Timed I/O Automata

Pedro Ferreira do Souto

Departamento de Engenharia Informtica
Faculdade de Engenharia
Universidade do Porto

Outline

Outline

Outline

Outline

Outline

Models of Computation

Synchronous model

Processes repeatedly execute rounds in lock-step. In each round, they:

1. Use their current state to generate messages to send to neighbors, and put them in the appropriate channels.
2. Compute the new state from the current state and the incoming messages, and remove all the messsages from the channels.

Asynchronous model

Makes no assumptions regarding the timing behavior of system components

1. Processes may take an arbitrary time to execute the actions prescribed by the algorithms.
2. Channels may take an arbitrary time to deliver messages that are sent through them.

Time and Models of Computation

- Both models abstract away the time.
- Time is sometimes used for the analysis of the (time) complexity.
  - But it is not part of the model itself.
- In practice, most systems use time, at least in the form of timeouts.
- Increasingly, systems interact with the real world, which some times imposes timing requirements. Correctness depends:
  - Not only on the outputs generated by the system;
  - But also on the time at which these outputs are generated.
    - ★ For some systems, being late is at least as bad as an omission fault.

Outline

Outline

MMT Timed Automata

Basic idea

Replace the fairness conditions of I/O automata with lower and upper bounds on the time to execute tasks.

Observation

- Imposing upper bounds only is not as interesting:
  - It would not restrict the set of executions that are produced by an I/O automaton.
- Imposing lower bounds in addition to upper bounds, may actually restrict the set of executions.

MMT Timed Automaton: Definition

Definition Let A be an I/O automaton with a finite number of tasks.

A **boundmap for A** *is a pair of mappings,* **lower** *and* **upper**, *that give lower and upper bounds* **for all tasks.** *Furthermore, for each task C, lower(C) and upper(C) must satisfy:*

1. $0 \leq lower(C) < \infty$
2. $0 < upper(C) \leq \infty$
3. $lower(C) \leq upper(C)$

MMT (Timed) Automaton is an I/O automaton *A* with a finite number of tasks together with a boundmap for A.

MMT Timed Execution (1/3)

Timed execution of an MMT automaton $B = (A, b)$ is defined to be a **finite** sequence:

$$\alpha = s_0, (\pi_1, t_1), s_1, (\pi_2, t_2), \ldots, (\pi_r, t_r), s_r$$

or an **infinite** sequence:

$$\alpha = s_0, (\pi_1, t_1), s_1, (\pi_2, t_2), \ldots, (\pi_r, t_r), s_r, \ldots$$

Where:

s's are the states of I/O automaton A
π's are actions of A
t's are times in $\mathbb{R}^{\geq 0}$

Furthermore:

1. The *untimed* sequence $s_0, \pi_1, s_1, \pi_2, \ldots, \pi_r, s_r, \ldots$ must be an execution of the I/O automaton A.
2. $t_i \leq t_{i+1}$, i.e. the times must be nondecreasing, and must satisfy the lower and upper bound requirements.

MMT Timed Execution (2/3)

Initial index (r) for task C, if C is enabled in s_r, and one of the following is true

1. $r = 0$
2. C is not enabled in s_{r-1}
3. $\pi_r \in C$ (Why?)

Essentially, this is a point at which we start measuring the time for execution of a task.

Safety properties

Upper bound If there exists $k > r$ with $t_k > t_r + upper(C)$, then there is $k' > r$ with $t_{k'} \leq t_r + upper(C)$ such that either $\pi_{k'} \in C$ or C is not enabled in $s_{k'}$.

Lower bound There is no $k > r$ with $t_k < t_r + lower(C)$ and $\pi_k \in C$.

MMT Timed Execution (3/3)

Admissibility (Liveness property)

If timed execution α is an infinite sequence, then the times of the actions approach ∞. If α is a finite sequence, then in the final state of α, if task C is enabled, then $upper(C) = \infty$.
Essentially, this means that:

1. Time advances normally.
2. An automaton does not stop processing, if the automaton is scheduled to perform some more work.

Set of admissible timed executions (atexecs(B))

Timed trace of a timed execution α of B ($ttrace(\alpha)$) is the subsequence of α consisting of all **external** actions, each paired with its associated time.

Set of admissible timed traces (attraces(B))

Outline

MMT Example: Reliable FIFO channel (1/2)

Add upper bound d on the delivery time for the **oldest message** in the *queue* of the $C_{i,j}$ I/O automaton.

$\mathbf{D_{i,j} = (C_{i,j}, b)}$

Signature:

Input:

$send(m)_{i,j}, m \in M$

Output:

$receive(m)_{i,j}, m \in M$

State:

queue, a FIFO queue of elements of M
initially empty

Transitions:

$send(m)_{i,j}$
Effect:
enqueue m in *queue*

$receive(m)_{i,j}$
Precondition:
m at head of *queue*
Effect:
dequeue m from *queue*

Tasks and bounds:

$\{receive(m)_{i,j} : m \in M\}$, bounds $[0, d]$, for some $d \in R^+$

MMT Example: Reliable FIFO channel (2/2)

Admissible timed traces of $D_{i,j}$

1. $(send(1)_{i,j}, 0), (send(2)_{i,j}, 0), (receive(1)_{i,j}, d), (receive(2)_{i,j}, d)$
2. $(send(1)_{i,j}, 0), (send(2)_{i,j}, 0), (receive(1)_{i,j}, 0), (receive(2)_{i,j}, 0)$
3. $(send(1)_{i,j}, 0), (receive(1)_{i,j}, d), (send(2)_{i,j}, d), (receive(2)_{i,j}, 2d),$
   $(send(3)_{i,j}, 2d), (receive(3)_{i,j}, 3d), \ldots$

Non Admissible timed traces of $D_{i,j}$ (Why?)

1. $(send(1)_{i,j}, 0), (send(2)_{i,j}, 0), (receive(1)_{i,j}, d)$
2. $(send(1)_{i,j}, 0), (receive(1)_{i,j}, 2d)$
3. $(send(1)_{i,j}, 0), (receive(1)_{i,j}, d), (send(2)_{i,j}, d), (receive(2)_{i,j}, d),$
   $(send(3)_{i,j}, d), (receive(3)_{i,j}, d), \ldots$

MMT Example: Timeout automaton

- P_2 waits for the reception of a message from another process P_1
- If no such message arrives within a certain amount of time, P_2 performs a *timeout* action.
- P_2 measures the elapsed time by counting a fixed number $k \geq 1$ of its own steps, which are supposed to have known lower and upper bounds $\ell_1, \ell_2, 0 < \ell_1 \leq \ell_2 < \infty$.
- Its *timeout* action is performed at most time ℓ after its *count* reaches 0.

MMT Example: Timeout automaton (2/2)

Signature:

Input: $receive(m)_{1,2}, m \in M$

Internal: $decrement$

Output: $timeout$

State:

$count \in \mathbb{N}$, initially k

$status \in \{active, done, disabled\}$, initially $active$

Transitions:

$receive(m)_{1,2}$

Effect:

if $status = active$

then $status := disabled$

$decrement$

Precondition:

$status = active$

$count > 0$

Effect:

$count := count - 1$

$timeout$

Precondition:

$status = active$

$count = 0$

Effect:

$status = done$

Tasks and bounds:

$\{decrement\}$, bounds $[\ell_1, \ell_2]$

$\{timeout\}$, bounds $[0, \ell]$

Outline

MMT Automata Composition

- Allows building larger models from smaller ones.
- Same basic approach:
  - Matching the names of outputs of one I/O (MMT) automaton with the names of inputs of other I/O automata.
- Same restrictions as in basic I/O automata. I.e., same definition of **automata compatibility**:
  1. Each set of internal actions is disjoint from all other action sets.
  2. Sets of output actions are disjoint.
  3. No action is contained in infinitely many automata.
- But, composition of only a finite number of MMT automata.
  - The third restriction above trivially satisfied.

MMT Automata Composition: Formal

Definition A finite set of MMT automata is **compatible**, if their underlying I/0 automata are compatible.

Definition
Let $\{(A_i, b_i)\}_{i \in I}$ be a finite set of compatible MMT automata.
The **composition** $(A, b) = \Pi_{i \in I}(A_i, b_i)$ is defined as follows:

1. $A = \Pi_{i \in I} A_i$, i.e. A is the composition of all the underlying I/0 automata A_i.
2. For each task C of A, $b's$ *lower* and *upper* bounds for C are the same as those of b_i, where A_i is the unique component I/O automaton having task C.

Note We can also use the infix operator $\times$ to denote compositions.

$$\Pi_{i \in I} = A_1 \times \ldots \times A_n$$

Basic Properties of MMT Composition

Let $\{B_i\}_{i \in I}$ be a compatible collection of MMT automata and let $B = \Pi_{i \in I} B_i$. Then we have the following three theorems:

1. If $\alpha \in atexecs(B)$, then $\alpha | B_i \in atexecs(B_i)$, for everey $i \in I$
2. If $\beta \in attraces(B)$, then $\beta | B_i \in attraces(B_i)$, for everey $i \in I$

Suppose $\alpha_i \in atexecs(B_i)$, for every $i \in I$
Suppose β is a sequence of $(action, time)$ pairs, where all *actions* in β are in $ext(A)$, such that $\beta | B_i = attrace(\alpha_i)$, for every $i \in I$
Then there is an admissible timed execution α of B such that $\beta = ttrace(\alpha)$ and $\alpha_i = \alpha | B_i$, for every $i \in I$

Suppose β is a sequence of $(action, time)$ pairs, where all *actions* in β are in $ext(A)$.
If $\beta | B_i \in attrace(B_i)$, for every $i \in I$,
then $\beta \in attrace(B)$

MMT Critique

- MMT automata are good for modelling systems at low level of detail.
  - Tasks are convenient to model system components.
  - Time bounds are convenient to model their "speed".
- They are not as good for higher level modeling.
- They are not always appropriate for proving the correctness of algorithms.

Outline

General Timed (I/O) Automata

Idea

Encode timing restrictions directly and explicitly in:

- the states
- the transitions

of an I/O automaton.

Advantage

Allows the use of state-based proof methods, such as:

- invariant assertions;
- simulation relations

to reason about both:

- state, and
- timing

properties.

Outline

General Timed Automata: Signature

time-passage action ($\nu(t)$) denotes the passage of time by amount $t \in \mathbb{R}^+$

timed signature S quadruple including:

- input actions ($in(S)$)
- output actions ($out(S)$)
- internal actions ($int(S)$)
- time-passage actions

other definitions as usual:

- visible actions ($vis(S) = in(S) \cup out(S)$)
- locally controlled actions ($local(S) = int(S) \cup out(S)$)
- all actions ($act(S)$)

external actions ($ext(S)$) $ext(S) = vis(S) \cup \{\nu(t) : t \in \mathbb{R}^+\}$

discrete actions ($disc(S)$) $disc(S) = vis(S) \cup int(S)$

General Timed Automata (GTA): Definition

Four components of a GTA A

sig(A) a timed signature;

states(A) a set of states;

start(A) a nonempty subset of *states*(A) known as the **start states** or **initial states**

trans(A) a **state transition relation** where

$$trans(A) \subseteq states(A) \times acts(sig(A)) \times states(A)$$

Note a GTA does not have *tasks*(A) components

Basic axioms a GTA A must satisfy

A1 (combination of time-passage transitions): If $(s, \nu(t), s')$ and $(s', \nu(t), s'')$ are in *trans*(A), then $(s, \nu(t+t'), s'')$ is in *trans*(A).

A2 (splitting of time-passage transitions): If $(s, \nu(t), s') \in trans(A)$ and $0 < t' < t$, then there is a state s'' such that $(s, \nu(t'), s'')$ and $(s'', \nu(t-t'), s')$ are in *trans*(A).

General Timed Automata (GTA): More definitions (1/2)

Shorthands *acts*(A) for *acts*(*sig*(A)), *in*(A) for *in*(*sig*(A)) and so on

Timed execution fragment of GTA A is either a **finite** sequence:

$$\alpha = s_0, \pi_1, s_1, \pi_2, \ldots, \pi_r, s_r$$

or an **infinite** sequence:

$$\alpha = s_0, \pi_1, s_1, \pi_2, \ldots, \pi_r, s_r, \ldots$$

where

s's are states of A

π's are actions of A

$(s_k, \pi_{k+1}, s_{k+1})$ is a transition of A for every k.

Timed execution of GTA A is a timed execution fragment that begins with a **start state**.

Time of occurrence of discrete action π_r in timed execution fragment α is the sum of all the reals of the time-passage actions preceding π_r in α.

Admissible timed execution fragment is a timed execution fragment in which the sum of all the reals in the time-passage actions in α is ∞.

atexecs(A) is the set of admissible timed executions of A.

General Timed Automata (GTA): More definitions (2/2)

State s is reachable in A if s is the final state of a **finite timed execution** of A.

Timed trace of a timed execution fragment α is the sequence of visible actions, i.e. input or output, in α each paired with its time of ocurrence.

Note Whereas MMT and GTA timed executions are syntatically different, MMT and GTA timed traces are syntatically and semantically identical.

Admissible timed traces of A ($attraces(A)$) is the set of timed traces of $atexecs(A)$.

Time-passage refinement of a timed execution fragment α is another timed execution fragment α' that is identical to α except for the fact that in α' some time-passage steps of α are replaced with finite sequences of time-passage steps, with the same initial and final states and the same total amount of time-passage.

Timed execution fragments α and α' are time-passage equivalent if they have a common time-passage refinement.

Outline

GTA Example: Reliable FIFO channel (1/3)

Signature:

Input:

$send(m)_{i,j}, m \in M$

Output:

$receive(m)_{i,j}, m \in M$

Internal:

Time-passage:

$\nu(t), t \in \mathbb{R}^+$

Modelling time

Use two state variables:

now is used to keep track of current time;

last is the deadline (latest time) for the channel to deliver (*receive*()) the next message, if any.

Note the values of both *now* and *last* are absolute times, not relative (*timeouts*).

GTA Example: Reliable FIFO channel (2/3)

State

queue, a FIFO queue of elements of M, initially empty
$now \in \mathbb{R}_0^+$, initially 0
$last \in \mathbb{R}^+ \cup \{\infty\}$, initially ∞

Transitions:

```
$send(m)_{i,j}$
  Effect:
    enqueue $m$ in $queue$
    if $|queue = 1|$ then
      $last := now + d$
$\nu(t)$
  Precondition:
    $now + t \leq last$
  Effect:
    $now := now + t$
```

- The channel's delay starts only when a message gets to the head of the *queue*, not when it is enqueued.
- Time cannot advance past the deadline to deliver the next message, otherwise the channel delay would exceed its upper bound.

GTA Example: Reliable FIFO channel (3/3)

Transitions:

```
$receive(m)_{i,j}$
  Precondition:
    $m$ at head of $queue$
  Effect:
    dequeue $m$ from $queue$
    if $queue$ is emtpy then
      $last := \infty$
    then $last := now + d$
```

- If the *queue* becomes empty, then we do not know what the deadline to deliver the next message will be, so we set it to ∞.
- Why don't we need to add $last \leq now$?
- Could we have used $last \leq now$ instead of $now + t \leq last$, in $\nu(t)$?

General approach

- Specify variable *now* to keep track of current time.
- For each timed local (output or internal) action specify variables *first* and *last* to keep track of the earliest and latest times when the action can be executed.

GTA Example: Another reliable FIFO channel (1/2)

- The difference wrt the previous model is that the channel delay bound, d, applies to the delay measured from the moment the message is submitted to the channel (*send*()) and not from the moment it gets at the head of the queue.
- Thus, we now need to keep track of the deadline for each of the messages separately:
  - Each element in the queue is now a pair (m, t), where m is the message in transit and t is the latest time at which the message must be delivered.

Signature: No changes.

Input:

$send(m)_{i,j}, m \in M$

Output:

$receive(m)_{i,j}, m \in M$

Internal:

Time-passage:

$\nu(t), t \in \mathbb{R}^+$

State

queue, a FIFO queue of elements of $M \times \mathbb{R}^+$, initially empty

$now \in \mathbb{R}_0^+$, initially 0

GTA Example: Another reliable FIFO channel (2/2)

Transitions:

```
$send(m)_{i,j}$
  Effect:
    enqueue $(m, now + d)$ in $queue$

$receive(m)_{i,j}$
  Precondition:
    $(m, t)$ at head of $queue$
  Effect:
    dequeue $(m, t)$ from $queue$

$\nu(t)$
  Precondition:
    if $queue$ is not emtpy then
      $now + t \leq t'$, where $t'$
        is the time in the pair
        at the head
  Effect:
    $now := now + t$
```

Note: There is no MMT automaton with the same admissible timed traces as this GTA.

GTA vs MMT Timed Automata

- GTA are more general/powerful than MMT timed automata
  - For each MMT timed automaton, it is possible to develop a GTA that has the same set of admissible timed traces.
- In Section 23.2.2 of the *hive book*, Nancy Lynch's presents a transformation from an MMT automaton (A, b) into a GTA $A' = gen(A, b)$ that generalizes the approach followed in the example of the channel model:
  1. Add state variable *now*.
  2. For each task C, add state variables $\mathit{first}(C)$ and $\mathit{last}(C)$, and update $\mathit{first}(C)$ and $\mathit{last}(C)$ as appropriate and according to the boundmap b.
  3. Add time-passage actions $\nu(t)$, but prevent time from passing beyond $\mathit{last}(C)$, for all tasks C.
  4. For all actions of task C add a pre-condition to ensure that the current time is at least as great as $\mathit{first}(C)$.

Outline

GMT Automata Composition: Formal

Timed signatures compatibility is defined as usual

Composition of a finite collection of compatible timed signatures

$S = \Pi_{i \in I} S_i$ is the timed signature with:

- $out(S) = \cup_{i \in I} out(S_i)$
- $int(S) = \cup_{i \in I} int(S_i)$
- $in(S) = \cup_{i \in I} in(S_i) \setminus out(S)$

Composition of a finite collection of GTAs $A = \Pi_{i \in I} A_i$ is the automaton:

1. $sig(A) = \Pi_i sig(A_i)$
2. $states(A) = \Pi_i states(A_i)$
3. $start(A) = \Pi_i start(A_i)$
4. $trans(A)$ is the set of triples (s, π, s') such that, for all $i \in I$,
   if $\pi \in acts(A_i)$, then $(s_i, \pi, s'_i) \in trans(Ai)$
   else $s_i = s'_i$

Note that all components that have π in their signature, participate simultaneously in steps involving π, otherwise they do nothing. This implies that all components participate in time-passage steps.

Outline

GTA: Properties

Theorem

The composition of a compatible set of GTA is a GTA.

Properties in GTAs

Invariant assertion for A is a property that is true of all reachable states.

- Same definition as for asynchronous systems.
- Proof by induction on the number of steps in a timed execution.

Timed trace propertiy P comprises:

- $sig(P)$, a timed signature containing no internal actions;
- $ttraces(P)$, a set of sequences of $(action, time)$ pairs, with timing restrictions similar to those of admissible timed traces of GTA.

GTA: Timed simulation relation (1/2)

Let A and B be two GTA with the same input and output actions.
Let f be a binary relation over $states(A)$ and $states(B)$
f is a timed simulation relation from A to B if both of the following are true:

1. If $s \in start(A)$, then $f(s) \cap start(B) \neq \emptyset$
2. If
   i. s is a reachable state of A, and
   ii. $u \in f(s)$ is a reachable state of B, and
   iii. $(s, \pi, s') \in trans(A)$,

   then there is a timed execution fragment α starting in u and terminating in $u' = f(s')$, such that
   i. $ttrace(\alpha) = ttrace(s, \pi, s')$
   ii. The total amount of time-passage in α is the same as the total amount of time-passage in (s, π, s')

Note π need not be a visible action. It can also be an internal action or a time-passage action.

GTA: Timed simulation relation (2/2)

Start condition (1) requires that the relation map a start state of A to a start state of B.

Step condition (2) requires that the relation preserve:

i. the sequence of visible actions, each paired with its time of occurrence;

ii. the total amount of time-passage.

Theorem

If there is a timed simulation relation from A to B, then $attraces(A) \subseteq attraces(B)$

- Simulations are most useful to prove time bounds.

Outline

Further Reading

- Chapter 23, *Modelling V: Partially Synchronous System Models*, of Nancy Lynch's *Distributed Algorithms*.
- Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. The Theory of Timed I/O Automata. Technical Report MIT-LCS-TR-917a, MIT Laboratory for Computer Science, Cambridge, MA, November, 2004.(PostScript available at Nancy Lynch's web page at MIT.)