

At a glance:

LiQuor / ProbMela

in context of
Lorentz Workshop 11/07
Leiden NL

Christel Baier
Frank Ciesinski
Marcus Größer
Joachim Klein

Overview

- **ProbMela**

- Guarded command language
- Strongly inspired by PROMELA (SPIN)
- Probabilistic extensions

- **LiQuor**

- Explicit model checker for quant. **LTL** and **MDPs**

$$\mathcal{M} \stackrel{?}{\models} [\varphi]_{\leq p} \equiv \mathcal{M} \stackrel{?}{\models} [\neg\varphi]_{\geq (1-p)}$$

MDP

LTL

ProbMela

- Imperative probabilistic guarded command language
- Similarities to *PROMELA*:
 - atomic, numeric datatypes, `do . . . od`, `if . . . fi` (nondeterministic choices), channels (synchronous and asynchronous), assertions, partial order reduction
- Differences /Extensions to *PROMELA*:
 - Minor syntactical differences
 - Probabilistic choice (`pi f . . . fi p`),
 - Lossy channels
 - MDP semantics
 - Complex data structures (members of classes)
 - Recursive functions (functions of classes)
 - Numerical solution is performed

- **Lossy channels**

- `lossy 0.3` channel of `{int}` size 8
- `c!x` loses message (i.e. = skip) with probability 0.3

- **Probabilistic choice**

- `pif :0.3:-> value = 3`
 `:0.7:->goto exit`

`fip`

ProbMela process handling:

```
constant int N = 2;
```

```
bool p_bootstrapped=false;
```

```
proctype p(int par){  
    p_bootstrapped; wait  
    skip  
}
```

```
active proctype starter()  
{
```

```
    int i=0;
```

```
    p[N] procs;
```

```
    do::i<N ~>
```

```
        procs[i] = p.create(i);
```

create processes

```
        procs[i].start(); i++
```

start processes

```
    ::i==N~>
```

```
        atomic{
```

```
            p_bootstrapped = true;
```

```
            me.destroy()
```

destroy myself

```
        }
```

```
    od;
```

```
}
```

ProbMela defines:

```
define tryEat(l,r)={
  if  :: forks[l]==false~>forks[l]=true;
    if
      :: forks[r]==false~>forks[r]=true;
      eat++;eat--;
      forks[r]=false; forks[l]=false
      :: forks[r]==true~>forks[l]=false
    fi
  :: forks[l]==true~>skip
fi
}

...
bool[N] forks;
...
tryEat(_mypid,_mypid+1%N)
...
```

ProbMela functions:

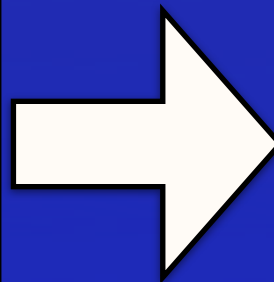
```
function fak(int n)
{
  if :: n==1 return n
    :: n> 1 return (n*fak(n-1))
  fi
}
```

Bytecode principle

```
// Probmela
proctype p() {
  ...

  i=42;

  ...
}
```



```
// PASM

_14_0_guard:
  guard[src = "i = 42 {15,0}"] {
    permit;
    loadca1 "_14_0";
    loadc16 "1";
    ndetenable;
  }

_14_0:
  code[src = "i = 42 {15,0}"] {
    loadc "42";
    storev "i";
    loadca1 "null";
  }

null:
  guard[src = "end_of_process"] {
    permit;
  }
```

Beispiel:
pushretaddr

- Semantical clarity “at least” at PASM level
- Very flexible to implement

Visualization of state space

processcreation

File Help

Model Model Checking Method Statistics and Results Other Options Developer's options

Program Graph Construction DFS on MDP Quantitative Reachability in MDP Quantitative Analysis

A depth first search on the whole state space will be performed. (Properties are processed on the fly)

Maximum search depth: 1 Millions ☐ Apply partial order reduction

Size of hashtable: 1 Millions ☐ LTL property: property.de

Visualization:

Graphviz JFF export State vector dump

☒ create graphviz file (enable only for "small" systems!)

Graphviz Output File: transition_system.graphviz ... Edit

LTL formula is "F a" with the following propositions:
a -> global.i==1

speed variables:

profiling counters:

text fields:

model checking was completed
*** execution complete

Open shell Explorer Full model checking Go

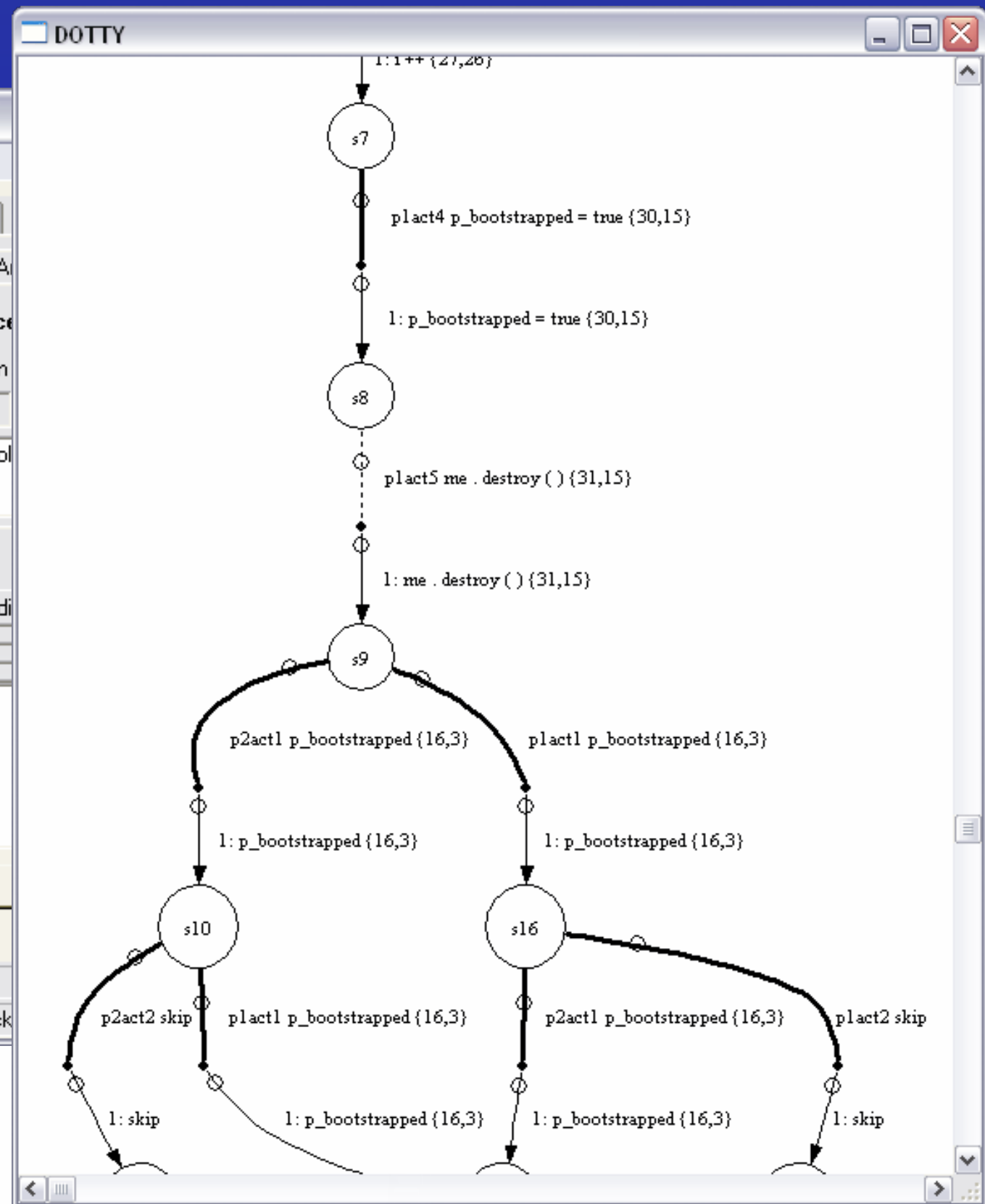
2029230 bytes

DFS only unchanged C:\Dokumente und Einstellungen\ciesinsk\Desktop\LiQuorCheck

```

atomic{
    p_bootstrapped = true;
    me.destroy()
}
od;
}

```



Quantitative analysis:

Vodafone_UMTS

File Help

Model Model Checking Method Statistics and Results

Statistical values of interest

Integer variables Floating point values Text fields

☐ Toggle all

☐ number of cache hits for state-hashing

☐ number of states in PROBQA

☐ number of states in PROB1E

☒ number of states in S_?

Output of results:

Report CSV output

☒ generate CSV file analysis.csv

☐ append to existing CSV file

speed variables:

profiling counters:

text fields:

model checking was completed

*** execution complete

Result of last run

LiQuor ended with the following message:

Results:

integer counters:

number of states: 35202

number of transitions: 36413

number of states in S_?: 19298

the bound (lp result): 0.0255815

time variables:

overall time (s) for model checking: 47

time (s) for solving linear program: 46

speed variables:

profiling counters:

text fields:

(see panel "Statistics and Results" for more result values)

OK

Open shell Explorer Full model checking Go Last results Quit

Cocktail V 1.8.15

1360446 bytes

quant.reach. unchanged C:\Dokumente und Einstellungen\ciesinsk\Desktop\LiQuorCheckout\LiQuor\LiQuor\moc

G (not a or F b)[a (USIM.syncNeeded)][b (USIM.nok le ...

File Help

LTL formula: Syntax help

Automaton type: deterministic Rabin

LiQuor/PASM

Probmela definition:

Assign variable evaluations to atomic propositions

Process variables

Process:

Variables:

Use Variable

Atomic propositions

Variable evaluation:

Define evaluation as proposition

Undefine /edit proposition

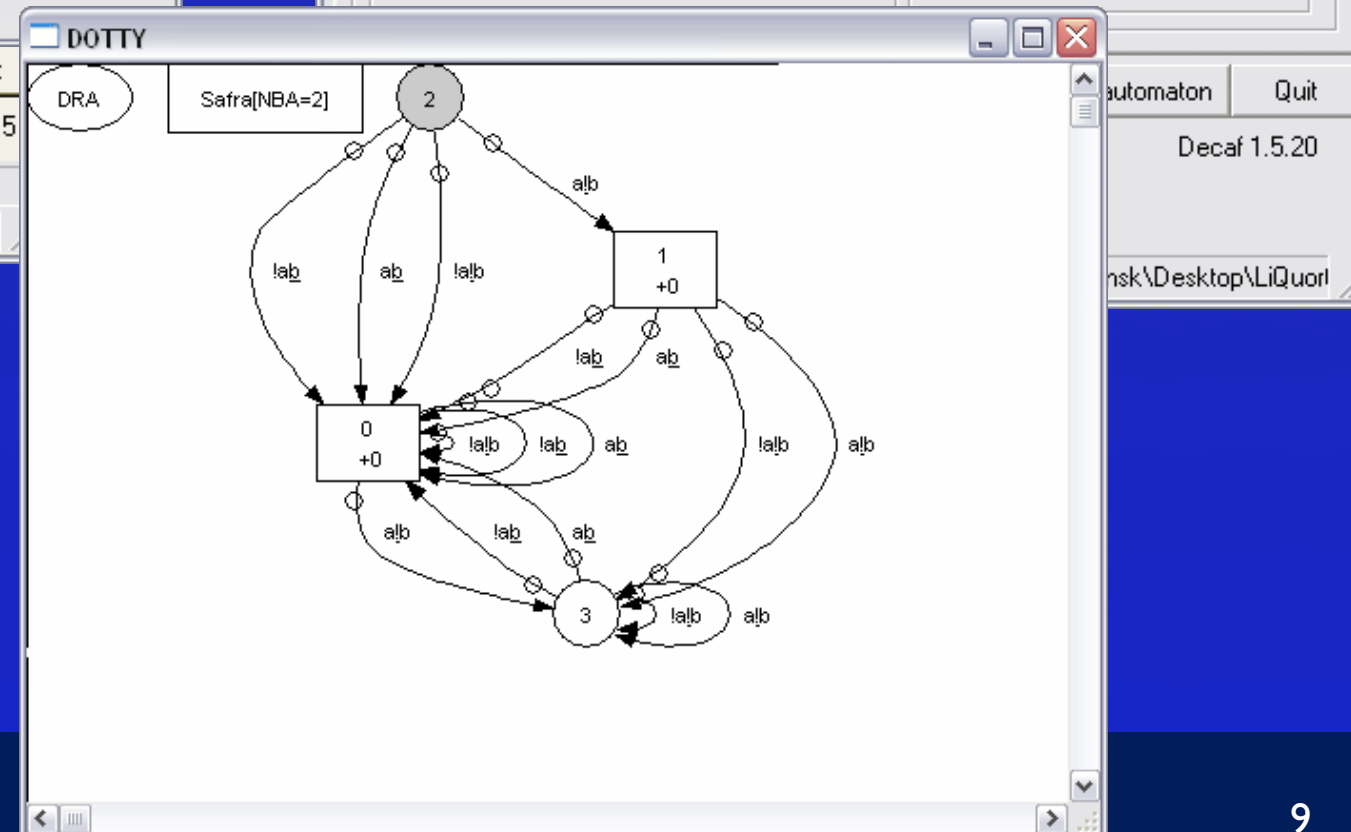
a (USIM.syncNeeded)

b (USIM.nok < 10)

Display after building

☒ automaton

☐ automaton pasm code



- Numerical Solution in LiQuor:
 - Value Iteration
 - Ipsolve (free LP lib)
 - export to file

Simulation /Step by step:

Processes

Process Type	Process Type ID	Process ID
global.pbin	1	1
p.pbin	2	3

Process Type: global.pbin
Process Type ID: 1
Process ID: 1

Process Type: p.pbin
Process Type ID: 2
Process ID: 3

#0: "@instance"==0
#1: "par"==0

Row: n/a Col: n/a

starter.pbin (proc 1, 4 user variables)

Process Type: starter.pbin
Process Type ID: 3
Process ID: 2

#0: "@instance"==0
#1: "i"==0
#2: "procs"==3
#3: "procs[1]"==0

Row: 27 Col: 8

active proctype star

uDraw(Graph) 3.1.1

File Edit View Navigation Abstraction Layout Options ?

p1procs[i] = p.create(i) (26,8)

proc #0 <PTID: 1> <PID: 1> <blocking> <PC : 56> vars: <#0:0> <#1:1> <#2:0>
proc #1 <PTID: 3> <PID: 2> <enabled> <PC : 64> vars: <#0:0> <#1:0> <#2:0> <#3:0>
length in bytes (whole vector): 25

s_2

proc #0 <PTID: 1> <PID: 1> <blocking> <PC : 56> vars: <#0:1> <#1:1> <#2:0>
proc #1 <PTID: 3> <PID: 2> <enabled> <PC : 278> vars: <#0:0> <#1:0> <#2:3> <#3:0>
proc #2 <PTID: 2> <PID: 3> <blocking> <PC : 48> vars: <#0:0> <#1:0>
length in bytes (whole vector): 34

SimulationControl

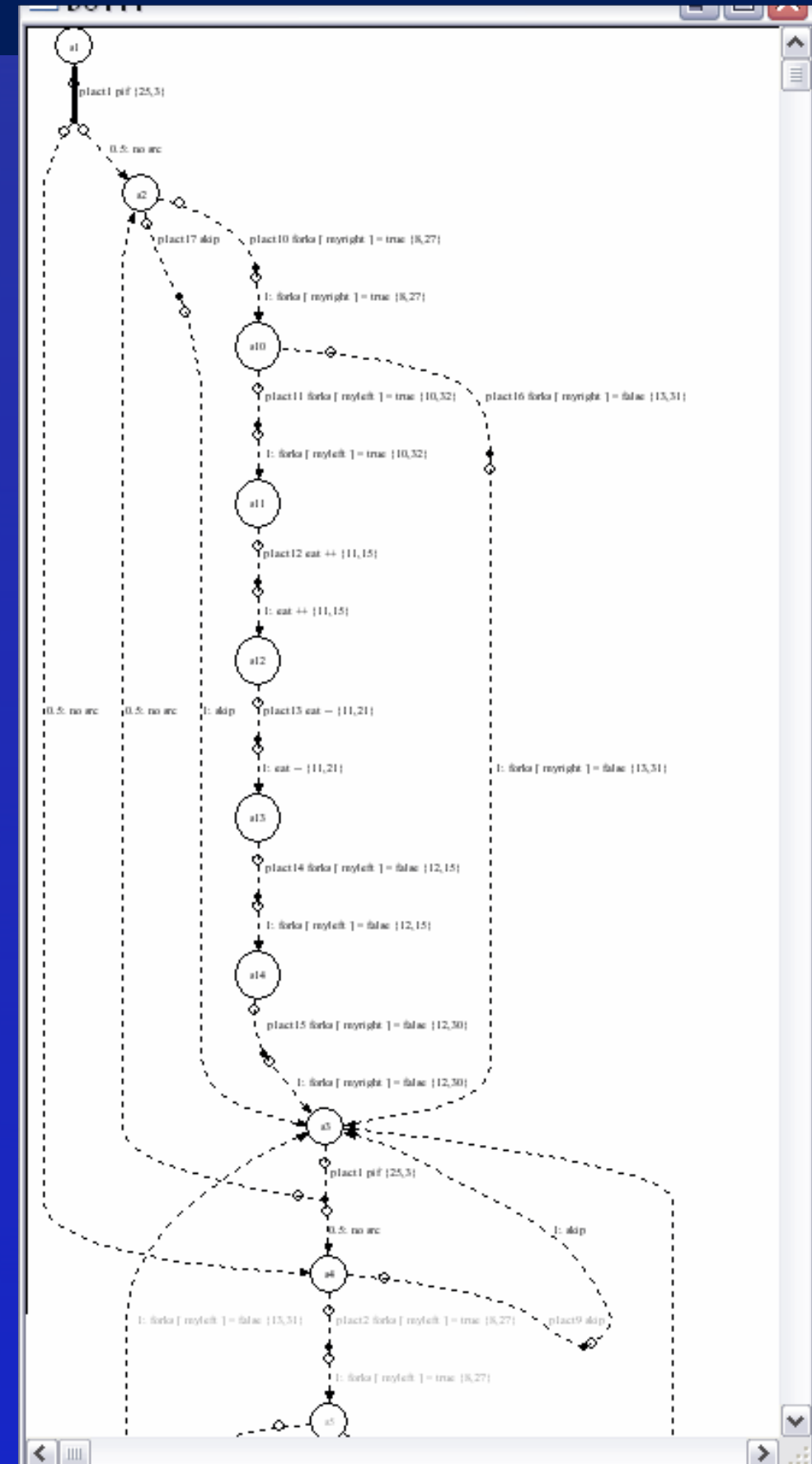
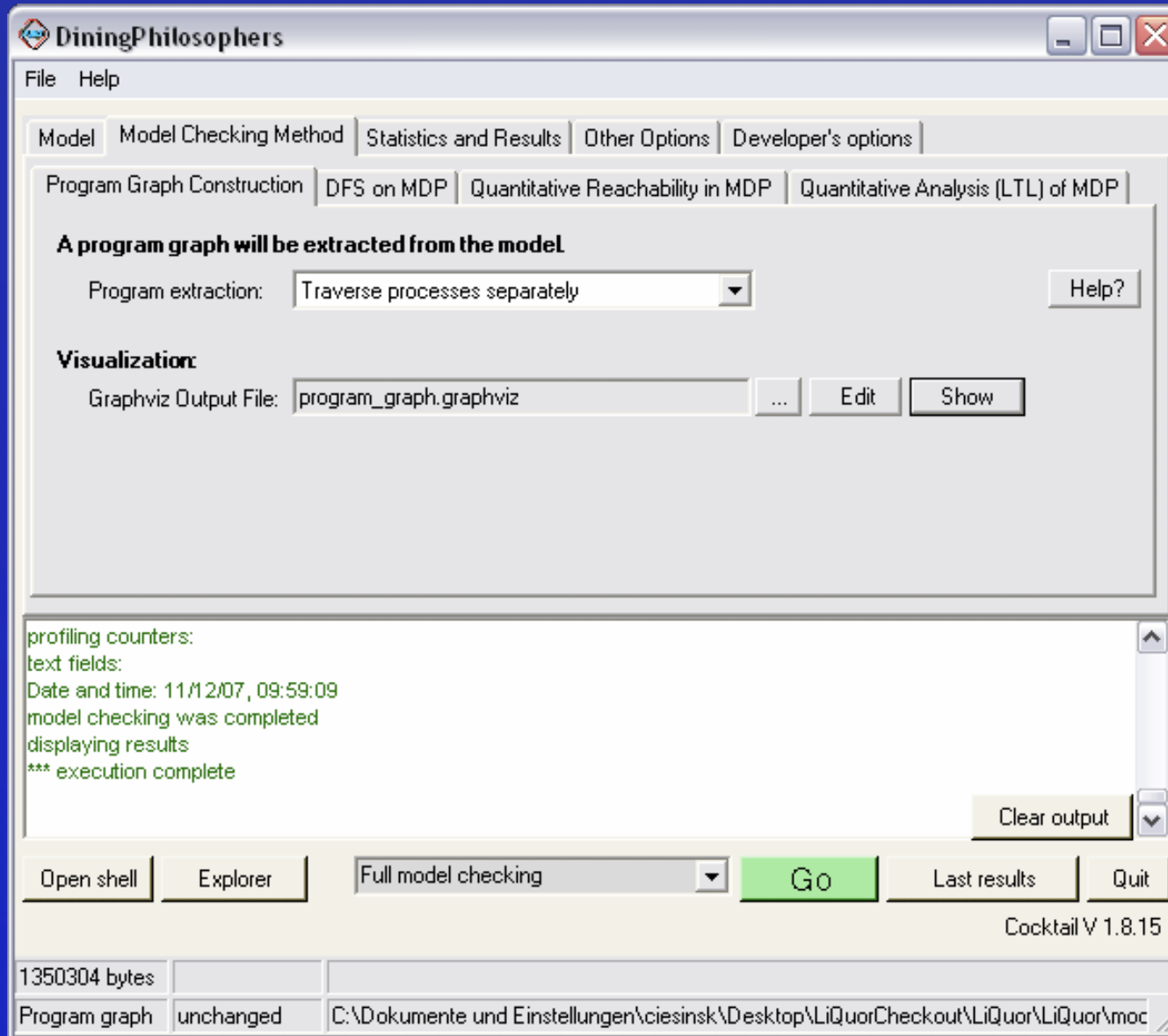
p1-act2@323-p1procs[i].start() {27,8}

Visited states:
Current state: 2
Last action: procs[i] = p.create

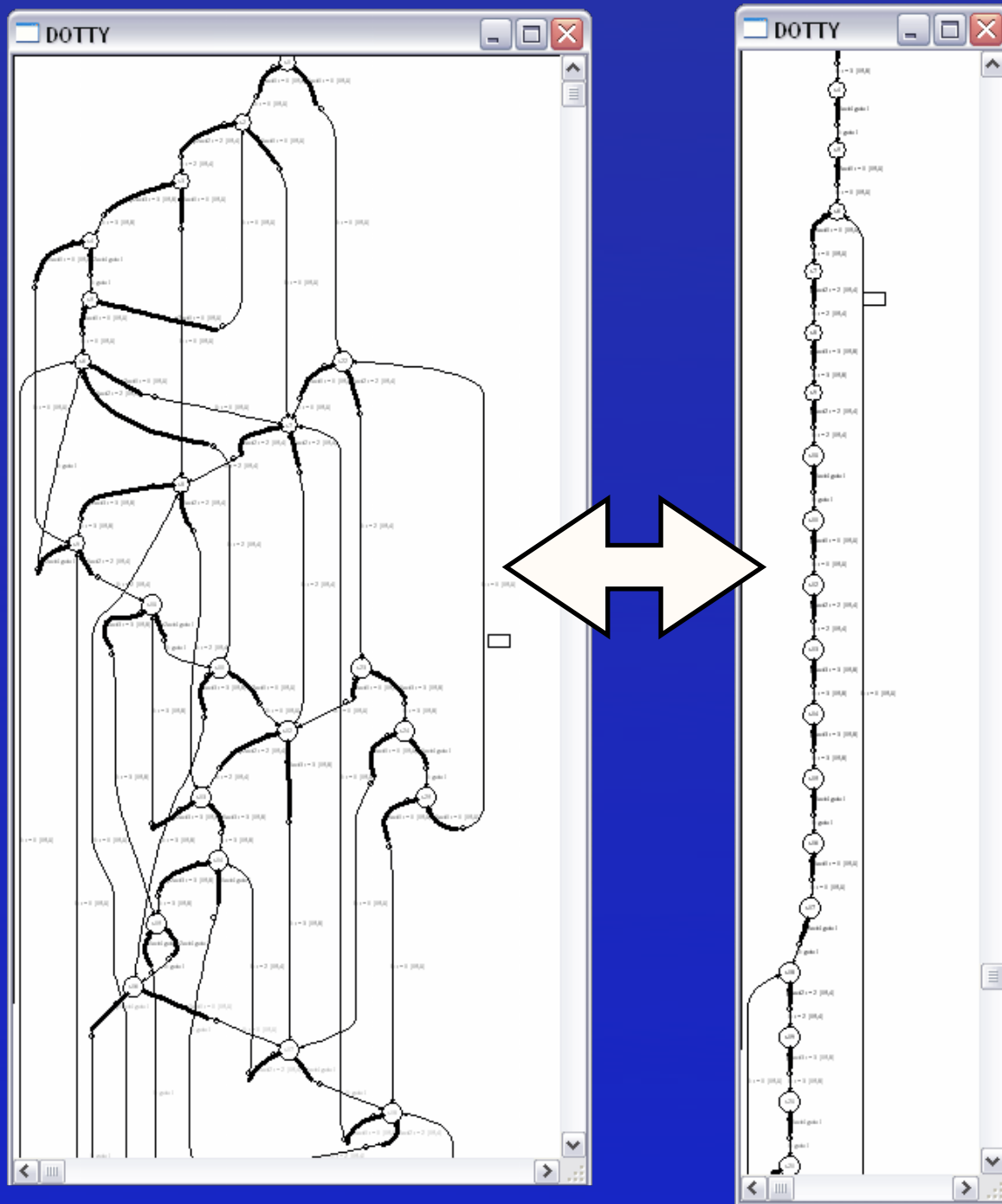
State history (select to set):
1
2
states: 2

Simulation data
Clear State history
Clear variable history
Show variable history
always on top
Close

Program Graph analysis:



Partial Order Reduction:



6 Dining Philosophers	w/o POR	with POR
states	411255	305757
actions	2875869	1133974

5 Leader Election (as)	w/o POR	with POR
states	>4 Mio	967855
actions	>12 Mio	1.5 Mio

Reachability
analysis

- **LiQuor for Research:**
 - **A flexible and dynamic language**
 - **Process handling is very dynamic**
 - **LTL**
 - **explicit (:-) or :-(???)**
- **Extending LiQuor**
 - **Cooperations are very welcome**
- **Availability of LiQuor:**
 - http://www.tcs.inf.tu-dresden.de/~ciesinsk/transfer_open/liquor_download.html
- **Available for Windows only**