Models for Computation (Chapter 2)

The standard model of Computation is a state machine, a.k.a. automaton, transition system, discrete transition system, Kripke structure

You have probably seen a finite state machine:

Finite Automaton

- States
- Start state
- Transitions

Generalization of FSMs

- Arbitrary number of states
- Variables implicity define states

Variables are the atoms or building blocks for models

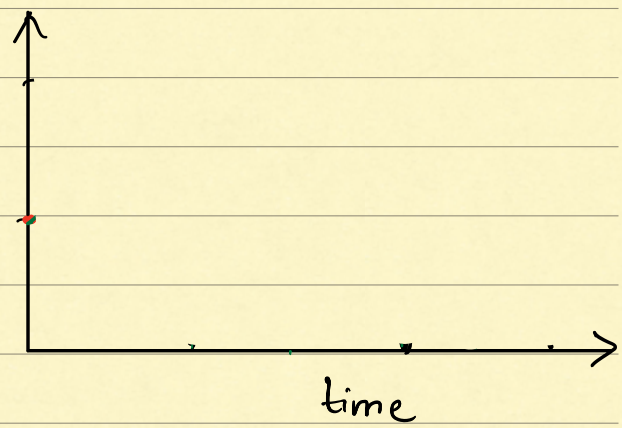
Once we have variables, we can use programs to define the transitions.

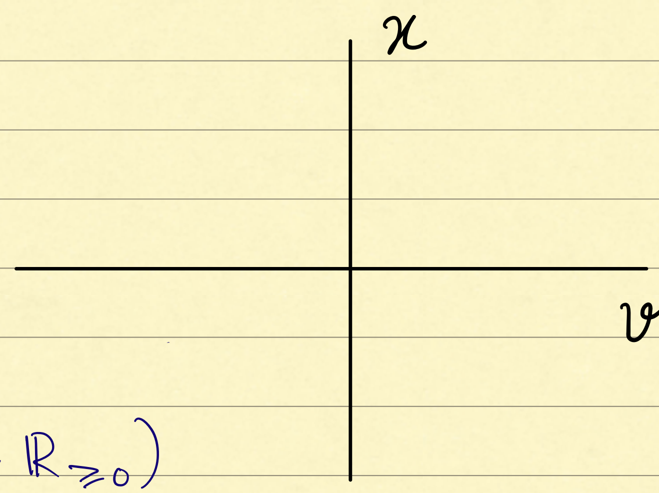
Example 1

Let us write the model of a ball bouncing on the floor.

$x : \mathbb{R} := h$

$v : \mathbb{R} := 0$





automaton $(g, h, \Delta : \mathbb{R}_{\geq 0})$

variables $x : \mathbb{R} := h$

$v : \mathbb{R} := 0$

transitions

Tick

Pre. $v < 0 \;\&\&\; x \geq 0$

Eff. $v := v - g\Delta t$

$x := x + v\Delta t$

Bounce

Pre. $x \leq 0 \;\&\&\; v < 0$

Eff. $x := 0$

$v := -v$

Variables, valuations

A set of names (identifiers) and <u>types</u>

E.g $V = \{x, v\}$

type(x) = other types

type(v) =

A <u>valuation</u> for V maps each $v \in V$ to a <u>value</u> in type(v).

E.g. A valuation for

Given a valuation ν of V a <u>restriction</u> of ν to a particular variable $x \in V$ is written as $\nu \lceil x$.

E.g. $\nu \lceil x =$ $\nu' \lceil v =$

Set of all possible valuations of V is denoted by val(V)

E.g. Val(V) = $\approx \mathbb{R} \times \mathbb{R}$

Automata

Def. An automaton is 4 tuple $\mathcal{A} = \langle V, \Theta, A, \mathcal{D} \rangle$

- V is a set of variables; $val(V)$ state space
- $\Theta \subseteq val(V)$ set of initial values or states
- A is a set of action names
- $\mathcal{D} \subseteq val(V) \times A \times val(V)$ set of labeled transitions

---

if $\langle v, a, v' \rangle \in \mathcal{D}$ we say that there is a valid transition from v to v' by performing action a.

We write this as $v \xrightarrow{a} v'$

E.g.

An action $a \in A$ is enabled at a state $v \in val(V)$ if $\exists\, v' \in val(V)$ such that $v \xrightarrow{a} v'$.

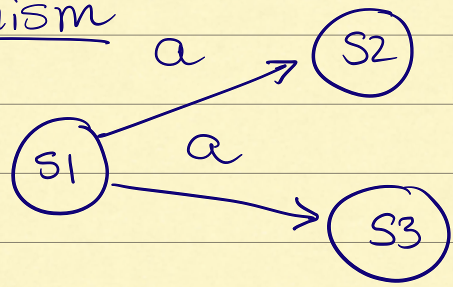
Q. When is bounce enabled?

The set of states where an action is enabled is called the guard or the precondition of the action.

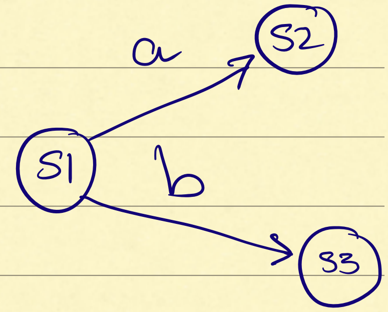
E.g Pre (bounce)

When is tick enabled?
Are they enabled at the same states?

Nondeterminism



internal



external

$\langle S1, a, S2 \rangle \in \mathcal{D}$
and $\langle S1, a, S3 \rangle \in \mathcal{D}$

An automaton is <u>determenistic</u> if from any state $v \in val(V)$ at most one action is enabled <u>and</u> the action uniquely determines the post state,

$\forall v, a_1, a_2, v_1, v_2$
if $v \xrightarrow{a_1} v_1$ and $v \xrightarrow{a_2} v_2$ then
$a_1 = a_2$ and $v_1 = v_2$.

Nondeterminism is the main mechasim for modeling uncertainty in automata.

Executions

An execution of an automaton $\mathcal{A}$ captures a particular run or behavior of $\mathcal{A}$.

An execution of $\mathcal{A}$ is an alternating sequence $\alpha = v_0 a_1 v_1 a_2 \cdots\cdots$ such that each $v_i \in val(V)$ $a_i \in A$ and

$v_i \xrightarrow{a_{i+1}} v_{i+1}$, $v_0 \in \Theta$.

$Execs_{\mathcal{A}}$: Set of all exeations of $\mathcal{A}$

$Execs_{\mathcal{A}}(\Theta)$

$Execs_{\mathcal{A}}(\Theta, k)$: Finite executions of length at most k.

For a finite execution $\alpha = v_0 a_1 \cdots\cdots v_k$ the last state (v_k) is denoted by $\alpha.lstate$.

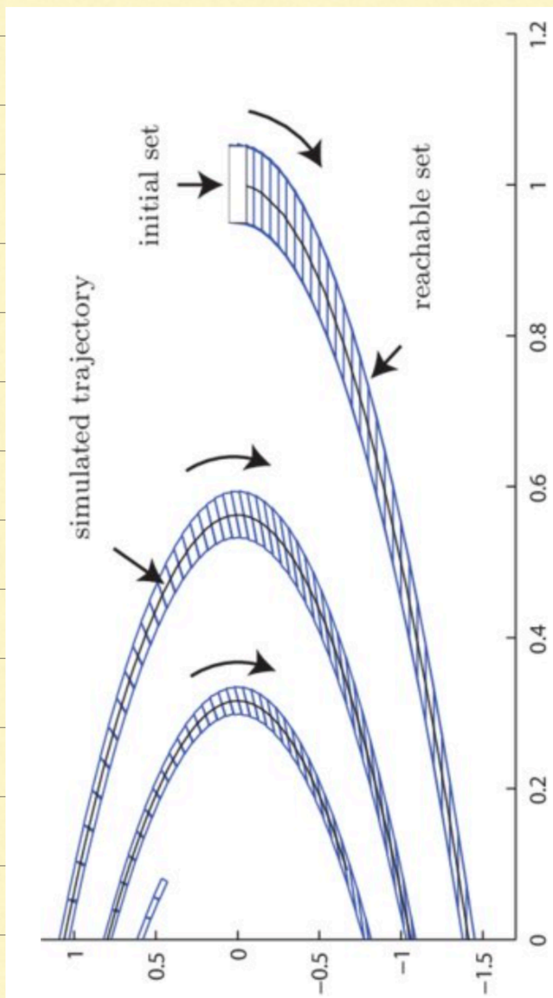
Reachable states

A state $v \in Val(V)$ is <u>reachable</u> if $\exists$ a finite execution α such that $\alpha.lstate = v$.

$Reach_{\mathcal{A}}$ Set of all reachable states of $\mathcal{A}$.

$Reach_{\mathcal{A}}(\Theta)$

$Reach_{\mathcal{A}}(\Theta, k)$



Reachable set of bouncing ball computed by CORA tool.

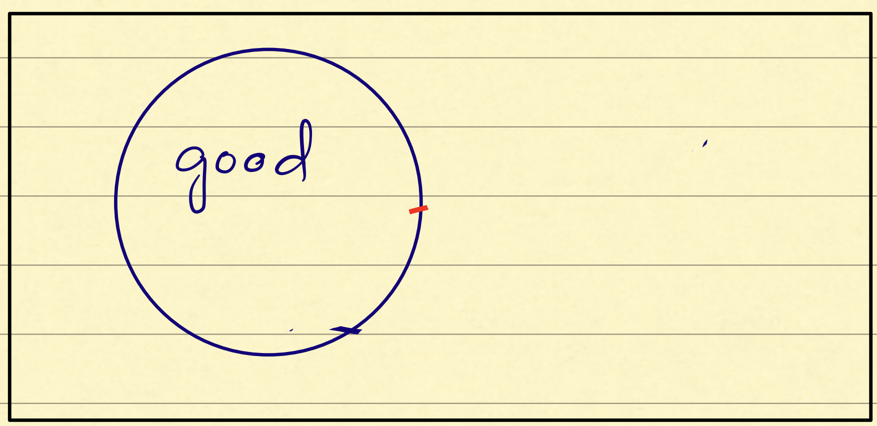
Ex2. Dijkstra's token ring algorithm

E. Dijkstra (1930 – 2002)

- Shortest path algorithm
- Dining philosophers' problem
- Structured programming
- Self Stabilization

Self-Stabilization

A system that is designed to be such that after a failure happens, it can go to arbitrary bad (illegal) states but as the system continues to run it automatically returns to a good (legal) state.



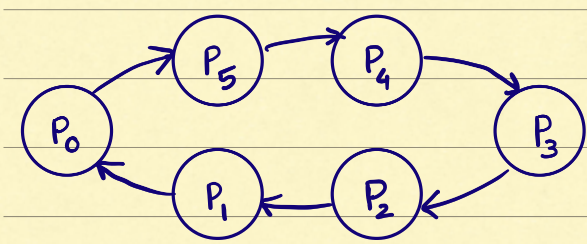
Token Ring

A distributed system in a ring topology in which only one process has the "token" at any given time.

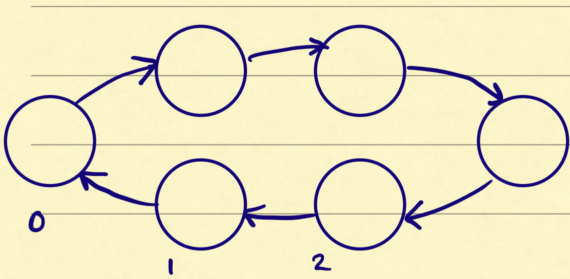
Used for

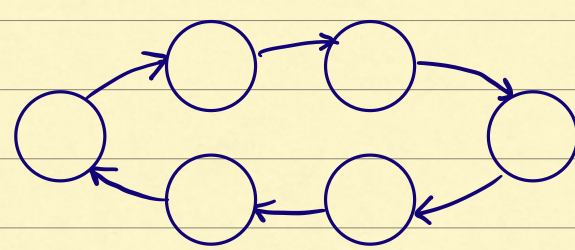
-
-

Dijkstra's token ring algorithm

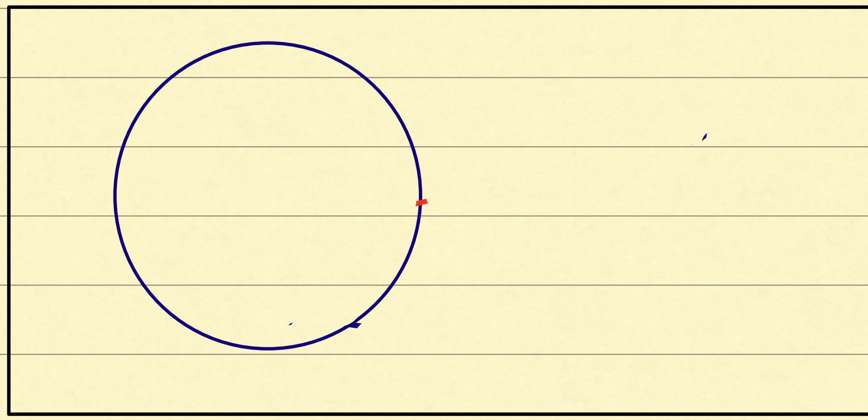


P_i <u>has token</u> if





Which processes have a token?

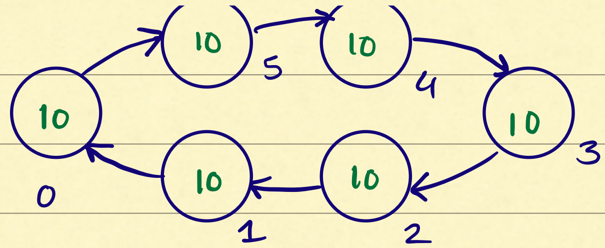


Dijkstra's Algorithm

P_i can update state only when it has a token. Update rule

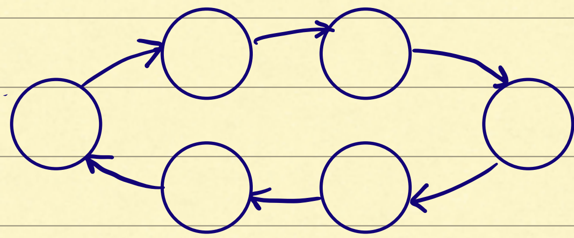
Execution of the system

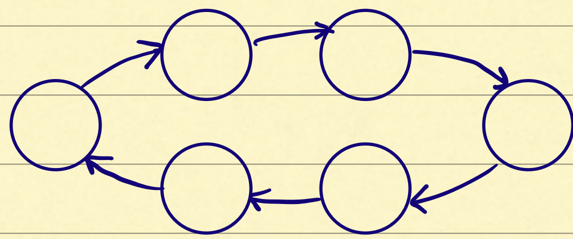
Starting from good state



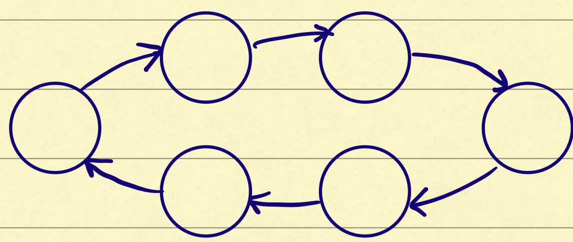
$K = 11 \qquad N = 6$

Which process has the token?

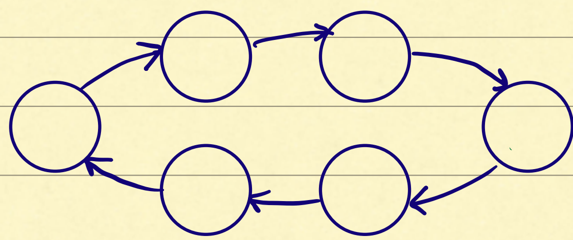


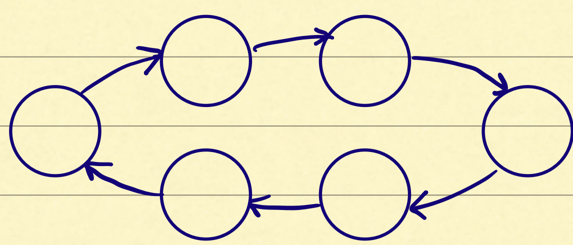


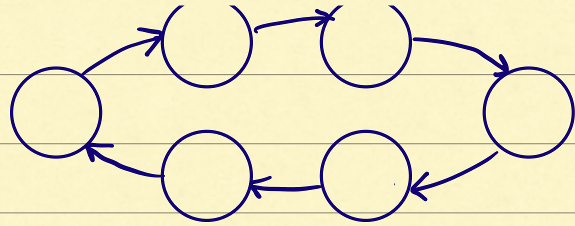
. . .



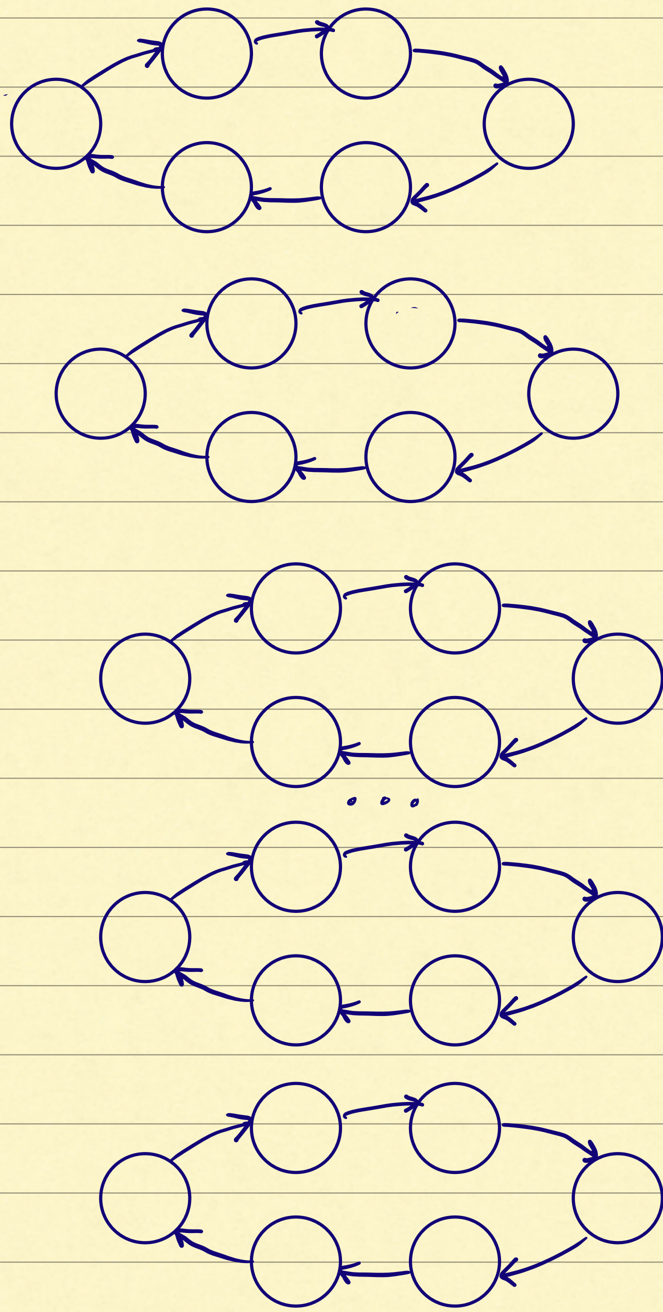
. . .







Execution from bad state



Automaton DijktraTR($N : \mathbb{N}$, $K : \mathbb{N}$)

variables

x :

actions

update (   )

transitions

update (i)

update (i)

This defines an automaton $\mathcal{A}_{Dijktra}$

$V = \{$

$\Theta =$

$A = \{$

$\mathcal{D} = \{$

Requirements for Token Ring

1. System always has at least one token
2. System always has exactly one token
3. System eventually has exactly one token
4. Process values are always $\leq K$.

Invariant. A requirement that holds always.

More formally: a property or a predicate that is satisfied in all reachable states of the system.

Conservation is related to invariants
Related to Conserved quantities

Def.

Invariant is an overapproximation of the reachable states.

I4.
I1.
I2'.

How to prove an invariant?

Given I, Check $Reach_{\mathcal{A}} \subseteq I$.

<u>Inductive Invariant</u>

<u>Theorem 7.1</u> if the following two conditions hold

<u>proof</u>. Consider any reachable state $v \in Reach_{\mathcal{A}}$.

It follows that any reachable state $v \in I$.

$\blacksquare$

<u>Remark</u> Thm 7.1 conditions (i) and (ii) give a method (sufficient condition) for proving invariant requirements.

<u>Exercise</u>. Check / Verify that $I2'$ is an invariant.

<u>Proof</u>.

Check init. $\Theta \subseteq I2'$ Follows from assumption that start state satisfies exactly one token.

Check transition. $\forall\, v \in I2'$ $a \in A$ if $v \xrightarrow{a} v'$ then $v' \in I2'$

Fix any $v \in I2'$ two cases to consider to show that $v' \in I2'$

1. $a = update(0)$

from precondition $v \lceil x[0] = v \lceil x[N-1]$

from $v \in I2'$ $\forall i \neq 0$ $\neg has\_token(i, v)$

i.e. $v \lceil x[i] = v \lceil x[i-1]$

from eff $v' \lceil x[0] = v \lceil x[0] + 1 \mod k$

$\forall i \neq 0$ $v' \lceil x[i] = v \lceil x[i-1] = v \lceil x[0]$

Therefore, only $i = 1$ has token

$v' \in I2'$

2. $a = update(i)$ $i \neq 0$

from precondition $v \lceil x[i] \neq v \lceil x[i-1]$

from $v \in I2'$ $\forall j \neq i$ $\neg has\_token(j, v)$

from eff we can check that

only $j+1 \bmod N$ has token.

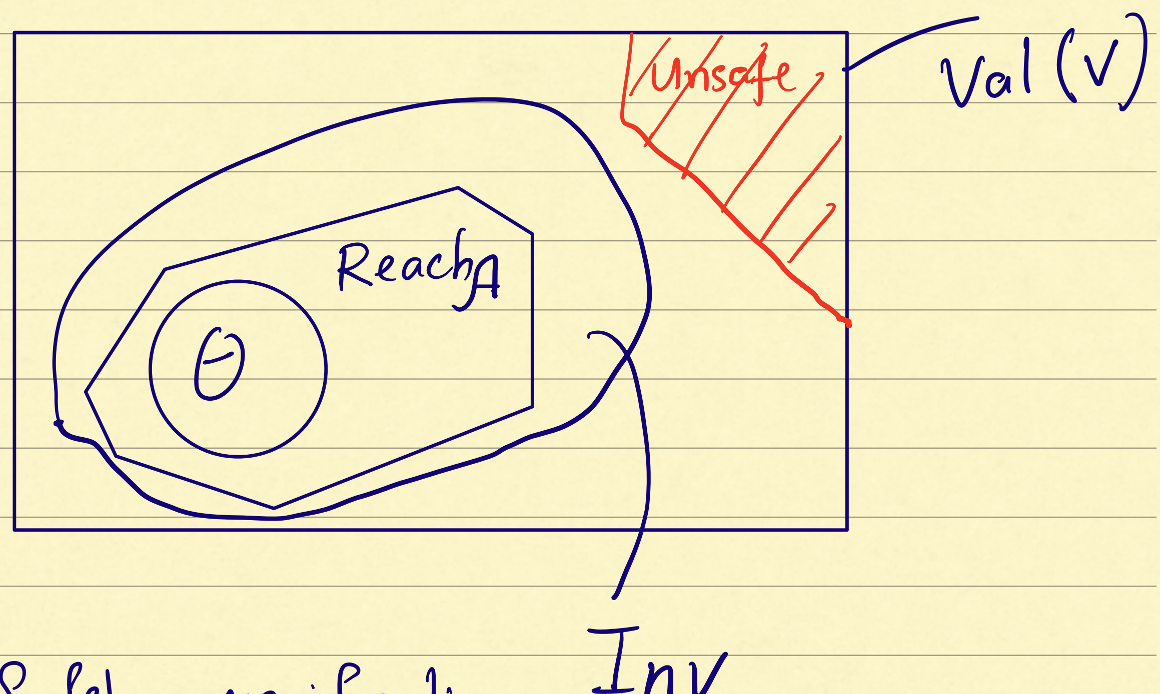
$V' \in I2'$ $\square$

Therefore $I2'$ is an invariant i.e.

if $\Theta \subseteq I2$ $\quad$ $\mathrm{Reach}_{\mathcal{A}} \subseteq I2$

Exercise. Prove invariance of $I1, I4$ using theorem 7.1.

Question. What if $I \subseteq \mathrm{Val}(V)$ is invariant but does not satisfy 7.1 (i) & 7.1 (ii)?



Safety verification