Lecture 2: Modeling Computation

Huan Zhang
huan@huan-zhang.com

Slides adapted from Prof. Sayan Mitra’s slides in Fall 2021
Outline

Goal of this course: model anything!

This lecture: model computations

Today: Automaton as a model for computations
Automata or discrete transition systems

• The “state” of a system captures all the information needed to predict the system’s future behavior
• Behavior of a system is a sequence of states
• *Our ultimate goal: write programs that prove properties about all behaviors of a system*
• “Transitions” capture how the state can change
All models are wrong, some are useful

The complete state of a computing system has a lot of information

• values of program variables, network messages, position of the program counter, bits in the CPU registers, etc.
• thus, modeling requires judgment about what is important and what is not

Mathematical formalism used is called automaton a.k.a. discrete transition system
Automata or discrete transition systems

• Example: you probably know the finite state machine (FSM)
  • States: \{1, 2, 3\}
  • Start state: \{1\}
  • Transitions

• Automata is more general:
  • We define “states” implicitly using variables
  • The number of state is arbitrary
Example: Dijkstra’s mutual exclusion algorithm

**Informal Description:** A token-based mutual exclusion algorithm on a ring network

- Collection of processes that send and receive bits over a ring network so that only one of them has a “token” to access a critical resource (e.g., a shared calendar)

**Discrete model**

- Each process has variables that take only discrete values
- Time elapses in discrete steps

N processes with ids 0, 1, ..., N-1
Unidirectional means: each $i > 0$ process $P_i$ reads the state of only the predecessor $P_{i-1}$; $P_0$ reads only $P_{N-1}$

1. Legal configuration = exactly one “token” in the ring
2. Single token circulates in the ring
3. 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
Dijkstra’s mutual exclusion Algorithm [‘74]

N processes: 0, 1, ..., N-1
state of each process j is a single integer variable x[j] ∈ {0, 1, 2, K-1}, where K > N
The “update” action is defined differently for P0 vs. others

\[ P_0 \quad \text{if } x[0] = x[N-1] \quad \text{then } x[0] := x[0] + 1 \text{ mod } K \]

\[ P_j, j > 0 \quad \text{if } x[j] \neq x[j-1] \quad \text{then } x[j] := x[j-1] \]

\[ p_i \text{ has TOKEN if and only if the blue conditional is true} \]
Sample executions: from a legal state (single token)
Execution from an illegal state

Legal in single “step”

Legal in two steps
Execution from an illegal state
A language for specifying automata (IOA)

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

  type ID: enumeration [0,...,N-1]
  type Val: enumeration [0,...,K-1]

actions
  update(i:ID)

variables
  x:[ID -> Val]

transitions
  update(i:ID)
    pre i = 0 \& x[i] = x[N-1]
    eff x[i] := (x[i] + 1) % K

  update(i:ID)
    pre i > 0 \& x[i] =~ x[i-1]
    eff x[i] := x[i-1]
A language for specifying automata

**automaton DijkstraTR**($N$:$\text{Nat}$, $K$:$\text{Nat}$), where $K > N$

- **type ID**: enumeration [0,...,$N-1$]
- **type Val**: enumeration [0,...,$K-1$]

**actions**
- **update**($i$:ID)

**variables**
- $x$: [ID -> Val]

**transitions**
- **update**($i$:ID)
  - **pre** $i = 0 \land x[i] = x[N-1]$
  - **eff** $x[i] := (x[i] + 1) \mod K$

- **update**($i$:ID)
  - **pre** $i > 0 \land x[i] \neq x[i-1]$
  - **eff** $x[i] := x[i-1]$

Name of automaton and formal parameters
A language for specifying automata

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

type ID: enumeration [0,...,N-1]
type Val: enumeration [0,...,K-1]

actions
update(i:ID)

variables
x: [ID -> Val]

transitions
update(i:ID)
  pre i = 0 \ x[i] = x[N-1]
  eff x[i] := (x[i] + 1) % K

update(i:ID)
  pre i >0 \ x[i] ~= x[i-1]
  eff x[i] := x[i-1]

symbols -> maps, \ and, \ or, ~= not equal, % mod

user defined type declarations
A language for specifying automata

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

type ID: enumeration [0,...,N-1]
type Val: enumeration [0,...,K-1]

actions
update(i:ID)

variables
x:[ID -> Val]

transitions
update(i:ID)
  pre i = 0 \& x[i] = x[N-1]
  eff x[i] := (x[i] + 1) % K

update(i:ID)
  pre i >0 \& x[i] =~= x[i-1]
  eff x[i] := x[i-1]

declaration of “actions” or transition labels; actions can have parameter; this declares the actions update(0), update(1), ..., update(N-1)

symbols -> maps, \& and, \lor or, =~ not equal, % mod
A language for specifying automata

**automaton** \( \text{DijkstraTR}(N:\text{Nat}, K:\text{Nat}), \text{where} \ K > N \)

**type** \( \text{ID} \): **enumeration** \([0,\ldots,N-1]\)

**type** \( \text{Val} \): **enumeration** \([0,\ldots,K-1]\)

**actions**

update(i:ID)

**variables**

\( x: [\text{ID} \rightarrow \text{Val}] \)

**transitions**

update(i:ID)

pre \( i = 0 \land x[i] = x[N-1] \)

eff \( x[i] := (x[i] + 1) \mod K \)

update(i:ID)

pre \( i > 0 \land x[i] \neq x[i-1] \)

eff \( x[i] := x[i-1] \)

declaration of state variables or variables; this declares an array \( x[0], x[1], \ldots, x[N-1] \) of Val’s symbols \( \rightarrow \) maps, \( \land \) and, \( \lor \) or, \( \neq \) not equal, \( \mod \)
A language for specifying automata

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

- type ID: enumeration [0,...,N-1]
- type Val: enumeration [0,...,K-1]

actions
- update(i:ID)

variables
- x:[ID -> Val]

transitions
- update(i:ID)
  - pre i = 0 \& x[i] = x[N-1]
  - eff x[i] := (x[i] + 1) % K

- update(i:ID)
  - pre i >0 \& x[i] \neq x[i-1]
  - eff x[i] := x[i-1]

declaration of transitions:
for each action this defines when the action can occur (pre) and how the state is updated when the action does occur (eff)

symbols -> maps, \& and, \lor or, \neq not equal, % mod
The language defines an automaton

An automaton is a tuple $\mathcal{A} = \langle X, \Theta, A, D \rangle$ where

- $X$ is a set of names of variables; each variable $x \in X$ is associated with a type, $\text{type}(x)$
  - A valuation for $X$ maps each variable in $X$ to its type
  - Set of all valuations: $\text{val}(X)$ this is sometimes identified as the state space of the automaton
- $\Theta \subseteq \text{val}(X)$ is the set of initial or start states
- $A$ is a set of names of actions or labels
- $D \subseteq \text{val}(X) \times A \times \text{val}(X)$ is the set of transitions
  - a transition is a triple $(u, a, u')$
  - We write it as $u \rightarrow_a u'$
Well formed specifications in IOA Language define automata variables and valuations

variables $s, v$: Real; $a$: Bool
$X = \{s, v, a\}$

Example valuations of $X$
- $\langle s \mapsto 0, \ v \mapsto 5.5, \ a \mapsto 0 \rangle$
- $\langle s \mapsto 10, \ v \mapsto -2.5, \ a \mapsto 1 \rangle$

set of all possible valuations or “state space” is written as $\text{val}(X)$

$$\text{val}(X) = \{\langle s \mapsto c_1, \ v \mapsto c_2, \ a \mapsto c_3 \rangle | c_1, c_2 \in R, \ c_3 \in \{0,1\}\}$$

type $\text{ID}$: $[0,...,N-1]$  
variables $x$: $[\text{ID}>\text{Vals}]$

Fix $N = 5, \ K = 7$

$x$: $\{0,...,4\} \rightarrow \{0,...,6\}$

Example valuations:
- $\langle x \mapsto \langle 0 \mapsto 0, \ 1 \mapsto 0, \ 2 \mapsto 0, \ 3 \mapsto 0, \ 4 \mapsto 0 \rangle \rangle$
- $\langle x \mapsto \langle 0 \mapsto 7, \ 1 \mapsto 0, \ 2 \mapsto 0, \ 3 \mapsto 0, \ 4 \mapsto 0, \ \rangle \rangle$

Valuations are usually denoted by bold small characters
E.g.,
$$\mathbf{u} = \langle x \mapsto \langle 0 \mapsto 0, \ 1 \mapsto 0, \ 2 \mapsto 0, \ 3 \mapsto 0, \ 4 \mapsto 0 \rangle \rangle$$

Notations
- $\mathbf{u}[x]$ is the value of variable $x$ in $\mathbf{u}$
- $\mathbf{u}[x[4] = 0$ array notation $[]$ works with $[$ as expected
States and predicates

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

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

A valuation $u$ satisfies a predicate $\phi$ if substituting the values of the variables in $u$ in $\phi$ makes it evaluate to True. We write $u \models \phi$

Examples: $u = \langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0 \rangle \rangle$; $v = \langle x \mapsto \langle 0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0 \rangle \rangle$

- $u \models \phi_2$, $(u \not\models \phi_1)$, $v \models \phi_1$ and $v \not\models \phi_2$

$[[\phi]]$: set of all valuations that satisfy $\phi$

- $[[\phi_1]] = \{ \langle x \mapsto \langle 1 \mapsto 1, i \mapsto c_i \rangle_{i=0,2,\ldots,5} \rangle | c_i \in \{0, \ldots, 7\} \}$
- $[[\phi_2]] = \{ \langle x \mapsto \langle 0 \mapsto 0, 1 \mapsto 0, 2 \mapsto 0, 3 \mapsto 0, 4 \mapsto 0, 5 \mapsto 0 \rangle \rangle \}$

- $\Theta \subseteq val(x)$ is the set of initial states of the automaton; often specified by a predicate over $X$
Actions

- **actions** section defines the set of Actions of the automaton
- **Examples**
  - **actions** `update(i:ID)`
    - defines $A = \{update[0], \ldots, update[5]\}$
  - **actions** `brakeOn`, `brakeOff`
    - defines $A = \{brakeOn, brakeOff\}$
Transitions defined by preconditions and effects

\[ \mathcal{D} \subseteq \text{val}(X) \times A \times \text{val}(X) \] is the set of transitions

\[ \mathcal{D} = \{ \langle u, a, u' \rangle \mid \text{such that } u \models Pre_a \text{ and } \langle u, u' \rangle \models Eff_a \} \]

\( \langle u, a, u' \rangle \in \mathcal{D} \) is written as \( u \rightarrow_a u' \)

Example:

**internal update(i:ID)**

- **pre** \( i = 0 \land x[i] = x[n-1] \)
- **eff** \( x[i] := x[i] + 1 \mod k; \)

**internal update(i:ID)**

- **pre** \( i \neq 0 \land x[i] \neq x[i-1] \)
- **eff** \( x[i] := x[i-1]; \)

\( \langle u, \text{update}(i), u' \rangle \in \mathcal{D} \) iff

(a) \( i = 0 \land u[x[0]] = u[x[5]] \land u'[x[0]] = u[x[0] + 1 \mod K] \lor \)

(b) \( i \neq 0 \land u[x[i]] \neq u[x[i-1]] \land u'[x[i]] = u[x[i-1]] \)
Executions, Reachability, and Invariants

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

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 $\alpha = u_0a_1u_1a_2u_3...$ such that:

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

For a finite execution, $\alpha = u_0a_1u_1a_2u_3$ the last state $\alpha.lstate = u_3$, the first state $\alpha.fstate = u_0$, and the length of the execution is 3.

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

For an action $a \in A$, $\text{Pre}(a)$ is the formula defining its precondition, and $\text{Eff}(a)$ is the relation defining the effect.

States satisfying precondition are said to *enable* the action

In general $\text{eff}(a)$ could be a relation, but for this example it is a function
Nondeterminism

• Multiple post-states from the same action (internal)
• Multiple actions enabled from the same state (external)
Reachable states and invariants

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

$\text{Reach}_A(\Theta)$: set of states reachable from $\Theta$ by automaton $A$

An **invariant** is a set of states $I$ such that $\text{Reach}_A \subseteq I$
Candidate invariants for token Ring

$I_1$: “Exactly one process has the token”.
$I_{\geq 1}$: “At least one process has a token”.
$I_3$: “All processes have values at most $K-1$”.

For any automaton

\[ \text{val}(X): \text{All states e.g. } I_3 \]

\[ \text{Invariant e.g. } I_1 \]

\[ \text{Reach}_A(u_0) \]

\[ u_0 \]
Reachability as graph search

• Q1. Given $A$, is a state $u \in val(X)$ reachable?

• Define a graph $G_A = \langle V, E \rangle$ where
  • $V = val(X)$
  • $E = \{(u,u')|\exists a \in A, u \rightarrow_a u'\}$

• Q2. Does there exist a path in $G_A$ from any state in $\Theta$ to $u$?

• Perform DFS/BFS on $G_A$
Theorem 7.1. Given an automaton $\mathcal{A} = \langle X, \Theta, A, D \rangle$ and a set of states $I \subseteq val(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_{\alpha} x'$ and $x \in I$ implies $x' \in I$

then $I$ is an (inductive) invariant of $\mathcal{A}$. That is $\text{Reach}_{\mathcal{A}}(\Theta) \subseteq I$. 

Proving invariants by induction (Chapter 7)

Theorem 7.1. Given a automaton $\mathcal{A} = \langle X, \Theta, A, \mathcal{D} \rangle$ and a set of states $I \subseteq \text{val}(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then $I$ is an (inductive) invariant of $\mathcal{A}$. That is $\text{Reach}_\mathcal{A}(\Theta) \subseteq I$.

Proof. Consider any reachable state $x$. By the definition of a reachable state, there exists an execution $\alpha$ of $\mathcal{A}$ such that $\alpha.\text{lstate} = x$.

We proceed by induction on the length $\alpha$

For the base case, $\alpha$ consists of a single starting state $\alpha = x \in \Theta$, and by the Start condition, $x \in I$.

For the inductive step, $\alpha = \alpha' a x$ where $a \in A$. By the induction hypothesis, we know that $\alpha'.\text{lstate} \in I$.

Invoking Transition closure on $\alpha'.\text{lstate} \rightarrow_a x$ we obtain $x \in I$. QED
Theorem 7.1. Given a automaton $A = \langle X, \Theta, A, D \rangle$ and a set of states $I \subseteq \text{val}(X)$ if:

- (Start condition) for any $x \in \Theta$ implies $x \in I$, and
- (Transition closure) for any $x \rightarrow_a x'$ and $x \in I$ implies $x' \in I$

then $I$ is an (inductive) invariant of $A$. That is $\text{Reach}_A(\Theta) \subseteq I$.

$I_1$: “Exactly one process has the token”.

(Start condition): Fix a $x \in \Theta$. $x \models \forall i \ x[i] = 0$ therefore $x \models I_1$

(Transition closure): Fix a $x \rightarrow_a x'$ such that $x \in I$.

Two cases to consider.

1. If $a = \text{update}(0)$ then
   a) since $x \models \text{Pre(update}(0))$ it follows that $x[x[0] = x[N - 1]$
   b) since $x \models I_1$ it follows that $\forall i > 0 \ x[x[i] = x[x[i - 1]$
   c) $x'[x[0] \neq x'[N - 1]$ by applying (a) and $\text{Eff(update}(0))$ to $x$
   d) $x'[x[1] \neq x'[x[0]]$ by applying (b) $\text{Eff(update}(0))$ to $x$
   e) $\forall i > 1 \ x'[x[i] = x'[x[i - 1]]$ by applying (b) $\text{Eff(update}(0))$ to $x$

   Therefore $x' \models I$.

2. If $a = \text{update}(i)$, $i > 0$ then fix arbitrary $i > 0$ ... (do it as an exercise)

From above **Theorem** it follows that $I_1$ is an invariant of DijstraTR
Assignments

• Read. Modeling computation: Chapter 2 of CPSBook, first part of Chapter 7 (7.1, 7.2), and section on SAT/SMT (7.5)

• HW1 due 02/10

• IOA Specification language: Appendix C of CPSBook

• Keep thinking about class project!