Timed & Hybrid Automata

1. Restrictions of the synchronous FSM model
2. Timed automata
3. Timed automata as a particular case of hybrid automata
4. Hybrid automata

Restrictions/Assumptions with synchronous FSMs

- FSMs react to inputs and generate, as response, outputs.
- The inputs are either present or absent; when inputs are present, the FSM reacts and generates outputs.
- Between the time instants when inputs are present, nothing interesting occurs; at each instant when inputs occur, a reaction (outputs) is computed instantly by the FSM.
- FSMs operate in a sequence of discrete reactions.
- The clock can be explicitly modelled as a FSM delivering ticks for the whole system; all transitions in the system are synchronised on this clock tick. Time (non-negative integer) is captured counting these clock ticks → discrete time model.

Example FSM: Thermostat

- Input event [temperature]
  - Signal temperature is received at certain instants of time from a sensor;
  - When input temperature is received and the guard on the transition is true, the system generates an output.
- Outputs [heat_on, heat_off]
- States (S₀, S₁)
  - S₀: system cools (heating off)
  - S₁: system is heating

☞ We want to keep the temperature close to 20°;
☞ To avoid chattering (turning on and off rapidly, all the time), we allow the temperature to be inside a band (technique called hysteresis).

Timed Automata

☞ For modeling real-time asynchronous systems, continuous time models are the natural representation.
☞ Real-time systems require measuring the passage of (continuous) time and performing actions at specific times.
☞ In timed automata time is considered a continuous quantity. No global synchronisation, in the sense of a unique clock, is assumed.
☞ Timed automata are an extension of the FSM model which allows modelling of certain real-time systems and formal reasoning.
- A timed automaton is a finite automaton (similar to a FSM) augmented with a finite set of real-valued clocks.
Example: Thermostat with Timed Automata

- Clock \( x \)
- Input \{temperature\}
- Outputs \{heat_on, heat_off\}
- States \( (S_0, S_1) \)
  - \( S_0 \): system cools (heating off)
  - \( S_1 \): system is heating

\[\begin{align*}
\text{We use a single temperature threshold - the desired level of } 20^\circ; \\
\text{To avoid chattering, the heater has to remain on/off for a minimum required time } T. \\
\end{align*}\]

Example: Gate Control System with Timed Automata

Specification:

- When the train approaches, it sends signal \( \text{app} \) at least 2 minutes before it enters the crossing; after leaving the crossing it sends the signal \( \text{out} \); it leaves the crossing maximum 5 minutes after signalling \( \text{app} \).
- When the controller gets signal \( \text{app} \) it closes the gate, which takes at least 1 minute, but less than 2; then it waits for signal \( \text{out} \); when \( \text{out} \) arrives it opens the gate within maximum 1 minute.

\[\begin{align*}
x < 5 \quad & \text{Train controller} \\
x \leq 5 \quad & \text{Gate controller}
\end{align*}\]

\[\begin{align*}
\text{Train:} & \\
S_0: \text{train far away} & S_1: \text{train approaching} \\
S_2: \text{train crossing} & S_3: \text{train has left}
\end{align*}\]

\[\begin{align*}
\text{Gate controller:} & \\
R_0: \text{controller waiting} & R_1: \text{train approaching} \\
R_2: \text{gate is down} & R_3: \text{train has left}
\end{align*}\]
Timed Automata: Rules/Properties

- Transitions are instantaneous; time elapses when the automaton is in a certain state.
- When a transition occurs, some of the clocks can be reset; at any moment, the value of a clock is equal to the time elapsed since the last time it has been reset.
- Time passes at the same rate for all clocks.
- When a transition occurs, signals (events) can be generated.
- Transitions can have associated guards expressed as conditions on clock values; the transition can be taken only if the current values of the clocks satisfy the guard.
- Transitions can have input signals (events) associated; when the signal arrives and the associated guard is satisfied, the transition will be taken.
- States can have associated invariants, expressed as conditions on the clocks; the automaton can stay in that state only as long as the invariant is true.

Timed Automata: Rules/Properties (cont’d)

- Like FSMs, timed automata can be extended with variables.
- Actions on variables can be associated to transitions.
- Guards expressed as conditions on the variables can be associated to transitions.

Timed automata are, by definition, infinite state models: At any time, the state of the system is defined not only by the actual state in the state machine (e.g. S0, S1, etc.), but also by the current values of the clocks! However, for verification, timed automata admit finite state representations (by exploiting equivalence relations on certain portions of the state space)!

- Model checking techniques can be used to prove properties of timed automata.
- The state explosion problem is more severe than for synchronous concurrent FSMs.

Hybrid Automata

- Timed automata are FSMs with addition of clocks.
- A clock is a continuous variable whose value can be described by the differential equation: \( \dot{x}(t) = 1 \).
- Timed automata are, in fact, the simplest form of hybrid automata.
- Hybrid automata are FSMs combined with a finite set of continuous variables whose values are described by a set of ordinary differential equations.

The Thermostat as a Hybrid Automaton

- In the above model, the equation describing the continuous variable representing clock \( x \) is made explicit. This model is completely equivalent with the one on slide 5.
Hybrid Automata: Rules/Properties

- A Hybrid Automaton associates with each state of an FSM a dynamic behavior.
- The dynamic behavior in each state is specified by a state refinement; a state refinement describes the dynamic behavior of the outputs (and other additional continuous state variables) as a function of the inputs (and possible state variables).
- State refinements are specified as ordinary differential equations.
- Transitions can have associated guards, assignments to variables, outputs, etc.
- Hybrid automata are extremely strong in their expressive power; they combine discrete and continuous behavior in one single model.

Example: Water Tank

- Each tank is leaking at constant rate $r_1$ ($r_2$).
- Water is added at a constant rate $\rho$.
- One tank is filled at a time; filling switches from one tank to the other in zero time.
- The goal is to keep the water volume above $v_1$ and $v_2$ respectively.
- The current water volume is $x_1$ ($x_2$).

Example: Water Tank (cont’d)

- This is a simulation considering:
  - Leaking rates: $r_1 = r_2 = 0.5$
  - Water inflow rate: $\rho = 0.75$
  - Keep the water above $v_1 = v_2 = 0$
  - Initial level: $x_1(0) = 0$, $x_2(0) = 1$

\[ \begin{align*}
  x_1(t) &\leq v_1, \text{ fill}_1! \\
  x_2(t) &\leq v_2, \text{ fill}_2!
\end{align*} \]

Since $\rho < r_1 + r_2$, both tanks will, eventually, become empty.

As the tanks come close to the 0 level, the number of switches per time unit increases.

With such a Hybrid Automata model, one can use formal verification to answer questions like: will tank 1 become empty before time 7? You cannot ask this with a simple timed automata model (since the flow equations are not part of the model).

\[ \begin{align*}
  v_1(t) &\leq 0, \text{ fill}_1! \\
  v_2(t) &\leq 0, \text{ fill}_2!
\end{align*} \]
In order to avoid Zeno behaviour one possible solution is to introduce intermediate states in which the system spends an amount of time $\tau$ (no tank is filled in that interval).

Example: Water Tank (cont’d)

$\begin{align*}
  x_1(t) &\leq v_1, s(t) := 0 \\
  s(t) &\geq \tau, \text{fill}_2!
\end{align*}$

Hybrid Automata: Final Comments

- Hybrid automata models can be used for simulation and formal verification.
- Hybrid automata, like timed automata, are, by definition, infinite state models. However, they admit a finite state representation (by exploiting equivalence relations on certain portions of the state space):
  - Model checking techniques can be used to prove properties of hybrid automata.
  - The state explosion problem is more severe than for timed automata!


What Modeling Approach to Choose?

- It depends on the characteristics of the system:
  - control or data flow dominated (e.g. DSP application or reactive system);
  - synchronous or asynchronous; centralised or distributed;
  - how large?
  - what aspects related to timing are we interested in?
- It depends on what you intend to do with the model:
  - simulation
  - formal verification
  - automatic synthesis
- It depends on what tools you have available and which approach you (or your company or your boss!) prefer.

Don’t use the “strongest!” Go for exactly that expressive power you need; not more. Remember Fig 3, slide 9:

- Large expressive power: imperative model (e.g. unrestricted use of C or Java):
  - Can specify “anything”.
  - No formal reasoning possible (or extremely complex).
- Limited expressive power, based on well chosen computation model (e.g. Esterel):
  - Only particular systems can be specified.
  - Formal reasoning is possible.
What Modeling Approach to Choose?

Large embedded systems are heterogeneous → mixture of models:

- Imperative FSMs
- Dataflow
- Discrete event

Specification

Refinement

Higher level of abstraction

partitioning

compiling

softw. synth

HLS

LS

Lower level of abstraction

Software model

Software model

ASIC

glue logic

Modeling Languages

☞ The choice of a modeling language is, to a large extent, connected to the choice of the modeling approach:

This, because certain modeling languages are strongly connected to a particular model of computation:

- Communicating asynchronous state machines: SDL, Lotos
- Synchronous systems: Esterel, StateCharts;
- Dataflow and continuous computation: Matlab, Lustre, Silage

Modeling Languages (cont’d)

☞ Some languages do not support particular models of computation

- General purpose programming languages:
  - imperative: C, C++, Java, Ada
  - functional: Lisp, Scheme, Haskell
  - logic: Prolog
- Hardware description languages:
  - VHDL, Verilog, SystemC: imperative, Discrete Event

☞ When used according to certain restrictions and programming guide-lines, models based on particular models of computation can be realised in these languages too.

Modeling Languages (cont’d)

☞ Different parts of the specification can be realised in different languages.

- Will we ever get “THE” System Modeling Language?
- Will multi-language specification become the standard?