EECS 20

Lecture 38 (April 27, 2001)

Tom Henzinger

Revised by Matteo Zavatteri and Tiziano Villa, Fall 2020

Transducive System

$\text{transduciveSystem} : \text{Values} \rightarrow \text{Values}$

Reactive System

$\text{reactiveSystem} : [\,\text{Time} \rightarrow \text{Values}\,] \rightarrow [\,\text{Time} \rightarrow \text{Values}\,]$

Discrete time: $\quad$ Time = $Nats_0$ = { 0, 1, 2, ... }

Continuous time: $\quad$ Time = $Reals_+$ = { $x \in$ Reals | $x \geq 0$ }

A reactive system

$$F : [\, Time \to Values \,] \to [\, Time \to Values \,]$$

is memory-free

iff

there exists a transducive system

$$f : Values \to Values$$

such that

$$\forall\, x \in [\, Time \to Values \,],\ \forall\, y \in Time,$$

$$(\, F(x)\,)(y) = f(\, x(y)\,).$$

A reactive system

$$F : [\,\text{Time} \to \text{Values}\,] \to [\,\text{Time} \to \text{Values}\,]$$

is causal

iff

$\forall\, x, y \in [\,\text{Time} \to \text{Values}\,],\ \forall\, z \in \text{Time},$

if $\quad (\forall\, t \in \text{Time},\ t \le z \Rightarrow x(t) = y(t))$

then $\quad (F(x))(z) = (F(y))(z)$.

The Delay System

$$\text{Delay}_c : [\text{Time} \to \text{Values}] \to [\text{Time} \to \text{Values}]$$

such that $\forall x \in [\text{Time} \to \text{Values}], \forall y \in \text{Time},$

$$(\text{Delay}(x))(y) = \begin{cases} c & \text{if } y < 1 \\ x(y-1) & \text{if } y \geq 1 \end{cases}$$

Discrete-time delay over finite set of values :

finite memory

Continuous-time delay, or infinite set of values:

infinite memory

Legal Transducive Block Diagrams

-all components are transducive systems

-no cycles

e.g., combinational circuits

Legal Reactive Block Diagrams

-all components are memory-free or delay systems

-every cycle contains at least one delay

e.g., sequential circuits

Discrete-time reactive systems with

finite memory

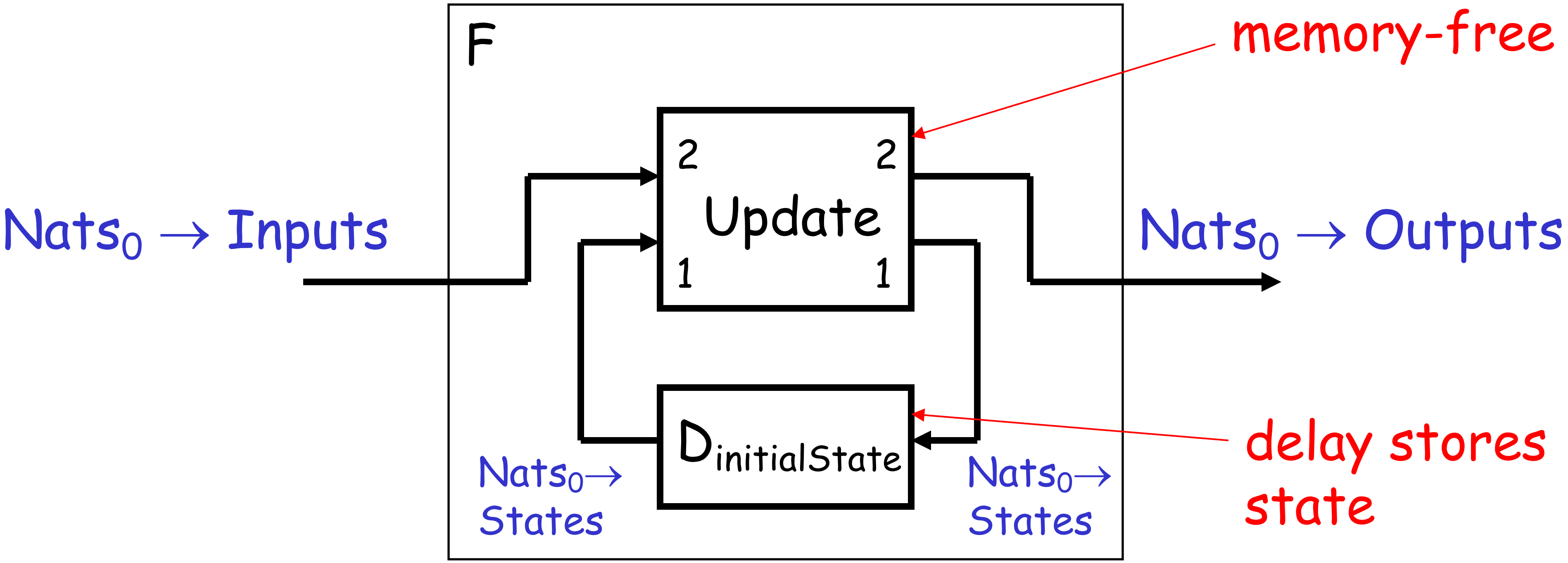
are naturally implemented as

finite state machines.

A Discrete-Time Reactive System

$$F : [\, Nats_0 \rightarrow Inputs \,] \rightarrow [\, Nats_0 \rightarrow Outputs \,]$$

State Machine Implementation



$$update: \ States \times Inputs \ \to \ States \times Outputs$$

$$initialState \in States$$

Deterministic State Machine

Inputs (set of possible input values)

Outputs (set of possible output values)

States (set of states)

$initialState \in States$

$update : States \times Inputs \rightarrow States \times Outputs$

Product of State Machines

Any block diagram of N state machines with the state spaces

States1, States2, … StatesN

can be implemented by a single state machine with the state space

States1 × States2 × … × StatesN .

This is called a "product machine".

Deterministic Reactive System:

for every input signal, there is exactly one output signal.

Function:

$$\text{DetSys} : [\ \text{Time} \to \text{Inputs}\ ] \to [\ \text{Time} \to \text{Outputs}\ ]$$

Nondeterministic Reactive System:

for every input signal, there is one or more output signals.

Binary relation:

$$\text{NondetSys} \subseteq [\ \text{Time} \to \text{Inputs}\ ] \times [\ \text{Time} \to \text{Outputs}\ ]$$

such that $\forall\ x \in [\ \text{Time} \to \text{Inputs}\ ]$,
$\exists\ y \in [\ \text{Time} \to \text{Outputs}\ ],\ (x,y) \in \text{NondetSys}$

Every pair $(x,y) \in \text{NondetSys}$ is called a behavior.

S1 is a more detailed description of S2;

S2 is an abstraction or property of S1.

System S1 refines system S2

iff

1. Time [S1] = Time [S2] ,
2. Inputs [S1] = Inputs [S2] ,
3. Outputs [S1] = Outputs [S2] ,
4. Behaviors [S1] $\subseteq$ Behaviors [S2] .

Systems S1 and S2 are equivalent

iff

1. Time [S1] = Time [S2] ,
2. Inputs [S1] = Inputs [S2] ,
3. Outputs [S1] = Outputs [S2] ,
4. Behaviors [S1] = Behaviors [S2] .

Nondeterministic State Machine

Inputs

Outputs

States

possibleInitialStates $\subseteq$ States

possibleUpdates :

$$\text{States} \times \text{Inputs} \to P(\text{States} \times \text{Outputs}) \setminus \emptyset$$

receptiveness (i.e., machine must be prepared to accept every input)

State Machines

Deterministic

$\Downarrow \qquad \not\Uparrow$

Output-deterministic

$\Downarrow \qquad \not\Uparrow$

Nondeterministic

A state machine is deterministic

iff

1. there is only one initial state, and
2. for every state and every input, there is only one successor state.

A state machine is output-deterministic

iff

1. there is only one initial state, and
2. for every state and every input-output pair, there is only one successor state.

For deterministic M2 :

M1 is simulated by M2 iff M1 is equivalent to M2.

For output-deterministic M2 :

M1 is simulated by M2 iff M1 refines M2.

For nondeterministic M2 :

M1 is simulated by M2 implies M1 refines M2.

relation between finitely many states

condition on infinitely many behaviors

A binary relation $S \subseteq \text{States}[M1] \times \text{States}[M2]$ is a simulation of M1 by M2

iff

1. $\forall\, p \in \text{possibleInitialStates}[M1]$,

   $\exists\, q \in \text{possibleInitialStates}[M2],\ (p, q) \in S$ and

2. $\forall\, p \in \text{States}[M1],\ \forall\, q \in \text{States}[M2]$,

   if $(p, q) \in S$,

   then $\forall\, x \in \text{Inputs},\ \forall\, y \in \text{Outputs},\ \forall\, p' \in \text{States}[M1]$,

   if $(p', y) \in \text{possibleUpdates}[M1](p, x)$

   then $\exists\, q' \in \text{States}[M2]$,

   $(q', y) \in \text{possibleUpdates}[M2](q, x)$ and

   $(p', q') \in S$.

To check if M1 refines M2,
check if M1 is simulated by det(M2):

M1 refines M2

iff

M1 refines det(M2)

iff

M1 is simulated by det(M2).

output-deterministic

If M2 is an output-deterministic state machine, then a simulation S of M1 by M2 can be found as follows:

1. If $p \in$ possibleInitialStates [M1] and

   possibleInitialStates [M2] $= \{ q \}$,

   then $(p,q) \in S$.

2. If $(p,q) \in S$ and

   $(p',y) \in$ possibleUpdates [M1] (p,x) and

   possibleUpdates [M2] $(q,x) = \{ (q',y) \}$,

   then $(p',q') \in S$.

Output-Determinization

Given: nondeterministic state machine M

Find: output-deterministic state machine $det(M)$ that is equivalent to M

$$\text{Inputs}\,[det(M)] = \text{Inputs}\,[M]$$

$$\text{Outputs}\,[det(M)] = \text{Outputs}\,[M]$$

The Subset Construction

Let initialState [det(M)] = possibleInitialStates [M] ;

Let States [det(M)] = { initialState [det(M)] } ;

Repeat as long as new transitions can be added to det(M) :

Choose $P \in$ States [det(M)] and $(x,y) \in$ Inputs $\times$ Outputs ;

Let $Q = \{ q \in$ States [M] $\mid \exists\, p \in P,\ (q,y) \in$ possibleUpdates [M] $(p,x) \}$;

If $Q \neq \emptyset$ then

Let States [det(M)] = States [det(M)] $\cup \{Q\}$;

Let update [det(M)] $(P,x) = (Q,y)$.

Minimization Algorithm

Input : nondeterministic state machine M

Output : minimize (M), the state machine with the fewest states that is bisimilar to M

(the result is unique up to renaming of states)

A binary relation $B \subseteq States[M1] \times States[M2]$ is a bisimulation between M1 and M2

iff

A1. $\forall\, p \in possibleInitialStates[M1]$,

$\exists\, q \in possibleInitialStates[M2]$, $(p, q) \in B$, and

A2. $\forall\, p \in States[M1]$, $\forall\, q \in States[M2]$,

if $(p, q) \in B$,

then $\forall\, x \in Inputs$, $\forall\, y \in Outputs$, $\forall\, p' \in States[M1]$,

if $(p', y) \in possibleUpdates[M1](p, x)$

then $\exists\, q' \in States[M2]$,

$(q', y) \in possibleUpdates[M2](q, x)$ and

$(p', q') \in B$, and

and

B1. $\forall\ q \in \text{possibleInitialStates}\ [M2]\ ,$

$\exists\ p \in \text{possibleInitialStates}\ [M1],\ (\ p, q\ ) \in B,$ and

B2. $\forall\ p \in \text{States}\ [M1]\ ,\ \forall\ q \in \text{States}\ [M2]\ ,$

if $(\ p, q\ ) \in B\ ,$

then $\forall\ x \in \text{Inputs}\ ,\ \forall\ y \in \text{Outputs}\ ,\ \forall\ q' \in \text{States}\ [M2]\ ,$

if $(\ q', y\ ) \in \text{possibleUpdates}\ [M2]\ (\ q, x\ )$

then $\exists\ p' \in \text{States}\ [M1]\ ,$

$(\ p', y\ ) \in \text{possibleUpdates}\ [M1]\ (\ p, x\ )$ and

$(\ p', q'\ ) \in B\ .$

For nondeterministic state machines M1 and M2,

M1 is equivalent to M2

$\not\Downarrow \quad \Uparrow$

M1 simulates M2 and M2 simulates M1

$\not\Downarrow \quad \Uparrow$

M1 and M2 are bisimilar.

For output-deterministic state machines M1 and M2,

M1 is equivalent to M2

$\Downarrow \quad \Uparrow$

M1 and M2 are bisimilar.

Minimization Algorithm

1. Let Q be set of all reachable states of M.
2. Maintain a set P of state sets:

   Initially let P = { Q }.

   Repeat until no longer possible: split P.
3. When done, every state set in P represents a single state of the smallest state machine bisimilar to M.

Split P

If there exist

two state sets $R \in P$ and $R' \in P$

two states $r1 \in R$ and $r2 \in R$

an input $x \in Inputs$

an output $y \in Outputs$

such that

$\exists\, r' \in R',\ (r', y) \in possibleUpdates\,(r1, x)$ and

$\forall\, r' \in R',\ (r', y) \notin possibleUpdates\,(r2, x)$

then

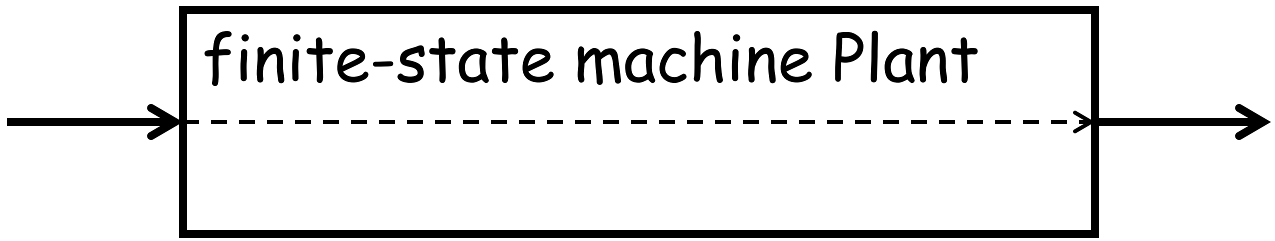
let $R1 = \{\, r \in R \mid \exists\, r' \in R',\ (r', y) \in possibleUpdates\,(r, x)\,\}$;

let $R2 = R \setminus R1$;

let $P = (P \setminus \{R\}) \cup \{R1, R2\}$.

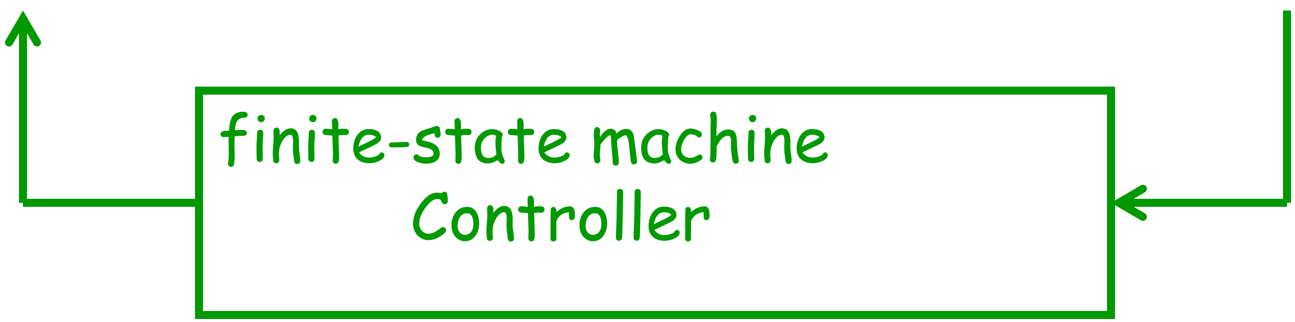
The Finite-State Safety Control Problem

Given

1. 

2. set Error of states of Plant

Find

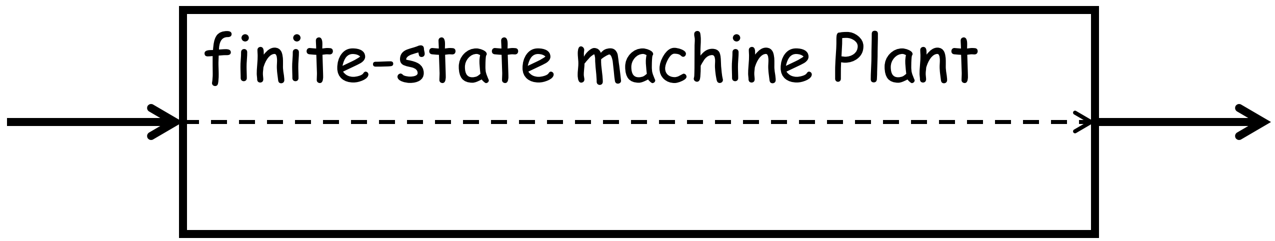


such that the composite system never enters a state in Error

The Finite-State Progress Control Problem

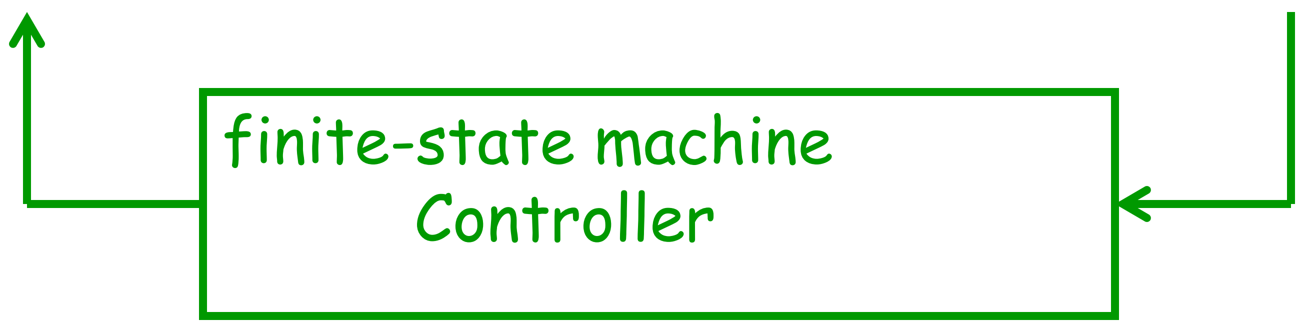
Given

1.



2. set Target of states of Plant

Find



such that the composite system is guaranteed to enter a state in Target

Compute the safety-uncontrollable states of Plant

1. Every state in Error is safety-uncontrollable.
2. For all states s,

   if for all inputs i
   there exist a safety-uncontrollable state s′ and an output o
   such that $(s', o) \in$ possibleUpdates (s, i)

   then s is safety-uncontrollable.

Compute the progress-controllable states of Plant

1. Every state in Target is progress-controllable.
2. For all states s,

   if there exists an input i
   for all states s' and outputs o
   if $(s',o) \in$ possibleUpdates (s,i)
   then s' is progress-controllable

   then s is progress-controllable.

Typical Exam Questions

A. Convert between the following system representations:

1. Mathematical input-output definition
2. Transition diagram
3. Block diagram

B. Apply the following algorithms on state machines:

1. Product construction
2. Subset construction
3. Check for existence of a simulation
4. Minimization
5. Compute controllable states

C. Explain the following concepts:

1. Memory-free vs. finite-state vs. infinite-state
2. Equivalence/refinement vs. simulation vs. bisimulation
3. Safety vs. progress control