Models for Computation (Chapter 2)

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

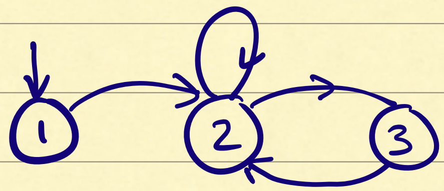
You have probably seen a finite state machine:

<u>Finite Automaton</u>

- States = $\{1, 2, 3\}$
- Start state = $\{1\}$
- Transitions $\subseteq S \times S$

$$\{\langle 1,2\rangle, \langle 2,2\rangle, \langle 2,3\rangle, \langle 3,2\rangle\}$$

behaviors : 1, 2, 3, 2, 3, ...



Generalization of FSMs

- <u>Arbitrary</u> number of states
- <u>Variables</u> implicity define states

Variables are the atoms or building blocks for models

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

8/26 Today

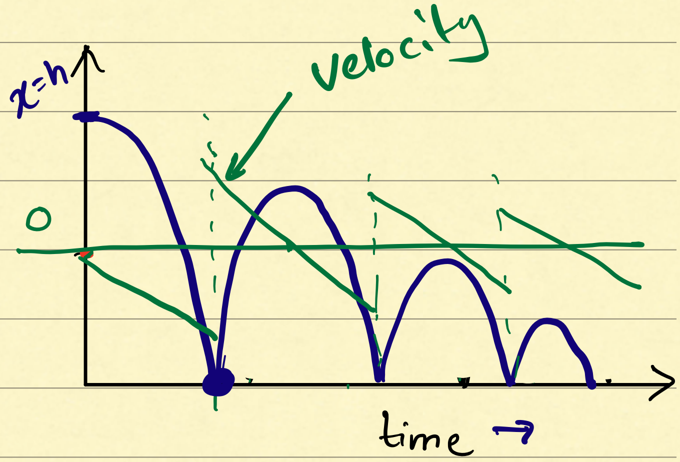
- Automata
- Executions
- Invariance
- Token Ring Example
- HW1 out

Example 1

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

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

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



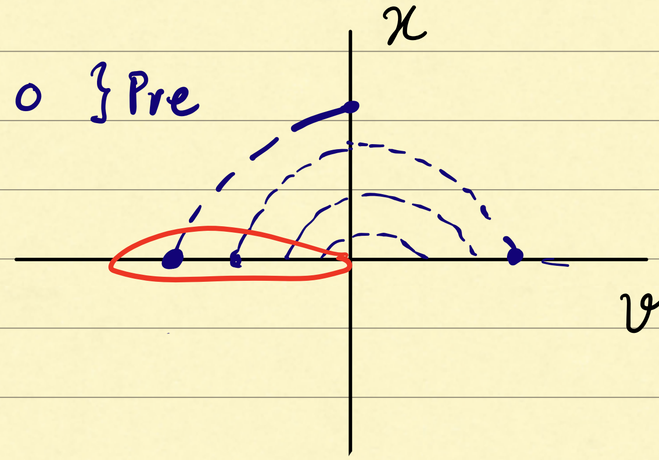
tick $x > 0$ Pre

eff. $v = v - g\Delta t$

$x = x + v\Delta t$

bouce $\left.\begin{array}{l} x = 0 \ \&\& \ v < 0 \end{array}\right\}$ Pre

eff. $v = -c.v$



Automaton Ball $(\Delta t, g, c > 0, h)$

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

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

Transitions

Tick. Pre $x > 0$

eff $v := v - g\Delta t$

$x := x + v\Delta t$

Bounce

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

Eff $v = -c.v$

Variables and Valuations

A **variable** is a name (a.k.a identifier) and an associated type

E.g $V = \{x, v\}$ is a set of variables

$type(x) = \mathbb{R}$

$type(v) = \mathbb{R}$

A **valuation** for a set of variables V maps each $x \in V$ to a value in $type(x)$.

E.g. $\mathbf{v} = \{x \mapsto 10.0 \; , \; v \mapsto 0.0\}$

$\mathbf{v}' = \{x \longrightarrow 0.2 \quad v \longmapsto 9.8\}$

$\Delta t = 1$

Given a valuation $\mathbf{v}$ of V a **restriction** of $\mathbf{v}$ to a particular variable $x \in V$ is written as $\mathbf{v}\lceil x = 10$ $\quad \mathbf{v}'\lceil x = 0.2$

E.g. $\mathbf{v}\lceil x =$ $\qquad \mathbf{v}'\lceil v =$

Set of all possible valuations of V : **Val(V)**

E.g. **Val(V)** $= \{\mathbf{v} \,/\, \mathbf{v}\lceil x \in \mathbb{R} \, , \, \mathbf{v}\lceil v \in \mathbb{R}\} \approx \mathbb{R} \times \mathbb{R}$

Automaton

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 $\langle \mathbf{v}, a, \mathbf{v}' \rangle \in \mathcal{D}$ 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. $v \xrightarrow{tick} v'$

Pre state $\quad$ Post state

$v \xrightarrow{a} v'$

a is enabled at state v

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

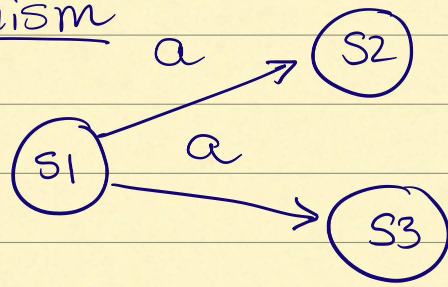
Q. When is bounce enabled?

The set of states where an action is enabled is called the <u>guard</u> or the <u>precondition</u> of the action.

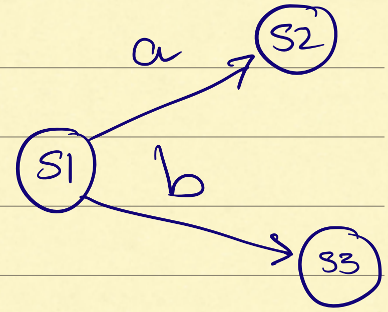
E.g Pre(bounce)

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 <u>execution</u> of $\mathcal{A}$ is an alternating sequence $\alpha = v_0 a_1 v_1 a_2 \ldots\ldots$ 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 executions of $\mathcal{A}$

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

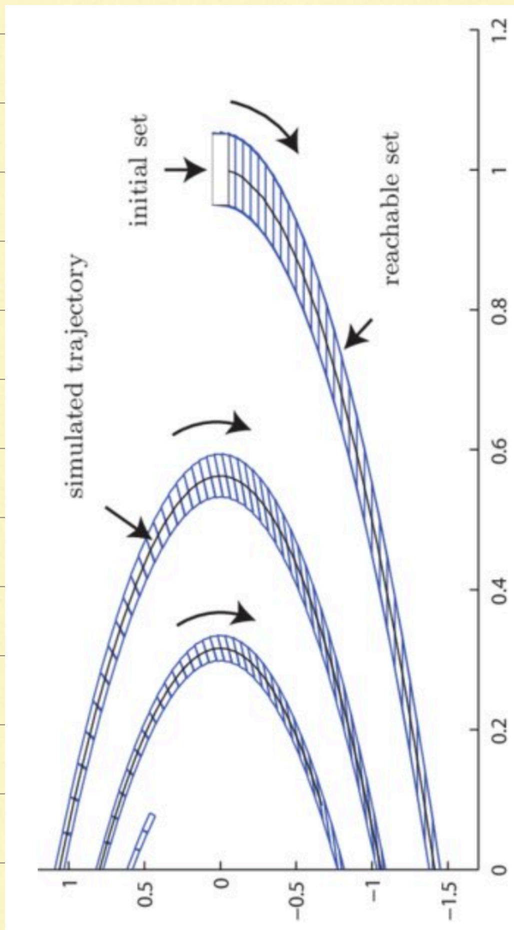
$Execs_{\mathcal{A}}(\Theta, k)$: Finite executions of length at most k.

$\alpha.fstate = v_0$

For a finite execution $\alpha = v_0 a_1 \ldots\ldots v_k$ 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 $\alpha.lstate = v$.



Reachable set of bouncing ball computed by CORA tool.

$Reach_A$ set of all reachable states of A.

$Reach_A(\Theta)$

$Reach_A(\Theta, k)$

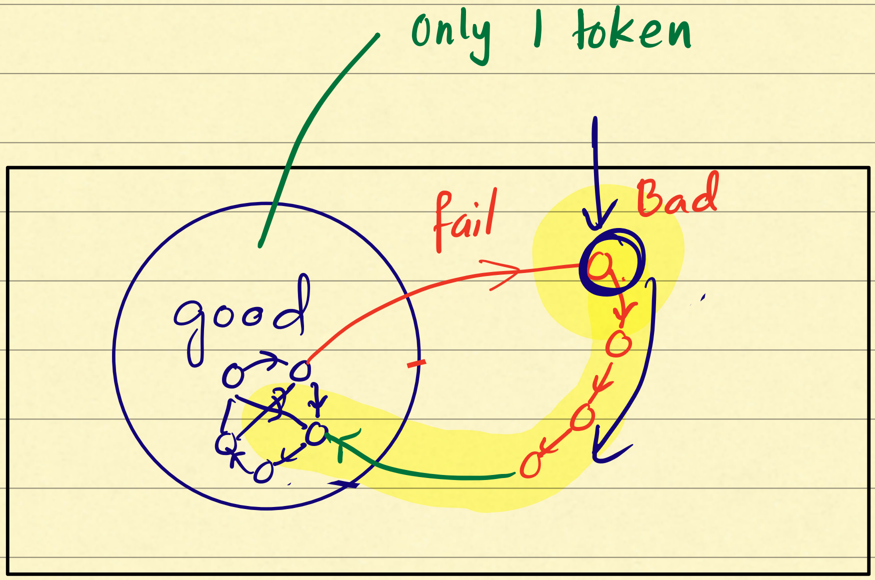
$v_0 a_1 v_1$

$\uparrow$

Ex2. Dijkstra's token ring algorithm

E. Dijkstra (1930 – 2002)

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



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

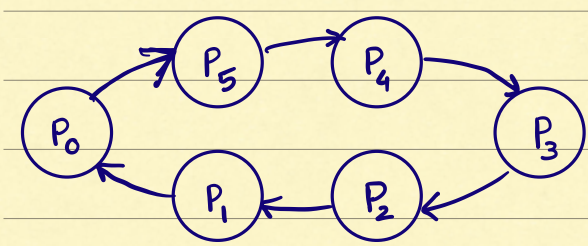
N processes

Each process has a single integer variable

$x_i \quad i \in [N] = \{0, \dots N-1\}$

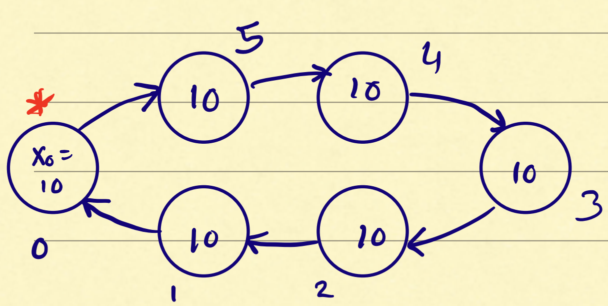
type(x_i) $[K]$

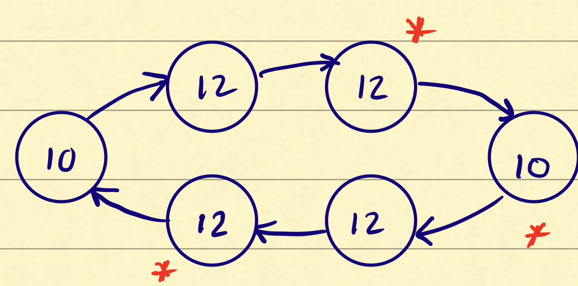
$K > N$



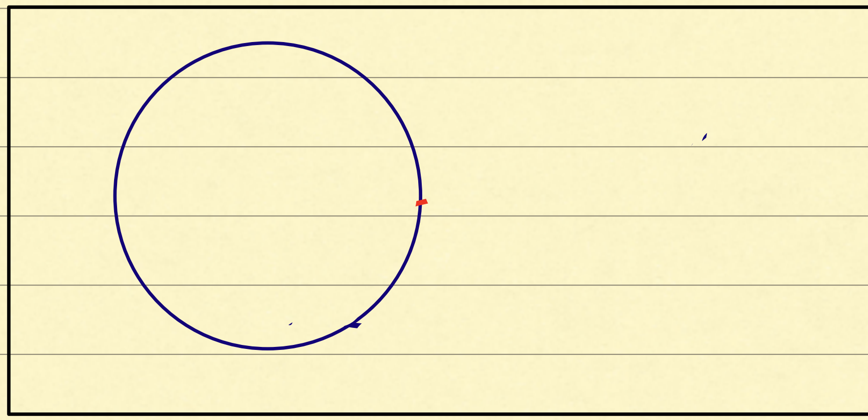
P_i <u>has token</u> if $\boxed{i = 0}$ then $x_i = x_{N-1}$

$i \neq 0$ $\quad x_i \neq x_{i-1}$





Which processes have a token?



Dijkstra's Algorithm

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

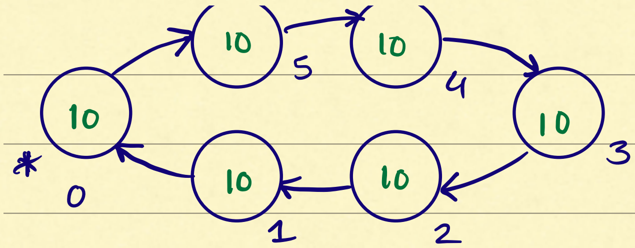
$$\left.\begin{array}{lll} \text{if } i=0 & \text{then} & x_0 := x_0 + 1 \bmod K \\ \quad\; i \neq 0 & \text{then} & x_i := x_{i-1} \end{array}\right\}$$

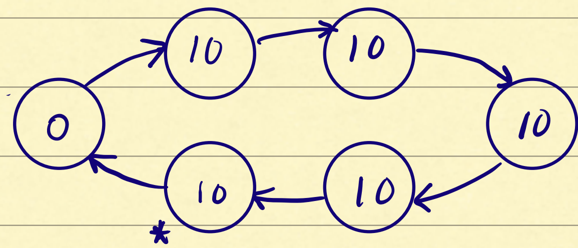
Execution of the system

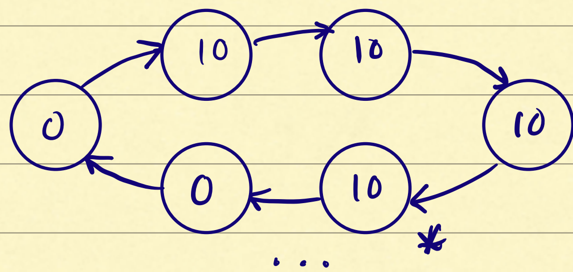
Starting from good state

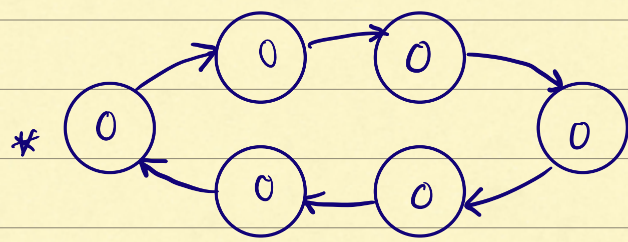
$K = 11 \qquad N = 6$

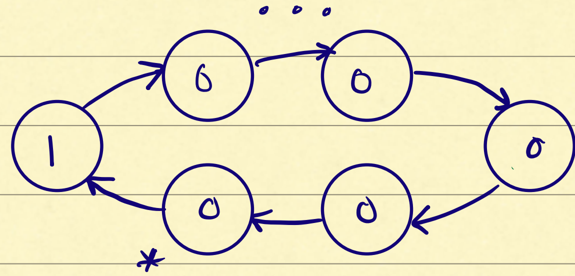
Which process has the token?

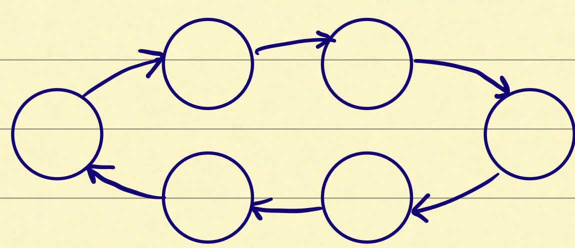




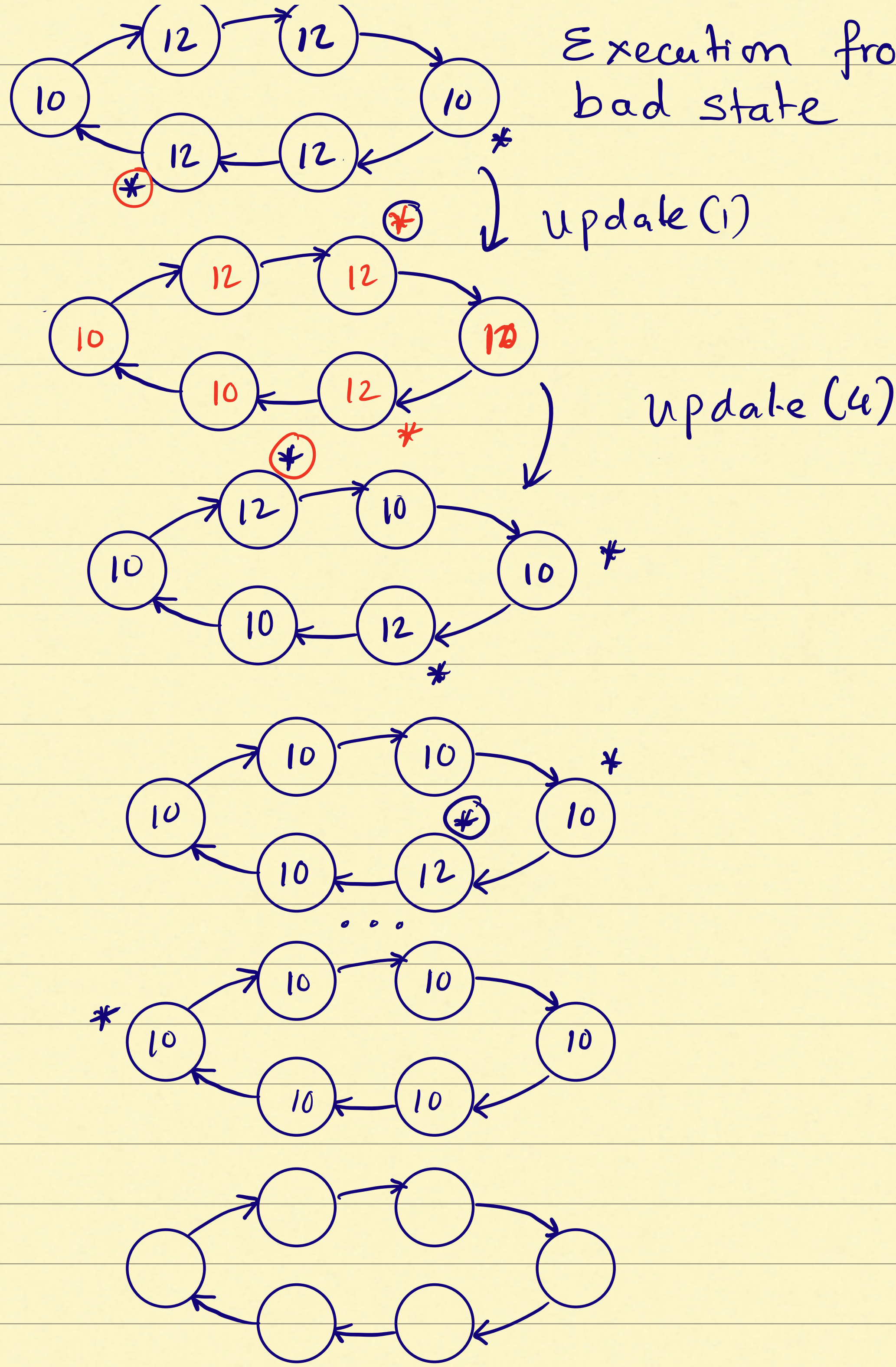








Execution from bad state



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

variables

$x : [[N] \to [K]]$, $\forall i \; x[i] = 10$

actions

update ($i : [N]$)

transitions

update (i), $i \neq 0$

Pre $x[i] \neq x[i-1]$

Eff $x[i] := x[i-1]$

update (i), $i = 0$

Pre $x[i] = x[N-1]$

Eff $x[i] := x[i] + 1 \mod K$

|| update (0), update (1), ... update (N-1)

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

$V = \{ x \}$

$\Theta = \{ \langle x[i] = 10 \quad \forall i \rangle \}$

$A = \{$ update(0) ... update (N-1) $\}$

$\mathcal{D} = \{ \langle x, \text{update}(i), x' \rangle \mid$ s.t

(i) if $i = 0$ the

$x[0] = x[N-1]$ and

$x'[0] = x[0] + 1 \mod K$ ...

(ii)

Requirements for Token Ring

1. ✓ System <u>always</u> has at least one token
2. * System <u>always</u> has exactly one token
3. ✗ System eventually has exactly one token
4. ✓ Process values are <u>always</u> $\leq K$.

<u>Invariant</u>. 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

<u>Def</u>. An invariant for automaton $\mathcal{A}$ is a set of states $I \subseteq val(V)$ such that

$$Reach_{\mathcal{A}} \subseteq I.$$

Invariant is an <u>overapproximation</u> of the reachable states.

I4.
I1.
I2'.

How to prove an invariant?

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

$\{x \mid$ System has single token in $x\}$

Inductive Invariant Floyd-Hoare Reasoning

Theorem 7.1 if the following two conditions hold

→ (a) Start. $\Theta \subseteq I$

→ (b) transition closure. $\forall v, v', a \in A$
if $v \in I$ and $v \xrightarrow{a} v'$ then $v' \in I$.

then I is an invariant, i.e. $\underbrace{\boxed{Reach_{\mathcal{A}} \subseteq I}}$.

$\mathcal{A} = \langle val(V), \Theta, A, \mathcal{D} \rangle$

proof. Consider any reachable state
$v \in Reach_A$. $\qquad v \in I$

From def of reachable states, $\exists$ finite $\alpha \in Exec_A$
s.t. $\alpha = v_0 a_1 v_1 \dots v_k = v$

We will prove $v \in I$ by induction on the length of α.

Base case: α has length 1 $\quad \alpha = v_0 = v$

Since α is an exec $\quad v_0 \in \Theta$

by (a) $\quad \Theta \subseteq I \Rightarrow v_0 \in I$.

Inductive step: $\alpha = \alpha' a v$ and α' satisfies the statement $\alpha'.lstate \in I$.

by (b) it follows that $v \in I$.

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

$\blacksquare$

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

$\forall n \quad P(n)$

by induction

$P(0)$

assuming $P(n)$

show $P(n+1)$

I2'. Assuming initial state has a single token, system always has a single token.

Proof. We will use theorem 7.1.

(a) initial state has a single token

(b) for any state with a single token any transition takes us to a new state with a single token.

Two cases $v \xrightarrow{a} v'$ and v has 1 token

(1) $a = update(0)$

$\forall i,j \; x[i] = x[j]$ from $v \in I2$ and pre update(0)

$x'[0] = x[0]+1 \mod K$

$x'[i] = x[i] \quad \forall i \neq 0$

in the postate $x[0] \neq x[N-1]$ 0 does not have token

$x[1] \neq x[0]$ 1 has token

$\forall i \notin \{0,1\}$ does not have token

$v \in I2$.

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

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

Proof.

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.

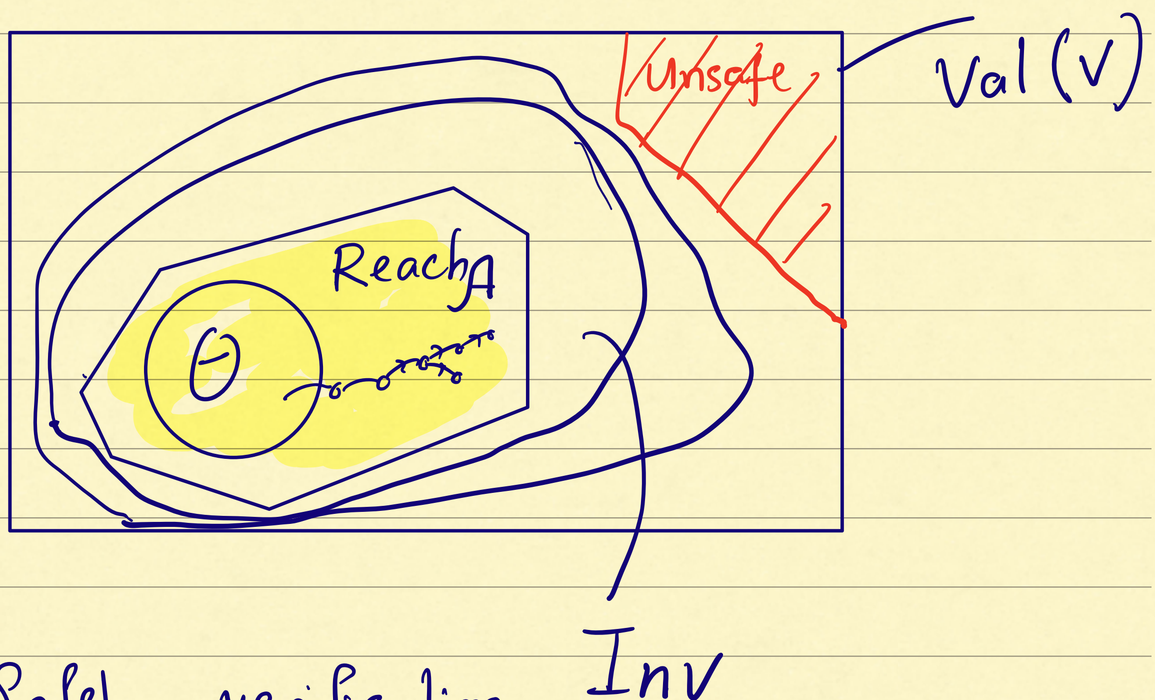
$v' \in I2'$ ☒

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

if $\Theta \subseteq I2$ $\quad Reach_A \subseteq I2$

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

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



Safety verification

ball dropped from $x = h$

I_{ball} : $x \leq h$ $\qquad\qquad$ $x \leq h$

This is an invariant but not
an inductive invariant