The MSM Framework

A. Heußner / T. Le Gall / G. Sutre

TACAS 2012
Tool Presentation
Finite-Control Infinite-Data Transition System API

verification algorithm
controller synthesis algorithm

communicating machines

Algorithms
Backends
developers
Sutre / Le Gall
Heußner

stable release
1.2
active

written in
OCaml

OS
cross platform

distribution
source & binary

license
BSD

website
forge, wiki,...
Basic Structure

Algorithms
- verification algorithm
- controller synthesis algorithm
- ...

Finite-Control Infinite-Data Transition System API

Backends
- communicating machines
- ...

Algorithms
- verification algorithm
- controller synthesis algorithm
- ...

Finite-Control Infinite-Data Transition System API
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:
a simple client-server example protocol:
Communicating Machines

- a simple client-server example protocol:

Client

Server

!o !c ?d

?o ?c !d

c

d
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

- log
- extrapolation
- scm wrapper
- absint
- cegar
- armc
- lart
- pathinvariant generator
- trace checker

External libraries: scm, latticeautomata
Modularity by McScM

- log
- extrapolation
- scm wrapper
- external libraries: scm, latticeautomata
- absint
- armc
- cegar
- lart
- pathinvariant generator
- trace checker
Modularity by McScM

external libraries:

```
scm, latticeautomata
```

```
absint
cegade
armc
lart
```

```
log
verify
validator
pathinvariant
generator
trace
checker
```
Modularity by McScM

external libraries: scm, latticeautomata

verify
log

in
out

validator
pathinvariant generator
trace checker

scm wrapper
extrapolation

absint
armc
cegar
lart
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

```plaintext
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 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
Demo

McSCM 1.3+dev0 - Model Checking Tool
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states

Loops Nodes Edges Cex len k
6 12 / 49 / 4 1

CEGAR loop: 6 loops, 12 nodes, 49 edges
Result: Model is unsafe (4).

Counterexample:
sender_0xreceiver_0 \xrightarrow{\text{-0}} \text{o} \xrightarrow{\text{-0}} \text{c} \xrightarrow{\text{-0}} \text{o} \xrightarrow{\text{owner=receiver,U}} \text{sender_0xreceiver_0}

Result validation:
Validation of forward feasibility of counterexample: passed.
Validation of backward feasibility of counterexample: passed.

Result validation: passed.

alex:~/Workspace/mcscm_trunk>
Model: 4 locations, 12 transitions, 1/2 init/error symbolic states

CEGAR loop: 6

Result: Model is unsafe (4).

Counterexample:

Validation of forward feasibility of counterexample: passed.
Validation of backward feasibility of counterexample: passed.

Result validation: passed.
Demo

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.
Demo

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

```
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=receiver,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 ? o - | > sender_1xreceiver_0
[owner=receiver,U]
sender_1xreceiver_1 | - 0 ! c - | > sender_0xreceiver_0
[owner=receiver,C]
```

Outbound transitions of each location:

```
sender_0xreceiver_0 :
```
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)\}\}.\{c, Top\}\}
  \{c, Top\}.\{o, Top\}.\{o, Top\}^*.\{ (#, Top)\}
  \{c, Top\}.\{ (#, Top)\}.\{d, Top\}.\{d, Top\}^*.\{o, Top\}^+.\{ (#, Top)\}\}
  \{c, Top\}.\{o, Top\}.\{o, Top\}^*.\{c, Top\}
  \{c, Top\}.\{c, Top\}\}.\{o, Top\}^+.\{c, Top\}\}.\{c, Top\}^*.\{o, 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\}^*.\{o, Top\}^+.\{ (#, Top)\}\}
  \{c, Top\}.\{c, Top\}\}.\{o, Top\}^+.\{c, Top\}\}.\{c, Top\}^*.\{o, Top\}^+.\{ (#, Top)\}\}.
Demo

```
{((c,Top)} \cdot \{(#\#,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

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Closure parameter: 1

CEGAR loop: 6
Abstract graph: 12 nodes, 49 edges
Counterexample trace (4):
| 0 | 0 | 0 | 0 |

Counterexample valid

CEGAR: 6 loops, 12 nodes, 49 edges

Result: Model is unsafe (4).

Counterexample:
- sender_0xreceiver_0 \rightarrow | 0 | 0 | 0 | 0 |
- sender_1xreceiver_0 \rightarrow | 0 | 0 | 0 | 0 |
- sender_0xreceiver_0 \rightarrow | 0 | 0 | 0 | 0 |
- sender_0xreceiver_0 \rightarrow | 0 | 0 | 0 | 0 |
- sender_0xreceiver_1 \rightarrow | 0 | 0 | 0 | 0 |
- sender_0xreceiver_1 \rightarrow | 0 | 0 | 0 | 0 |

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> ./bin/verify.native -mc-engine absint examples/scm
```

```
/pop3.scm
```
Demo

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.
Demo

McScM 1.3+dev0 - Model Checking Tool
Model: 460 locations, 2018 transitions, 1/2 init/error symbolic states

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
Highlights

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