

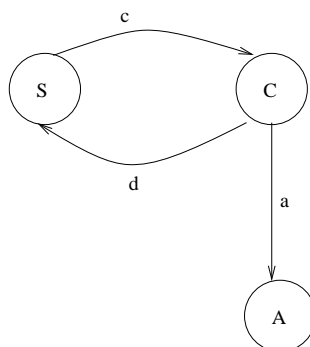
## Today's Topic

### Slide 1

- Today we be looking at *Timed Automata*.
- Timed Automata extend ordinary simple automata so that we reason about real time systems.
- Timed Automata complement the scheduling analysis of systems.

## Revision? Finite State Automata

Consider the following automata:



### Slide 2

It has 3 states and 3 transitions.

## States and Transition

### Slide 3

In a compiler course you are interested in Automata to recognise languages. In an automata you have some start state and a number of termination states.

In our example if **S** was the start state and **A** was the termination state then the language accepted by the automata would be:

$$c(dc)^*a$$

## Automata as Reactive Objects

### Slide 4

We need to view an automata as a reactive system, where the actions act as synchronisation points with other automata or the environment.

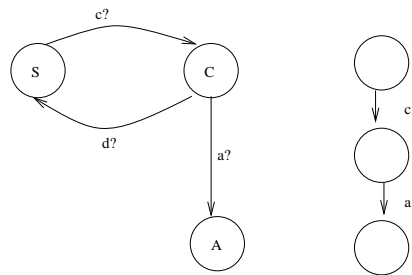
For example the previous example could be some sort of network protocol where the labels **c** and **d** represent connect and a disconnect events and **a** represents an abort event.

We shall refer a state-transition as an event, or sometimes we will refer to the label on a transition as an action.

## Networks of Automata

We use ? and ! for pairs of complementary actions that synchronise.

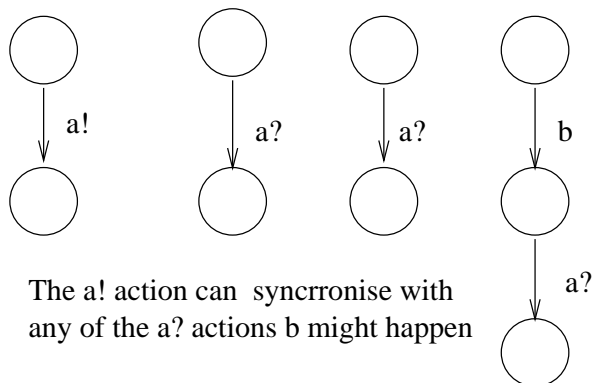
Slide 5



The two networks can first synchronise on the  $c$  action and then on the  $a$  action.

## Synchronisation and can be non-deterministic

Slide 6



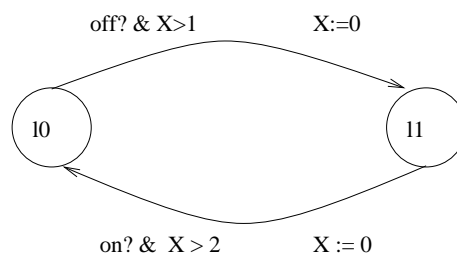
The  $a!$  action can synchronise with any of the  $a?$  actions  $b$  might happen

## Ideal Clocks

- Slide 7** We assume that we have a finite number of clocks, at the beginning of the run of our timed automata all the clocks are set to zero.
- Each clock proceeds at the same rate. At any point an automata can reset a clock.
- Note clocks can not be stopped and restarted only reset to zero.

## An example timed automata

**Slide 8**



This has two action `off` and `on`, but there are some differences with un-timed automata.

## Guards and Actions

A transition in a timed-automata is not only labelled with an action but also with a guard and a set of clocks to reset.

**Slide 9** In the example there are two transitions:

- $\text{off?} \ \& \ X > 1$  which can only happen if there is an  $\text{off}$  action to synchronise with and the clock  $X$  is greater than 1, upon doing this transition the clock  $X$  is reset to 1.
- $\text{on?} \ \& \ X > 2$  this transition happens when there is an  $\text{on}$  action to synchronise with and the clock  $X$  is greater than 2, again upon completing the transition the clock  $X$  is reset to zero.

## Timed Guards

In the definition and semantics of timed automata the following relations on a set of clocks,  $X_1, \dots, X_n$  are allowed:

- Slide 10**
- $X_i \sim c$
  - $X_i - X_j \sim c$

where  $c$  is an integer and  $\sim$  of one of  $<, >, \leq, \geq$ .

A guard  $g$  is simply a conjunction of constraints e.g.

$$X_1 > 1 \ \& \ X_3 - X_4 \leq 4$$

Given a set of clocks we shall refer to the set of allowed guards as  $\Phi$ .

## Definition of Timed Automata without Invariants

A *Timed Automata* is a tuple  $(S, A, C, \rightarrow, r, i)$  where

Slide 11

- $S$  is a set of *locations*
- $A$  is a set of *actions*
- $C$  is a set of *clocks*
- $\rightarrow \subseteq S \times (A \times \Phi) \times S$  is a set of edges.
- $r$  is a function from  $S$  to  $2^C$  called the resetting function
- $i \in S$  is some initial state.

## Semantics of Timed Automata

If  $(s, (a, g), s') \in \rightarrow$  and  $r(s') = \{X_1, \dots, X_k\}$  we write

Slide 12

$$s \xrightarrow{a, g, \{X_1, \dots, X_k\}} s'$$

The semantics of timed-automata is defined in terms of a transition system on pairs

$$(s, X_1 := x_1, \dots, X_n := x_n)$$

where  $s$  is from  $S$  and  $X_1 := x_1, \dots, X_n := x_n$  is some assignment of the clocks.

## Semantics of Timed Automata discrete transitions

### Slide 13

We say that there is a transition from a reachable state  $(s, X_1 := x_1, \dots, X_n := x_n)$  to  $(s', X_1 := x'_1, \dots, X_n := x'_n)$  if there exists some

$$s \xrightarrow{(a,g),\{X_{i_1}, \dots, X_{i_k}\}} s'$$

such that the guard  $g$  satisfies the assignment

$X_1 := x_1, \dots, X_n := x_n$  and  $x'_i = 0$  for  $x'_i \in \{X_{i_1}, \dots, X_{i_k}\}$ .

## Semantics of Timed Automata, time passing

### Slide 14

**Without Invariants:** Given a reachable state  $(s, X_1 := x_1, \dots, X_n := x_n)$  we say that time can pass to a state  $(s, X_1 := x'_1, \dots, X_n := x'_n)$  where for all  $x'_i = x_i + \delta$ .

**With Invariants:** We haven't specified invariants formally but an invariant is simply a guard on a location which has to be satisfied if time is to pass. With invariants it is very easy to stop time.

## Reachable States

Given the initial state

**Slide 15**

$$(s, X_1 := 0, \dots, X_n := 0)$$

of an automata many verification problems can be reduced to asking if some

$$(s', X_1 := x_1, \dots, X_n := x_n)$$

can be reached from the initial state by a sequence of transitions.

## Example

**Slide 16** Our timed-automata can do the following transitions (and more):

$$(l_0, X := 0) \rightarrow (l_0, X := 1.1)$$

$$(l_0, X := 1.1) \rightarrow (l_1, X := 0)$$

$$(l_1, X := 0) \rightarrow (l_1, X := 2.5) \rightarrow (l_0, X := 0)$$



## Networks of Automata

**Slide 17** Real-time systems are often concurrent, so far we have only considered single processes.

Given a network of automata  $A_1, \dots, A_n$  the state is defined as a tuple:

$$(s_1, \dots, s_n, X_1 := x_1, \dots, X_n = x_n)$$

## Networks of Automata :- The passage of time

For networks Time passes uniformly.

**Slide 18** (As long as no invariants are violated, see next lecture) Given a state

$$(s_1, \dots, s_n, X_1 := x_1, \dots, X_n = x_n)$$

there can be a transition to the state:

$$(s_1, \dots, s_n, X_1 := x_1 + \delta, \dots, X_n = x_n + \delta)$$

for ever  $\delta > 0$ .

## Networks of Automata :- Synchronisation

We need to find a pair of automata with complementary actions.

Remember synchronisation can be non-deterministic (there can be more than one pair that will synchronise).

**Slide 19**

If there exists a pair of automata  $A_i, A_j$  such that

$$(s_i, X_1 := x_1, \dots, X_n = x_n) \xrightarrow{a?,g} (s'_i, X_1 := x'_n, \dots, X_n = x'_n)$$

and

$$(s_j, X_1 := x_1, \dots, X_n = x_n) \xrightarrow{a!,g'} (s'_j, X_1 := x'_1, \dots, X_n = x'_n)$$

## Networks of Automata :- Synchronisation

Once we have found a pair that synchronises then they proceed in parallel and no-time passes.

**Slide 20**

$$(s_1, \dots, s_n, X_1 = x_1, \dots, X_n = x_n)$$

to

$$(s_1, \dots, s'_i, \dots, s'_j, \dots, s_n, X_1 := x'_1, \dots, X_n = x'_n)$$

that is the two automata synchronise on the action  $a$ .

## Model Checking



### Slide 21

Given a network of timed-automata it is possible to decide (that is write a computer program) if a given state is reachable from an initial state. In fact it is possible to decide a whole lot more.

This is quite an amazing fact! There are many possible states in between two states, in fact there is an uncountable number of them. But we can still symbolically represent the problem finitely.

Using reachability tests you can check many correctness properties of systems, for example you can check if something bad never happens.

## Times and Timed-Automata




### Slide 22

Times is a tool which has been developed here in Uppsala and in Aalborg, Denmark which checks reachability problems for networks of timed-automata.

This tool has been used to verify industrial sized case studies, such as the Mercel gear controller and an Audio control program (see course web page for more details).

In one of your laborations you are given a real-time controlled production cell to analysis using Timed-automata.

## Extra Properties of the Times System



- Slide 23**
- **Invariants in States** Time can only pass if some guard in a state is satisfied.
  - **Urgent Channels** A channel (that is a communication  $a!$  and  $a?$ ) can be declared urgent, that is it has to happen as soon as it is able to happen.