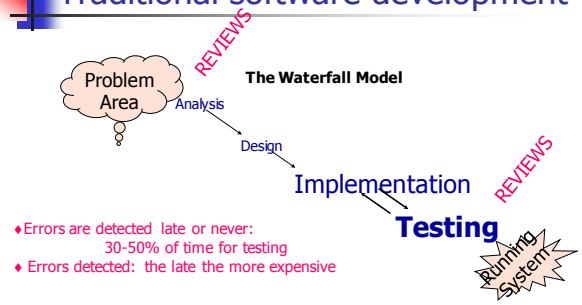
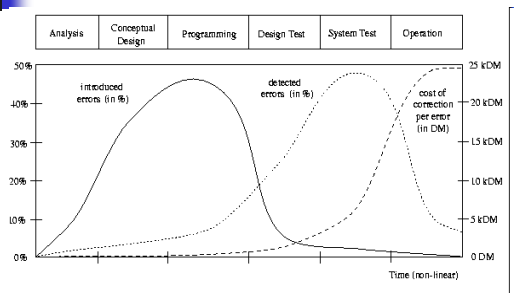
Modeling Real-Time Systems

Traditional software development



- Errors are detected late or never: 30-50% of time for testing
- Errors detected: the late the more expensive

Introducing, Detecting and Correcting errors



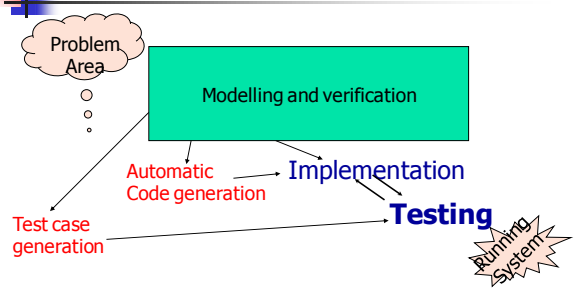
Finding errors as early as possible!

HOW?

Modeling and Verification

- "modeling" is a design process: describe systematically the abstract behaviour of a system
  - It is to create a model or a design proposal of the system to be developed
- "verification" is to check whether the model satisfies given requirement specifications
  - It is similar to testing, but testing on the model

Software development

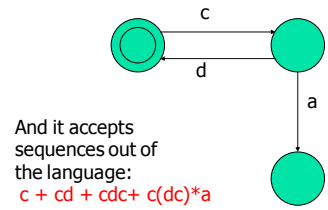


Basic ideas

- Modeling
  - Use state-machines, also known as automata, to describe system components
  - A system will be a network of automata
- Verification
  - Check properties of the automata using software tools like UPPAAL

7

Finite State Automata



It has 3 states and 3 transitions

And it accepts sequences out of the language:
c + cd + cdc+ c(dc)*a

8

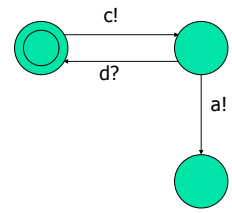
Automata as tasks

- We view an automaton as a task where the action labels stand for synchronization actions e.g. Ada's rendezvous
- For example, the previous example could be a network protocol where c and d stand for "connect !" and "disconnect?" and a for "abort!"

9

Input and Output Actions

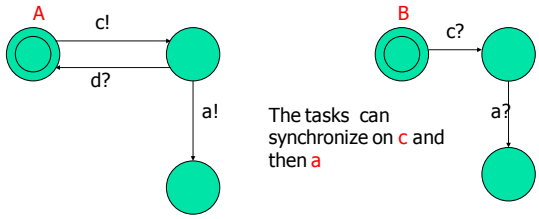
We use ? and ! to denote the pairs of complementary actions in rendezvous communication



10

Networks of automata

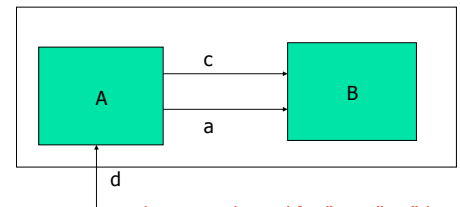
We use ? and ! to denote the pairs of complementary actions in rendezvous communication



The tasks can synchronize on c and then a

11

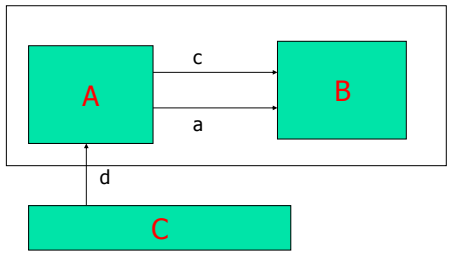
The static architecture of a concurrent system: A || B



where c, a, d stand for "ports" or "channels"

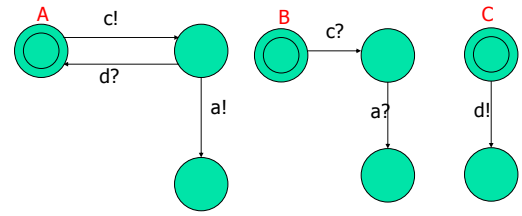
12

The static architecture of a concurrent system: A || B || C



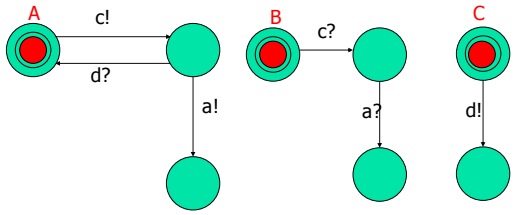
13

Networks of automata



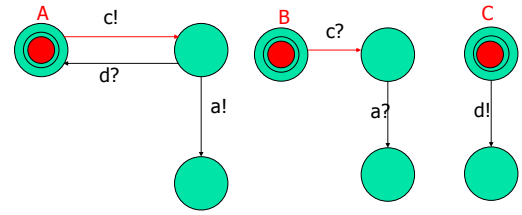
14

Networks of automata: initial state



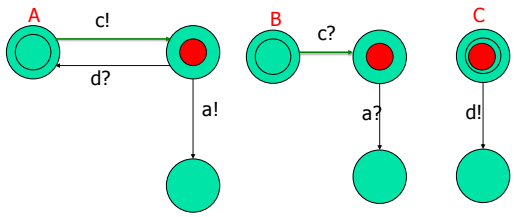
15

Networks of automata: enabled transition



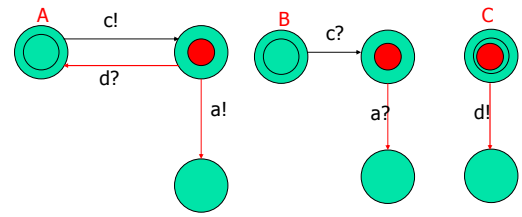
16

Networks of automata: transition taken



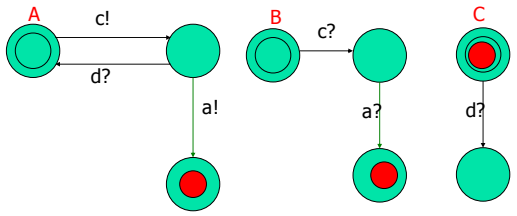
17

Networks of automata: nondeterministic choices (transitions)



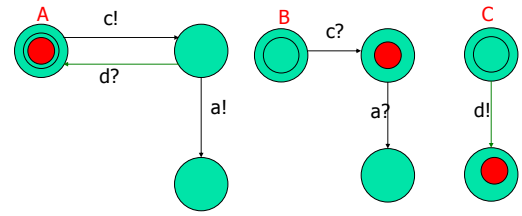
18

Networks of automata: non-deterministic transitions (1)



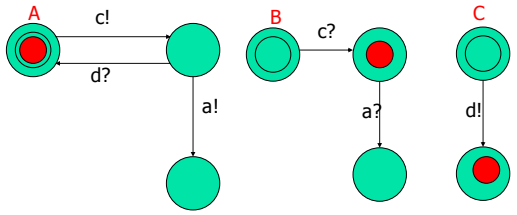
19

Networks of automata: non-deterministic transition (2)



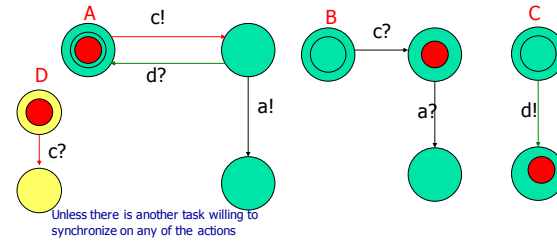
20

Networks of automata: deadlock!



21

Networks of automata: deadlock!



22

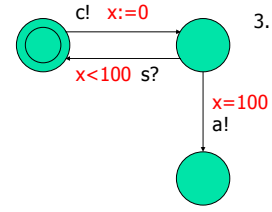
Clocks and timing constraints

- Now we assume that the system has a finite number of clocks
  - The clocks start to increase from 0 when the system starts, and they run at the same rate
  - The clocks can be tested using e.g. x<100
  - The clocks can be reset to 0 on a transition using an assignment: x:=0

23

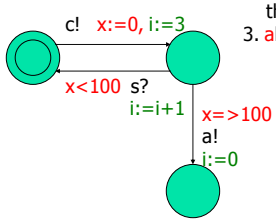
Timed automata: clocks

1. connect and reset x to 0
2. If succeed within 100ms, then go to initial
3. abort after 100ms !



24

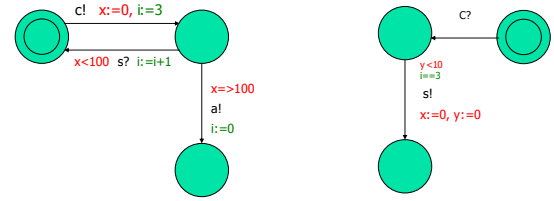
Timed automata: integers



1. connect and reset x to 0
2. If succeed within 100ms, then go to initial
3. abort after 100 ms !

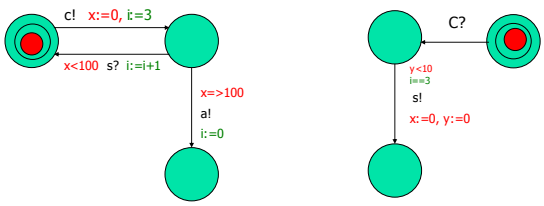
25

Network of Timed automata



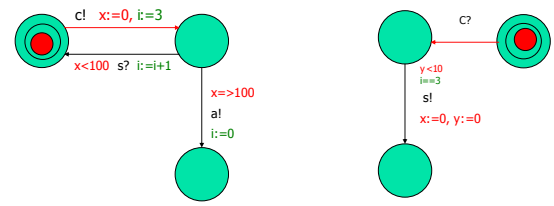
26

Network of Timed automata: initial states



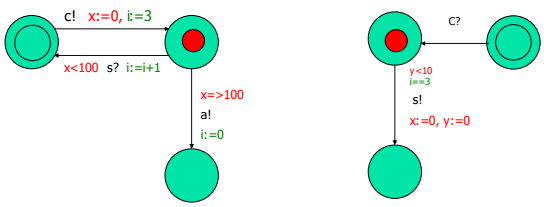
27

Network of Timed automata: enabled ransition



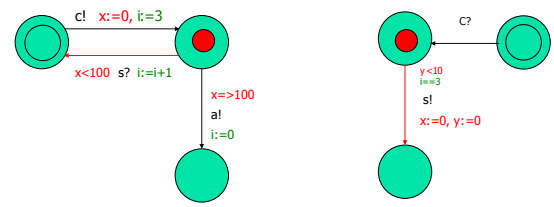
28

Network of Timed automata: transition taken



29

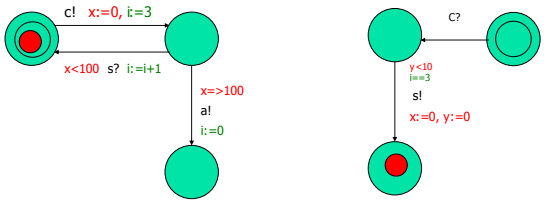
Network of Timed automata: enabled ransition



This depends on the values of x, y, and i

30

Network of Timed automata: transition taken



This depends on the values of x, y, and i

Timed automata (definition)

- a timed automaton is a finite graph
  - a finite many nodes N with an initial node
  - a finite many edges E between nodes
  - an edge may be labelled with three elements
    - guard
    - action (a?, a!, or nothing)
    - assignment

  (they may not appear)

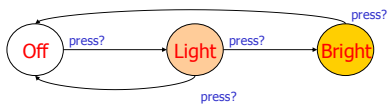
Guard

- a clock constraint
  - g ::= $x \leq n$ | $x \geq n$ | $x<n$ | $x>n$ | $g \wedge g$
  - where n is any natuaral number
- a predicate over data variables
  - "any logical expression" you may write in C

assignment

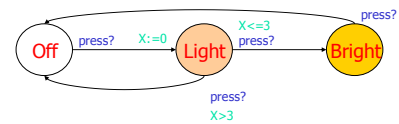
- a clock reset: x:=0 for any clock x
- a sequence of assignments in the form i:=e
  (or most of the programming constructs in C)

Intelligent Light Control



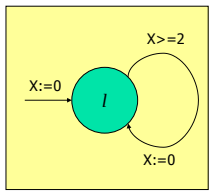
WANT: if press is issued twice quickly
then the light will get brighter; otherwise the light is turned off.

Intelligent Light Control (with timer)

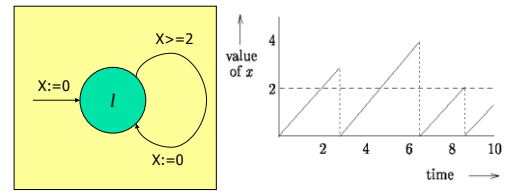


Solution: Add real-valued clock x

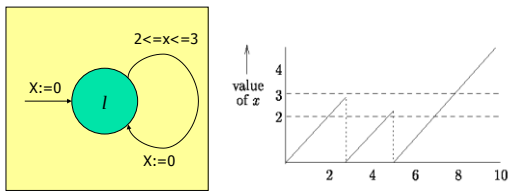
Timed Automata: Example



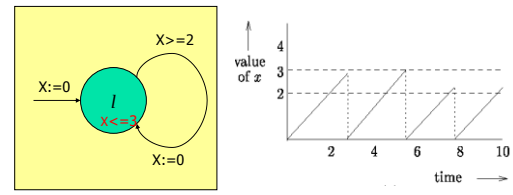
Timed Automata: Example



Timed Automata: Example



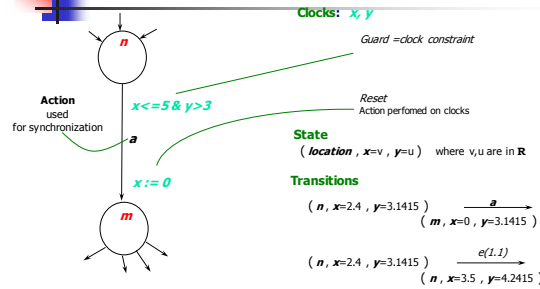
Timed Automata: Example



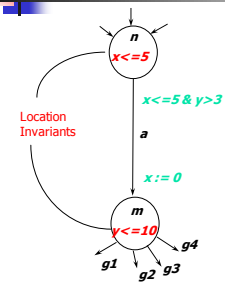
Timed Automata: Semantics

- States: (n, u) where
  - n stands for the current node or location
  - u stands for the current values of clocks and integers (i.e. the memory of a machine)
- Transitions: delay and actions
  - (n,u) moves to (n, u+d) after d time units
  - (n,u) moves to (m, v) when an action over channel "a" is taken, and v is the new values of clocks and integer variables

Timed Automata: Semantics



Timed Automata with *Invariants*



Clocks: *x, y*

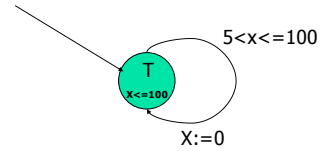
Transitions

(*n* , *x*=2.4 , *y*=3.1415) — *3.2* →

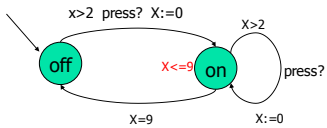
(*n* , *x*=2.4 , *y*=3.1415) — *1.1* → (*n* , *x*=3.5 , *y*=4.2415)

Invariants insure progress!!

Timed Automata: Example (task with jitter)

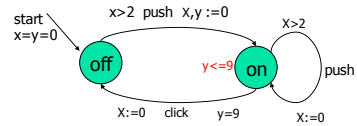


Timed Automata: Light Switch



- Switch may be turned on whenever at least 2 time units has elapsed since last "turn off"
- Light automatically switches off after 9 time units if it is not pressed.

Timed Automata: Example

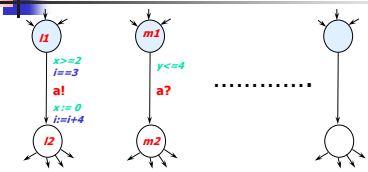


$$(off, x = y = 0) \xrightarrow{3.5} (off, x = y = 3.5) \xrightarrow{push}$$
$$(on, x = y = 0) \xrightarrow{\pi} (on, x = y = \pi) \xrightarrow{push}$$
$$(on, x = 0, y = \pi) \xrightarrow{3} (on, x = 3, y = \pi + 3) \xrightarrow{9-(\pi+3)}$$
$$(on, x = 9 - (\pi + 3), y = 9) \xrightarrow{click} (off, x = 0, y = 9) \ldots$$

Modeling Concurrency

- Products of automata
- Parallel composition

Networks of Timed Automata + Integer Variables + ...



Two-way synchronization on *complementary* actions.

Closed Systems!

Example transitions

(*l1, m1*,....., x=2, y=3.5, i=3,.....) ⟶ (*l2,m2*,.....,x=0, y=3.5, i=7,.....)

Modeling Ada Programs

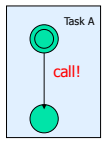
Rendezvous

```
task body A is
begin
...
B.Call;
...
end A
```

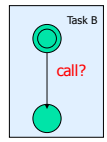
```
task body B is
begin
...
accept Call do
....
end Call
...
end A
```


Rendezvous

```
task body A is
begin
...
B.Call;
...
end A
```




```
task body B is
begin
...
accept Call do
....
end Call
...
end A
```



Buffer

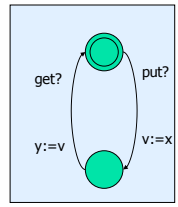
```
task buffer is
entry put(X: in integer)
entry get(x: out integer)
end;

task body buffer is
 v: integer;
begin
loop accept put(x: in integer) do v:= x end put;
      accpet get(x: out integer) do x:= v end get;
end loop;
end buffer;
---
buffer.put(...)   <---------------------------------------- other tasks (users)!!
buffer get(...)
----
```


Buffer

```
task buffer is
entry put(X: in integer)
entry get(x: out integer)
end;

task body buffer is
 v: integer;
begin
loop accept put(x: in integer) do v:= x end put;
      accpet get(y: out integer) do y:= v end get;
end loop;
end buffer;
---
buffer.put(...)   <---------------------------------------- other tasks (users)!!
buffer get(...)
----
```

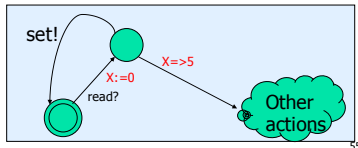


Conditional/Timed entry call

```
loop
   --get temperature
   select
        Controller.set(T);     -- set new temperature
   or  delay 5                  --other actions
   end select;
end loop;
```

Conditional/Timed entry call

```
loop
   --get temperature
   select
        Controller.set(T);     -- set new temperature
   or  delay 5                 --other actions
   end select;
end loop;
```

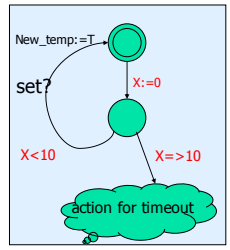


Timeout and message passing

```
loop
   select
        accept set(T : temperature) do
                New_temp:=T;
        end Call;
   or
        delay 10.0;
                --action for timeout
   end select;
   --other actions
end loop;
```


Timeout and message passing

```
loop
   select
        accept set(T : temperature) do
                New_temp:=T;
        end Call;
   or
        delay 10.0;
                --action for timeout
   end select;
   --other actions
end loop;
```



Periodic Activity

```
task body T is
  Interval : constant Duration := 5.0;
  Next_Time : Time;
begin
  Next_Time := Clock + Interval;
  loop
    Action;
    delay until Next_Time;
    Next_Time := Next_Time + Interval;
  end loop;
end T;
```


Periodic Activity

```
task body TaskP is
  T : constant Duration := 5.0;
  Next_Time : Time;
begin
  Next_Time := Clock + T;
  loop
    Action;
    delay until Next_Time;
    Next_Time := Next_Time + T;
  end loop;
end TaskP;
```