02246 - Model Checking

$$M \models \Phi?$$

Lecture 01 - Transition Systems

Technical University of Denmark

$$M \models \Phi?$$

System

refine/implement

Requirements

abstract/model

$M \models \Phi?$

some logic here

Finite-state Transition System

some smart algorithm here

yes :)

maybe :|

no :(

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

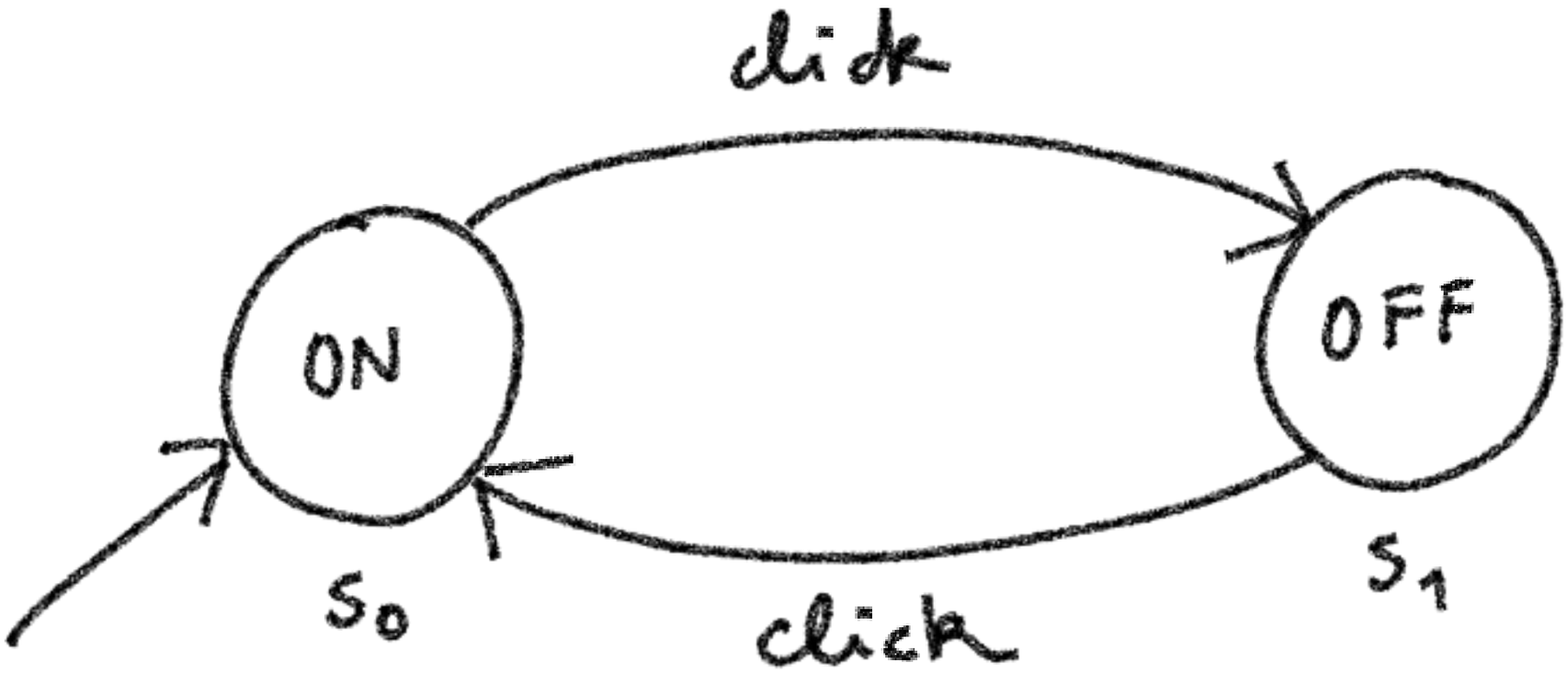
Reading material

Chapter 2 of “Principles of Model Checking”

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

Transition systems, graphically



Transition systems, formally

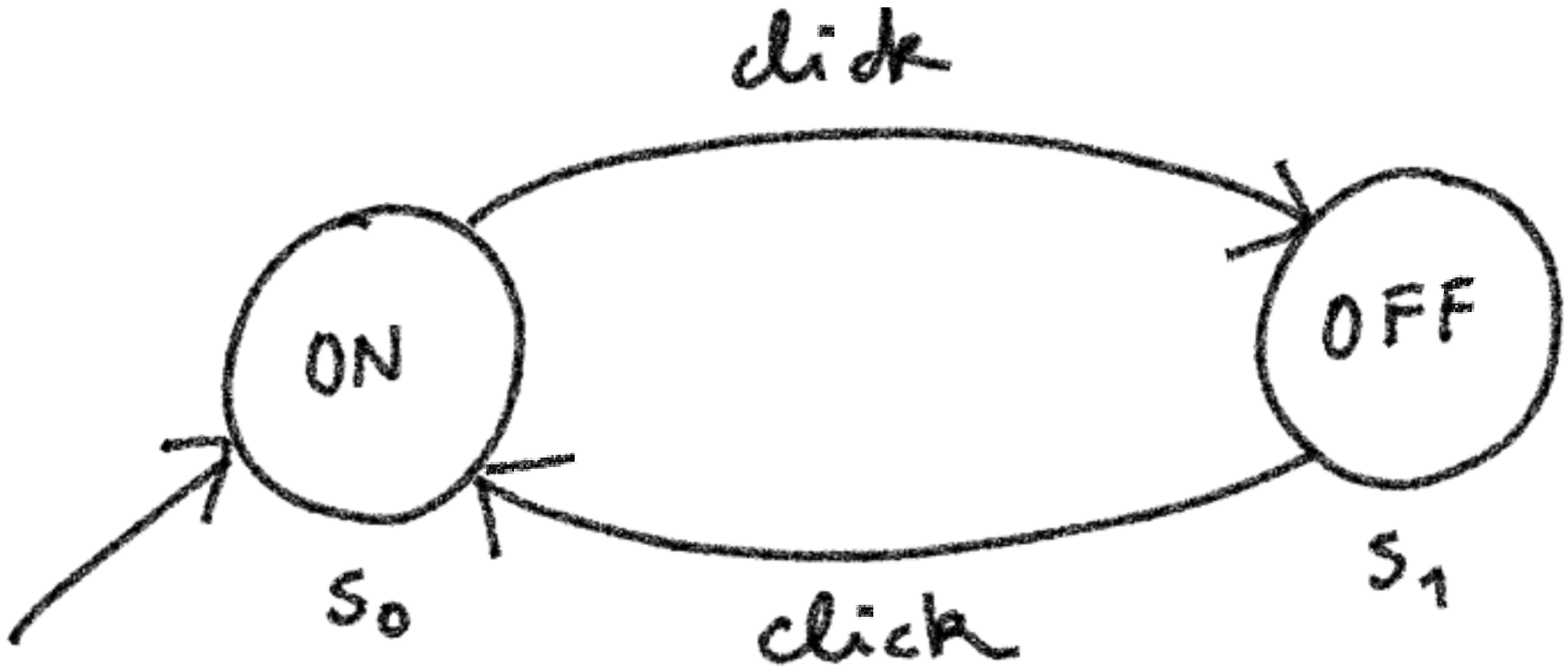
Def. A transition system is a tuple

$$\langle S, A, \rightarrow, L, AP, I \rangle$$

such that

- S is a set of states
- A is a set of "Actions"
- $\rightarrow \subseteq S \times A \times S$ is a set of transitions
- $L : S \longrightarrow 2^{AP}$ is a labelling function
- AP is a finite state of "Atomic Propositions"
- $I \subseteq S$ is the set of initial states

Our example, formally



$S = \{ s_0, s_1 \}$

$\rightarrow = \{ (s_0, click, s_1), (s_1, click, s_0) \}$

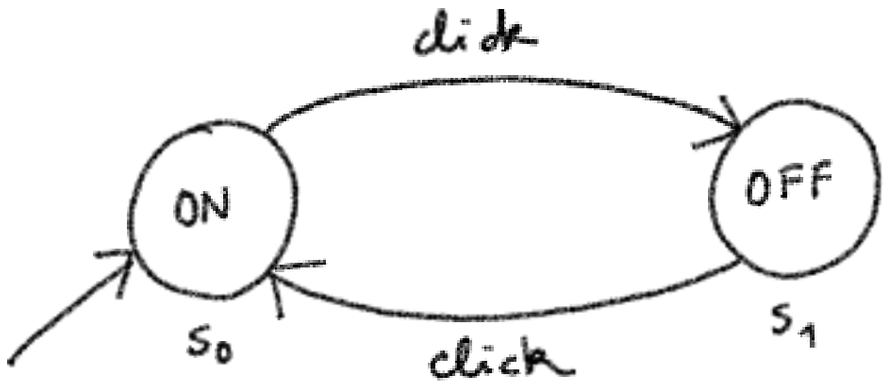
$A = \{ click \}$

$L = \{ s_0 \mapsto \{ON\}, s_1 \mapsto \{OFF\} \}$

$AP = \{ ON, OF \}$

$S_0 = \{ s_0 \}$

Our example, in PRISM



$S = \{ s_0, s_1 \}$

$T = \{ (s_0, click, s_1), (s_1, click, s_0) \}$

$A = \{ click \}$

$L = \{ s_0 \mapsto \{ON\}, s_1 \mapsto \{OFF\} \}$

$AP = \{ ON, OFF \}$

$s_0 = \{ s_0 \}$

PRISM 4.3.1

File Edit Model Properties Simulator Log Options

PRISM Model File: switch.prism*

```
1  mdp
2
3  module SWITCH
4
5    state : [0..1] init 0;
6
7    [click] state=0 -> (state'=1);
8    [click] state=1 -> (state'=0);
9
10 endmodule
11
12 label "ON" = (state=0);
13 label "OFF" = (state=1);
14
```

Model Properties Simulator Log

Parsing model... done.

DEMO?

Forward and backwards reachability

We sometimes need to talk about how states can reach each other.

The set of direct successors of a state s is defined as

$$Post(s) = \{s' \mid \exists \alpha. s \xrightarrow{\alpha} s'\}$$

Forward and backwards reachability

We sometimes need to talk about how states can reach each other.

The set of direct successors of a state s is defined as

$$Post(s) = \{s' \mid \exists \alpha. s \xrightarrow{\alpha} s'\}$$

Similarly, we can define the direct predecessors of a state s as

$$Pre(s) = \{s' \mid \exists \alpha. s' \xrightarrow{\alpha} s\}$$

Forward and backwards reachability

We sometimes need to talk about how states can reach each other.

The set of direct successors of a state s is defined as

$$Post(s) = \{s' \mid \exists \alpha. s \xrightarrow{\alpha} s'\}$$

Similarly, we can define the direct predecessors of a state s as

$$Pre(s) = \{s' \mid \exists \alpha. s' \xrightarrow{\alpha} s\}$$

The above definitions can be lifted to sets of states A.

$$Post(A) = \ldots \qquad Pre(A) = \ldots$$

Successors/predecessors in any number of transitions can be defined likewise.

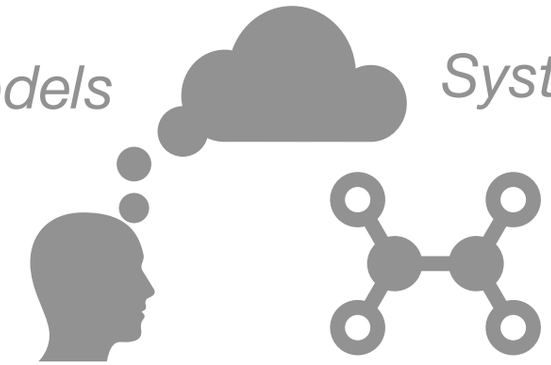
On a related note, one is typically interested in the part of a transition system that is reachable from the initial state(s).

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

Models

Systems



Modelling/Specification/Implementation language

```
Thread 1 { x:=y+z }
Thread 2 { y:=x*z }
```

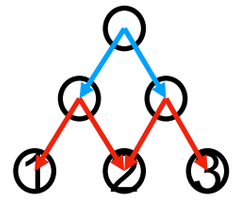
PRISM modelling language specification

```
module M1
…
endmodule

module M2
…
```

PRISM-model specification

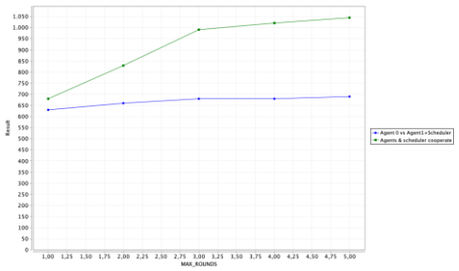
```
…
33 0 6 1 action1
33 1 0 1 action2
…
```



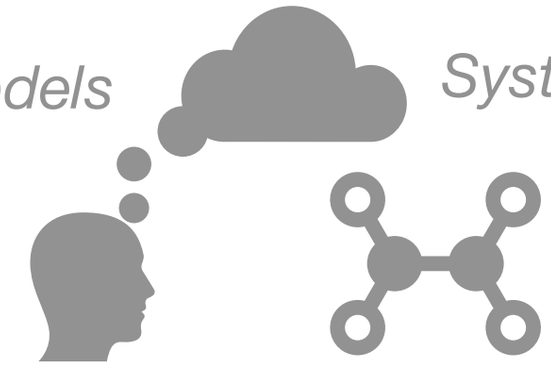
PRISM

ANALYSIS RESULTS

Yes/No, Max/Min Probability/Cost/Reward, plots, etc.



Models *Systems*



Modelling/Specification/Implementation language

```
Thread 1 { x:=y+z }
Thread 2 { y:=x*z }
```

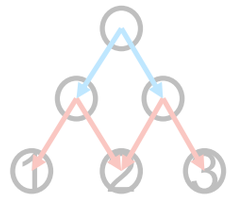
PRISM modelling language specification

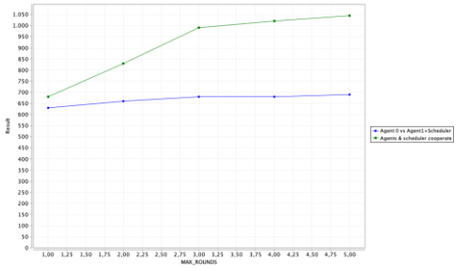
PRISM-model specification

```
module M1
…
endmodule

module M2
…
```

```
…
33 0 6 1 action1
33 1 0 1 action2
…
```

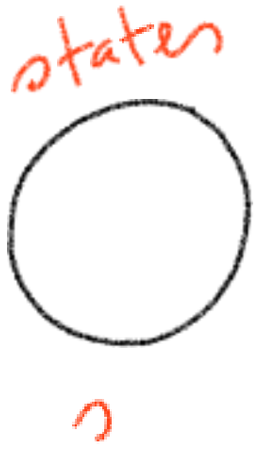


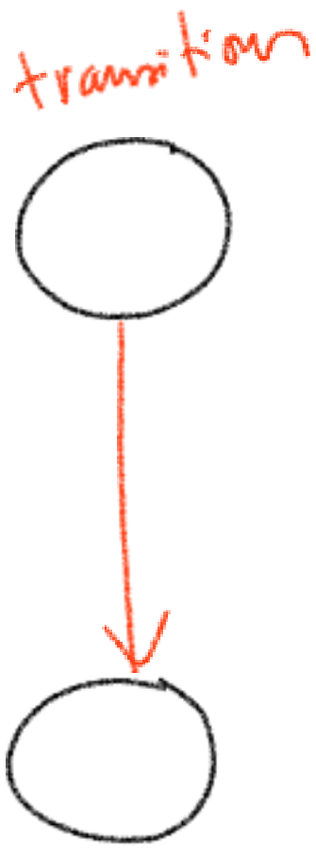


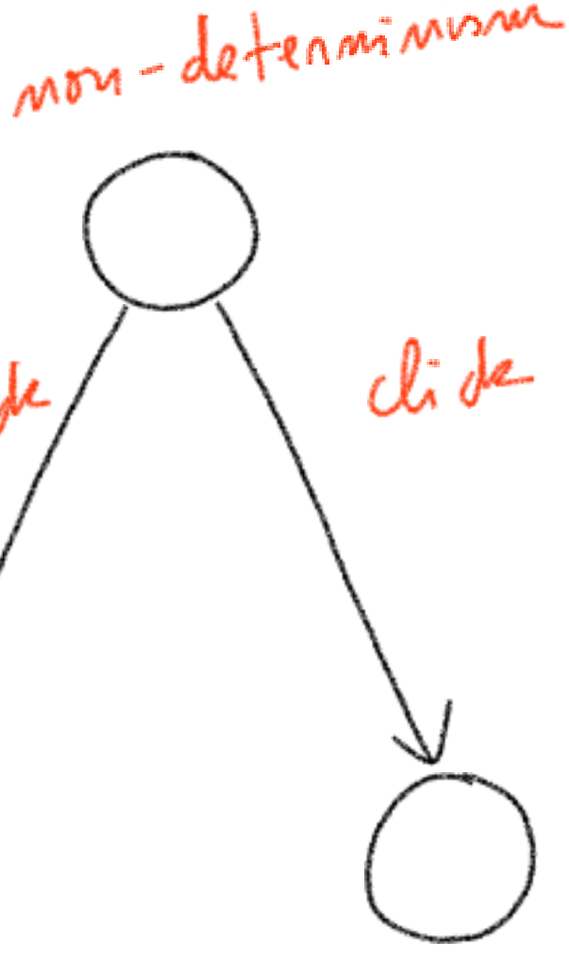
ANALYSIS RESULTS

Yes/No, Max/Min Probability/Cost/Reward, plots, etc.

Modeling with transition systems



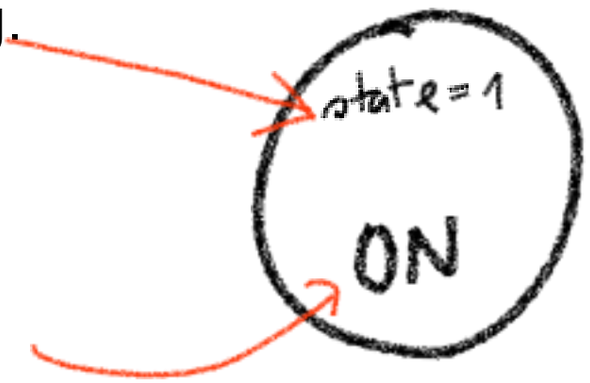




State names, state descriptions, and state labels

State names: a unique name to distinguish each individual state

State descriptions: whatever actually defines a state (e.g. the value of each variable, the content of each communication channel, etc.).



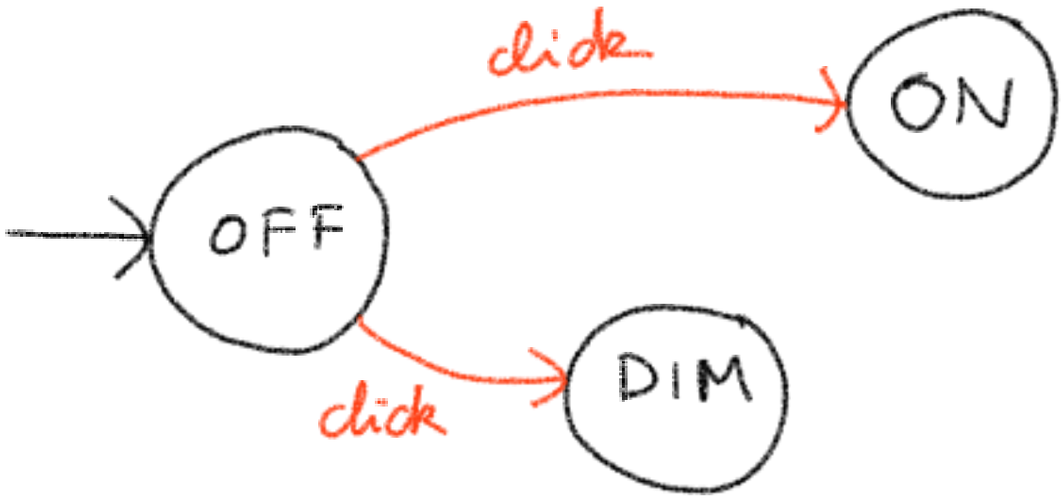
State labels: atomic properties (facts) that are true in a specific state. This is the abstract way of looking at state by focusing on its properties of interest.

Sometimes you may see these concepts mixed up due to (often unwritten) conventions… pay attention!

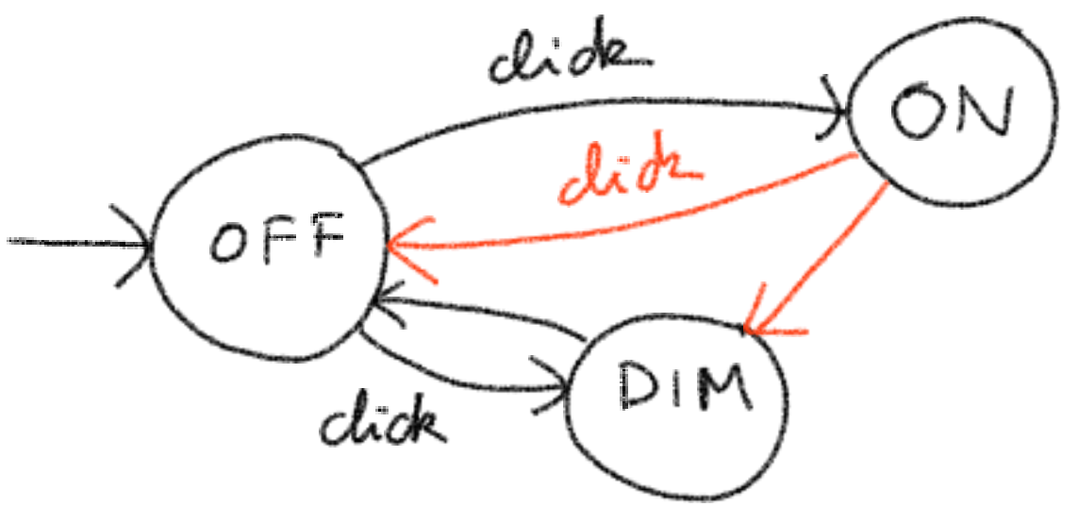
Non-determinism

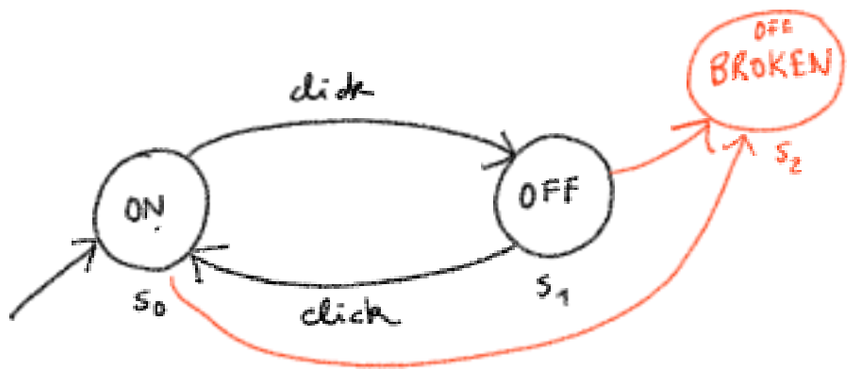
Non-determinism is a fundamental abstraction.

It may represent internal decisions of the system that are intentionally underspecified (e.g. an “if-then-else” whose logic will be decided later).



Or it may represent the uncertainty of the environment where the system will operate (users, schedulers, network, etc.).





also possible errors/malfunctions!

Deterministic transition system

A transition system is action-deterministic iff

(1) There is no more than 1 initial state

$$|I| \leq 1$$

(2) For every state ***s*** and every action ***a***, there is only one outgoing transition from s labelled with a.

$$|\{s' \mid s \xrightarrow{a} s'\}| \leq 1$$

Terminal states

A state is a terminal state if it has no outgoing transition.

We can write this formally in several ways: a state s is terminal iff

Terminal states

A state is a terminal state if it has no outgoing transition.

We can write this formally in several ways: a state s is terminal iff

$$\neg\exists s'.\, s \to s'$$

Terminal states

A state is a terminal state if it has no outgoing transition.

We can write this formally in several ways: a state s is terminal iff

$$\neg\exists s'.\, s \to s'$$

$$Post(s) = \emptyset$$

Terminal states

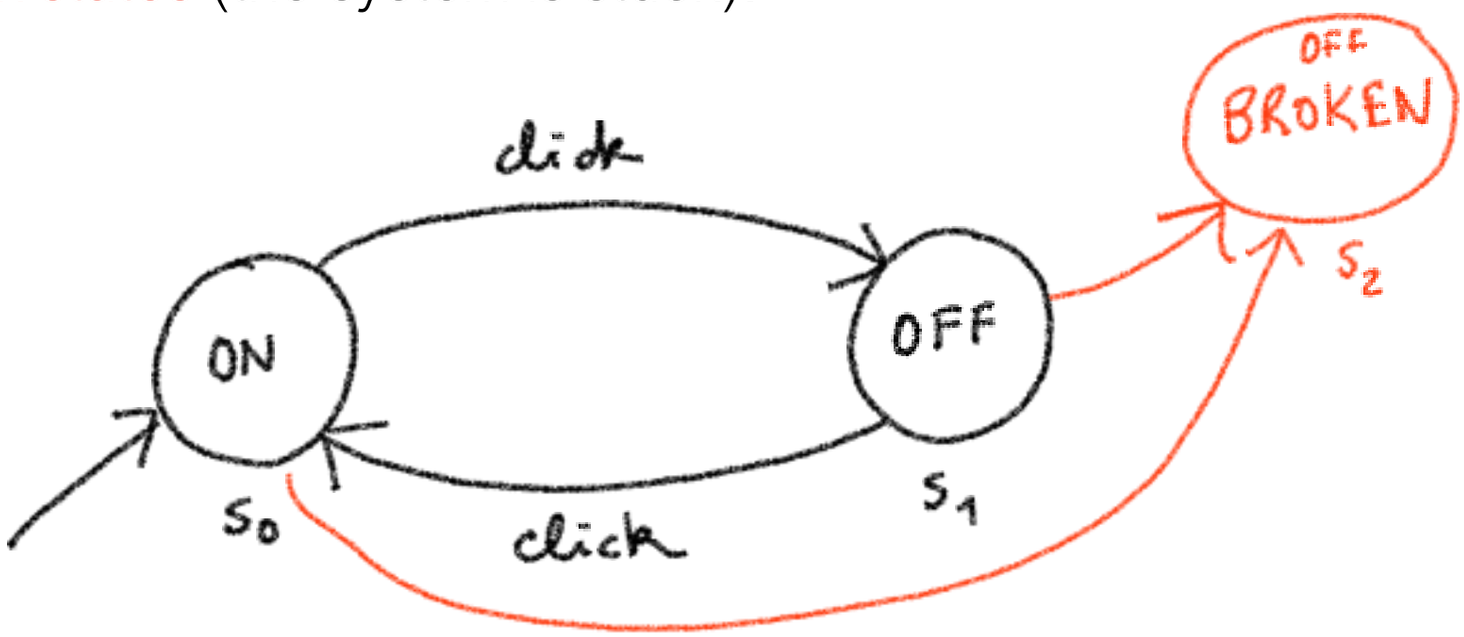
A state is a terminal state if it has no outgoing transition.

We can write this formally in several ways: a state s is terminal iff

$$\neg\exists s'.\, s \to s'$$

$$Post(s) = \emptyset$$

Terminal states usually represent final states (the system finished doing its job)
…or deadlock states (the system is stuck).

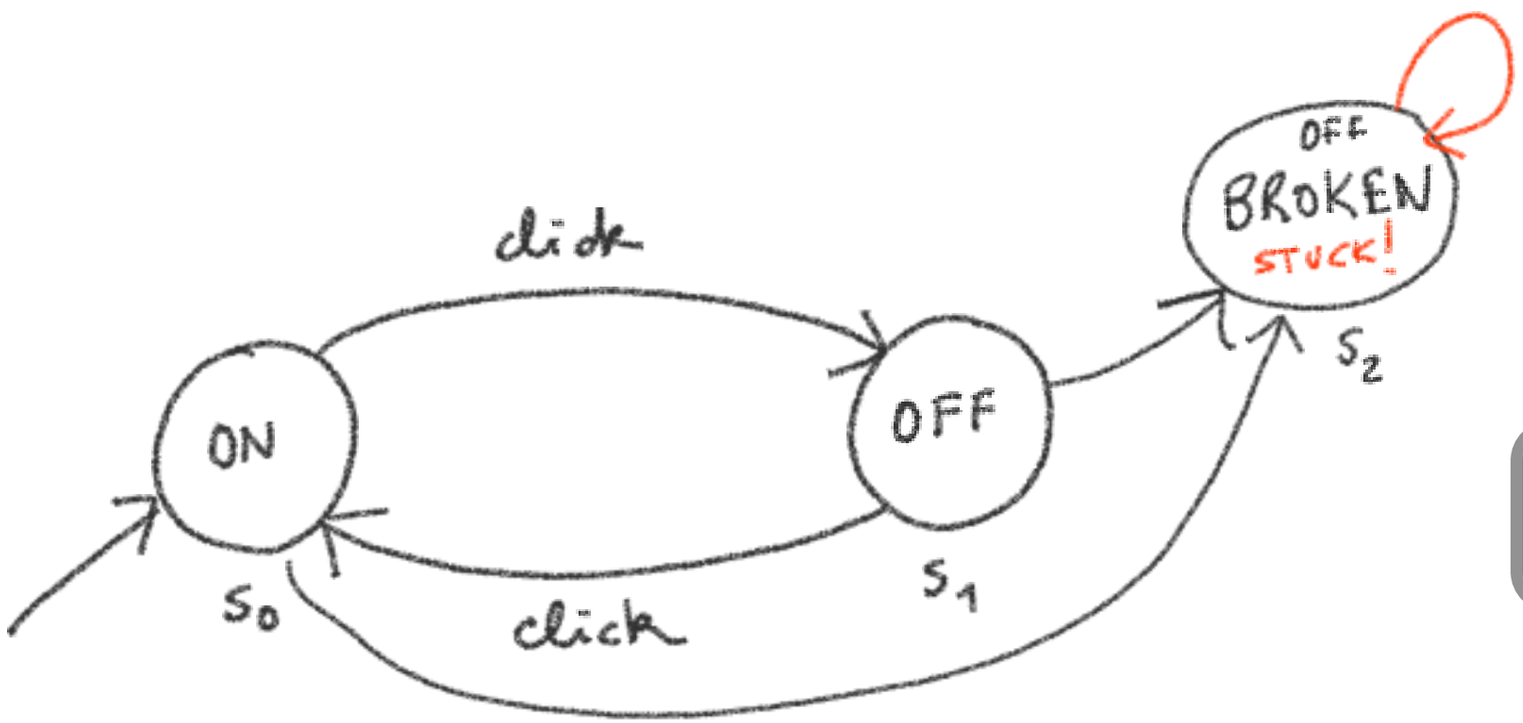


Transition systems with no terminals

From now on we consider w.l.o.g. transition systems with no terminal states.

If you have a terminal state, just add a self-loop.

If you are interested in marking the state as “terminal” and distinguish it from proper self-loops, just add as special label to it.

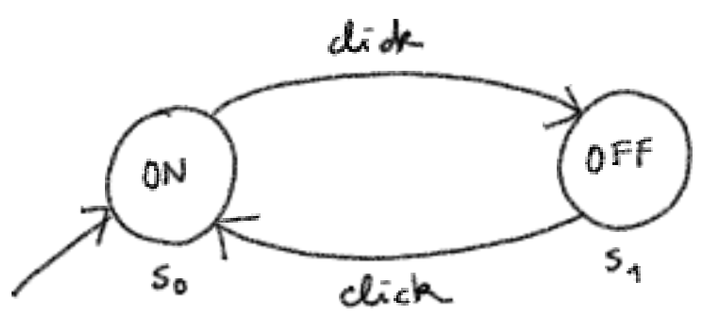


DEMO?

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

Executions



An **execution fragment** is a sequence of transitions.

$$s_0 \xrightarrow{click} s_1 \xrightarrow{click} s_0 \xrightarrow{click} \dots$$

An **execution is** finite/infinite if the sequence is finite/infinite.

$$s_0 \xrightarrow{click} s_1 \xrightarrow{click} s_0 \qquad (s_0 \xrightarrow{click} s_1 \xrightarrow{click})^{\omega}$$

An **execution is** initial if the first state of the sequence is in *I*.

$$s_0 \xrightarrow{click} s_1 \xrightarrow{click} s_0 \to \cdots \qquad s_1 \xrightarrow{click} s_0 \xrightarrow{click} s_1 \xrightarrow{click} \dots$$

An **execution is** maximal if it cannot be extended: either it is finite and the last state is a terminal state, or it is infinite.

NOTE: by our assumption of "no terminal state", only the second case applies.

NOTE: often, we drop the arrows.

Traces

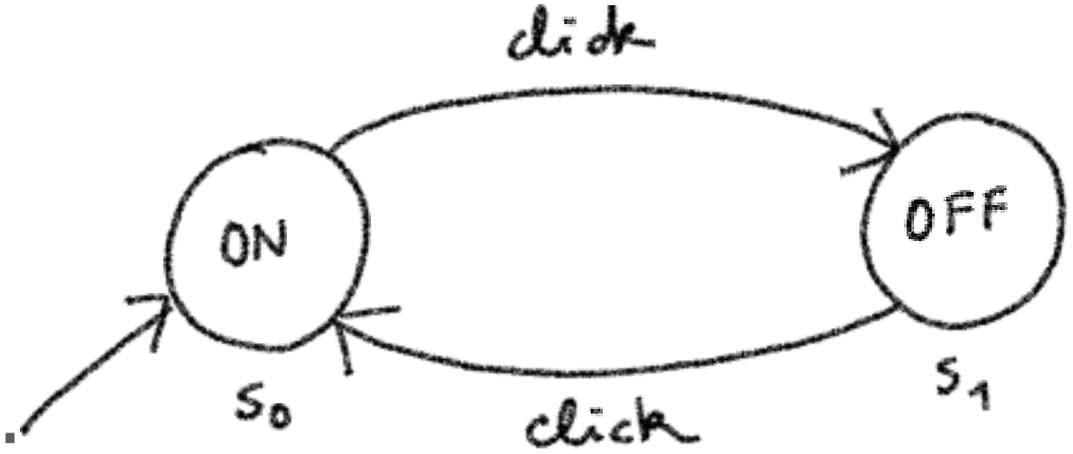
Executions may contain too much information, i.e. actual states.

Often, one is interested in the atomic properties of states.

Replacing every state in an execution by its atomic properties yields a trace.

$$s_0 \xrightarrow{click} s_1 \xrightarrow{click} s_0 \rightarrow \cdots$$

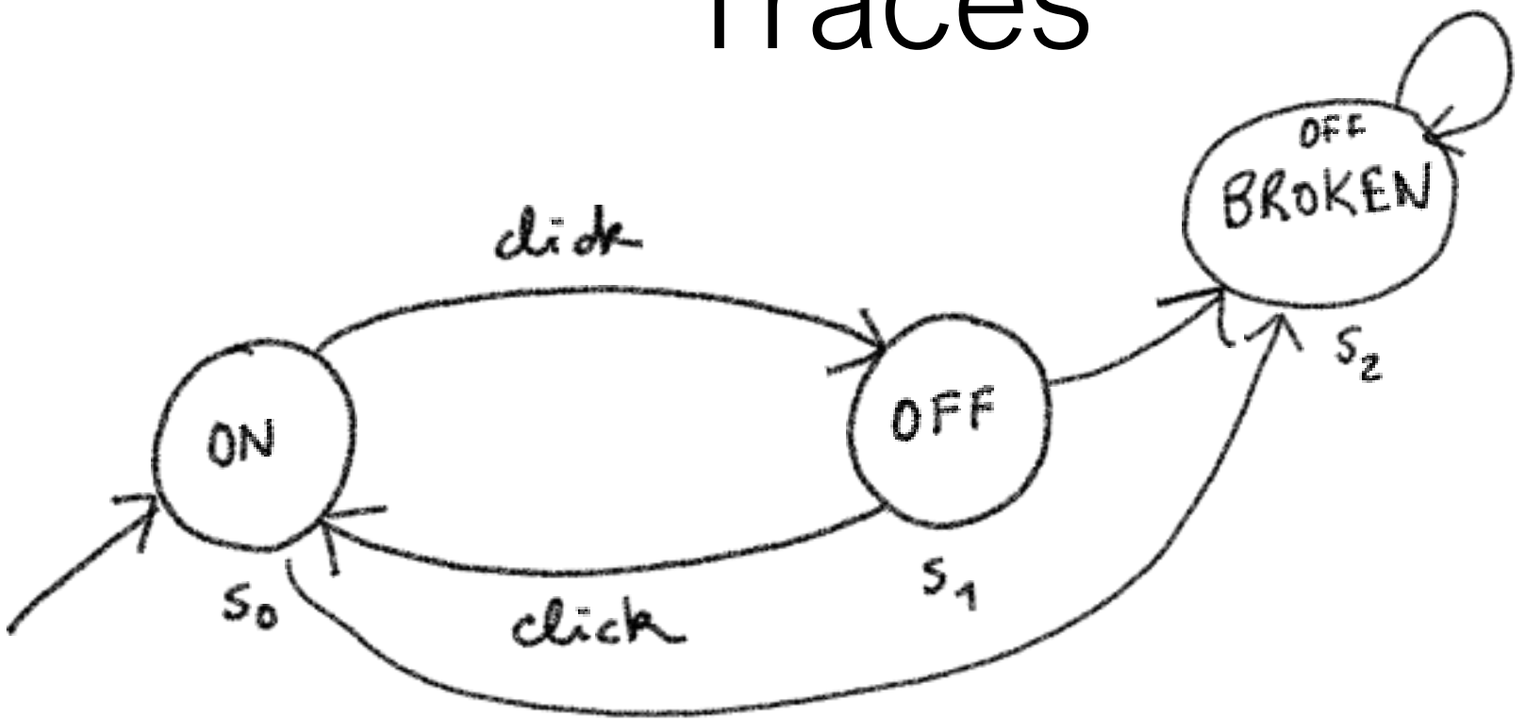
$$ON \xrightarrow{click} OFF \xrightarrow{click} ON \xrightarrow{click} \ldots$$

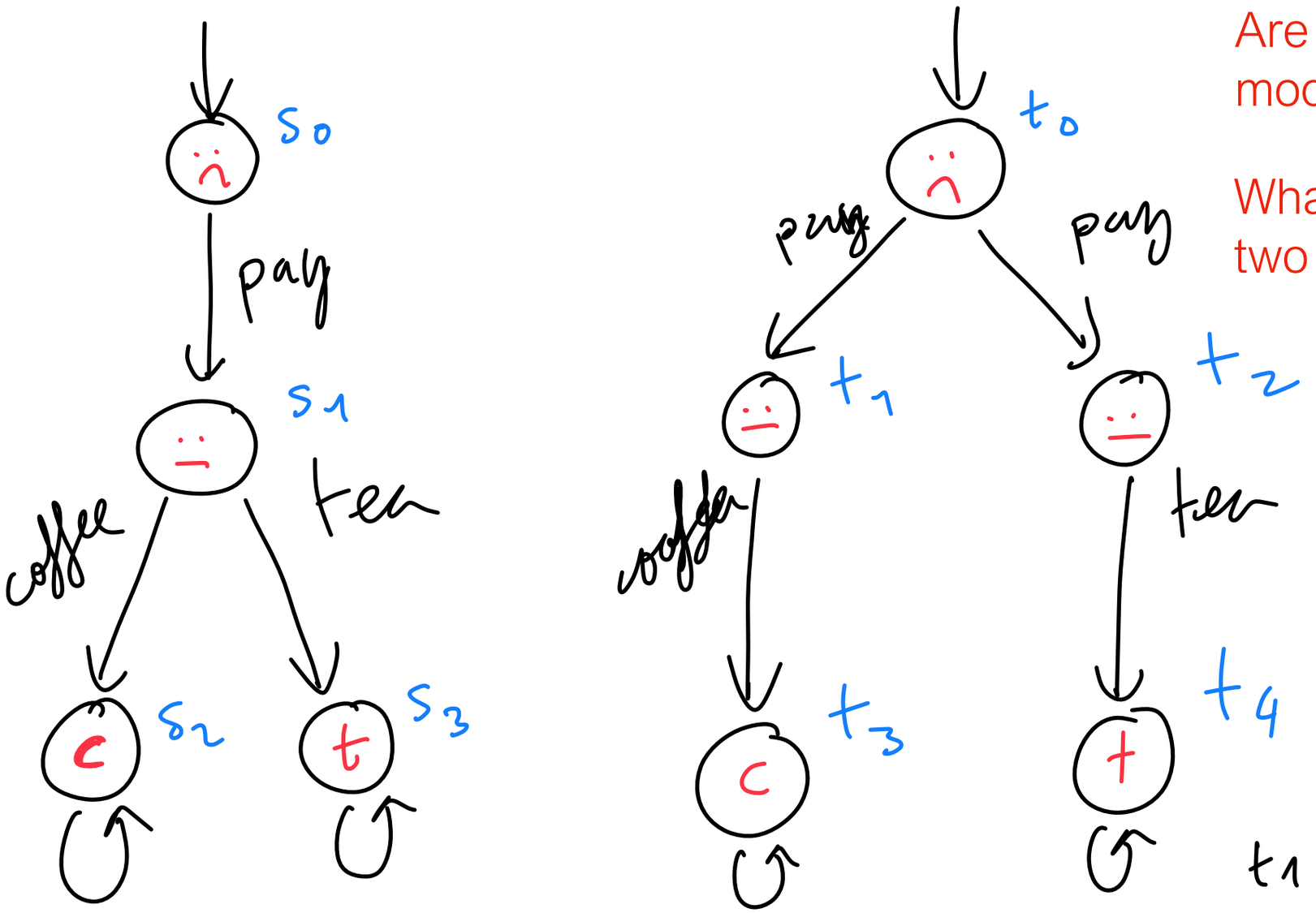


We may even drop the transitions and their actions

$$ON, OFF, ON, \ldots$$

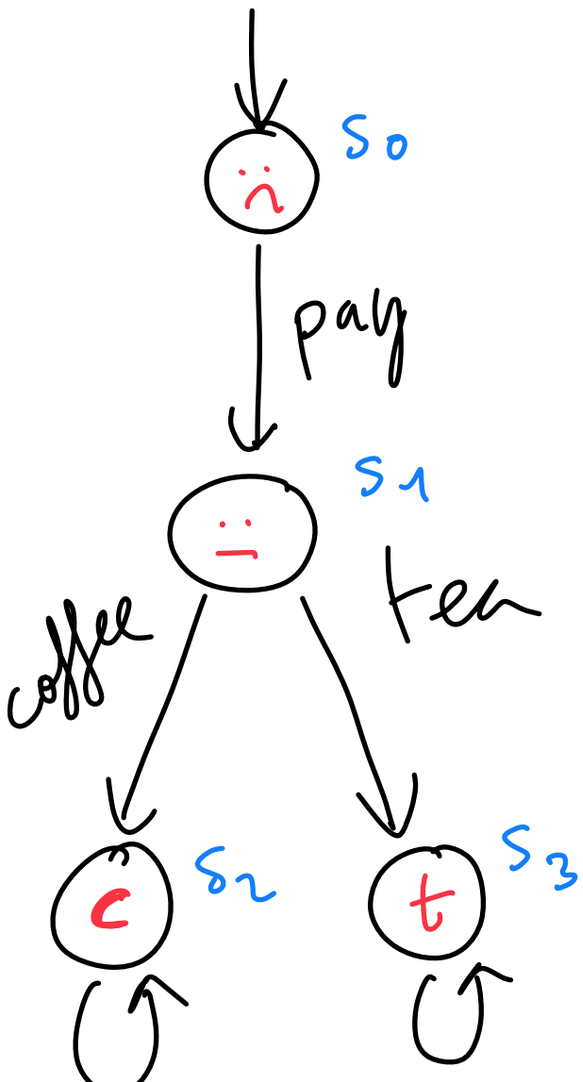
Traces

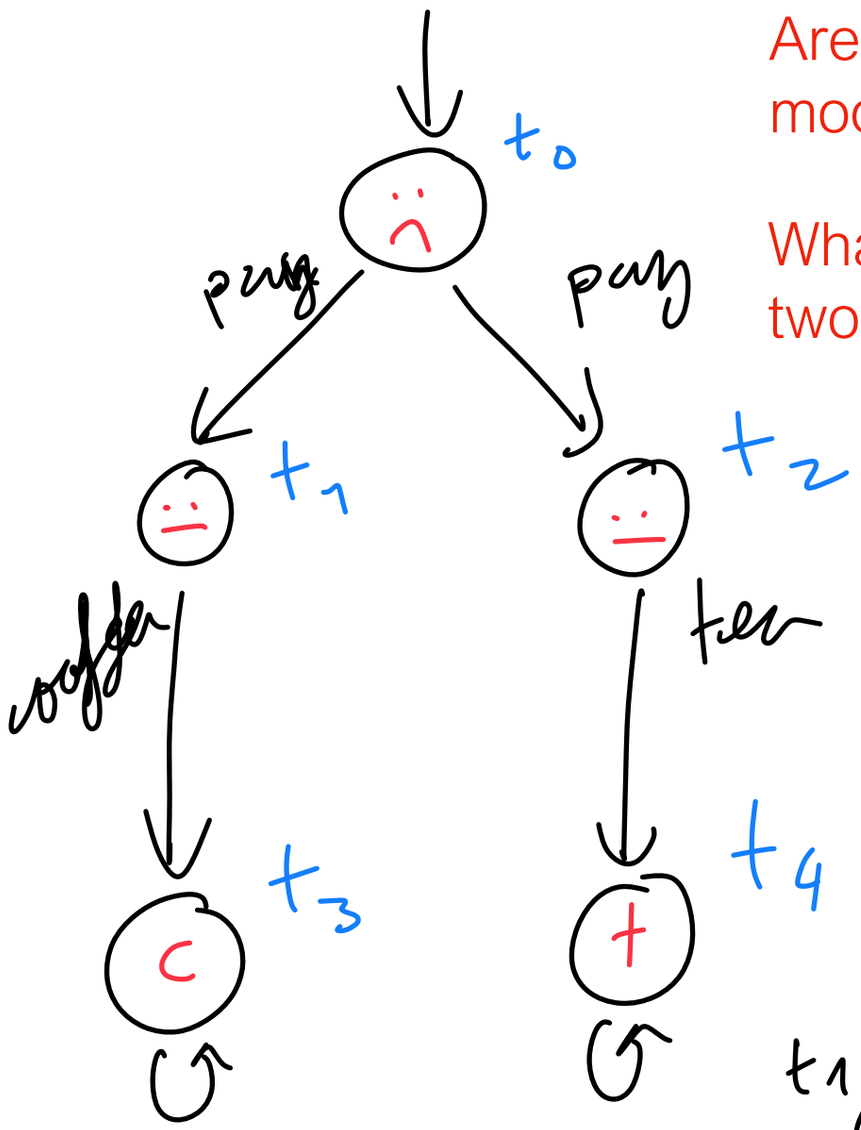




Are these two TSs modelling the same thing?

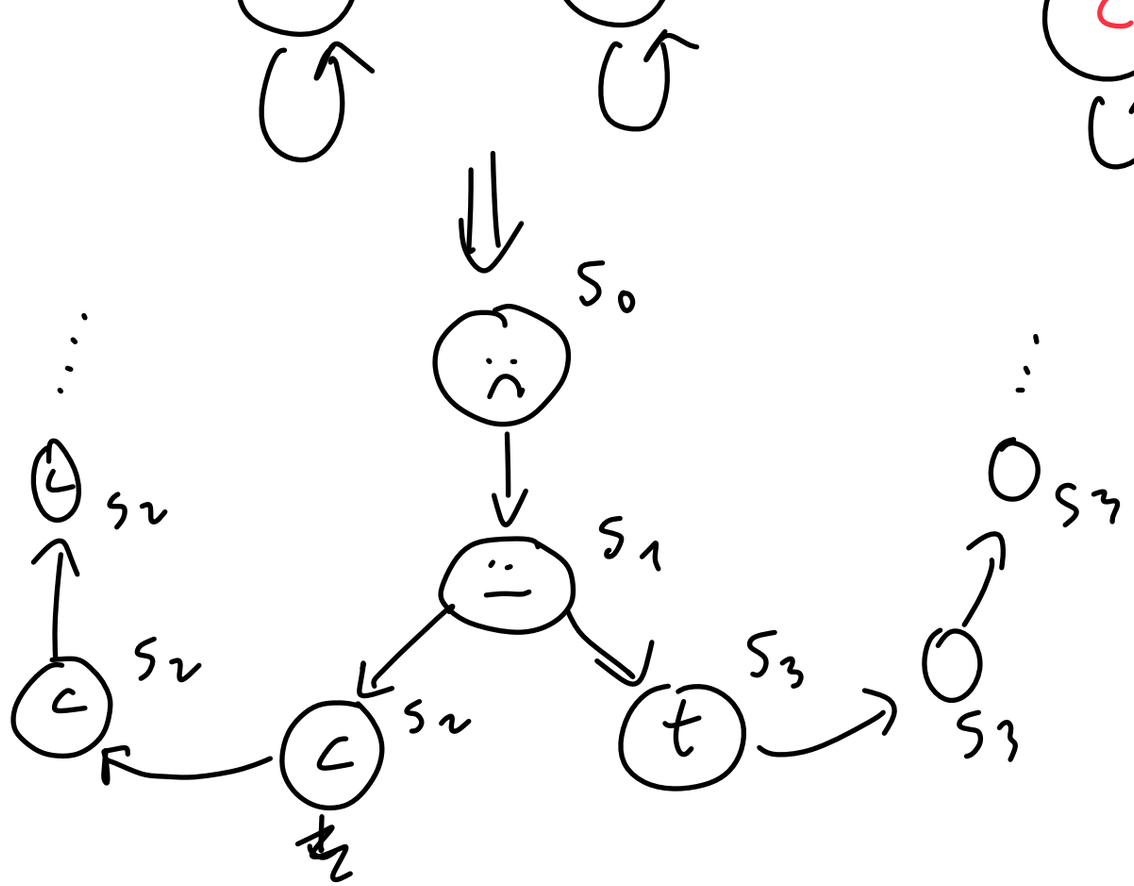
What are the traces in these two TSs?

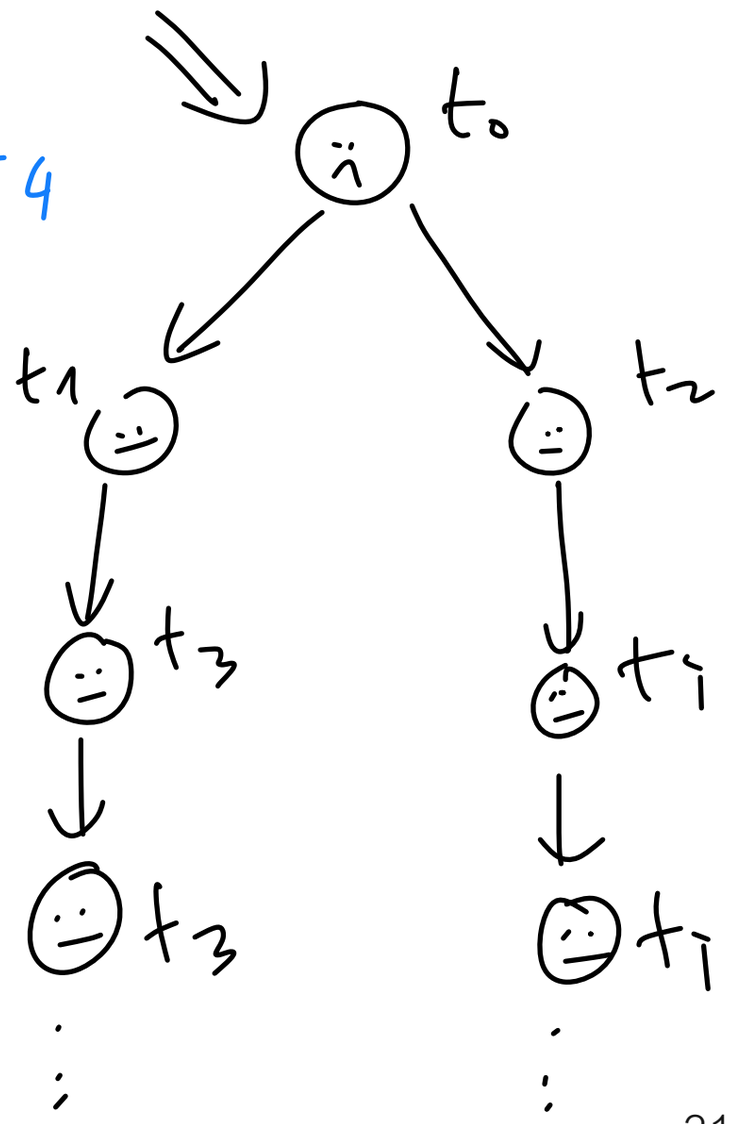




Are these two TSs modelling the same thing?

What are the traces in these two TSs?





Computation trees

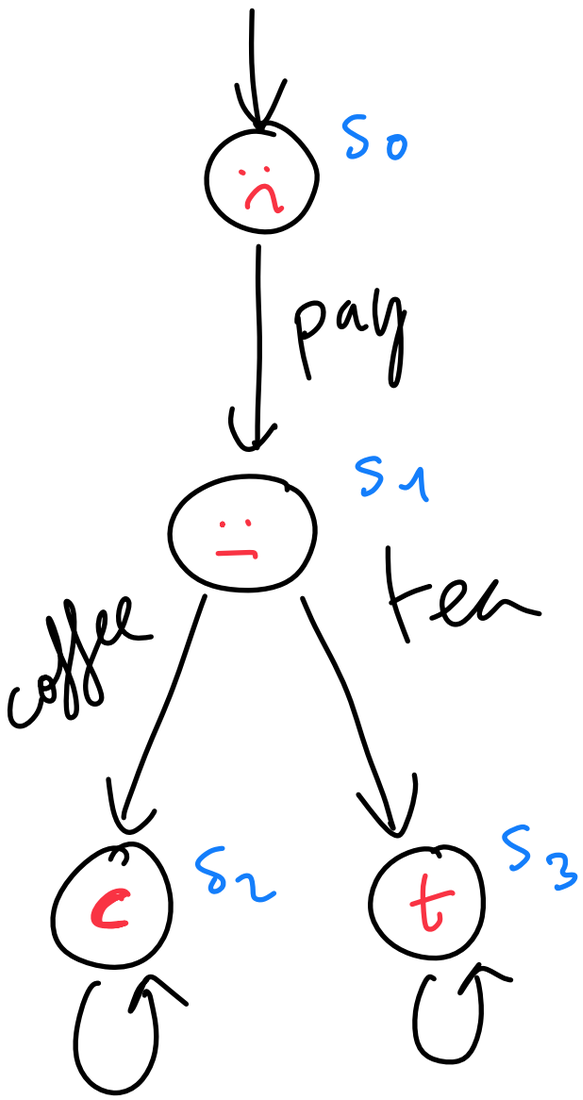
Executions and traces can be seen as the system running with a fixed scheduler.

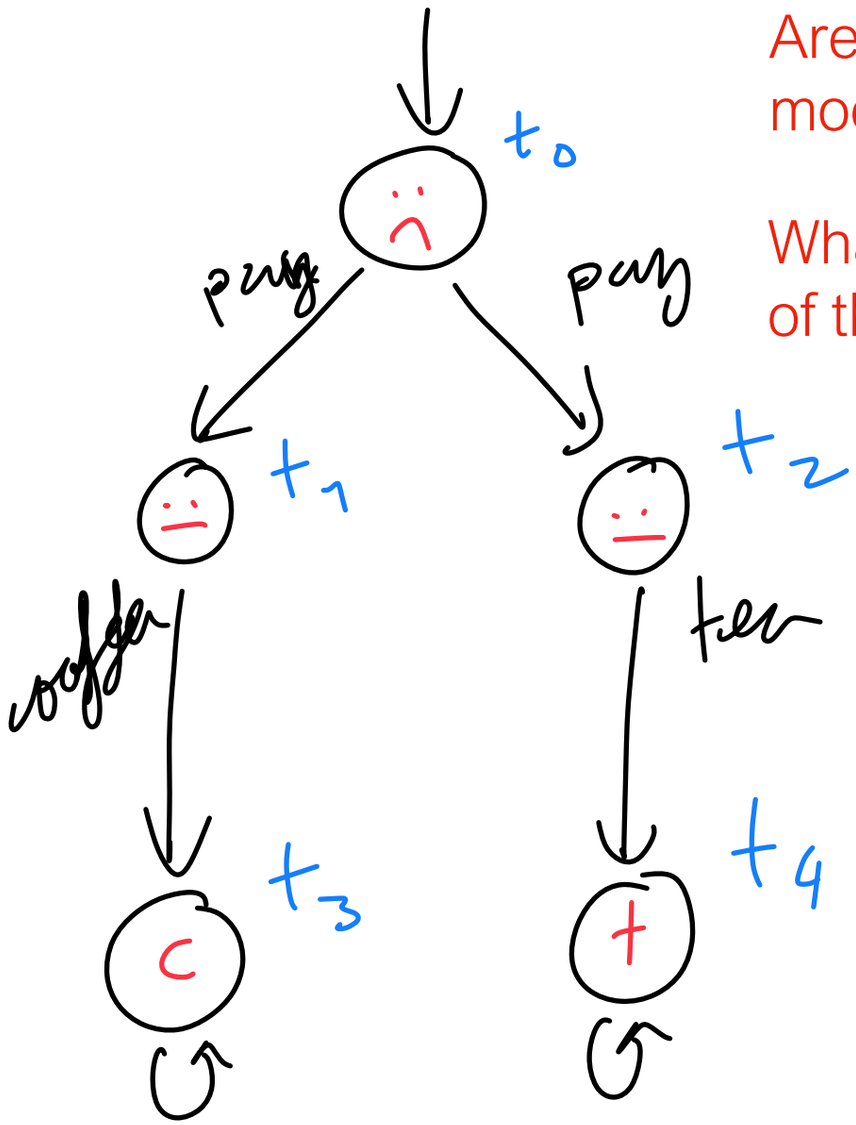
They are not appropriate if we want to see how the system makes choices.

Computation trees can save the day!

A computation tree is a tree whose nodes are states of a TS (or their atomic propositions).
The successor of each state in the computation tree is the immediate successor of the state as it appears in the TS.

A computation tree is the unfolding of the transition system.

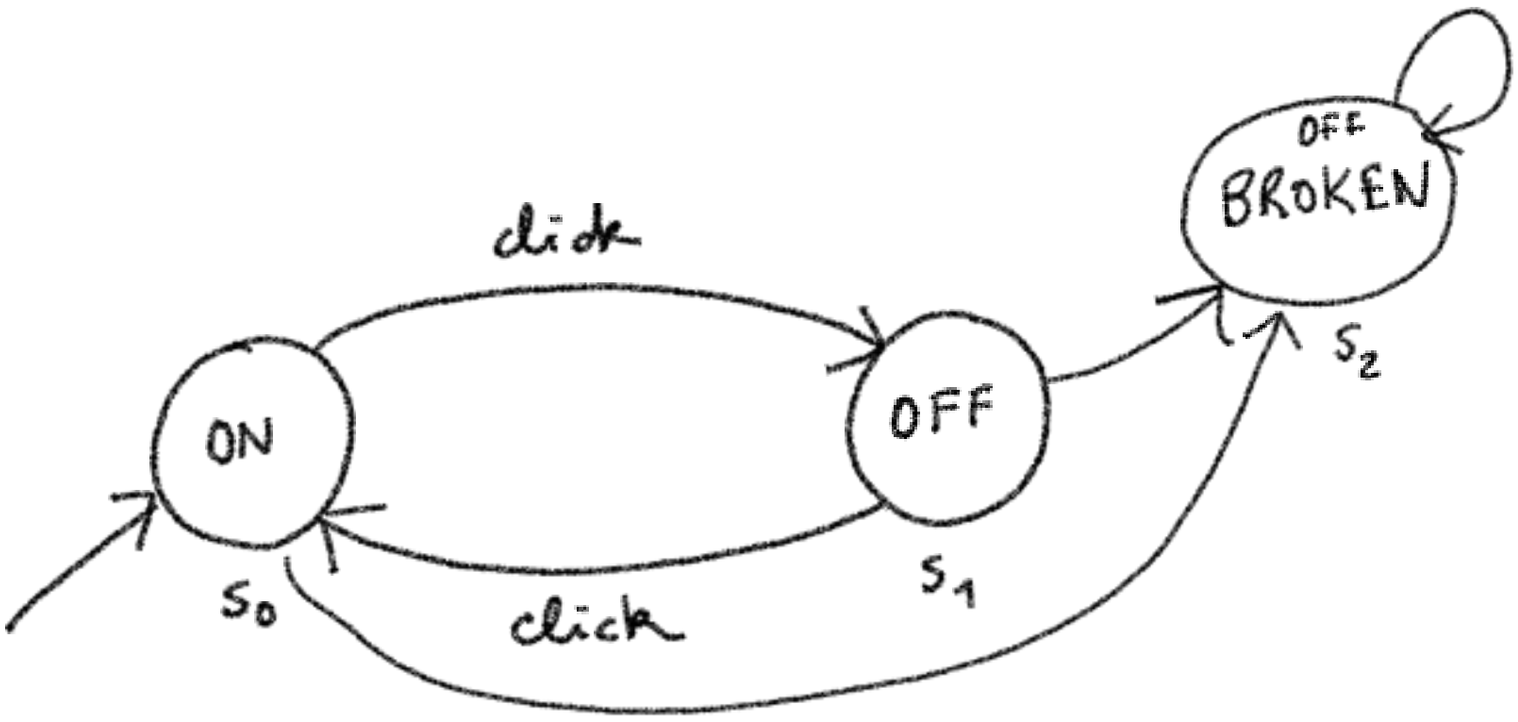




Are these two TSs modelling the same thing?

What computations trees of these two TSs?

Computation trees

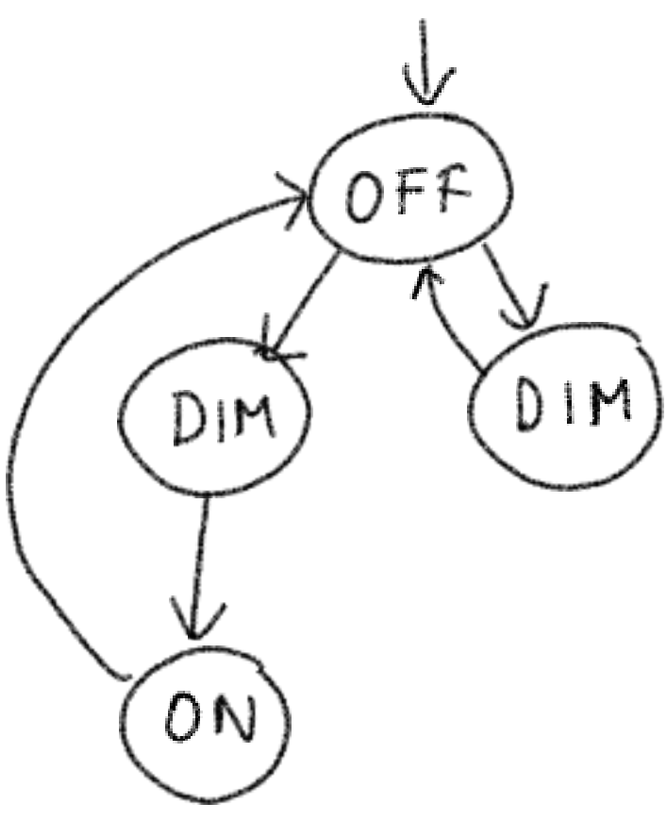


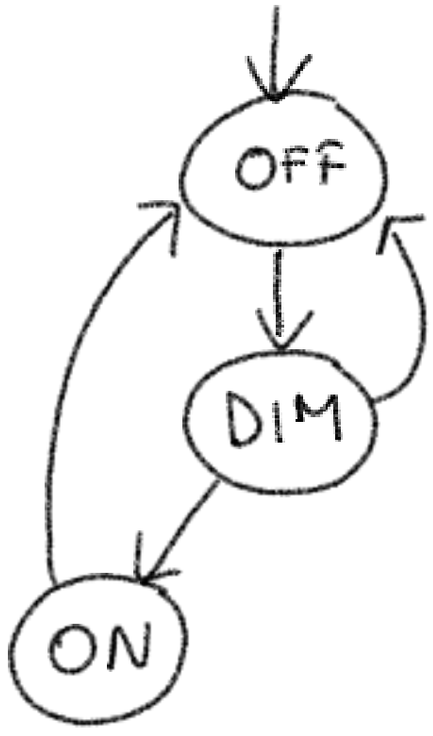
Let’s draw the computation tree…

Traces vs computation trees

Computation trees contain more information than traces.

Indeed, all (initial) traces of the system are encoded in its (initial) computation tree(s).





Let’s check whether the above transition systems have the same traces and computation trees.

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

System interactions

Often, systems (and their models) are made of several components, which interact with each other.

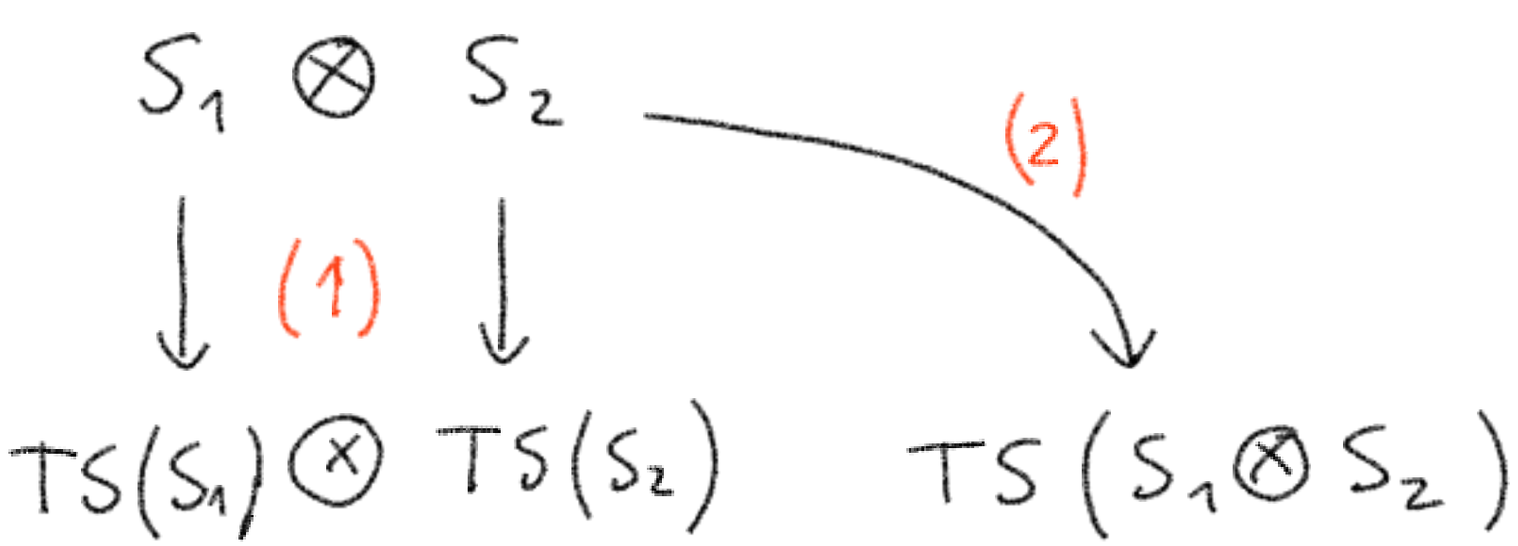
There is a plethora of interaction mechanisms:

- Shared memory / objects / storage (consistent, weak, safe, etc.)
- Networks (synchronous/asynchronous, binary/multi-party, value-passing/rendezvous, etc.)

Composing systems or transition systems?

The transition system of the composition S1⊗S2 of two systems S1, S2 can be obtained in two ways.

(1) Build TS(S1) and TS(S2), then compose
(2) Build the composed TS directly from S1,S2

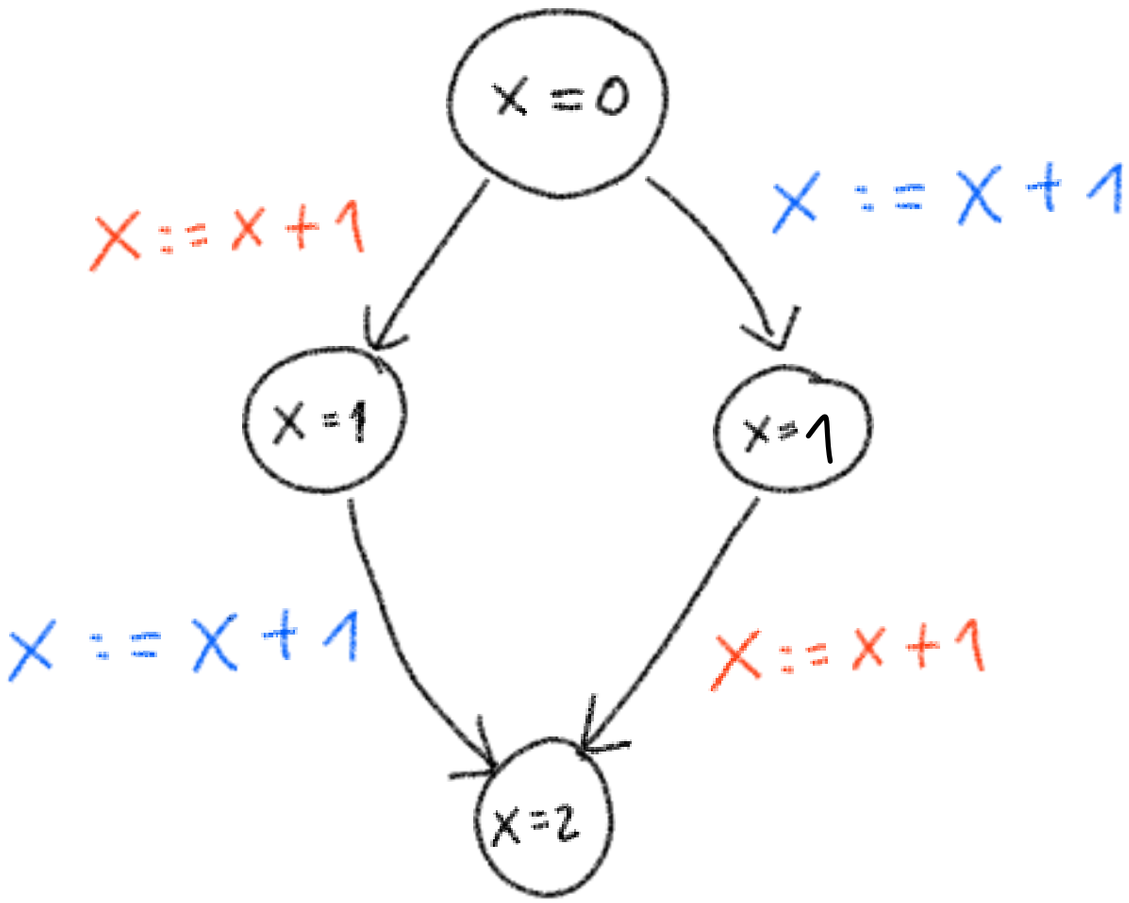


We will see
2 examples of (1): pure interleaving and action-synchronisation
and 1 example of (2): concurrent threads with shared memory

Interleaving composition of concurrent threads with shared memory

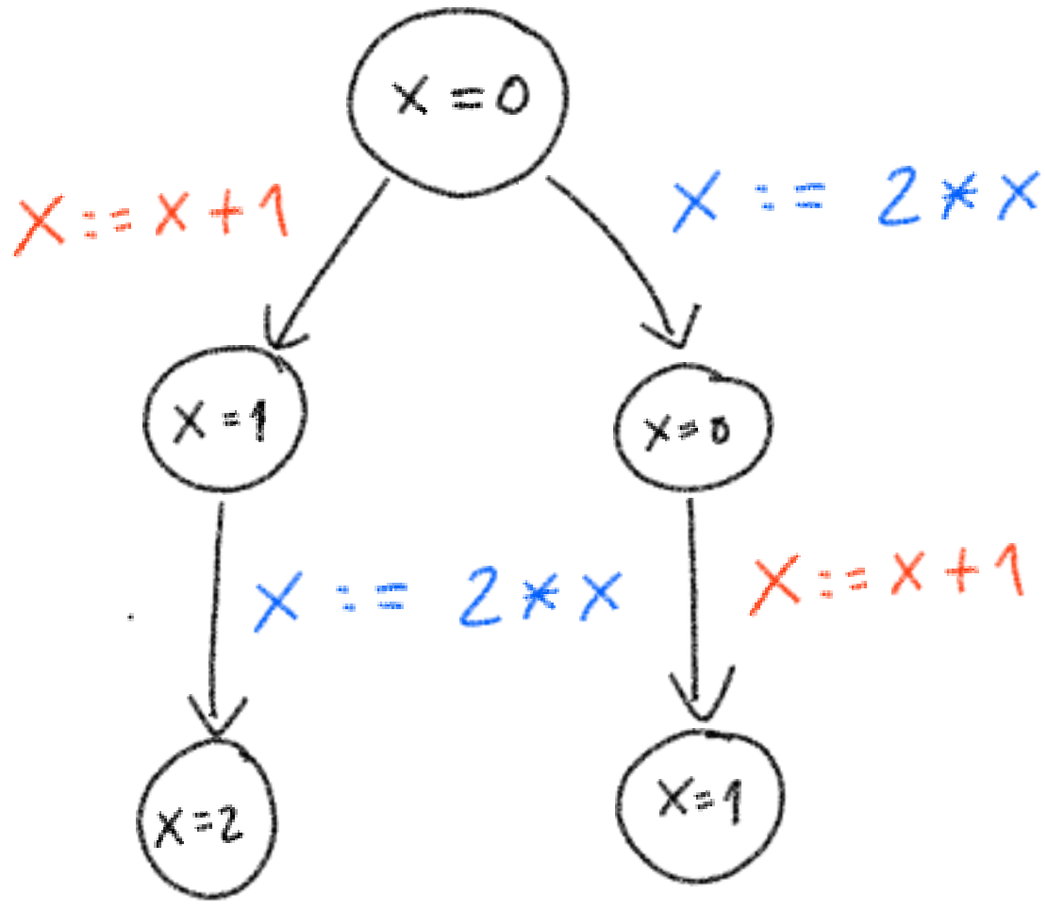
Thread 1
`x:=x+1`

Thread 2
`x:=x+1`



Thread 1
`x:=x+1`

Thread 2
`x:=2*x`



Formal semantics of the composition typically specified using operational semantics (common in programming languages).

Interleaving composition of transition systems

Interleaving composition of T1 and T2 is defined as follows

$$T_1 = \langle S_1, A_1, \rightarrow_1, L_1, AP_1, I_1 \rangle$$

$$T_2 = \langle S_2, A_2, \rightarrow_2 L_2, AP_2, I_2 \rangle$$

$$T_1 \,|||\, T_2 = \langle S_1 \times S_2, A_1 \cup A_2, \rightarrow, L, AP_1 \cup AP_2, I_1 \times I_2 \rangle$$

where the key point is the new transition function

$$\frac{s_1 \xrightarrow{\alpha} s_1'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha} s_2'}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

$$L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$$

Synchronised composition of transition systems

Synchronised composition of T1 and T2 over actions ***B***

$$T_1 = \langle S_1, A_1, \rightarrow_1, L_1, AP_1, I_1 \rangle$$

$$T_2 = \langle S_2, A_2, \rightarrow_2, L_2, AP_2, I_2 \rangle$$

$$T_1 \,|B|\, T_2 = \langle S_1 \times S_2, A_1 \cup A_2, \rightarrow, L, AP_1 \cup AP_2, I_1 \times I_2 \rangle$$

$$L(\langle s_1, s_2 \rangle) = L_1(s_1) \cup L_2(s_2)$$

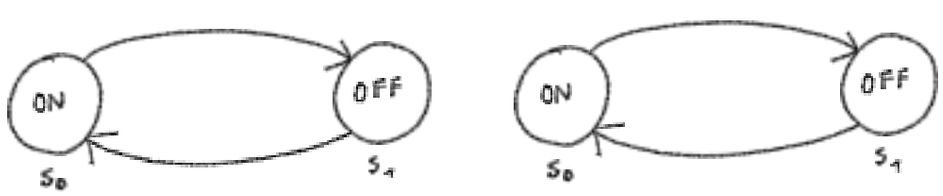
again, the key point is the new transition function (see last rule)

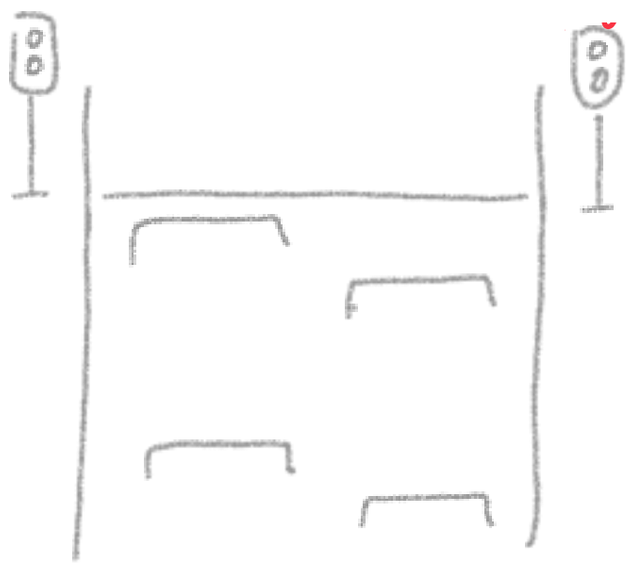
$$\frac{s_1 \xrightarrow{\alpha} s_1' \quad \alpha \notin B}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2 \rangle} \qquad \frac{s_2 \xrightarrow{\alpha} s_2' \quad \alpha \notin B}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s_2' \rangle}$$

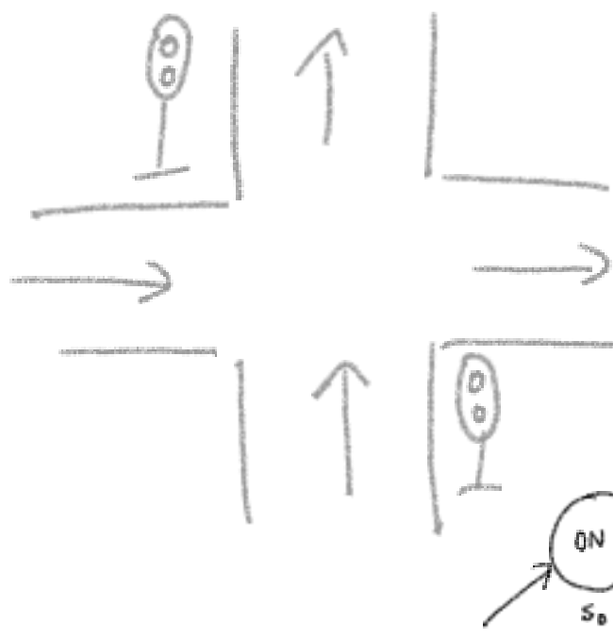
$$\frac{s_1 \xrightarrow{\alpha} s_1' \qquad s_2 \xrightarrow{\alpha} s_2' \quad \alpha \in B}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1', s_2' \rangle}$$

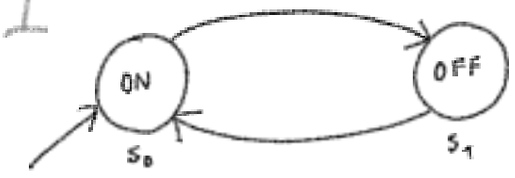
System interactions, an example

How would you synchronise the actions of these traffic light controllers?





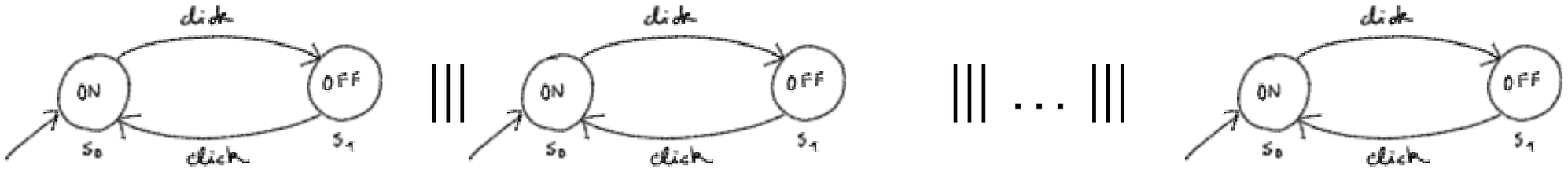




DEMO?

State space explosion

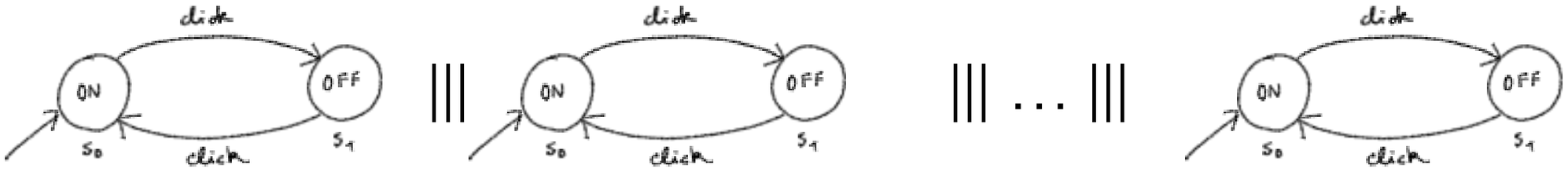
How many states has this transition system?



In general, the size of the interleaving of n transition systems of m states each is…

State space explosion

How many states has this transition system?



In general, the size of the interleaving of n transition systems of m states each is…m^n

Synchronizations may reduce the size composition but the worst-case is still exponential in the number of components.

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- Exercises & Homework

Key points of this lecture

Definition of transition systems and related concepts (successors, predecessors, etc.).

Definition of executions, traces and computation trees, and related concepts (finite, maximal, etc.).

Concept of non-determinism as powerful modelling abstraction and its formal definition.

Interaction and composition of systems and transition systems and the state space explosion problem.

How the above concepts map to PRISM.

Lecture 01 - Transitions Systems

- Reading Material
- What are Transition Systems?
- Modelling with Transition Systems
- Semantics of Transition Systems
- Composing Transition Systems
- PRISM: a Tool for Transition Systems (and more)
- Exercises & Homework

Getting Ready Today

1. Form groups for the assignment and for working on the exercises and register them on DTU Learn
2. Get PRISM (http://www.prismmodelchecker.org/) up and running on your laptop.
3. Run the PRISM tutorial Part 1 (http://www.prismmodelchecker.org/tutorial) until and including section “Model checking with PRISM” to get a feeling for what PRISM can do. You are not expected to understand all the concepts in the tutorial; we will cover them during the course. You can also navigate the manual (http://www.prismmodelchecker.org/manual/Main/Welcome) and play a bit with the tool.

Exercises

Do the following exercise

Exercise 01.1
Exercise 01.2

NOTE: the exercise is described at the end of the slide deck.

APPENDIX: Exercises

Exercise 01.1

Consider the following PRISM specification

```
mdp

// Shorthands for the states of Alice
const s = 0; // Susceptible
const i = 1; // Infected
const r = 2; // Recovered

// Module for Alice (potential victim of the virus)
module Alice

// state of Alice
state : [s..r] init 0;

// Transitions
[      ] (state = s) -> (state' = s) ;
[infect] (state = s) -> (state' = s) ;
[infect] (state = s) -> (state' = i) ;
[      ] (state = i) -> (state' = i) ;
[      ] (state = i) -> (state' = r) ;
[      ] (state = r) -> (state' = r) ;

endmodule

// Module for the virus
module Virus

// Virus charge
charge : [0..1] init 1;

// Virus transitions, every infection attempt reduces the charge
[infect] (charge > 0) -> (charge' = charge-1) ;

endmodule

system
  Alice || Virus
endsystem

label "SAFE" = (state = r | (state != i & charge = 0));
```

Exercise 01.1

… and answer the following questions:

(a) Depict the entire transition system and highlight the reachable part.
(b) How many states does the entire transition system have?
(c) How many states are reachable from the initial state?
(d) Write formally the (reachable) transition system.
(e) Is there non-determinism? If yes, what does it model?
(f) Are there terminal states? If yes, replace them by self-loops.
(g) Write down all initial executions of length 3.
(h) Write down the computation tree up to depth 3.

NOTE: For some of the above exercises above you can use PRISM to double-check your solutions, but try first “by-hand”.

Exercise 01.2

Build the transition systems that result from the following combinations