Model Checking

Step by Step

Martin Mach

Outline

- Model checking overview
  - Principles
  - Possibilities
- Model checking tools
  - SPIN
  - Bandera

Phases of model checking

- Creating model
  - In terms of input language for selected tool
- Specifying its properties
  - Usually in temporal logic terms
- Running simulations or verification

Deadlocks and Livelocks

- **Deadlock** can be described as a state, when system hasn‘t finished its run but all subprocess are blocked.
- **Livelock** is state, when some subprocesses are running. But system doesn‘t perform desired progress.

SPIN

- Designed for concurrent systems and communication protocols via synchronous and asynchronous channel communications
- Checking for desired properties, deadlocks, livelocks, improper terminations and receptions and unexecutable code
- Simulation and verification support
- Support for large space states

Promela - data types

- 4 base data types: bool, byte, short, int

```
byte input;
```

- Arrays of base types

```
byte input[N];
```

- Enumerations

```
msgtype = {ack, err};
msgtype type;
```

- No pointers, no compound types

Promela - message channels

```
chan in = [N] of {short,int}
```

- Channel operations

  Sending: **`in!expr1,expr2`**

  Receiving: **`in?var1,var2`**

  Head test: **`in?[ack,var2]`**

  Msg count: **`len(in)`**

Promela - processes

```
byte globstate;
proctype A(byte locstate) {
    globstate = locstate
}
```

- Processes are instantiated by operator *run*

```
run A(5)
```

Promela - processes (cont.)

- Initially, one special process is started

```
init {
     run A(5);
     run A(6)
}
```

Promela - case selection

```
if
:: (a == 1) -> option1
:: (a == 2) -> option2
:: (a != 1 && a != 2) -> option3
fi
```

- If several conditions are fulfilled, checker is free to select one of branches to perform
- If no conditions is fulfilled, process is blocked

Promela - repetiton

```
do
:: (a == 1) -> option1
:: (a == 2) -> option2
:: (a != 1 && a != 2) -> break
od
```

Promela - jump

```
proctype Euclid(int x, y) {
  do
  :: (x > y) -> x = x - y
  :: (x < y) -> y = y - x
  :: (x == y) -> goto done
  od;
  done:
     skip
}
```

Promela - assertions

- Designed for setting system properties, if condition is not held, error is produced by verification

```
assert(state = = 1)
```

Promela - miscellaneous

- Blocking statements (until condition is true)

```
(a == 1)
```

- Atomic executions (useful for reducing amount of states)

```
atomic {
    statement1;
    statement2;
}
```

Dining philosophers

```
mtype = {thinking, waiting, eating};
bit fork[3];
proctype philosopher(byte left, right) {
  mtype state = thinking;
  do
  :: ((state==thinking) && (fork[left]==0)) ->
      atomic{fork[left]=1; state=waiting;}
  :: ((state==waiting) && (fork[right]==0)) ->
       atomic{fork[right]=1; state= eating;}
  :: (state==eating) ->
      if
      :: skip;
      :: atomic{state=thinking; fork[right]=0;
            fork[left]=0;}
      fi
  od; }
```

Dining philosophers (cont.)

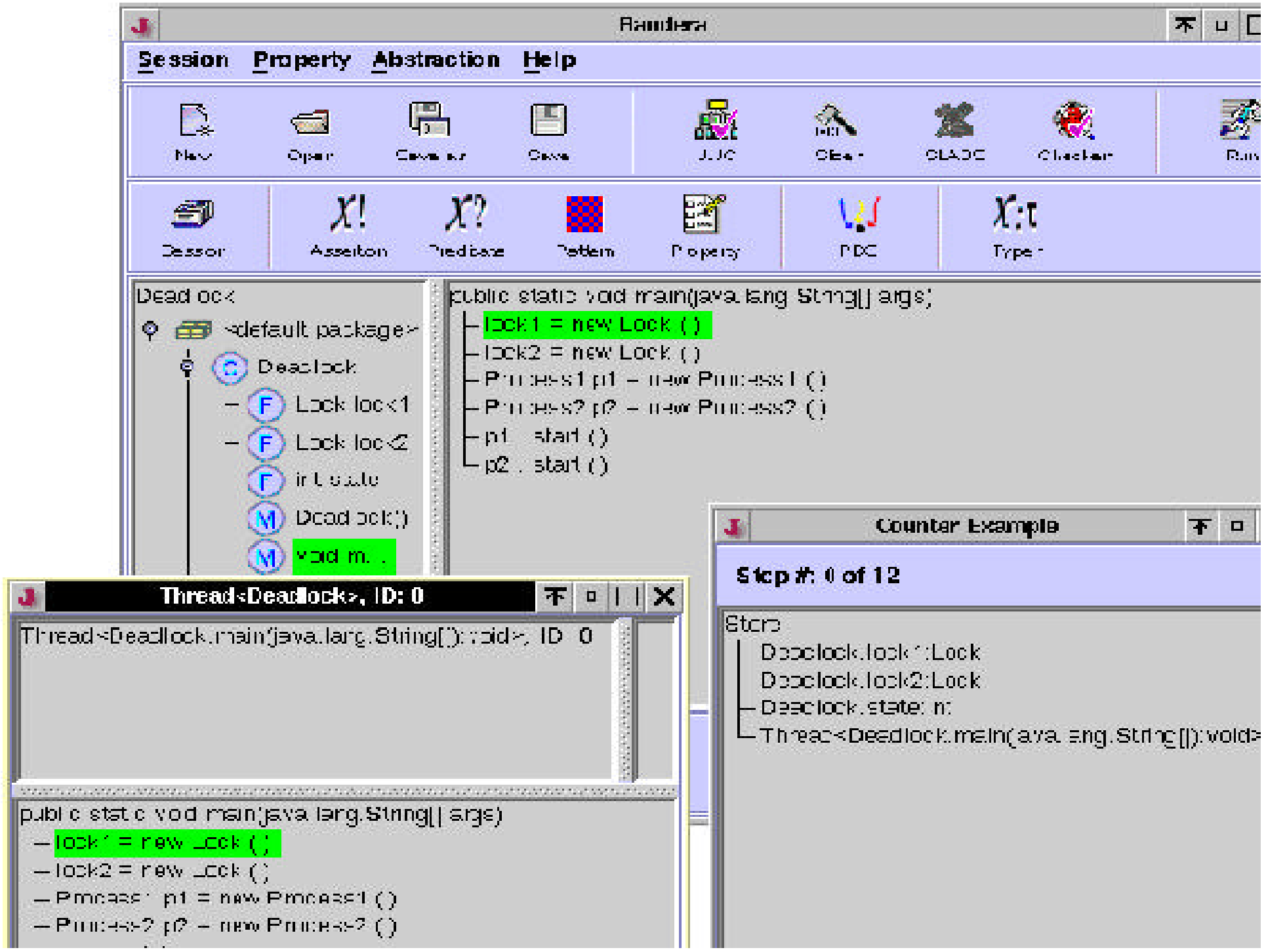
```
init {
  fork[0]=0;
  fork[1]=0;
  fork[2]=0;
  run philosopher(0, 1);
  run philosopher(1, 2);
  run philosopher(2, 0);
}
```

Bandera

- Input language: Java
- Supported model checkers:
  - Spin
  - SVM
  - JPF
- Mapping Java to input language of other tools
- Reverse mapping of error trace back to Java

Bandera - example

```
public class Deadlock {
  static Lock lock1; static Lock lock2;
  static int  state;
  public static void main(String[] args) {
      lock1 = new Lock(); lock2 = new Lock();
      Process1 p1 = new Process1();
      Process2 p2 = new Process2();
      p1.start(); p2.start();
  }}
class Process1 extends Thread {
  public void run() {
        Deadlock.state++;
        synchronized (Deadlock.lock1) {
            synchronized (Deadlock.lock2) {
                  Deadlock.state++;
  }}}}
```


Session
Property
Abstraction
Help
New
Open
Save
Assertion
Predicate
Pattern
Property
PDC
Type
Deadlock
<default package>
Deadlock
Lock lock1
Lock lock2
Deadlock()
void m...
public static void main(java.lang.String[] args)
lock1 = new Lock ()
lock2 = new Lock ()
p1 . start ()
p2 . start ()
Thread<Deadlock>, ID: 0
Thread<Deadlock.main(java.lang.String[]):void>, ID: 0
public static void main(java.lang.String[] args)
lock1 = new Lock ()
lock2 = new Lock ()
Counter Example
Step #: 0 of 12
Deadlock.lock1:Lock
Deadlock.lock2:Lock
Deadlock.state: n:
Thread<Deadlock.main(java.lang.String[]):void>