

Synchronous FSMs & Synchronous Languages

1. FSM and Extended FSM models
2. The State Explosion Problem
3. Hierarchical Concurrent FSMs
4. Time and Synchrony
5. Synchronous/Reactive Languages
6. How to Implement a Synchronous System? Problems.

Finite State Machines

- The system is characterised by *explicitly* depicting its states as well as the transitions from one state to another.
- One particular state is specified as the initial one
- States and transitions are in a finite number.
- Transitions are triggered by input events.
- Transitions generate outputs.
- FSMs are suitable for modeling control dominated reactive systems (react on inputs with specific outputs; not much computation)

Finite State Machines (cont'd)

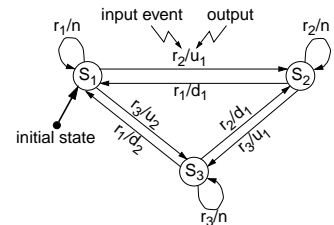
Extended FSM

- Variables can be associated to the FSM.
 - Changes of variables can be specified as actions associated to transitions.
 - Actions can also be specified to be performed at the entry and/or exit of a state.
 - Extended FSMs are suitable for systems which are both control and computation intensive.
- Guards (expressed as conditions) may be specified for transitions: The transition is performed when the associated event(s) occur and if the associated guard is true

FSM Example

Elevator controller

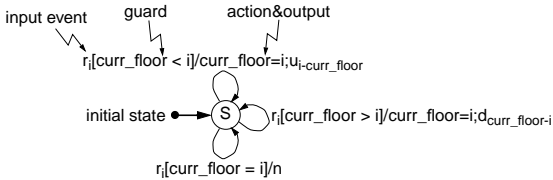
- Input events $\{r_1, r_2, r_3\}$
 - r_i : request from floor i .
- Outputs $\{d_2, d_1, n, u_1, u_2\}$
 - d_i : go down i floors
 - u_i : go up i floors
 - n : stay idle
- States $\{S_1, S_2, S_3\}$
 - S_i : elevator is at floor i .



FSM Example (cont'd)

Elevator controller with extended FSM

☞ We associate to the FSM a variable storing the current floor.



- You might wonder: *Do we really have one single state of the system? Of course not!*
The global system state is now encoded in the FSM state plus the value of the associated variable.



State Explosion

- Complex systems tend to have very large number of states. This particularly is the case in the presence of concurrency. The phenomenon is called *state explosion*.
- Every global state of a concurrent system must be represented individually \Rightarrow interleaving of independent actions leads to an exponential number of states.
- Expressing such a system as a FSM (or extended FSM) is very difficult.

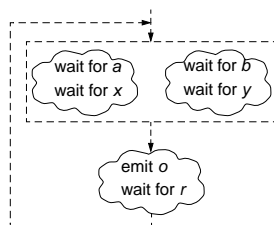


State Explosion (cont'd)

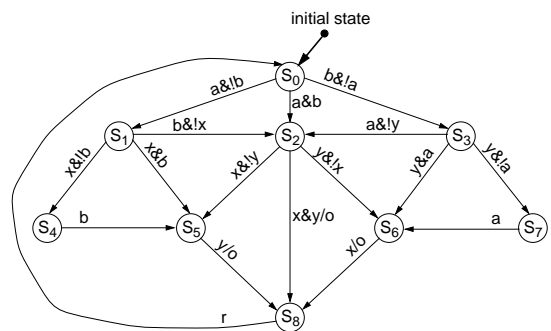
Example

After starting the system, it waits simultaneously for event *a* followed by *x*, and event *b* followed by *y*. Events can arrive in any order, except that *x* follows *a* and *y* follows *b*. Once the events are received, output *o* is emitted. Then the system waits for the reset signal *r* to return into the initial state.

- Input events {*a*, *b*, *x*, *y*, *r*}
- Output {*o*}
- States {*S*₀, *S*₁, ..., *S*₈}



State Explosion (cont'd)



Hierarchical Concurrent Finite State Machines

⇒ There are two important mechanisms that reduce the size of an FSM model:

1. Hierarchy
2. Concurrency

Important

- Using Hierarchy and concurrency we only reduce the size of the graphical or textual model; the intrinsic complexity - the number of states of the actual system - cannot be reduced.
- However, the difficulty of realising the model can be drastically reduced.



Hierarchical Concurrent Finite State Machines (cont'd)

Hierarchy

- A single state S can represent an enclosed state machine F:

Being in state S means that state machine F is active ⇒ the system is in one of the states of the state machine F (*or states*).

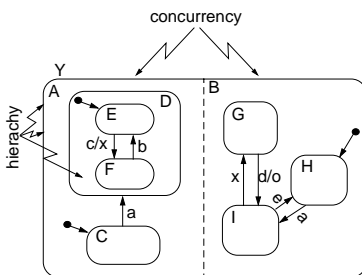
Concurrency

- Two or more state machines are viewed as being simultaneously active ⇒ the system is in one state of each parallel state machine simultaneously (*and states*).



Hierarchical Concurrent Finite State Machines (cont'd)

⇒ *Statecharts* is a graphical language for hierarchical concurrent FSMs

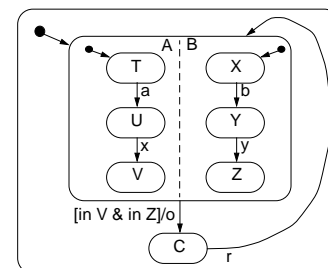


- System enters state Y ⇒ it will be in both A and B.
- A consists of D and C; C is initial state. D consists of E and F; E is initial state for D.
- B consists of G, I, and H; H is initial state for B.
- Entering Y, the system will be simultaneously in C and H; event a occurs ⇒ system transfers to E and I; event c occurs ⇒ system transfers to F, generates event x which triggers transition from I to G.



Hierarchical Concurrent Finite State Machines (cont'd)

The example from slide 7/8 using Statecharts:



FSMs: Time and Synchrony

☞ (hierarchical concurrent) FSMs are *synchronous models*.

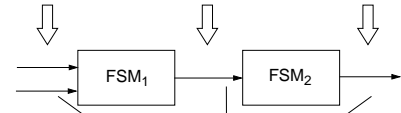
- **The synchrony hypothesis:**

The time is a sequence of instants (clock ticks) between which nothing interesting occurs. In each instant, some events (inputs) occur in the environment, and a reaction (output) is computed *instantly* by the modelled design.

- Computation and internal communication (between the FSMs composing the system, like event *x* in slide 11) take no time (compare to Discrete Event, where components can have arbitrary delays!).
- Events are either simultaneous (occur at the same clock tick) or one strictly precedes the other (as opposed to dataflow and Petri Nets where we only have a partial order of events).

FSMs: Time and Synchrony (cont'd)

input events internal events output events



synchronized: input events are at the same time with the internal and output events generated as response.

FSMs: Time and Synchrony (cont'd)

Question

Is the synchronous model sufficiently realistic to be used in practice?

Answer

For some applications yes!

It is the case when the following assumption is true:

The reaction time of the system (including internal communication) is neglectable compared to the rate of external events.

Why Do We Like Synchronous Models?

- A set of communicating, concurrent FSMs behaves exactly like one equivalent FSM (see hierarchical concurrent FSM in slide 12 vs. FSM in slide 8).



Models are deterministic.

It is possible to formally reason about models and to formally check certain properties of the model. This is important for certain class of applications (e.g. safety critical systems)

- It is possible to efficiently synthesise (compile) synchronous models to hardware or software.

Why Do We Not Like Synchronous Models?

- Reasoning, verification and synthesis based on synchronous models are meaningful and correct only as long as:
 - A completely synchronous implementation of the whole system is possible (the whole system acts similar to one single FSM).
 - We are sure that for the *implemented system* the assumption on slide 15 is true.
- Implementing *large* models as synchronous systems is expensive and often technically impossible.

Zero Delay Loops in Synchronous Models

Zero delay loops: Introduce similar problems as discussed with Discrete Event models (Fö 5, slides 25 to 28).



- Problems which could occur:
 - Nondeterministic behavior
 - Unstable systems (the output cannot be determined)

Zero Delay Loops in Synchronous Models (cont'd)

- The compiler detects the potential of such problems and rejects the model.
- We do not have this problem if the FSM is implemented as a Moore machine (output is delayed by one clock cycle with respect to input). But this is not real synchronous FSM semantic!

Synchronous/Reactive Languages

- Synchronous/reactive languages describe systems as a set of concurrently executing synchronized activities.
 - Communication is through signals.
 - Signals are either present or absent at a certain *tick*.
 - The presence of a signal is called an event and possibly carries a value.
- These language are semantically equivalent to the (extended hierarchical concurrent) FSM model !!!*
- Esterel is a well known synchronous/reactive language. Every Esterel model can be compiled to an extended FSM.

Esterel Example

The *Esterel* program corresponding to the problem described on slide 7 and represented as an FSM on slide 8 and in *Statecharts* on 12:

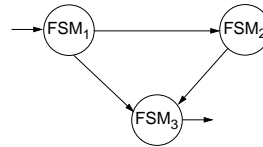
```

module Example
input A, X, B, Y, R;
output O;
loop
  [await A; await X || await B; await Y]
  emit O;
  await R
end loop
end module

```

How to implement a synchronous system?

A synchronous model (concurrent FSMs):



- Signals are propagated instantaneously through the system.
- all FSMs react instantaneously to events.
- No buffering.

How to implement a synchronous system? (cont'd)

☞ In hardware:

- System described as single FSM:
 - implementation as a state machine.
- System described as several FSMs:
 - several communicating synchronous state machines or
 - implement the equivalent single (very large) state machine

But if the system is large:

- How do you distribute the clock signal on a large chip, in order to keep synchrony?
- If there are several chips, keeping synchrony is even more difficult.

How to implement a synchronous system? (cont'd)

☞ In software:

- One single FSM or several FSMs:
Generate a sequential program which *emulates the equivalent state machine*.

Problems:

- For large concurrent systems ⇒ state explosion ⇒ difficult to compile large programs.
- It is practically impossible to implement the software on a multiprocessor system (extremely inefficient to keep the global synchrony of such a multiprocessor software).

How to implement a synchronous system? (cont'd)

☞ In hardware/software:

- If part of the application is implemented as hardware (ASICs) and another part as software running on one or several microprocessors



Practically impossible to implement a globally synchronous system.

- In addition to problems highlighted before with pure software or hardware implementations, we have the very different behavior of software and hardware in terms of execution and communication timing.

How to implement a synchronous system? (cont'd)

- ☞ If the model is impossible (or very difficult and expensive) to implement, there is no use that it is elegant, simple, and can be formally verified.
We get a correct verified model but we cannot implement it correctly!



Synchronous models are very good for relatively small systems implemented in hardware or software.

- ☞ For larger systems and those implemented in hardware and software, we have to *give up the assumption of **global synchrony***.

Summary

- FSMs are the typical state based model. System states are explicitly depicted. As response to input events, an FSM reacts with a transition to a new state and also generates output events.
- FSMs are suitable for modeling control dominated reactive systems.
- FSMs can be extended, in order to enhance the expressive power, by associating variables. Actions related to the variables can be specified, as well as guards on the transitions.
- Complex systems can have an extremely large number of states: state explosion. This is critical in the case of concurrent systems.
- In order to make the models more concise, the mechanisms of hierarchy and concurrency can be used. Languages like Statecharts support such a modelling approach.

Summary (cont'd)

- FSMs are synchronous models. The whole system reacts instantaneously to a set of simultaneous input events. Events are either simultaneous or strictly ordered.
- This is a realistic model for systems where the reaction time is neglectable compared to the rate of external events.
- FSM models are deterministic. Formal reasoning is possible, as well as efficient synthesis.
- Synchronous/Reactive languages (like Esterel) are based on the same semantics as extended synchronous concurrent FSMs.
- Implementing synchronous systems can be done efficiently under certain circumstances. However, it is practically impossible for large systems and, in particular, distributed systems and, even more, hardware/software systems.