ECE584 L2

Discrete models, reachability, and Invariance

August 28th 2025, ECEB 3013

Sayan Mitra

CSL 266

mitras@illinois.edu

@Mitrasayn

Announcements

- Class Campuswire created: https://campuswire.com/c/G0D38569B/feed
  - Use to create discuss projects
  - We may use it for announcements
  - Main source of information will be website
- Links to notable past projects restored (mostly)
  - https://mitras.ece.illinois.edu/ECE584/project.html
- HW1 out tomorrow due in 2 weeks

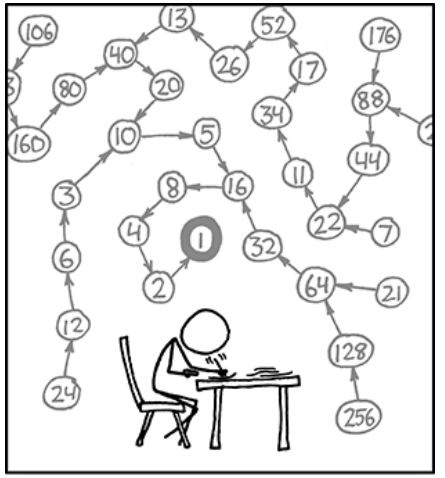
Recall automata and requirements

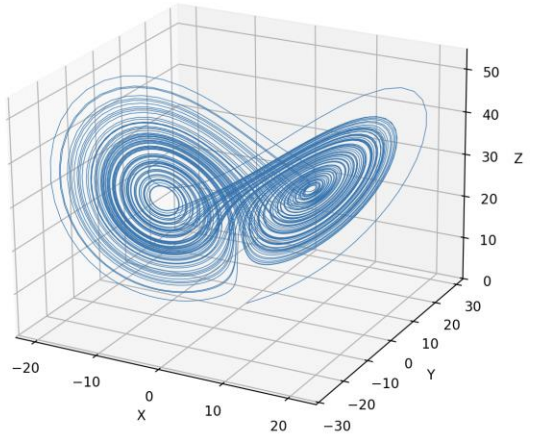
Automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$

Liveness/Progress

All executions of Collatz (with $\Theta \equiv \mathbb{Z}^+$) eventually enter reaches $\langle x \mapsto 1 \rangle$

Do all executions of Lorentz converge?





Safety/Invariance

All executions of Collatz with $x_0 \in [1,200]$ stays inside $x_i \leq 10{,}000$

Safety/Invariance. Do they stay bounded?

Example 3: Dijkstra's Token Ring algorithm

A mutual exclusion algorithm is a distributed algorithm that assures that a collection of participating processes can (1) always eventually access a shared resource (e.g., a printer) and (2) no two processes ever access the resource simultaneously.

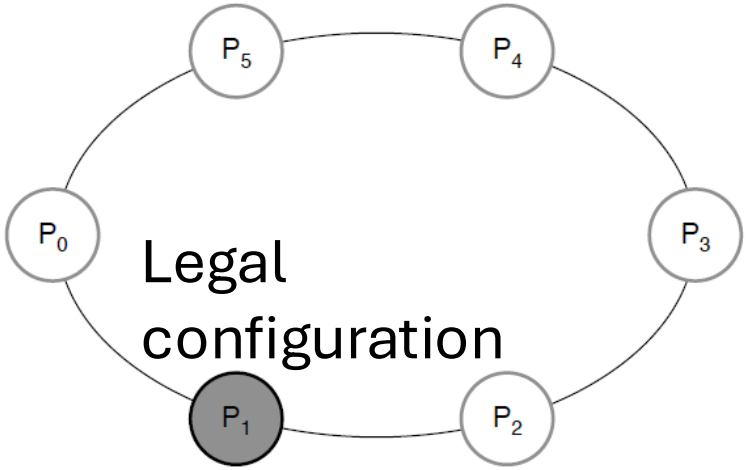
A token-based mutual exclusion algorithm uses a "token" to provide access to the shared resource

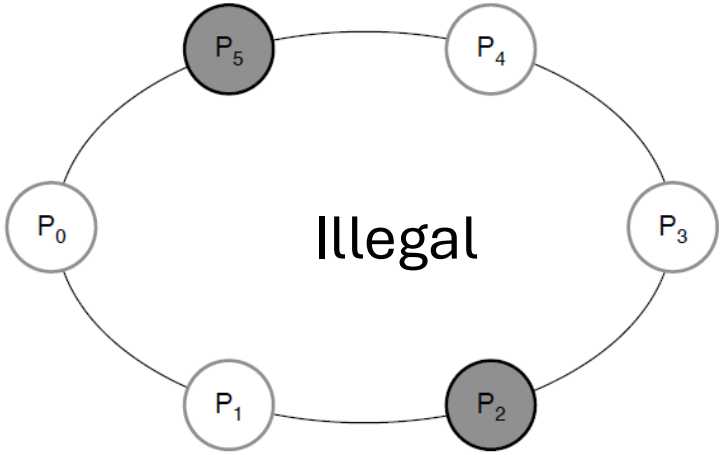
N processes with ids 0, 1, …, N-1

Unidirectional: each j>0 process P_j reads the state of only the predecessor P_{j-1}; P_0 reads only P_{N-1}

Legal configuration = exactly one "token" in the ring and this token circulates in the ring

Even if multiple tokens arise because of faults, if the algorithm continues to work correctly, then eventually there is a single token; this is the self stabilizing property





Self-stabilizing Systems in Spite of Distributed Control, CACM, 1974.

Dijkstra's Algorithm ['74]

State of each process j is a single integer variable $x[j] \in [K]$, where K > N

A process can update its state only when it has the token which is determined by its state and its neighbor's state

$$hasToken(x, j) \equiv (j = 0 \wedge x[j] = x[N-1]) \vee (j \neq 0 \wedge x[j] \neq x[j-1])$$

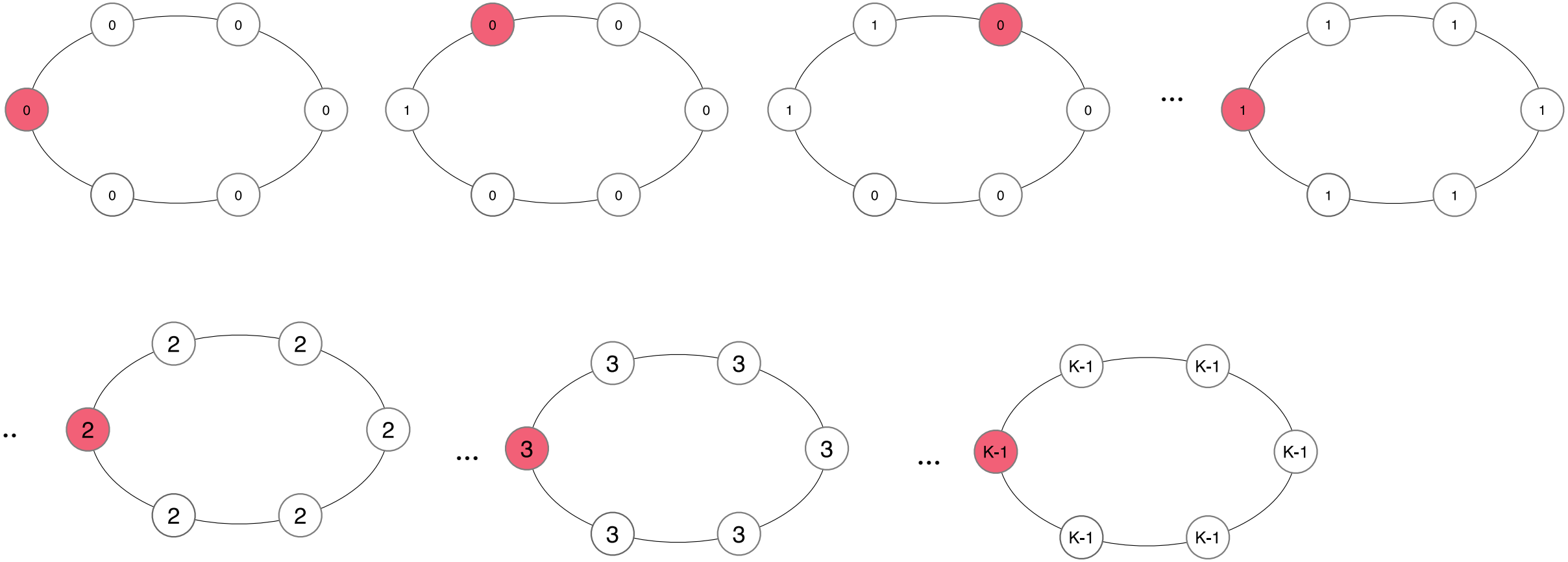
Update(i: [N]): // transition rule

If $hasToken(x, i)$

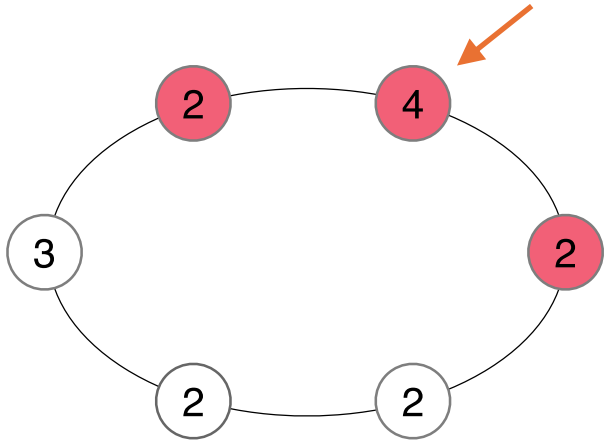
If $i = 0$ **then** $x[0] := x[0] + 1 \; mod \; K$

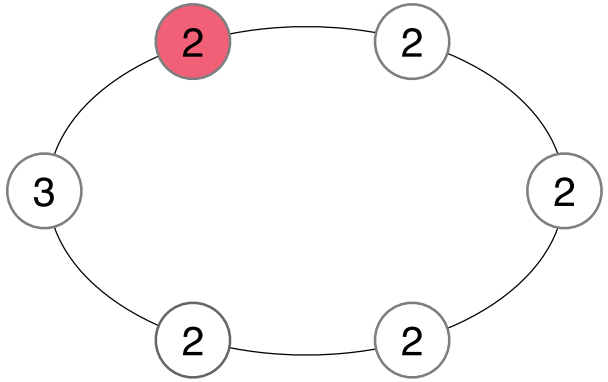
else $x[j] := x[j-1]$

Sample executions: from a legal state (single token)

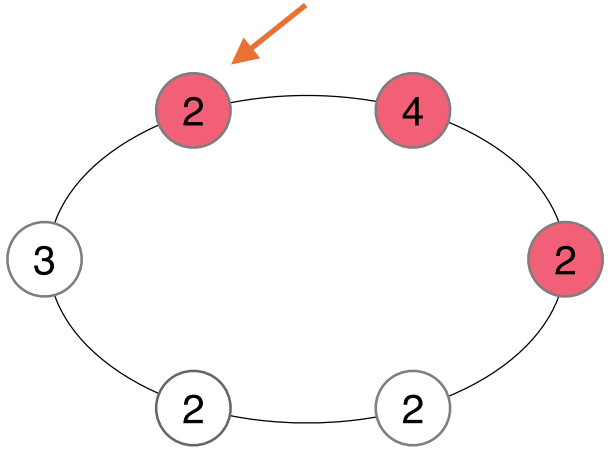


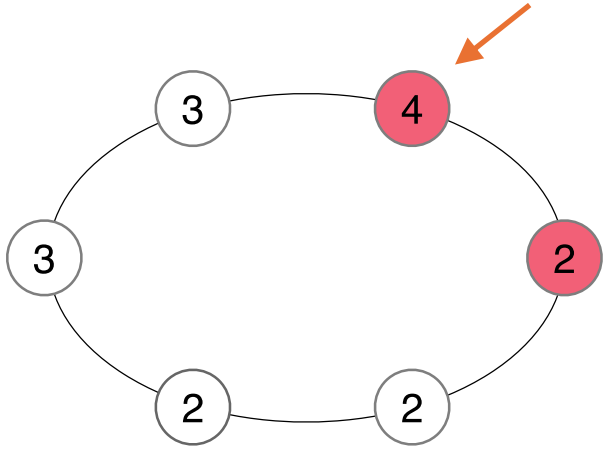
Execution from an illegal state



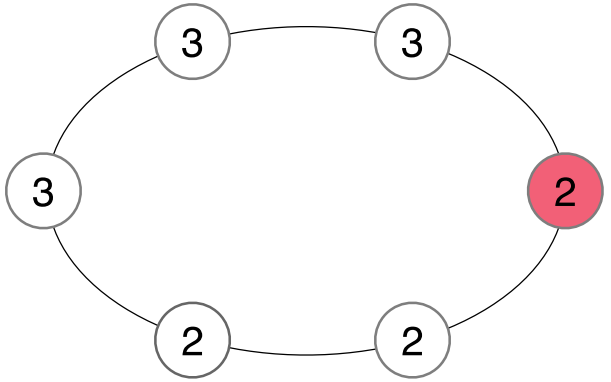


Legal in single “step”





Legal in two steps



Synchronous model: All processes update simultaneously

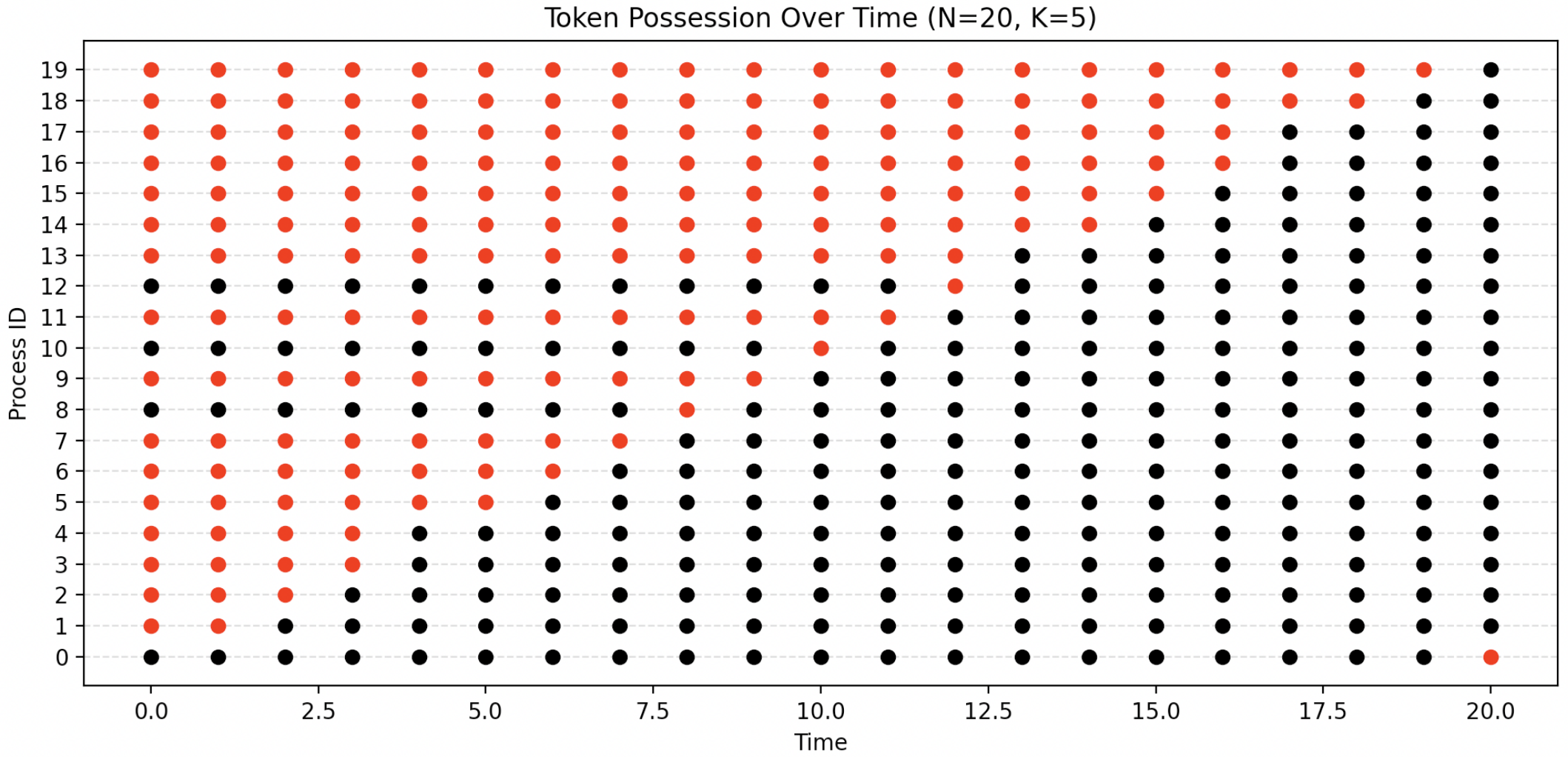
```
def has_token(x, i, N):
"""Return True if process i has the token."""
if i == 0:
  return x[0] == x[N - 1]
else:
  return x[i] != x[i - 1]

def simulate(N, K, T, delta_t):
x = [0]*N

while t <= T:
  x_old = x.copy() // Update

  for i in range(N):
    if has_token(x_old, i, N):
      if i == 0:
        x[i] = (x_old[0] + 1) % K
      else:
        x[i] = x_old[i - 1]
```

Synchronous Token Ring



Asynchronous Nondeterministic Automaton

We will write automata precisely using this style

automaton DijkstraTR(N:ℕ, K: ℕ), **where** K > N

variables
x:[[N] -> [K]]

actions
update(i:[N])

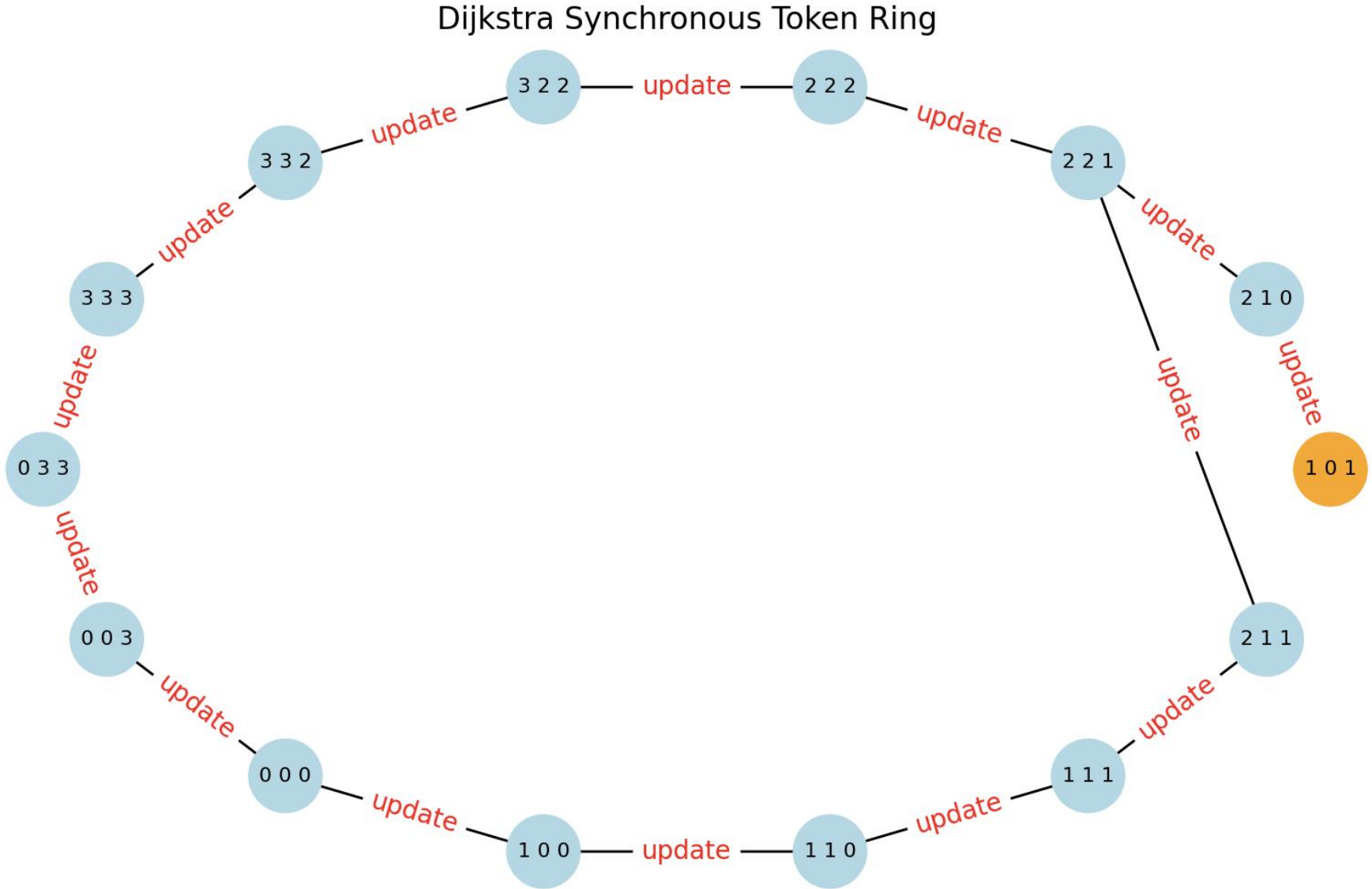
transitions
Update(p=0)
Pre has_token(i) and i== p
Eff x[i] = (x[i] + 1) % K
Update(p>0)
Pre has_token(i) and i == p
Eff x[i] = x[i-1]

An **automaton** $\mathcal{A}$ is a 4-tuple $\langle V, \Theta, A, \mathcal{D} \rangle$ where

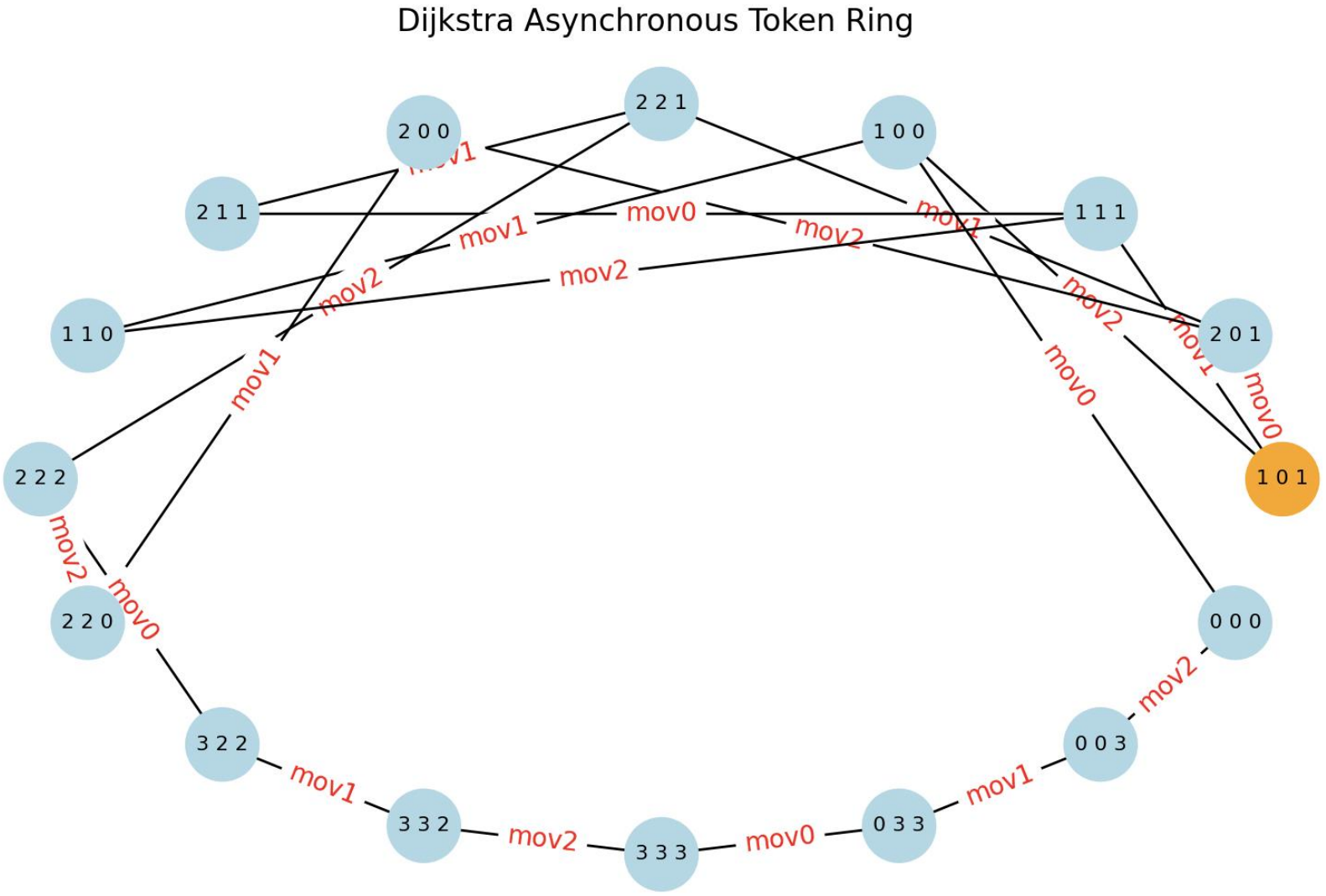
- $V = \{x\}$
- $type(x) = [N] \rightarrow [K]$
- **State space** $\equiv [K]^N$
- $\Theta = \cdots$
- $A = \{update(0), update(1), \ldots, update(N-1)\}$
- $\mathcal{D} = \{\langle \boldsymbol{x}, update(i), \boldsymbol{x}' \rangle \mid i \in [N] \wedge hasToken(\boldsymbol{x}, i)$
  - If $i = 0$ then $\boldsymbol{x}'[0] = \boldsymbol{x}[N-1] + 1 \ \% \ K$
  - If $i \neq 0$ then $\boldsymbol{x}'[i] = \boldsymbol{x}[i-1]\}$

This is a nondeterministic automaton.

All processes with tokens **could** perform a transition


Dijkstra Synchronous Token Ring
3 2 2
update
2 2 2
update
2 2 1
update
2 1 0
update
1 0 1
update
2 1 1
update
1 1 1
update
1 1 0
update
1 0 0
update
0 0 0
update
0 0 3
update
0 3 3
update
3 3 3
update
3 3 2
update

Nondeterministic execution



Internal and External Nondeterminism

$\mathcal{A}$ is **deterministic** if $|\Theta| = 1$ and there is at most one next state $\boldsymbol{v'}$ from any state $\boldsymbol{v}$

$enabled(\boldsymbol{v}) = \{a \in A \mid \exists \boldsymbol{v'}\ \boldsymbol{v} \rightarrow_a \boldsymbol{v'}\}$ is the set of action that **can** occur at $\boldsymbol{v}$.

If $|enabled(\boldsymbol{v})| > 1$ the automaton has **external nondeterminism**, i.e., there is a choice in which action **should be** performed at each state (e.g., which process updates)

$post(\boldsymbol{v}, \boldsymbol{a}) = \{\boldsymbol{v'} \mid \boldsymbol{v} \rightarrow_a \boldsymbol{v'}\}$ is the set of next states of $\boldsymbol{v}$ after performing $\boldsymbol{a}$

If $|post(\boldsymbol{v}, \boldsymbol{a})| > 1$ the automaton has **internal nondeterminism**, i.e., the choice of the action does not completely resolve uncertainty (e.g., $coin = \boldsymbol{choose}\ \{0,1\}$)

Nondeterminism

- Multiple initial states
- External nondeterminism: Multiple actions enabled from the same state
- Internal nondeterminism: Multiple post-states from the same state and action

Executions, Reachability, and Invariants

Automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$

An executions models a particular behavior of the automaton $\mathcal{A}$

An ***execution*** of $\mathcal{A}$ is an alternating (possibly infinite) sequence of states and actions $\alpha = u_0 a_1 u_1 a_2 u_3$ …such that:

1. $u_0 \in \Theta$
2. $\forall\, i$ in the sequence, $u_i \rightarrow_{a_{i+1}} u_{i+1}$

For a ***finite*** execution, $\alpha = u_0 a_1 u_1 a_2 u_3$ the ***last state*** $\alpha.lstate = u_3$ and the length of the execution is 3.

In general, how many executions does an $\mathcal{A}$ have?

Reachable states and invariants

A state $\boldsymbol{u}$ is ***reachable*** if there exists an execution α such that $\alpha.lstate = \boldsymbol{u}$

$Reach_{\mathcal{A}}(\Theta, K)$: set of states reachable from Θ by automaton $\mathcal{A}$ in at most K steps

$Reach_{\mathcal{A}}(\Theta)$: set of states reachable from Θ by automaton $\mathcal{A}$ (in any finite number of steps)

We write $Reach_{\mathcal{A}}$ if the initial state is clear from context

$Reach_{\mathcal{A}}$ "Everything that **can** happen"

$(Reach_{\mathcal{A}})^c$ "Things that **never** happen"

Reachability as graph search

- Q1. Given finite state $\mathcal{A}$, is a state $\boldsymbol{v} \in val(V)$ reachable?
- Define a graph $G_{\mathcal{A}} = \langle Ver, Edg \rangle$ where
  - $Ver = val(V)$
  - $Edg = \{(u, u') | \exists\, a \in A, u \rightarrow_a u'\}$
- Q2. Does there exist a path in $G_{\mathcal{A}}$ from any state in Θ to u ?
- Perform DFS/BFS on $G_{\mathcal{A}}$
- This called **explicit state model checking** because each individual state is stored in memory
- Works only for finite state automata and does not scale to very large state spaces

Symbolic Model Checking: 10^{20} States and Beyond

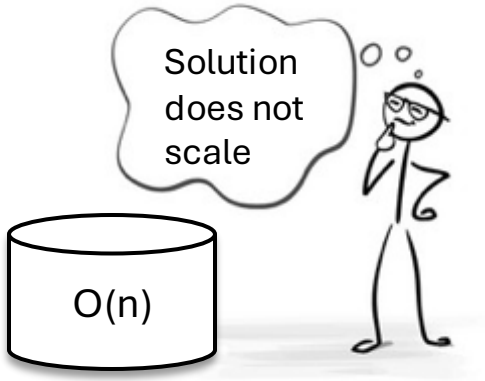
J. R. Burch E. M. Clarke K. L. McMillan*
School of Computer Science
Carnegie Mellon University

D. L. Dill L. J. Hwang
Stanford University

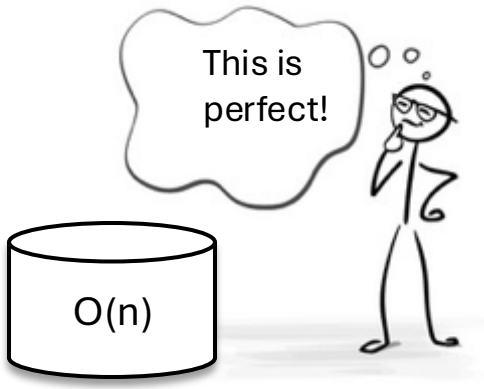
Abstract

Many different methods have been devised for automatically verifying finite state systems by examining state-graph models of system behavior. These methods all depend on decision procedures that explicitly represent the state space using a list or a table that grows in proportion to the number of states. We describe a general method that represents the state space *symbolically* instead of explicitly. The gener-

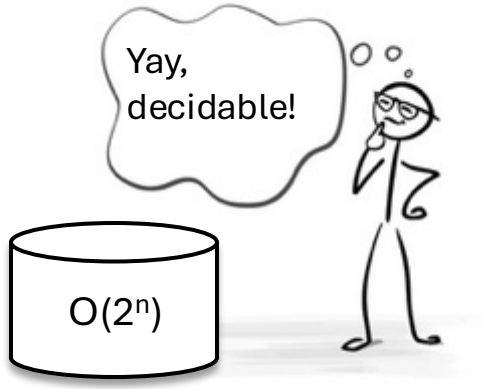
Perspectives on scalability



data scientist



algorithmist



verification engineer

Sets of states are symbolic properties or **predicates**

A set of states defines a **property** or a **predicate** and vice versa

For example: $\langle x \mapsto 5\rangle, \langle x \mapsto 6\rangle, \ldots$ defines the predicate $x \geq 5$

A ***predicate*** over a set of variable X is a Boolean-valued formula involving the variables in X.

- $\phi_1 : x[1] = 1$
- $\phi_2 : \forall i \in [N], x[i] = 0$

A valuation **u** **satisfies a predicate** $\boldsymbol{\phi}$ if substituting the values of the variables in **u** in ϕ makes it evaluate to **True.** We this as write $\mathbf{u} \vDash \boldsymbol{\phi}$

Examples: $\boldsymbol{u} = \langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, \ldots \rangle\rangle$; $\boldsymbol{v} = \langle x \mapsto \langle 0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 0, \ldots \rangle\rangle$

- $\boldsymbol{u} \vDash \phi_2$, $(\boldsymbol{u} \nvDash \phi_1)$, $\boldsymbol{v} \vDash \boldsymbol{\phi_1}$ and $\boldsymbol{v} \nvDash \boldsymbol{\phi_2}$

We will find it convenient to represent state sets (e.g., $Reach_{\mathcal{A}}$ and Θ) by predicates

$[[\boldsymbol{\phi}]] = \{\boldsymbol{u}$:Val(V) | $\boldsymbol{u} \vDash \phi\}$ set of all states that satisfy $\boldsymbol{\phi}$

- $[[\phi_2]] = \{\langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0, 5 \mapsto 0 \rangle\rangle\}$
- $\Theta \subseteq val(x)$ is the set of initial states of the automaton; often specified by a **predicate** over X

Invariance

An ***invariant*** is a set of states (or a property) I such that $Reach_{\mathcal{A}} \subseteq I$, i.e., a property that **always holds**

Invariants capture conservation laws

Invariants

All executions of Collatz with $x_0 \in [1,200]$ stays inside $x_i \leq 10{,}000$

"Exactly one process has a token"

$I_1: |\{i \in [N] \,|hasToken(\boldsymbol{x}, i)\}| \geq 1$

"At least one process has a token"

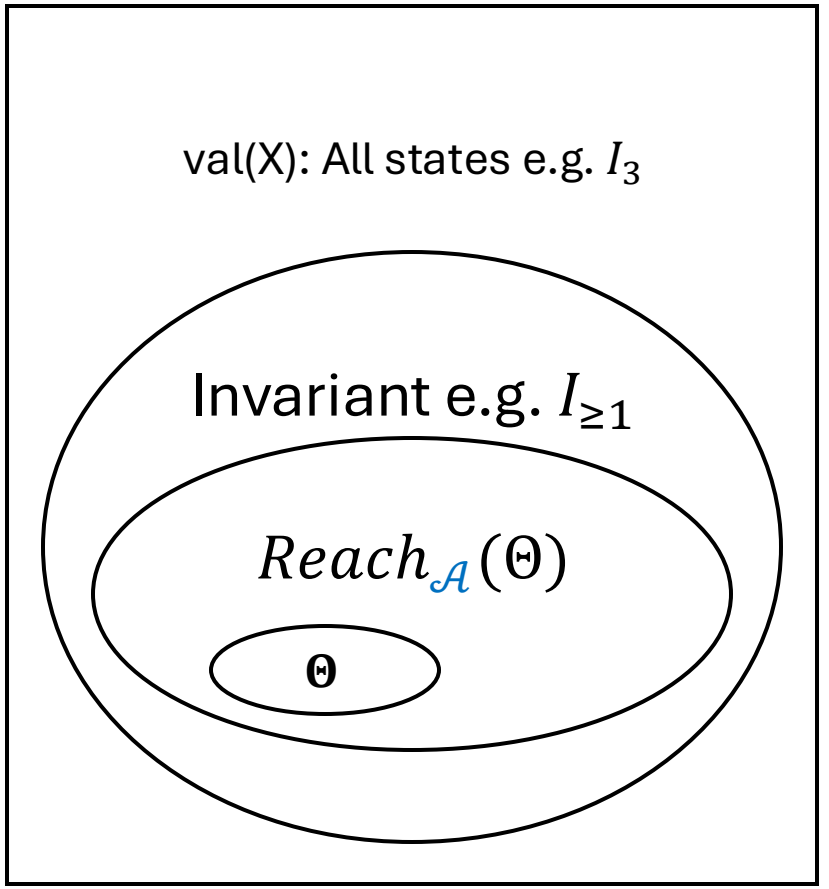
$I_{\geq 1}: |\{i \in [N] \,|hasToken(\boldsymbol{x}, i)\}| \geq 1$

"All processes have values at most K-1"

$I_3: \forall i \in [N], x[i] \leq K - 1$

"Car always stops at stop sign"

"Drone does not deviate more that 5m from ref path"



Summary

- Invariants over-approximate reachable states

- Given an automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a candidate invariant $I \subseteq Val(X)$, to check that $Reach_{\mathcal{A}} \subseteq I$.
  - A sufficient condition is to check that I satisfies start condition and transition condition of **Theorem 7.1**
  - Such an invariant is called an inductive invariant
  - Use proof failure to get I' which may be a stronger invariant, i.e., $I' \Rightarrow I$

Assignments

- Read. Modeling computation: Chapter 2 of CPSBook, first part of Chapter 7, and section on SAT/SMT

- Form team (optional*) and go through Project Jumpstart exercise

- Next: Satisfiability