Model Checking

Theory, SPIN, Java PathFinder

Yegor Bugayenko

Lecture #9 out of 10
90 minutes

All videos are in this YouTube playlist.

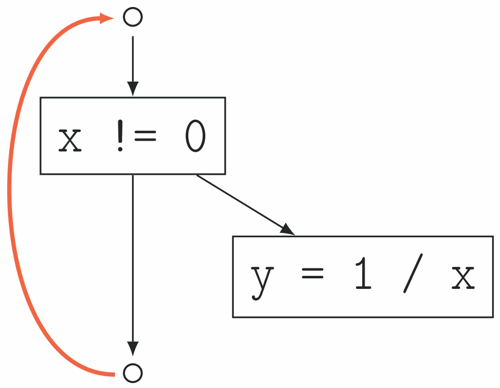
Chapter #1:

Motivating Example

Div by Zero

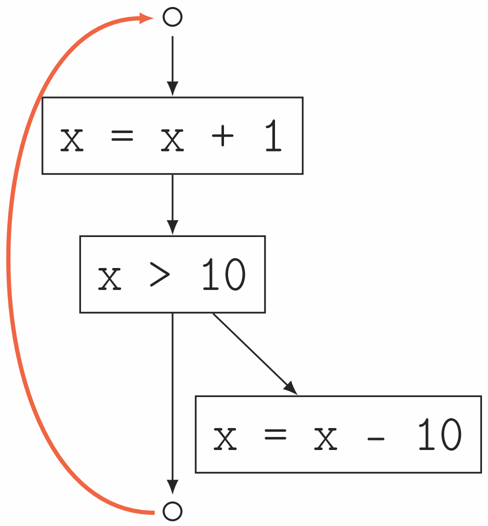
```
// Process no. 1:

extern int x;
extern double y;
int measure() {
  if (x != 0) {
    y = 1.0 / x;
  }
}
```




```
// Process no. 2:

extern int x;
void roll() {
  x += 1;
  if (x > 10) {
    x -= 10;
  }
}
```



Can we detect “division by zero” using symbolic execution? Is “division by zero” the only error here?

ProMeLa (Process Meta Language)

```
extern int x;
extern double y;
int measure() {
  if (x != 0) {
    y = 1.0 / x;
  }
}
void roll() {
  x += 1;
  if (x > 10) {
    x -= 10;
  }
}
```

```
int x; bool dbz;
active proctype measure() {
  do :: true ->
    if
    :: (x != 0) -> dbz = (x == 0)
    :: skip
    fi
  od
}
active proctype roll() {
  do :: true ->
    x = x + 1;
    if
    :: x > 10 -> x = x - 10
    :: skip
    fi
  od
}
```

SPIN (Simple ProMeLa Interpreter)

```
int x; bool dbz;
active proctype measure() {
  do :: true ->
    if
    :: (x != 0) -> dbz = (x == 0)
    :: skip
    fi
  od
}
active proctype roll() {
  do :: true ->
    x = x + 1;
    if
    :: x > 10 -> x = x - 10
    :: skip
    fi;
    printf("x = %d\n", x);
  od
}
```

```
$ spin main.pml | head
          x = 1
          x = 2
          x = 3
          x = 4
          x = 5
          x = 6
          x = 7
          x = 8
          x = 9
          x = 10
$ spin main.pml | tail
...
```

Just checkout this repo and run `make`, the `spin` binary will be compiled.

Monitoring Process

```
int x; bool dbz;
active proctype measure() {
  do :: true ->
    if
    :: (x != 0) -> dbz = (x == 0)
    :: skip
    fi
  od
}
active[2] proctype roll() {
  do :: true ->
    x = x + 1;
    if
    :: x > 10 -> x = x - 10
    :: skip
    fi
  od
}
```

```
active proctype monitor() {
  do :: true ->
    assert(!dbz);
    assert(x >= 0);
  od
}
```

Pay attention to the [2] suffix after the `active` keyword. It tells SPIN to start two instances of the `roll` process.

Fail on Assertion

```
int x; bool dbz;
active proctype measure() {
  do :: true ->
    if
    :: (x != 0) -> dbz = (x == 0)
    :: skip
    fi
  od }
active[2] proctype roll() {
  do :: true ->
    x = x + 1;
    if
    :: x > 10 -> x = x - 10
    :: skip
    fi
  od }
active proctype monitor() {
  do :: true -> assert(!dbz); assert(x >= 0); od
}
```

```
$ spin main.pml
spin: main.pml:22, Error: assertion violated
spin: text of failed assertion: assert((x>=0))
#processes: 4
    x = -9
    dbz = 0
584:  proc  3 (monitor:1) main.pml:22 (state 3)
584:  proc  2 (roll:1) main.pml:17 (state 7)
584:  proc  1 (roll:1) main.pml:18 (state 9)
584:  proc  0 (measure:1) main.pml:9 (state 8)
4 processes created
```

Chapter #2:

The Theory

The Idea

Model checking is a method for checking whether a finite-state model of a system meets a given specification.

1. Represent software as a model
2. Define constraints on the model (using temporal logic)
3. Evaluate the model until constraints are violated/met
4. Refine the model and constraints

The Model

```
$ spin -f "[](p U q)"
never {
T0:
        if
        :: (p) -> goto T0
        :: (q) -> goto accept
        fi;
accept:
        if
        :: ((p) || (q)) -> goto T0
        fi
}
```

```
$ spin -f "[]<>p"
never {
T0:
        if
        :: (true) -> goto T0
        :: (p) -> goto accept
        fi;
accept:
        if
        :: (true) -> goto T0
        fi
}
```

Fig. 3. PROMELA syntax for two LTL formulae.

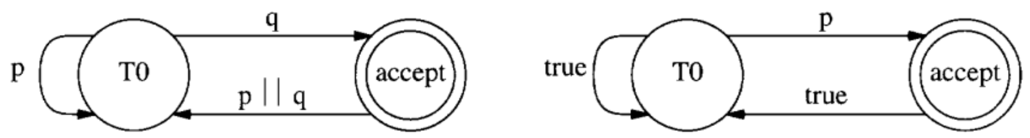
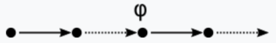
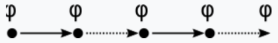
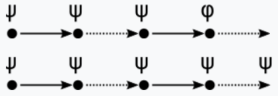
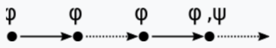


Fig. 4. Büchi automata for the LTL formulae `[] (pUq)` (left) and `[]<> p` (right).

The picture is taken from “The Model Checker SPIN” paper by Gerard J. Holzmann.

Linear Temporal Logic

The semantics for the temporal operators are pictorially presented as follows.

Textual	Symbolic	Explanation	Diagram
Unary operators:			
X ϕ	$\bigcirc\varphi$	ne**X**t: ϕ has to hold at the next state.	
F ϕ	$\Diamond\varphi$	**F**inally: ϕ eventually has to hold (somewhere on the subsequent path).	
G ϕ	$\Box\varphi$	**G**lobally: ϕ has to hold on the entire subsequent path.	
Binary operators:			
ψ **U** ϕ	$\psi\,\mathcal{U}\,\varphi$	**U**ntil: ψ has to hold *at least* until ϕ becomes true, which must hold at the current or a future position.	
ψ **R** ϕ	$\psi\,\mathcal{R}\,\varphi$	**R**elease: ϕ has to be true until and including the point where ψ first becomes true; if ψ never becomes true, ϕ must remain true forever.	
ψ **W** ϕ	$\psi\,\mathcal{W}\,\varphi$	**W**eak until: ψ has to hold *at least* until ϕ; if ϕ never becomes true, ψ must remain true forever.	
ψ **M** ϕ	$\psi\,\mathcal{M}\,\varphi$	Strong release: ϕ has to be true until and including the point where ψ first becomes true, which must hold at the current or a future position.	

Chapter #3:

Model-less Model Checking

Race Condition

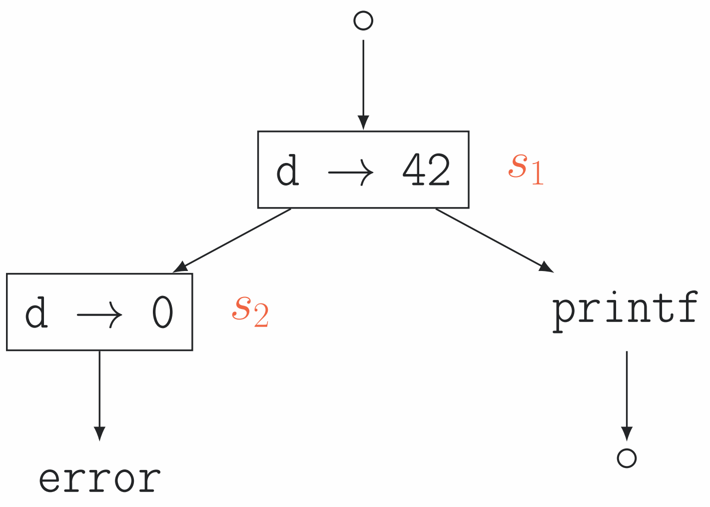
A race condition is the condition of where the system's substantive behavior is dependent on the sequence or timing of other uncontrollable events.

```
public class Race {
  static int d = 42;
  public static void main (String[] args)
    throws Exception {
    new Thread(
      () -> {
        d = 0;
      }
    ).start();
    System.out.printf("x = %d\n", 420 / d);
  }
}
```

```
$ javac Race.java
$ while true; do java Race; done
x = 10
x = 10
x = 10
x = 10
Exception in thread "main"
  java.lang.ArithmeticException: / by zero
  at Race.main(Race.java:9)
x = 10
x = 10
^C
```

States and Their Explosion

```
public class Race {
  static int d = 42;
  public static void main (String[] args)
    throws Exception {
    new Thread(
      () -> {
        d = 0;
      }
    ).start();
    System.out.printf("x = %d\n", 420 / d);
  }
}
```



As the number of state variables in the system increases, the size of the system state space grows exponentially. This is called the state explosion problem.

Java PathFinder

```
$ java -jar build/RunJPF.jar src/examples/Race.jpf
JavaPathfinder core system v8.0 (rev 3408119d115e539956a3d920e22e856e05bb9d23)
- (C) 2005-2014 United States Government. All rights reserved.

==================================== system under test
Race.main()

==================================== search started: 4/21/23 5:43 AM
x = 10

==================================== error 1
gov.nasa.jpf.listener.PreciseRaceDetector
race for field Race.d
  main at Race.main(Race.java:9)
    "System.out.printf("x = %d\n", 420 / d);"  READ:  getstatic Race.d
  Thread-1 at Race.lambda$main$0(Race.java:6)
    "d = 0;"  WRITE: putstatic Race.d

==================================== trace #1
------------------------------------------------------- transition #0 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"ROOT" ,1/1,isCascaded:false}
      [6345 insn w/o sources]
  Race.java:2                    : static int d = 42;
  Race.java:1                    : public class Race {
      [1 insn w/o sources]
  Race.java:4                    : new Thread(
      [145 insn w/o sources]
  Race.java:8                    : ).start();
      [1 insn w/o sources]
------------------------------------------------------- transition #1 thread: 1
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"START" ,2/2,isCascaded:false}
      [3 insn w/o sources]
  Race.java:6                    : d = 0;
------------------------------------------------------- transition #2 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_CLASS" ,1/2,isCascaded:false}
      [2 insn w/o sources]
  Race.java:9                    : System.out.printf("x = %d\n", 420 / d);
------------------------------------------------------- transition #3 thread: 0
gov.nasa.jpf.vm.choice.ThreadChoiceFromSet {id:"SHARED_CLASS" ,1/2,isCascaded:false}
  Race.java:9                    : System.out.printf("x = %d\n", 420 / d);

==================================== results
error #1: gov.nasa.jpf.listener.PreciseRaceDetector
  "race for field Race.d   main at Race.main(Race.jav..."

==================================== statistics
elapsed time:       00:00:00
states:             new=6,visited=0,backtracked=2,end=1
search:             maxDepth=4,constraints=0
choice generators:  thread=5 (signal=0,lock=1,sharedRef=2,...
heap:               new=741,released=22,maxLive=722,gcCycles=4
instructions:       6546
max memory:         491MB
loaded code:        classes=83,methods=1817

==================================== search finished: 4/21/23 5:43 AM
```

What about ChatGPT?

Do you see any problems with this Java code:

```
public class Foo {
 static int d = 42;
 public static void main (String[] args)
  throws Exception {
  new Thread(
   () -> {
    d = 0;
   }
  ).start();
  System.out.printf("x = %d\n", 420 / d);
 }
}
```

Yes, there is a problem with this Java code.

The issue is that the code is creating a new thread and updating the value of the static variable 'd' inside that thread. At the same time, the main thread is trying to access the value of 'd' to perform a division operation.

Since the main thread and the new thread are executing concurrently, there is no guarantee that the new thread will finish updating the value of 'd' before the main thread tries to access it. If the new thread updates 'd' to 0 before the division operation in the main thread, the code will throw a division-by-zero exception.

To avoid this issue, you can use synchronization or locks to ensure that the updates to 'd' are completed before it is accessed in the main thread. Alternatively, you can use Java's volatile keyword to ensure that changes to 'd' made by one thread are visible to other threads immediately. Here's an example using synchronization:

typescript Copy code

```
public class Foo {
  static int d = 42;
  public static void main (String[] args)
    throws Exception {
    Object lock = new Object();
    new Thread(
      () -> {
        synchronized(lock) {
          d = 0;
        }
      }
    ).start();
    synchronized(lock) {
      System.out.printf("x = %d\n", 420 / d);
    }
  }
}
```

This code uses an object 'lock' to synchronize access to 'd'. The new thread acquires the lock, updates 'd', and then releases the lock. The main thread also acquires the lock before accessing 'd', ensuring that it sees the updated value. This way, the code avoids the possibility of a division-by-zero exception.

Chapter #4:

Further Reading/Watching

Introduction lecture by Joost-Pieter Katoen

A Primer on Model Checking by Mordechai Ben-Ari

The Model Checker SPIN by Gerard J. Holzmann