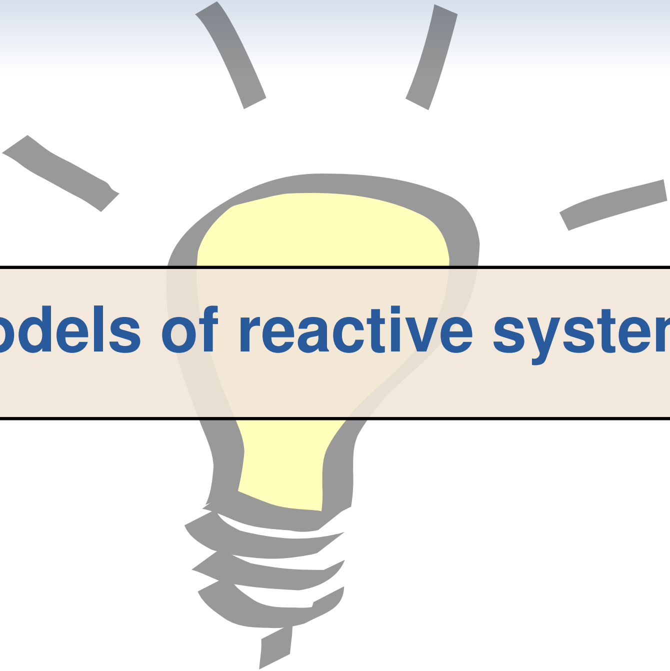
Model Checking and Games

Part II - Basics of Modelling Reactive Systems

Rüdiger Ehlers, Clausthal University of Technology

September 2019



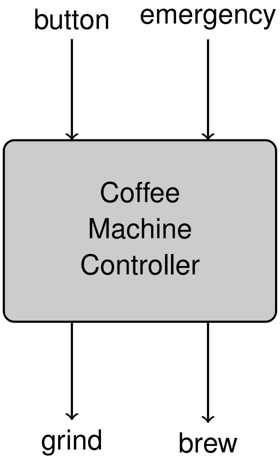
Models of reactive systems

Traces

Reactive systems

- We assume that a system evolves in *steps*
- At every point in time, a reactive system reads its *input* and writes to its *output*
- When we continually observe the input/output of a system, we obtain a *trace* of the system

Reactive systems - example



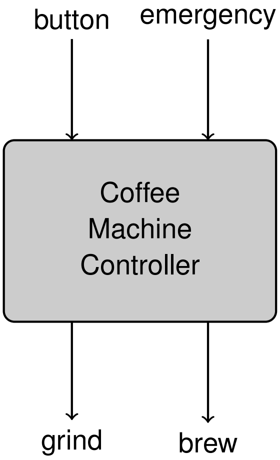
Input/output shape

Here: In every step, the input *bit* has a value and the output *bits* have values, too.

Atomic propositions

- $\mathsf{AP}_I = \{$*button*, *emergency*$\}$
- $\mathsf{AP}_O = \{$*grind*, *brew*$\}$

Reactive systems - example



Input/output shape

Here: In every step, the input *bit* has a value and the output *bits* have values, too.

Atomic propositions

- $\mathrm{AP}_I = \{\mathit{button}, \mathit{emergency}\}$
- $\mathrm{AP}_O = \{\mathit{grind}, \mathit{brew}\}$

A trace of the system

$$\rho = \begin{pmatrix}0\\0\\0\\0\end{pmatrix}\begin{pmatrix}1\\0\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\1\\0\end{pmatrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}\begin{pmatrix}0\\0\\0\\0\end{pmatrix}\cdots$$

Mealy & Moore machines

Order of the input and output

- **Mealy machine:** in every step of the system's execution, the system first reads the input and then produces the output.
- **Moore machine:** in every step of the system's execution, the system first produces the output and then reads the input.

Mealy machines

Definition

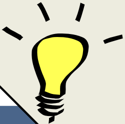
A Mealy machine is a tuple $\mathcal{M} = (S, \Sigma^I, \Sigma^O, \delta, s_0)$ with:

- the set of states S,
- the input alphabet Σ^I,
- the output alphabet Σ^O,
- the transition function $\delta : S \times \Sigma^I \rightarrow S \times \Sigma^O$, and
- the initial state $s_0 \in S$.

Traces & runs

A *trace* of $\mathcal{M}$ is an infinite sequence $\rho = (\rho_0^O, \rho_0^I)(\rho_1^O, \rho_1^I) \ldots \in (\Sigma^I \times \Sigma^O)^\omega$ such that there exists a corresponding *run* of $\mathcal{M}$ of the form $\pi = \pi_0 \pi_1 \ldots \in S^\omega$ with $(\pi_{i+1}, \rho_i^O) = \delta(\pi_i, \rho_i^I)$ for for all $i \in \mathbb{N}$ and $\pi_0 = s_0$.

On notation

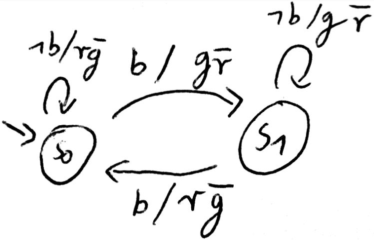


A note on alphabets

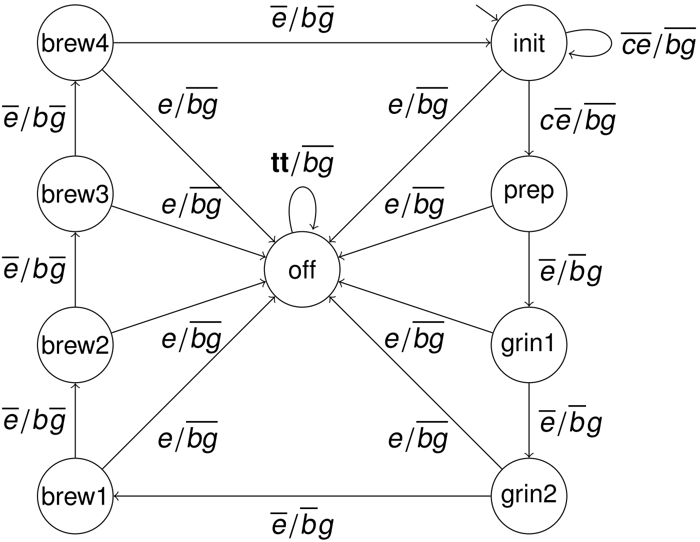
In most examples, we will have $\Sigma^I = 2^{AP^I}$ and $\Sigma^O = 2^{AP^O}$ for some sets of *atomic propositions* AP^I and AP^O.
The definition of Mealy machines does not require this, however.

Graphical notation for Mealy machines

Example: $\mathcal{M} = (S, \Sigma^I, \Sigma^O, \delta, s_0)$ with $S = \{s_0, s_1\}$, $\Sigma^I = 2^{\{b\}}$, $\Sigma^O = 2^{\{r,g\}}$, $\delta(s_0, \emptyset) = (s_0, \{r\})$, $\delta(s_0, \{b\}) = (s_1, \{g\})$, $\delta(s_1, \emptyset) = (s_1, \{g\})$, $\delta(s_1, \{b\}) = (s_0, \{r\})$.



Example: Coffee machine



Moore machines

Definition

A Moore machine is a tuple $\mathcal{M} = (S, \Sigma^I, \Sigma^O, \delta, L, s_0)$ with:

- the set of states S,
- the input alphabet Σ^I,
- the output alphabet Σ^O,
- the labeling function $L : S \to \Sigma^O$,
- the transition function $\delta : S \times \Sigma^I \to S$, and
- the initial state $s_0 \in S$.

Traces & runs

A *trace* of $\mathcal{M}$ is an infinite sequence $\rho = (\rho_0^O, \rho_0^I)(\rho_1^O, \rho_1^I) \ldots \in (\Sigma^I \times \Sigma^O)^\omega$ such that there exists a corresponding *run* of $\mathcal{M}$ of the form $\pi = \pi_0\pi_1 \ldots \in S^\omega$ with $\pi_{i+1} = \delta(\pi_i, \rho_i^I)$ and $L(\pi_i) = \rho_i^O$ for all $i \in \mathbb{N}$.

Mealy vs. Moore machines (1)

Differences

- Mealy machines read the input *before* selecting the next output
- Moore machines read the input *after* selecting the next output

Mealy $\rightarrow$ Moore translation

Translation Moore $\rightarrow$ Mealy

Given a Moore machine $\mathcal{M} = (S, \Sigma^I, \Sigma^O, \delta, L, s_0)$, we define its corresponding translation to a Mealy machine $\mathcal{M}' = (S, \Sigma^I, \Sigma^O, \delta', s_0)$ with:

- $\delta'(s, i) = (\delta(s, i), L(s))$ for all $s \in S$, $i \in \Sigma^I$

Lemma

Claim: $\mathcal{M}$ and $\mathcal{M}'$ induce the same set of traces.

Proof: $\Rightarrow$ Let $\rho = (\rho_0^O, \rho_0^I)(\rho_1^O, \rho_1^I)\ldots \in$ be a trace of $\mathcal{M}$ and $\pi = \pi_0\pi_1\ldots$ be the corresponding run. So we have that for every $i \in \mathbb{N}$, by definition, $\pi_{i+1} = \delta(\pi_i, \rho_i^I)$ and $L(\pi_i) = \rho_i^O$ for all $i \in \mathbb{N}$. By the definition of δ', we also have $\delta'(\pi_i, \rho_i^I) = (\pi_{i+1}, \rho_i^O)$ for all $i \in \mathbb{N}$. Since $\pi_0 = s_0$ (by assumption), this means that ρ is also a trace of $\mathcal{M}'$.

$\Leftarrow$: Analogous

Less structured models: Transition systems

Definition

A *transition system* is a tuple (S, S_0, T) with:

- a set of states S,
- a set of initial states S_0, and
- a set of transitions $T \subseteq S \times S$.

Transition systems (2)

Definition

Given a transition system $\mathcal{T} = (S, S_0, T)$, we say that

- a sequence $\pi = \pi_0\pi_1 \ldots \in S^\omega$ is an *infinite run* of $\mathcal{T}$ if we have $\pi_0 \in S_0$ and for every $i \in \mathbb{N}$, we have $(\pi_i, \pi_{i+1}) \in T$, and
- a sequence $\pi = \pi_0\pi_1 \ldots \pi_n \in S^*$ is a *finite run* of $\mathcal{T}$ if we have $\pi_0 \in S_0$ and for every $i \in \{0, \ldots, n-1\}$, we have $(\pi_i, \pi_{i+1}) \in T$.

Transition systems vs. Mealy and Moore machines

Differences

- Transition systems have less structure: No explicit input or output, no labels.
- Transition systems have *non-determinism*: no single initial state, system can behave arbitrarily.

Kripke structures: *Labeled transition systems*

Definition

A *Kripke structure* is a tuple $(S, S_0, T, \mathrm{AP}, L)$ with:

- a set of states S,
- a set of initial states S_0, and
- a set of transitions $T \subseteq S \times S$.
- a set of *propositions* AP, and
- a labelling function $L : S \to 2^{\mathrm{AP}}$.

Traces of Kripke structures

Definition (infinite trace)

If $\pi = \pi_0\pi_1 \ldots \in S^\omega$ is an *infinite run* of $\mathcal{T}$, then we call $\rho = L(\pi_0)L(\pi_1)\ldots$ an infinite *trace* of $\mathcal{T}$.

Definition (finite trace)

If $\pi = \pi_0\pi_1 \ldots \pi_n \in S^*$ is a *finite run* of $\mathcal{T}$, then we call $\rho = L(\pi_0)L(\pi_1)\ldots L(\pi_n)$ a finite *trace* of $\mathcal{T}$.

Comparison: LTS vs Mealy & Moore machines

Comparison

Aspect	Mealy / Moore Machines	Labeled Transition systems
Input & Output	Clearly separated	No separation
Determinism	Yes	No
Time	Discrete time steps	No concrete notion of time
Ability to abstract from details	No	Yes
Non-binary alphabets	Yes	No

Use of Kripke structues

They are a good model for verifying systems to be correct!

If we want to analyze for whether a system works correctly, they are a suitable model.

If we capture the desired behavior of the system as a *set of (allowed) traces*, then we can check if the traces of the model are all contained in them.

Use of Kripke structues

They are a good model for verifying systems to be correct!

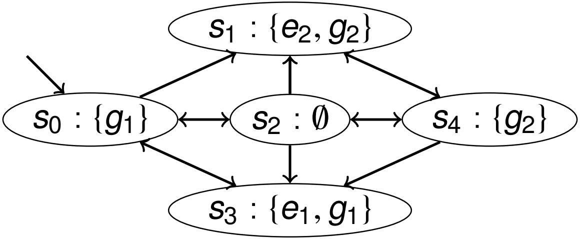
If we want to analyze for whether a system works correctly, they are a suitable model.

If we capture the desired behavior of the system as a *set of (allowed) traces*, then we can check if the traces of the model are all contained in them.

But beware!

The LTS must capture *all* possible executions of the system for this to work!

Revisiting the traffic light with an ambulance override



Question: Are there traces induced by this Kripke structure on which the second green light (g_2) is never lit?

Translation from Mealy/Moore machines to Kripke structures

Definition by example of Moore machines

Given a Moore machine $\mathcal{M} = (S, \Sigma^I, \Sigma^O, \delta, L, s_0)$ with $\Sigma^I = 2^{\mathrm{AP}^I}$ and $\Sigma^O = 2^{\mathrm{AP}^O}$, we define the *corresponding* Kripke structure $\mathcal{K} = (S', S'_0, T', \mathrm{AP}, L')$ with:

- $S' = S \times \Sigma^I \uplus \{\cdot\}$
- $S'_0 = \{\cdot\}$
- $\mathrm{AP} = \mathrm{AP}^I \uplus \mathrm{AP}^O$
- $T = \{(\cdot, (s, i)) \mid i \subseteq 2^{\mathrm{AP}^I}, s \in S_0\} \cup \{((s, i), (s', i')) \mid (s, i), (s', i') \in S \times \Sigma^I . (s, i, s') \in \delta\}$
- For all $(s, i) \in S \times \Sigma^I$, $L'((s, i)) = L(s) \cup i$
- $L'(\cdot) = \emptyset$

Consequence

Consequence

If we want to reason about the set of possible traces of a Mealy or Moore machine (e.g., for verifying that the Mealy or Moore machine has some desired properties), we can do so on the corresponding Kripke structure.

Side-note

...but non-binary alphabets may require some care.

Let’s look at circuits

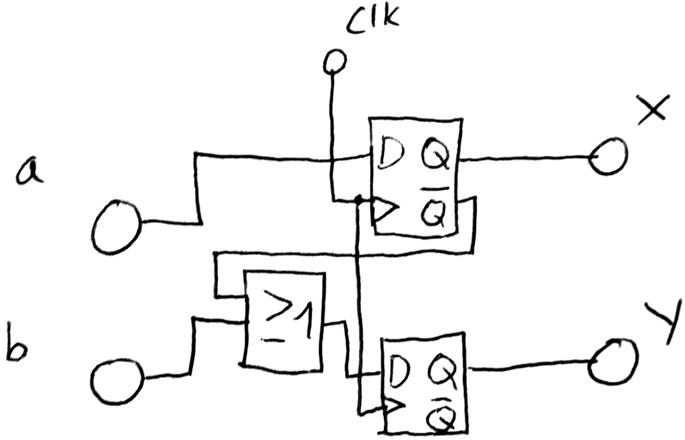
Modelling the behavior of a circuit as a Kripke structure

We can also analyze the behavior of a circuit

Let's look at circuits

Modelling the behavior of a circuit as a Kripke structure

We can also analyze the behavior of a circuit

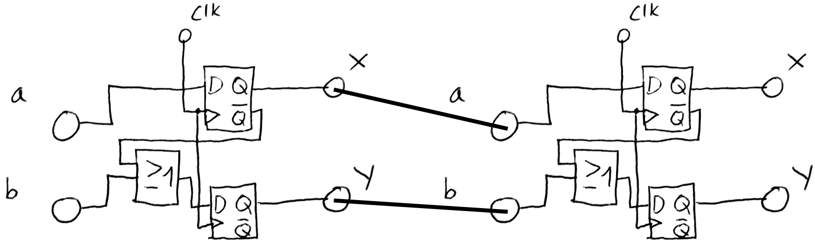


Large circuits

Observation

The Kripke structures can grow quite large in this process. For larger circuits, we need some **structured way** to compose the parts:

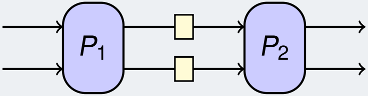
Example:



Synchronous communication between processes

Possibility 1: Synchronous composition in the same time step

Possibility 2: Synchronous composition with delay



Synchronous composition (delayed version)

Definition

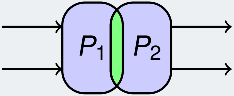
Let $\mathcal{K}^1 = (S^1, S_0^1, T^1, \mathrm{AP}^1, L^1)$ and $\mathcal{K}^2 = (S^2, S_0^2, T^2, \mathrm{AP}^2, L^2)$ be Kripke structures.

We define the *synchronous product (with delay)* of $\mathcal{K}^1$ and $\mathcal{K}^2$ as the Kripke structure $\mathcal{K}^P = (S^P, S_0^P, T^P, \mathrm{AP}^P, L^P)$ with:

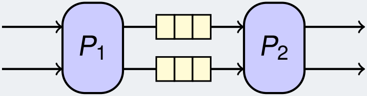
- $S^P = S^1 \times S^2$
- $S_0^P = S_0^1 \times S_0^2$
- $\mathrm{AP}^P = \{p^1 \mid p \in \mathrm{AP}^1\} \cup \{p^2 \mid p \in \mathrm{AP}^2\}$
- $L((s^1, s^2)) = \{p^1 \mid p \in L^1(s^1)\} \cup \{p^2 \mid p \in L^2(s^2)\}$
- $T^P = \{((s^1, s^2), (s'^1, s'^2)) \subseteq S^P \mid L^2(s'^2) \cap \mathrm{AP}^1 \cap \mathrm{AP}^2 = L^1(s^1) \cap \mathrm{AP}^1 \cap \mathrm{AP}^2\}$.

Ansynchronous composition between processes

Possibility 3: Shared variables



Possibility 4: Asynchronous composition with channels



Towards asynchronous composition: Modelling software as Kripke structures

Example program (Peterson's Mutex algorithm, part 1)

```
bool flag0 = false;
bool flag1 = false;
bool turn = false;
while (true) {
    flag0 = true;
    turn = false;
    while (flag1 == true && turn == false) {
        skip;
    }
    while (havoc) {
        skip;
    }
    flag0 = false;
    while (havoc) {
        skip;
    }
}
```

Towards asynchronous composition: Modelling software as Kripke structures

Example program (Peterson's Mutex algorithm, part 1)

```
    bool flag0 = false;
    bool flag1 = false;
    bool turn = false;
l0: while (true) {
l1:     flag0 = true;
l2:     turn = false;
l3:     while (flag1 == true && turn == false) {
l4:         skip;
        }
l5:     while (havoc) {
l6:         skip;
        }
l7:     flag0 = false;
l8:     while (havoc) {
l9:         skip;
        }
    }
```

Towards asynchronous composition: Modelling software as Kripke structures

Example program (Peterson's Mutex algorithm, all parts)

```
                          bool flag0 = false;
                          bool flag1 = false;
                          bool turn = false;
l0: while (true) {                  ||  m0: while (true) {
l1:     flag0 = true;               ||  m1:     flag1 = true;
l2:     turn = false;               ||  m2:     turn = true;
l3:     while (flag1 == true &&     ||  m3:     while (flag0 == true &&
            turn == false) {        ||              turn == true) {
l4:         skip;                   ||  m4:         skip;
        }                           ||          }
l5:     while (havoc) {             ||  m5:     while (havoc) {
l6:         skip;                   ||  m6:         skip;
        }                           ||          }
l7:     flag0 = false;              ||  m7:     flag1 = false;
l8:     while (havoc) {             ||  m8:     while (havoc) {
l9:         skip;                   ||  m9:         skip;
        }                           ||          }
    }                               ||      }
```

Asynchronous composition

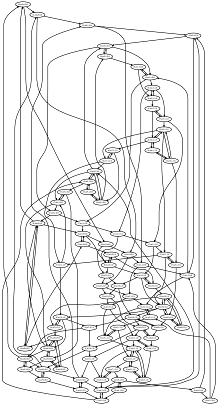
Definition

Let $\mathcal{K}^1 = (S^1, S_0^1, T^1, \mathrm{AP}^1, L^1)$ and $\mathcal{K}^2 = (S^2, S_0^2, T^2, \mathrm{AP}^2, L^2)$ be Kripke structures such that $S^1 = V \times M^1$ and $S^2 = V \times M^2$.

We define the *asynchronous product* of $\mathcal{K}^1$ and $\mathcal{K}^2$ as the Kripke structure $\mathcal{K}^P = (S^P, S_0^P, T^P, \mathrm{AP}^P, L^P)$ with:

- $S^P = M^1 \times M^2 \times V$
- $S_0^P = \{(m^1, m^2, v) \in S^P \mid (m^1, v) \in S_0^1, (m^2, v) \in S_0^2\}$
- $\mathrm{AP}^P = \mathrm{AP}^1 \cup \mathrm{AP}^2$
- $L((m^1, m^2, v)) = L^1((m^1, v)) \cup L^2((m^2, v))$
- $T^P = \{((m^1, m^2, v), (m'^1, m^2, v')) \mid ((m^1, v), (m'^1, v') \in T^1\} \cup \{((m^1, m^2, v), (m^1, m'^2, v')) \mid ((m^2, v), (m'^2, v') \in T^2\}$

Complete product

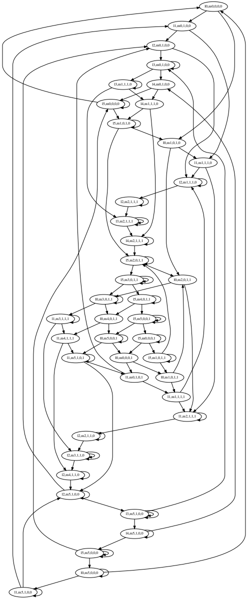


Simplified example

Simplified program (Peterson's Mutex algorithm, all parts)

```
                        bool flag0 = false;
                        bool flag1 = false;
                        bool turn = false;
l0: while (true) {                     ||  m0: while (true) {
        flag0 = true;                  ||          flag1 = true;
l1:     turn = false;                  ||  m1:     turn = true;
l2:     while (flag1 == true &&        ||  m2:     while (flag0 == true &&
            turn == false) {           ||              turn == true) {
            skip;                      ||              skip;
        }                              ||          }
l3:     while (havoc) {                ||  m3:     while (havoc) {
            skip;                      ||              skip;
        }                              ||          }
l4:     flag0 = false;                 ||  m4:     flag1 = false;
l5:     while (havoc) {                ||  m5:     while (havoc) {
            skip;                      ||              skip;
        }                              ||          }
    }                                  ||      }
```

Complete product for the simplified program



Discussion

Kripke structures & transition systems

- Give a account of what states in a system can be reached
- They can be labeled or not depending on what we want to analyze about them:
  - Are certain states reachable?
  - What traces does the Kripke structure induce?
- We already saw the **state space explosion** problem: even for small circuits/programs/models, the number of states can become **huge**!

The model checker `spin`

Some facts

- Developed by Gerard Holzmann
- Uses the specification language `Promela` (for "Process meta language")
- Runs under Unix/Linux, compiles models into executable code for enumerating the states in a transition system

Official Website: `http://spinroot.com`

Getting spin to run

Local “Installation” (using the RZ application server):

```
ssh -Y username@cloud-249.rz.tu-clausthal.de
git clone https://github.com/nimble-code/Spin.git
cd Spin
make
cd ..
```

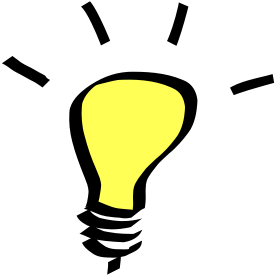
Using the GUI after logging in (with `ssh -Y`):

```
bash
export PATH=$PATH:~/Spin/Src
~/Spin/optional_gui/ispin.tcl
```

Unter Windows, you may want want to try `MobaXTerm` or a similar tool instead of `ssh` (or start from a Linux virtual machine).

Summary / List of Concepts

- Synchronous reactive systems, Mealy and Moore machines
- Labeled transition systems & Kripke structures
- Traces of Mealy machines, Moore machine, and Kripke structures
- Programs as models
- Composition of Kripke structures: synchronous (delayed) and asynchronous (with shared variables)
- Model checker `spin`



References I