CS 520

Theory and Practice of Software Engineering
Fall 2019

Model Checking

October 24, 2019

1

coming up

- The third in-class will take place next Tuesday, Oct 29.
  - Sign up for a team on moodle
  - Bring a laptop to class
- Final project
  - Mid-point report due Nov 7
    https://people.cs.umass.edu/~brun/class/2019Fall/CS520/midProjectReport.pdf

2

Systems are known to be error-prone

- Capture complex aspects such as:
  - Threads and synchronization (e.g., Java locks)
  - Dynamically heap allocated structured data types (e.g., Java classes)
  - Dynamically stack allocated procedures (e.g., Java methods)
  - Non-determinism
- Challenging to reason about all possible traces through the systems

3

3

Potential uses of model checking

- Verification and validation (V&V)
- Debugging

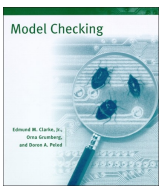
4

4

Goal of model checking

Automate reasoning about whether all **traces** through a **system** satisfy a given **property specification**

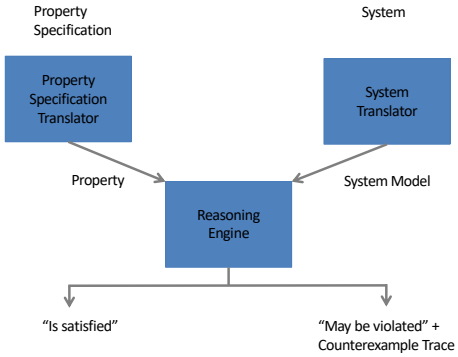
- If so, report "Is satisfied"
- If not, report "May be violated" and generate a **counterexample trace** that illustrates a potential violation of the property specification



5

5

Model checker architecture



6

6

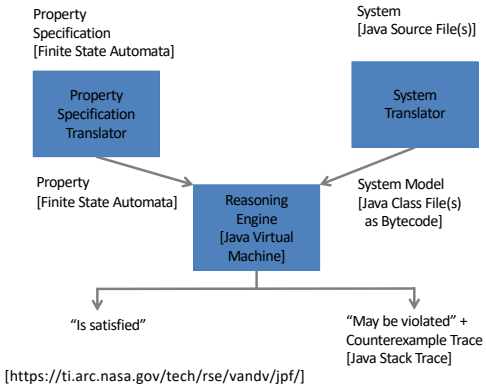
Examples of model checkers

- NuSMV
- Spin
- Java Pathfinder (JPF)
- UPPAAL
- PRISM

7

7

JPF: Model checker architecture



[https://ti.arc.nasa.gov/tech/rse/vandv/jpf/]

8

8

Running example

```
public class BoundedBuffer {

  static int BUFFER_SIZE = 1;
  static int N_PRODUCERS = 4;
  static int N_CONSUMERS = 4;

  static Object DATA = "fortytwo";

  //--- the bounded buffer implementation
  protected Object[] buf;
  protected int in = 0;
  protected int out= 0;
  protected int count= 0;
  protected int size;

  public BoundedBuffer(int size) { ... }

  public synchronized void put(Object o) throws InterruptedException { ... }

  public synchronized Object get() throws InterruptedException { ... }

  static class Producer extends Thread {
    // Iteratively calls put method
    ...
  }

  static class Consumer extends Thread {
    // Iteratively calls get method
    ...
  }

  public static void main(String [] args) {
      // 1. Creates bounded buffer, producers, and consumers
      // based on command line arguments
      // 2. Starts the produces and consumers
      ...
  }
}
```

9

9

Property specification

- System requirements are represented as a set of property specifications
- Each property specification formally defines an intended (or unintended) behavior of the system
  - May take into account real-time or probabilistic constraints

10

10

Examples of property specifications

- Multi-threading and synchronization
  - Deadlock
  - Race detection
- Data structure consistency
  - No buffer under/over flow

11

11

Commonly used property specification languages

- Graphs where the nodes (or edges) are associated with elements from an alphabet
  - Finite state automata (FSAs): Events, Labeled transition system: Propositions
- Temporal logics
  - Linear temporal logic (LTL), Computational tree logic (CTL), CTL*

12

12

JPF: Property specifications

- *gov.nasa.jpf.jvm.NotDeadlockedProperty*
- *gov.nasa.jpf.listener.PreciseRaceDetector*
- *No buffer under/over flow: Never throw exception ArrayIndexOutOfBounds:*

```
public class NoUncaughtExceptionsProperty extends GenericProperty {

  // <2do> that's a hack for now (makes us de-facto a singleton)
  static ExceptionInfo uncaughtXi;

  public NoUncaughtExceptionsProperty (Config config) {
    uncaughtXi = null;
  }

  static void setExceptionInfo (ExceptionInfo xi) { uncaughtXi = xi; }

  public ExceptionInfo getUncaughtExceptionInfo() { return uncaughtXi; }

  public String getExplanation () { return null; }

  public String getErrorMessage () { ... }

  public void reset() { uncaughtXi = null; }

  public boolean check (Search search, JVM vm) { return (uncaughtXi == null); }
}
```

13

13

Commonly used system modeling languages

- Automata such as communicating, timed, …
- Labeled transition systems
- Compiler internal representations
  - e.g., flow graphs, **bytecode**

14

14

System translation

Possible alternatives:

- Manually translate (e.g., Java to Labeled Transition System)
- Automatically translate before execution
- Automatically "translate" during execution (e.g., JPF)

NOTE) The translation often incorporates compiler optimizations to improve applicability and scalability.

15

15

Examples of system translation optimizations

- Method inlining
- Slicing based on the property specification
  - Keep code related to that property specification, e.g., BoundedBuffer code related to synchronization
  - Remove code NOT related to it, e.g., Consumer code related to how data gotten from the BoundedBuffer is used

16

16

JPF: System translation

Steps:

1) Translate Java source file(s) to Java class file(s) represented as bytecode
2) Optimize Java class file(s)

```
public BoundedBuffer(int);
  Code:
    0:   aload_0
    1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
    4:   aload_0
    5:   iconst_0
    6:   putfield        #2; //Field in:I
    9:   aload_0
    10:  iconst_0
    11:  putfield        #3; //Field out:I
    14:  aload_0
    15:  iconst_0
    16:  putfield        #4; //Field count:I
    19:  aload_0
    20:  iload_1
    21:  putfield        #5; //Field size:I
    24:  aload_0
    25:  iload_1
    26:  anewarray       #6; //class java/lang/Object
    29:  putfield        #7; //Field buf:[Ljava/lang/Object;
    32:  return
```

17

17

Reasoning engine: Conceptually

Generate the **reachability graph** for the given system model and property:

- Each **node** captures a system model execution state, e.g.,

  <P1: Not started,…,C4: Not started,
  BB lock is free, BB buf is empty, BB counts are zero>

- Each **edge** captures a current node executing an "instruction" (e.g., P1 start) to generate the next node, e.g.,

  <P1: Started,…,C4: Not started,
  BB lock is free, BB buf is empty, BB counts are zero>

18

18

Reasoning engine: Determine results

Report:

- "May be violated" if a node is encountered that illustrates a potential violation of the property (and generate the counterexample trace)
- "Is satisfied" if no such nodes are encountered

19

19

Examples of reasoning engines

- Explicitly generate the reachability graph (e.g., Spin, JPF)
- Symbolically generate the reachability graph (e.g., nuSMV)
  - Save space by encoding each set of nodes as a binary decision diagram
  - Can complicate counterexample generation
- SAT solvers

20

20

Counterexample traces: What?

Represented as a sequence of reachability graph nodes where:

1. Start at the initial node
2. For each current node at index i, be able to generate its next node at index i + 1
3. End at a final node illustrating the potential violation of the property

21

21

JPF: Counterexample trace

```
thread BoundedBuffer$Producer:{id:1,name:P1,status:WAITING,priority:5,lockCount:1,suspendCount:0}
  waiting on: BoundedBuffer@148
  call stack:
        at java.lang.Object.wait(Object.java)
        at BoundedBuffer.put(BoundedBuffer.java:55)
        at BoundedBuffer$Producer.run(BoundedBuffer.java:91)

thread BoundedBuffer$Producer:{id:2,name:P2,status:WAITING,priority:5,lockCount:1,suspendCount:0}
  waiting on: BoundedBuffer@148
  call stack:
        at java.lang.Object.wait(Object.java)
        at BoundedBuffer.put(BoundedBuffer.java:55)
        at BoundedBuffer$Producer.run(BoundedBuffer.java:91)

thread BoundedBuffer$Producer:{id:3,name:P3,status:WAITING,priority:5,lockCount:1,suspendCount:0}
  waiting on: BoundedBuffer@148
  call stack:
        at java.lang.Object.wait(Object.java)
        at BoundedBuffer.put(BoundedBuffer.java:55)
        at BoundedBuffer$Producer.run(BoundedBuffer.java:91)

thread BoundedBuffer$Producer:{id:4,name:P4,status:WAITING,priority:5,lockCount:1,suspendCount:0}
  waiting on: BoundedBuffer@148
  call stack:
        at java.lang.Object.wait(Object.java)
        at BoundedBuffer.put(BoundedBuffer.java:55)
        at BoundedBuffer$Producer.run(BoundedBuffer.java:91)

thread BoundedBuffer$Consumer:{id:5,name:C1,status:WAITING,priority:5,lockCount:1,suspendCount:0}
  waiting on: BoundedBuffer@148
  call stack:
        at java.lang.Object.wait(Object.java)
        at BoundedBuffer.get(BoundedBuffer.java:66)
        at BoundedBuffer$Consumer.run(BoundedBuffer.java:110)
```

22

22

Counterexample traces: Why?

Caused by issue with:

- Property specification
- System model
- System - Actual bug

23

23

JPF: Demonstration

- System: Bounded buffer
- Property specifications:
  - No deadlock: Violated
  - No data races: Satisfied
  - Never ArrayIndexOutOfBounds: Satisfied
- Configuration: jpf.properties and .jpf files

24

24

Model checker evaluation

- Applied to benchmarks and actual systems
  - Have found actual bugs
- Compared in terms of:
  - performance: space and time
  - counterexample traces generated: usually by their length

25

25

Potential benefits of model checking

- Automatically checks that all traces through a given system model satisfies its property specifications
  - Can be re-checked after any changes
- Generates counterexample traces that can be used for debugging
- Generally requires less expertise than for formal verification techniques

26

26

Disadvantages of model checking

- Translating from the system to the system model can be error-prone
- Writing property specifications can also be error-prone
- May not scale well because of the state space explosion problem
- May not generate counterexample traces that are useful for debugging (e.g., too long, to similar to each other)

27

27

Property specification patterns: Overview

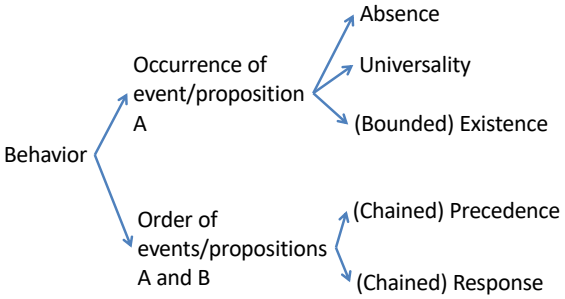
- Consists of a:
  - **Behavior** captures occurrence or order of events/propositions (e.g., Absence of *ThrowArrayIndexOutOfBounds*)
  - **Scope** captures parts of the trace where behavior must be satisfied (e.g., Globally)
- Provides mapping to various property specification languages (e.g., regular expressions)
  - e.g., "[- *ThrowArrayIndexOutOfBounds*]"

[https://matthewbdwyer.github.io/psp/]

28

28

Property specification patterns: Behaviors

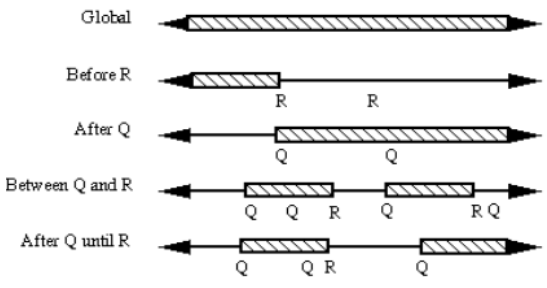


[https://matthewbdwyer.github.io/psp/]

29

29

Property specification patterns: Scopes



[https://matthewbdwyer.github.io/psp/]

30

30

PROPEL: Overview

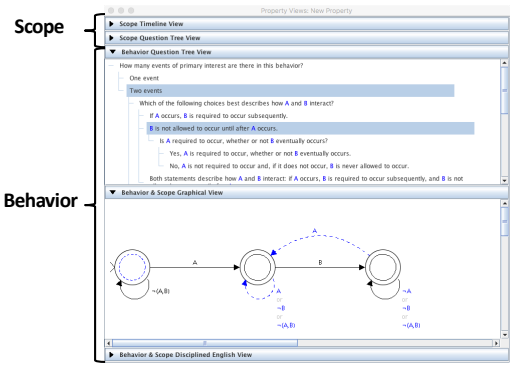
- Builds on the property specification patterns specified as finite state automata
- Provides guidance to select among the patterns and customize that pattern

[http://laser.cs.umass.edu/tools/propel.shtml]

31

31

PROPEL: Tool



[Available from http://laser.cs.umass.edu/release/]

32

32

Property specification patterns: Extensions

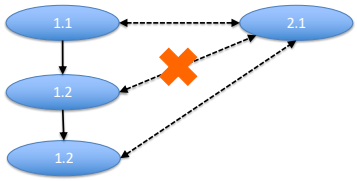
- Add new property specification patterns
- Map to new property specification languages
- Provide support for real-time or probabilistic constraints [http://ps-patterns.wikidot.com]

33

33

Examples of model checker optimizations

- Abstraction of variables
  - e.g., Integer type abstracted as zero or non-zero
- Partial order reduction for thread scheduler
  - e.g., Thread 1 Thread 2



34

34

Search-based counterexample trace generation

- Want to support:
  - Breadth first search: Generally slow but short counterexample traces that are different
  - (Bounded) depth first search: Generally fast but long counterexample traces that are similar
  - A* search with heuristics
- Iteratively generate the reachability graph
  - Store a worklist of current nodes (e.g., BFS queue)
  - Store a visited set of nodes (e.g., BFS hash set of nodes)

35

35