SE547: Lecture 7

Overview

Overview

Software Security

Software Model Checking

Logical Satisfaction

Binary Decision Diagrams

Implementing BDDs

Software Security

Secure software is intended to grant *rights* to users acting in certain *roles*.

What are examples of rights and roles?

Incorrect software can result in *rights escalation*. What is this? What are examples?

What are common strategies for attackers achieving rights escalation?

What can we do about these attacks?

Software Security

What is a buffer overflow attack?

Can buffer overflow attacks occur in C? In Java?

What language feature of C or Java allows buffer overflows?

What can we do about it? Statically? Dynamically?

What are the tradeoffs between static and dynamic checks?

Software Model Checking

What is a software model checker?

How does a software model checker compare to conventional testing?

Can a software model checker find all bugs?

Software Model Checking

Software model checkers translate source code into (possibly big!) logical formulas. For example:

```
void foo (int x, int y) {
  char[] z = new char[x+1];
  if (y < x) { z[y] = 'a'; }
}
```

Is this program safe? What logical formula do we generate from it?

We have turned the problem of software model checking into the problem of logical satisfaction.

More on software model checkers later...

Logical Satisfaction

What is first order predicate logic? What is logical satisfaction?

Logical satisfaction is NP-complete. What does this mean? What is the best-known running time for an NP-complete problem?

For example, how long will it take to determine satisfaction for:

$$\forall\, x : \text{uint16} \,.\, \forall\, y : \text{uint16} \,.\, \forall\, z : \text{uint16} \,.\, (x + y) + z = x + (y + z)$$

Try it... simplebdd.zip contains a TestArithBrute class.

Logical Satisfaction

Logical satisfaction is solved by SAT-solvers.

High-quality production SAT-solvers exist Chaff, zChaff, ... See satlive.org.

> We have success stories of using zChaff to solve problems with more than one million variables and 10 million clauses.
> – *zChaff web site*

A common simple solution is *binary decision diagrams*.

Try it... simplebdd.zip contains a BDD-based TestArith class.

Binary Decision Diagrams

A first shot at solving satisfaction: convert to Disjunctive Normal Form (DNF).

What is DNF?

How do we check satisfiability of a formula in DNF?

Why is this not an acceptable solution?

Binary Decision Diagrams

A better shot; use:

$s \rightarrow t, u$ read as 'if s then t else u'

Convert to If-then-else Normal Form (INF).

What is INF?

What is the INF for the following:

$x \oplus y \oplus z$

We can draw INFs as *binary decision trees*. What is a decision tree? What is the decision tree for this formula?

Decision trees are still not acceptable: why not?

Binary Decision Diagrams

A decision *diagram* is a decision tree where we *share* nodes.

What is the decision diagram for the following:

$$x \oplus y \oplus z$$

What is the decision diagram for the follwing:

$$x_1 \oplus ... \oplus x_n$$

Hooray, much better!

Binary Decision Diagrams

Binary Decision Diagrams are a graphical representation of the grammar:

$$
\begin{array}{l}
s, t, u ::= \\
\quad 0 \\
\quad 1 \\
\quad x \rightarrow t, u
\end{array}
$$

A couple of improvements to Binary Decision Diagrams:

a) What should we do about $x \rightarrow t, t$? [*Reduced* BDDs.]

b) There are two representations of $x \oplus y$: what are they? What should we do about this? [*Ordered* BDDs.]

Binary Decision Diagrams

Reduced, Ordered BDDs have the following very nice property:

$t \Leftrightarrow u$ is a tautology just when $t = u$

that is *syntactic equality* is the same as *semantic equality*.

In particular, for ROBDDs:

- The only tautology is 1.
- The only unsatisfiable formula is 0.

This makes checking for satisfiability pretty easy! (But remember that satisfiability is NP-complete; where did the hard work go?)

Implementing BDDs

How can we convert this grammar into a class hierarchy?

s, t, u ::=
- 0
- 1
- $x \rightarrow t, u$

Hint: start with:

```
interface BoolPred {
  BoolPred ite (BoolPred p, BoolPred q); // this -> p, q
  ...
  static final BoolPred T = ...;
  static final BoolPred F = ...;
}
```

Implementing BDDs

How can we implement the ite method?

```
interface BoolPred {
  BoolPred ite (BoolPred p, BoolPred q); // this -> p, q
  ...
  static final BoolPred T = ...;
  static final BoolPred F = ...;
}
```

Implementing BDDs

Implementation trick: *use the flyweight pattern*. What is this? Why does it help?

(Other names for this kind of flyweighting: dynamic programming, memoization...)

A very simple implementation of BDDs is in simplebdd.bool.BoolPred in simplebdd.zip (approx. 150loc!).

Implementing BDDs

BDDs are just about binary data, but we can code up fixed-width arithmetic...

How do we code an n-bit integer variable?

How do we code an n-bit integer expression?

How do we code n-bit integer arithmetic?

A partial implementation is in simplebdd.integer.IntPred in simplebdd.zip (approx. 200loc).

Next week

Model checking.