CS 520
Theory and Practice of Software Engineering
Fall 2021

Model Checking

November 9, 2021
Ways to get your code right

• Verification & Validation
  – Purpose is to uncover problems and increase confidence
  – Combination of manual and automated reasoning (e.g., model checkers, theorem provers) as well as testing

• Debugging
  – Purpose is finding out why a program is not functioning as intended
  – Pinpoint location + cause of problem

• Defensive programming
  – Programming with validation and debugging in mind
Why use parallelism?

• Provide service for multiple requests, e.g.,

• Decompose a computation, e.g.,
  – Java Swing event handling (https://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html)

• Collaborative work, e.g.,
  – Operating room team composed of multiple specialty teams

• …
Distributed versus Concurrent Systems

• **Distributed system**
  – Usually supported by the operating system
  – Autonomous parallel processors that do not share memory
  – Heavy weight processor context switching

• **Concurrent system**
  – Usually supported by programming languages, libraries, or both (e.g., Java)
  – Autonomous parallel threads that do share memory
  – Lighter weight thread scheduling
Support for concurrency

• **Design modeling languages**, e.g.,

• **Programming languages**
  – Language feature, e.g., Java Thread (https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/lang/Thread.html)
public class BoundedBuffer {

    static int BUFFER_SIZE = 2;
    static int N_PRODUCERS = 2;
    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 InterruptedIOException { ... } 

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

    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
        ...
    }
}
Example: BB producer thread

//---- the producer
static class Producer extends Thread {
    static int nProducers = 1;
    BoundedBuffer buf;

    Producer(BoundedBuffer b) {
        buf = b;
        setName("P" + nProducers++);
    }

    public void run() {
        try {
            while(true) {
                // to ease state matching, we don't put different objects
                // in the buffer
                buf.put(DATA);
            }
        } catch (InterruptedException e) { }
    }
}

from main method:

for (int i=0; i<N_PRODUCERS; i++) {
    new Producer(buf).start();
}
Example: BB producer thread

```java
//---- the producer
static class Producer extends Thread {
    static int nProducers = 1;
    BoundedBuffer buf;

    Producer(BoundedBuffer b) {
        buf = b;
        setName("P" + nProducers++);
    }

    public void run() {
        try {
            while(true) {
                // to ease state matching, we don't put different objects
                // in the buffer
                buf.put(DATA);
            }
        } catch (InterruptedException e) {}
    }
}

for (int i=0; i<N_PRODUCERS; i++) {
    new Producer(buf).start();
}
```

From main method:
Example: BB producer thread

```java
//---- the producer
static class Producer extends Thread {
    static int nProducers = 1;
    BoundedBuffer buf;

    Producer(BoundedBuffer b) {
        buf = b;
        setName("P" + nProducers++);
    }

    public void run() {
        try {
            while(true) {
                // to ease state matching, we don't put different objects
                // in the buffer
                buf.put(DATA);
            }
        } catch (InterruptedException e) {
        }
    }
}

for (int i=0; i<N_PRODUCERS; i++) {
    new Producer(buf).start();
}

From main method:
```
Example: BB producer thread

```java
// --- the producer
static class Producer extends Thread {
    static int nProducers = 1;
    BoundedBuffer buf;

    Producer(BoundedBuffer b) {
        buf = b;
        setName("P" + nProducers++);
    }

    public void run() {
        try {
            while(true) {
                // to ease state matching, we don't put different objects
                // in the buffer
                buf.put(DATA);
            }
        } catch (InterruptedException e) {}{
    }
}

for (int i=0; i<N_PRODUCERS; i++) {
    new Producer(buf).start();
}
```

From main method:
Example: BB producer thread

```java
//---- the producer
static class Producer extends Thread {
    static int nProducers = 1;
    BoundedBuffer buf;

    Producer(BoundedBuffer b) {
        buf = b;
        setName("P" + nProducers++);
    }

    public void run() {
        try {
            while (true) {
                // to ease state matching, we don't put different objects
                // in the buffer
                buf.put(DATA);
            }
        } catch (InterruptedException e) {
        }
    }
}

for (int i=0; i<N_PRODUCERS; i++) {
    new Producer(buf).start();
}
```
Example: BB consumer thread

```java
//--- the consumer
static class Consumer extends Thread {
    static int nConsumers = 1;
    BoundedBuffer buf;

    Consumer(BoundedBuffer b) {
        buf = b;
        setName( "C" + nConsumers++);
    }

    public void run() {
        try {
            while(true) {
                Object tmp = buf.get();
            }
        } catch(InterruptedException e ){}
    }

    for (int i=0; i<N_CONSUMERS; i++) {
        new Consumer(buf).start();
    }
```
Example: BB consumer thread

```java
// the consumer
static class Consumer extends Thread {
    static int nConsumers = 1;
    BoundedBuffer buf;

    Consumer(BoundedBuffer b) {
        buf = b;
        setName( "C" + nConsumers++);
    }

    public void run() {
        try {
            while(true) {
                Object tmp = buf.get();
            }
        } catch(InterruptedException e) {};
    }
}
```

From main method:

```java
for (int i=0; i<N_CONSUMERS; i++) {
    new Consumer(buf).start();
}
```
Support for synchronization

• Monitor
  – In Java, **synchronized** keyword
  – In java.util.concurrent.locks, **Lock** interface

• Other synchronizers
  – In java.util.concurrent (e.g., Semaphore)

• ...


Example: BB constructor

```java
//--- 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) {
    this.size = size;
    buf = new Object[size];
}
```
Example: BB put method

```java
public synchronized void put(Object o) throws InterruptedException {
    while (count == size) {
        wait();
    }
    buf[in] = o;
    //System.out.println("PUT from " + Thread.currentThread().getName() + "+ " + ++count);
    in = (in + 1) % size;
    notify();
}
```
Example: BB put method

```java
public synchronized void put(Object o) throws InterruptedException {
    while (count == size) {
        wait();
    }
    buf[in] = o;
    //System.out.println("PUT from " + Thread.currentThread().getName());
    ++count;
    in = (in + 1) % size;
    notify();
}
```
Each BoundedBuffer object is associated with a reentrant lock:
- Acquire that lock at the beginning of a synchronized method
- Release the lock at the end of that method
Example: BB put method

```java
public synchronized void put(Object o) throws InterruptedException {
    while (count == size) {
        wait();
    }
    buf[in] = o;
    ++count;
    in = (in + 1) % size;
    notify();
}
```

Each BoundedBuffer object is also associated with a wait set:
- Add the current thread to that set for wait method
- Non-deterministically remove a thread from the set for notify method
Example: BB get method

```java
public synchronized Object get() throws InterruptedException {
    while (count == 0) {
        wait();
    }
    Object o = buf[out];
    buf[out] = null;
    //System.out.println("GET from " + Thread.currentThread().getName());
    --count;
    out = (out + 1) % size;
    notify();
    return (o);
}
```
Remote Method Invocation (RMI): Java RMI

- Java Remote Method Invocation (RMI)

- Marshall/Unmarshall input parameters and result using Java Serializable interface
  [https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/io/Serializable.html](https://docs.oracle.com/en/java/javase/15/docs/api/java.base/java/io/Serializable.html)
Middleware (e.g., CORBA)

From their website: “The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.”

http://www.ois.com/Products/what-is-corba.html
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 (e.g., Java HashSet)
  – Many input/output pairs

• Challenging to reason about all possible traces through the systems
Overview 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
**Mars Pathfinder**, robotic U.S. spacecraft launched to Mars to demonstrate a new way to land a spacecraft on the planet’s surface and the operation of an independent robotic rover. Developed by NASA as part of a low-cost approach to planetary exploration, Pathfinder successfully completed both demonstrations, gathered scientific data, and returned striking images from Mars. Its observations added to evidence that, at some time in its history, Mars was much more Earth-like than it is today, with a warmer, thicker atmosphere and much more water.
Architecture of a model checker

[Java PathFinder or JPF]

Property Specification [FSA(s)]

Property Specification translator

System [Java source file(s)]

System translator

System Model [Java class file(s) as bytecode]

Reasoning engine [Java virtual machine]

“Is satisfied”

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

“May be violated” + Counterexample trace

[Java stack trace]
Architecture of a model checker
[Java PathFinder or JPF]

Property Specification [FSA(s)]

Property Specification translator

Reasoning engine [Java virtual machine]

System [Java source file(s)]

System model [Java class file(s) as bytecode]

“Is satisfied”
[https://ti.arc.nasa.gov/tech/rse/vandv/jpf/]

“May be violated” + Counterexample trace [Java stack trace]
Architecture of a model checker
[Java PathFinder or JPF]

Property Specification
[FSA(s)]

Property Specification
translator

Reasoning engine
[Java virtual machine]

System
[Java source file(s)]

System model
[Java class file(s) as bytecode]

"Is satisfied"

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

"May be violated" +
Counterexample trace
[Java stack trace]
Architecture of a model checker
[Java PathFinder or JPF]

Property Specification [FSA(s)]

Property Specification translator

Reasoning engine [Java virtual machine]

System [Java source file(s)]
System translator

System model [Java class file(s) as bytecode]

"Is satisfied"

"May be violated" + Counterexample trace

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

• Architecture of JPF model checker
  – System and translation
  – Property specification and translation
  – Reasoning engine

• Application of model checking
  – JPF demo
  – Evaluation
  – Potential benefits and disadvantages
Architecture of a model checker [Java PathFinder or JPF]

Property Specification [FSA(s)]

Property Specification translator

Reasoning engine [Java virtual machine]

System model [Java class file(s) as bytecode]

System translator

“Is satisfied”

“May be violated” + Counterexample trace [Java stack trace]

[https://ti.arc.nasa.gov/tech/rse/vandv/jpf/]
Example: Bounded buffer (BB) system

```java
public class BoundedBuffer {
    static int BUFFER_SIZE = 2;
    static int N_PRODUCERS = 2;
    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
        ...
    }
}
```
System translation

Steps:
1) Translate Java source file(s) to Java class file(s) represented as bytecode
2) Optimize Java class file(s)
Architecture of a model checker
[Java PathFinder or JPF]

Property Specification
[FSA(s)]

Property Specification translator

Reasoning engine
[Java virtual machine]

System model
[Java class file(s) as bytecode]

System translator

Property [FSA(s)]

“Is satisfied”

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

“May be violated” + Counterexample trace
[Java stack trace]
Property specifications

• System requirements are represented as a set of property specifications

• Each property specification formally defines an intended (or unintended) behavior of the system [Represented conceptually as an FSA]
Example: No buffer under/over flow

```java
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); }
}
```

In our case, BB buf field accesses (e.g., buf[in], buf[out]) don’t throw ArrayIndexOutOfBoundsException exceptions
Example: No buffer under/over flow

1

Throw ArrayIndexOutOfBoundsException

viol

Throw ArrayIndexOutOfBoundsException
Architecture of a model checker [Java PathFinder or JPF]

Property Specification [FSA(s)]

Property Specification translator

Reasoning engine [Java virtual machine]

System model [Java class file(s) as bytecode]

System translator

Property [FSA(s)]

“Is satisfied”

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

“May be violated” + Counterexample trace [Java stack trace]
Reasoning engine: Reachability graph (1)

• Generate the **reachability graph** for the given system model and property specification (PS)

• Each **node** captures a system model execution state and property state, e.g., initial node is:

  \[ P1: \text{Not started}, \ldots, C4: \text{Not started}, \]

  \[ BB \text{ reentrant lock is free, BB wait set is empty}, \]

  \[ BB \text{ buf is empty, BB count, in, and out are zero}, \]

  \[ \ldots, \]

  \[ PS: \text{Start state} \]
Reasoning engine: Reachability graph (2)

- Each edge captures a current node, e.g., initial node, executing an “instruction”, e.g., P1 start, to generate the next node, e.g.,:
  
  \[ P1: \text{Started}, \ldots, C4: \text{Not started}, \]
  \[ BB \text{ reentrant lock is free}, BB \text{ wait set is empty}, \]
  \[ BB \text{ buf is empty}, BB \text{ count, in, and out are zero}, \]
  
  \[ \ldots, \]
  
  \[ PS: \text{Start state} \]
Reasoning engine: Determining 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
Counterexample trace

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 how to potentially reach the violation state of the property
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

• 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)
Example: JPF inputs
(No buffer under/over flow)

**System:** BoundedBuffer(2,2,4)

**Property specification:**
gov.nasa.jpf.jvm.NoUncaughtExceptionsProperty
Example: JPF results
(No buffer under/over flow)

arguments: 2 2 4

search started: 3/23/20 7:15 PM

results
no errors detected

statistics
elapsed time: 00:00:31
states: new=75560, visited=295582, backtracked=371142, end=34
search: maxDepth=752, constraints hit=0
choice generators: thread=75559 (signal=35179, lock=8790, shared ref=14960), data=0
heap: new=10944, released=271392, max live=392, gc-cycles=345483
instructions: 10880846
max memory: 233MB
loaded code: classes=83, methods=1354

search finished: 3/23/20 7:15 PM
Example: JPF inputs (Data race)

Common definition of a data race:
• Two or more threads concurrently access a memory location
• One or more access is a write
• One or more access is unsynchronized

System: BoundedBuffer(2,2,4)

Property specification:
gov.nasa.jpf.listener.PreciseRaceDetector
Example: JPF results (Data race)

search started: 3/23/20 7:24 PM

results
no errors detected

statistics
elapsed time: 00:00:30
states: new=75560, visited=295582, backtracked=371142, end=34
search: maxDepth=752, constraints hit=0
choice generators: thread=75559 (signal=35179, lock=8790, shared ref=14960), data=0
heap: new=10944, released=271392, max live=392, gc-cycles=345483
instructions: 10880846
max memory: 235MB
loaded code: classes=83, methods=1354

search finished: 3/23/20 7:24 PM
Property specification: Deadlock

1. **Mutual exclusion**: At least one resource must be held in a non-shareable mode. Otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.[6]

2. **Hold and wait or resource holding**: a process is currently holding at least one resource and requesting additional resources which are being held by other processes.

3. **No preemption**: a resource can be released only voluntarily by the process holding it.

4. **Circular wait**: each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, \( P = \{P_1, P_2, \ldots, P_N\} \), such that \( P_1 \) is waiting for a resource held by \( P_2 \), \( P_2 \) is

Example: JPF inputs (Deadlock 1)

• **System**: BoundedBuffer(2,2,4)

• **Property specification**:
  
  `gov.nasa.jpf.jvm.NotDeadlockedProperty`
Example: JPF results (Deadlock 1)

======================================== results
error #1: gov.nasa.jpf.jvm.NotDeadlockedProperty "deadlock encountered:
    thread BoundedBuffer$Produ...

======================================== statistics
elapsed time: 00:00:00
states: new=239, visited=124, backtracked=362, end=1
search: maxDepth=73, constraints hit=0
choice generators: thread=118 (signal=105, lock=3, shared ref=3), data
    =0
heap: new=425, released=50, max live=392, gc-cycles=333
instructions: 16775
max memory: 81MB
loaded code: classes=83, methods=1354

======================================== search finished:
3/23/20 7:30 PM
Example: Counter-example trace (Deadlock 1)
Example: BB put and get methods (2)

```java
public synchronized void put(Object o) throws InterruptedException {
    while (count == size) {
        wait();
    }
    buf[in] = o;
    //System.out.println("PUT from " + Thread.currentThread().getName());
    ++count;
    in = (in + 1) % size;
    notifyAll(); // if this is not a notifyAll() we might notify the wrong waiter
}

public synchronized Object get() throws InterruptedException {
    while (count == 0) {
        wait();
    }
    Object o = buf[out];
    buf[out] = null;
    //System.out.println("GET from " + Thread.currentThread().getName());
    --count;
    out = (out + 1) % size;
    notifyAll(); // if this is not a notifyAll() we might notify the wrong waiter
    return (o);
}
```
Example: JPF results (Deadlock 2)

arguments: 2 2 4

search started: 3/23/20 7:35 PM

results
no errors detected

statistics
elapsed time: 00:00:21
states: new=49883, visited=228511, backtracked=278394, end=0
search: maxDepth=123, constraints hit=0
choice generators: thread=49884 (signal=19032, lock=7348, shared ref=1331), data=0
heap: new=9504, released=201420, max live=392, gc-cycles=255671
instructions: 8954221
max memory: 81MB
loaded code: classes=83, methods=1354

search finished: 3/23/20 7:36 PM
HeathernboysMBP:examples hconboy$
Evaluation of model checking

• 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
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 theorem provers
Disadvantages of model checking

• Writing property specifications can be error-prone
  – Use property patterns (or PROPEL)

• May not scale well because of the state space explosion problem
  – Use optimizations for system model translation and/or reasoning engine

• May not generate counterexample traces that are useful for debugging (e.g., too long, too similar to each other)
  – Use A* search with various heuristics
For a chemotherapy case study, 70% reduction in errors reaching patients. [Mertens2012]