Synchronous Dataflow Programming

CS684: Embedded Systems
Topic 5

Paritosh Pandya

Indian Institute of Technology, Bombay

February 9, 2023

Summary

Multi-mode controller as FSA + Dataflow Equations.

- Automaton can be in one active state at each cycle.
- Each state is a mode with associated set of equations. Equations of the active state are applied.
- Each state is a name space and clock domain. $pre(x)$ refers to mode local copy of x. Construct $last(x)$ allows values to be shared between modes.
- Each state has outgoing until (or weak) transitions. After executing the equations of the active state, the guards of its until transitions are evaluated to decide the next state.
- Thus, weak state transition takes effect from the next cycle.
- In a continue type transition, the state change occurs without resetting the equations of the target state.
- In a then type transition, the state change occurs with resetting of the equations of the target state.

Structure of a Node with Concurrent Automata

```
node myautomaton () returns (y:int)
let
    y = x + z;

    automaton --autA
      state S1 do x = 10 -> pre(x)+1;
          ...
    end

    automaton --autB
      state T1 do z = 20;
          ...
      end
tel
```

Equations and automata all execute in parallel in lock-step (synchronous) manner.

Concurrent Automata: Example

```
node myautomaton () returns (ping,pong : bool)
let
automaton  -- A_ping                automaton -- A_pong
    state S1a                          state S1b
        do   ping = true                    do   pong = false
    until true then S2a                until ping then S2b
    state S2a                          state S2b
        do   ping = false                   do   pong = true
    until pong then S1a                until  true then S1b
end;                                end
                                    tel
```

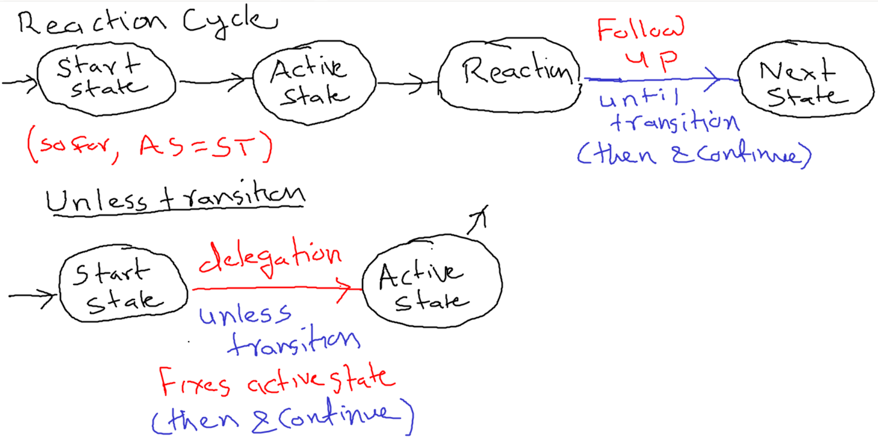
ST	$(S1a, S1b)$	$(S2a, S2b)$	$(S1a, S1b)$	$(S2a, S2b)$	$(S1a, S1b)$	...
ping	1	0	1	0	1	...
pong	0	1	0	1	0	...
NS	$(S2a, S2b)$	$(S1a, S1b)$	$(S2a, S2b)$	$(S1a, S1b)$	$(S2a, S2b)$	...

Automaton with until and reset transition (Revision)

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       until c then Down
       state Down
          do o = 150 -> -2 * i; stup = false;
       until c then Up
     end
  tel
```

ST	U	U	U	U	D	D	D	D	U	U	U	D	D	...
i	4	4	3	3	3	3	3	3	3	3	3	3	3	...
c	0	0	0	1	0	0	0	1	0	0	1	0	0	...
o	60	5	4	4	150	−6	−6	−6	60	4	4	150	−6	...
stup	1	1	1	1	0	0	0	0	1	1	1	0	0	...
NS	U	U	U	D	D	D	D	U	U	U	D	D	D	...

Reaction Cycle of an Automaton



Unless Transitions: Motivation

Unless transitions are also called strong transitions.

- They allow delegation of reacting in current cycle to another state.
- A state can have one or more unless transitions.
- Guard of an unless transition is evaluated before looking at the equations (i.e. before reacting).
  Hence, guard cannot use current values of equations.
- If guard is true, control moves to the target state in the same cycle.
- Thus, target state becomes the active state. Its equations are applied.
- If the guards of all unless transitions are false, the current state remains the active state.
- After the reaction, the until transitions of the active state decides the next state.
- At most one delegation is allowed per cycle.

Reaction Cycle

- Start State
- Delegation Apply unless transition to determine the active state of the current cycle.
- Reaction Equations of the active state are applied to compute output from input.
- Followup AFTER the reaction, apply the until transition of the active state:

  Guard of each until transition is evaluated.

  Guard can refer to outputs the equations.

  If guard is true the transition is taken and next state is changed to target state.
- Next State: this is the start state of the next cycle.
- For both unless and until transitions, the equations are reset for **then** type transition. They are not reset for a **continue** type transition.

Unless Transitions: Example

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       unless c then Down
       state Down
          do o = 150 -> -2 * i; stup = false;
       unless c then Up
     end
  tel
```

ST	U	U	U	U	D	D	D	U	...
AS	U	U	U	D	D	D	U	U	...
i	4	4	4	4	4	4	4	4	...
c	0	0	0	1	0	0	1	0	...
o	60	5	5	150	−8	−8	60	5	...
stup	1	1	1	0	0	0	1	1	...
NS	U	U	U	D	D	D	U	U	...

Mixing unless and until Transitions: Example

```
node myautomaton(i : int; c: bool) returns (o: int; stup:bool)
  let
     automaton
       state Up
          do o = 60 -> i+1; stup = true;
       unless c then Down
       state Down
                                              False
          do o = 150 -> -2 * i; stup = true;
       until c then Up
     end
  tel
```

ST	U	U	U	U	U			U				...
AS	U	U	U	D				D				...
i	4	4	4	4	4	4	4	4	4	4	4	...
c	0	0	0	1	0	0	0	1	1	0	0	...
o	60	5	5	150	60	5	5	150	150	60	5	...
stup	1	1	1	0	1	1	1	0	0	1	1	...
NS	U	U	U	U			U	U				...

Example: Farm Road Traffic Light Controller

A farm road (or side road) crosses a main road. Traffic light controller must turn on or off the lights maingreen, mainred, sidegreen, sidered. An input carwait is true if a car is waiting on the farm road. Input second is the timer input which becomes true for one clock cycle every one second. Thus the count of "second" gives how much time has elapsed.

Controller: Farm Road Traffic Light

```
node traffic(carwait,second:bool)
      returns (maingreen, mainred,sidegreen,sidered:bool)

let
  automaton
    state Maingreen
       do

       until
    state Sidegreen
       do

       until
  end
tel
```

```
node traffic(carwait,second:bool)
      returns (maingreen, mainred,sidegreen,sidered:bool)
var timegreen:int;
let
  automaton
    state Maingreen
       do

           maingreen = true; mainred = false;
           sidegreen = false; sidered = true;
       until  ?? then  Sidegreen
    state Sidegreen
       do

           maingreen = false; mainred = true;
           sidegreen = true; sidered = false;
       until ?? then  Maingreen
  end
tel
```

```
node traffic(carwait,second:bool)
      returns (maingreen, mainred,sidegreen,sidered:bool)
var timegreen:int;
let
  automaton
    state Maingreen
       do  timegreen = 180 -> if (((pre(timegreen)) > 0) and second) then
                                     pre(timegreen)-1  else pre(timegreen);
           maingreen = true; mainred = false;
           sidegreen = false; sidered = true;
       until (timegreen <= 0) then Sidegreen
    state Sidegreen
       do
           timegreen = 60 ->
                if (((pre(timegreen)) > 0) and second) then pre(timegreen)-
                          else pre(timegreen);
           maingreen = false; mainred = true;
           sidegreen = true; sidered = false;
       until (timegreen <=0) then Maingreen
  end
tel
```

Controller: Farm Road Traffic Light

```
node traffic(carwait,second:bool)
      returns (maingreen, mainred,sidegreen,sidered:bool)
var timegreen:int;
let
  automaton
    state Maingreen
       do  timegreen = 180 -> if (((pre(timegreen)) > 0) and second) then
                                     pre(timegreen)-1  else pre(timegreen);
           maingreen = true; mainred = false;
           sidegreen = false; sidered = true;
       until ((timegreen <= 0) and carwait) then Sidegreen
    state Sidegreen
       do
           timegreen = 60 ->
                if (((pre(timegreen)) > 0) and second) then pre(timegreen)-
                           else pre(timegreen);
           maingreen = false; mainred = true;
           sidegreen = true; sidered = false;
       until ((timegreen <=0)  and not carwait) then Maingreen
  end
```

Conclusion

Synchronous Dataflow Programming using language Heptagon:

- Finite State Automata with Synchronous Data Flow Equationes.
- Automata provide modes of operation.
  Dataflow equations define equations for transforming input into output.
- Highly complex modes of operation can be programmed:
  Hierachy, concurrency, communication with shared flows.
- A heptagon program (node) compiles into a reactive kernel. Target C and Java.

A modelling langauge.

Desirable Feature

- Clean Semantics
- Deterministic execution: easy to certify.
- Simulation, testing and verification tools.
- Qualified code generator,
- Efficient Code
- Industrial Strength: Used widely in Nuclear reactor control, Aerospace and railway industries.