```
main([Ljava/lang/String;)V
  1
   1
    assumptions
        Value at static java.lang.System.out is defined
    Value at static java.lang.System.out.buffer
      = Value at static java.lang.System.out.buffer + Hello World!<nl>
    No return value.
```

Assumptions about input in **brown**

Description of output in **blue**

Loop of clear() method

#12_iload_1<Alpha>

1

1

assumptions

i is greater than or equal to 0

i is less than length of the Array x

op0 is less than 10

x is defined

x[i] = 0

i = i + 1

op0 = i + 1

```
ItsAWrap
 <init>()V
  1
   1
    assumptions
     No return value.
 clear([I)V
  1
   1
    assumptions
          length of the Array x is greater than or equal to 10
          x is defined
          i^ is greater than or equal to 10
     output heap of #12_iload_1<Alpha>()
     No return value.
  #12_iload_1<Alpha>
   1
    1
     assumptions
          i is greater than or equal to 0
          i is less than length of the Array x
          op0 is less than 10
          x is defined
       x[i] = 0
       i = i + 1
       op0 = i + 1
```

Assumptions on loop input state in **green**

Conjectures about loop output state in **red**

Task of automation theorem prover (ACL2) to prove: **green** implies **red**

```
main([I)Z
  1
   1
    assumptions
        length of the Array x is greater than or equal to 10
        x is defined
        x[5]^ equals x[6]^
    specializations
     1
          [main(int[]): returns] --> 1
     output heap of #12_iload_1<Alpha>()
     Returns boolean: [main(int[]): returns]
```

Precise statement x[5]==x[6] (All this stuff generated automatically)

```
(defthm |x[5]^ equals x[6]^ theorem|
  (implies (and (array-p x 'integerp 1)
                (heap-p H)
                (heapObject-p nH)
                (stack-p S)
                (staticArea-p SA)
                (|length of the Array x is greater than or equal to 10| x H)
                (|x is defined| x))
           (|x[5]^ equals x[6]^| x H nH S SA)))
```

Proof ……(lots of stuff here)

```
But simplification reduces this to T, using the :definitions FRAME-5,
MV-NTH, PUSHSTACK-1, |calling stack|,
|length of the Array x is greater than or equal to 10|,
```

and so on…, until at last

Q.E.D.

```
The storage of |x[5]^ equals x[6]^ theorem| depends upon the :type-
prescription rule |x[5]^ equals x[6]^|.

Summary
Form:  ( DEFTHM |x[5]^ equals x[6]^ theorem| ...)
Rules: ((:DEFINITION FRAME-5)
        (:DEFINITION MV-NTH)
        (:DEFINITION PUSHSTACK-1)
```

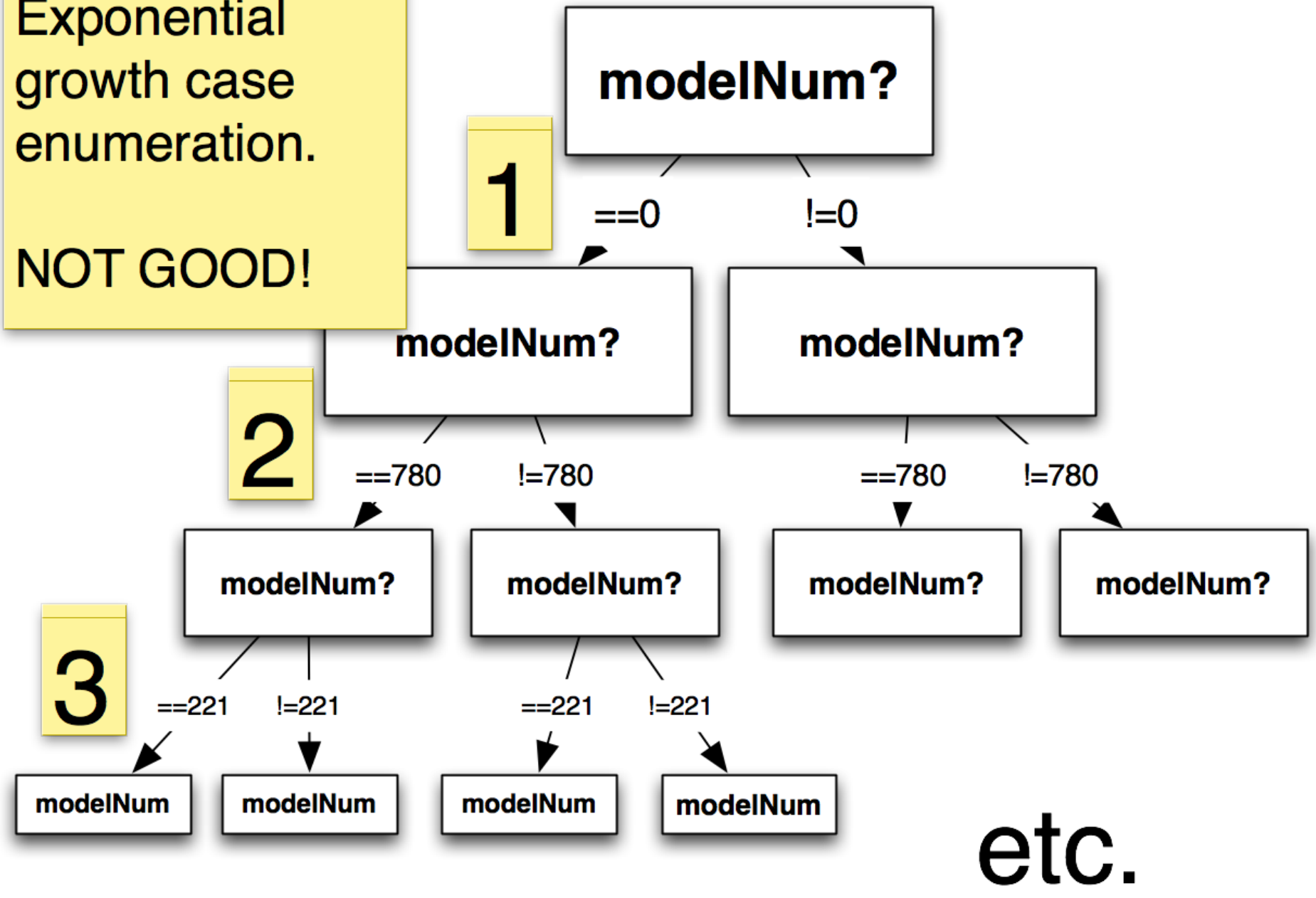
lots more rules were used ……

```
Warnings:  Non-rec
Time:  0.00 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other: 0.00)
 |x[5]^ equals x[6]^ theorem|
ACL2 !>>Bye.
 :EOF
```

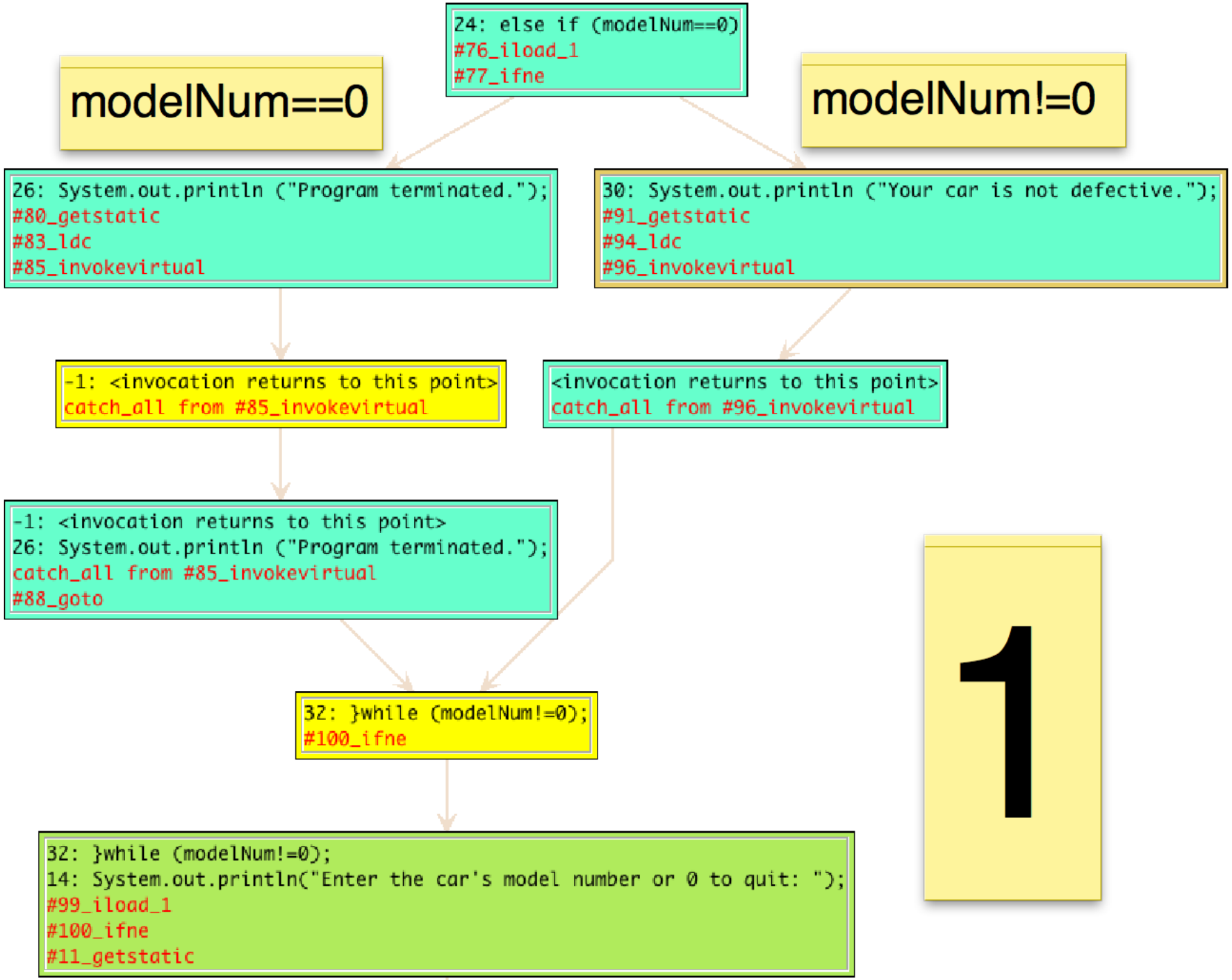
ACL2 !> ACL2 is a theorem prover developed at UT Austin.

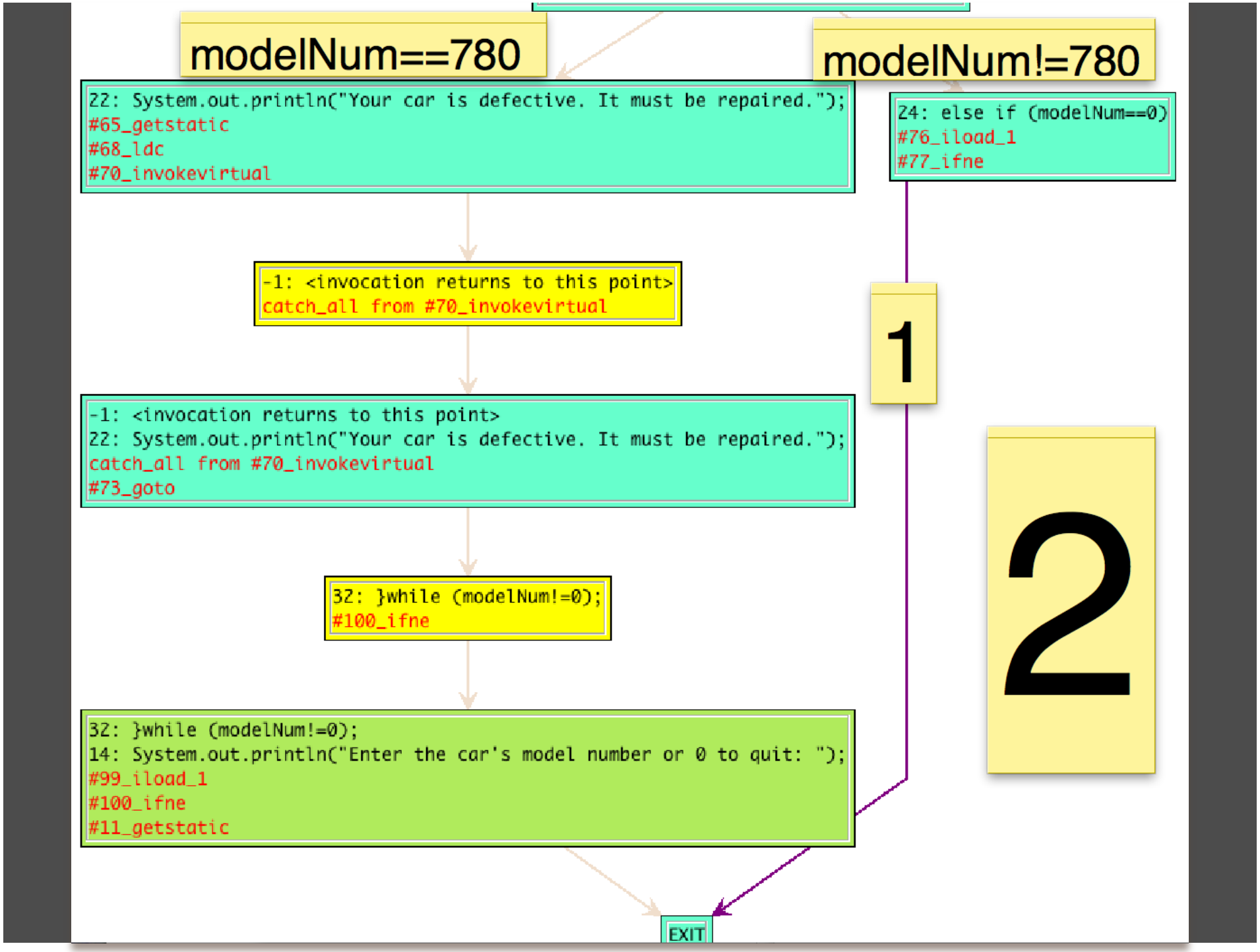
Exponential growth case enumeration.

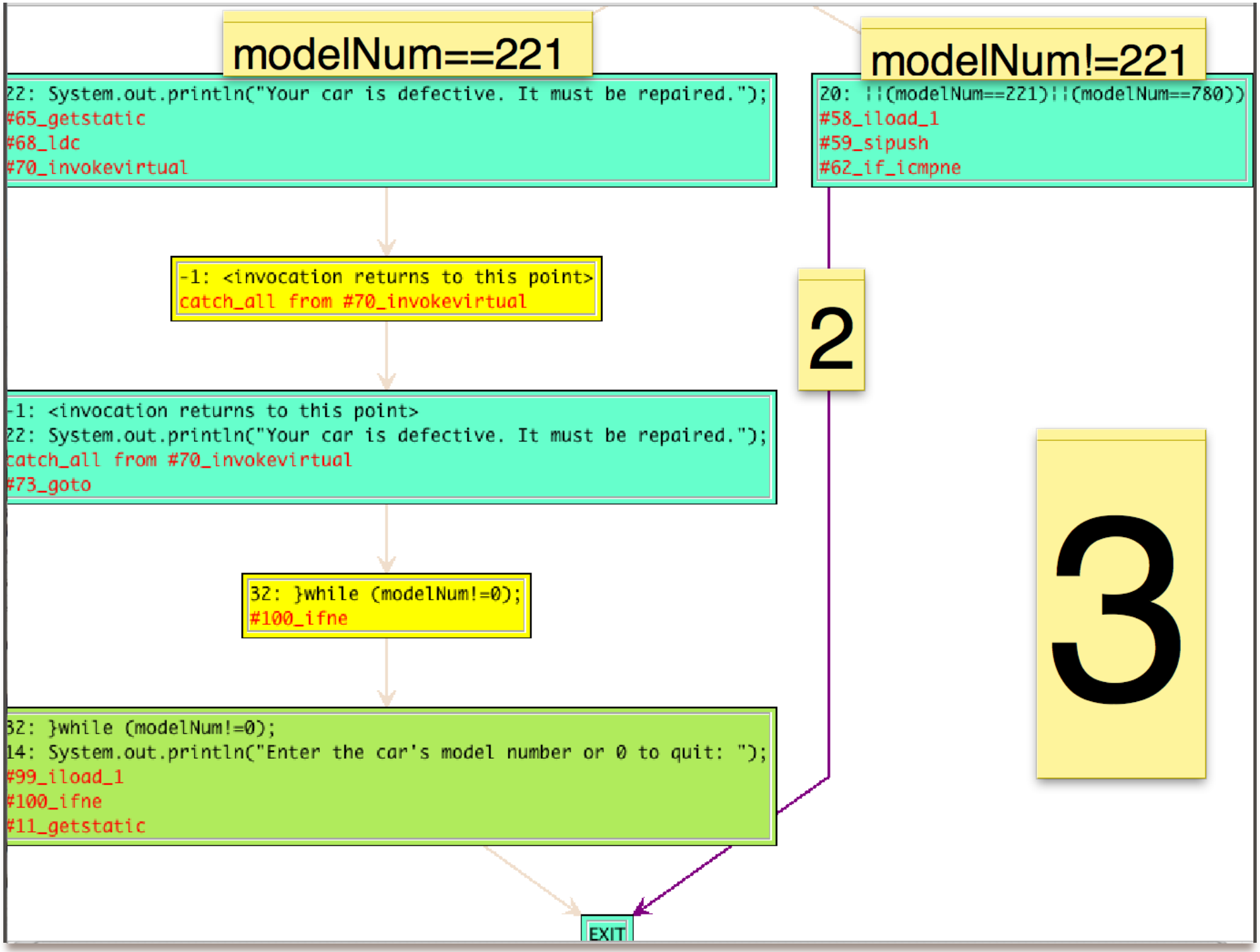
NOT GOOD!



etc.


24: else if (modelNum==0)
#76_iload_1
#77_ifne
modelNum==0
modelNum!=0
26: System.out.println ("Program terminated.");
#80_getstatic
#83_ldc
#85_invokevirtual
30: System.out.println ("Your car is not defective.");
#91_getstatic
#94_ldc
#96_invokevirtual
-1: <invocation returns to this point>
catch_all from #85_invokevirtual
<invocation returns to this point>
catch_all from #96_invokevirtual
-1: <invocation returns to this point>
26: System.out.println ("Program terminated.");
catch_all from #85_invokevirtual
#88_goto
32: }while (modelNum!=0);
#100_ifne
32: }while (modelNum!=0);
14: System.out.println("Enter the car's model number or 0 to quit: ");
#99_iload_1
#100_ifne
#11_getstatic
1


modelNum==780
modelNum!=780
22: System.out.println("Your car is defective. It must be repaired.");
#65_getstatic
#68_ldc
#70_invokevirtual
24: else if (modelNum==0)
#76_iload_1
#77_ifne
-1: <invocation returns to this point>
catch_all from #70_invokevirtual
1
-1: <invocation returns to this point>
22: System.out.println("Your car is defective. It must be repaired.");
catch_all from #70_invokevirtual
#73_goto
2
32: }while (modelNum!=0);
#100_ifne
32: }while (modelNum!=0);
14: System.out.println("Enter the car's model number or 0 to quit: ");
#99_iload_1
#100_ifne
#11_getstatic
EXIT


modelNum==221
modelNum!=221
22: System.out.println("Your car is defective. It must be repaired.");
#65_getstatic
#68_ldc
#70_invokevirtual
20: ||(modelNum==221)||(modelNum==780))
#58_iload_1
#59_sipush
#62_if_icmpne
-1: <invocation returns to this point>
catch_all from #70_invokevirtual
2
-1: <invocation returns to this point>
22: System.out.println("Your car is defective. It must be repaired.");
catch_all from #70_invokevirtual
#73_goto
3
32: }while (modelNum!=0);
#100_ifne
32: }while (modelNum!=0);
14: System.out.println("Enter the car's model number or 0 to quit: ");
#99_iload_1
#100_ifne
#11_getstatic
EXIT

caseSplits

[1.1.1.1.1.1.1.r.1]1 more instruction to firstYearCode.CarRecall.main([Ljava/lang/String;)V#77_ifne

1

1

assumptions

modelNum does not equal 0

Value at static java.lang.System.out is defined

Value at static java.lang.System.out.buffer = Value at static java.lang.Sys ffer + Your car is not defective.<nl>

op0 = Value at static java.lang.System.out

modelNum!=0

1

[1.1.1.1.1.1.r.1](2)firstYearCode.CarRecall.main([Ljava/lang/String;)V#62_if_icmpne

1

1

assumptions

modelNum equals 780

Value at static java.lang.S

modelNum==780

specializations

1

var2 --> Your car is defective. It must be repaired.

var1 --> Your car is defective. It must be repaired.<nl>

2

assumptions

modelNum does not equal 0

modelNum does not equal 780

Value at static java.lang.System.ou

modelNum!=780

specializations

1

var2 --> Your car is not defective.

var1 --> Your car is not defective.<nl>

Value at static java.lang.System.out.buffer = Value at static java.lang.System.out.buffer + var1

op0 = Value at static java.lang.System.out

2

[1.1.1.1.1.r.1](2)firstYearCode.CarRecall.main([Ljava/lang/String;)V#55_if_icmpeq

1

1

assumptions

<[1.1.1.1.1.1.r.1](2)firstYearCode.CarRecall.main([Ljava/lang/String;)V#62_if_icmpne.1 (root.1_0)>

()

modelNum does not equal 221

specializations

1

var2 --> var2

var1 --> var1

modelNum!=221

2

assumptions

modelNum equals 221

Value at static java.lang.System.out is defined

specializations

1

var2 --> Your car is defective. It must be repaired.

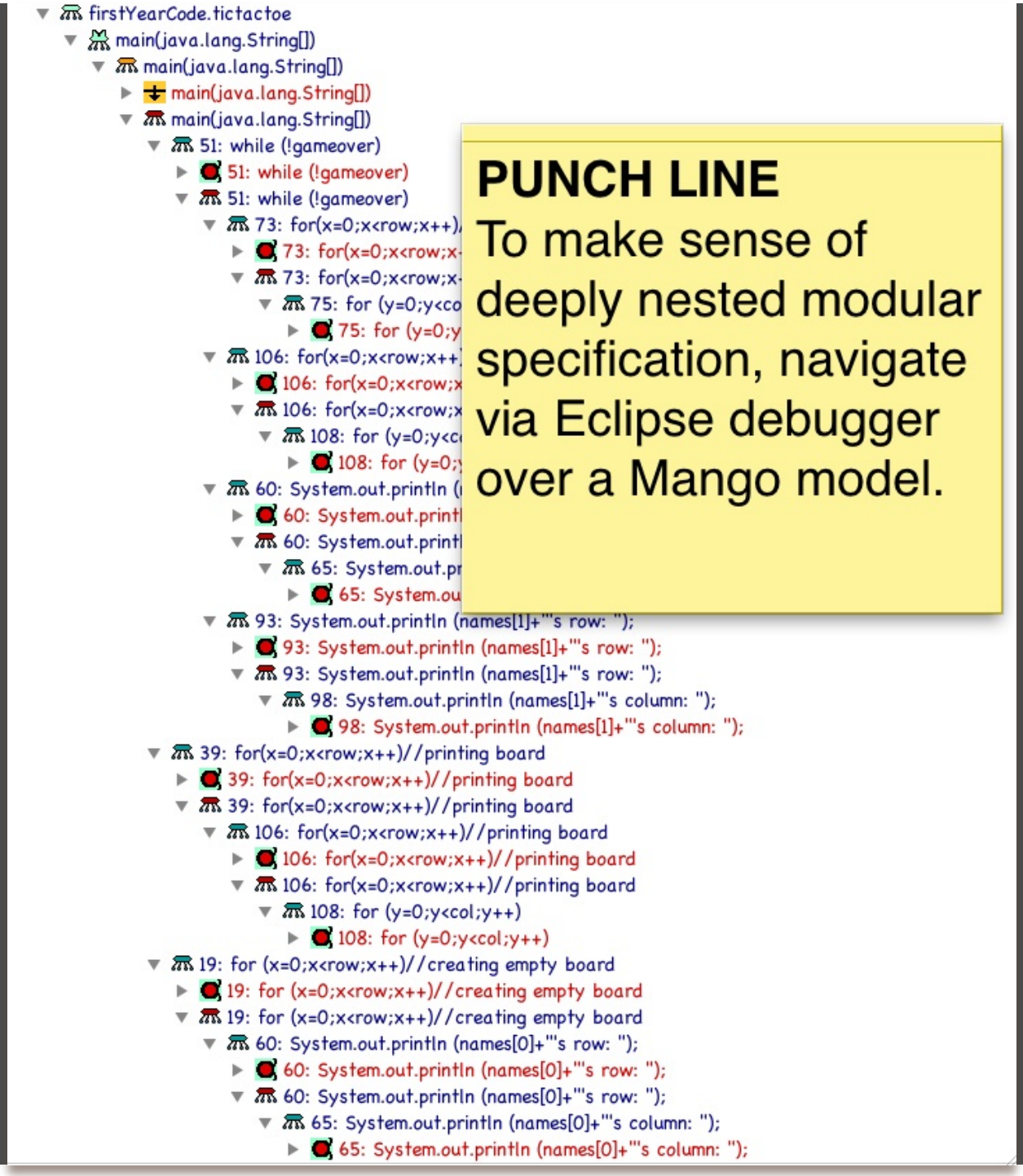
var1 --> Your car is defective. It must be repaired.<nl>

modelNum==221

Value at static java.lang.System.out.buffer = Value at static java.lang.System.out.buffer + var1

op0 = Value at static java.lang.System.out

3



PUNCH LINE

To make sense of deeply nested modular specification, navigate via Eclipse debugger over a Mango model.

Debugger perspective with Mango semantics

Go/pause buttons control mango build

debug | **mango**

The mango view appears as a peer of the standard debug view. Selecting the mango tab switches to mango semantics. Instead of the call chain, the mango view shows the mango subdivision hierarch of the targeted code.

variables | breakpoints | assumptions | conjectures

selected tab will expose relevant part of mango model, in a format parallel to existing debugger views

full browser capbility (backwards/ fowards hyperlinks) for mining additional info, such as input state for conjectures

- breakpoints now control mango model build
- pc may be moved to any point where model already built, synced with vars, asms conjectures, and selected subdivision

Source Code View

retains standard Eclipse pop-up menus

console | rulebase | modules | shadows

Console shows existing mango stream
rulebase modules and and shadows expose
views of corresponding mango persistent artifact