Modbat

A Model-Based API Tester for Event-Driven Systems

Cyrille Artho, Armin Biere, Masami Hagiya
Eric Platon, Martina Seidl, Yoshinori Tanabe, Mitsuharu Yamamoto

HVC 2013

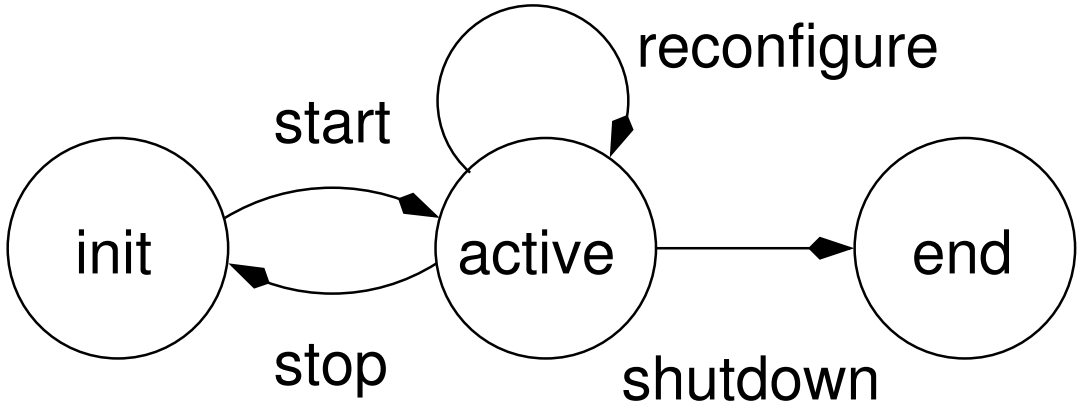
Haifa Verification Conference

IBM Research, Haifa, Israel

Tuesday, November 5, 2013

`http://staff.aist.go.jp/c.artho/modbat`
`http://fmv.jku.at/modbat`

Example




```
"init" -> "active" := { c = new Component; c.start; }
                      ...
```

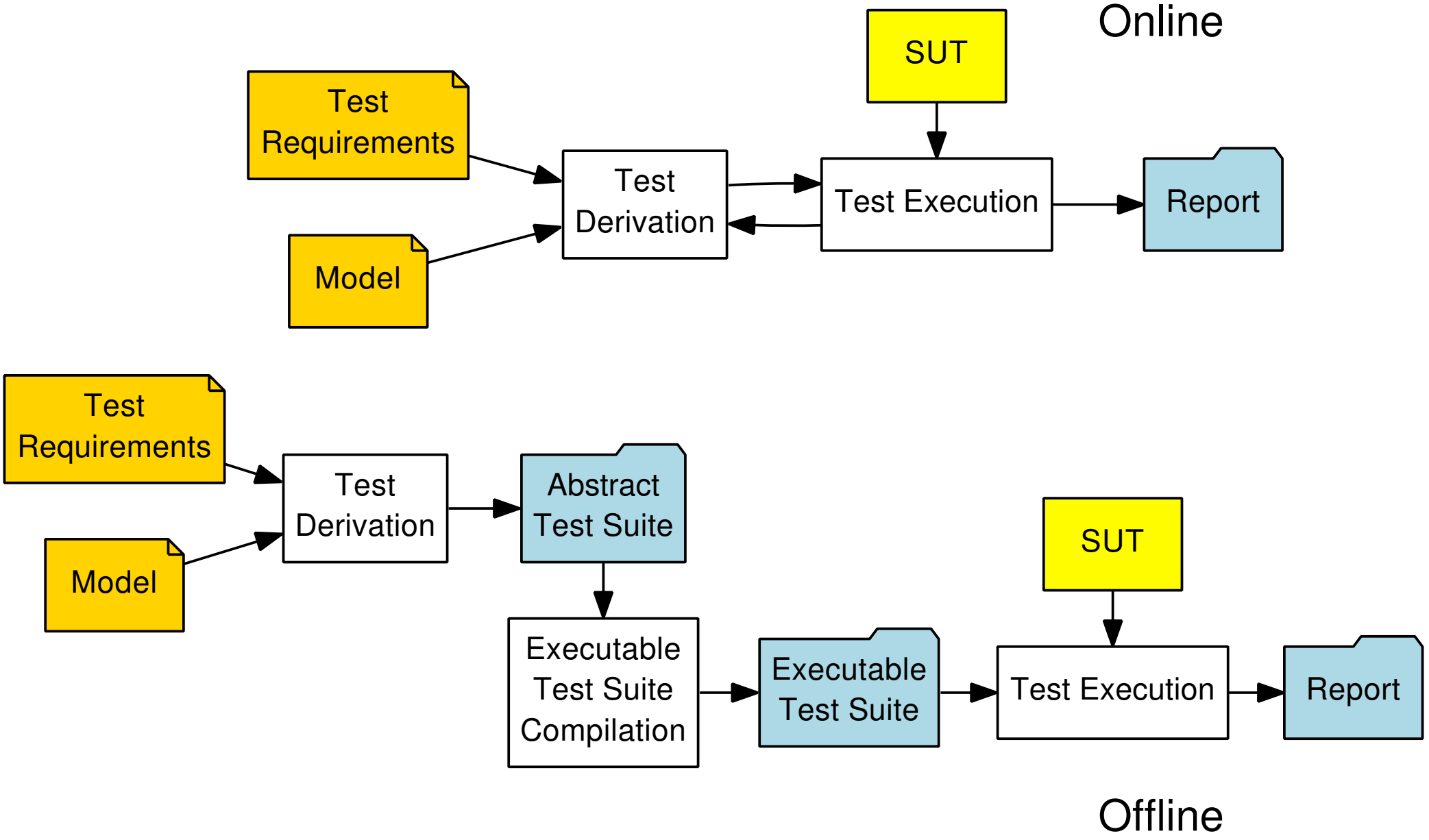
specify model for testing as *extended finite state machine*

randomly explore model to produce test cases as traces

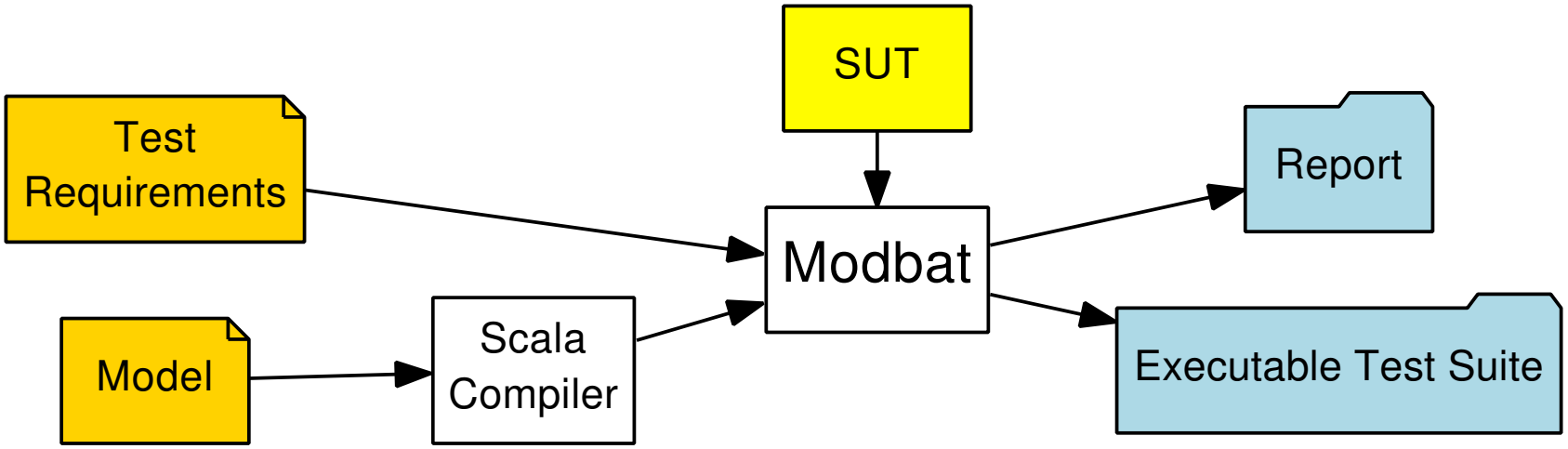
execute traces on *system under test*

dump failing traces for replay and debugging

Online vs Offline



Model written in *Domain Specific Language* (DSL) embedded in Scala



Execution on *Java Virtual Machine*

support for C through JNI

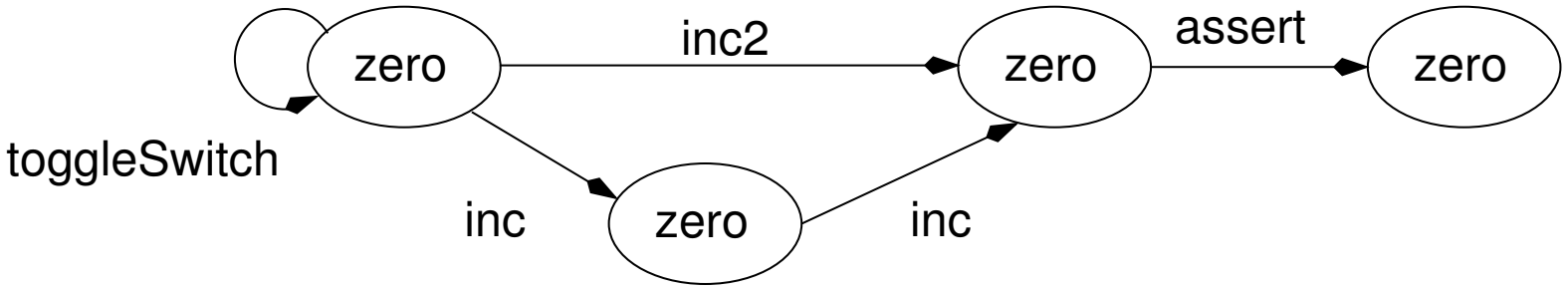
report contains state & transition coverage

executable test suite compatible with JUnit

Counter Example


```
// Model (embedded Scala DSL)
//
class CounterModel extends Model {
  var counter = new SimpleCounter ()
  def instance () = {
    new MBT (
      "zero" -> "zero" := {
         counter.toggleSwitch
      },
      "zero" -> "one" := {
         counter.inc
      }
      "one" -> "two" := {
         counter.inc
      }
      "zero" -> "two" := {
         counter.inc2
      },
      "two" -> "end" := {
        assert (counter.value == 2)
      }
    )
  }
}
```

```
// SUT (Java)
//
public cass Simple Counter {
  int count = 0;
  boolean flag = true;
  public void toggleSwitch () {
    flag = !flag;
  }
  public void inc () {
    if (flag) count += 1;
  }
  public void inc2 () {
    count += 2;
  }
  public int value () {
    return count;
  }
}
```



Exceptions and Non-Determinism

- default transition template
```
"pre" -> "post" := { action }
```
- *required* exceptions
```
{ action }
 throws ("Exception1")
```
- *optional* exceptions
```
"pre" -> "post" := { action }
  catches ("Exception" -> "exceptionState")
```
- `Choose` used to pick transition based on random number in a range
- `Maybe` picks transition with random probability of 50%
- conditional next state

```
"accepting" -> "connected" := {
    connection = ch.accept ()
} nextIf ({ () => connection == null} -> "accepting")
```

Lines of Code per Feature

Modbat	ModelJUnit	ScalaCheck

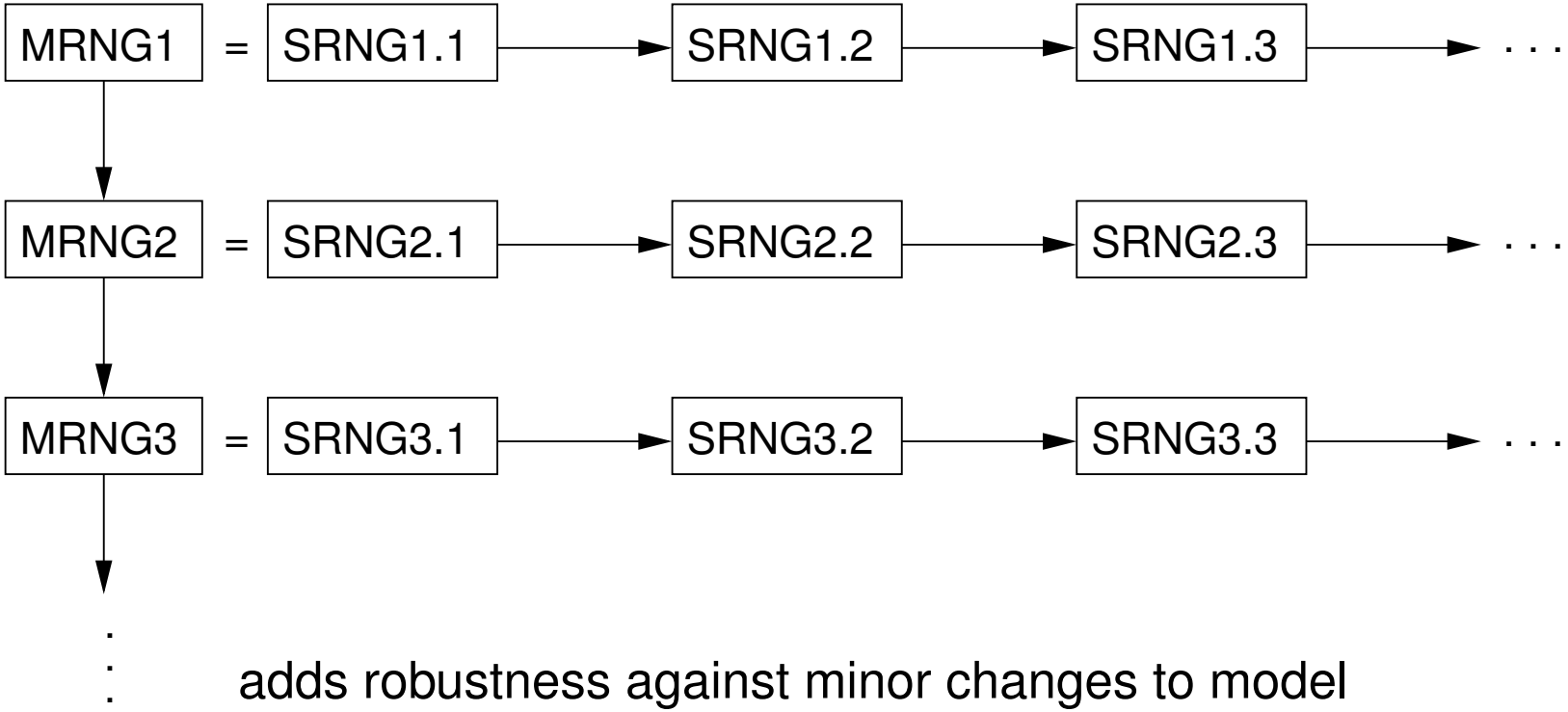
`"pre" -> "post" :=`
`{ action }`
	`boolean action0Guard() {`
	`  return state == 0; }`
	`@Action void action0() {`
	`  action();`
	`  state = 1; }`
		`case object Transition0`
		`  extends Command {`
		`preConditions += (s==...)`
		`def run(s:  State) = action`
		`def nextState(s:State)=...`

Modbat	ModelJUnit	ScalaCheck

`{ action }`
`throws("Exception")`
	`{ boolean ok = false;`
	`try {`
	`  action();`
	`} catch(Exception e){`
	`  ok = true; }`
	`  assert (ok); }`
		`{ var ok = false`
		`try {`
		`  action`
		`} catch { case e:`
		`  Exception => ok=true }`
		`  assert (ok) }`

generation of test runs *deterministic*

unless SUT non-deterministic



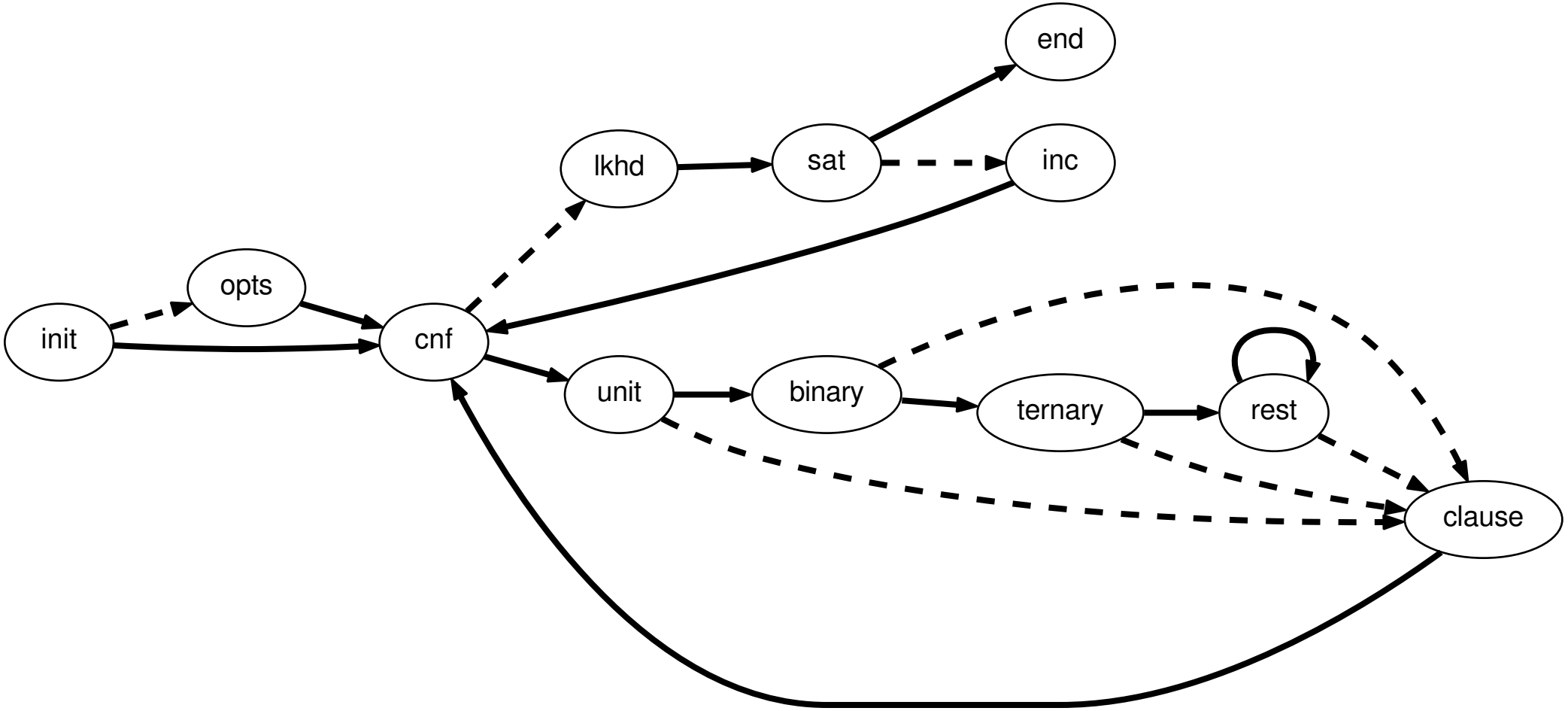
adds robustness against minor changes to model

enables *parallel* or *separate* execution

Lingeling Case Study

- testing API of state-of-the-art SAT solver Lingeling [Biere 2010-2013]
  - 7 medals (incl. 4 gold) in SAT competition 2013
  - more than a dozen in/pre-processing algorithms with complex interactions
  - **incremental API** required in many applications and **error prone** to implement
  - hard to test with file / grammar based fuzzing [BrummayerLonsingBiere-SAT10]
- model based testing with manually coded C model [ArthoBiereSeidl-TAP13]
  - more efficient and effective than regression testing in detecting and shrinking faults
- same case study using Modbat:
  - 12 states + 18 transitions + 300 lines Scala code *vs* 300 (+400) lines C code
  - similar efficiency in detecting faults
  - throughput (test-cases per second) slighty lower (2-4x) Java/JNI overhead
  - no shrinking / delta debugging (yet) future work

Lingeling API Model Visualization



very coarse grained visualization actually

“sat” action is made of 124 lines of code

model needs refactorization

```
001  import modbat.mbt.Predef._
002  import modbat.mbt.MBT._
003  import modbat.mbt.MBT

004  import com.sun.jna.Pointer;
005  import com.sun.jna.Structure;
006  import com.sun.jna.ptr.IntByReference;
007  import com.sun.jna.ptr.PointerByReference;
008  import java.util.UUID;
009  import java.util.ArrayList;

010  object LingelingModelInc {

011          var lglAPI : LGLJavaAPI  = new LGLJavaAPI();
012          var lgl : LGLJavaAPI.LGLib = null;
013          var l : LGLJavaAPI.LGLib.LGL = null;

014    def instance() = {
015      var num_variables : Int = 0
016      var num_clauses : Int = 0
017      var created_clauses : Int = 0
018      var added_clauses : Int = 0
019      var available : Array [Int] = new Array [Int](0)
020      var frozen : Array [Int] = new Array [Int](0)
021      var navailable : Int = 0
022      var nfrozen : Int = 0
023      var i = 0
024      var next : Int = 0

025      def lit () : Int = {
026        var sel: Int = choose (0, navailable);
027        var l = available(sel);
028        var pol: Int = choose (0,2);
029        if (pol == 1) l = -l;
```

```
030         return l;
031       }

032       def gcd (a1 : Int, b1 : Int) : Int = {
033         var r : Int = 0;
034         var a : Int = a1;
035         var b : Int = b1;
036         while (b > 0) {
037           r = a%b;
038           a = b;
039           b = r;
040         }
041         return a;
042       }

043       new MBT (
044           "init" -> "cnf" := {
045               lgl = lglAPI.getLibAccess ();
046               l = lglAPI.initLgl (lgl,"modbattmp-"+UUID.randomUUID()+".lgltrace");

047               created_clauses = 0;
048               num_variables = choose (10, 201);
049               var threshold = choose (390, 451);
050               num_clauses = (num_variables * threshold) / 100;

051               available = new Array [Int] (num_variables);
052               navailable = num_variables;

053               for (i <- 0 to num_variables-1) {
054                 available(i) = i+1;
055               }
056               i = choose (0,2);
057           } nextIf ({ () => i == 1 } -> "opts"),
```

```
058         "opts" -> "cnf" := {
059           i = choose (0,10);
060           if (i == 0) lgl.lglsetopt (l,"plain",1);
061           else {
062             var n, m : Int = 0;
063             var v : IntByReference = new IntByReference ();
064             var min : IntByReference = new IntByReference ();
065             var max : IntByReference = new IntByReference ();
066             var name : PointerByReference = new PointerByReference ();
067             var opt : Pointer = null;
068             var next : Pointer = lgl.lglfirstopt(l);
069             next = lgl.lglnextopt(l, next, name, v, min, max);
070             while ( next != null ) {
071               next = lgl.lglnextopt(l, next, name, v, min, max);
072               n = n+1;
073             }
074             m = choose (2, 10);
075             lgl.lglsetopt(l,"verbose",-1);
076             lgl.lglsetopt(l,"drup",-1);
077             next = lgl.lglfirstopt(l);
078             next = lgl.lglnextopt(l, next, name, v, min, max);
079             while (next != null) {
080               if (choose (1, m) == 1) {
081                 var nameV = name.getValue ().getString(0);
082                 if (!nameV.equals("log") && !nameV.equals("check") &&
083                     !nameV.equals("verbose") && !nameV.equals("sleeponabort") &&
084                     !nameV.equals("witness")) {
085                   if (choose (0,2) == 1) {
086                     while ((choose (0,2) == 1) && (v.getValue() < max.getValue())
087                             && v.getValue() < (Integer.MAX_VALUE / 2)) {
088                       if (v.getValue() < 4) v.setValue (v.getValue()+1);
089                       else v.setValue (v.getValue() * 2);
090                     }
091                     if (v.getValue () > max.getValue()) v.setValue (max.getValue);
```

```
092                 } else {
093                   while ((choose (0,2) == 1) && (v.getValue() < min.getValue())
094                           && v.getValue() > (Integer.MIN_VALUE / 2)) {
095                     if (v.getValue > 0) v.setValue(v.getValue() / 2);
096                     if (v.getValue() > -4) v.setValue (v.getValue()-1);
097                     else v.setValue (v.getValue() * 2);
098                   }
099                   if (v.getValue () > min.getValue()) v.setValue (min.getValue);
100                 }
101                 lgl.lglsetopt(l, name.getValue().getString(0), v.getValue());
102               }
103             }
104             next = lgl.lglnextopt(l, next, name, v, min, max);
105           }
106         }
107       },
108       "cnf" -> "unit" := {
109       } nextIf ({ () => (num_clauses <= created_clauses ) } -> "lkhd"),
110       "unit" -> "binary" := {
111         var literal : Int = lit ();
112               lgl.lgladd(l, literal);
113               i = choose (0, 100)
114       } nextIf ({ () => (i == 0) } -> "clause"),
115       "binary" -> "ternary" := {
116               var literal : Int = lit ();
117               lgl.lgladd(l, literal);
118               i = choose (0, 10)
119       } nextIf ({ () => i == 0} -> "clause"),
120       "ternary" -> "rest" := {
121               var literal : Int = lit ();
122               lgl.lgladd(l, literal);
123               i = choose (0, 3)
124       } nextIf ({ () => i == 0 } -> "clause"),
```

```
125         "rest" -> "rest" :=  {
126                 var literal : Int = lit ();
127                 lgl.lgladd(l, literal);
128                 i = choose (0, 4)
129         } nextIf ({ () => i != 0 } -> "clause"),
130         "clause" -> "cnf" := {
131                 created_clauses = created_clauses+1;
132                 lgl.lgladd(l, 0);
133         },
134         "lkhd"  -> "sat" := {
135                 i = choose (0, 11);
136                 if (i == 0) lgl.lglookahead(l);
137         },
138         "sat" -> "end" := {
139           var delta : Int = 0;
140           var freeze : Int = choose (0,11);
141           var pos : Int = 0;
142           var li : Int = 0;
143           var res : Int = 0;
144           next = 0;
145           var nassumed,szassumed : Int = 0;
146           val  assumed = new ArrayList[java.lang.Integer];

147           if (freeze != 0) {
148             if (navailable > 1) {
149               nfrozen = choose ((navailable+9)/10, 2*(navailable+2)/3);
150               frozen = new Array[Int](nfrozen);
151               delta = choose (1, navailable);
152               while (gcd (navailable, delta) != 1) {
153                 delta = delta + 1;
154                 if (delta == navailable) delta = 1;
155               }

156               pos = choose (0, navailable);
```

```
157               for (i <- 0 to nfrozen-1) {
158                 assert(0 <= pos && pos < navailable);
159                 li = available(pos);
160                 frozen(i) = li;
161                 pos = pos + delta;
162                 if (pos >= navailable) pos = pos - navailable;
163               }
164               for (i <- 0 to nfrozen-1) {
165                 lgl.lglfreeze(l, frozen(i));
166               }
167             } else if (navailable ==  1) {
168               nfrozen = 1;
169               frozen = new Array [Int] (1);
170               frozen (0) = available (0);
171             } else {
172               nfrozen = 0;
173               frozen = new Array [Int] (0);
174             }

175             i = choose (0,4);
176             if ((navailable != 0) && (i == 0)) {
177               nassumed = 0;
178               szassumed = 1;
179               do {
180                 pos = choose (0, navailable);
181                 li = available (pos);
182                 if (choose (0,2) == 0) li = -li;
183                 lgl.lglassume(l,li);
184                 assumed.add(li);
185                 nassumed = nassumed + 1;
186               } while (choose (0,11) == 0);
187             } else {
188               nassumed = 0; szassumed = 0;
```

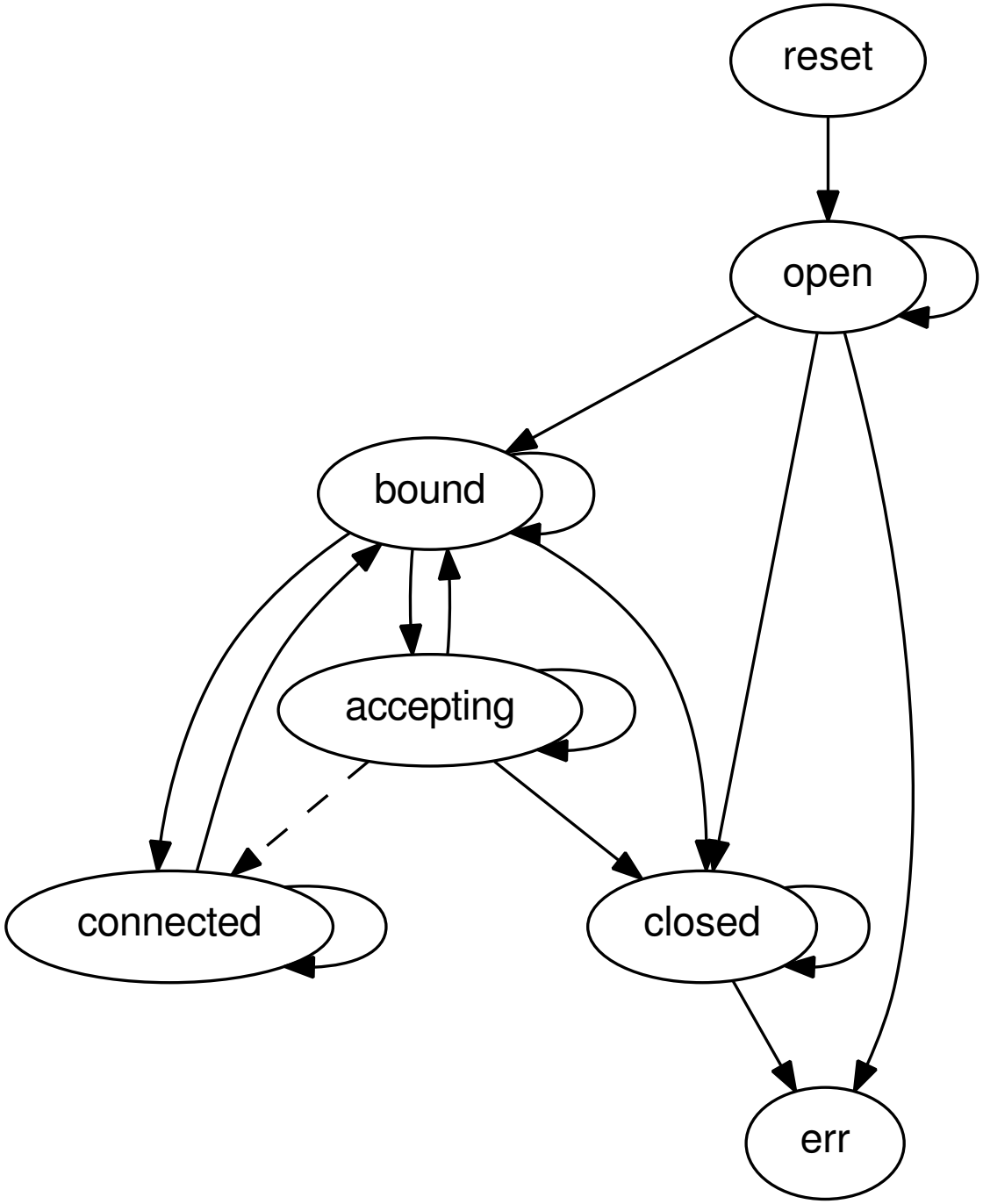
```
189               assumed.clear();
190             }
191             if (choose (0, 5) == 0) {
192               pos = choose (0, navailable);
193               li = available (pos);
194               if (choose (0,2) == 0) li = -li;

195               if (choose(0,4) == 0) lgl.lglresetphase (l, li);
196               else lgl.lglsetphase (l,li);

197             }
198             if (choose (0, 67) == 0) lgl.lglfixate (l);
199             if (choose (0, 21) != 0) res = lgl.lglsat(l);
200             else res = lgl.lglsimp(l, choose(0,11));
201             if (res == 10) {
202               if (choose (0, 5) == 0) lgl.lglinconsistent(l);
203               if (choose (0, 21) == 0) lgl.lglsetphases (l);
204               i = (choose (0, num_variables));
205               while (i > 0) {
206                 i = i-1;
207                 li = choose (1, 2*num_variables);
208                 if (choose (0,2) == 0) li = -li;
209                 lgl.lglderef (l, li);
210               }
211               if (choose (0,31) == 0) lgl.lglsetphases (l);
212               if (freeze != 0) {
213                 navailable = 0;
214                 for (i <- 0 to nfrozen-1) {
215                   li = frozen (i);
216                   (i % 5) match {
217                     case 0 => lgl.lglmelt (l, li);
218                               available (navailable) = li;
219                               navailable = navailable + 1;
220                     case 1 =>
```

```
221                                 available (navailable) = li;
222                                 navailable = navailable + 1;
223                     case 2 =>
224                                 num_variables = num_variables + 1;
225                                 available (navailable) = num_variables;
226                                 navailable = navailable + 1;
227                     case 3 =>
228                                 lgl.lglmelt (l, li);
229                     case _ =>
230                   }

231                 }
232               }
233               if (freeze >= 2) {
234                 num_clauses = (choose (101, 131)  * num_clauses + 99) / 100;
235                 next = 1;
236               }
237               if (choose (0, 5) == 0) lgl.lglchanged (l);
238             } else if (res == 20) {
239               if (nassumed > 0) {
240                 i = choose (0, 3*nassumed / 2);
241                 while (i > 0) {
242                   i = i - 1;
243                   lgl.lglfailed (l, assumed.get(choose (0, nassumed)));
244                 }
245               }
246               if (choose (0,5) == 0) lgl.lglinconsistent (l);
247             }
248             if (choose (0, 8) == 0) lgl.lglflushcache (l);
249           }
250         } nextIf ({ () => next != 0 } -> "inc"),
```

```
251          "inc" -> "cnf" := {
252              var oldavailable : Int = 0;
253              var newvars : Int = 0;

254              if (choose (0,2) != 0) lgl.lglmaxvar (l);
255              while (choose (0,5) != 0) {
256                lgl.lglfixed (l, choose (1, num_variables));
257              }
258              newvars = choose (0, 51);
259              if (newvars != 0) {
260                if (choose (0,2) != 0) lgl.lglincvar (l);
261                oldavailable = navailable;
262                navailable = navailable + newvars;
263                var tmp : Array [Int] = new Array [Int] (oldavailable);

264                for (i <- 0 to oldavailable-1) {
265                  tmp (i) = available (i);
266                }
267                available = new Array [Int] (navailable);

268                for (i <- 0 to oldavailable - 1 ) {
269                  available (i) = tmp (i);
270                }
271                for (i <- 0 to newvars-1) available (oldavailable+i) = num_variables+i+1;
272                num_variables = num_variables + newvars;
273              }
274          }
275      )
276    }
277  }
```

- Asynchronous / Non-Blocking I/O
  - goes back to the `select()` function in Unix , `java.nio` similar
  - allows to wait for multiple messages on multiple sockets
    single threaded and without polling
- Java Path Finder (JPF) executing Modbat
  - JPF software model checker for Java bytecode generates all thread schedules
  - JPF can not handle I/O, so used our own *model library* for `java.nio`
  - JPF in principle would allow to explore non-determinism in both model and SUT
  - Modbat deterministic thus JPF only used to remove SUT non-determinism
  - JPF execution much slower ($>$ 1000x)
- model based testing of our `java.nio` *model library* as example for I/O driven system
  library can be used for model checking with JPF too (not the focus here)

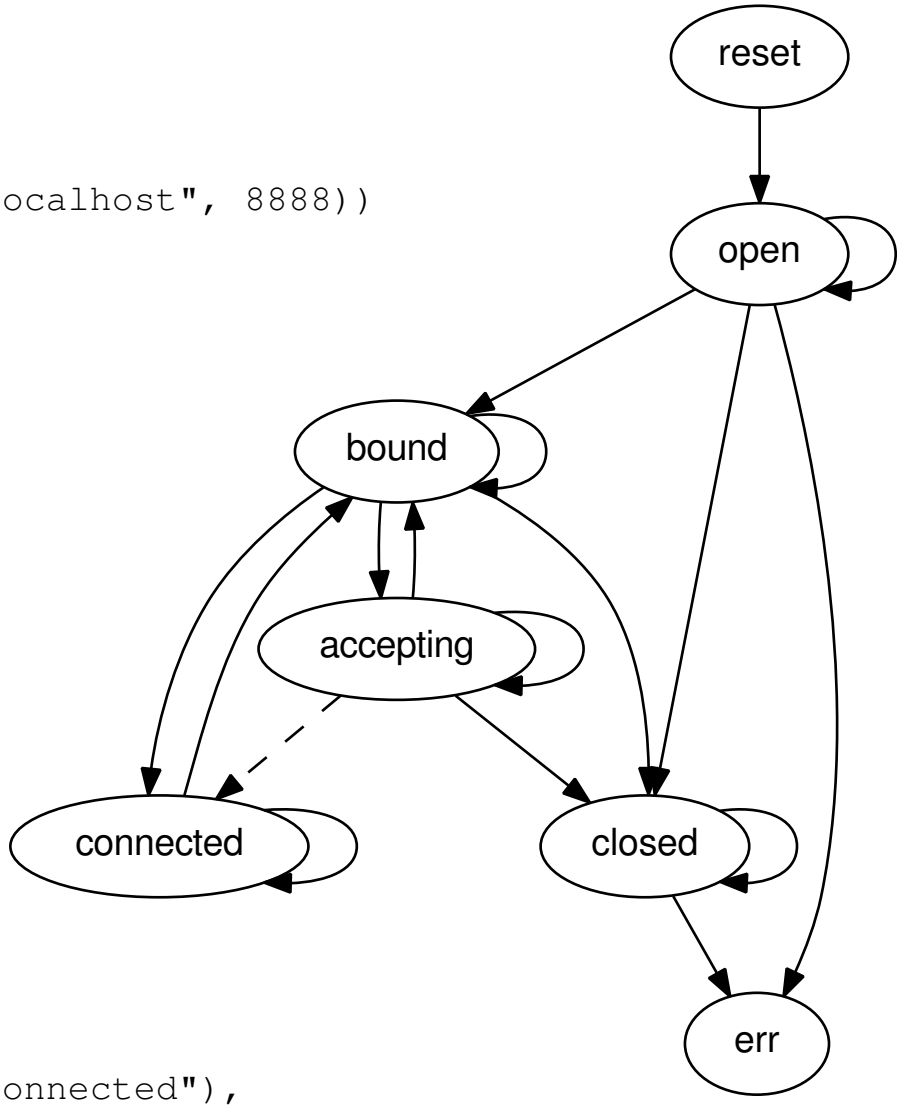
Java.NIO Server Model Visualization



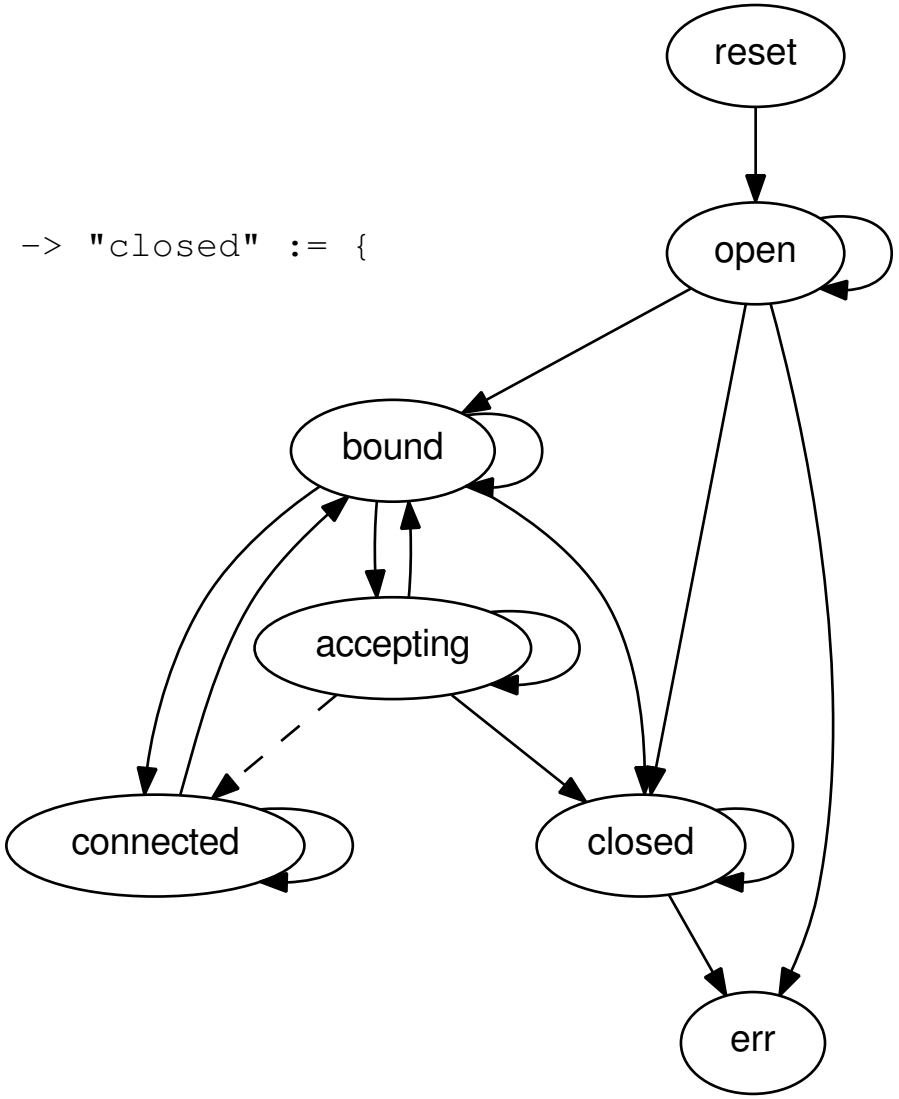
```
001  package modbat.examples

002  import modbat.mbt._
003  import modbat.mbt.Predef._
004  import java.io.IOException
005  import java.net.InetSocketAddress
006  import java.nio.ByteBuffer
007  import java.nio.channels.ClosedByInterruptException
008  import java.nio.channels.ServerSocketChannel
009  import java.nio.channels.SocketChannel

010  object JavaNioServerSocket extends Model {
011    var ch: ServerSocketChannel = null
012    var connection: SocketChannel = null
013    var client: TestClient = null

014    class TestClient extends Thread {
015      override def run() {
016        try {
017          val connection = SocketChannel.open()
018          connection.connect(new InetSocketAddress("localhost", 8888))
019          val buf = ByteBuffer.allocate(2)
020          buf.asCharBuffer().put("\n")
021          connection.write(buf)
022          connection.close()
023        } catch {
024          case e: ClosedByInterruptException => {
025            if (connection != null) {
026              connection.socket().close()
027            }
028          }
029        }
030      }
031    }
```

```
032     def toggleBlocking(ch: ServerSocketChannel) {
033       ch.configureBlocking(!ch.isBlocking())
034     }

035     @after def cleanup() {
036       if (connection != null) {
037         connection.close()
038         connection = null
039       }
040       if (ch != null) {
041         ch.close()
042         ch = null
043       }
044       if (client != null) {
045         client.interrupt()
046         client = null
047       }
048     }

049     def readFrom(ch: SocketChannel) {
050       val buf = ByteBuffer.allocate(1)
051       assert(ch.read(buf) != -1)
052     }

053     def startClient {
054       require(client == null)
055       client = new TestClient()
056         client.run()
057     }
```

```
058    def instance() = {
059      new MBT (
060        "reset" -> "open" := {
061          ch = ServerSocketChannel.open()
062        },
063        "open" -> "open" := {
064          toggleBlocking(ch)
065        },
066        "open" -> "bound" := {
067          ch.socket().bind(new InetSocketAddress("localhost", 8888))
068        },
069        "bound" -> "bound" := {
070          toggleBlocking(ch)
071        },
072        "open" -> "err" := {
073          connection = ch.accept()
074        } throws ("NotYetBoundException"),
075        "bound" -> "connected" := {
076          require(ch.isBlocking())
077          startClient
078          connection = ch.accept()
079        },
080        "bound" -> "accepting" := {
081          require(!ch.isBlocking())
082          startClient
083        },
084        "accepting" -> "accepting" := {
085          assert(client != null)
086          connection = null
087          maybe (connection = ch.accept())
088        } nextIf ({ () => connection != null} -> "connected"),
```



```
089         "connected" -> "connected" := {
090           readFrom(connection)
091         },
092         "connected" -> "bound" := {
093           connection.close()
094           client = null
095         },
096         "accepting" -> "bound" := {
097           client.interrupt()
098           client = null
099         },
100         Set("open", "bound", "accepting", "closed") -> "closed" := {
101           ch.close()
102         },
103         "closed" -> "err" := {
104           connection = ch.accept()
105         } throws ("ClosedChannelException")
106       )
107     }
108   }
```



Client

# tests	Model coverage		JPF states		other JPF statistics		
	# states	# trans.	new	visited	ins. [1,000s]	mem. [MB]	time
100	5	13	1,070	5	11,667	616	0:15
200	6	15	2,142	3	23,316	1,173	0:31
300	6	19	6,256	24	68,312	2,913	1:51
400	6	20	16,302	61	177,605	4,802	6:03
500	6	20	26,054	121	283,699	6,127	10:20

Server

# tests	Model coverage		JPF states		other JPF statistics		
	# states	# trans.	new	visited	ins. [1,000s]	mem. [MB]	time
100	7	17	676	0	7,185	294	0:16
200	7	17	4,798	223	52,128	551	1:10
400	7	17	6,917	230	74,823	1,131	1:46
800	7	17	11,973	273	128,910	2,607	3:37
1,200	7	17	14,760	282	158,872	4,396	5:33
1,600	7	17	20,194	338	217,039	5,996	9:57

Discussion Java Non-Blocking I/O Case Study

- two defects found
  - wrong exception thrown in `finishConnect` after `close`
    - due to incorrect check in the unit test suite
    - because of cumbersome checking code
  - revealed possibility to read spurious data after end-of-file (EOF)
    - monitored property actually expressed programmatically
    - refactoring into states would have been better though
- expressiveness and succinctness of the Modbat approach helped to find these bugs or could have helped
- generating succinct and correct models is a challenge

Discussion

- model (usually) is an imprecise abstraction — right!?
  - how to generate the model?
  - how to debug failing traces?
  - how to differentiate SUT/API contract violations from SUT faults?
- Modbat has no coverage driven model exploration yet
  - combination with concolic testing
    - does “symbolic only on the model” make sense?
    - compare with “constrained random verification” for HW
  - is there a simple (non-symbolic) “poor man’s version” for more directed testing?
- shrinking / delta debugging — non-intrusive?