Software Model Checking

Xiangyu Zhang
Symbolic Software Model Checking

- Symbolic analysis explicitly explores individual paths, encodes and resolves path conditions.
- Model checking directly encodes both the program and the property to check to constraints.
A (very) simple example (1)

Program

```c
int x;
int y=8, z=0, w=0;
if (x)
    z = y - 1;
else
    w = y + 1;
assert (z == 7 ||
         w == 9)
```

Constraints

```c
y = 8,
z = x ? y - 1 : 0,
w = x ? 0 : y + 1,
z != 7,
w != 9
```

UNSAT
no counterexample
assertion always holds!
A (very) simple example (2)

Program

```c
int x;
int y=8, z=0, w=0;
if (x)
    z = y - 1;
else
    w = y + 1;
assert (z == 5 ||
       w == 9)
```

Constraints

```c
y = 8,
z = x ? y - 1 : 0,
w = x ? 0 : y + 1,
z != 5,
w != 9
```

SAT

counterexample found!

`y = 8, x = 1, w = 0, z = 7`
Procedure

Unroll loops

Program

Claim

Bound (n)

Analysis Engine

CNF

SMT Solver

SAT
counterexample exists

UNSAT
no counterexample of bound n

Translate to SSA form

SSA to SMT constraints
Loop Unwinding

• All loops are unwound
  • can use different unwinding bounds for different loops
  • to check whether unwinding is sufficient special “unwinding assertion” claims are added

• If a program satisfies all of its claims and all unwinding assertions then it is correct!

• Same for backward goto jumps and recursive functions
Loop Unwinding

- while() loops are unwound iteratively
- Break / continue replaced by goto

```c
void f(...) {
    ...
    while(cond) {
        Body;
    }
    Remainder;
}
```
Loop Unwinding

- while() loops are unwound iteratively
- Break / continue replaced by goto

```c
void f(...) {
    ...if(cond) {
        Body;
        while(cond) {
            Body;
        }
    }
    Remainder;
}
```
Loop Unwinding

- while() loops are unwound iteratively
- Break / continue replaced by goto

```c
void f(...) {
    ...
    if(cond) {
        Body;
        if(cond) {
            Body;
            while(cond) {
                Body;
            }
        }
    }
    Remainder;
}
```
Unwinding assertion

- while() loops are unwound iteratively
- Break / continue replaced by goto
- Assertion inserted after last iteration: violated if program runs longer than bound permits

```c
void f(...) {
    ...if(cond) {
        Body;
        if(cond) {
            Body;
            if(cond) {
                Body;
                while(cond) {
                    Body;
                }
            }
        }
    }
    Remainder;
}
```
- while() loops are unwound iteratively
- Break / continue replaced by goto
- Assertion inserted after last iteration: violated if program runs longer than bound permits

```c
void f(...) {
  ... if(cond) {
    Body;
    if(cond) {
      Body;
      if(cond) {
        Body;
        assert(!cond);
      }
    }
  }
  }
  Remainder;
}
```
Example: Sufficient Loop Unwinding

```c
void f(...) {
    j = 1
    while (j <= 2)
        j = j + 1;
    Remainder;
}

–unwind = 3
```

```c
void f(...) {
    j = 1
    if(j <= 2) {
        j = j + 1;
        if(j <= 2) {
            j = j + 1;
            if(j <= 2) {
                j = j + 1;
                assert(!(j <= 2));
            }
        }
    }
    Remainder;
}
```
Example: Insufficient Loop Unwinding

```c
void f(...) {
    j = 1
    while (j <= 10)
        j = j + 1;
    Remainder;
}

–unwind = 3
```

```c
void f(...) {
    j = 1
    if(j <= 10) {
        j = j + 1;
        if(j <= 10) {
            j = j + 1;
            if(j <= 10) {
                j = j + 1;
                assert(!(j <= 10));
            }
        }
    }
    Remainder;
}
```
Easy to transform when every variable is only assigned once!
Transforming Loop-Free Programs Into Equations (2)

When a variable is assigned multiple times, use a new variable for the RHS of each assignment

Program

\[
\begin{align*}
x &= x + y; \\
x &= x \times 2; \\
a[i] &= 100;
\end{align*}
\]

SSA Program

\[
\begin{align*}
x_1 &= x_0 + y_0; \\
x_2 &= x_1 \times 2; \\
a_1[i_0] &= 100;
\end{align*}
\]
What about conditionals?

Program

```plaintext
if (v)
    x = y;
else
    x = z;
w = x;
```

SSA Program

```plaintext
if (v0)
    x0 = y0;
else
    x1 = z0;
w1 = x0;
```

What should ‘x’ be?
What about conditionals?

For each join point, add new variables with selectors
Encoding

- Declare symbolic variables for each (SSA) scalar variables
- Assignments to equivalence
- Phi functions to ITE expressions
- Array accesses to select/store operations
- Scalar pointer dereferences to identify operations
- Heap dereferences to select/store operations

```c
if (v)
  p = &x;
else
  p = &y;

*p = 10;
q = p+i
z = *q
```

```c
p = (int*) malloc(100);
i = 10;
q = p+i
*q = 10
```
CBMC: C Bounded Model Checker

- Developed at CMU by Daniel Kroening et al.
- Available at: http://www.cs.cmu.edu/~modelcheck/cbmc/
- Supported platforms: Windows (requires VisualStudio’s CL), Linux
- Provides a command line and Eclipse-based interfaces
- Known to scale to programs with over 30K LOC
- Was used to find previously unknown bugs in MS Windows device drivers
Explicit State Model Checking

The program is indeed executing

- `jpf <your class> <parameters>`
  - Very similar to "java <your class> <parameters>
- Execute in a way that all possible scenarios are explored
  - Thread interleaving
  - Undeterministic values (random values)
- **Concrete input is provided**
- A state is indeed a concrete state, consisting of
  - Concrete values in heap/stack memory
import java.util.Random;

public class Rand {
    public static void main (String[] args) {
        Random random = new Random(42);  // (1)
        int a = random.nextInt(2);      // (2)
        System.out.println("a=" + a);

        //... lots of code here
        int b = random.nextInt(3);      // (3)
        System.out.println("  b=" + b);
        int c = a/(b+a-2);              // (4)
        System.out.println("    c=" + c);
    }
}

> java Rand
a=1
    b=0
        c=-1
>
An Example (cont.)

- One execution corresponds to one path.

```
1. Random random = new Random();
2. int a = random.nextInt(2);
3. int b = random.nextInt(3);
4. int c = a/(b+a-2);
```
> bin/jpf Rand

JavaPathfinder v4.1 - (C) 1999-2007 RIACS/NASA Ames Research Center
system under test
application: /Users/pcmehlitz/tmp/Rand.java

search started: 5/23/07 11:48 PM
a=1
  b=0
    c=-1

results
no errors detected

search finished: 5/23/07 11:48 PM
>
> bin/jpf +vm.enumerate_random=true Rand
JavaPathfinder v4.1 - (C) 1999-2007 RIACS/NASA Ames Research Center
==================================== system under test
application: /Users/pcmehlitz/tmp/Rand.java

==================================== search started: 5/23/07 11:49 PM
a=0
  b=0
    c=0
  b=1
    c=0
  b=2

==================================== error #1
gov.nasa.jpf.jvm.NoUncausedExceptionsProperty
java.lang.ArithmeticException: division by zero
    at Rand.main(Rand.java:15)
....
>
JPF explores multiple possible executions GIVEN THE SAME CONCRETE INPUT
Two Essential Capabilities

**Backtracking**
- Means that JPF can restore previous execution states, to see if there are unexplored choices left.
  - While this is theoretically can be achieved by re-executing the program from the beginning, backtracking is a much more efficient mechanism if state storage is optimized.

**State matching**
- JPF checks every new state if it already has seen an equal one, in which case there is no use to continue along the current execution path, and JPF can backtrack to the nearest non-explored non-deterministic choice
  - Heap and thread-stack snapshots.
State Abstraction

- Eliminate details irrelevant to the property

- Obtain simple finite models sufficient to verify the property

Disadvantage

- Loss of Precision: False positives/negatives
Data Abstraction

Abstraction Function  \( h : \text{from } S \text{ to } S' \)
Abstraction proceeds component-wise, where variables are components

- **x**: int, values ..., -2, 0, 2, 4, ...
  - Even
  - Odd

- **y**: int, values ..., -3, -2, -1, 0, 1, 2, 3, ...
  - Neg, Zero, Pos
How do we Abstract Behaviors?

Abstract domain $A$

- Abstract concrete values to those in $A$

Then compute transitions in the abstract domain
Data Type Abstraction

Code

```java
int x = 0;
if (x == 0)
    x = x + 1;
```

Abstract Data domain

```
int
( n<0) : NEG
( n==0): ZERO
( n>0) : POS
```

```
Signs x = ZERO;
if (Signs.eq(x,ZERO))
    x = Signs.add(x,POS);
```
Existential/Universal Abstractions

Existential

- Make a transition from an abstract state if at least one corresponding concrete state has the transition.
- Abstract model $M'$ simulates concrete model $M$

Universal

- Make a transition from an abstract state if all the corresponding concrete states have the transition.
Universal Abstraction (Under-Approximation)
Guarantees from Abstraction

Assume $M'$ is an abstraction of $M$

- **Strong Preservation:**
  - $P$ holds in $M'$ iff $P$ holds in $M$

- **Weak Preservation:**
  - $P$ holds in $M'$ implies $P$ holds in $M$
Guarantees from Exist. Abstraction

Let $\varphi$ be a *hold-for-all-paths* property

$M'$ existentially abstracts $M$

**Preservation Theorem**

\[ M' \models \varphi \implies M \models \varphi \]

Converse does not hold

\[ M' \not\models \varphi \not\implies M \not\models \varphi \]

$M' \not\models \varphi$: counterexample may be spurious
Guarantees from Univ. Abstraction

Let $\varphi$ be a existential-quantified property and $M$ simulates $M'$

**Preservation Theorem**

$$M' \models \varphi \rightarrow M \models \varphi$$

Converse does not hold

$$M \not\models \varphi \rightarrow M' \not\models \varphi$$
Spurious counterexample in Overapproximation
Problem: Deadend and Bad States are in the same abstract state.

Solution: Refine abstraction function.

The sets of Deadend and Bad states should be separated into different abstract states.
Refinement: $h'$
Automated Abstraction/Refinement

- **Good abstractions are hard to obtain**
  - Automate both Abstraction and Refinement processes

- **Counterexample-Guided AR (CEGAR)**
  - Build an abstract model $M'$
  - Model check property $P$, $M' \models P$?
  - If $M' \models P$, then $M \models P$ by Preservation Theorem
  - Otherwise, check if Counterexample (CE) is spurious
  - Refine abstract state space using CE analysis results
  - Repeat
Counterexample-Guided Abstraction-Refinement (CEGAR)

1. Build New Abstract Model
2. Obtain Refinement Cue
3. Check Counterexample
   - Spurious CE
   - Real CE
4. Model Check
   - Pass
     - No Bug
   - Fail
     - Bug
Predicate Abstraction

- Extract a finite state model from an infinite state system
- Used to prove assertions or safety properties
- Successfully applied for verification of C programs
  - SLAM (used in windows device driver verification)
  - MAGIC, BLAST, F-Soft
Example for Predicate Abstraction

int main() {
    int i;
    i=0;
    while(even(i))
        i++;
}

void main() {
    bool p1, p2;
    p1=TRUE;
    p2=TRUE;
    while(p2)
    {
        p1=p1?FALSE:nondet();
        p2=!p2;
    }
}

Predicates:

\[ p_1 \Leftrightarrow i=0 \]
\[ p_2 \Leftrightarrow \text{even}(i) \]

C program

[Ball, Rajamani '01]

Boolean program

[Graf, Saidi '97]

[Ball, Rajamani '01]
Computing Predicate Abstraction

- How to get predicates for checking a given property?
- How do we compute the abstraction?
- Predicate Abstraction is an over-approximation
  - How to refine coarse abstractions
Counterexample Guided Abstraction Refinement loop

**C Program**

- Initial Abstraction
- Abstract model
- Refinement
- Simulator
- Verification
- Model Checker
- No error or bug found
- Property holds
- Simulation successful
- Bug found
- Spurious counterexample
Example ( ) {
1:   do{
    lock();
    old = new;
    q = q->next;
2:     if (q != NULL){
3:         q->data = new;
       unlock();
        new ++;
    }
4:   } while (new != old);
5:   unlock ();
    return;
}
What a program really is...

Example ( ) {
    1: do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            2: q->data = new;
                unlock();
                new ++;
        }  
    } while(new != old);
    4: } while(new != old);
    5: unlock (old);
    return;
}
The Safety Verification Problem

Is there a path from an initial to an error state?

Problem: Infinite state graph

Solution: Set of states' logical formula
Idea 1: Predicate Abstraction

- **Predicates** on program state:
  - `lock`
  - `old = new`

- States satisfying **same** predicates are **equivalent**
  - Merged into one abstract state

- **#abstract states is finite**
Abstract States and Transitions

State

```
3: unlock();
new++;
4: }
```

```
pc  \rightarrow  3
lock \rightarrow  \bullet
old  \rightarrow  5
new  \rightarrow  5
q    \rightarrow  0x133a
```

```
lock
old=new
```

```
! lock
! old=new
```

pc  \rightarrow  4
lock \rightarrow  \bullet
old  \rightarrow  5
new  \rightarrow  6
q    \rightarrow  0x133a
Existential Approximation

Abstraction

State

3: unlock();
   new++;
4: } ...

pc \mapsto 3
lock \mapsto 
old \mapsto 5
new \mapsto 5
q \mapsto 0x133a

pc \mapsto 4
lock \mapsto 
old \mapsto 5
new \mapsto 6
q \mapsto 0x133a

lock
old=new

! lock
! old=new
Abstraction

State

3: unlock();
    new++;
Analyze Abstraction

Analyze finite graph

Over Approximate:
Safe => System Safe

Problem
Spurious counterexamples
Idea 2: Counterex.-Guided Refinement

Solution
Use spurious counterexamples to refine abstraction!
Idea 2: Counterex.-Guided Refinement

Solution
Use spurious counterexamples to refine abstraction

1. Add predicates to distinguish states across cut
2. Build refined abstraction
Iterative Abstraction-Refinement

Solution

Use spurious counterexamples to refine abstraction

1. Add predicates to distinguish states across cut
2. Build refined abstraction - eliminates counterexample
3. Repeat search Till real counterexample or system proved safe

[Kurshan et al 93] [Clarke et al 00] [Ball-Rajamani 01]
Example () {
1: do{
    lock();
    old = new;
    q = q->next;
2:   if (q != NULL){
3:     q->data = new;
        unlock();
        new ++;
    }
4:}while(new != old);
5: unlock();
}

Predicates: LOCK
Example ( ) {
  do{
    lock();
    old = new;
    q = q->next;
    if (q != NULL){
      q->data = new;
      unlock();
      new ++;
    }
  }while(new != old);
  unlock();
}
Example ( ) {
    do{
        lock();
        old = new;
        q = q->next;
        if (q != NULL){
            q->data = new;
            unlock();
            new ++;
        }
    }while(new != old);
    unlock();
}
Example () {
  1: do{
      lock();
      old = new;
      q = q->next;
  2:   if (q != NULL){
  3:     q->data = new;
      unlock();
      new ++;
  4:   }
  5: while(new != old);
  6: unlock ();
}

Predicates: LOCK
Example ( ) {
1:  do{
    lock();
    old = new;
    q = q->next;
2:    if (q != NULL){
3:      q->data = new;
       unlock();
      new ++;
    }
} while (new != old);
5:  unlock();
}

Predicates: LOCK
Example () {
  1:   do{
       lock();
       old = new;
       q = q->next;
       if (q != NULL){
         3:           q->data = new;
         unlock();
         new ++;
       }
   }while(new != old);
  5:   unlock ();
}
Analyze Counterexample

Example ( ) {
1:   do{
       lock();
       old = new;
       q = q->next;
2:       if (q != NULL){
3:         q->data = new;
            unlock();
            new ++;
        }
4:   }while(new != old);
5:   unlock();
}
Analyze Counterexample

Example ( ) {
  1: do{
      lock();
      old = new;
      q = q->next;
  2:   if (q != NULL){
  3:     q->data = new;
         unlock();
         new ++;
  }
  4: }while(new != old);
  5: unlock();
}

[Inconsistent]

Predicates: LOCK

old = new
new++

new == old
Repeat Build-and-Search

Example ( ) {
1: do{
    lock();
    old = new;
    q = q->next;
2:   if (q != NULL){
3:     q->data = new;
        unlock();
        new ++;
    }
4:}while(new != old);
5: unlock ()
}

Predicates:  LOCK, new==old
Repeat Build-and-Search

Example () {
  do{
    lock();
    old = new;
    q = q->next;
    if (q != NULL){
      q->data = new;
      unlock();
      new ++;
    }
  }while(new != old);
  unlock();
}

Predicates: \textit{LOCK, new==old}
Repeat Build-and-Search

Example () {
  1: do{
      lock();
      old = new;
      q = q->next;
    2:   if (q != NULL){
      3:     q->data = new;
      4:     unlock();
      5:     new ++;
    }
  }while(new != old);
  6: unlock();
}

Predicates: LOCK, new==old
Repeat Build-and-Search

Example ( ) {
  1: do{
      lock();
      old = new;
      q = q->next;
  2:     if (q != NULL){
  3:       q->data = new;
               unlock();
               new ++;
  4:   }while(new != old);
  5: unlock ();
}

Predicates:  \textit{LOCK, new==old}
Repeat Build-and-Search

Example ( ) {
  1: do{
      lock();
      old = new;
      q = q->next;
  2:   if (q != NULL){
  3:     q->data = new;
        unlock();
        new ++;
  4: } while(new != old);
  5: unlock();
}

Predicates: LOCK, new==old
Repeat Build-and-Search

Example ( ) {
1: do{
   lock();
   old = new;
   q = q->next;
2:   if (q != NULL){
3:     q->data = new;
        unlock();
        new ++;
   }
4:}while(new != old);
5: unlock ();
}

Predicates: $\text{LOCK, new==old}$
Another Example

C program

1: \texttt{x = ctr;}
2: \texttt{y = ctr + 1;}
3: \texttt{if (x = i-1){}
4: \texttt{if (y != i){
   ERROR:}
   \}}

Abstract

1: \texttt{skip;}
2: \texttt{skip;}
3: \texttt{if (*){
4: \texttt{if (*){
   ERROR:
   \}}
   \}}

No predicates available currently
Checking the abstract model

Abstract model has a path leading to error state

Is ERROR reachable?

yes

Abstract model has a path leading to error state

1: skip;
2: skip;
3: if (*){
4:   if (*){
      ERROR:
        }
      }
}
1: skip;
2: skip;
3: if (*){
4:   if (*){
ERROR:
    }
}

Does this correspond to a real bug?

1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume (y != i)

Check using a SAT solver
Not possible

Concrete trace
Refinement

Spurious Counterexample

1: x = ctr;
2: y = ctr + 1;
3: assume (x == i-1)
4: assume (y != i)

Initial abstraction

1: skip;
2: skip;
3: if (*){
4:   if (*){
      ERROR:
    }
  }
}
Refinement

1: \( x = \text{ctr}; \)
2: \( y = \text{ctr} + 1; \)
3: \( \text{assume}(x == i-1) \)
4: \( \text{assume}(y != i) \)

1: \( \text{skip}; \)
2: \( \text{skip}; \)
3: \( \text{if} \ (*) \{ \)
4: \( \text{if} \ (b0) \{ \)
5: \( \text{ERROR:} \)
6: \( \} \)
7: \( \} \)

boolean \( b0 : y != i \)
Refinement

1: x = ctr;
2: y = ctr + 1;
3: assume(x == i-1)
4: assume(y != i)

1: skip;
2: skip;
3: if (b1){
4:   if (b0){
          ERROR:
          }
    }

boolean b0 : y != i
boolean b1 : x == i-1
Weakest Preconditions

\[WP(P, OP)\]

Weakest formula \(P'\) s.t.
if \(P'\) is true before \(OP\)
then \(P\) is true after \(OP\)
**Weakest Preconditions**

\[ WP(P, OP) \]

Weakest formula \( P' \) s.t.
- if \( P' \) is true before \( OP \)
- then \( P \) is true after \( OP \)

Assign

\[ x = e \]

\[ P[e/x] \]

\[ new + 1 = old \]

\[ new = new + 1 \]
Weakest precondition of $y 
eq i$

```java
define x = ctr;
define y = ctr + 1;
assume (x == i - 1);
assume (y != i);
```

```java
define skip;
define b0 = b2;
if (b1) {
  if (b0) {
    ERROR:
  }
}
```
Refinement

1. $x = \text{ctr}$;
2. $y = \text{ctr} + 1$;
3. assume ($x == i - 1$)
4. assume ($y != i$)

1. $b1 = b3$;
2. $b0 = b2$;
3. if ($b1$){
   
   ERROR:
   
   }
4. if ($b0$){
   
   }

boolean $b2 : ctr + 1 != i$
boolean $b3 : ctr == i - 1$

boolean $b0 : y != i$
boolean $b1 : x == i - 1$
What about initial values of b2 and b3?
- b2 = 1, b3 = 0
- b2 = 0, b3 = 1

b2 and b3 are mutually exclusive.

So system is safe!

Refinement

boolean b0 : y != i
boolean b1 : x == i - 1

boolean b2 : ctr + 1 != i
boolean b3 : ctr == i - 1

1: b1 = b3;
2: b0 = b2;
3: if (b1){
4:   if (b0){
       ERROR:
   }
}

So system is safe!
Tools for Predicate Abstraction of C

- **SLAM at Microsoft**
  - Used for verifying correct sequencing of function calls in windows device drivers

- **MAGIC at CMU**
  - Allows verification of concurrent C programs
  - Found bugs in MicroC OS

- **BLAST at Berkeley**
  - Lazy abstraction, interpolation

- **SATABS at CMU**
  - Computes predicate abstraction using SAT
  - Can handle pointer arithmetic, bit-vectors

- **F-Soft at NEC Labs**
  - Localization, register sharing