Satisfiability Modulo Theories Lecture 8 - Introduction to SMT-based Model-Checking

(slides revision: Saturday 14^{th} March, 2015, 11:47)

Roberto Bruttomesso

Seminario di Logica Matematica
(Corso Prof. Silvio Ghilardi)

15 Dicembre 2011

Outline

1. Basics
   - Modeling
   - Checking
   - Implementing a Model-Checker

2. MCMT
   - Two simple protocols

Introduction

Model-Checking is a set of techniques to approach the verification of a system (e.g., a hardware circuit, a program, a protocol)

It was proposed by Clarke-Emerson and Sifakis-Quine as a way of automatically prove properties of a system

The authors received the Turing Award in 2007

The idea of model-checking was in contrast with the established "philosophy" at that time ($\sim$ 1980) which was suggesting semi-automatic human-driven approaches: MC is loved by industry because of this "push-button" characteristic

Model-Checking - Modeling

In MC we model the behavior of a system with the notion of **state**. A state is a configuration of the system at a particular time instant

The system can change state by means of a **transition**

We are interested in a **property** of the system

In MC we model the behavior of a system with the notion of **state**. A state is a configuration of the system at a particular time instant

The system can change state by means of a **transition**

We are interested in a **property** of the system

Example:

- System: a washing machine
- A state: “the door is open and the engine is off”
- A transition: “if the door is open then close the door”
- A property: “When the engine is on, the door is closed”

System to Model

Modeling - States

State variables can be used to describe a particular state

State variable	Values
door	open, closed
tray	empty, filled
engine	off, on

Modeling - States

State variables can be used to describe a particular state

State variable	Values
door	open, closed
tray	empty, filled
engine	off, on

E.g.:

door=open
engine=on
tray=empty

which stands for “the door is open, the engine is on, and the tray is empty”.

Modeling - States

State variables can be used to describe a particular state

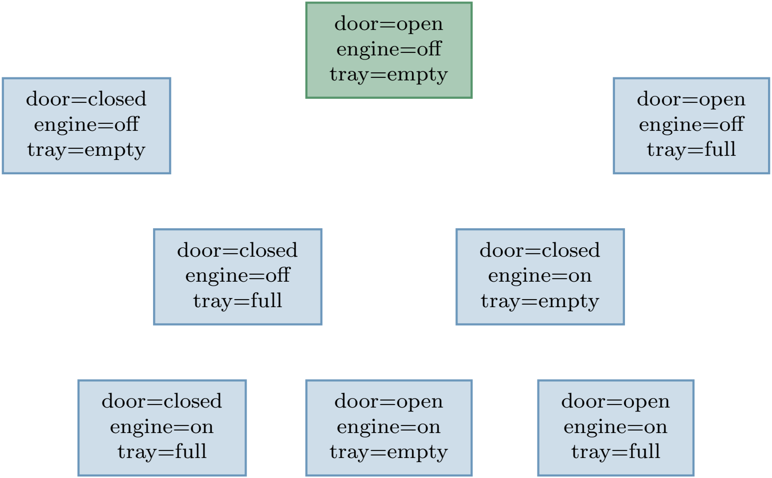
State variable	Values
door	open, closed
tray	empty, filled
engine	off, on

E.g.:

door=open
engine=on
tray=empty

which stands for "the door is open, the engine is on, and the tray is empty".How many different states can we describe with our state variables ?

Modeling - States



Some states are called **initial** (green). Initial states are the configurations of the system at time 0

Modeling - Transitions

Transitions describe the evolution of the system. They transform the "current" state into a "next" state

Modeling - Transitions

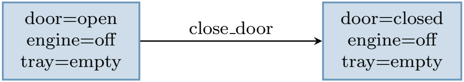
Transitions describe the evolution of the system. They transform the "current" state into a "next" state

Transition				Name
if	door=open	then	door'=closed	[close_door]
if	tray=empty	then	tray'=full	[fill_tray]
if	engine=off door=closed	then	engine'=on tray'=empty	[start_wash]
if	door=closed	then	door'=open engine'=off	[open_door]

var' indicates the value of var in the next state

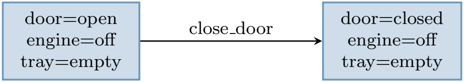
Modeling - Transitions

if door=open then door'=closed [close_door]

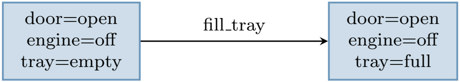


Modeling - Transitions

if door=open then door'=closed [close_door]

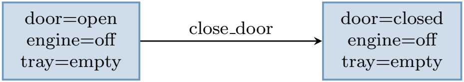


if tray=empty then tray'=full [fill_tray]

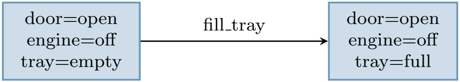


Modeling - Transitions

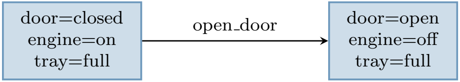
if door=open then door'=closed [close_door]



if tray=empty then tray'=full [fill_tray]

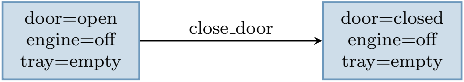


if door=closed then door'=open engine'=off [open_door]

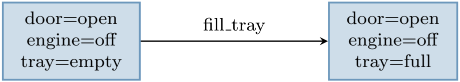


Modeling - Transitions

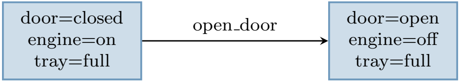
if door=open then door'=closed [close_door]



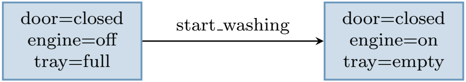
if tray=empty then tray'=full [fill_tray]



if door=closed then door'=open engine'=off [open_door]



if door=closed engine=off then tray'=empty engine'=on [start_washing]



Modeling - (Safety) Property

Last step, we need to model the property

"when the engine is on the door is closed"

It is a **safety** property: they are easy to define as they are properties of the states

Modeling - (Safety) Property

Last step, we need to model the property

"when the engine is on the door is closed"

It is a **safety** property: they are easy to define as they are properties of the states

We call **bad state** (or unsafe state) a state that does not satisfy the property

door=open engine=on tray=full

Checking (= Reachability)

To establish if a model satisfies a safety property amounts to check if some **bad state is reachable** from the set of initial states

This can be done automatically by **visiting** the set of states that are **reachable** from the initial state with the application of a transition

Checking (= Reachability)

To establish if a model satisfies a safety property amounts to check if some **bad state is reachable** from the set of initial states

This can be done automatically by **visiting** the set of states that are **reachable** from the initial state with the application of a transition

Let $S^{(0)}$ be the set of initial states. Algorithmically, it amounts to implement the following loop (iteration i)

Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**

$T(S^{(i)})$ = states that can be reached from $S^{(i)}$ with a transition

Checking - Forward Reachability - Property Verified

Forward-Reachability	
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**

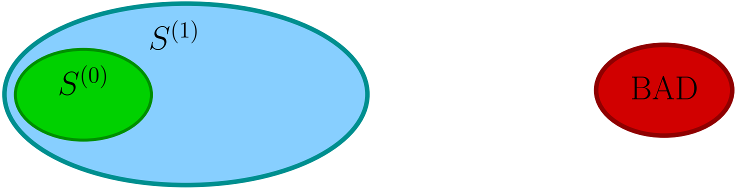
Checking - Forward Reachability - Property Verified

Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**

Checking - Forward Reachability - Property Verified

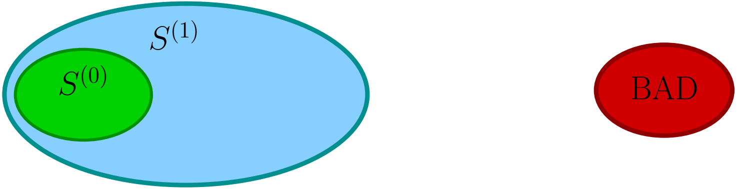
Forward-Reachability	
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

Forward-Reachability

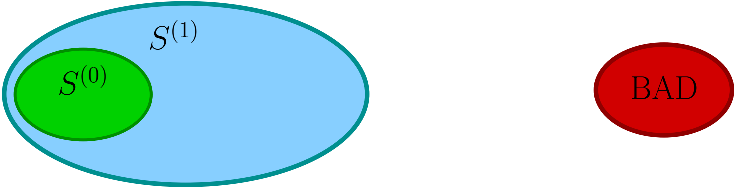
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

Forward-Reachability

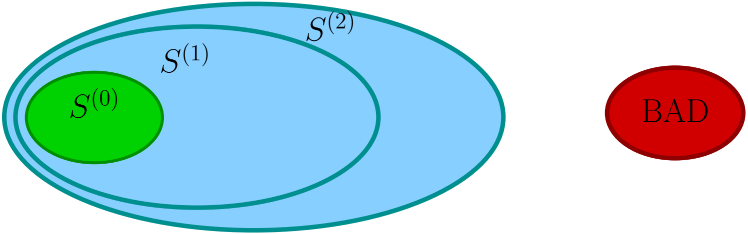
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

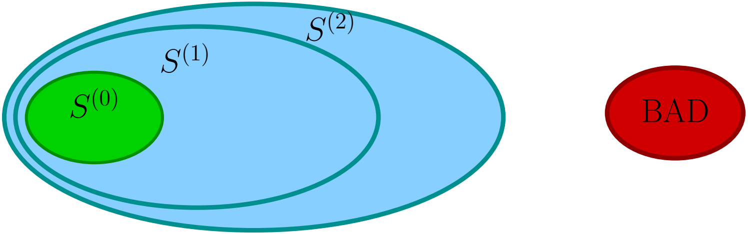
Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



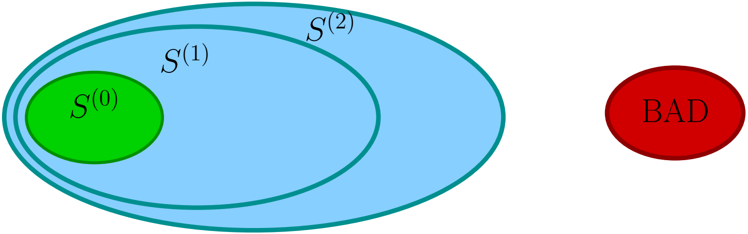
Checking - Forward Reachability - Property Verified

Forward-Reachability	
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Forward-Reachability

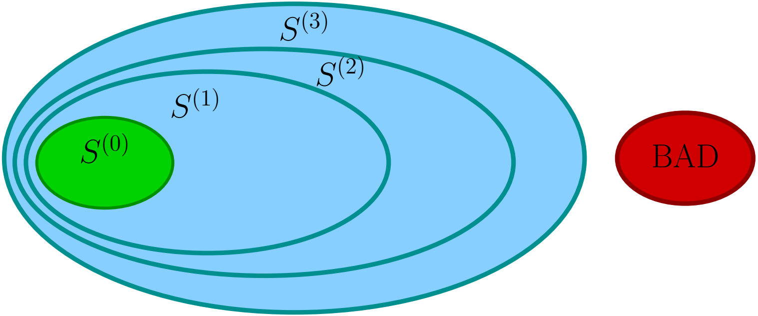
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

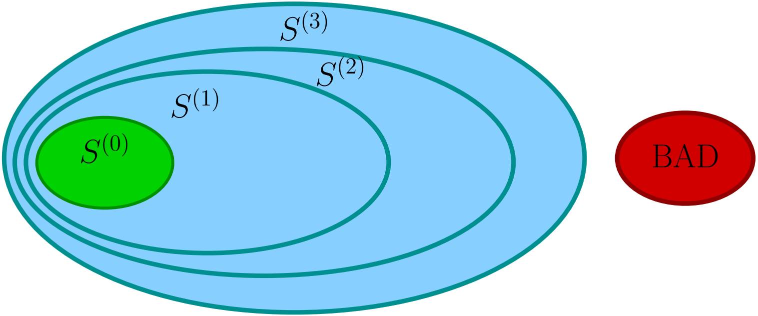
Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



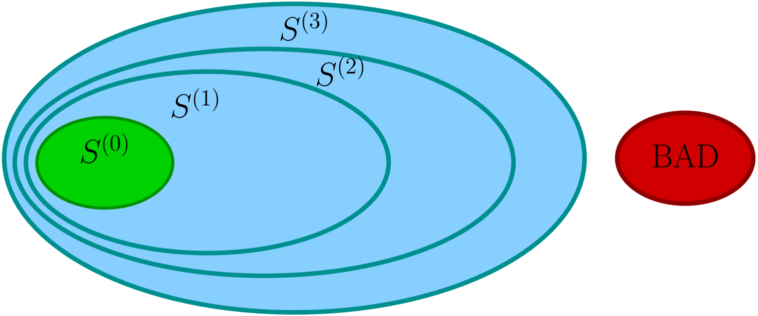
Checking - Forward Reachability - Property Verified

Forward-Reachability	
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Forward-Reachability

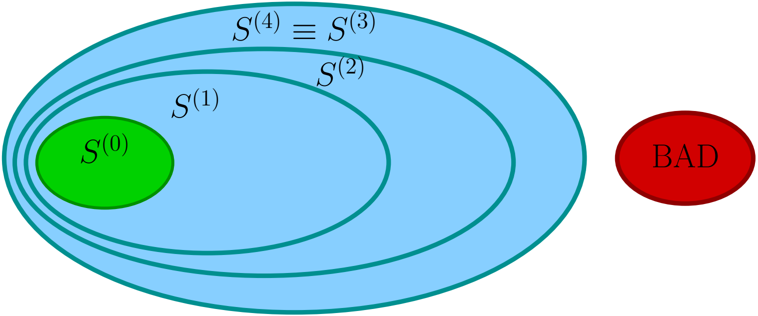
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

Forward-Reachability

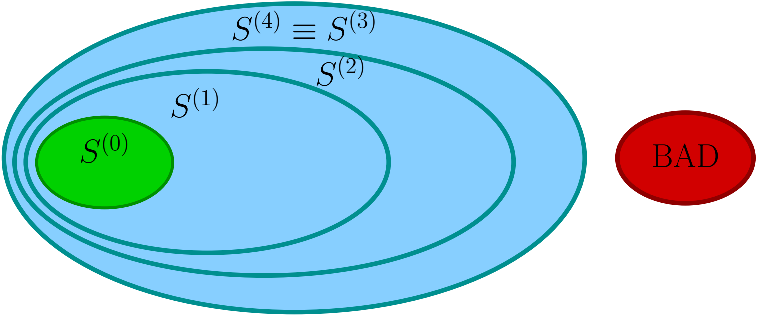
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Forward Reachability - Property Verified

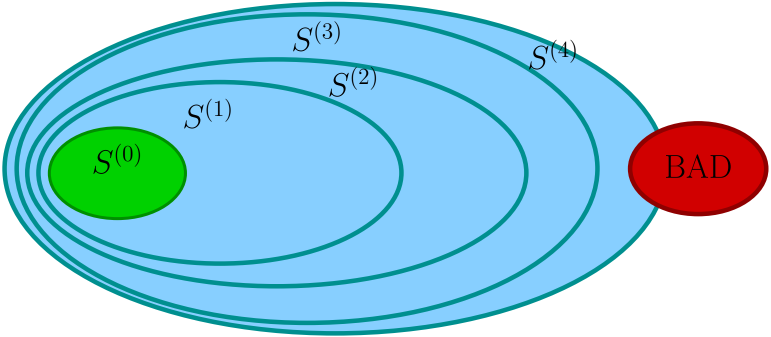
Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



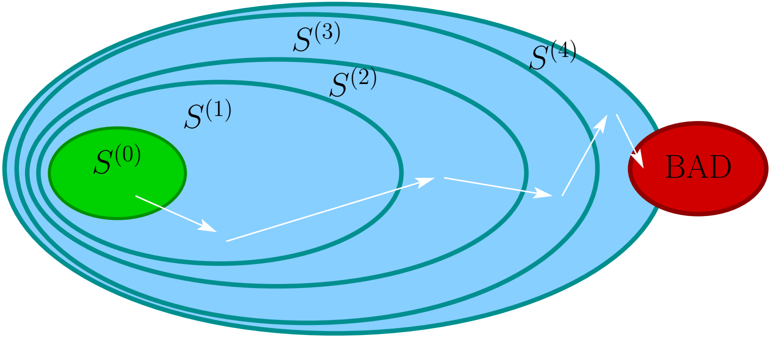
Checking - Forward Reachability - Property Not Verified

	Forward-Reachability
Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



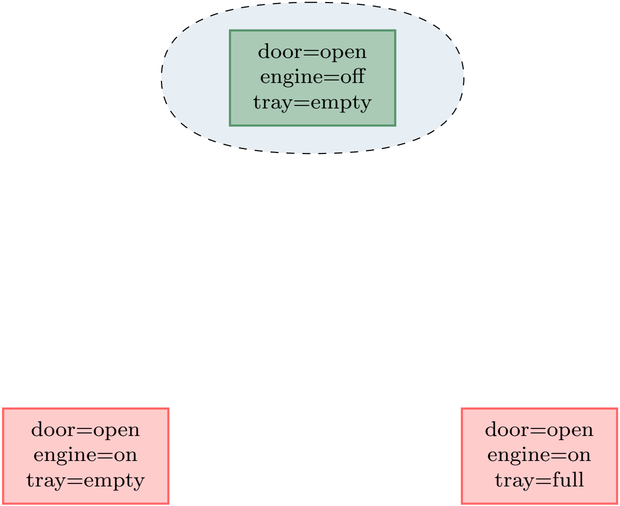
Forward-Reachability

Safety Check	If $S^{(i)}$ contains a bad state, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



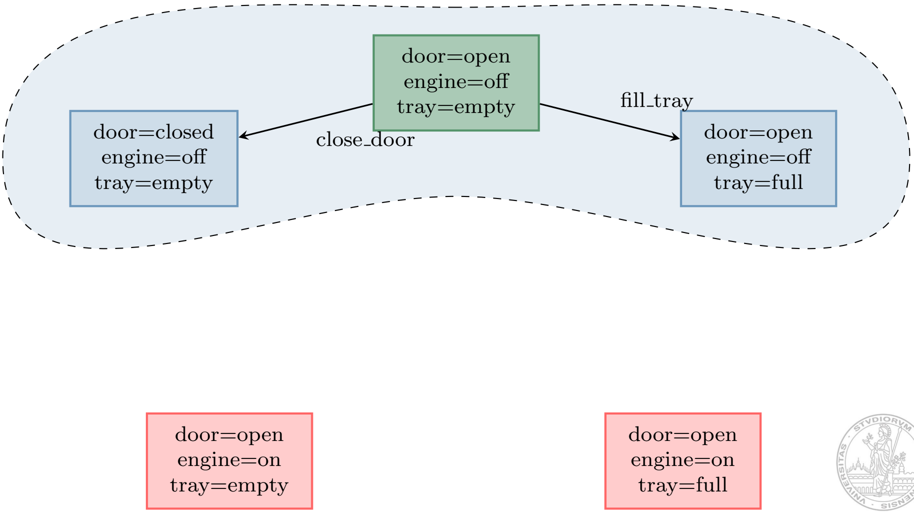
Back to the washing machine

Iteration: 0



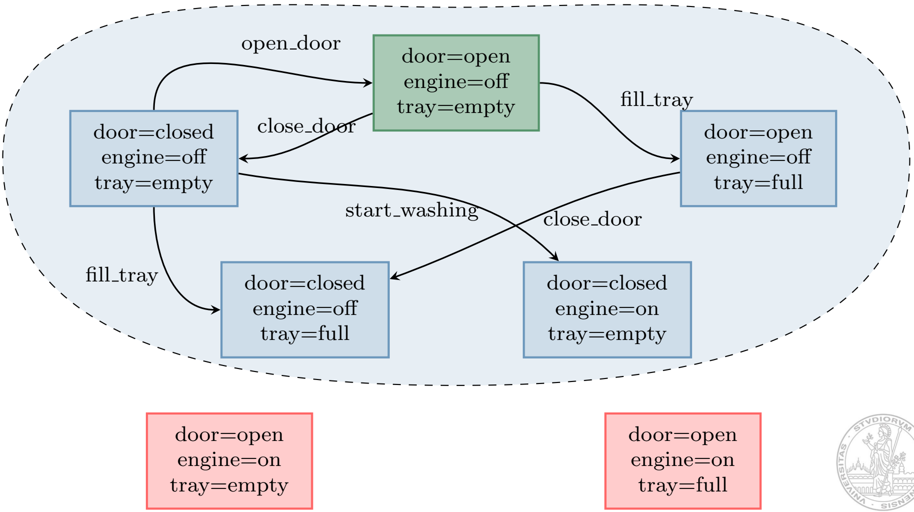
Back to the washing machine

Iteration: 1



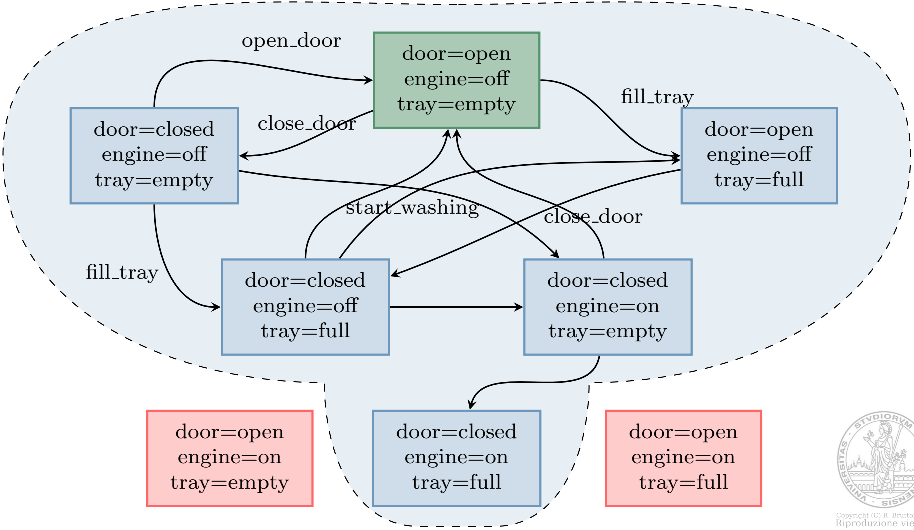
Back to the washing machine

Iteration: 2

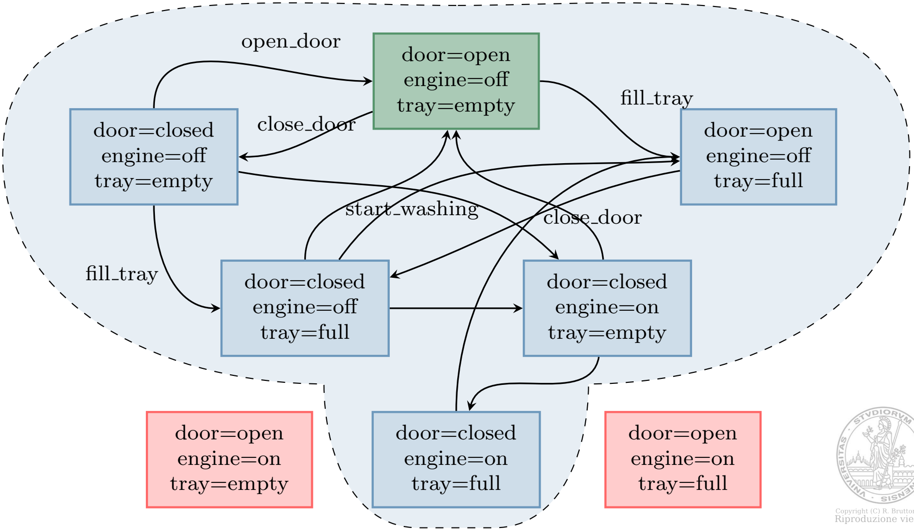


Back to the washing machine

Iteration: 3



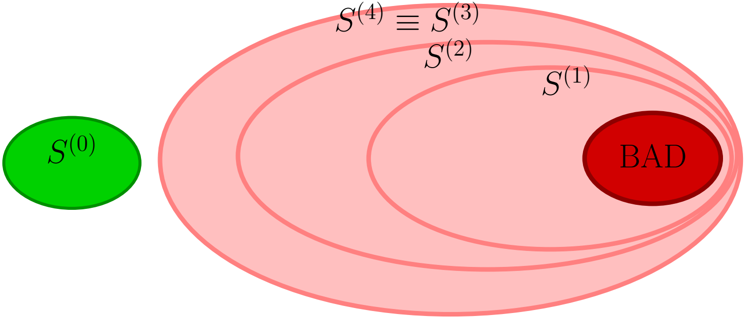
Iteration: 4 - Fix Point Reached - System is SAFE



Checking - Backward Reachability

Backward-Reachability ($S^{(0)} \equiv$ "bad states")

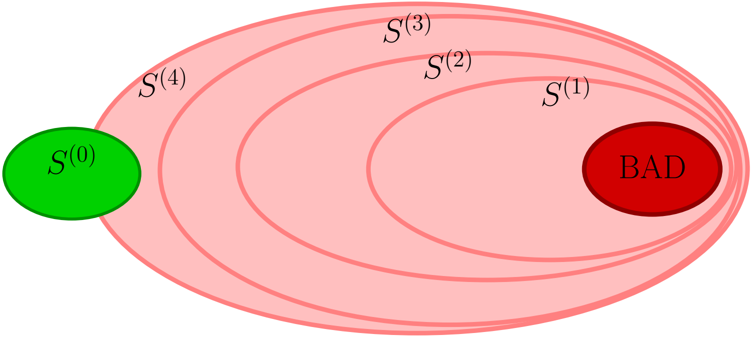
Safety Check	If $S^{(i)}$ contains an initial, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T^{-1}(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Checking - Backward Reachability

Backward-Reachability ($S^{(0)} \equiv$ "bad states")

Safety Check	If $S^{(i)}$ contains an initial, return **unsafe**
Next States	Compute $S^{(i+1)} := S^{(i)} \cup T^{-1}(S^{(i)})$
Fix-Point Check	If $S^{(i+1)} \equiv S^{(i)}$, return **safe**



Implementing a Model-Checker

In order to implement model-checker we need:

1. representing large sets of states
2. computing $T(S^{(i)})$
3. check if bad states are in $S^{(i)}$
4. check if $S^{(i)} \equiv S^{(i+1)}$

Implementing a Model-Checker

In order to implement model-checker we need:

1. representing large sets of states
2. computing $T(S^{(i)})$
3. check if bad states are in $S^{(i)}$
4. check if $S^{(i)} \equiv S^{(i+1)}$

The naive way would be to represent states **explicitly** (e.g., with a C `struct` containing values for state variables)

Very few model-checkers adopt this method (e.g., SPIN)

Implementing a Model-Checker

In order to implement model-checker we need:

1. representing large sets of states
2. computing $T(S^{(i)})$
3. check if bad states are in $S^{(i)}$
4. check if $S^{(i)} \equiv S^{(i+1)}$

The naive way would be to represent states **explicitly** (e.g., with a C `struct` containing values for state variables)

Very few model-checkers adopt this method (e.g., SPIN)

A more powerful approach represents states **symbolically**, by means of SAT/SMT-formulæ: each set of states S is represented by a formula ϕ such that S corresponds to the models of ϕ

Symbolic Model-Checking - Representing States

Examples:

door=open engine=on tray=empty

door_open $\wedge$ engine_on $\wedge$ $\neg$ tray_full

Symbolic Model-Checking - Representing States

Examples:

door=open engine=on tray=empty	$\text{door\_open} \wedge \text{engine\_on} \wedge \neg\ \text{tray\_full}$
door=closed engine=on tray=empty	$\neg\ \text{door\_open} \wedge \text{engine\_on} \wedge \neg\ \text{tray\_full}$

Examples:

door=open
engine=on
tray=empty

$$\mathrm{door\_open} \wedge \mathrm{engine\_on} \wedge \neg\, \mathrm{tray\_full}$$

door=closed
engine=on
tray=empty

$$\neg\, \mathrm{door\_open} \wedge \mathrm{engine\_on} \wedge \neg\, \mathrm{tray\_full}$$

door=open
engine=on
tray=empty

door=closed
engine=on
tray=empty

$$\mathrm{engine\_on} \wedge \neg\, \mathrm{tray\_full}$$

Examples:

door=open engine=on tray=empty		door_open $\wedge$ engine_on $\wedge$ $\neg$ tray_full
door=closed engine=on tray=empty		$\neg$ door_open $\wedge$ engine_on $\wedge$ $\neg$ tray_full
door=open engine=on tray=empty	door=closed engine=on tray=empty	engine_on $\wedge$ $\neg$ tray_full

Also, it is easy to see that:

$S_1 \cup S_2$	$\phi_1 \vee \phi_2$
$S_1 \cap S_2$	$\phi_1 \wedge \phi_2$
$S_1 \subseteq S_2$	$\phi_1 \rightarrow \phi_2$

Symbolic Model-Checking - Representing Transitions

Transitions are also represented as formulæ between state variables and their primed versions

if	door=closed engine=off	then	tray'=empty engine'=on	[start_washing]

$$\neg \text{door\_open} \wedge \neg \text{engine\_on} \wedge \neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}$$

Transitions are also represented as formulæ between state variables and their primed versions

if	door=closed engine=off	then	tray'=empty engine'=on	[start_washing]

$$\neg \text{door\_open} \wedge \neg \text{engine\_on} \wedge \neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}$$

This formula says that the following pair of states are related

$\neg \text{door\_open} \wedge \neg \text{engine\_on} \wedge \neg \text{tray\_full}$	$\neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}$
$\neg \text{door\_open} \wedge \neg \text{engine\_on} \wedge \text{tray\_full}$	$\neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}$

Symbolic Model-Checking - Computing Next State

From a set of states $S^{(i)}$, represented symbolically by a formula $\phi(\vec{s})$, and a transition t_j, represented symbolically by a formula $\psi(\vec{s}, \vec{s'})$, the next states $t_j(S^{(i)})$ can be expressed as

$$\exists \vec{s}.\ \phi(\vec{s}) \wedge \psi(\vec{s}, \vec{s'})$$

By means of an operation called **quantifier elimination**, we can remove $\vec{s}$. If then we rename $\vec{s'}$ as $\vec{s}$ we obtain the symbolic representation of $t_j(S^{(i)})$

Symbolic Model-Checking - Computing Next State

From a set of states $S^{(i)}$, represented symbolically by a formula $\phi(\vec{s})$, and a transition t_j, represented symbolically by a formula $\psi(\vec{s}, \vec{s'})$, the next states $t_j(S^{(i)})$ can be expressed as

$$\exists \vec{s}.\ \phi(\vec{s}) \wedge \psi(\vec{s}, \vec{s'})$$

By means of an operation called **quantifier elimination**, we can remove $\vec{s}$. If then we rename $\vec{s'}$ as $\vec{s}$ we obtain the symbolic representation of $t_j(S^{(i)})$

Example:

$$\begin{aligned}\phi &\equiv \neg \text{door\_open} \wedge \neg \text{engine\_on} \\ \psi &\equiv \neg \text{door\_open} \wedge \neg \text{engine\_on} \wedge \neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}\end{aligned}$$

Quantifier elimination of $\exists\ \text{door\_open}, \text{engine\_on}.\ \phi \wedge \psi$ is

$$\neg \text{door\_open'} \wedge \text{engine\_on'} \wedge \neg \text{tray\_full'}$$

and therefore

$$\neg \text{door\_open} \wedge \text{engine\_on} \wedge \neg \text{tray\_full}$$

is $t_j(S^{(i)})$. The whole set of next states $T(S^{(i)})$ is $\bigvee_j t_j(S^{(i)})$

Symbolic Model-Checking - Bad states in $S^{(i)}$

Suppose that ϕ is the symbolic representation of $S^{(i)}$, and that β is the symbolic representation of the **bad states**

checking if some bad state is in $S^{(i)}$ can be simply done with

Symbolic Model-Checking - Bad states in $S^{(i)}$

Suppose that ϕ is the symbolic representation of $S^{(i)}$, and that β is the symbolic representation of the **bad states**

checking if some bad state is in $S^{(i)}$ can be simply done with

$$\phi \wedge \beta \text{ is satisfiable ?}$$

Symbolic Model-Checking - Fix point test

Suppose that ϕ_i is the symbolic representation of $S^{(i)}$ and that ϕ_{i+1} is the symbolic representation of $S^{(i+1)}$ how do I that $S^{(i)} \equiv S^{(i+1)}$?

Symbolic Model-Checking - Fix point test

Suppose that ϕ_i is the symbolic representation of $S^{(i)}$ and that ϕ_{i+1} is the symbolic representation of $S^{(i+1)}$ how do I that $S^{(i)} \equiv S^{(i+1)}$?

First of all, notice that $S^{(i)} \equiv S^{(i+1)}$ if and only if

$$S^{(i)} \subseteq S^{(i+1)} \text{ and } S^{(i+1)} \subseteq S^{(i)}$$

Symbolic Model-Checking - Fix point test

Suppose that ϕ_i is the symbolic representation of $S^{(i)}$ and that ϕ_{i+1} is the symbolic representation of $S^{(i+1)}$ how do I that $S^{(i)} \equiv S^{(i+1)}$?

First of all, notice that $S^{(i)} \equiv S^{(i+1)}$ if and only if

$$S^{(i)} \subseteq S^{(i+1)} \text{ and } S^{(i+1)} \subseteq S^{(i)}$$

$S^{(i)} \subseteq S^{(i+1)}$ always holds (explored states grow monotonically)

Symbolic Model-Checking - Fix point test

Suppose that ϕ_i is the symbolic representation of $S^{(i)}$ and that ϕ_{i+1} is the symbolic representation of $S^{(i+1)}$ how do I that $S^{(i)} \equiv S^{(i+1)}$?

First of all, notice that $S^{(i)} \equiv S^{(i+1)}$ if and only if

$$S^{(i)} \subseteq S^{(i+1)} \text{ and } S^{(i+1)} \subseteq S^{(i)}$$

$S^{(i)} \subseteq S^{(i+1)}$ always holds (explored states grow monotonically)

$S^{(i+1)} \subseteq S^{(i)}$ can be perfomed with the following check

$\phi_{i+1} \rightarrow \phi_i$ is a tautology ? or equivalently
$\phi_{i+1} \wedge \neg\phi_i$ is unsafisfiable ?

Symbolic Model-Checking - Summary

Model-Checking can be implemented by representing states and transitions symbolically with SAT/SMT-formulæ

Next states $T(S^{(i)})$ can be computed using quantifier elimination

Presence of bad states can be computed with a satisfiability call of the form $\phi \wedge \beta$

Fix-point check can be computed with a satisfiability call of the form $\phi_{i+1} \wedge \neg \phi_i$

Symbolic Model-Checking - Termination

Forward-Reachability

Safety Check	If $\phi_i \wedge \beta$ is satisfiable, return **unsafe**
Next States	Compute ϕ_{i+1} with quantifier elimination
Fix-Point Check	If $\phi_{i+1} \wedge \neg\phi_i$ is unsatisfiable, return **safe**

Model-Checking always terminates if the satisfiability tests above terminates

- If the system under inspection is a **finite state machine**, everything can be encoded into Booleans, and so they always terminate (SAT-solver is enough)
- If the system has **infinite states** (e.g., $0 \leq x \wedge y \geq 2$), it terminates if everyting can be encoded into a decidable SMT theory (e.g., $\mathcal{LIA}$) (SMT-Solver necessary)
- If quantifiers are needed to express states, then Forward-Reachability might not terminate (SMT-Solver plus clever way of handling quantifiers)

Outline

1. Basics
   - Modeling
   - Checking
   - Implementing a Model-Checker

2. MCMT
   - Two simple protocols

MCMT is a Model-Checker invented and developed by S. Ghilardi and S. Ranise et al. (see http://www.dsi.unimi.it/ ghilardi/mcmt/ for complete and precise acknowledgements)

It implements a Symbolic Backward-Reachability algorithm (it relies on yices)

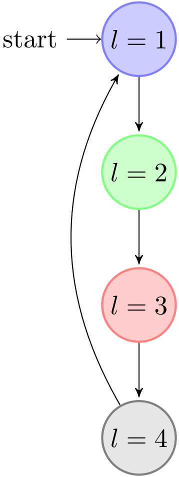
It was invented to handle safety properties for distributed algorithms (protocols), which are infinite-state systems

The following example is taken from the tutorial

Model Checking Modulo Theories: Theory and Practice

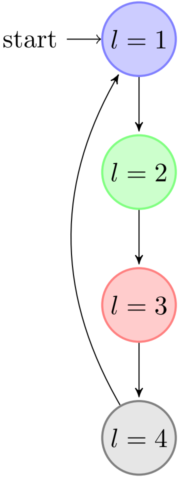
available at `http://st.fbk.eu/MCMTtutorial`

A simple protocol
Description



- No data, only locations
- All processes start from the 1st location
- A process in location 3 is inside the critical section
- We want to check if the protocol ensures the mutual exclusion, i.e., at most one process is inside the critical section

A simple protocol
Variable

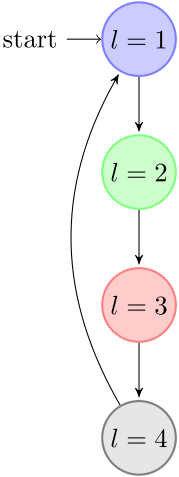


- One *local* variable `l`

```
:smt (define-type locations (subrange 1 4))

:local l locations
```

A simple protocol

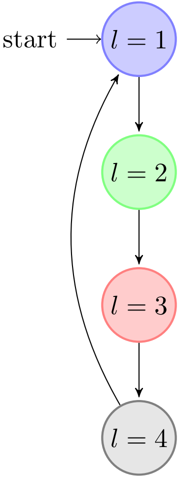
Initial configuration



- All processes start in location 1

$$\forall x.(\texttt{l}[x] = 1)$$

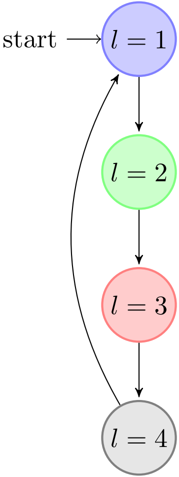
A simple protocol
Initial configuration



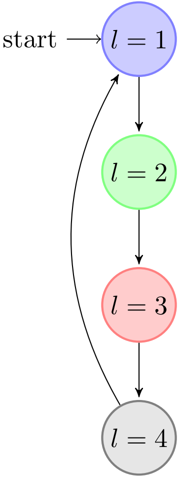
- All processes start in location 1

$$\forall x.(\mathtt{l}[x] = 1)$$

```
:initial
:var x
:cnj (= l[x] 1)
```



- Mutual exclusion: At most one process is in location 3

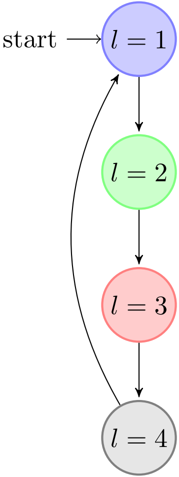


- Mutual exclusion: At most one process is in location 3

$$U := \exists z_1, z_2. (\mathtt{l}[z_1] = 3 \land \mathtt{l}[z_2] = 3 \land z_1 \neq z_2)$$

A simple protocol

Unsafe configuration



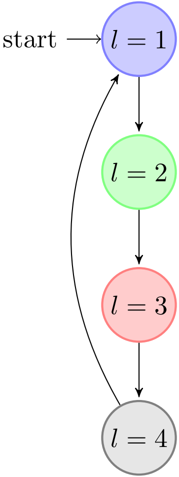
- Mutual exclusion: At most one process is in location 3

$$U := \exists z_1, z_2. (\mathtt{l}[z_1] = 3 \land \mathtt{l}[z_2] = 3 \land z_1 \neq z_2)$$

```
:unsafe
:var z1
:var z2
:cnj (= l[z1] 3) (= l[z2] 3)
```

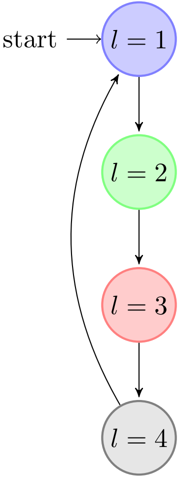
A simple protocol

Transitions



- A process in location 1 moves to location 2

$$\tau_1 := \exists x. \left(\begin{array}{l} \mathtt{l}[x] = 1 \wedge \\ \mathtt{l}' = \lambda j. (\textsf{if } (x = j) \textsf{ then } 2 \textsf{ else } \mathtt{l}[j]) \end{array} \right)$$



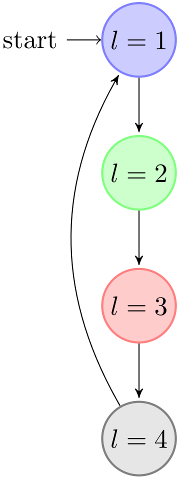
- A process in location 1 moves to location 2

$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1 \wedge \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \end{pmatrix}$$

```
:transition
:var x
:var j
:guard (= l[x] 1)
:numcases 2
:case (= x j)
 :val 2
:case (not (= x j))
 :val l[j]
```

A simple protocol

Transitions



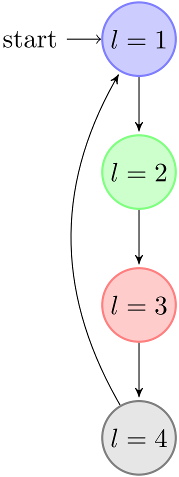

```
:transition
:var x
:var j
:guard (= l[x] 1)
:numcases 2
:case (= x j)
 :val 2
:case (not (= x j))
 :val l[j]
```

```
:transition
:var x
:var j
:guard (= l[x] 2)
:numcases 2
:case (= x j)
 :val 3
:case (not (= x j))
 :val l[j]
```

```
:transition
:var x
:var j
:guard (= l[x] 3)
:numcases 2
:case (= x j)
 :val 4
:case (not (= x j))
 :val l[j]
```

```
:transition
:var x
:var j
:guard (= l[x] 4)
:numcases 2
:case (= x j)
 :val 1
:case (not (= x j))
 :val l[j]
```

A simple protocol
Execution



- `$ ./mcmt simple_unsafe.in`

A simple protocol

Execution - Get informations from counterexample

```
[...]
Doing state space exploration...
 node 1= [t2_1][0]
 node 2= [t1_1][t2_1][0]
 node 3= [t2_2][t2_1][0]
 node 4= [t2_2][t1_1][t2_1][0]
 node 5= [t4_1][t1_1][t2_1][0]
 node 6= [t1_2][t2_2][t1_1][t2_1][0]

============================================================================
System is UNSAFE!
[...]
```

A simple protocol

Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

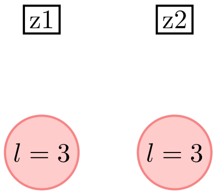
Counter-example: `node 6 = [t1_2][t2_2][t1_1][t2_1][0]`

A simple protocol
Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`



`[0]` $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

A simple protocol

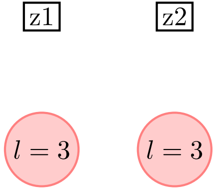
Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\text{if } (x = j) \text{ then } 3 \text{ else } \mathtt{l}[j]) \end{pmatrix}$$



`[0]` $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

A simple protocol

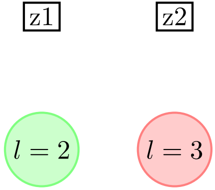
Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j. (\text{if } (x = j) \text{ then } 3 \text{ else } \mathtt{l}[j]) \end{pmatrix}$$



[0]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

A simple protocol

Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1 \ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \end{pmatrix}$$

$\boxed{\mathrm{z1}}$ $\boxed{\mathrm{z2}}$

$l = 2$ $l = 3$

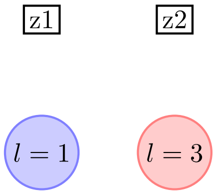
[0]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \land \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2][t2_2][t1_1][t2_1][0]`

$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \land \\ \mathtt{l}' = \lambda j. (\textsf{if}\ (x = j)\ \textsf{then}\ 2\ \textsf{else}\ \mathtt{l}[j]) \end{pmatrix}$$



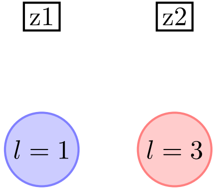
[0]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \land \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \land \mathtt{l}[\mathtt{z2}] = 3\ )$
[t1_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \land \mathtt{l}[\mathtt{z2}] = 3\ )$

A simple protocol

Counterexample analysis from trace

Initial state:	$\forall i.\ (\ \mathtt{l}[i] = 1\ )$
Unsafe state:	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
Counter-example:	`node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j. (\text{if } (x = j) \text{ then } 3 \text{ else } \mathtt{l}[j]) \end{pmatrix}$$



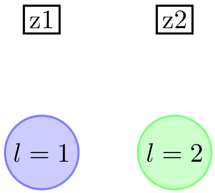
`[0]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
`[t2_1]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
`[t1_1]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 3\ \textsf{else}\ \mathtt{l}[j]) \end{pmatrix}$$



[0]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t1_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_2]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 2\ )$

A simple protocol

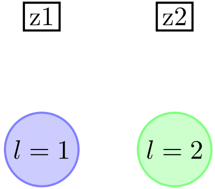
Counterexample analysis from trace

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\text{if } (x = j) \text{ then } 2 \text{ else } \mathtt{l}[j]) \end{pmatrix}$$



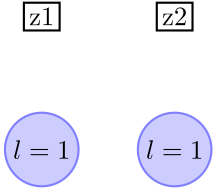
[0]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t1_1]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
[t2_2]	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 2\ )$

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1\ )$

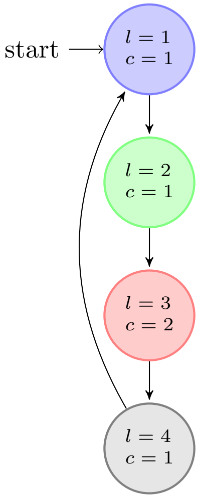
Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$

Counter-example: `node 6 = [t1_2] [t2_2] [t1_1] [t2_1] [0]`

$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \end{pmatrix}$$



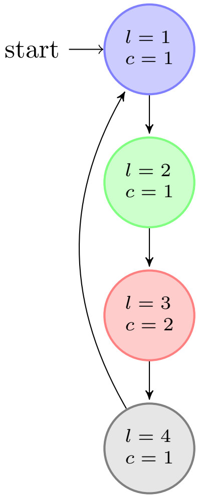
`[0]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
`[t2_1]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 2 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
`[t1_1]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$
`[t2_2]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 2\ )$
`[t1_2]`	$\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 1 \wedge \mathtt{l}[\mathtt{z2}] = 1\ )$



- Like before, but with a **global** flag `c` that takes care of mutual exclusion
- All processes start from the 1st location
- A process in location 3 is inside the critical section
- We want to check if the protocol ensures the mutual exclusion, i.e., at most one process is inside the critical section

Another simple protocol

Variable(s)

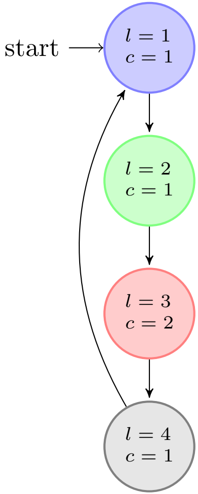


- One *local* variable `l`

```
:smt (define-type locations (subrange 1 4))
:smt (define-type counter (subrange 1 2))

:local l location
:global c counter
```

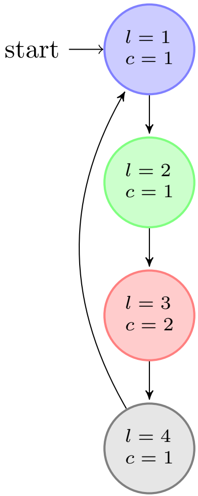
Another simple protocol

Initial configuration



- All processes start in location 1, with counter set to 1

$$\forall x.\ (\ \texttt{l}[x] = 1 \wedge \texttt{c}[x] = 1\ )$$



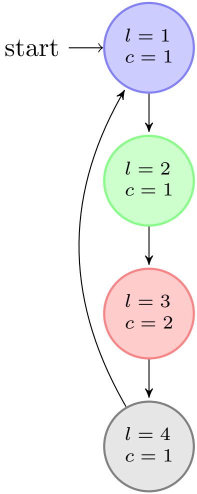
- All processes start in location 1, with counter set to 1

$$\forall x.\ (\ \mathtt{l}[x] = 1 \land \mathtt{c}[x] = 1\ )$$

```
:initial
:var x
:cnj (= l[x] 1) (= c[x] 1)
```

Another simple protocol

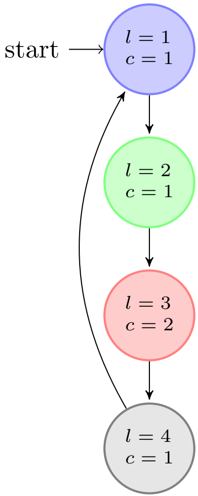
Unsafe configuration



- Mutual exclusion: At most one process is in location 3

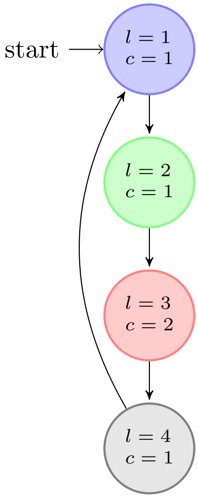
Another simple protocol

Unsafe configuration



- Mutual exclusion: At most one process is in location 3

$$U := \exists z_1, z_2. (\mathtt{l}[z_1] = 3 \land \mathtt{l}[z_2] = 3 \land z_1 \neq z_2)$$



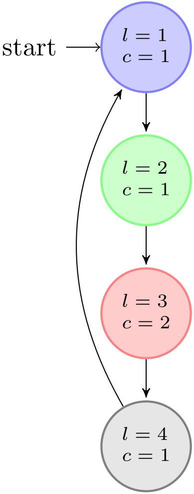
- Mutual exclusion: At most one process is in location 3

$$U := \exists z_1, z_2. (\mathtt{l}[z_1] = 3 \wedge \mathtt{l}[z_2] = 3 \wedge z_1 \neq z_2)$$

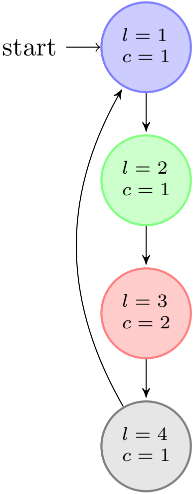
```
:unsafe
:var z1
:var z2
:cnj (= l[z1] 3) (= l[z2] 3)
```

Another simple protocol

Transitions



$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathsf{c}[x] = 1 \;\wedge \\ \mathtt{l}' = \lambda j.\, (\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.2 \end{pmatrix}$$

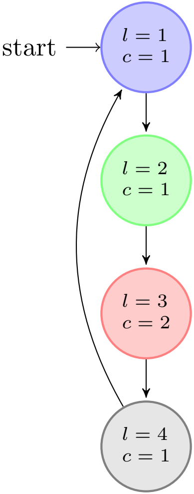


$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \land \mathsf{c}[x] = 1 \ \land \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.2 \end{pmatrix}$$

```
:transition
:var x
:var j
:guard (= l[x] 2) (= c[x] 1)
:numcases 2
:case (= x j)
 :val 3
 :val 2
:case (not (= x j))
 :val l[j]
 :val 2
```

Another simple protocol

Transitions




```
:transition
:var x
:var j
:guard (= l[x] 1)
:numcases 2
:case (= x j)
 :val 2
 :val c[x]
:case (not (= x j))
 :val l[j]
 :val c[x]
```

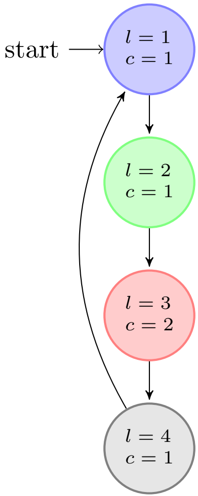
```
:transition
:var x
:var j
:guard (= l[x] 2) (= c[x] 1)
:numcases 2
:case (= x j)
 :val 3
 :val 2
:case (not (= x j))
 :val l[j]
 :val 2
```

```
:transition
:var x
:var j
:guard (= l[x] 3) (= c[x] 2)
:numcases 2
:case (= x j)
 :val 4
 :val 1
:case (not (= x j))
 :val l[j]
 :val 1
```

```
:transition
:var x
:var j
:guard (= l[x] 4)
:numcases 2
:case (= x j)
 :val 1
 :val c[j]
:case (not (= x j))
 :val l[j]
 :val c[j]
```

Another simple protocol

Execution



- `$ ./mcmt simple_safe.in`

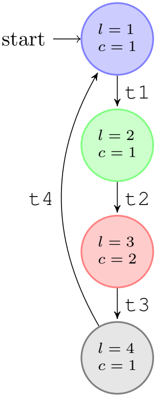
Another simple protocol

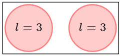
Execution

```
[...]
Doing state space exploration...
 node 1 = [t2_1][0]
 node 2 = [t1_1][t2_1][0]
 node 3 = [t4_1][t1_1][t2_1][0]

============================================================================
Global fixpoint reached!

System is SAFE!
[...]
```

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \wedge \mathtt{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathtt{c}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.2 \end{pmatrix}$$

$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \wedge \mathtt{c}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 4\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.1 \end{pmatrix}$$

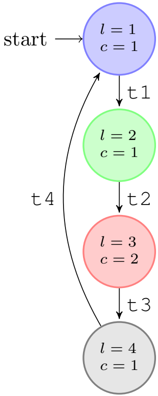
$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 1\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$

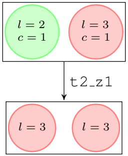
A simple protocol

Set of (un)reachable states

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \wedge \mathsf{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.\mathsf{c}[j] \end{pmatrix}$$

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathsf{c}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.2 \end{pmatrix}$$

$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \wedge \mathsf{c}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 4\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.1 \end{pmatrix}$$

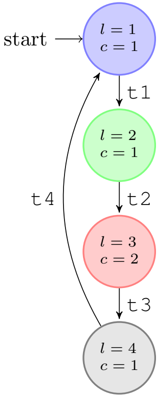
$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 1\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.\mathsf{c}[j] \end{pmatrix}$$

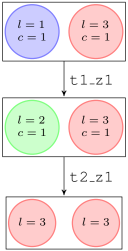
A simple protocol

Set of (un)reachable states

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \land \mathtt{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \land \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1 \land \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 2\ \textsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.\mathsf{c}[j] \end{pmatrix}$$

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \land \mathsf{c}[x] = 1 \land \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 3\ \textsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.2 \end{pmatrix}$$

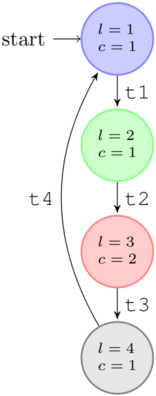
$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \land \mathsf{c}[x] = 2 \land \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 4\ \textsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.1 \end{pmatrix}$$

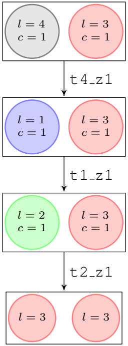
$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4 \land \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 1\ \textsf{else}\ \mathtt{l}[j]) \\ \mathsf{c}' = \lambda j.\mathsf{c}[j] \end{pmatrix}$$

A simple protocol
Set of (un)reachable states

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \wedge \mathtt{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j. \mathtt{c}[j] \end{pmatrix}$$

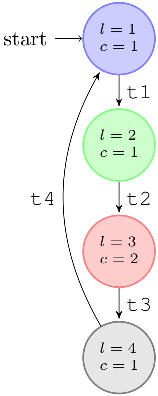
$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathtt{c}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j. 2 \end{pmatrix}$$

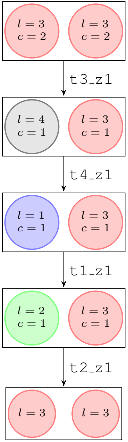
$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \wedge \mathtt{c}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 4\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j. 1 \end{pmatrix}$$

$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4\ \wedge \\ \mathtt{l}' = \lambda j. (\mathsf{if}\ (x = j)\ \mathsf{then}\ 1\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j. \mathtt{c}[j] \end{pmatrix}$$

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \wedge \mathtt{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 2\ \textsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathtt{c}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 3\ \textsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.2 \end{pmatrix}$$

$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \wedge \mathtt{c}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 4\ \textsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.1 \end{pmatrix}$$

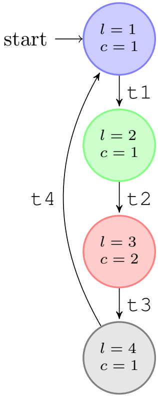
$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4\ \wedge \\ \mathtt{l}' = \lambda j.\,(\textsf{if}\ (x = j)\ \textsf{then}\ 1\ \textsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$

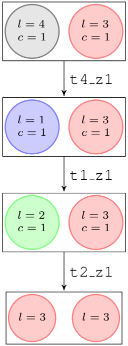
A simple protocol

Set of (un)reachable states

Initial state: $\forall i.\ (\ \mathtt{l}[i] = 1 \wedge \mathtt{c}[i] = 1\ )$

Unsafe state: $\exists \mathtt{z1}, \mathtt{z2}.\ (\ \mathtt{l}[\mathtt{z1}] = 3 \wedge \mathtt{l}[\mathtt{z2}] = 3\ )$





$$\tau_1 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 2\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$

$$\tau_2 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 2 \wedge \mathtt{c}[x] = 1\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 3\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.2 \end{pmatrix}$$

$$\tau_3 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 3 \wedge \mathtt{c}[x] = 2\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 4\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.1 \end{pmatrix}$$

$$\tau_4 := \exists x. \begin{pmatrix} \mathtt{l}[x] = 4\ \wedge \\ \mathtt{l}' = \lambda j.\,(\mathsf{if}\ (x = j)\ \mathsf{then}\ 1\ \mathsf{else}\ \mathtt{l}[j]) \\ \mathtt{c}' = \lambda j.\mathtt{c}[j] \end{pmatrix}$$