SDVRP and SLAM-2

Static Driver Verifier Research Platform & SLAM-2 Analysis Engine
Outline

• Introduction
• SDV Motivation
• Model vs Real Implementation
• SLIC
• SDVRP
• SLAM-2
• Comparisons
• Conclusions
SDV Research Platform

• Academic release of SDV (Static Driver Verifier), based on the code that ships with Windows 7 WDK (Windows Driver Kit)
• Developed in Microsoft in years 2001-2006.
• Write custom models for device and custom API rules
• Apply SDV to verify modules written in C that use the APIs
• Based on the new, robust SLAM-2 engine
Importance of Driver API 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
Motivation

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

- Module exposes a set of entry points that the Platform may call with requests.
- During processing a request the Module may call Platform API.
- In WDK, the Platform represents OS, while the Module represents a driver.

Platform

Platform Manager

DriverCreate()  DriverWrite()

Module

AcquireLock()  ReleaseLock()

Platform API
Complexity of **Module** and **Platform**

- Platform might contain millions lines of code.
- Checking that Module obeys API usage is easier than checking that the Module fulfills its intended feature.
Platform Model

- Platform Model (in C) allows verification flow control and introduces non-determinism.
Checking Module and Platform Model

Platform Model

Platform Manager Model

DriverCreate()  DriverWrite()

Module

AcquireLock()  ReleaseLock()

Platform API Models

*.slic

Add your rule here
Example for Driver

```c
state {
    enum {unlocked, locked} s = unlocked;
}

RunDispatchFunction.exit {
    if (s != unlocked) abort "...";
}

KeAcquireSpinLock.entry {
    if (s != unlocked) abort "...";
    else s = locked;
}

KeReleaseSpinLock.entry {
    if (s != locked) abort "...";
    else s = unlocked;
}
```
SLIC (a Specification Language for Interface Checking)

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

• Specifications are like programs
• It is hard to get them right the first time
• They evolve just like programs
• Tools need to tie specifications to programs
• You can hire people to write them!
Plugin for Static Verification

- The analysis engine of SDVRP (SLAM-2) verifies whether the Module adheres to the plugin API usage rules, in the context of the plugin platform model.

✓ Thus SDVRP may be applied to many other pieces of code besides device drivers.

- WDM contains plugins for standard driver platforms.
Module’s Source Code in C

API Usage Rules (SLIC)

Platform model

Rules

Defects

100% path coverage

SDVRP

Static Driver Verifier

Plugin
SLAM-2

- SLAM-2 is SDVRP’s underlying analysis engine
- SLAM innovations
  - Boolean programs: a new model for software
  - Model creation (c2bp)
  - Model checking (bebop)
  - Model refinement (newton)
SLAM Process
Boolean Programs

- An interesting innovative approach of SLAM.
- Permit declaration of Boolean variables only.
- Have the same control flow as the original C program.

- Invariant checking, reachability and termination are decidable for the Boolean programs.
Bebop

- Bebop is SLAM’s model checker for the Boolean programs.

✓ SDVRP allows a user to substitute other Boolean model checker in place of Bebop
Example: Source Code of Module

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

KeReleaseSpinLock();
```

Does this code obey the locking rules?
Example: Boolean Program (v.1)

do {
    KeAcquireSpinLock();
    skip;
    if (*) {
        skip;
        KeReleaseSpinLock();
        skip;
    }
} while (*);

KeReleaseSpinLock();
Example: Feasibility Check by Newton

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

KeReleaseSpinLock();
```

Is the error path feasible in C program?
Example: Introducing a Predicate

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

KeReleaseSpinLock();

Add a new predicate to the Boolean program:

\[ b : nPacketsOld = nPackets \]
Example: Introducing a Predicate

```
Example: Introducing a Predicate

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

KeReleaseSpinLock();
```
Example: Refined Boolean Program (v.2)

do {
    KeAcquireSpinLock();
    b = true;
    if (*) {
        skip;
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b)

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
b = true;
if (*) {
    skip;
    KeReleaseSpinLock();
b = b ? false : *;
}
} while (!b)

KeReleaseSpinLock();
SDV GUI

```c
: sdv_main
42: int choice = SdvMakeChoice();
44: sdv_stub_custom_main_begin
46: switch (choice) {
50: sdv_stub_custom_read_begin
51: DriverRead
    39: CustomAcquireLock
39: SLIC_CustomAcquireLock_exit
42: CustomUnlock
43: if (status==CUSTOM_STATUS_UNSUCCESSFUL)
45: return CUSTOM_STATUS_UNSUCCESSFUL;
45: return
52: sdv_stub_custom_read_end
73: sdv_stub_custom_main_end

Source Code

customlock.slic  sdv-harness.c  fail_driver1.c

26: {
27:    CUSTOM_LOCK Lock;
28:    int reads;
29:    int writes;
30:    char buffer[512];
31: } DriverData;
32:
33: CUSTOM_STATUS
34: DriverRead{
35:    PCUSTOM_IRP Irp
36:    }
37:
38: CUSTOM_STATUS status;
39: CustomAcquireLock(&DriverData.Lock);
40: /* Left out: Somehow magically read DriverData.by
41:  status=CustomMemMove(Irp->buffer, DriverData.buffer
42:  if (status==CUSTOM_STATUS_UNSUCCESSFUL)
43:  {
45: return CUSTOM_STATUS_UNSUCCESSFUL;
46: }
47: DriverData.reads++;
48: CustomReleaseLock(&DriverData.Lock);
49: return CUSTOM_STATUS_SUCCESS;
50: }
51:
52: CUSTOM_STATUS
53: DriverWrite{
54:    PCUSTOM_IRP Irp
55: }
```

File: fail_driver1.c  Line: 45  Function 'DriverRead'

The driver has returned from an entry point without releasing the lock.
Remarks about SLAM

- Automatic discovery of invariants
  - driven by property and a finite set of (false) execution paths
  - abstraction + model checking computes inductive invariants (Boolean combinations of observations)

- A hybrid static analysis
  - c2bp+bebop explore all paths through abstraction
  - newton executes path through C code symbolically

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

- Boolean program model has proved itself
- Successful for domain of device drivers
  - control-dominated safety properties
  - few Boolean variables required to prove or find real counterexamples
- Counterexample-driven refinement
  - terminates in practice
  - struggles incompleteness of a theorem prover
Challenges

• Abstracting
  • from a language with pointers (C)
  • to one without pointers (Boolean programs)
• Z3 solver in SLAM-2 can efficiently evaluate the pointer predicates
SLAM-2 Improvements vs SLAM

Improvements include

- Boolean abstraction on basic blocks
- Error Trace validation: combination of forward and backwards symbolic execution
- Uses Z3, new axiomatization of pointers
- Optimized predicate discovery
- Easier use for non-driver related models.
SLAM-2 Performance vs SLAM

SLAM 2.0 released with SDV 2.0, part of Windows 7 WDK

<table>
<thead>
<tr>
<th>Parameter for WDM drivers</th>
<th>SDV 2.0 (SLAM2)</th>
<th>SDV 1.6 (SLAM1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False defects</td>
<td>0.4% (2/512)</td>
<td>19.7% (31/157)</td>
</tr>
<tr>
<td>Give-up results</td>
<td>3.2% (187/5727)</td>
<td>6% (285/4692)</td>
</tr>
</tbody>
</table>
Static Driver Verifier (SDV):

- Compile-time verification tool
- Supports standard driver
- Uses SLAM-2 as the verification engine
  - Based on CEGAR loop
  - Boolean abstraction of input C programs
- API-specific components:
  - platform model
  - API rules in SLIC language
SDVRP vs Alloy

• Has a language for defining specifications like in Alloy.
SDVRP vs Java Pathfinder

• Unlike Java Pathfinder, SLAM-2 is a static instrumentation.
• Both are incomplete.
• SLAM-2 works with C files with problems related to a C code (e.g. pointers).
• SLAM-2 always terminates.
SDVRP vs JML&ESC/Java2

- Like ESC, SLAM executes a static analysis.
- Like ESC, SLAM performs symbolic execution.
- Like ESC, May analysis.
- Both are highly customizable.
- Unlike JML, SLIC defines specifications in a stand-alone file.
- SLAM-2 works with C files with problems related to a C code (e.g. pointers).
SDVRP vs CBMC

• Both tools can verify C code.
• SDVRP does not support C++ code.
• Unlike CBMC, SLIC defines specifications in a standalone file.
• Unlike CBMC, SDVRP is highly customizable.
SDVRP vs BLAST

- Both based on CEGAR loop.
- Both apply lazy analysis.
- Both support C programs.
- Both mainly target verification of driver programs.
- BLAST checks reachability, while SDVRP defines a modeling language.
- SDVRP sends a Boolean program to the underlying model checker.
Conclusion

• Programmer provides redundant partial specifications
• Code is automatically checked for consistency
• Different from proving whole program correctness
  • Specifications are not complete
• SDVRP toolkit for customizable verification of client code against API rules
• SLAM-2 is SDVRP’s analysis engine