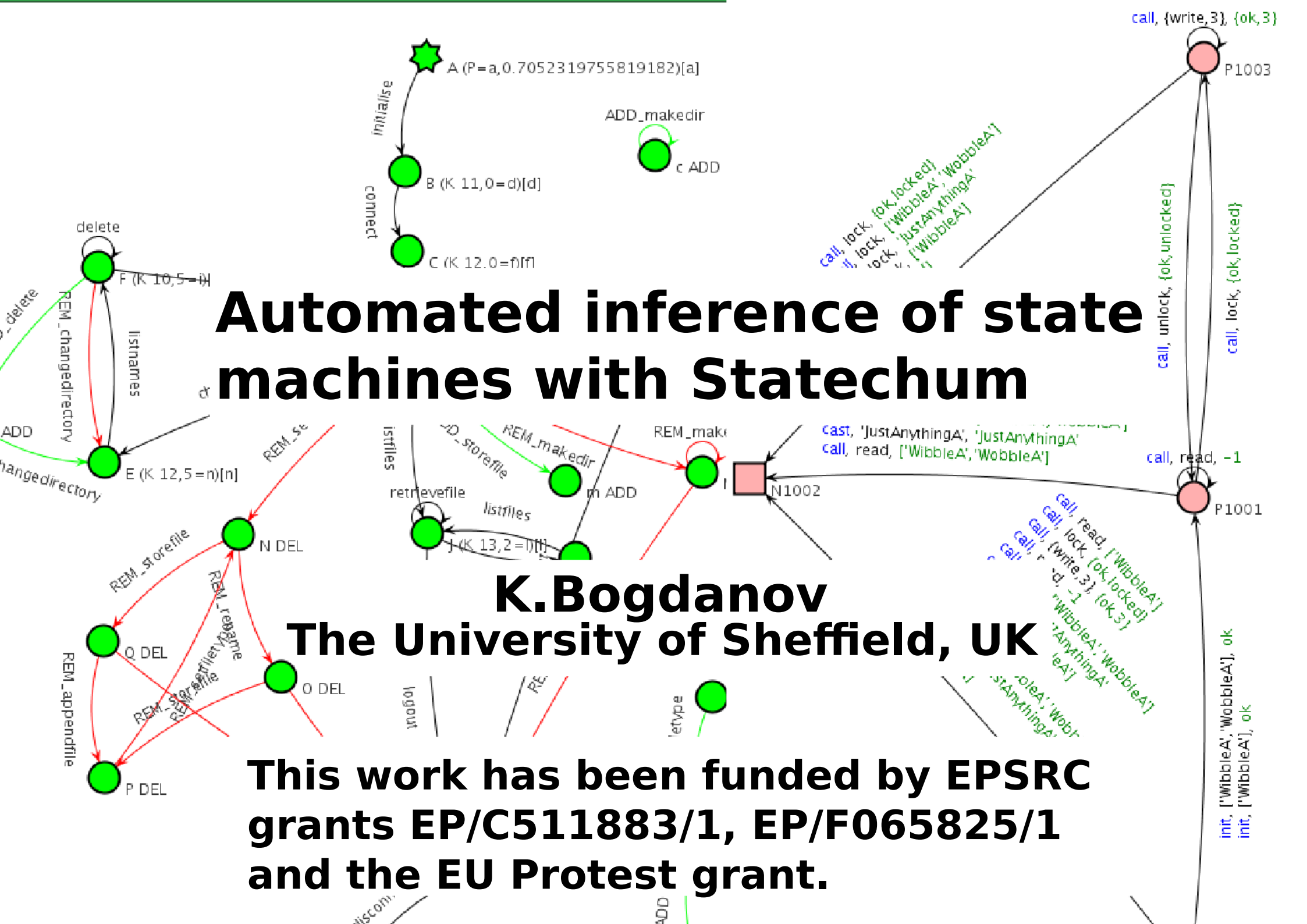


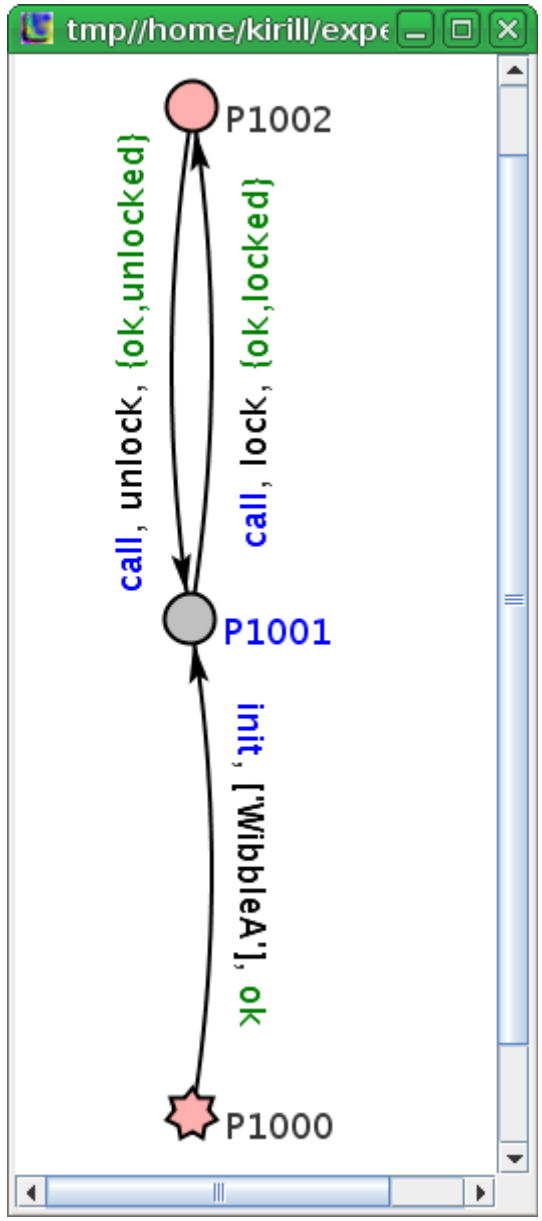
Automated inference of state machines with Statechum

K.Bogdanov

The University of Sheffield, UK

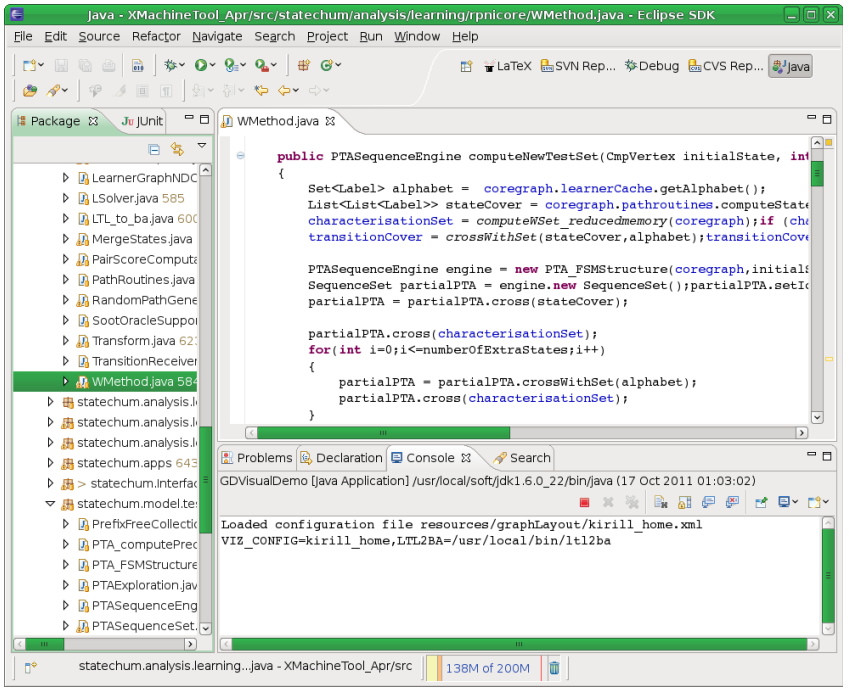
This work has been funded by EPSRC grants EP/C511883/1, EP/F065825/1 and the EU Protest grant.

Statechum is a tool for model inference using dynamic analysis, 1/3.

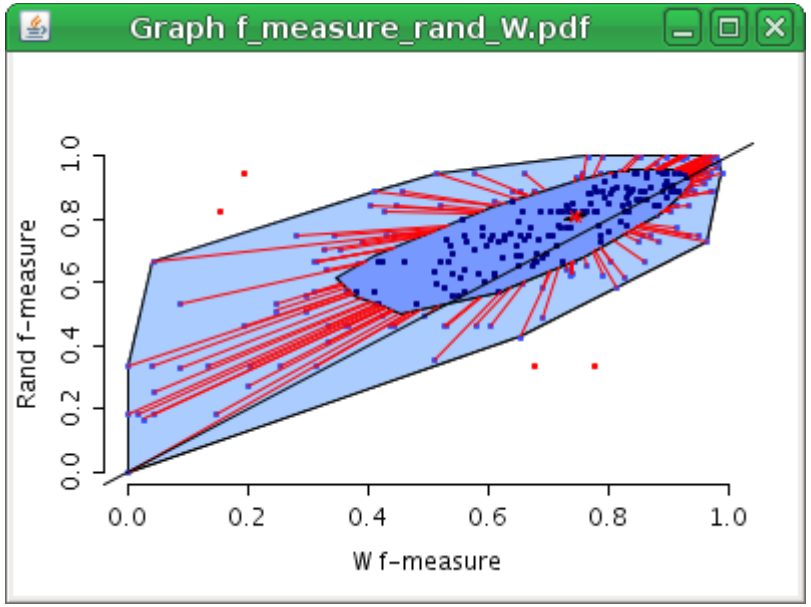


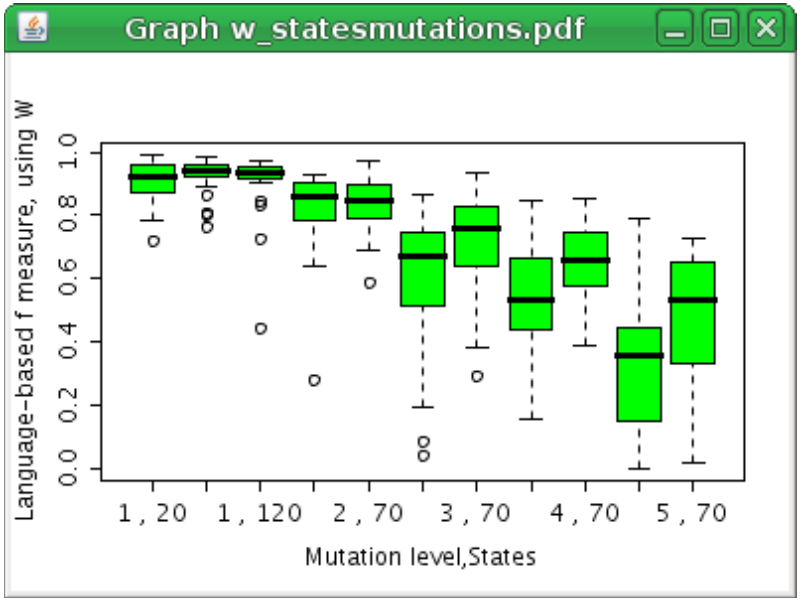
- Given a collection of traces, it can QSM-infer a corresponding FSM,
  - Passively or
  - Actively where decisions are verified by generating and running tests
- Can make use of domain-specific constraints,
  - using Typer or
  - using user-written automata or LTL formulae.

Statechum is a tool for model inference using dynamic analysis, 2/3.



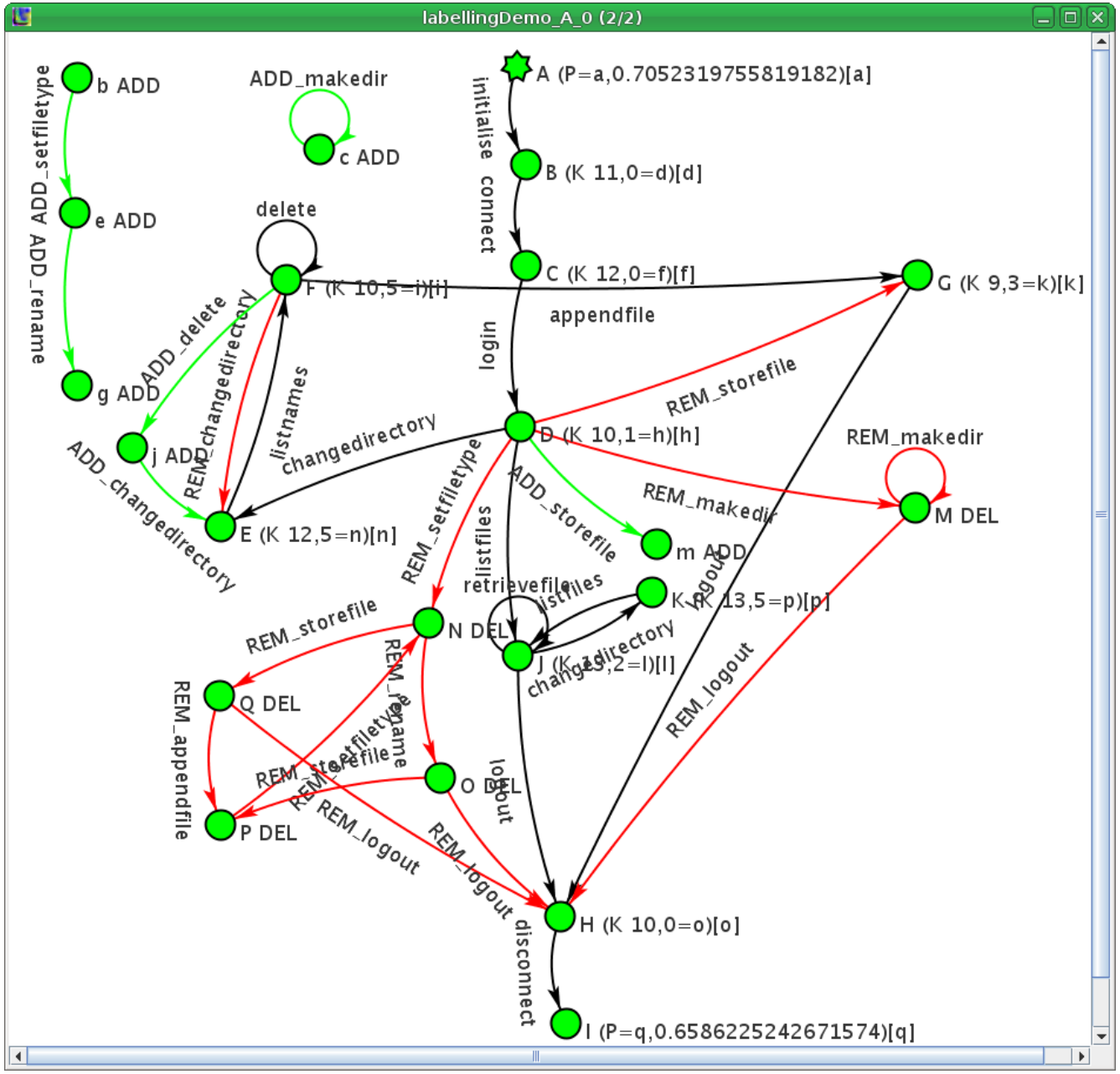
- Supports test generation using W method
- Makes it possible to generate random FSM and random walks from FSM.
- Graph comparison experiments:





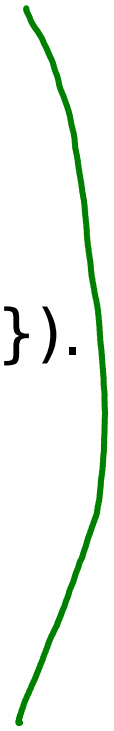
Statechum is a tool for model inference using dynamic analysis, 3/3.

- Markov learner v.s. QSM passive learner applied to CVS protocol.
- Green means new to Markov,
- Red means new to QSM,
- Black edges are matched.



Locker - sample trace


```
Eshell V5.8  (abort with ^G)
1> gen_server:start_link({local, mod_under_test}, locker,[56],[]).
{ok,<0.34.0>}
2> gen_server:call(mod_under_test,read).
-1
3> gen_server:call(mod_under_test,lock).
{ok,locked}
4> gen_server:call(mod_under_test,{write,3}).
{ok,3}
5> gen_server:call(mod_under_test,unlock).
{ok,unlocked}
6> gen_server:call(mod_under_test,read).
3
7> gen_server:call(mod_under_test,{write,3}).
=ERROR REPORT==== 16-Oct-2011::21:38:58 ===
** Generic server mod_under_test terminating
** Last message in was {write,3}
** When Server state == {unlocked,3}
```



Successful calls

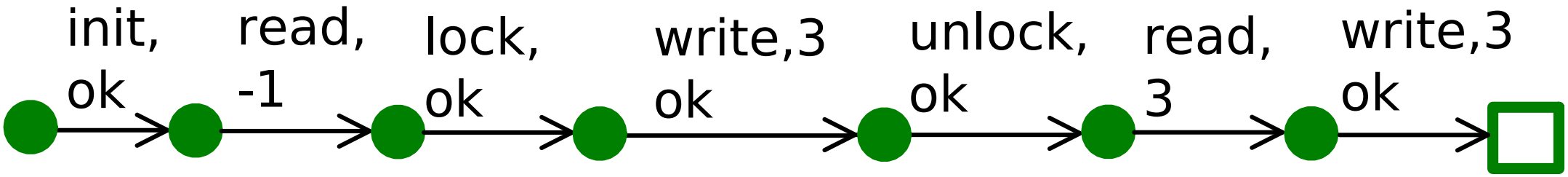
Failure

A trace and the corresponding inputs/outputs

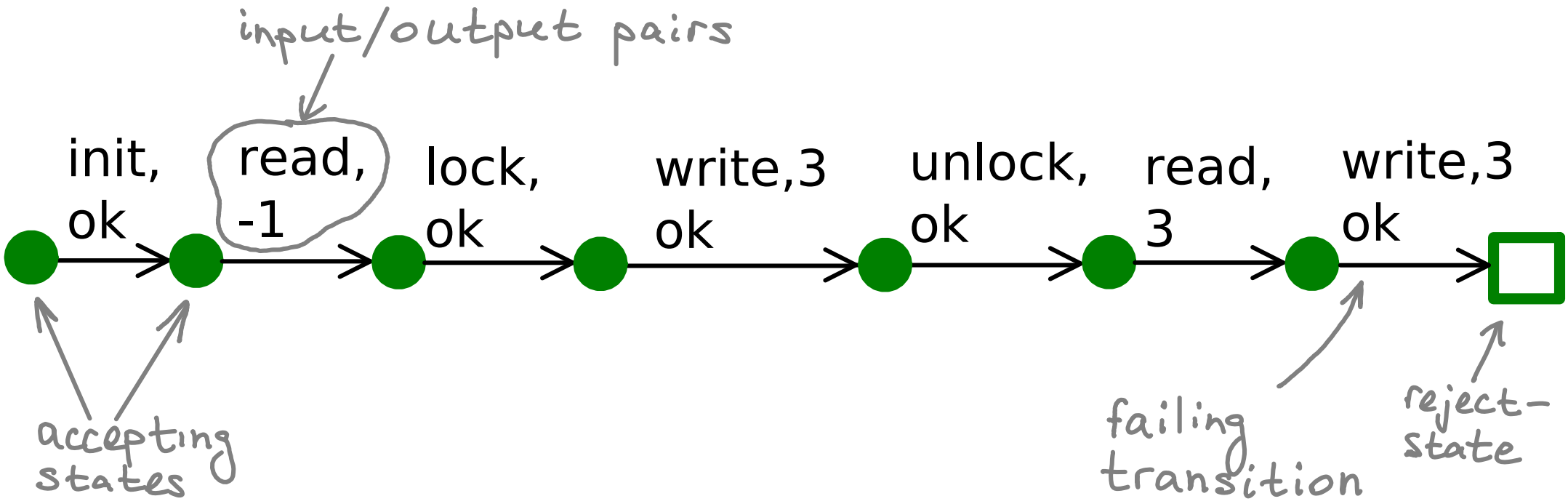
```
Eshell V5.8  (abort with ^G)
1> gen_server:start_link({local, mod_under_test}, locker,[],[]).
{ok,<0.34.0>}
2> gen_server:call(mod_under_test,read).
-1
3> gen_server:call(mod_under_test,lock).
{ok,locked}
4> gen_server:call(mod_under_test,{write,3}).
{ok,3}
5> gen_server:call(mod_under_test,unlock).
{ok,unlocked}
6> gen_server:call(mod_under_test,read).
3
7> gen_server:call(mod_under_test,{write,3}).
=ERROR REPORT==== 16-Oct-2011::21:38:58 ===
** Generic server mod_under_test terminating
** Last message in was {write,3}
** When Server state == {unlocked,3}
```

```
{call, init, [],       ok}
 {call,read,[],
                        -1}
 {call,lock,[],
              {ok,locked}}
{call,write,[3],
                   {ok,3}}
 {call,unlock,[],
          {ok, unlocked}}
 {call,read,[],
                        3}
 {call,write,[3],
                   {ok,3}}
```

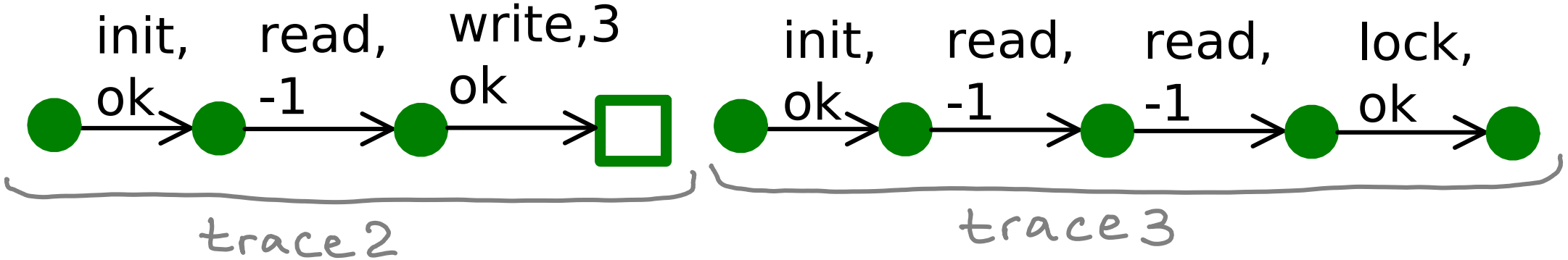
Prefix-tree automaton (PTA) of traces, 1/3.



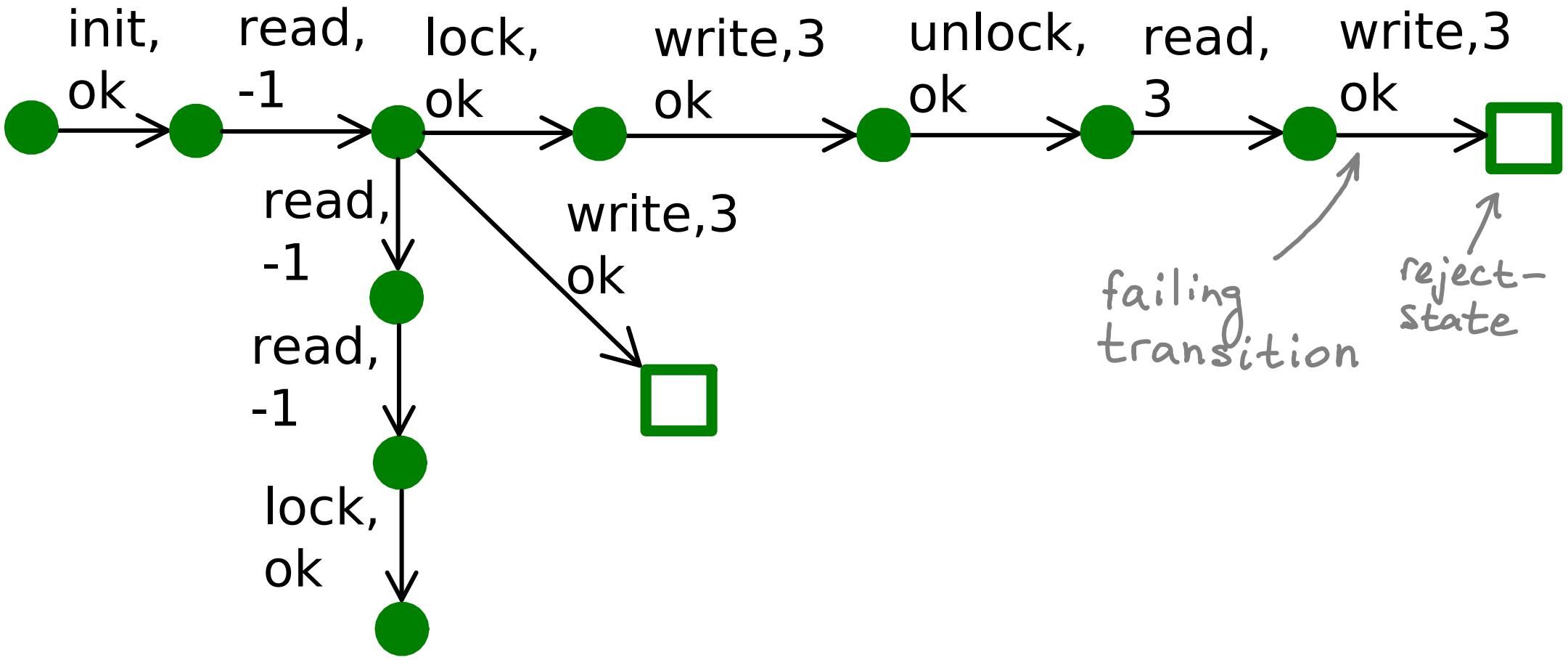
Prefix-tree automaton (PTA) of traces, 2/3.



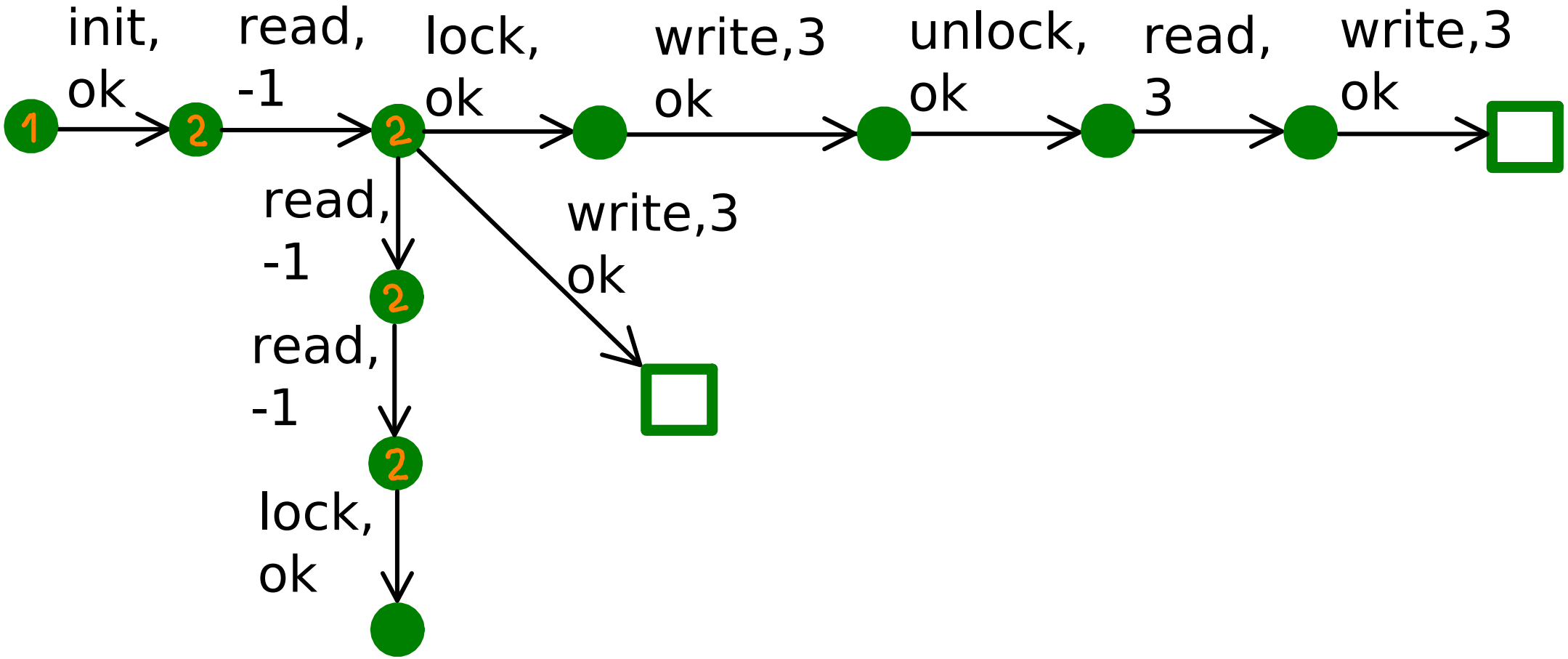
If there is more than a single trace available,



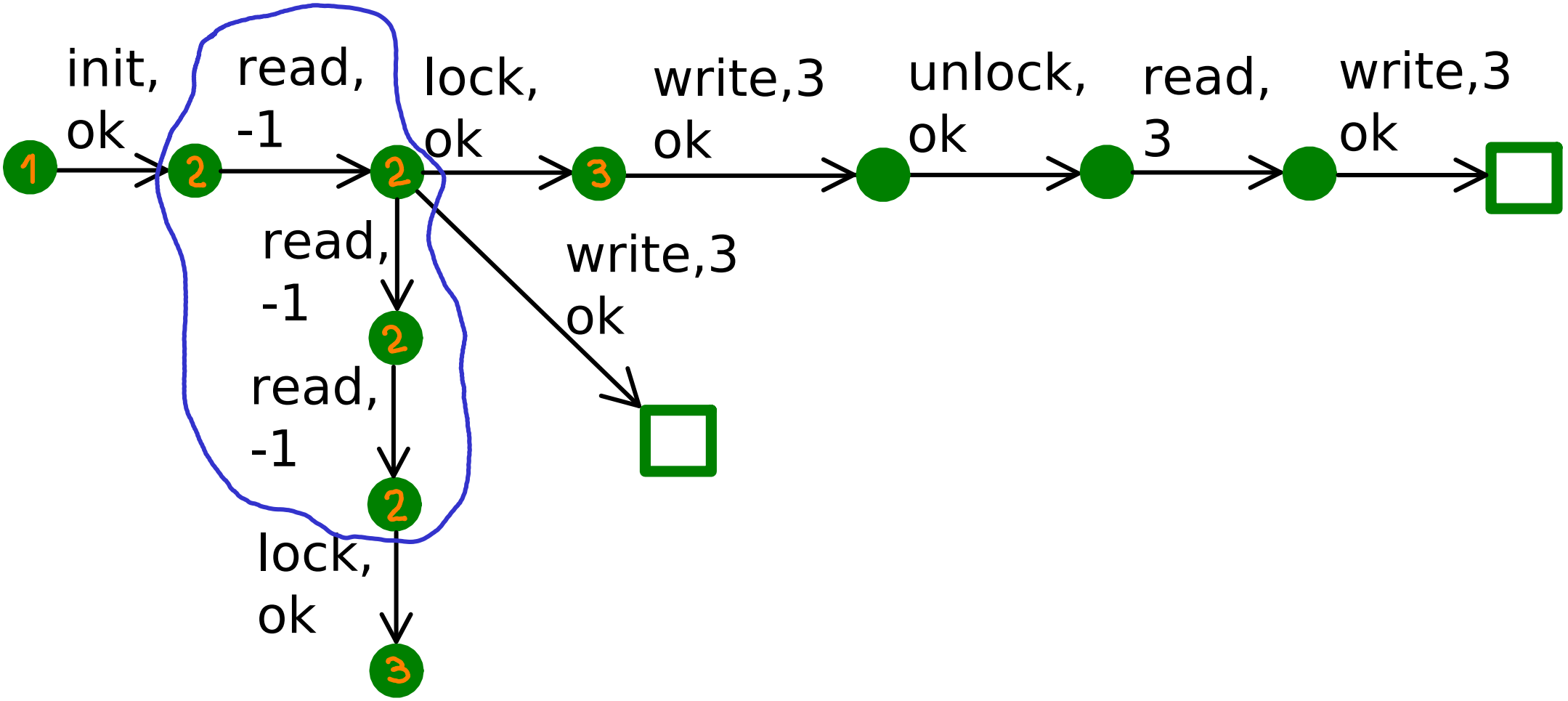
Prefix-tree automaton (PTA) of traces, 3/3.



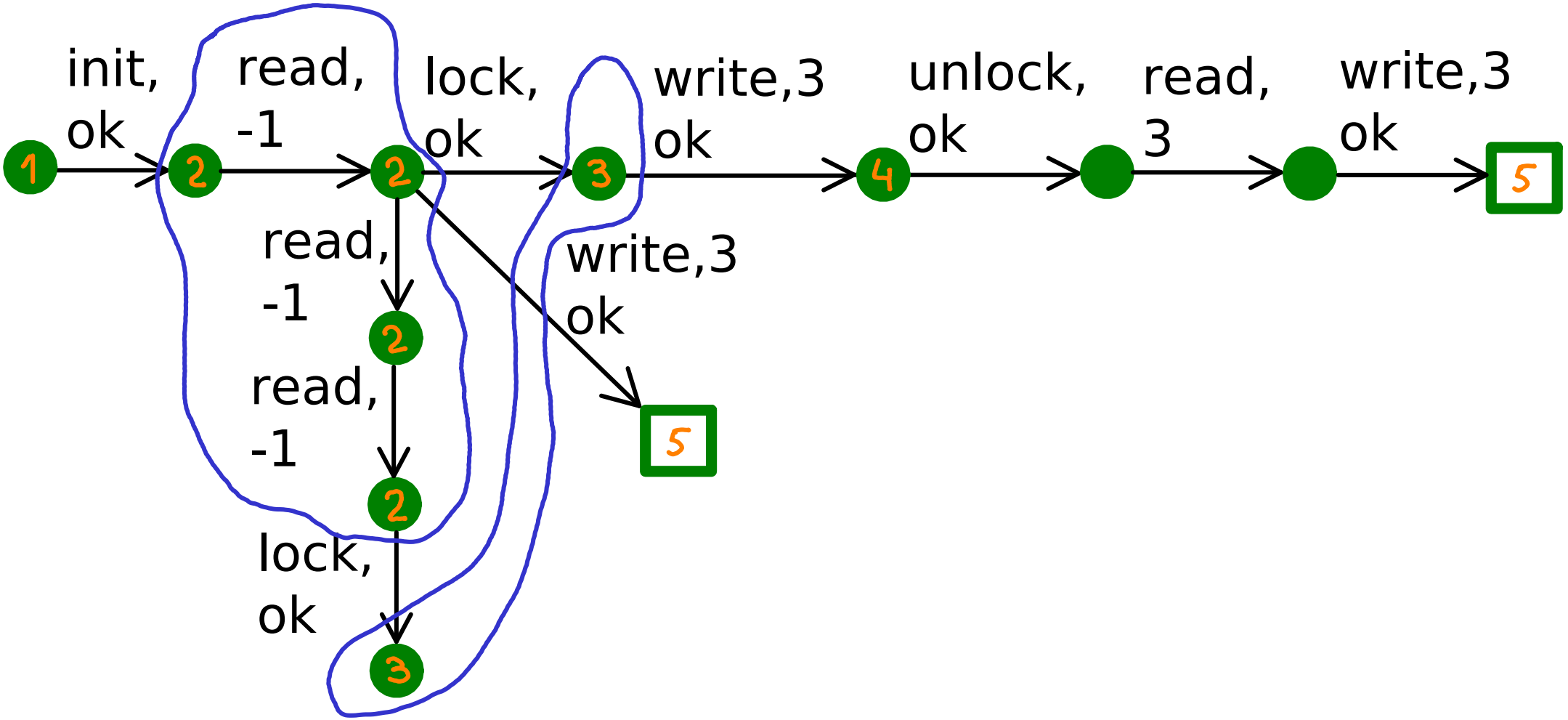
The idea of state-merging, 1/4



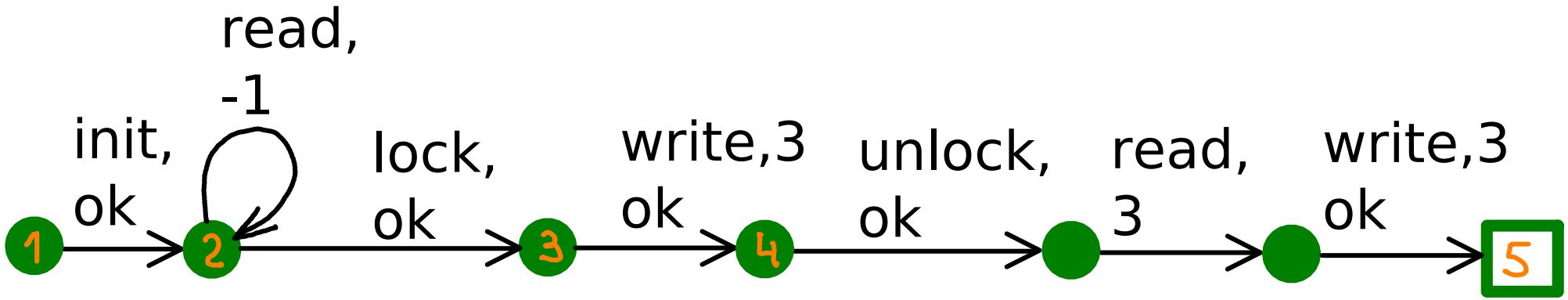
The idea of state-merging, 2/4



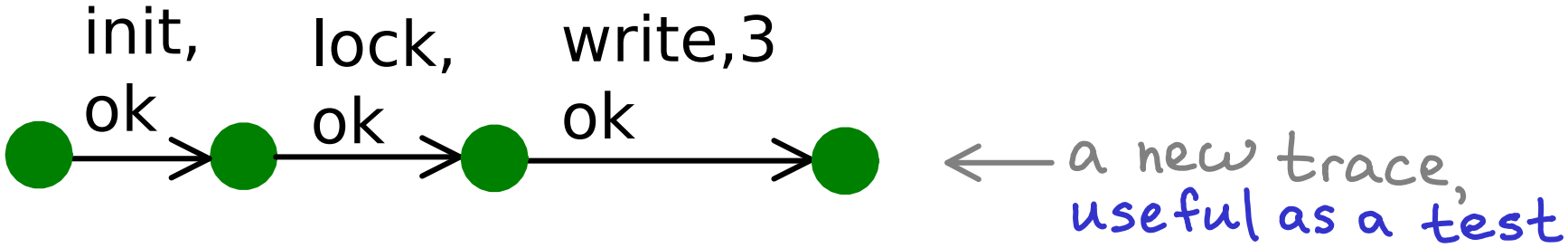
The idea of state-merging, 3/4



The idea of state-merging, 4/4



- Merged automaton accepts more traces than the original



- Decisions to merge states are based on behaviour of those states.

learner configuration

```
config erlangSourceFile /home/kirill/experiment/
                statechum_tutorial_Nov_4/locker.erl
config labelKind LABEL_ERLANG
config erlangModuleName locker
+[ [ {'init',[],'ok'},{'call','read',-1},
       {'call','lock',{ok,locked}},
       {'call',{write,3},{ok,3}},
       {'call','unlock',{ok,unlocked}},{'call','read',3}]
]
```

module file

labels are Erlang

module name

traces

Traces are given in the form of

```
+ [ trace11, trace12, trace13 ... ],
- [ trace21, trace22, trace23 ...] ,
+[  trace31, trace32, trace33 ...]
```

positive

negative

Where each trace is a list of labels

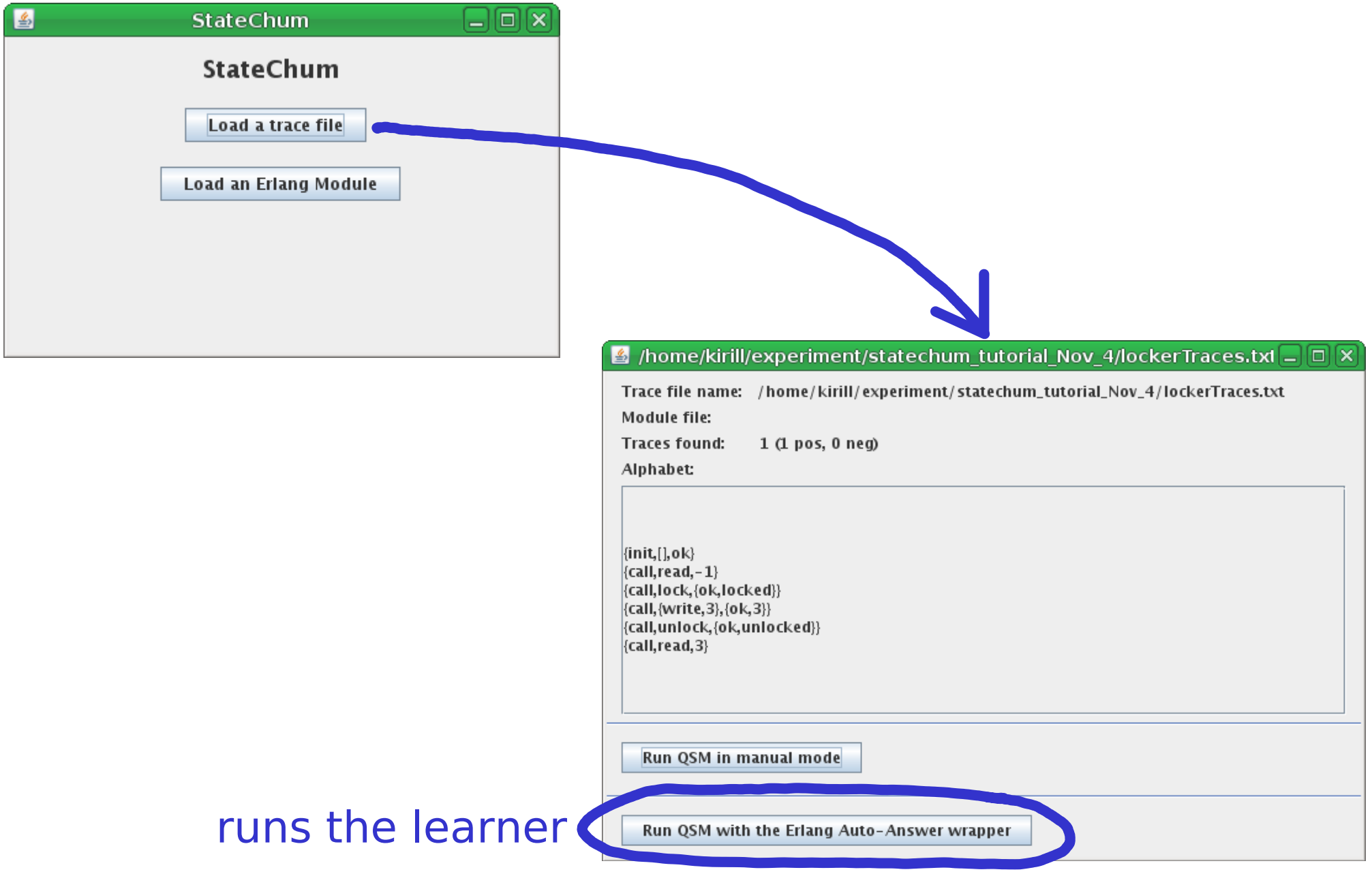
```
[ { 'call','read',-1 }, ... ]
```

handle_call

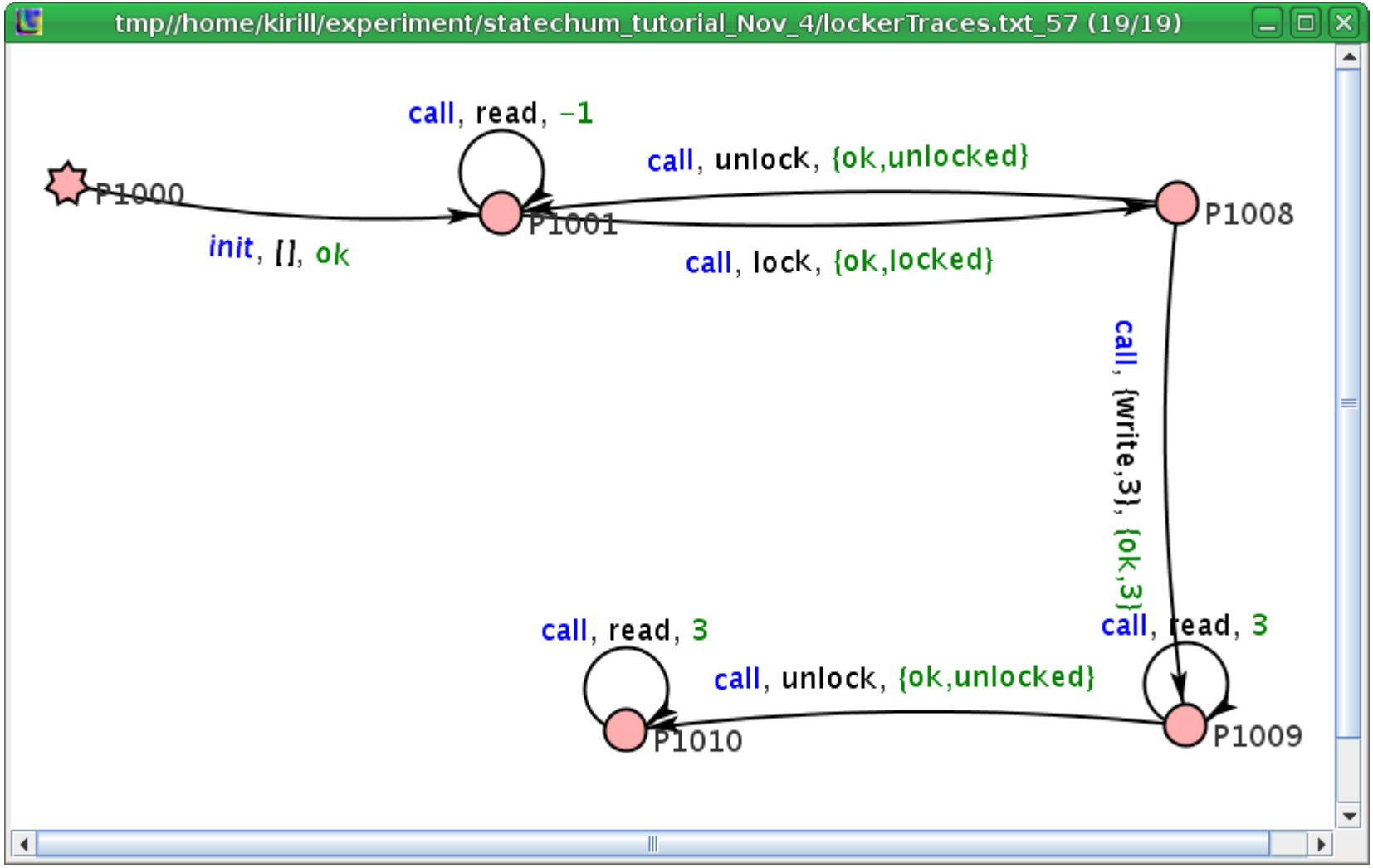
input

output

Finally, the demo



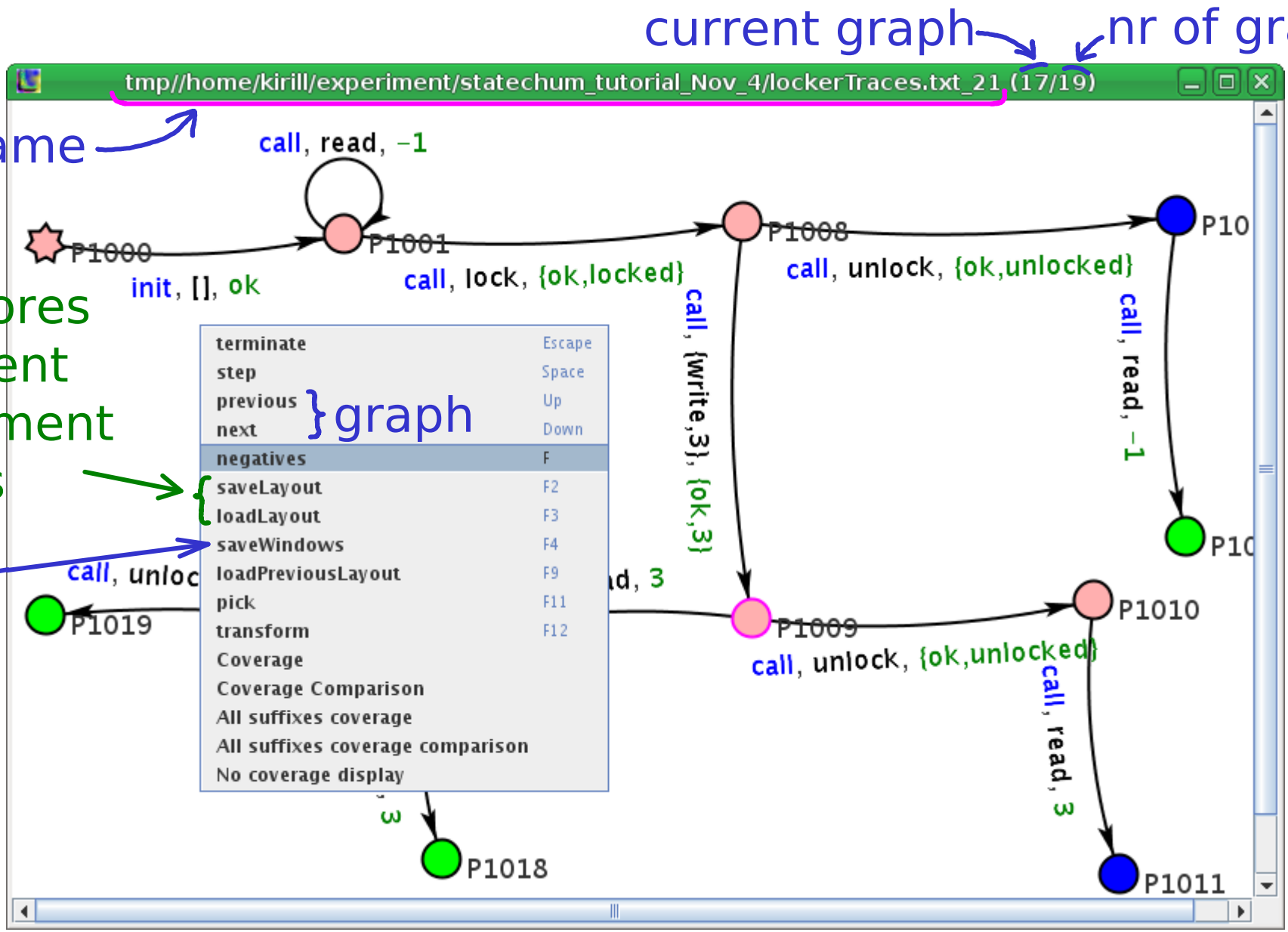
The outcome of learning



Tests and responses from Erlang

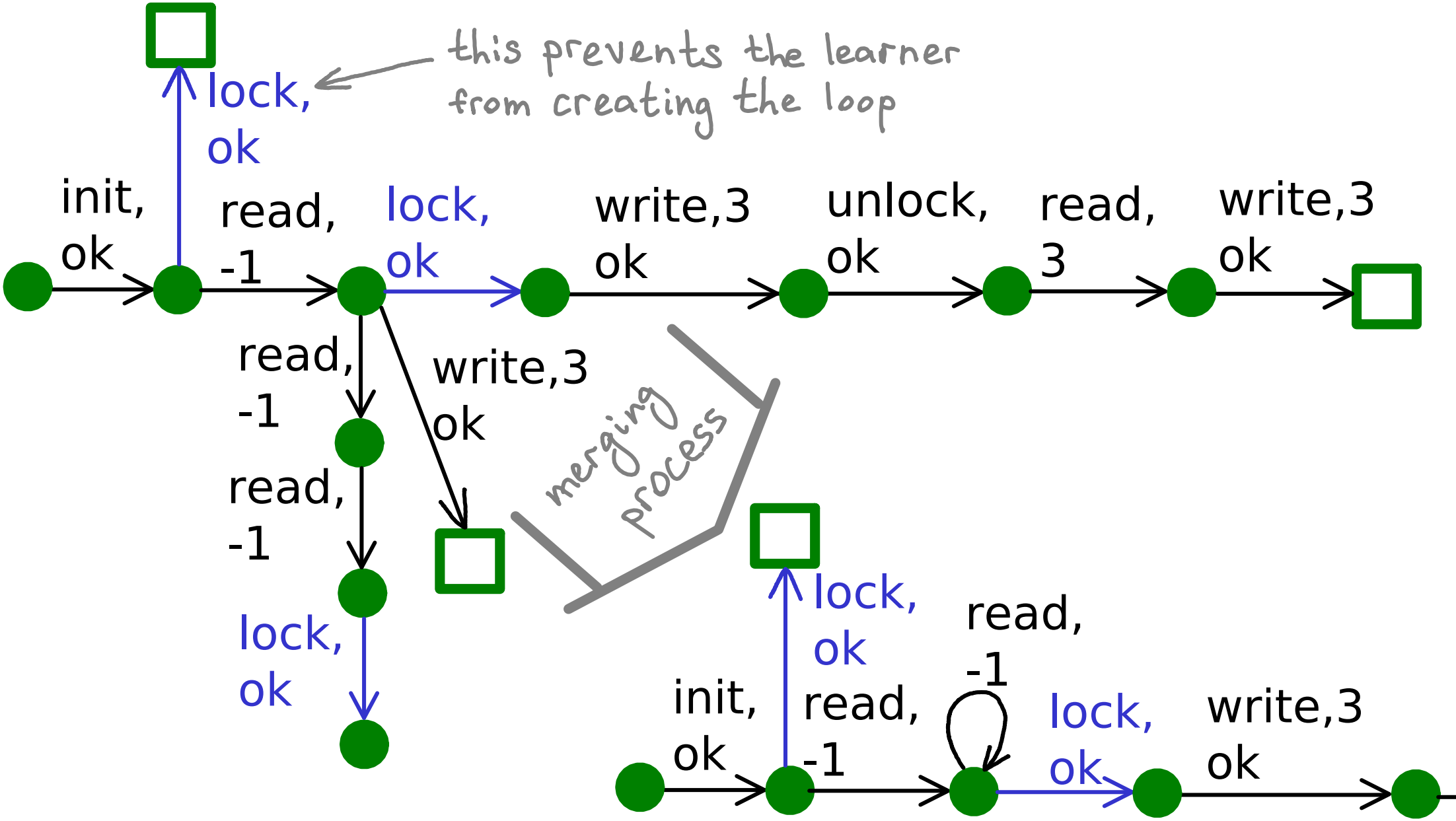
Asked erlang about {init, [] ,ok}{init, [] ,ok}
and the answer is TRACE_FAIL: {init, [] ,ok}{init, [] ,ok}
Asked erlang about {init, [] ,ok}{call, read ,-1}{call, read ,-1}
and the answer is TRACE_OK: {init, [] ,ok}{call, read ,-1}
{call, read ,-1}

Graph window via Jung

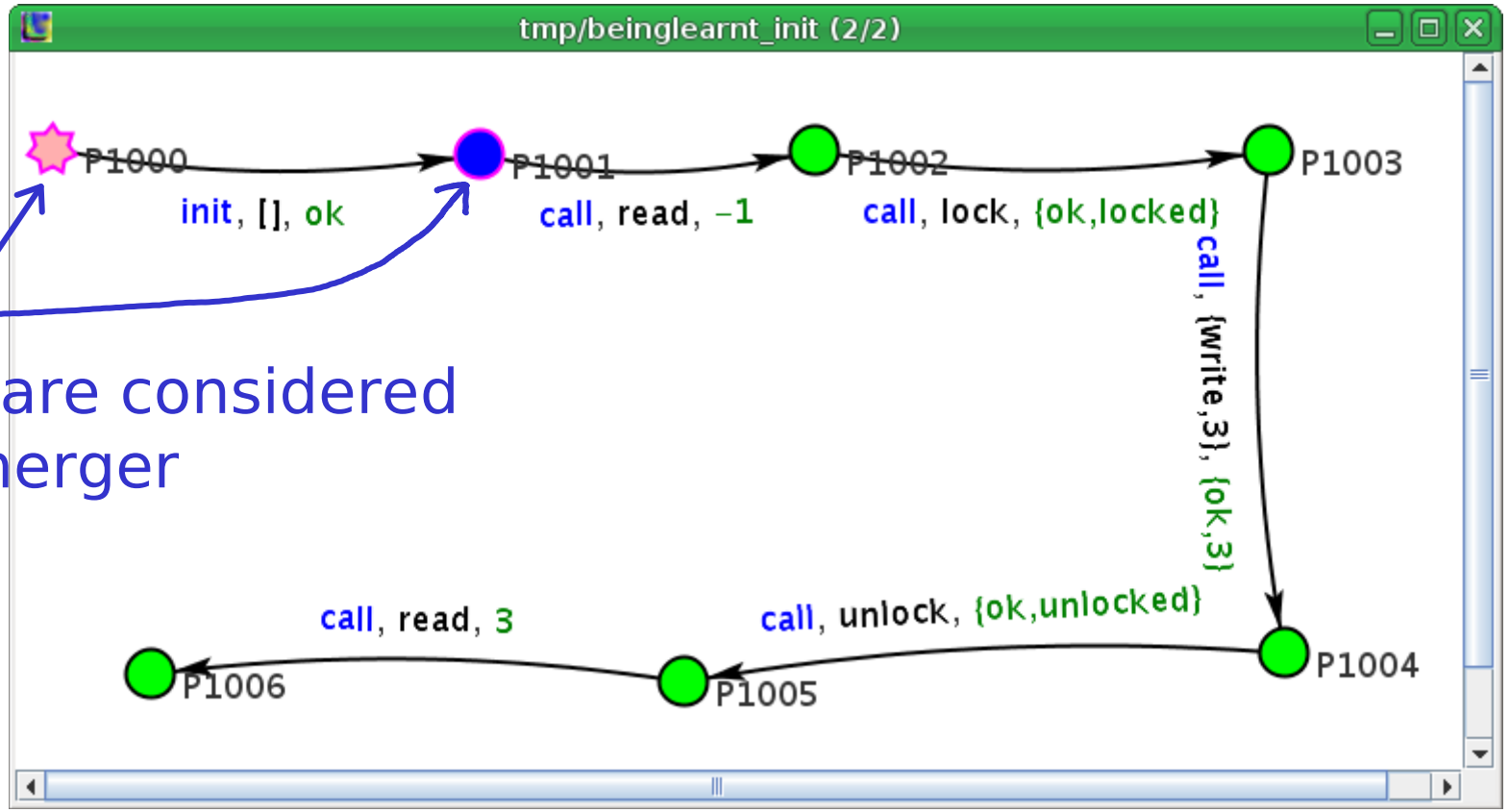


In the above graph, negative state is not shown (default).

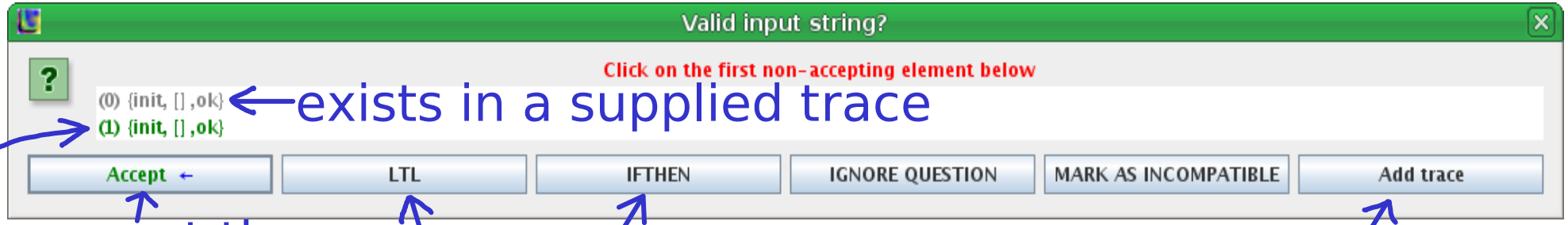
Importance of negative traces



Learner - manual mode



these states are considered for a merger



exists in a supplied trace

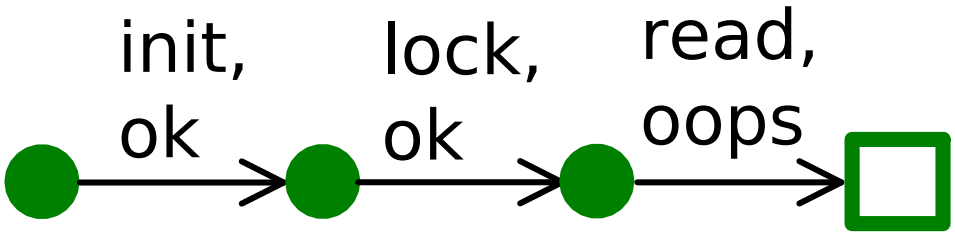
accept the whole trace

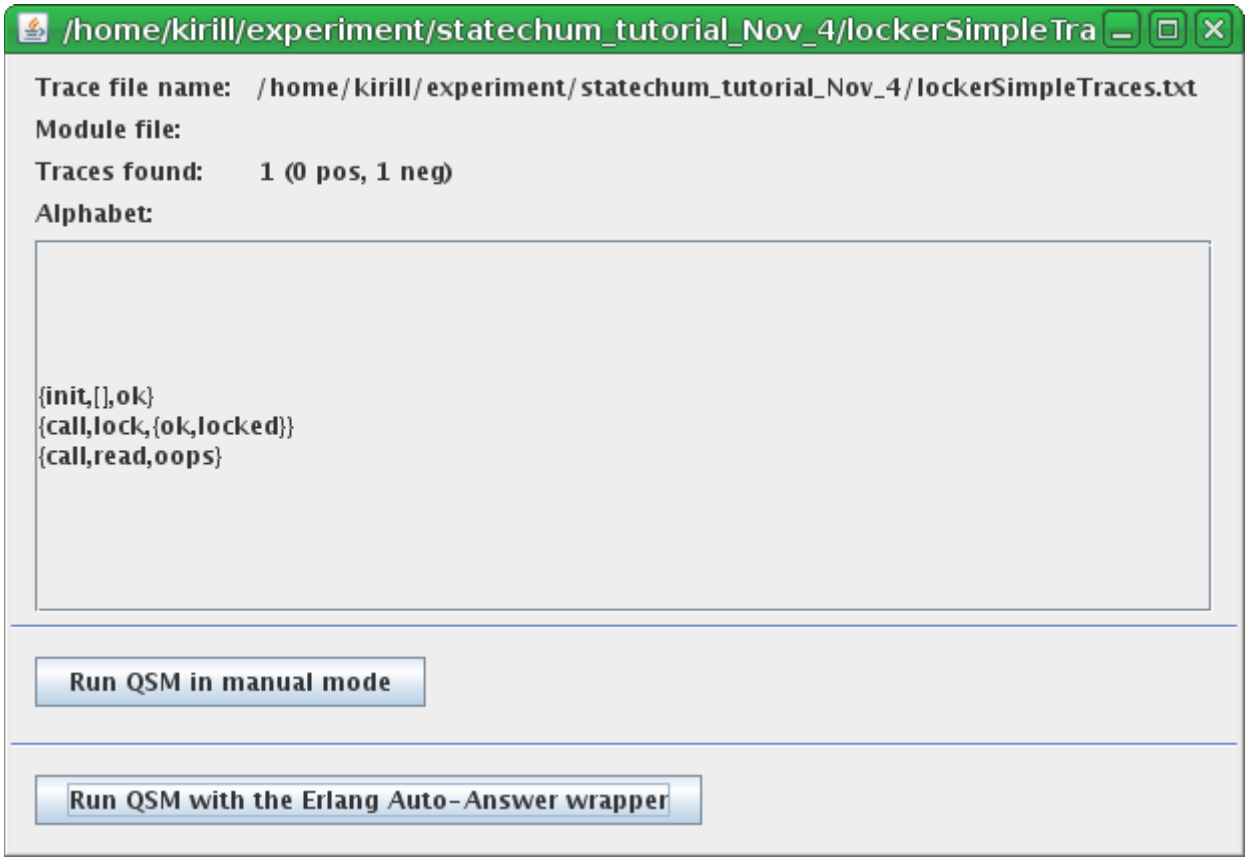
reject the second 'init' call

add domain-specific knowledge

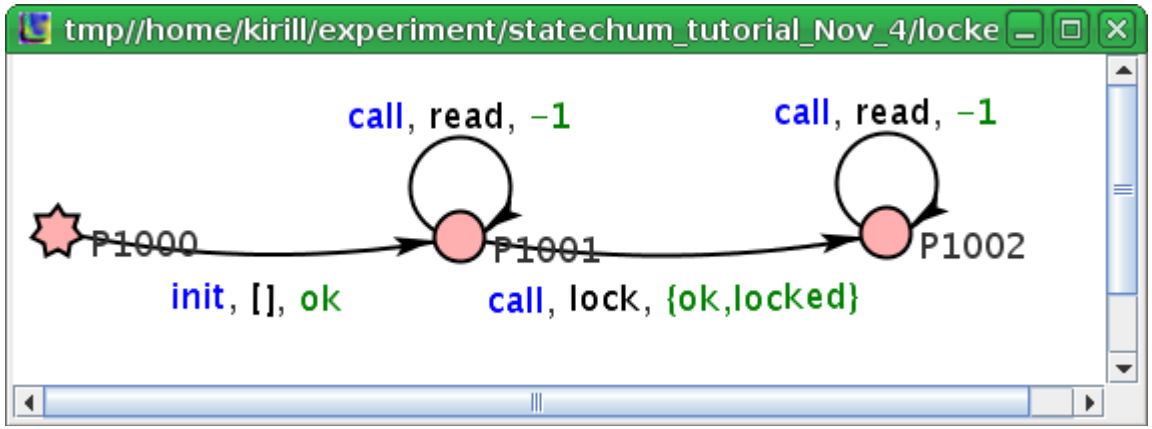
Added for Erlang to add positive/negative traces

Naughty trace file





Let's infer from these traces, 1/2

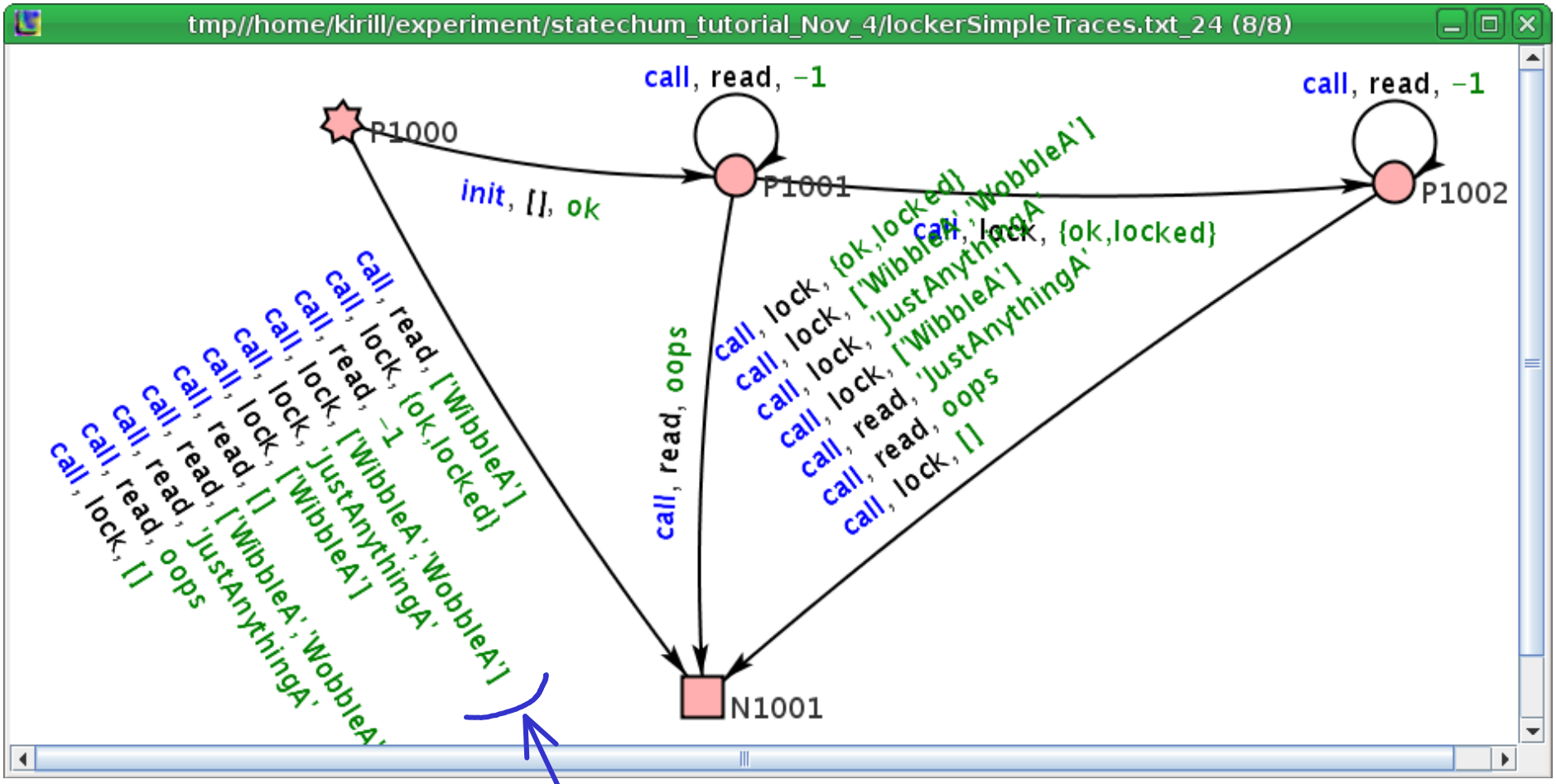


Asked erlang about {call, read ,oops}
and the answer is TRACE_FAIL:
{call, read ,oops}
Asked erlang about {init, [] ,ok}{call, read ,-1}{call, read ,oops}
and the answer is TRACE_DIFFERENTOUTPUT:
{init, [] ,ok}{call, read ,-1}{call, read ,-1}

expected, got

Let's infer from these traces, 2/2

this time with reject-state visible.



These have been generated
from the output of "typer".

Where typer is used

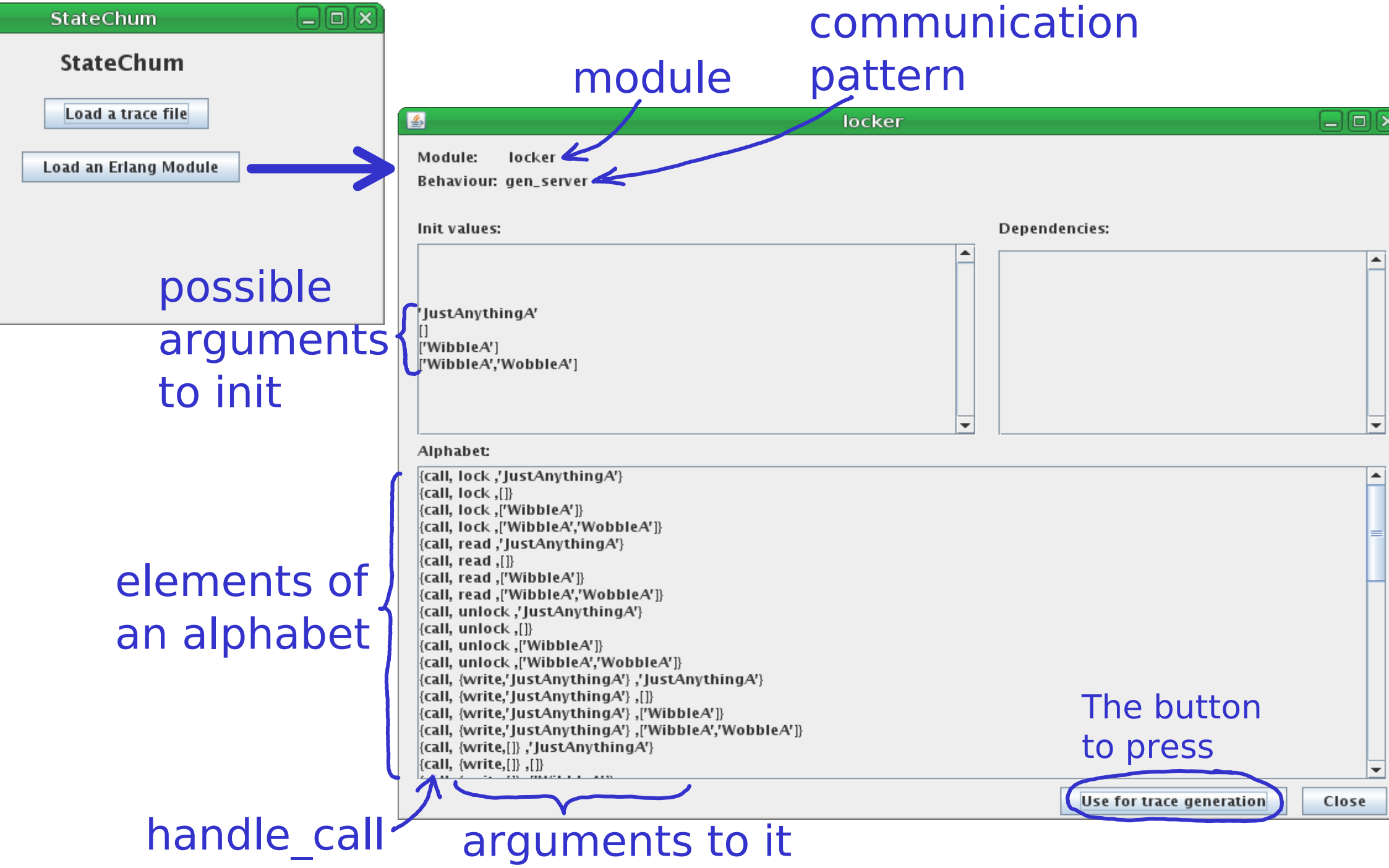
- Rather often, there is not enough traces,
  Why not generate them randomly?

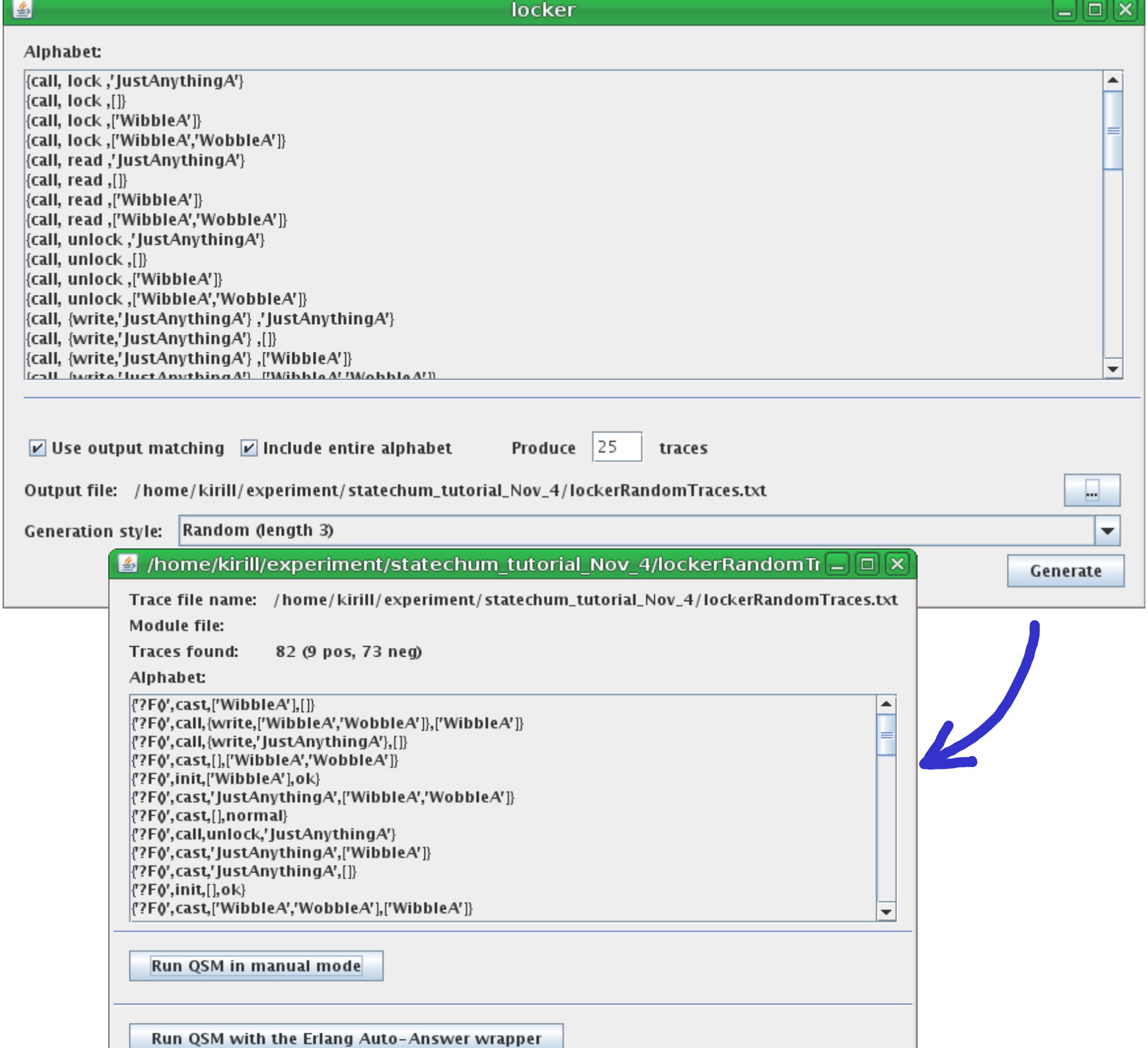
- We do not have test data, but can use typer to obtain
  data types.
  (one can also use Quickcheck for this).

```
%% File: "locker.erl"
%% ------------------
-spec init(_) -> {'ok',{'unlocked',-1}}.
-spec handle_call('lock' | 'read' | 'unlock' | {'write',_},{_,_}) ->
        {'reply',_,{_,_}}.
-spec handle_call('lock' | 'read' | 'unlock' | {'write',_},_,{_,_}) ->
        {'reply',_,{_,_}}.
-spec handle_cast(_,_) -> {'noreply',_} | {'stop','normal','stopped'}.
-spec terminate(_,_) -> 'ok'.
```

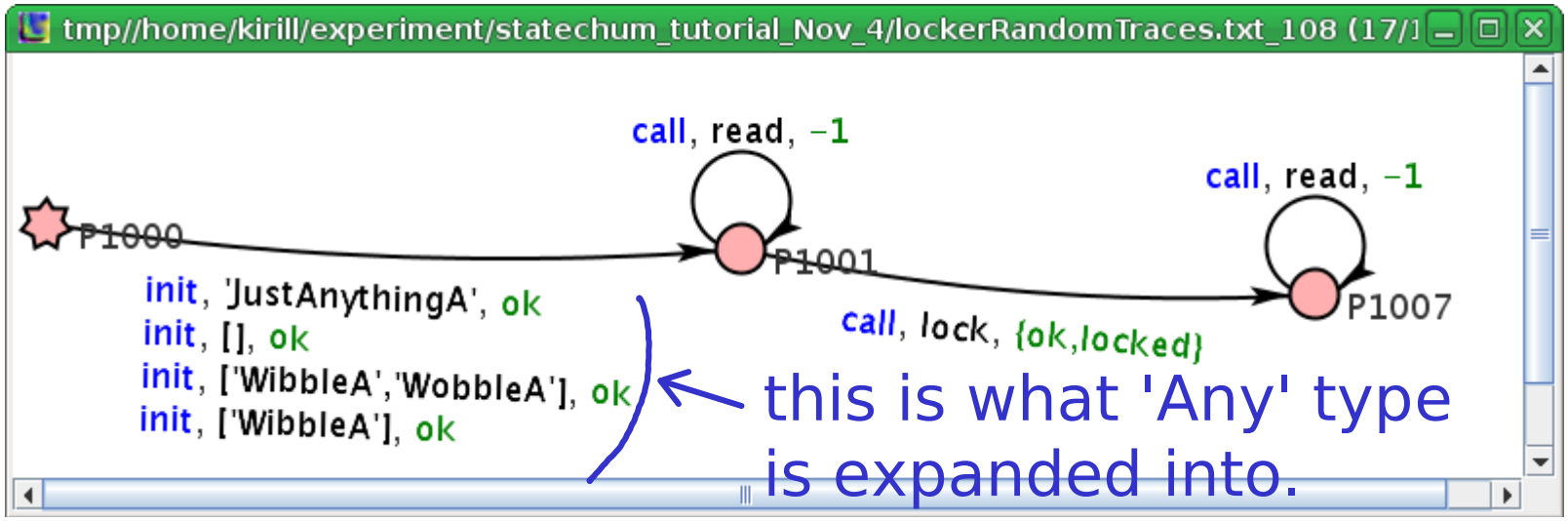
means "any type"

Random alphabet generation




locker
Alphabet:
{call, lock ,'JustAnythingA'}
{call, lock ,[]}
{call, lock ,['WibbleA']}
{call, lock ,['WibbleA','WobbleA']}
{call, read ,'JustAnythingA'}
{call, read ,[]}
{call, read ,['WibbleA']}
{call, read ,['WibbleA','WobbleA']}
{call, unlock ,'JustAnythingA'}
{call, unlock ,[]}
{call, unlock ,['WibbleA']}
{call, unlock ,['WibbleA','WobbleA']}
{call, {write,'JustAnythingA'} ,'JustAnythingA'}
{call, {write,'JustAnythingA'} ,[]}
{call, {write,'JustAnythingA'} ,['WibbleA']}
Use output matching
Include entire alphabet
Produce
25
traces
Output file: /home/kirill/experiment/statechum_tutorial_Nov_4/lockerRandomTraces.txt
...
Generation style:
Random (length 3)
Generate
/home/kirill/experiment/statechum_tutorial_Nov_4/lockerRandomTr
Trace file name: /home/kirill/experiment/statechum_tutorial_Nov_4/lockerRandomTraces.txt
Module file:
Traces found: 82 (9 pos, 73 neg)
Alphabet:
{'?F0',cast,['WibbleA'],[]}
{'?F0',call,{write,['WibbleA','WobbleA']},['WibbleA']}
{'?F0',call,{write,'JustAnythingA'},[]}
{'?F0',cast,[],['WibbleA','WobbleA']}
{'?F0',init,['WibbleA'],ok}
{'?F0',cast,'JustAnythingA',['WibbleA','WobbleA']}
{'?F0',cast,[],normal}
{'?F0',call,unlock,'JustAnythingA'}
{'?F0',cast,'JustAnythingA',['WibbleA']}
{'?F0',cast,'JustAnythingA',[]}
{'?F0',init,[],ok}
{'?F0',cast,['WibbleA','WobbleA'],['WibbleA']}
Run QSM in manual mode
Run QSM with the Erlang Auto-Answer wrapper

And the outcome of learning is ...



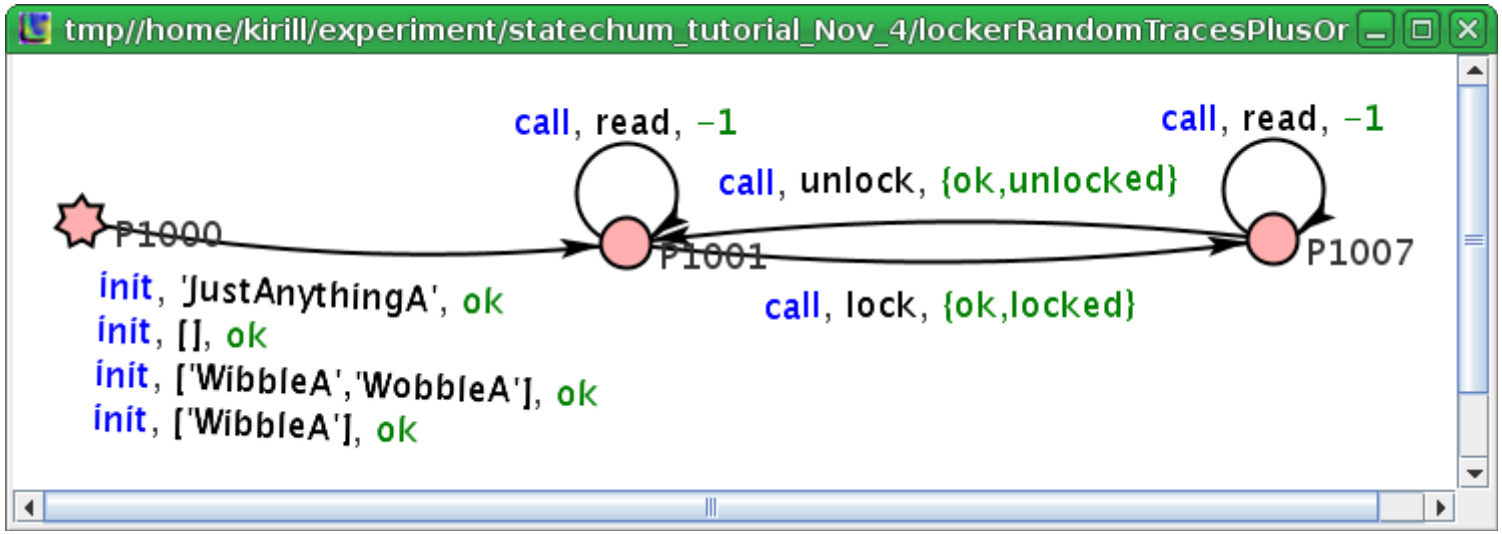
... incomplete - no transition from P1007.

This is because the length of traces was 3, we need longer ones to discover the behavior of that state.

We could help the learner by offering a new trace.


```
+ [[  {?F(),'init',[],'ok'},{?F(),'call','lock',{ok,locked}},
      {?F(),'call','read',-1},{?F(),'call','unlock',{ok,unlocked}},
      {?F(),'call','read',-1}]]
```

What a trace is loaded, this is replaced by the prototype of the function.



The number of questions, 1/2

```
config erlangDisplayStatistics true
config certaintyThreshold -1
```

threshold when to merge without asking questions

```
STATISTICS: 88 questions, 2 restarts
```

```
config erlangDisplayStatistics true
config certaintyThreshold 4
```

In all 3 cases we get the same automaton!

```
STATISTICS: 46 questions, 3 restarts
```

```
config erlangDisplayStatistics true
config certaintyThreshold 2
```

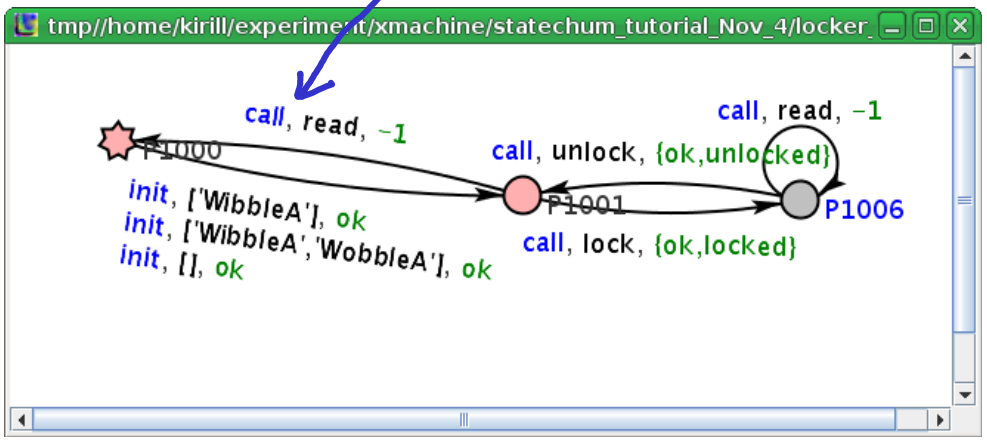
```
STATISTICS: 27 questions, 3 restarts
```

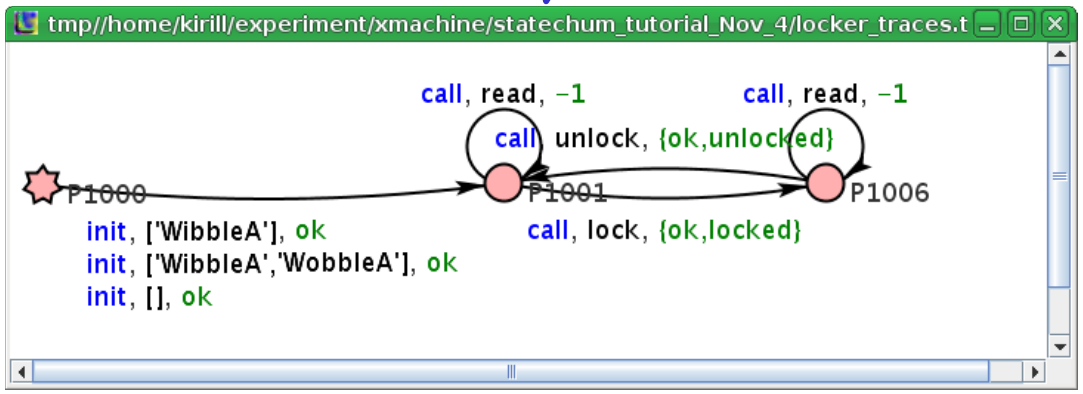
The number of questions, 2/2

What if we try a threshold of 1 ?

config erlangDisplayStatistics true
config certaintyThreshold 1

wrong target state for this transition, compared to

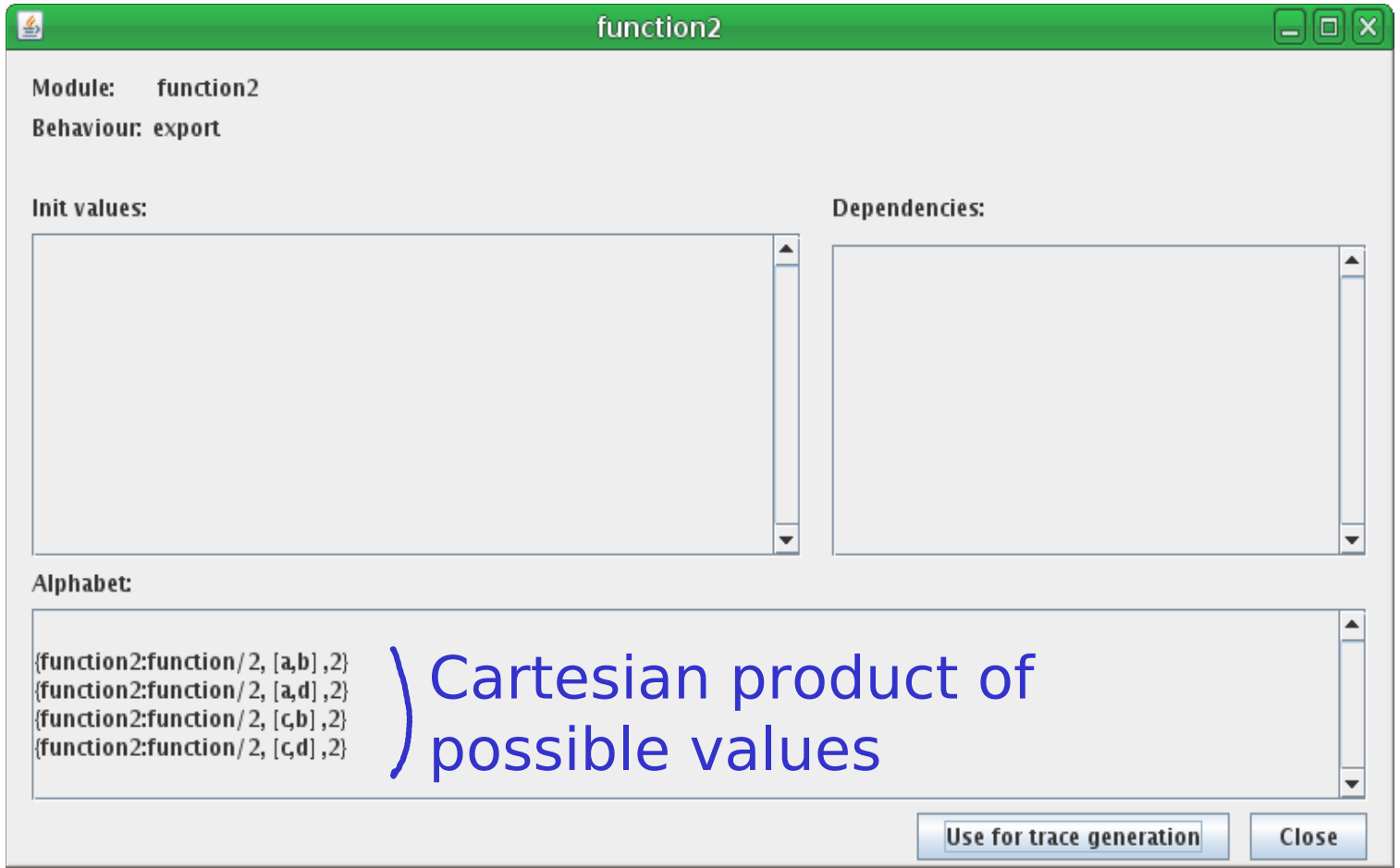




A more interesting type

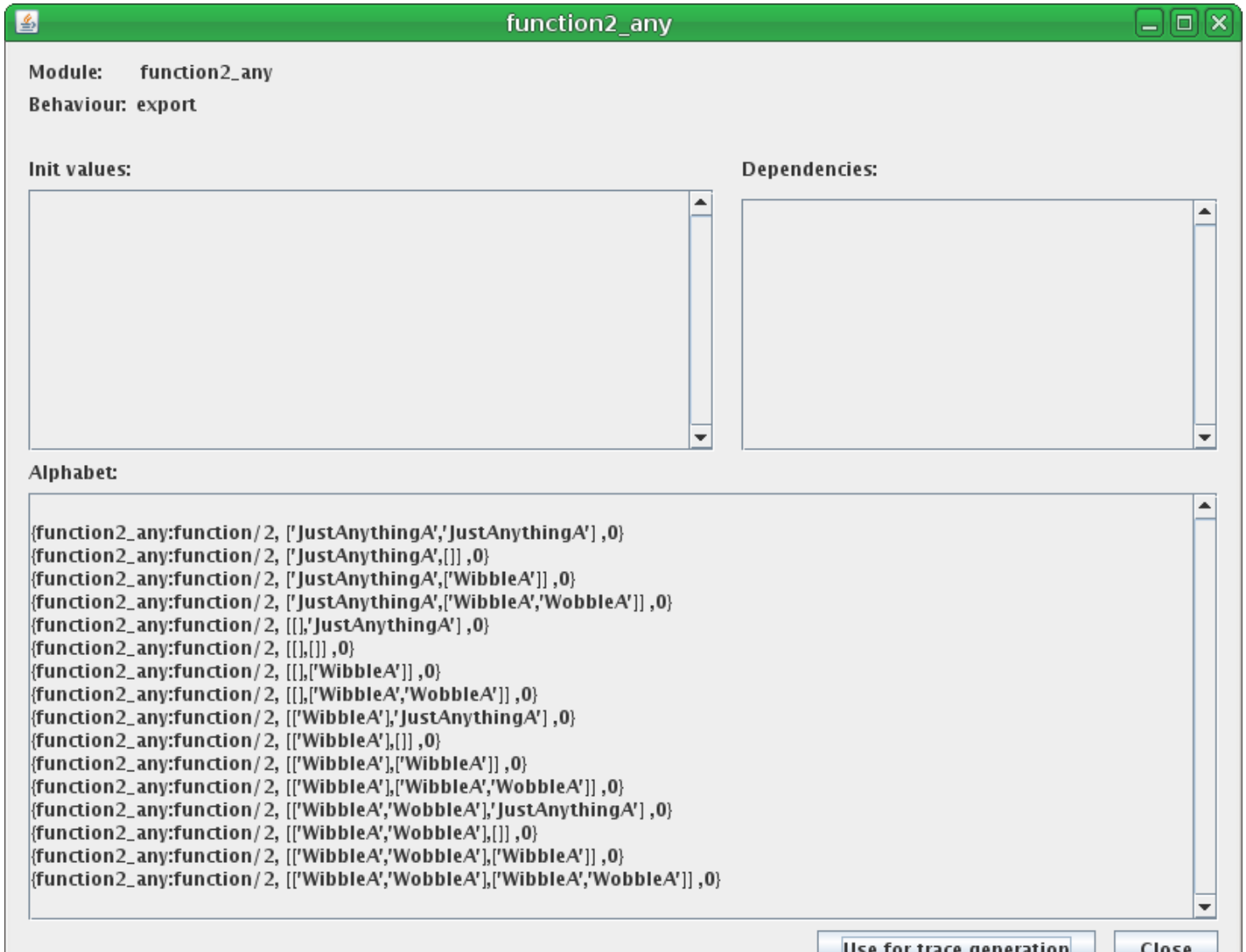
```
-module(function2).
-export([function/2]).

function('a','b') -> 2;
function('c','d') -> 3.
```

```
%% File: "function2.erl"
%% --------------------
-spec function('a' | 'c','b' | 'd') -> 2 | 3.
```



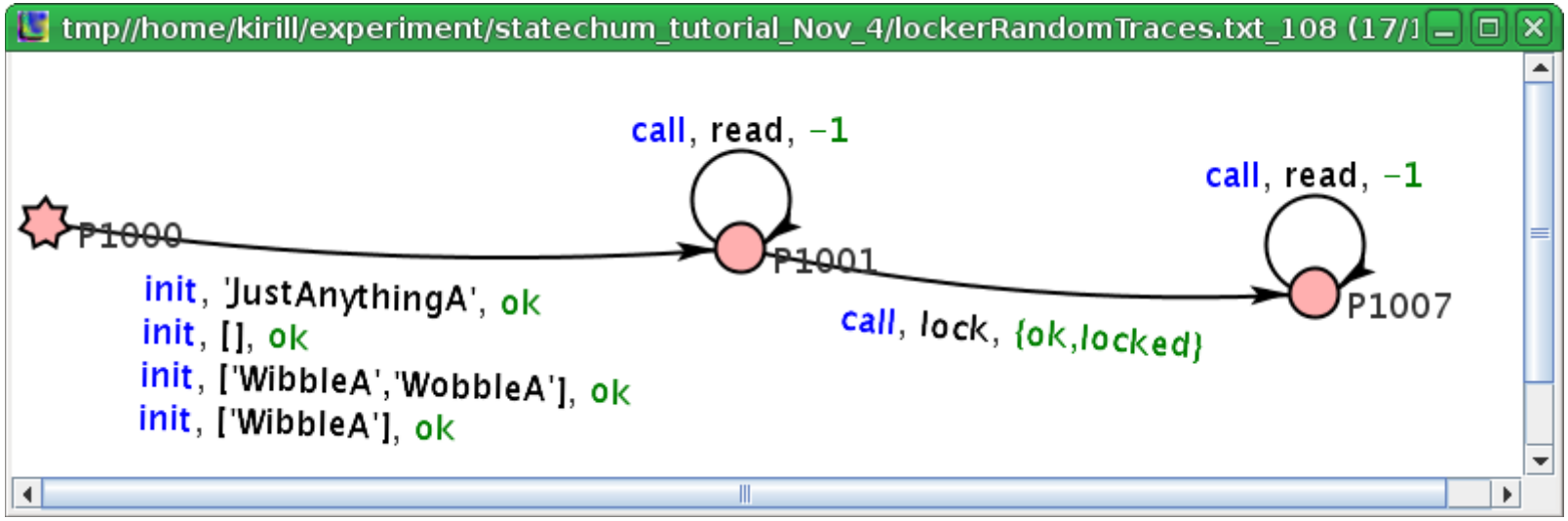
```
-module(function2_any).
-export([function/2]).
function('a','b') -> 2;
function('c','d') -> 3;
function(_X,_Y) -> 0.
```

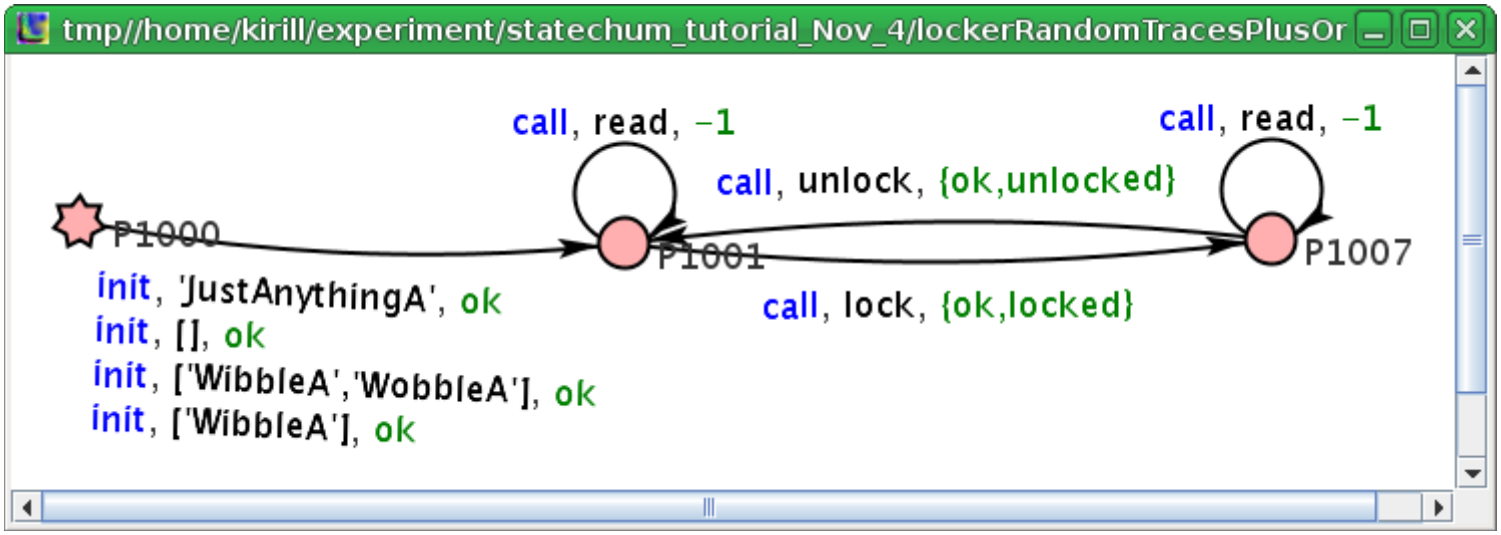
```
%% File: "function2_any.erl"
%% ----------------------
-spec function(_,_) -> 0 | 2 | 3.
```



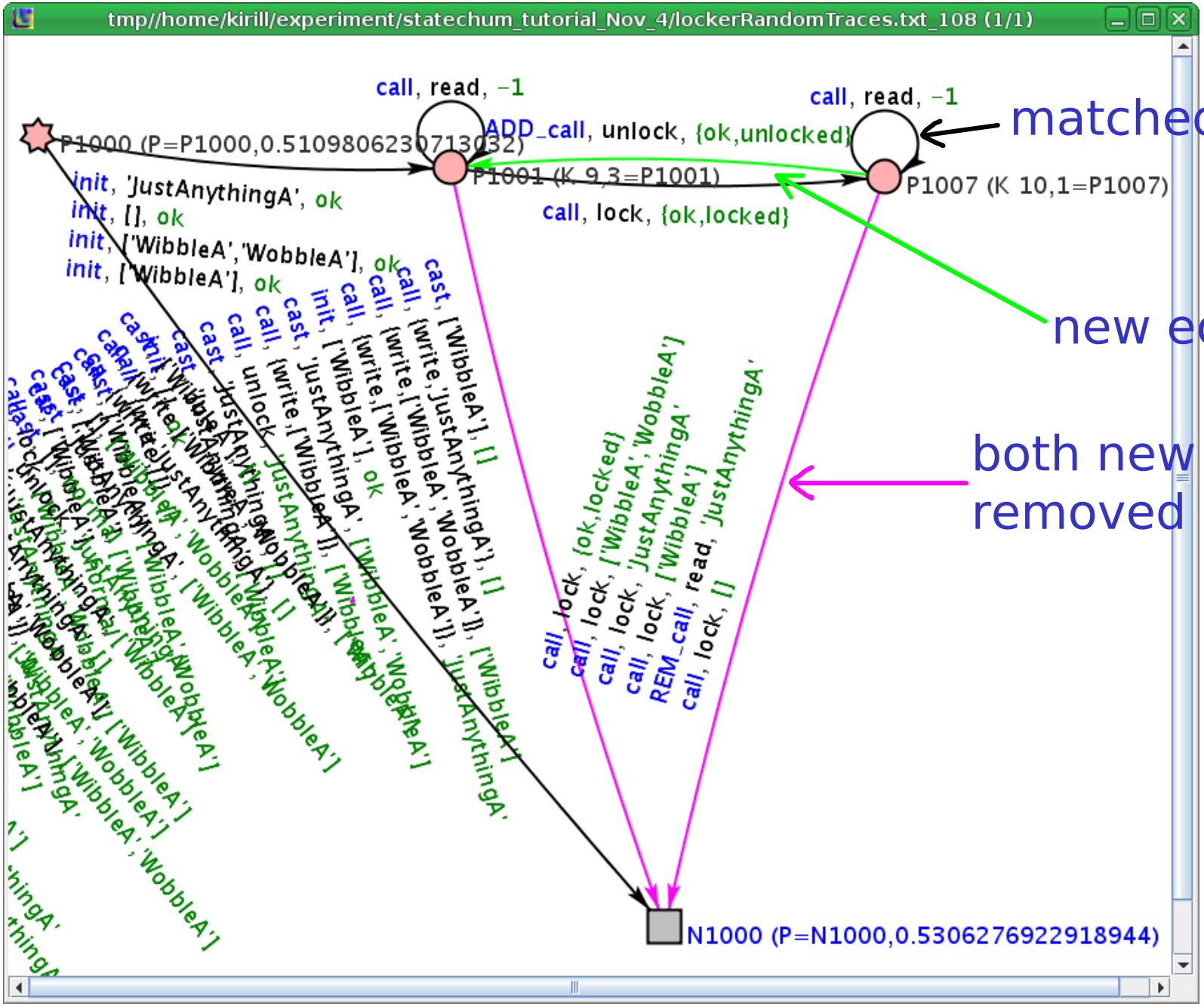
Graph difference, 1/3

Compare the outcome of learning of the two graphs,

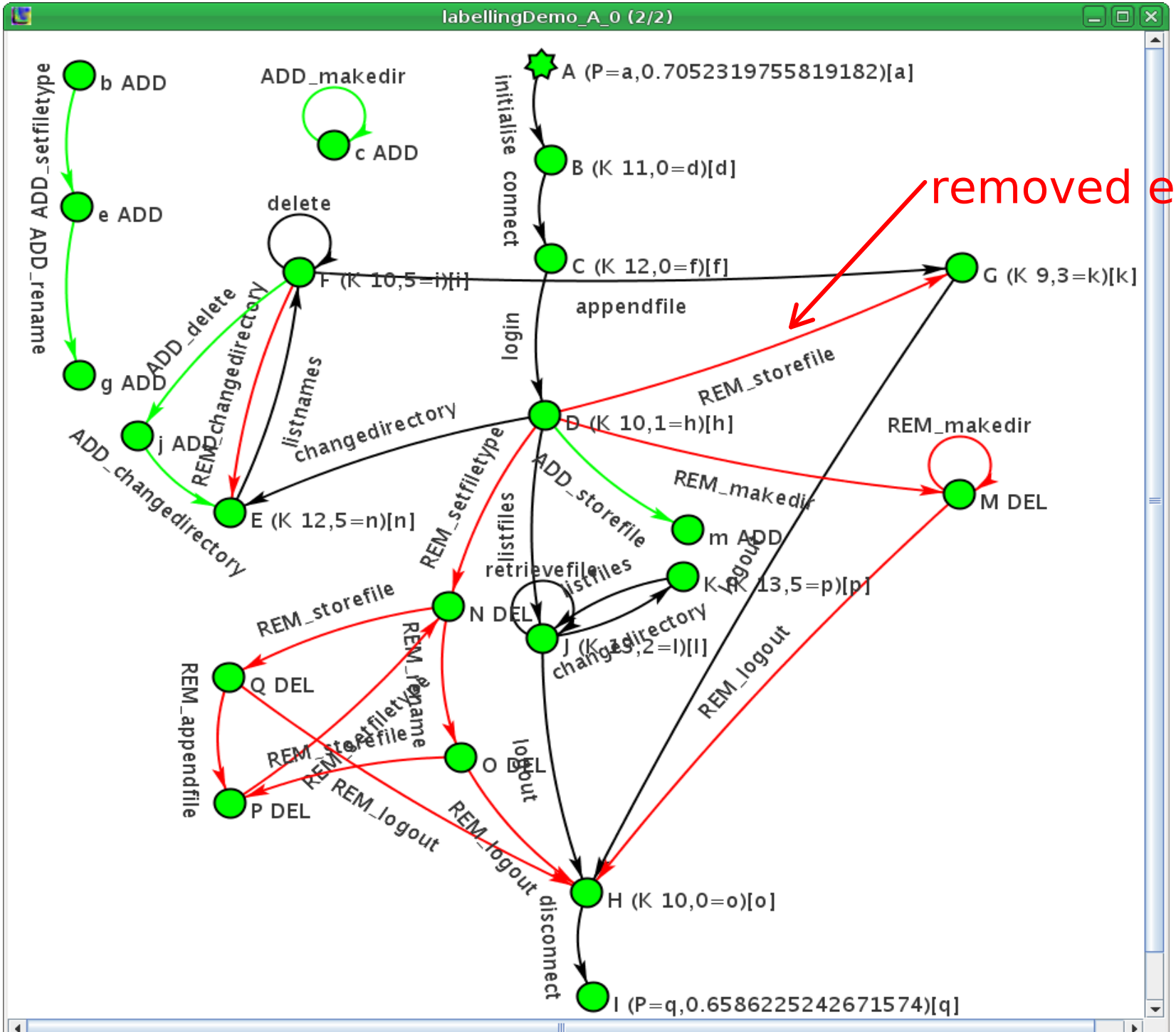




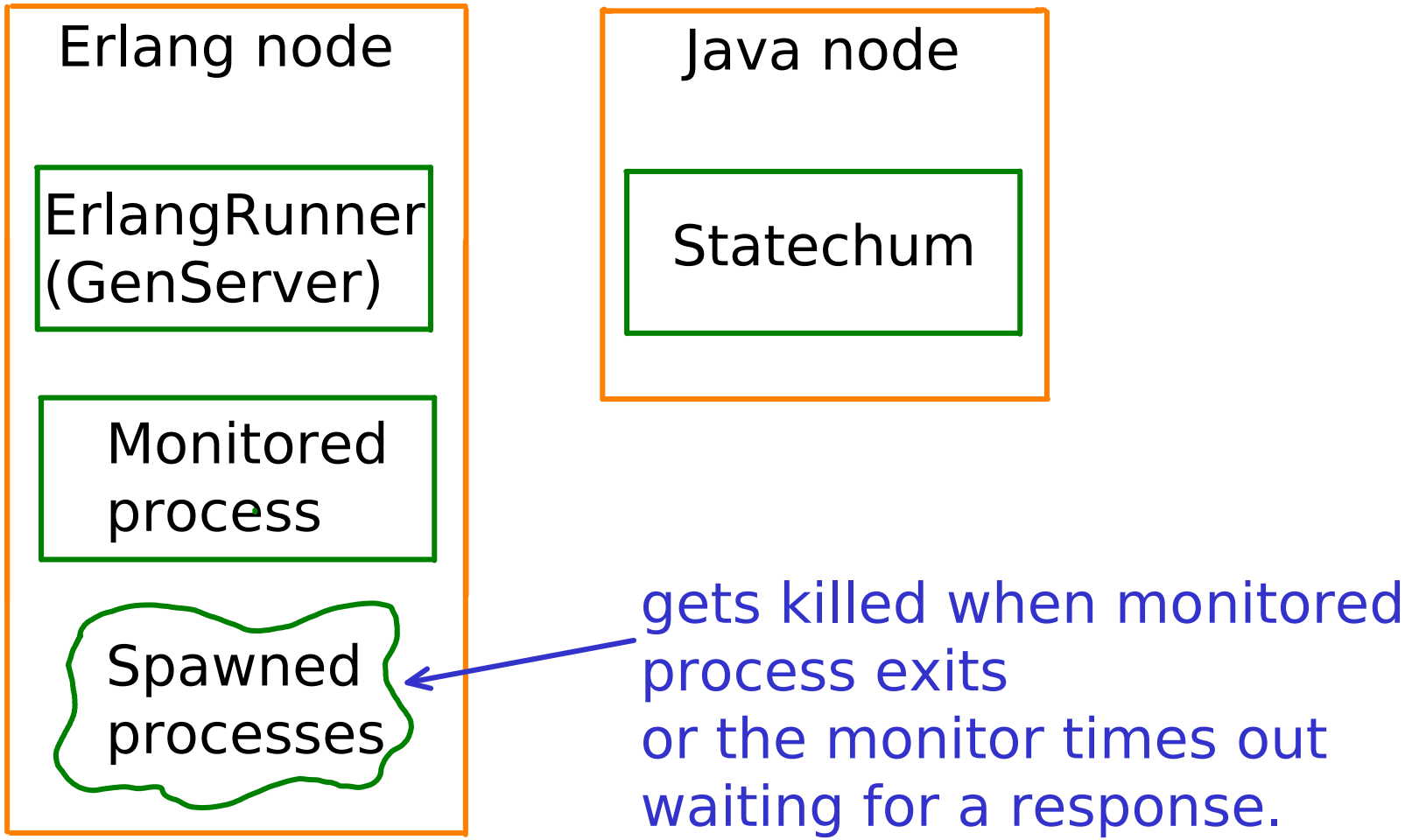
Graph difference, 2/3



Graph difference, 3/3



Interface to Erlang



Installation

- Available from statechum.sourceforge.net
- Can be checked out from SVN.
- Written mostly in Java, builds with Eclipse (can also be built with Ant).
- All libraries needed to run the examples are included.
- Other libraries include:
  - C version of the backend for graph diff.
  - SPIN modelchecker and lbl2ba tool.
  - Integration with the R tool.
  - Yices SMT solver.
- How to run is described in Visualiser.java