The MCSM Framework

A. Heußner / T. Le Gall / G. Sutre

TACAS 2012
Tool Presentation

Basic Structure

Algorithms

verifi
algo

Finite-Cont

ystem API

Backends

comm
ma

developers	Sutre / Le Gall Heußner
stable release	1.2
development	active
written in	OCaml
OS	cross platform
distribution	source & binary
license	BSD
website	forge,wiki,...

Basic Structure

Algorithms

verification algorithm

controler synthesis algorithm

...

Finite-Control Infinite-Data Transition System API

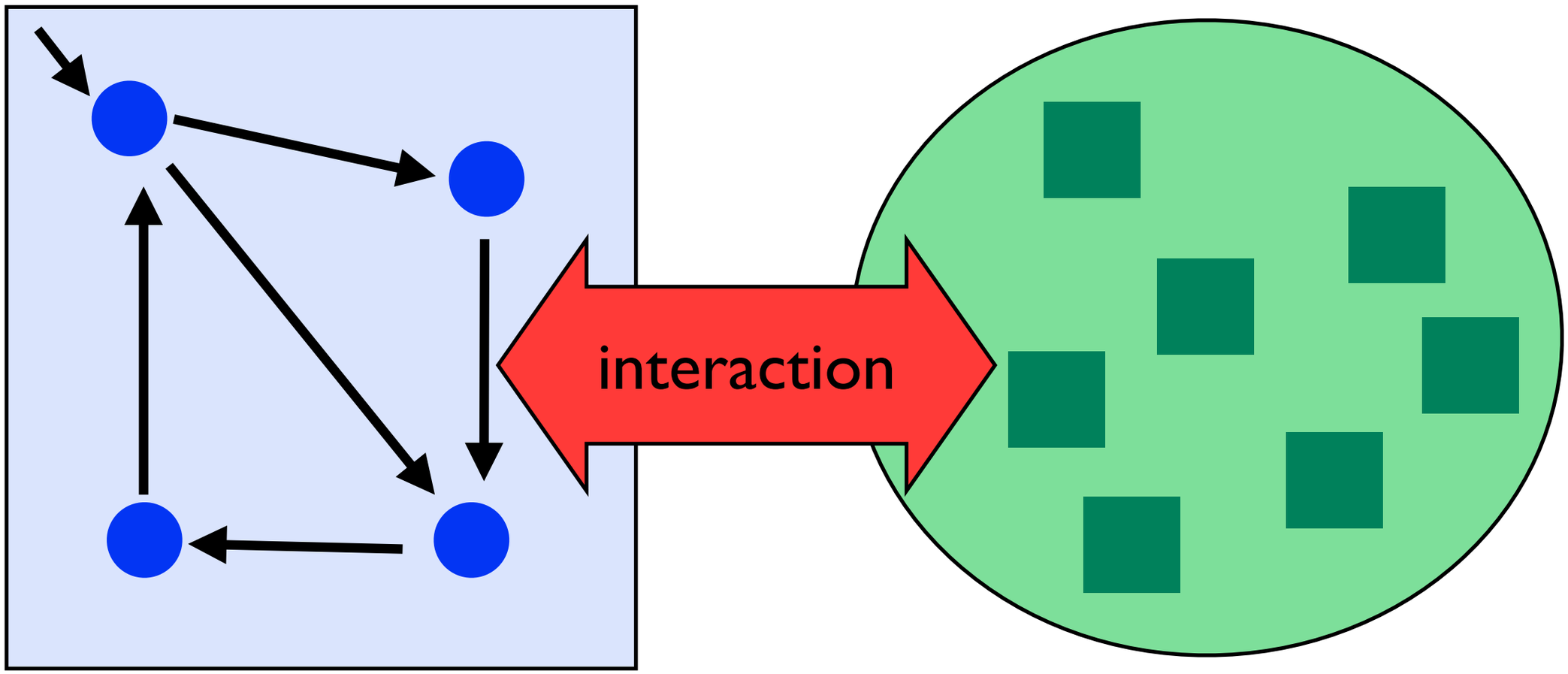
Backends

communicating machines

...

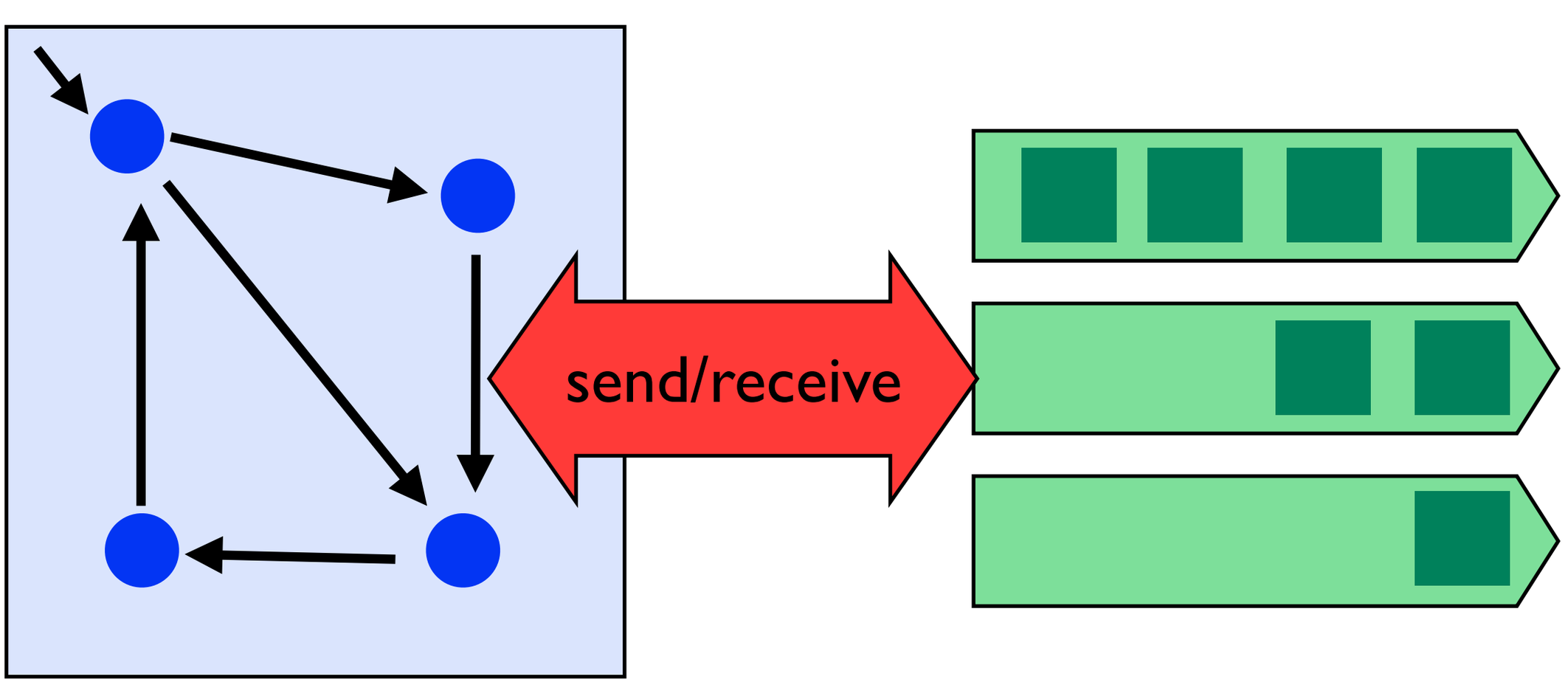
Formal Model

- Finite-Control Infinite-Data Transition Systems



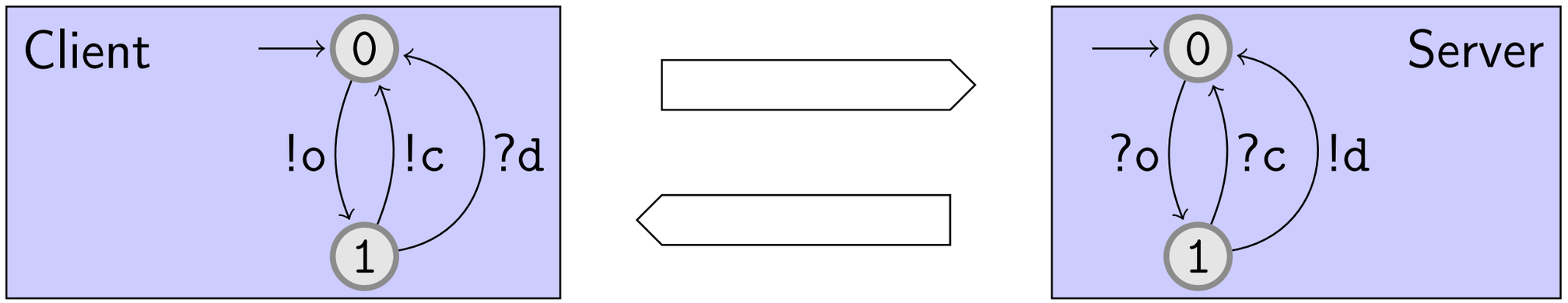
Communicating Machines

- Finite-Control Infinite-Data Transition Systems



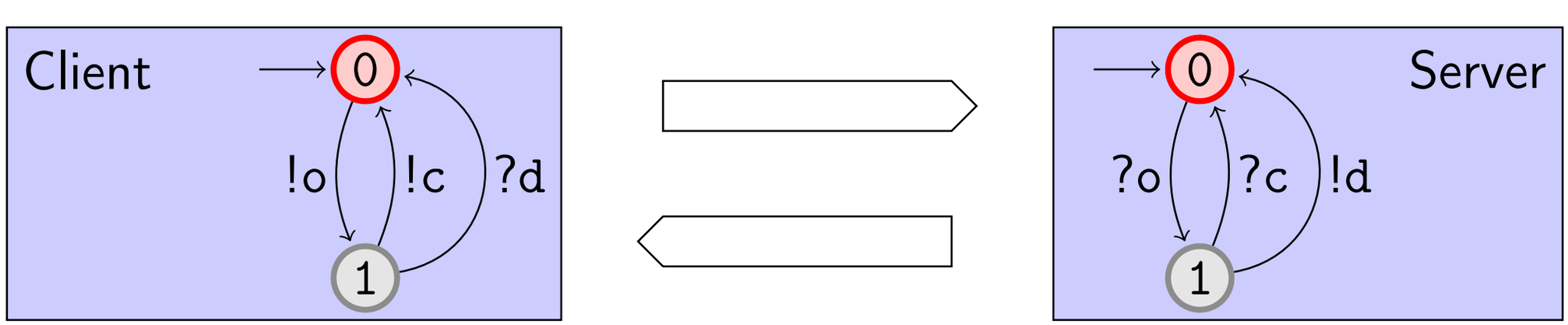
Communicating Machines

- a simple client-server example protocol :



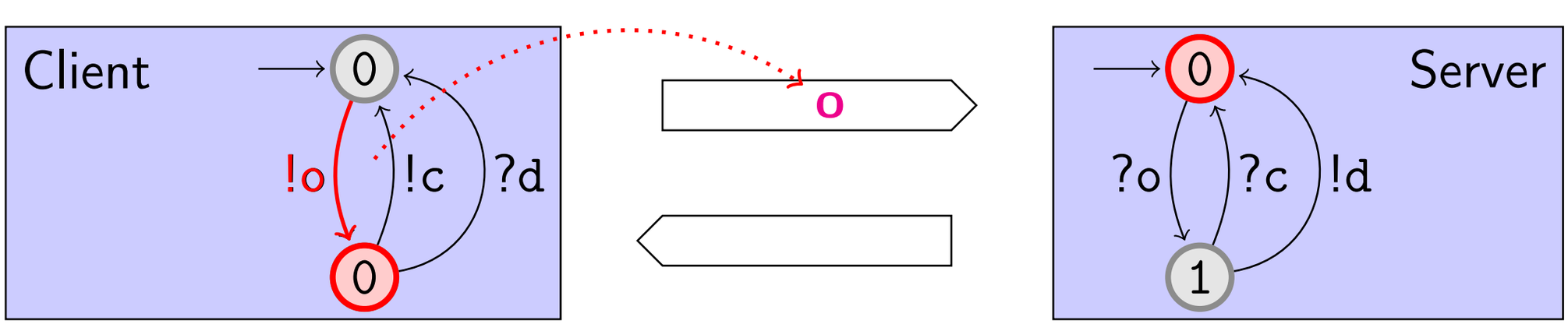
Communicating Machines

- a simple client-server example protocol :



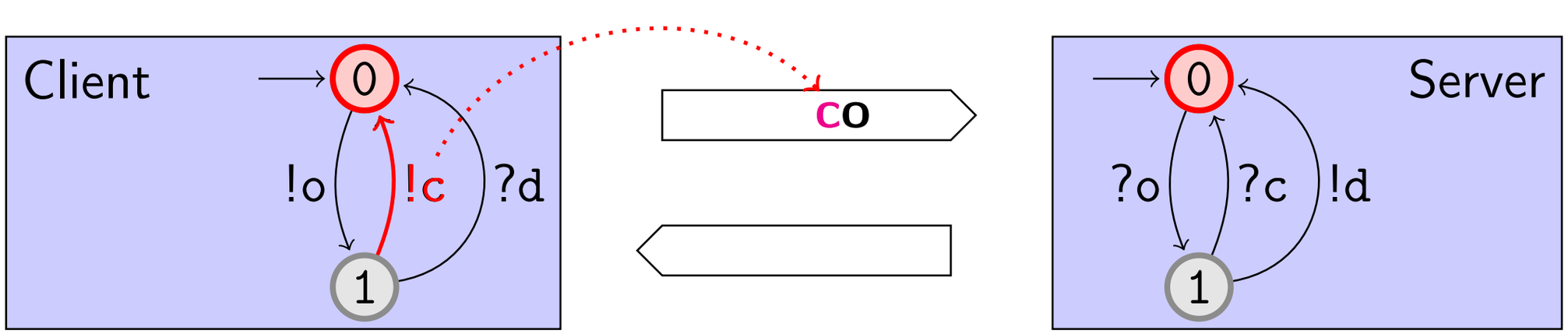
Communicating Machines

- a simple client-server example protocol :



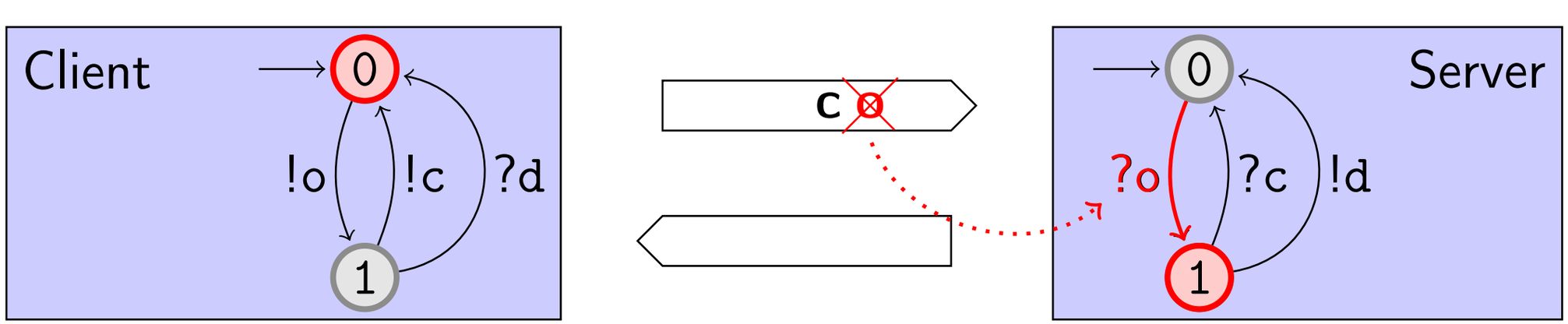
Communicating Machines

- a simple client-server example protocol :



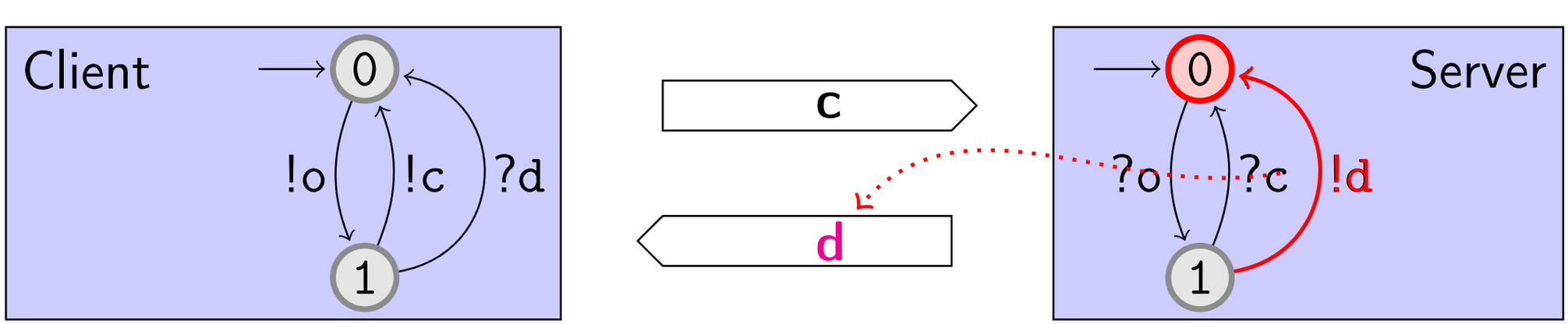
Communicating Machines

- a simple client-server example protocol :



Communicating Machines

- a simple client-server example protocol :



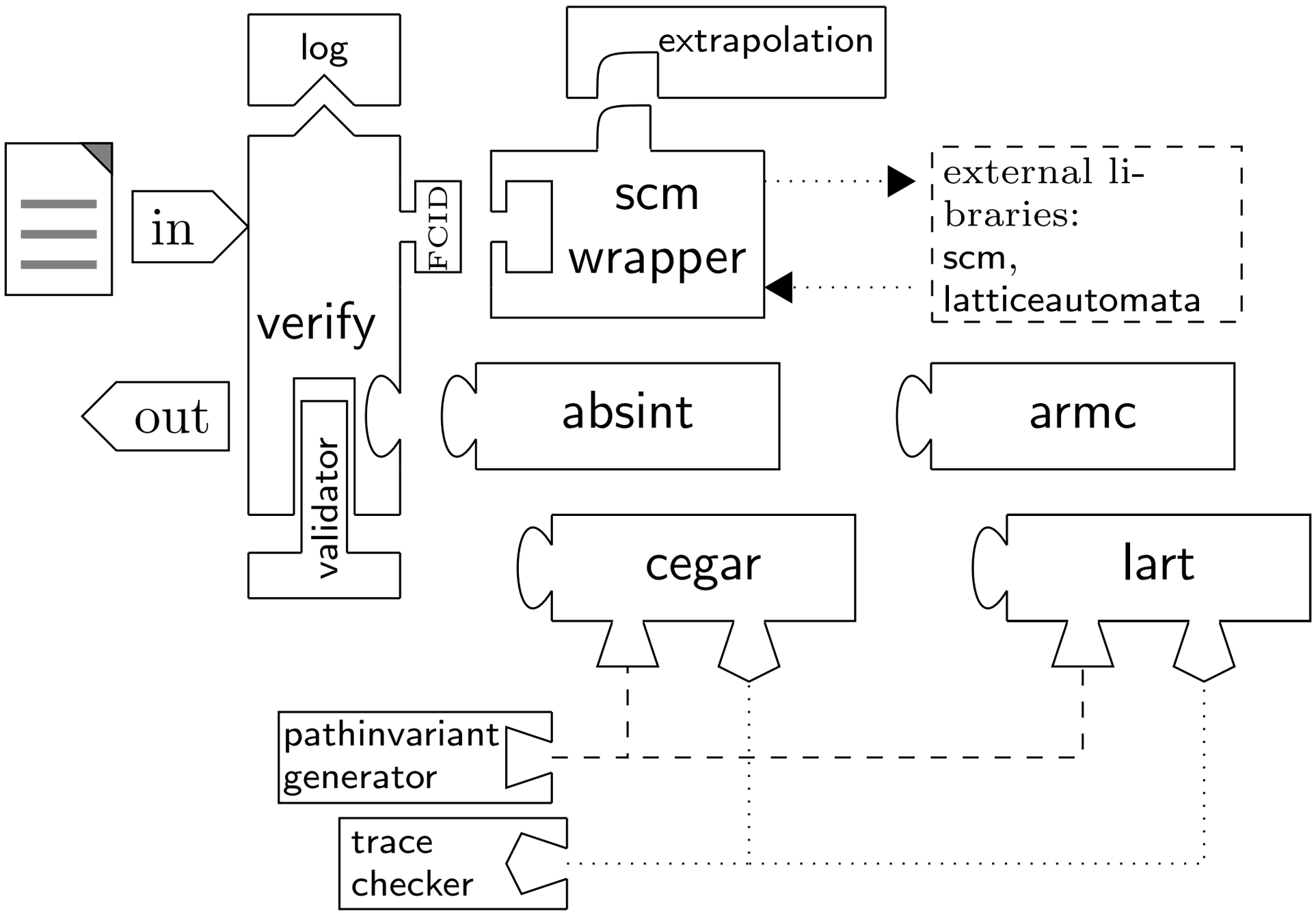
Verification Tool

- safety verification for communicating machines
- common front end for suite of verification algorithms
  - input : automata encoding of CM
  - output :
    - safe (guaranteed by inductive invariant)
    - unsafe + counterexample
    - unknown

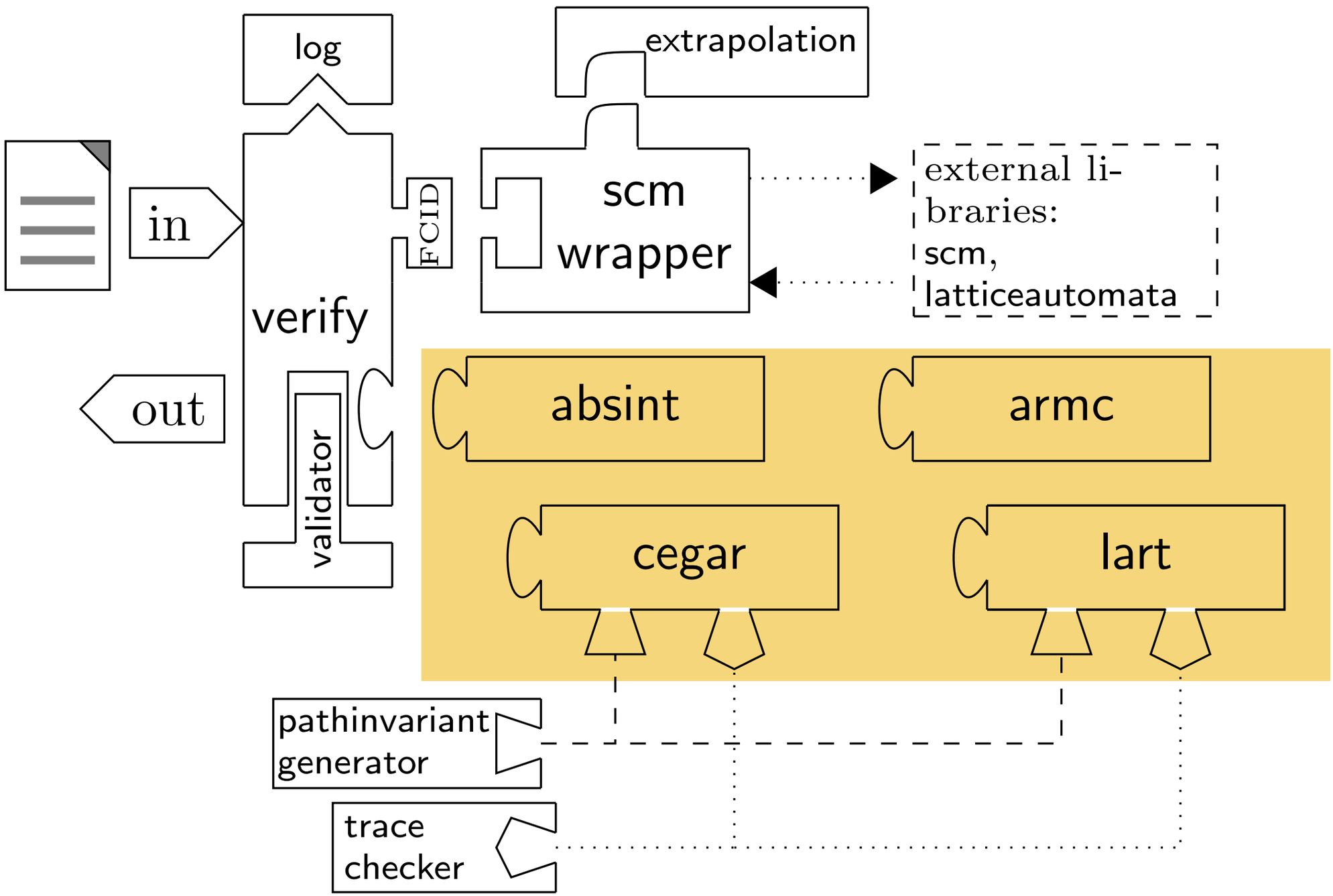
Algorithms

- currently the following algos are supported
  - CEGAR
  - Abstract Interpretation
  - Abstract Regular Model Checking
  - Lazy Abstraction Refinement

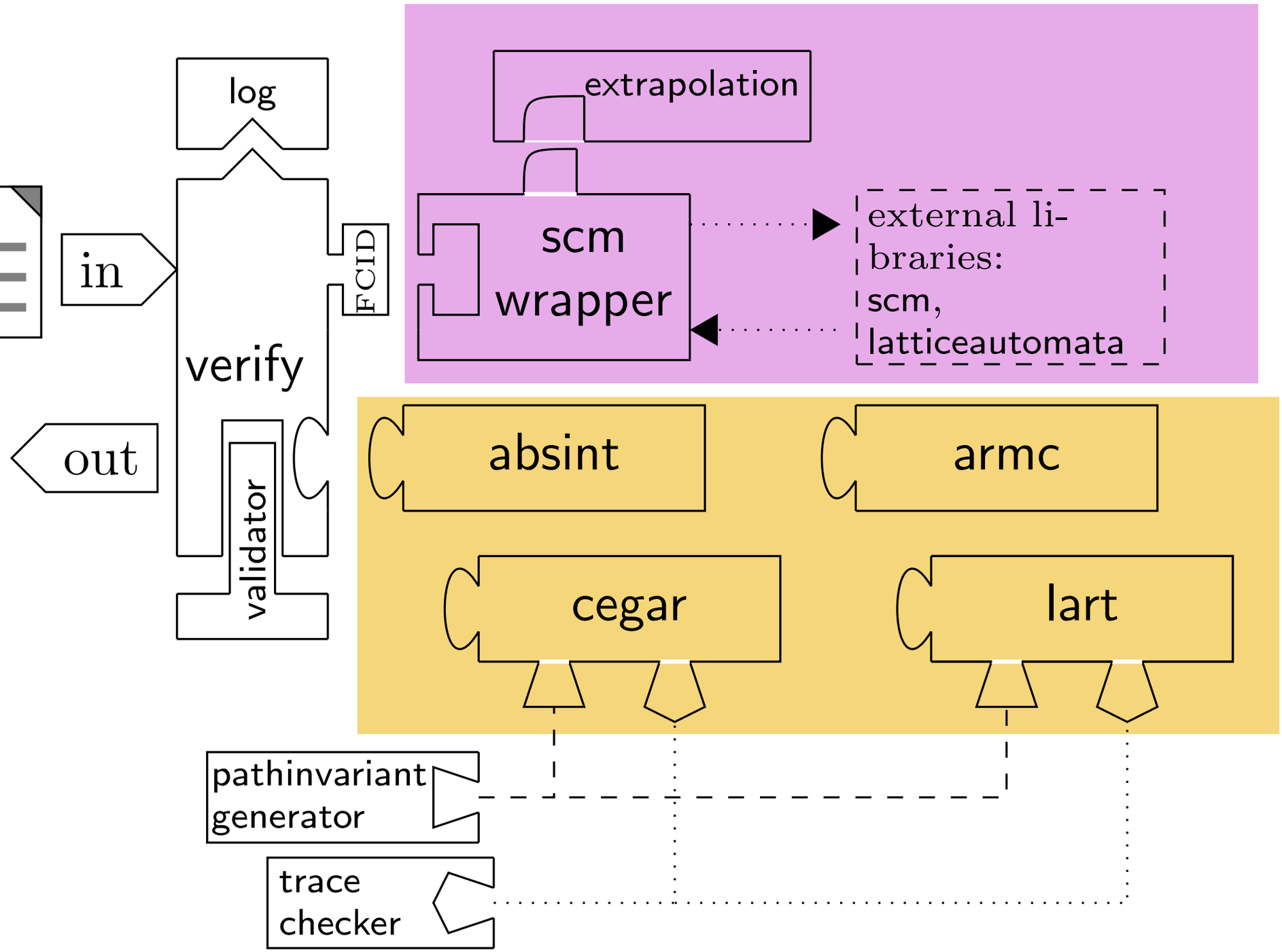
Modularity by McScM



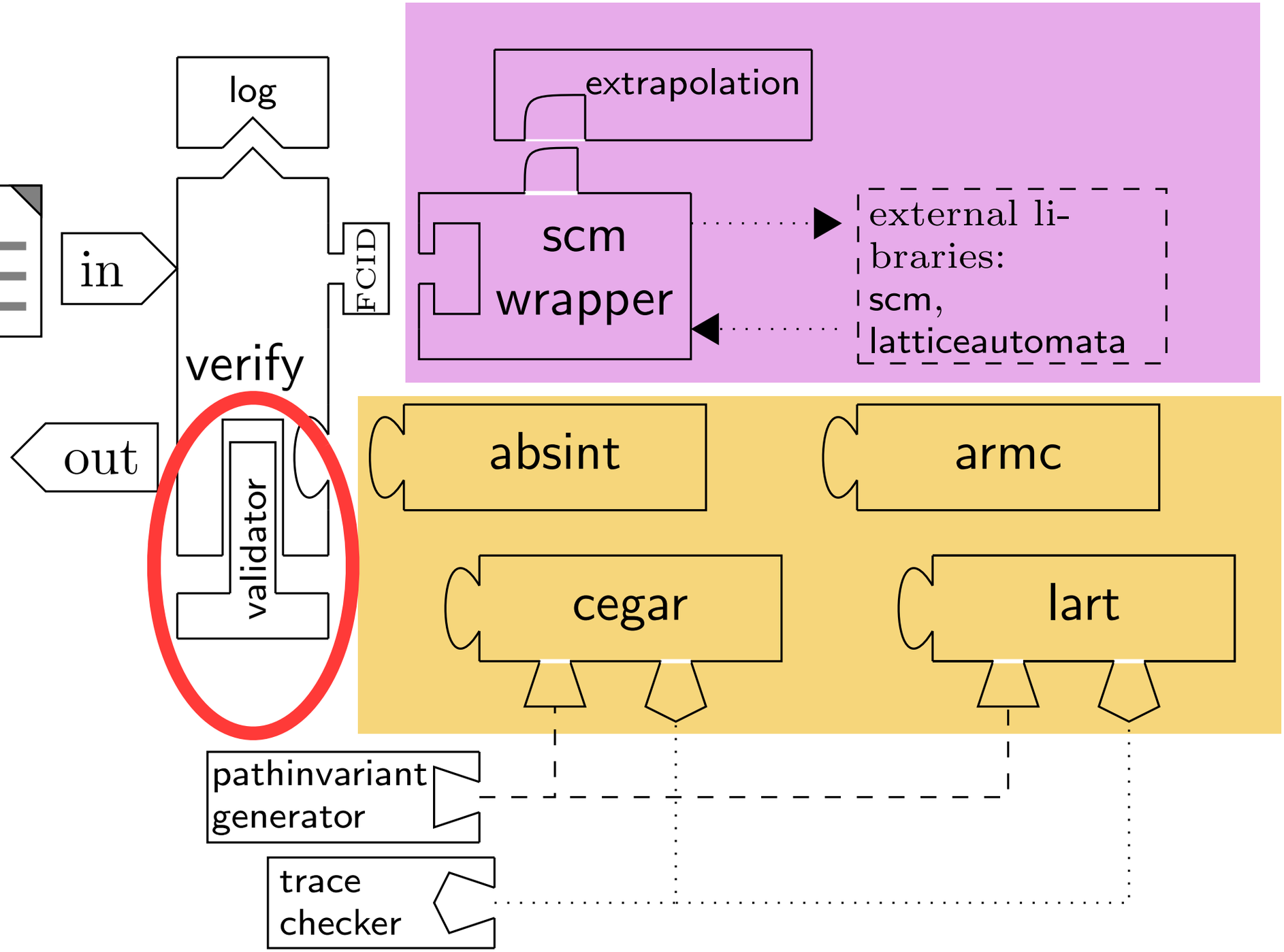
Modularity by McScM



Modularity by McScM



Modularity by McScM



Controller-Synthesis Tool

- supervisory control à la Ramadge & Wonham
  - same input (model/property) as verification tool
  - output : distributed controllers that communicate via given architecture
- includes simulator for CM

SCM Input Format

```
scm connection_disconnection :

nb_channels = 2 ;
parameters :
real o ;
real c ;
real d ;

automaton client :

initial : 0

state 0 :
to 1 :  when true , 0 ! o  ;

state 1 :
to 0 :  when true , 0 ! c ;
to 0 :  when true , 1 ? d ;
```

```
automaton server :

initial : 0

state 0 :
to 1 :  when true , 0 ? o ;

state 1 :
to 0 :  when true , 0 ? c ;
to 0 :  when true , 1 ! d ;


bad_states:
(automaton receiver:
 in 0 : true with c.(o|c)^*.#.d^*)
```

SCM Input Format

```
scm connection_disconnection :

nb_channels = 2 ;
parameters :
real o ;
real c ;
real d ;

automaton client :

initial : 0

state 0 :
to 1 :  when true , 0 ! o  ;

state 1 :
to 0 :  when true , 0 ! c ;
to 0 :  when true , 1 ? d ;
```

```
automaton server :

initial : 0

state 0 :
to 1 :  when true , 0 ? o ;

state 1 :
to 0 :  when true , 0 ? c ;
to 0 :  when true , 1 ! d ;


bad_states:
(automaton receiver:
 in 0 : true with c.(o|c)^*.#.d^*)
```

SCM Input Format

```
scm connection_disconnection :

nb_channels = 2 ;
parameters :
real o ;
real c ;
real d ;
```

```
automaton client :

initial : 0

state 0 :
to 1 :  when true , 0 ! o  ;

state 1 :
to 0 :  when true , 0 ! c ;
to 0 :  when true , 1 ? d ;
```

```
automaton server :

initial : 0

state 0 :
to 1 :  when true , 0 ? o ;

state 1 :
to 0 :  when true , 0 ? c ;
to 0 :  when true , 1 ! d ;
```

```
bad_states:
(automaton receiver:
 in 0 : true with c.(o|c)^*.#.d^*)
```

SCM Input Format

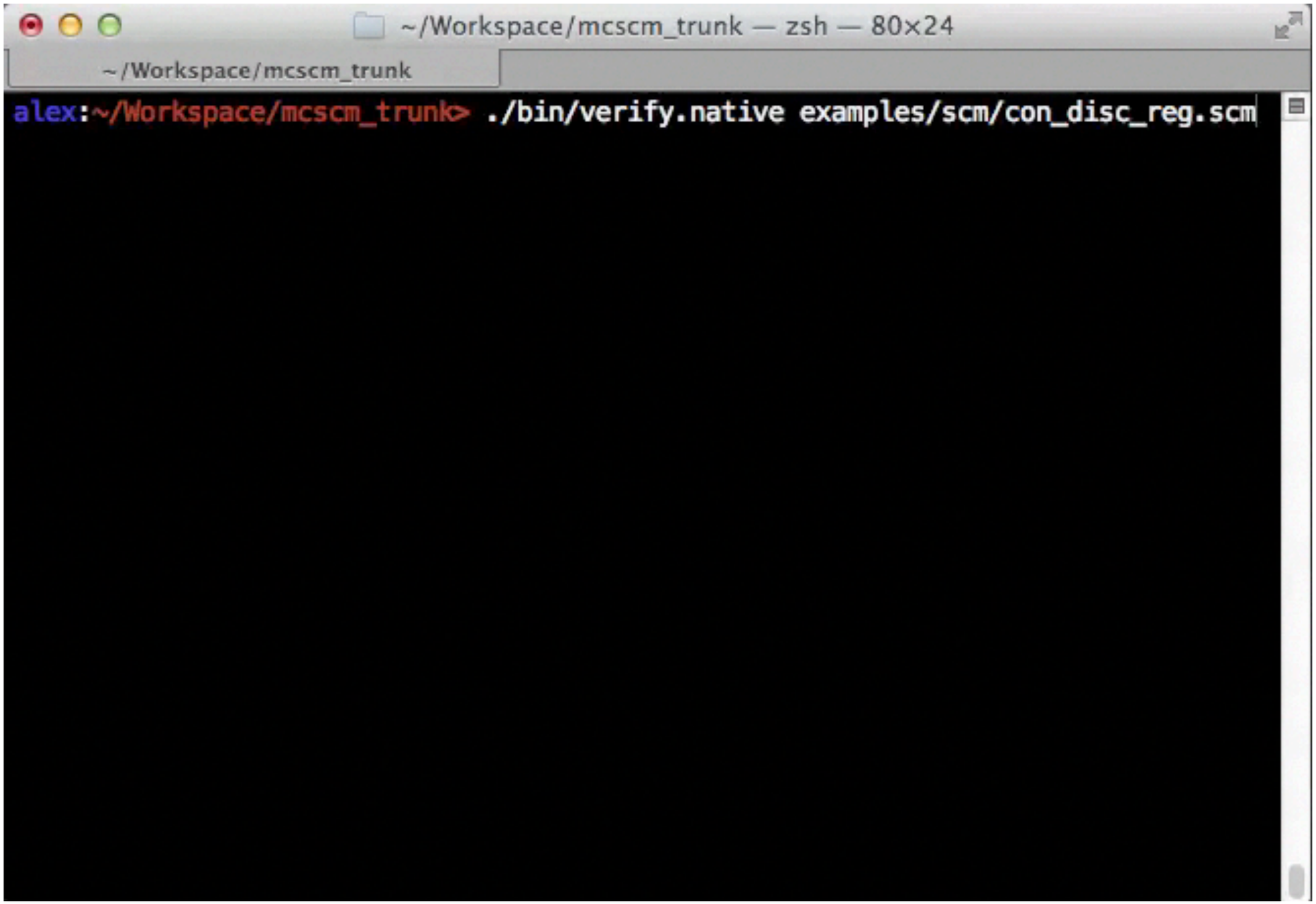
```
scm connection_disconnection :

nb_channels = 2 ;
parameters :
real o ;
real c ;
real d ;
```

```
automaton client :

initial : 0

state 0 :
to 1 :  when true , 0 ! o  ;

state 1 :
to 0 :  when true , 0 ! c ;
to 0 :  when true , 1 ? d ;
```

```
automaton server :

initial : 0

state 0 :
to 1 :  when true , 0 ? o ;

state 1 :
to 0 :  when true , 0 ? c ;
to 0 :  when true , 1 ! d ;
```

```
bad_states:
(automaton receiver:
 in 0 : true with c.(o|c)^*.#.d^*)
```

Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native examples/scm/con_disc_reg.scm

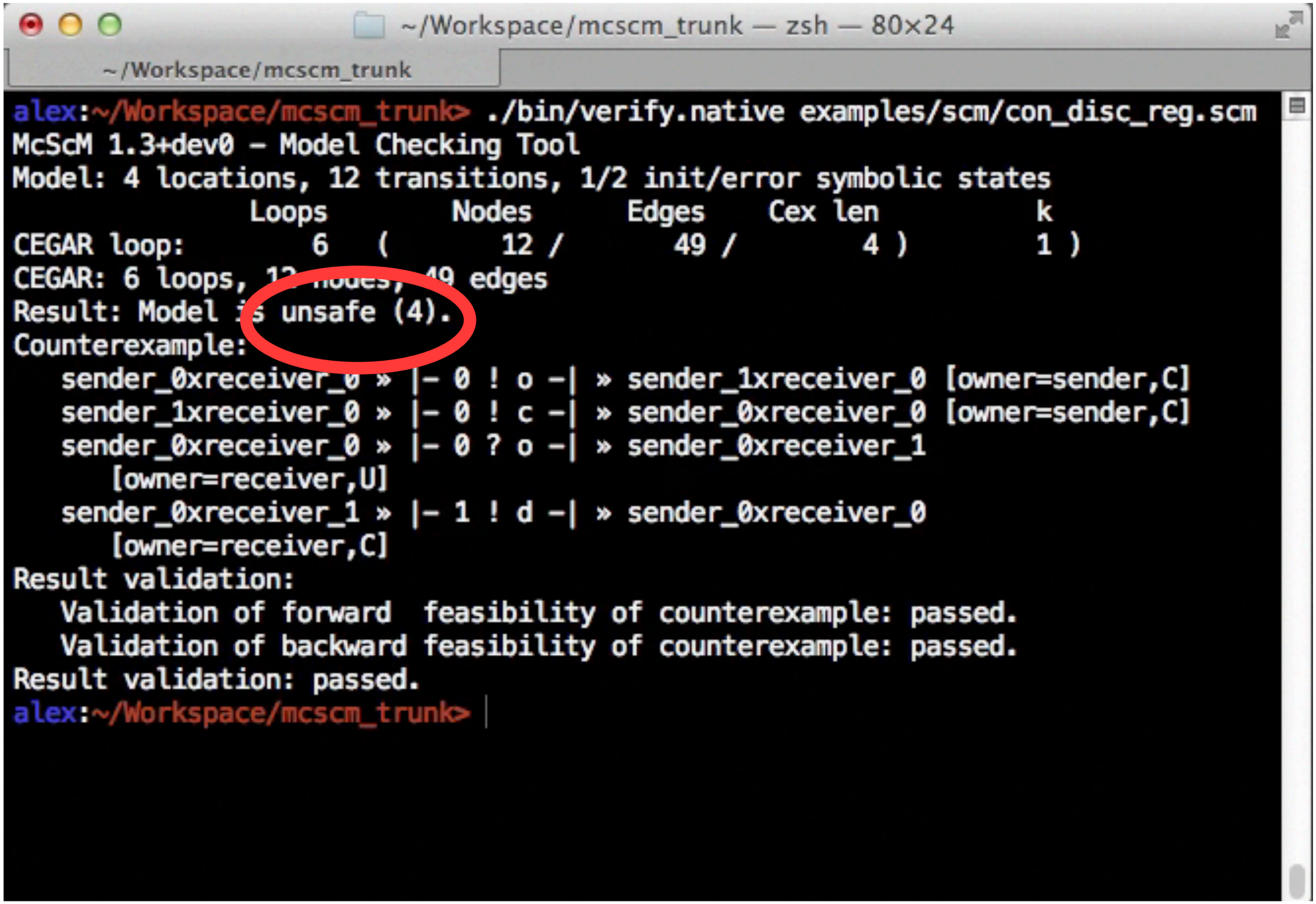
Demo

~/Workspace/mcscm_trunk — zsh — 80×24

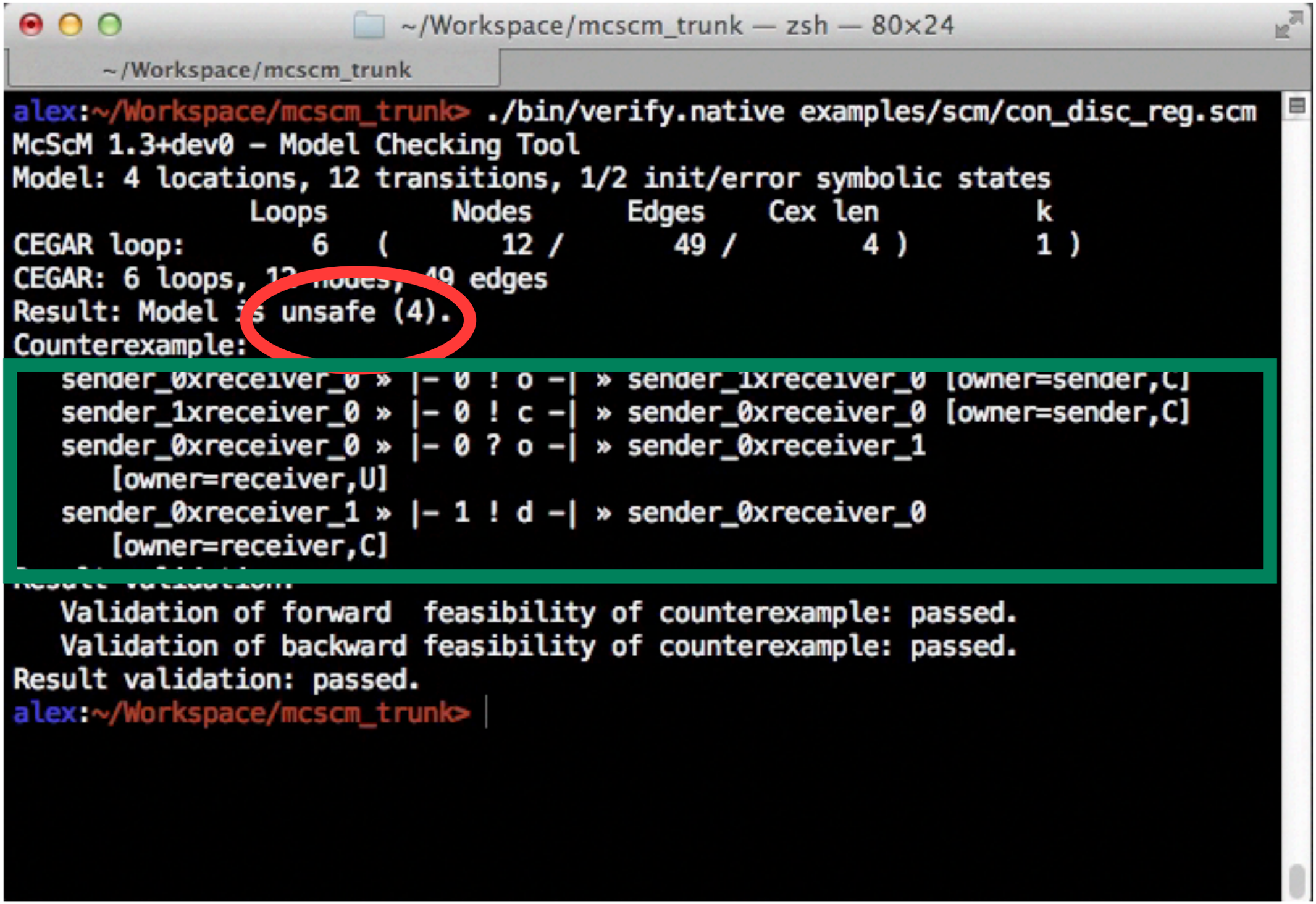
~/Workspace/mcscm_trunk

```
alex:~/Workspace/mcscm_trunk> ./bin/verify.native examples/scm/con_disc_reg.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states
                Loops        Nodes      Edges    Cex len           k
CEGAR loop:         6   (       12 /       49 /         4 )       1 )
CEGAR: 6 loops, 12 nodes, 49 edges
Result: Model is unsafe (4).
Counterexample:
   sender_0xreceiver_0 » |- 0 ! o -| » sender_1xreceiver_0 [owner=sender,C]
   sender_1xreceiver_0 » |- 0 ! c -| » sender_0xreceiver_0 [owner=sender,C]
   sender_0xreceiver_0 » |- 0 ? o -| » sender_0xreceiver_1
      [owner=receiver,U]
   sender_0xreceiver_1 » |- 1 ! d -| » sender_0xreceiver_0
      [owner=receiver,C]
Result validation:
   Validation of forward  feasibility of counterexample: passed.
   Validation of backward feasibility of counterexample: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>
```

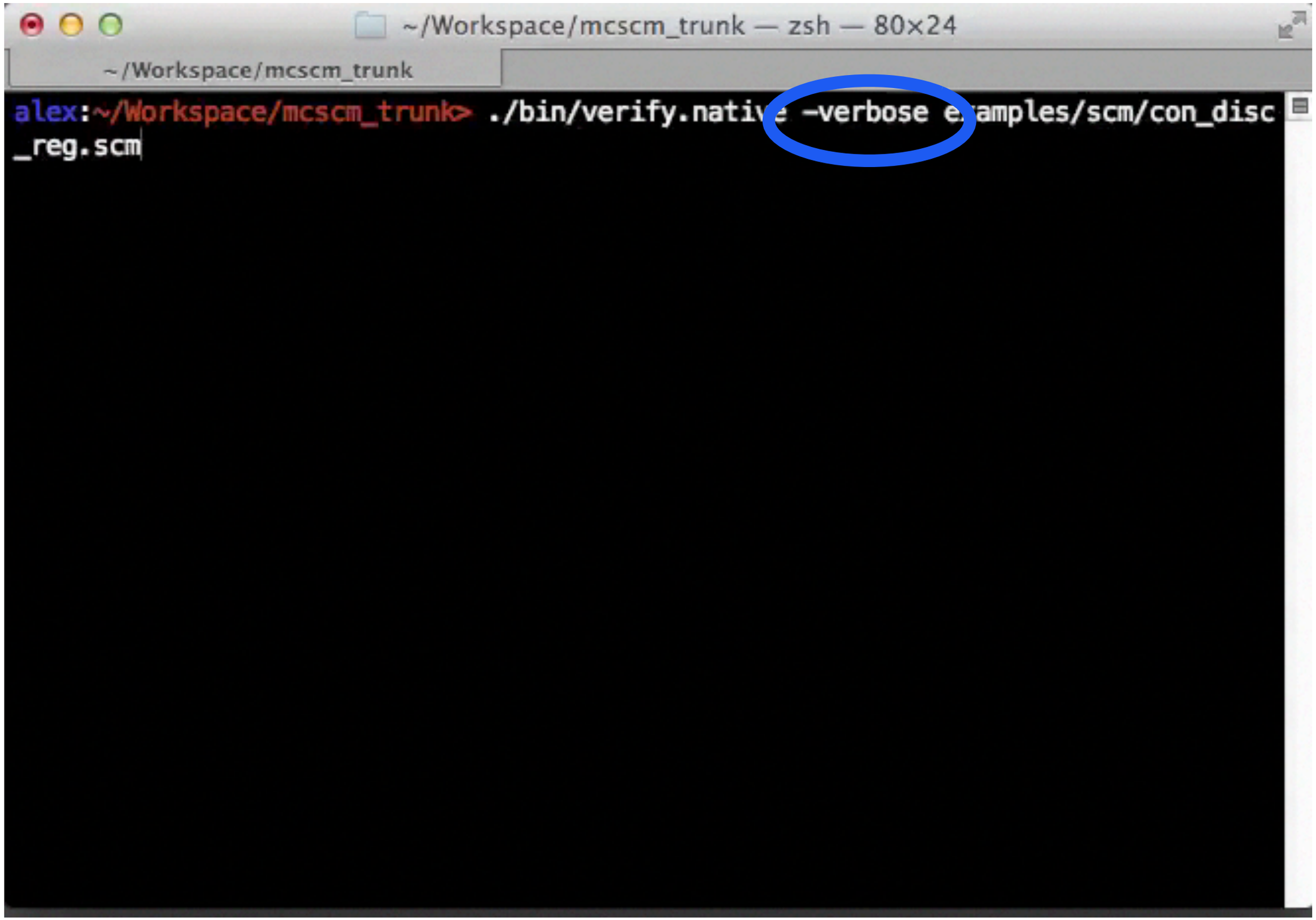
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native examples/scm/con_disc_reg.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states
Loops Nodes Edges Cex len k
CEGAR loop: 6 (12 / 49 / 4) 1)
CEGAR: 6 loops, 12 nodes, 49 edges
Result: Model is unsafe (4).
Counterexample:
sender_0xreceiver_0 » |- 0 ! o -| » sender_1xreceiver_0 [owner=sender,C]
sender_1xreceiver_0 » |- 0 ! c -| » sender_0xreceiver_0 [owner=sender,C]
sender_0xreceiver_0 » |- 0 ? o -| » sender_0xreceiver_1
[owner=receiver,U]
sender_0xreceiver_1 » |- 1 ! d -| » sender_0xreceiver_0
[owner=receiver,C]
Result validation:
Validation of forward feasibility of counterexample: passed.
Validation of backward feasibility of counterexample: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>

Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native examples/scm/con_disc_reg.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states
Loops Nodes Edges Cex len k
CEGAR loop: 6 (12 / 49 / 4) 1)
CEGAR: 6 loops, 12 nodes, 49 edges
Result: Model is unsafe (4).
Counterexample:
sender_0xreceiver_0 » |- 0 ! o -| » sender_1xreceiver_0 [owner=sender,C]
sender_1xreceiver_0 » |- 0 ! c -| » sender_0xreceiver_0 [owner=sender,C]
sender_0xreceiver_0 » |- 0 ? o -| » sender_0xreceiver_1
[owner=receiver,U]
sender_0xreceiver_1 » |- 1 ! d -| » sender_0xreceiver_0
[owner=receiver,C]
Validation of forward feasibility of counterexample: passed.
Validation of backward feasibility of counterexample: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>

Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native -verbose examples/scm/con_disc
_reg.scm

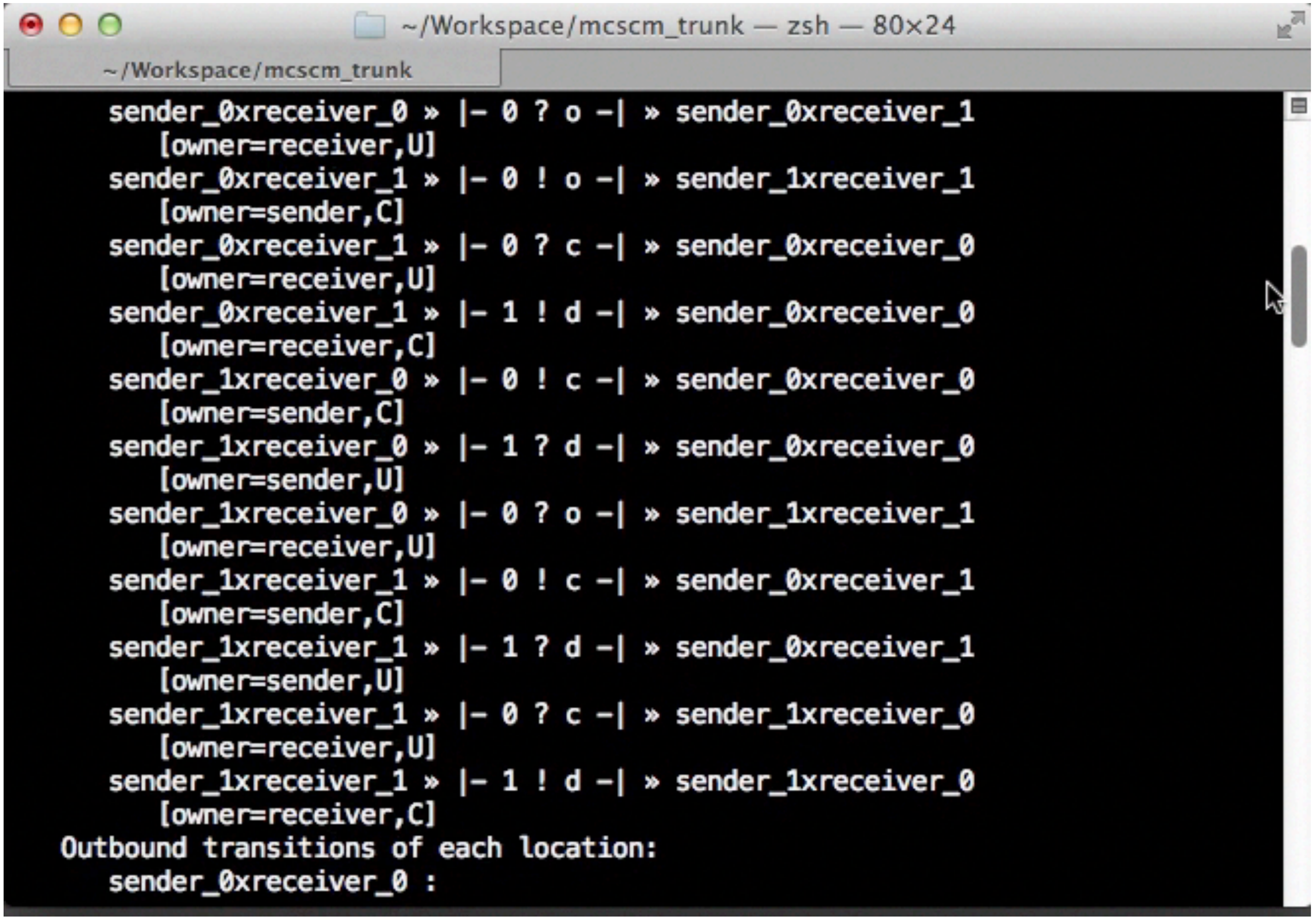
Demo

~/Workspace/mcscm_trunk — zsh — 80×24

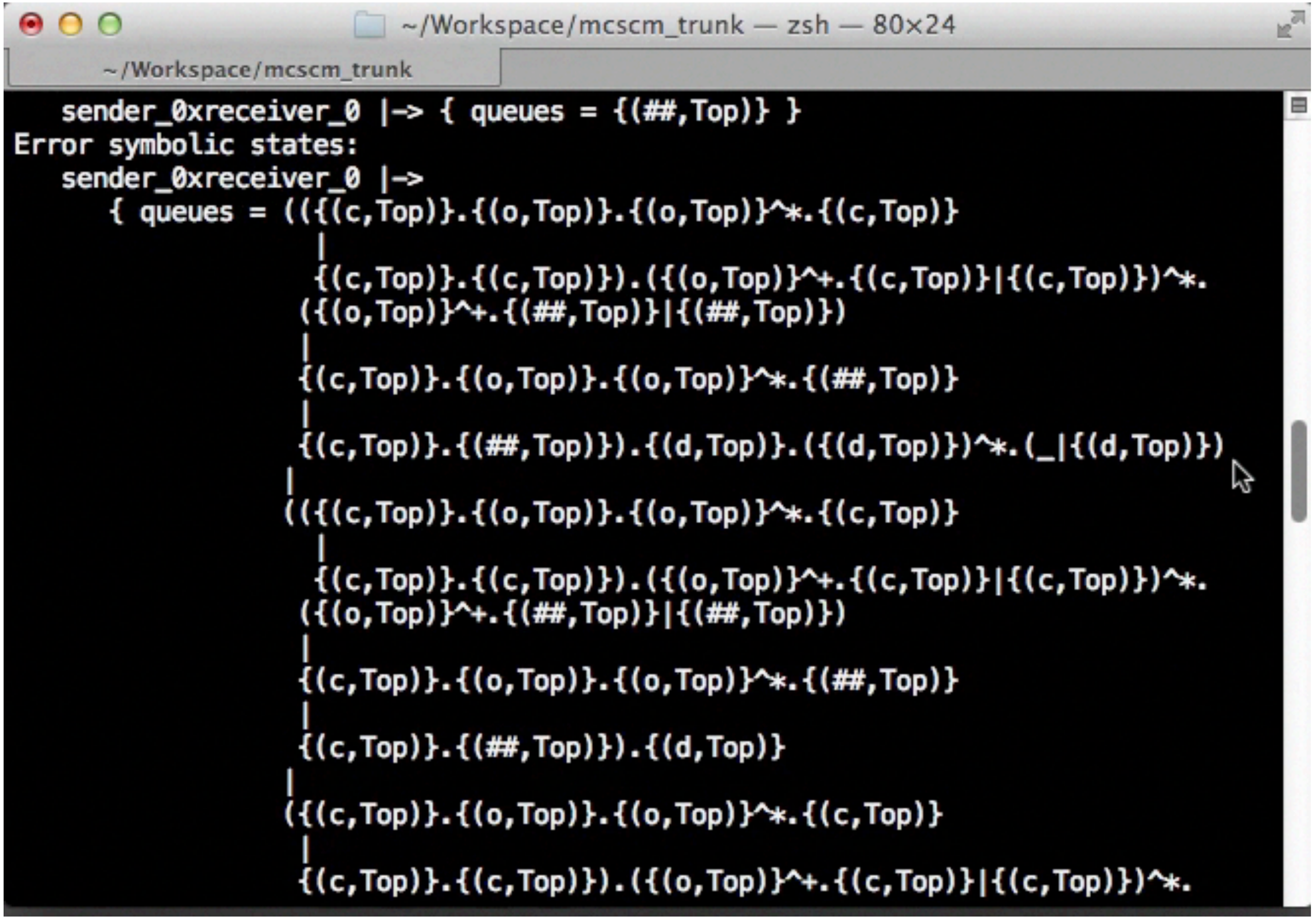
~/Workspace/mcscm_trunk

```
alex:~/Workspace/mcscm_trunk> ./bin/verify.native -verbose examples/scm/con_disc
_reg.scm
McScM 1.3+dev0 - Model Checking Tool
Parsing input SCM description... done.
Computing global SCM by cartesian product of local systems... done.
Checking consistency of global SCM... done.
Building SCM wrapper module (with control flow automaton)... done.
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states
------------------------------------------------------------------------------
Model: Global SCM with name: connection_disconnection
Channels:
   0, 1
Variables:

Parameters:
   o, c, d
Automaton:
   All 4 locations:
      sender_0xreceiver_0
      sender_0xreceiver_1
      sender_1xreceiver_0
      sender_1xreceiver_1
   Initial locations: sender_0xreceiver_0
   Error locations: sender_0xreceiver_0, sender_1xreceiver_0
```

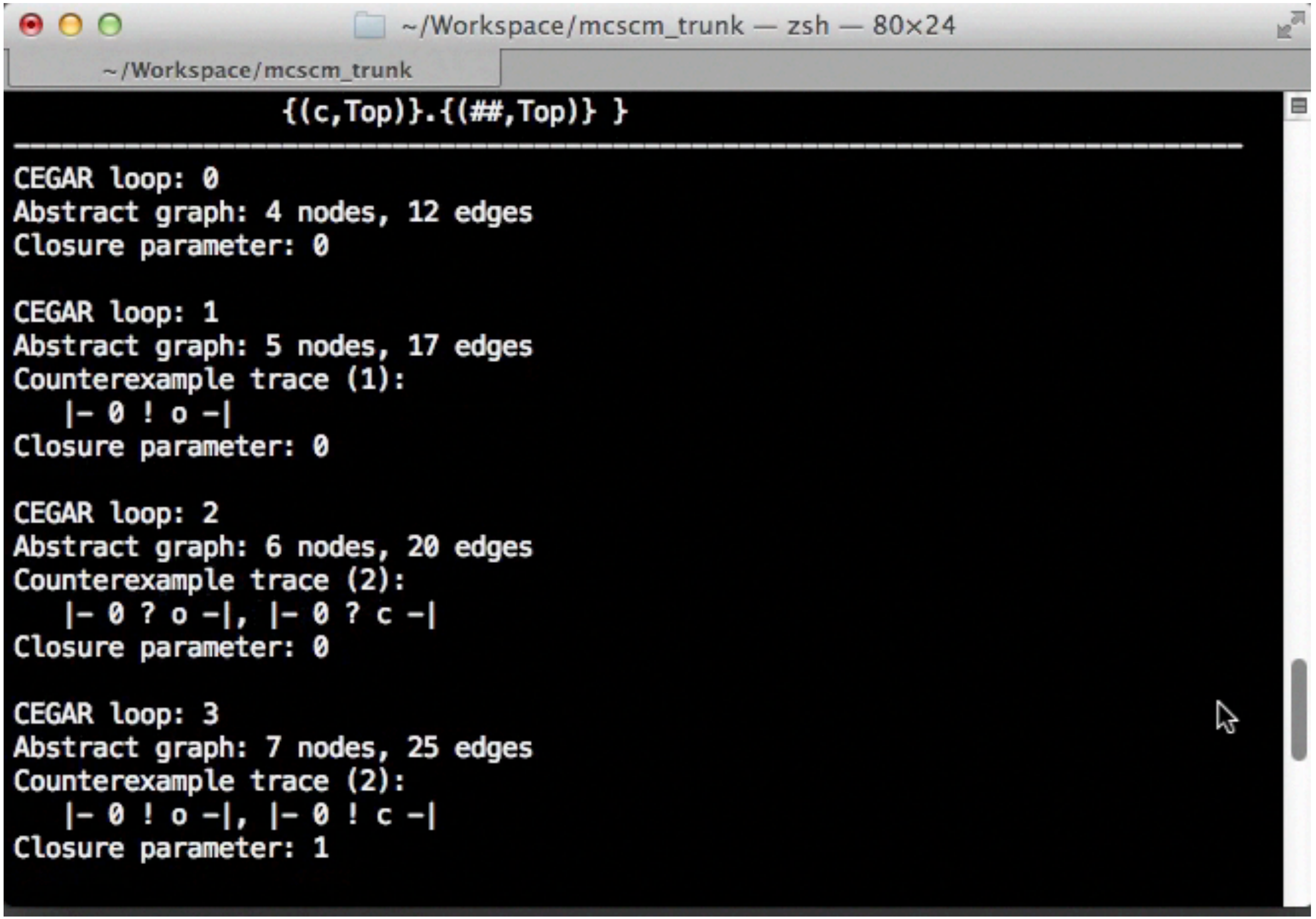
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
sender_0xreceiver_0 » |- 0 ? o -| » sender_0xreceiver_1
[owner=receiver,U]
sender_0xreceiver_1 » |- 0 ! o -| » sender_1xreceiver_1
[owner=sender,C]
sender_0xreceiver_1 » |- 0 ? c -| » sender_0xreceiver_0
[owner=receiver,U]
sender_0xreceiver_1 » |- 1 ! d -| » sender_0xreceiver_0
[owner=receiver,C]
sender_1xreceiver_0 » |- 0 ! c -| » sender_0xreceiver_0
[owner=sender,C]
sender_1xreceiver_0 » |- 1 ? d -| » sender_0xreceiver_0
[owner=sender,U]
sender_1xreceiver_0 » |- 0 ? o -| » sender_1xreceiver_1
[owner=receiver,U]
sender_1xreceiver_1 » |- 0 ! c -| » sender_0xreceiver_1
[owner=sender,C]
sender_1xreceiver_1 » |- 1 ? d -| » sender_0xreceiver_1
[owner=sender,U]
sender_1xreceiver_1 » |- 0 ? c -| » sender_1xreceiver_0
[owner=receiver,U]
sender_1xreceiver_1 » |- 1 ! d -| » sender_1xreceiver_0
[owner=receiver,C]
Outbound transitions of each location:
sender_0xreceiver_0 :

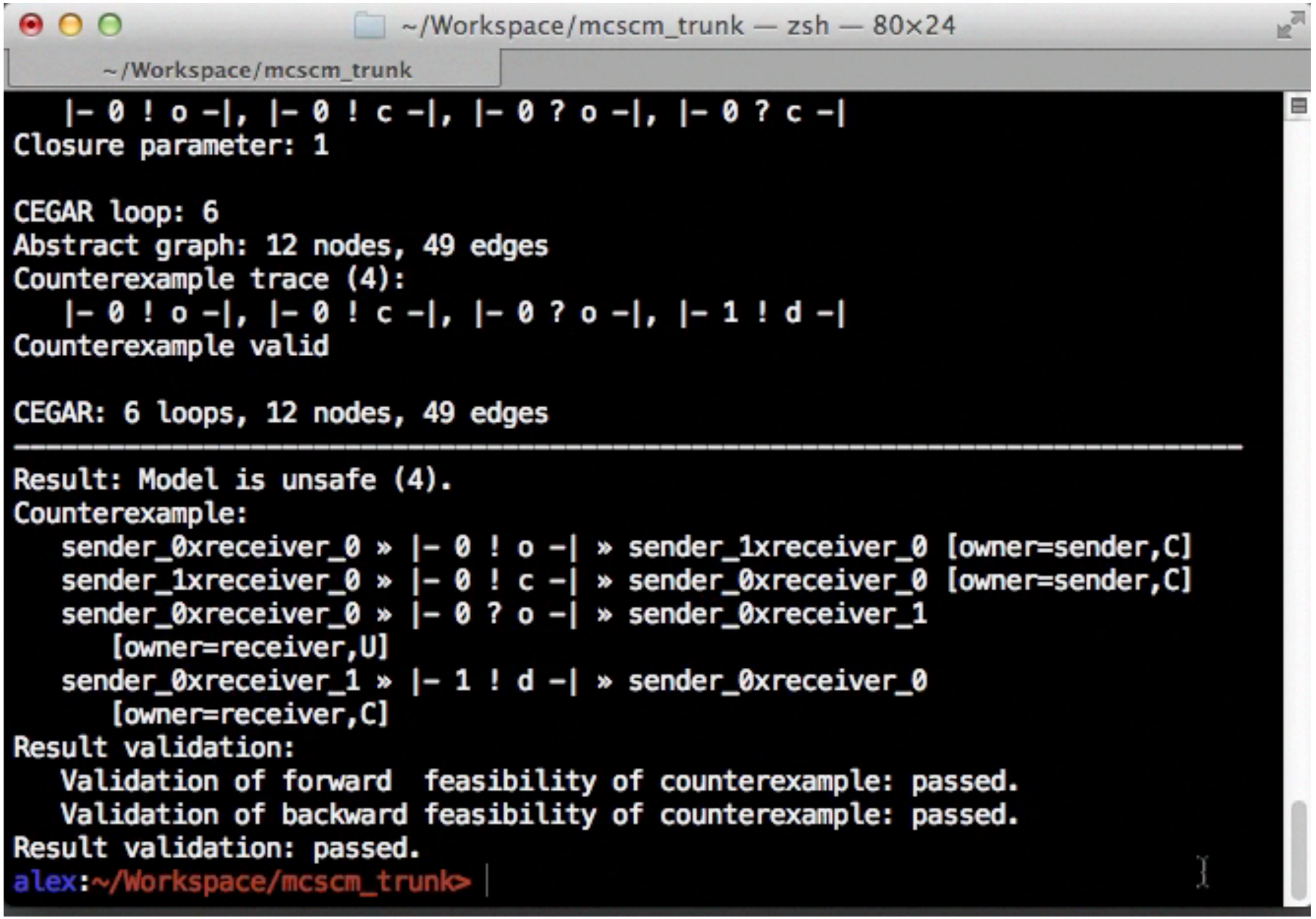
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
sender_0xreceiver_0 |-> { queues = {(##,Top)} }
Error symbolic states:
sender_0xreceiver_0 |->
{ queues = (({(c,Top)}.{(o,Top)}.{(o,Top)}^*.{(c,Top)}
|
{(c,Top)}.{(c,Top)}).({(o,Top)}^+.{(c,Top)}|{(c,Top)})^*.
({(o,Top)}^+.{(##,Top)}|{(##,Top)})
|
{(c,Top)}.{(o,Top)}.{(o,Top)}^*.{(##,Top)}
|
{(c,Top)}.{(##,Top)}).{(d,Top)}.({(d,Top)})^*.(_|{(d,Top)})
|
(({(c,Top)}.{(o,Top)}.{(o,Top)}^*.{(c,Top)}
|
{(c,Top)}.{(c,Top)}).({(o,Top)}^+.{(c,Top)}|{(c,Top)})^*.
({(o,Top)}^+.{(##,Top)}|{(##,Top)})
|
{(c,Top)}.{(o,Top)}.{(o,Top)}^*.{(##,Top)}
|
{(c,Top)}.{(##,Top)}).{(d,Top)}
|
({(c,Top)}.{(o,Top)}.{(o,Top)}^*.{(c,Top)}
|
{(c,Top)}.{(c,Top)}).({(o,Top)}^+.{(c,Top)}|{(c,Top)})^*.

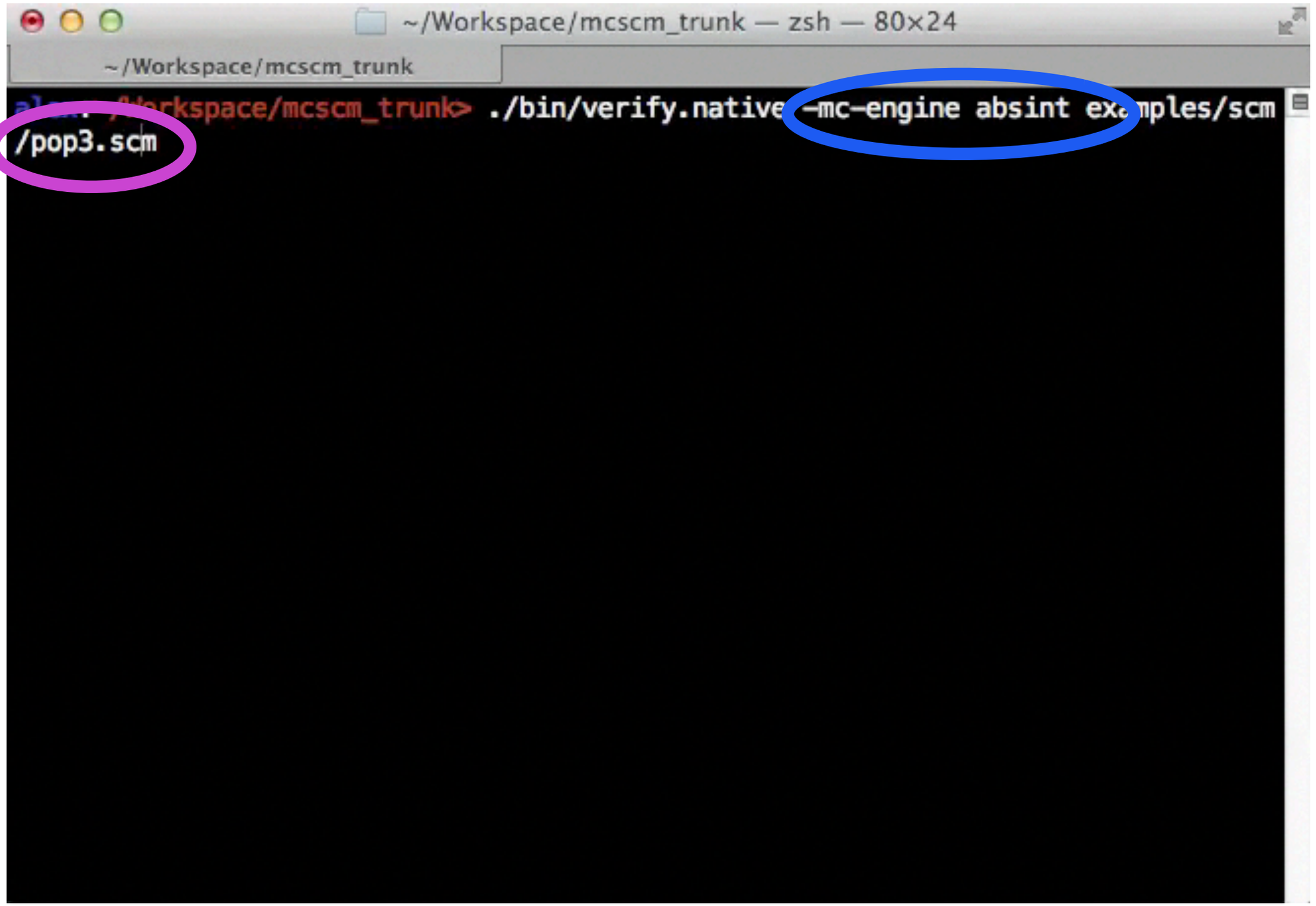
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
{(c,Top)}.{(##,Top)} }
CEGAR loop: 0
Abstract graph: 4 nodes, 12 edges
Closure parameter: 0
CEGAR loop: 1
Abstract graph: 5 nodes, 17 edges
Counterexample trace (1):
|- 0 ! o -|
Closure parameter: 0
CEGAR loop: 2
Abstract graph: 6 nodes, 20 edges
Counterexample trace (2):
|- 0 ? o -|, |- 0 ? c -|
Closure parameter: 0
CEGAR loop: 3
Abstract graph: 7 nodes, 25 edges
Counterexample trace (2):
|- 0 ! o -|, |- 0 ! c -|
Closure parameter: 1

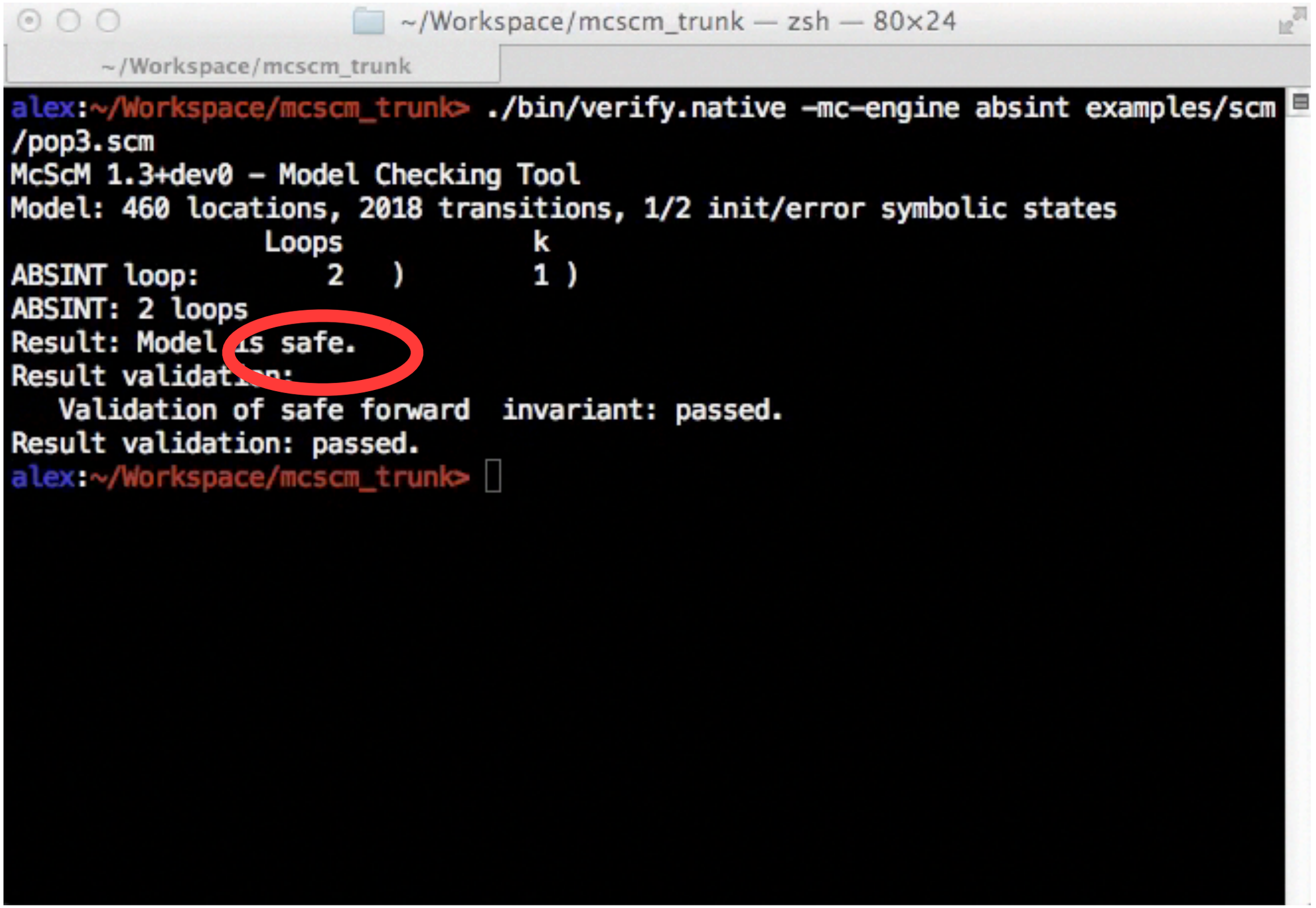
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
|- 0 ! o -|, |- 0 ! c -|, |- 0 ? o -|, |- 0 ? c -|
Closure parameter: 1
CEGAR loop: 6
Abstract graph: 12 nodes, 49 edges
Counterexample trace (4):
|- 0 ! o -|, |- 0 ! c -|, |- 0 ? o -|, |- 1 ! d -|
Counterexample valid
CEGAR: 6 loops, 12 nodes, 49 edges
Result: Model is unsafe (4).
Counterexample:
sender_0xreceiver_0 » |- 0 ! o -| » sender_1xreceiver_0 [owner=sender,C]
sender_1xreceiver_0 » |- 0 ! c -| » sender_0xreceiver_0 [owner=sender,C]
sender_0xreceiver_0 » |- 0 ? o -| » sender_0xreceiver_1
[owner=receiver,U]
sender_0xreceiver_1 » |- 1 ! d -| » sender_0xreceiver_0
[owner=receiver,C]
Result validation:
Validation of forward feasibility of counterexample: passed.
Validation of backward feasibility of counterexample: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>

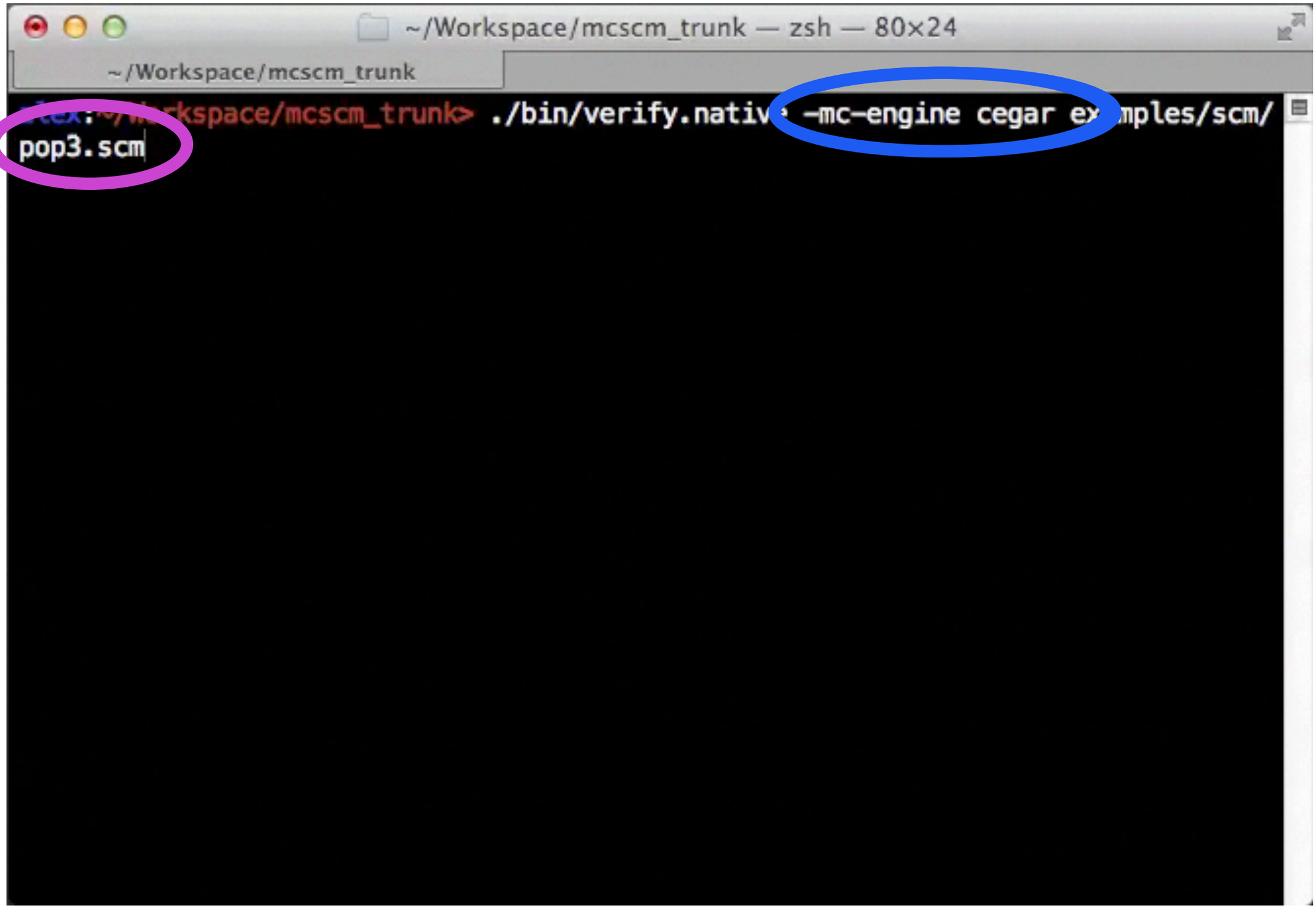
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
./bin/verify.native -mc-engine absint examples/scm
/pop3.scm

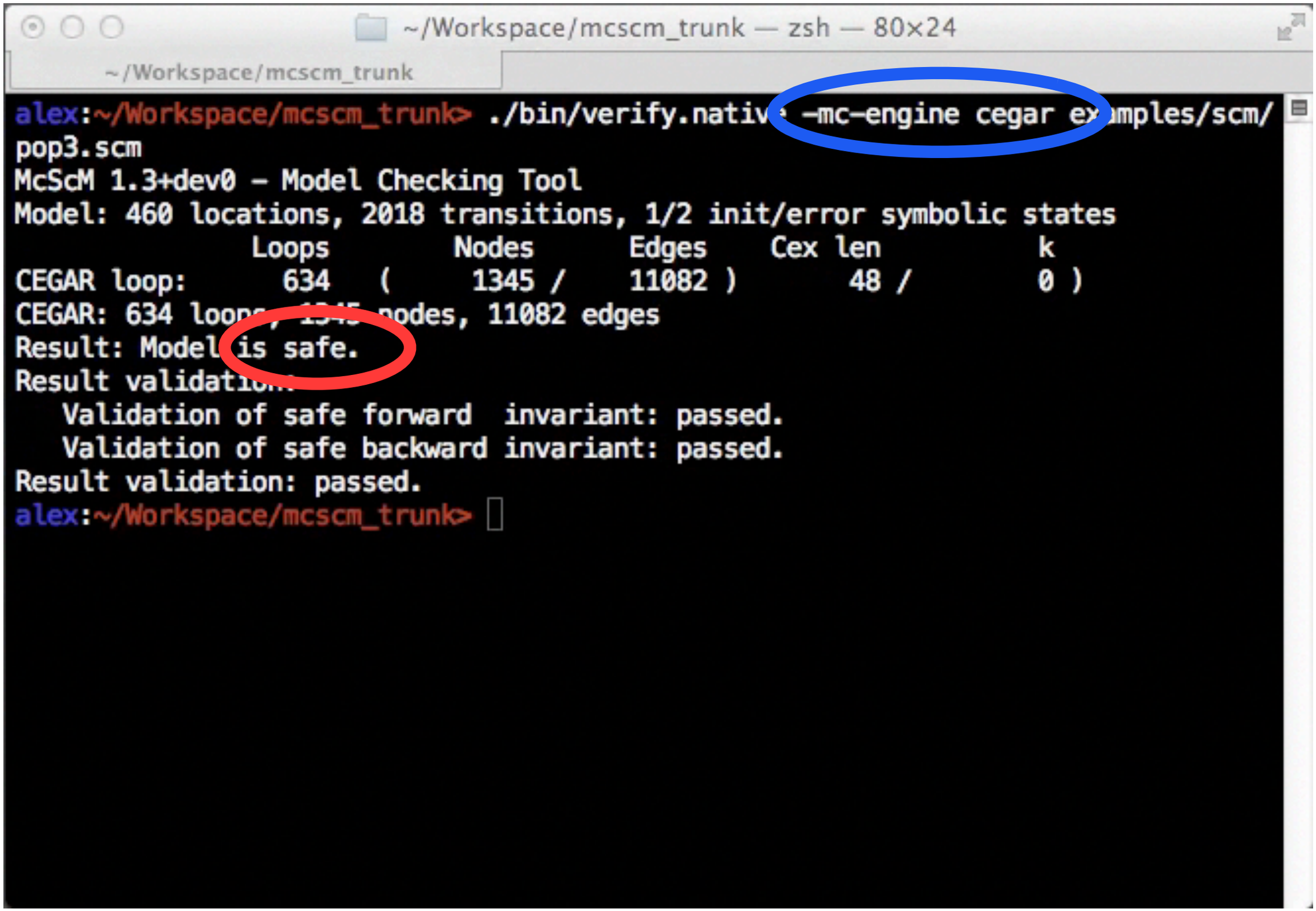
Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native -mc-engine absint examples/scm/pop3.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 460 locations, 2018 transitions, 1/2 init/error symbolic states
Loops k
ABSINT loop: 2) 1)
ABSINT: 2 loops
Result: Model is safe.
Result validation:
Validation of safe forward invariant: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>

Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
./bin/verify.nativ -mc-engine cegar ex mples/scm/
pop3.scm

Demo


~/Workspace/mcscm_trunk — zsh — 80×24
~/Workspace/mcscm_trunk
alex:~/Workspace/mcscm_trunk> ./bin/verify.native -mc-engine cegar examples/scm/pop3.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 460 locations, 2018 transitions, 1/2 init/error symbolic states
Loops Nodes Edges Cex len k
CEGAR loop: 634 (1345 / 11082) 48 / 0)
CEGAR: 634 loops, 1345 nodes, 11082 edges
Result: Model is safe.
Result validation:
Validation of safe forward invariant: passed.
Validation of safe backward invariant: passed.
Result validation: passed.
alex:~/Workspace/mcscm_trunk>

Extended Demo Offline

Chrome File Edit View History Bookmarks Window Help — Mar 27 10:55

McScM – Wiki – ALTARICA

altarica.labri.fr/forge/projects/mcscm/wiki/

Home Projects Help — Sign in Register

Analysis of Concurrent Systems » McScM

Search:

Overview Download Activity Roadmap Issues Documents Wiki Files

about the tool

History

McScM (Model Checker for Systems of Communicating fifo Machines) is a small framework for implementing model checking algorithms in the context of infinite state systems of the form "finite control plus infinite data". The main constituent is the `verify` tool for the safety verification of systems of communicating fifo machines (CFM). The goal of `verify` is the model checking of network protocols (given as CFM) with respect to a given set of undesired behaviours or "bad" states. The main difference to other tools for communicating automata is the modeling of asynchronous, reliable fifo channels which is indispensable for checking protocols that presuppose a reliable connection between the peers (for example as supplied by TCP or MPI). Further, the `verify` tool allows to choose among distinct verification algorithms which are itself highly parametrizable. This leads to a generic approach that can easily be adapted to a variety of concrete safety verification problems. In addition, McScM includes a tool for supervisory `control` that helps to fix unsafe protocols.

how to get it

Please refer to **Download** for instructions.

documentation and manual pages

We provide an onl

- verify - s
- control -
- scm - CF

We provide a desc

- Svar - sa
- Util - util

references

McScM is the outc

More about the th

- Alexander H Proceedings

or even more deta

- Alexander H 1459-09, M

or take a look at t

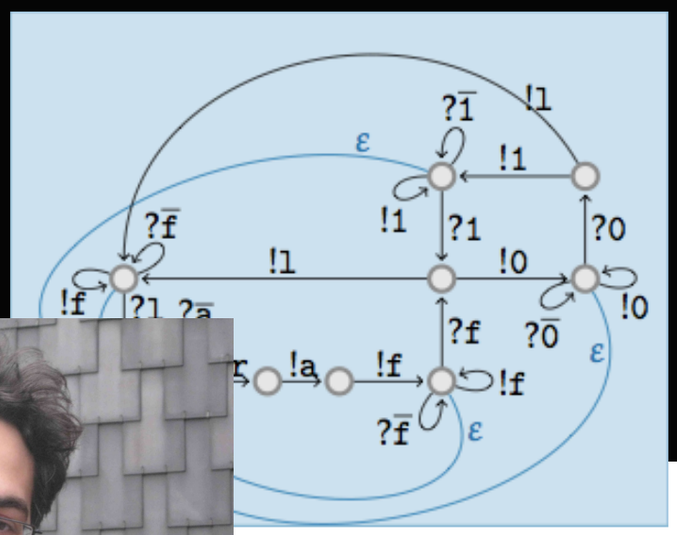
The theoretical un

- Gabriel Kaly Conference

<bin/verify.native -statistics -mc-engine cegar -tc-engine apinv-bwd -extrap

<bin/verify.nati...ace/mcscm_trunk

```
alex:~/Workspace/mcscm_trunk> bin/verify.native -statistics -mc-engine ce
-engine apinv-bwd -extrapolation bisim-bwd examples/scm/brp_like.scm
McScM 1.3+dev0 - Model Checking Tool
Model: 100 locations, 550 transitions, 1/37 init/error symbolic states
                Loops       Nodes      Edges    Cex len          k
CEGAR loop:       401   (     740 /     6846 /       17 /         1 )
```



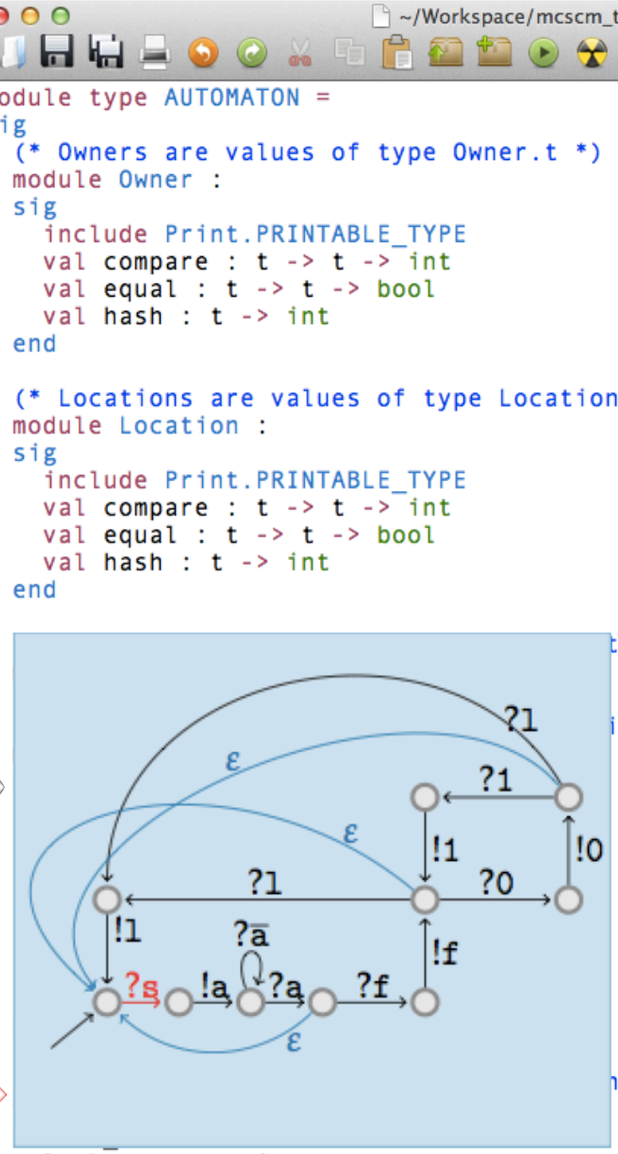
data

ack

sync

~/Workspace/mcscm_trunk/src/svar/Model.mli

```
module type AUTOMATON =
sig
  (* Owners are values of type Owner.t *)
  module Owner :
  sig
    include Print.PRINTABLE_TYPE
    val compare : t -> t -> int
    val equal : t -> t -> bool
    val hash : t -> int
  end

  (* Locations are values of type Location.t *)
  module Location :
  sig
    include Print.PRINTABLE_TYPE
    val compare : t -> t -> int
    val equal : t -> t -> bool
    val hash : t -> int
  end

                                                    t *)

                                                    ition.t *)

                                                    ns *)

  val nb_owners : int

  (* Iterators over all locations and over all transitio
  val iter_locations :   (Location.t   -> unit) -> unit
[1] src/svar/Model.mli
```



McScM Highlights

- general framework for
  finite-control infinite-data transition systems
- modular (and well-documented) API
  - apply implemented algos directly to other FCID
  - easily prototype new algos on top
- free license and open development
- includes verification  & controler-synthesis tool

http://altarica.labri.fr/forge/projects/mcscm

Ongoing/Future

- input language (extended) PROMELA
- exchange of infinite data via channels (distributed calculation, needs some theoretical work first)
- new algos (learning-based, SAT, etc.)
- new backends (“real” lossy channels)
- basis for prototypes for synthesis algos...

Related Tools/Frameworks

- SPIN - finite(!) communicating processes
- TReX - parametric + timed + data + lossy channels
- LaSH - framework for QDD, NDD, RVA
- LTSA - synchronizing finite automata, plugins
- TaPAS - VAS, Presburger Automata, reachability
- …