# Introduction to Model-checking

## CS 6335: Language-based Security

Kevin W. Hamlen

September 25, 2023

# Software Verification Approaches

- Unit Testing / Fuzzing
  - Throw many test inputs (often randomly generated) at software and see whether it fails.
  - Good for fault detection. Inadequate for security.
    - input space usually infinite
    - attackers seek out and exploit untested inputs
- Program-Proof Co-Development (Coq)
  - Implement software in a "nice" (e.g., functional) language.
  - Write formal correctness properties and proofs.
  - Proofs are *machine-checked* (not trusted).
  - Pros: highest assurance, covers infinite state space
  - Con: painful to write proofs
- Today: Model-checking
  - a middle-ground between random fuzzing and formal proofs
  - Express software as an abstract, finite-state *model* $\mathcal{M}$.
  - Express security property as a logical predicate $\phi$.
  - Decide $\mathcal{M} \models \phi$ by exhaustive state-space search.

# Some History

- First developed in 1980s by Clarke, Emerson, and Sifakis (Turing Award 2007)
  - primarily targeted hardware verification
  - disillusionment with proofs in 80s and 90s
  - found previously undetected errors in 1992 IEEE Future+ cache coherence protocol
  - 1994 Intel Pentium floating-point bug
    - passed unit testing
    - cost Intel $400–500 million
    - could have been detected by model-checking
  - model-checking now routinely used by Intel, AMD, IBM, Lucent, etc.
- Rise of Software Model-checking in late 90s
  - VeriSoft (Lucent), SPIN (Holtzmann, Bell Labs)
  - Big challenge: state-space explosion

# Example (from JavaPathFinder documentation)

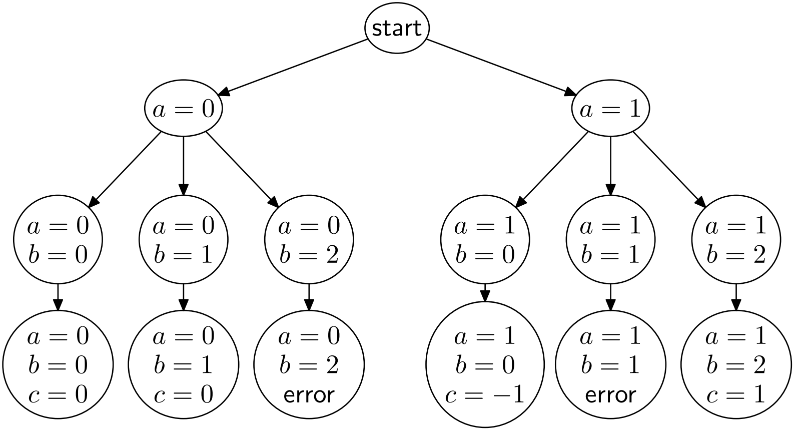
```
Random random = new Random();
int a = random.nextInt(2);
System.out.println("a=" + a);

// lots of code here

int b = random.nextInt(3);
System.out.println("b=" + b);
int c = a/(b+a-2);
System.out.println("c=" + c);
```

- Sample run:
  - $a = 1$
  - $b = 0$
  - $c = -1$

# State Space


start
$a = 0$
$a = 1$
$a = 0$ $b = 0$
$a = 0$ $b = 1$
$a = 0$ $b = 2$
$a = 1$ $b = 0$
$a = 1$ $b = 1$
$a = 1$ $b = 2$
$a = 0$ $b = 0$ $c = 0$
$a = 0$ $b = 1$ $c = 0$
$a = 0$ $b = 2$ error
$a = 1$ $b = 0$ $c = -1$
$a = 1$ $b = 1$ error
$a = 1$ $b = 2$ $c = 1$

# State Spaces

- Not always (or even usually) trees
  - conditionals = multiple in-edges
  - program loops = cycles
- Does not always match control-flow graph structure
  - One program line could correspond to many different states, depending on the values of its variables.
  - Abstracting coalesces states (more on this later...)
- Can be *huge*
  - How many states if we change the "2" argument in line 2?

# Properties

- Typically expressed in a temporal logic
- Flagship example: Linear Temporal Logic (LTL)
- Assertions: $\pi \models \phi$ — path $\pi$ models property $\phi$
  - atomic propositions (e.g., is_error, $a = 2$, etc.)
  - $\neg\phi$ — negation
  - $\phi_1 \vee \phi_2$ — disjunction
  - $\mathbf{X}(\phi)$ — next $\phi$
  - $\mathbf{U}(\phi_1, \phi_2)$ — $\phi_1$ until $\phi_2$
  - $\mathbf{F}(\phi)$ — finally $\phi$
  - $\mathbf{G}(\phi)$ — globally $\phi$
- Exercise: Do all paths from "start" model the following?
  - $\mathbf{X}(a = 0)$
  - $\mathbf{U}(\neg\text{is\_error}, b > 0)$
  - $\mathbf{F}(\mathbf{U}(\text{false}, b \leq 2))$

# Branching Temporal Logics

- LTL cannot express most existential properties
  - Example: "for every state *there exists* a non-error step"
- Solution: Branching Temporal Logics
- Flagship example: Modal $\mu$-Calculus
- Assertions: $s \models \psi$ — state $s$ is a member of the set of all states denoted by $\psi$
  - $\psi_1 \wedge \psi_2$ — conjunction (intersection)
  - $\psi_1 \vee \psi_2$ — disjunction (union)
  - $[a]\psi$ — all outgoing $a$-transitions model $\psi$
  - $\langle a \rangle \psi$ — some outgoing $a$-transitions model $\psi$
  - $\mu X \,.\, \psi$ — least fixed point
  - $\nu X \,.\, \psi$ — greatest fixed point
- What are least and greatest "fixed points"?

# Fixed Point Semantics

**Definition:** A *fixed point* of a function $f : A \rightharpoonup A$ is a value $x \in A$ such that $f(x) = x$.

- Examples:
  - What is a fixed point of $f(x) = x + 1$ ?
  - What is a fixed point of $g(x) = x^2$ ?
  - What is a fixed point of $h(S) = \{x^2 \mid x \in S\}$ ?
- When $f$ is a function from sets to sets, we say $S$ is...
  - ...a *least fixed point* if $S$ is a fixed point and all other fixed points are supersets of $S$.
  - ...a *greatest fixed point* if $S$ is a fixed point and all other fixed points are subsets of $S$.
- Can a function have multiple least fixed points or multiple greatest fixed points?

# Fixed Point Operators

- Back to modal $\mu$-calculus:
  - $\mu X \,.\, \psi$ is the *least* set $S$ such that $S = \psi[X := S]$
  - $\nu X \,.\, \psi$ is the *greatest* set $S$ such that $S = \psi[X := S]$
- Finding least/greatest fixed points:
  - Find $\mu X \,.\, \psi$ inductively:
    - start with $X = \emptyset$
    - keep adding things to $X$ until no progress
  - Find $\nu X \,.\, \psi$ co-inductively:
    - start with $X =$universe of all states
    - keep removing things from $X$ until no progress
  - Examples:
    - What is $\mu X \,.\, (X \vee \langle\rangle\text{is\_error})$ ?
    - What is $\nu X \,.\, (\text{is\_error} \vee \langle\rangle X)$ ?

# State Space Explosion Problem

- Main challenge: What if the state space is huge?
- Example: How many states does the following program have?

```
int i = 0;
while true do
    i := i + 1;
```

- Solution: Abstract Interpretation
  - Instead of having one state for every mapping of variables to values, label states with abstract properties.
  - Example: What if we only care about whether `i` is zero (e.g., to avoid division-by-zero)?
  - Could instead just have one state for each possible *sign* of `i`
    - $\textit{zero} + \textit{positive} = ?$
    - $\textit{positive} + \textit{positive} = ?$

# State Space Explosion Problem

- Main challenge: What if the state space is huge?
- Example: How many states does the following program have?

```
int i = 0;
while true do
    i := i + 1;
```

- Solution: Abstract Interpretation
  - Instead of having one state for every mapping of variables to values, label states with abstract properties.
  - Example: What if we only care about whether `i` is zero (e.g., to avoid division-by-zero)?
  - Could instead just have one state for each possible *sign* of `i`
    - $\textit{zero} + \textit{positive} = \textit{positive}$
    - $\textit{positive} + \textit{positive} = ?$

# State Space Explosion Problem

- Main challenge: What if the state space is huge?
- Example: How many states does the following program have?

```
int i = 0;
while true do
    i := i + 1;
```

- Solution: Abstract Interpretation
  - Instead of having one state for every mapping of variables to values, label states with abstract properties.
  - Example: What if we only care about whether `i` is zero (e.g., to avoid division-by-zero)?
  - Could instead just have one state for each possible *sign* of `i`
    - *zero* + *positive* = *positive*
    - *positive* + *positive* = *positive*
    - We're finished with only 2 states to explore!

# Counterexample Guided Abstraction Refinement (CEGAR)

- Over-abstraction Problem
  - If model-check succeeds on abstract model, then we're done. But...
  - Abstracting often forgets information needed to prove correctness.
  - Results in false rejection (model-checker signals fault where there is none)
- Solution: Iteratively Abstract and Refine
  1. Abstract until search space is feasible.
  2. Exhaustively search the space. If model-check rejects...
  3. Test the counterexample on the original (non-abstract) search space. If it's a real counterexample, we found a real bug. Otherwise...
  4. We must have abstracted too much. Refine (opposite of abstract) and repeat.
- Next time: Information flow analysis