Model Checking

Bernd Schoeller

Zurich, January 23rd, 2005
We don’t want ...

A fatal exception 0E has occurred at 0028:C0011E36 in UXD UMM(01) + 00010E36. The current application will be terminated.

* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue _
Did you know?

Microsoft does not like blue screens, too!
On Blue Screens

- The majority of blue screens are caused by 3\textsuperscript{rd} party software
- Most of this software is device drivers
  - Complex software (concurrency, race conditions, lock keeping)
  - Running “unprotected” by the OS
  - Written for top performance
  - Written by non-software-engineers
  - Difficult to debug
Overview

- What is Model Checking?
- The SLAM project
- SAT solving
- Bounded Model Checking
Model Checking

Does a program $P$ satisfy a certain property $Q$?

- Proving is difficult
- Testing is not complete
Models Checking: Lets test every possible input

(this works for hardware !)
But:

We just have too many states (*state space explosion*)

```pascal
positive_max (a,b : INTEGER): INTEGER is
  require
    a_positive: a >= 0
    b_positive: b >= 0
  do
    if a > b then Result := a else Result := b end
  ensure
    result_positive: Result >= 0
  end

has got $2^{64} = 18,446,744,073,709,551,616$ different inputs
```
Boolean Abstraction

Let replace every $x \geq 0$ by $\text{POS}_x$

positive\_max (POS\_a,POS\_b : BOOLEAN): BOOLEAN is
  require
    a\_positive: POS\_a
    b\_positive: POS\_b
  do
    if ? then POS\_Result := POS\_a
             else POS\_Result := POS\_b end
  ensure
    result\_positive: POS\_Result
  end

How many possible input do we have now?
SLAM

- Model Checker for C device drivers
- Looks for the possible violation of temporal properties
  - Properties describe well-known mistakes in driver development
- Uses Boolean abstraction
- Part of the Windows Driver Foundation
The SLAM process

C-Code → Boolean Program Generator
   C2BP → Boolean Program
   Bug found → Predicate Discoverer
   NEWTON → Error Path
   Error Path → Check successful?
   No → New workflow
   Yes → Code correct

Model Checker for Boolean Programs
   BEBOP
SLAM specification

```c
enum { Unlocked = 0, Locked = 1 }
    state = Unlocked;

void slic_abort() {
    SLIC_ERROR: ;
}

void KeAquireSpinLock_return ()
    if (state == Locked)
        abort;
    else
        state = Locked;
}

void KeReleaseSpinLock_return {
    if (state == Unlocked)
        slic_abort();
    else
        state = Unlocked;
}
```

**Formal Specification**

**Instrumented C code**
void example() {
    do {
        KeAcquireSpinLock();

        nPacketsOld = nPackets;
        req = devExt->WLHV;
        if (req && req->status) {
            devExt->WLHV = req->Next;
            KeReleaseSpinLock();

            irp = req->irp;
            if (req->status > 0) {
                irp->IoS.Status = SUCCESS;
                irp->IoS.Info = req->Status;
            } else {
                irp->IoS.Status = FAIL;
                irp->IoS.Info = req->Status;
            }
            SmartDevFreeBlock(req);
            IoCompleteRequest(irp);
            nPackets++;
        }
    } while (nPackets != nPacketsOld);
    KeReleaseSpinLock();
}

SLAM example
void example() {
    do {
        KeAcquireSpinLock();
        KeAcquireSpinLock_return();
        nPacketsOld = nPackets;
        req = devExt->WLHV;
        if (req && req->status) {
            devExt->WLHV = req->Next;
            KeReleaseSpinLock();
            KeReleaseSpinLock_return();
            irp = req->irp;
            if (req->status > 0) {
                irp->IoS.Status = SUCCESS;
                irp->IoS.Info = req->Status;
            } else {
                irp->IoS.Status = FAIL;
                irp->IoS.Info = req->Status;
            }
            SmartDevFreeBlock(req);
            IoCompleteRequest(irp);
            nPackets++;
        }
    } while (nPackets != nPacketsOld);
    KeReleaseSpinLock();
    KeReleaseSpinLock_return();
}
Let \( l \): state == Locked

Let \( u \): state == Unlocked

```c
void KeAquireSpinLock_return () {
    if (l)
        slic_abort();
    else
        l,u := T,F;
}

void KeReleaseSpinLock_return {
    if (u)
        slic_abort();
    else
        l,u := F,T;
}
```
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        skip;
        skip;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            skip;
            if (*) {
                skip;
                skip;
            } else {
                skip;
                skip;
            }
            skip;
            skip;
            skip;
        }
    } while (*);
    skip;
    KeReleaseSpinLock_return();
}

Will we reach "abort"?
SLAM example (cont.)

We need another Boolean predicate:

Let \( b: \text{nPackets} == \text{nPacketsOld} \)
void example() {
    do {
        skip;
        KeAcquireSpinLock_return();
        b := T;
        skip;
        if (*) {
            skip;
            skip;
            KeReleaseSpinLock_return();
            skip;
            if (*) {
                skip;
                skip;
            } else {
                skip;
                skip;
            }
        } else {
            skip;
            skip;
        }
        skip;
        skip;
        b := choose (F,b);
    } while (!b);
    skip;
    KeReleaseSpinLock_return();
}
Boolean Formulas?

Given a Boolean formula, is there an assignment for all variables with TRUE or FALSE that will make the formula true?

Like: \((b \text{ or } T) \implies ((a \implies F) \text{ and } (b \text{ or } a))\)

SAT (Satisfiability) is NP-complete
SAT solving

Question: How difficult are NP-complete problems?
Question: How difficult are NP-complete problems?

Answer: Very difficult
SAT solving

Question: How difficult are NP-complete problems?

Answer: Very difficult, but who cares?

Modern SAT solvers can do formulas with thousand of Boolean variables and in the size of a couple of megabytes.
ZChaff

- SAT solver developed at Princeton University
- One of the fastest provers around
- Problems with millions of variables, with tens of million clauses
Limmat

- Developed by Prof. Biere (now at Linz, Austria)
- [http://fmv.jku.at/software/](http://fmv.jku.at/software/)
- Won a couple of competitions
- Now replaced by Quantor
Bounded Model Checking

Knowing about the power of SAT solvers, can we do more than Boolean abstractions?

- Relations and Graphs
- Bounded to a maximum size of the graph
- Most problems with graph algorithms show up with small graphs
- Question: Where are the Booleans for SAT?!
Application of SAT solvers

With SAT solvers, we are able to analyze complex Boolean properties

Example: Alloy

Developed by: Daniel Jackson

http://people.csail.mit.edu/dnj/

Model Checker for Graphs
Alloy

Life Demo