolean Programs**, Thomas Ball, Sriram K. Rajamani, Workshop on Advances in Verification (with CAV 2000).
- **From Symptom to Cause: Localizing Errors in Counterexample Traces**, Thomas Ball, Mayur Naik, Sriram Rajamani (POPL 2003).
- **From Symptom to Cause: Localizing Errors in Counterexample Traces** (Presentation POPL 2003)
- **Generating Abstract Explanations of Spurious Counterexamples in C Programs,** Thomas Ball, Sriram K. Rajamani, MSR-TR-2002-09.
- **Parameterized Verification of Multithreaded Software Libraries**, Thomas Ball, Sagar Chaki, Sriram K. Rajamani, TACAS 2001, LNCS 2031, April 2001, pp. 158-173.
- **Parameterized Verification of Thread-safe Libraries** (Presentation TACAS 2001)
- **Polymorphic Predicate Abstraction**, Thomas Ball, Todd Millstein, Sriram K. Rajamani, MSR-TR-2001-10.
- **Refining Approximations in Software Predicate Abstraction,** Thomas Ball, Byron Cook, Satyaki Das, Sriram K. Rajamani. TACAS 2004.
- **Relative Completeness of Abstraction Refinement for Software Model Checking,** Thomas Ball, Andreas Podelski, Sriram K. Rajamani, TACAS 2002, LNCS 2280, April 2002, pp. 158-172.
- **Secrets of Software Model Checking** (Presentation SAS 2002)
- **SLIC: A Specification Language for Interface Checking (of C)**, Thomas Ball, Sriram K. Rajamani, MSR-TR-2001-21.
- **Specifying and Checking Properties of Software** (Presentation CSTB's July 2001 Symposium on Fundamentals of Computer Science, July 2001)
- **Speeding Up Dataflow Analysis Using Flow-Insensitive Pointer Analysis**, Stephen Adams, Thomas Ball, Manuvir Das, Sorin Lerner, Sriram K. Rajamani, Mark Seigle, Westley Weimer. SAS 2002, LNCS 2477.
- **The SLAM Project: Debugging System Software via Static Analysis**, Thomas Ball, Sriram K. Rajamani, POPL 2002, January 2002, pages 1-3.
- **The SLAM Project: Debugging System Software via Static Analysis** (Presentation POPL 2002)
- **The SLAM Toolkit**, Thomas Ball, Sriram K. Rajamani, CAV 2001.