Timed I/O Automata

Pedro Ferreira do Souto

Departamento de Engenharia Informática
Faculdade de Engenharia
Universidade do Porto
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
Outline

1. Introduction

2. Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3. Further Reading
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
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 messages 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.
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 sometimes imposes timing requirements. Correctness depends:
  ▶ Not only on the outputs generated by the system, or even their order;
  ▶ 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

1. Introduction

2. Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3. Further Reading
There are several variants of Timed I/O Automata
  ▶ They are all based on standard I/O automata
  ▶ They all include extensions to reason about timing properties

We consider the variant described in the first reference
  ▶ It is a simplification of Hybrid I/O Automata
  ▶ It relies only on a few concepts that do not appear in the standard I/O automata

Dynamic Type of a Variable
Trajectory
Timed Input/Output Automata (TIOA)

- A timed (I/O) automaton is a state machine whose states are the valuations of its variables
  - Variables are internal to an automaton
- The state of a timed automaton may change
  - instantaneously, by the occurrence of discrete transitions
  - over an interval of time via trajectories, which are functions that describe the evolution of the state variables with time
- The discrete transitions are labeled with actions, which may be one of input, output or internal actions
  - Input and output actions are used for communication with the automaton’s enviroment
    - Internal actions are not visible externally
  - Output and internal actions are under the automaton control
    - But input actions are not
- Communication of a TIOA with its enviroment is limited to discrete transitions
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
Dynamic Type of a Variable

- Each variable in a IO automaton has a (static) type, which specifies the values it may take.
- In the case of a TIOA, every variable has also a dynamic type, which specifies how its value may evolve over time.
- We consider essentially two types:
  - **Discrete** The value changes only at discrete points in time, remaining constant between those points.
    - The values of discrete variables change only upon occurrence of transitions.
    - All variables in standard I/O automata are discrete.
  - **Analog** The value may change continuously over a time interval.
    - This type is particularly useful to model timers/clocks, i.e. variables that measure the passage of time.
Trajectory

- A trajectory, $\tau$, describes the evolution of a set of variables over an interval of time, $\tau$’s domain, which:
  - Always starts at 0
  - May not be right closed

- Trajectories can be concatenated, using a concatenation operation $\circlearrowright$. The result is a trajectory:
  - Over a time interval whose duration is the sum of the time intervals of each of the trajectories
  - Obtained by time-shifting by the necessary amount of time each of the operand trajectories

- Given a trajectory, $\nu$, we can define a prefix trajectory $\tau$, by restricting $\nu$ to a time interval starting at 0 that is a subset of $\nu$’s domain

$$\tau \leq \nu$$
The last valuation of a trajectory, which may not agree with the first valuation of the following operand trajectory, is the one that appears in the concatenation.

src: Kaynar et. al. 2005
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
Timed I/O Automata: Formal Definition

A timed I/O automaton (TIOA) $A = (X, Q, \Theta, I, 0, H, D, T)$ consists of:

- A set $X$ of **internal variables**
- A set $Q \subseteq \text{val}(X)$ of **states**
- A nonempty set $\Theta \subseteq Q$ of **start states**
- A set $I$ of **input actions**, a set $O$ of **output actions** and a set $H$ of **internal actions**, disjoint from each other. We write:
  
  $E \trianglerighteq I \cup O$ \hspace{1cm} $\text{the set of external actions}$
  
  $A \trianglerighteq E \cup H$ \hspace{1cm} $\text{the set of all actions}$
  
  $L \trianglerighteq O \cup H$ \hspace{1cm} $\text{the set of locally controlled actions}$

- A set $D \subseteq Q \times A \times Q$ of **discrete transitions**
  
  $\trianglerighteq$ We write $x \xrightarrow{a} x'$ as a shorthand of $(x, a, x') \in D$
  
  $\trianglerighteq$ We say that $a$ is **enabled** in $x$ if $x \xrightarrow{a} x'$ for some $x'$
  
  $\trianglerighteq$ We say that a set $C$ of actions is enabled in a state $x$ if some action in $C$ is enabled in $x$

- A set $T \subseteq \text{trajs}(Q)$ of trajectories, which must satisfy a set of axioms.
Timed I/O Automata: Trajectories (1/2)

- Given a trajectory \( \tau \in \mathcal{T} \), we denote
  - \( \tau.fstate \) the value of the state variables at time 0
  - \( \tau.lstate \) the last value of the state variables, if \( \tau \) is closed
  - When \( \tau.fstate = x \) and \( \tau.lstate = x' \), we write \( x \stackrel{\tau}{\rightarrow} x' \)

- The set of trajectories \( \mathcal{T} \) of tim
ed automaton (TA) must satisfy the following axioms:
  - **T0** Existence of point trajectories
    - If \( x \in Q \) then \( \wp(x) \in \mathcal{T} \)
  - **T1** Prefix closure
    - For every \( \tau \in \mathcal{T} \) and every \( \tau' \leq \tau, \tau' \in \mathcal{T} \)
  - **T2** Suffix closure
    - For every \( \tau \in \mathcal{T} \) and every \( t \in \text{dom}(\tau), \tau \triangleright t \in \mathcal{T} \)
  - **T3** Concatenation closure
    - Let \( \tau_0 \tau_1 \tau_2 \ldots \) be a sequence of trajectories in \( \mathcal{T} \) such that for each nonfinal index \( i \), \( \tau_i \) is closed and \( \tau_i.lstate = \tau_{i+1}.fstate \).
    - Then \( \tau_0 \triangleleft \tau_1 \triangleleft \tau_2 \ldots \in \mathcal{T} \)
Axioms **T1, T2** are needed for the parallel composition operation for TA

- In a composed system, any trajectory of any component automaton may be interrupted at any time by a discrete transition of another (possibly independent) automaton
  - Axiom **T1** ensures that the part of the trajectory up to the discrete transition is a trajectory
  - Axiom **T2** ensures that the remainder is a trajectory

**Axiom T3** is required because the environment of a timed automaton may change its dynamics repeatedly, and the automaton must be able to follow this behavior.
Timed I/O Automata: Axioms

- The following axioms are satisfied:

  **E1: Input action enabling**
  For every $x \in Q$ and every $a \in I$, there exists $x' \in Q$ such that $x \xrightarrow{a} x'$
  - I.e., a TIOA is able to perform every input action at any time
  - Standard IOA must also satisfy this axiom

  **E2: Time-passage enabling**
  For every $x \in Q$, there exists $\tau \in \mathcal{T}$ such that $\tau.fstate = x$ and either:
  1. $\tau.ltime = \infty$ or
  2. $\tau$ is closed and some $l \in L$ is enabled in $\tau.lstate$
  - I.e., a TIOA either allows time to advance forever or only up to a point where it is able to perform some locally controlled action
Outline

1 Introduction

2 Timed I/O Automata
   • Preliminaries
   • Definition
   • Examples
     • TIOA Behavior
     • Composition of TIOA
     • Timed I/O Automata with Bounds

3 Further Reading
Timed I/O Automata Specification

Based on a TIOA language, in which a specification consists of four main parts:

**Signature** lists the actions along with their kinds (input, output or internal) and parameter types

**State variables list** declares the names and types of state variables. The dynamic type is defined implicitly

- Variables of type **Real** are analog, and all other variables are discrete

**Collection of transition definitions** defined in **precondition-effect** style

**Trajectories definition**

It differs from the IO Automata specification in that instead of a tasks definition we have trajectories definition
TIOA Example 4.1: A Time-Bounded Channel (1/6)

- The channel is reliable, i.e. does not drop messages
- The channel is FIFO, i.e. delivers the messages in the order they are sent
- Furthermore, it is time-bounded, i.e. it delivers the messages within a certain time bound (b) from being sent
TIOA Example 4.1: A Time-Bounded Channel (2/6)

```plaintext
automaton TimedChannel(b: Real, M: Type)

type Packet = tuple of message: M, deadline: Real

signature
    input send(m: M)
    output receive(m: M)

states
    queue: Seq[Packet] := {},
    now: Real := 0

initially b ≥ 0

transitions
    input send(m)
        eff
            queue := append([m, now+b], queue)
    output receive(m)
        pre
            head(queue).message = m
        eff
            queue := tail(queue)

trajectories
    stop when
        ∃p: Packet p ∈ queue ∧ (now = p.deadline)

    evolve
        d(now) = 1
```
TIOA Example 4.1: A Time-Bounded Channel (3/6)

The TimedChannel automaton has two parameters:
- \( b \) the bound on the time to deliver a message
- \( M \) is the type of messages communicated by the channel

The line:

\[
\text{type } \text{Packet} = \text{tuple of message: } M, \text{ deadline: } \text{Real}
\]

defines the type packet, which:
- Associates a message with its delivery deadline
- Is used in the definition of variable queue

Signature specifies actions (this TIOA has no internal actions)

\[
\text{signature}
\]

\[
\text{input send (m: M)}
\]

\[
\text{output receive (m: M)}
\]

both of which take as a parameter the message being sent/received
TIOA Example 4.1: A Time-Bounded Channel (4/6)

**State** Comprises two variables:

```plaintext
states
    queue: Seq[Packet] := {},
    now: Real := 0
initially b ≥ 0
```

*queue* is a queue with the packets in transit, it uses the built-in type *Seq[]* for sequences/queues

*now* is used to measure the time

The *initially* clause specifies a predicate that must be true of the automaton parameters and its initial state
TIOA Example 4.1: A Time-Bounded Channel (5/6)

Transitions

Defines 2 actions:

\[ \text{send}(m) \]

\[ \begin{align*}
\text{input} & \quad \text{send}(m) \\
\text{eff} & \quad \text{queue} := \text{append}([m, \text{now} + b], \text{queue})
\end{align*} \]

- Transitions on input actions have no preconditions, i.e. it is as if the precondition was \textbf{true}, which is omitted

\[ \text{receive}(m) \]

\[ \begin{align*}
\text{output} & \quad \text{receive}(m) \\
\text{pre} & \quad \text{head(queue).message} = m \\
\text{eff} & \quad \text{queue} := \text{tail(queue)}
\end{align*} \]

- A \text{receive}(m) transition can occur only when \( m \) is the first message in the queue and it results in the removal of the first message from the queue
TIOA Example 4.1: A Time-Bounded Channel (6/6)

trajectories
stop when
\[ \exists p: \text{Packet } p \in \text{queue} \land (\text{now} = p.\text{deadline}) \]
evolve
d(now) = 1

stop when specifies a stopping condition, which must hold only in the last state of the trajectory
- It ensures that time does not advance beyond the point where the stopping condition is true

evolve specifies the algebraic and differential equations that must be satisfied by the trajectories
- It is assumed that each variable follows a continuous function throughout a trajectory
- This implies that the value of a discrete variable is constant throughout a trajectory
TIOA Example 4.2: Periodic Sending Process

Process that sends messages every $u$ time units

```plaintext
automaton PeriodicSend(u: Real, M: Type)

signature
  output send(m: M)

states
  clock: Real := 0
  initially u ≥ 0

transitions
  output send(m)
    pre
      clock = u
    eff
      clock := 0

trajectories
  stop when
    clock = u
  evolve
    d(clock) = 1
```
TIOA Example 4.3: Periodic Sending Process with Crashes

Process that send messages every \( u \) time units, unless it crashes

\[
\text{automaton } \text{PeriodicSend2}(u: \text{Real}, M: \text{Type})
\]

\[
\text{signature} \\
\text{input} \quad \text{crash} \\
\text{output} \quad \text{send}(m: M)
\]

\[
\text{states} \\
\text{crashed}: \text{Bool} := \text{false}, \\
\text{clock}: \text{Real} := 0 \\
\text{initially} \quad u \geq 0
\]

\[
\text{transitions} \\
\text{output} \quad \text{send}(m) \\
\text{pre} \\
\neg \text{crashed} \land \text{clock} = u \\
\text{eff} \\
\text{clock} := 0 \\
\text{input} \quad \text{crash} \\
\text{eff} \\
\text{crashed} := \text{true}
\]

\[
\text{trajectories} \\
\text{stop when} \\
\neg \text{crashed} \land \text{clock} = u \\
\text{evolve} \\
\text{d} (\text{clock}) = 1
\]
TIOA Example 4.4: Timeout Process

Process that awaits the receipt of a message from another process, performing a timeout action if \( u \) time units elapse without receiving it.

```plaintext
automaton Timeout(u: Real, M: Type)
  signature
    input receive(m: M)
    output timeout
  states
    suspected: Bool := false,
    clock: Real := 0,
    initially u ≥ 0
  transitions
    input receive(m)
      eff
        clock := 0
        suspected := false
    output timeout
      pre
        ¬suspected ∧ clock = u
      eff
        suspected := true
  trajectories
    stop when
      ¬suspected ∧ clock = u
    evolve
d(clock) = 1

Alternatively:

trajectories
  trajdef suspected
    invariant suspected
    evolve d(clock) = 1
  trajdef notsuspected
    invariant ¬suspected
    stop when clock = u
    evolve d(clock) = 1
```
TIOA Example 4.5: Clock Synchronization (1/2)

- Process in a clock synchronization algorithm. Each process:
  - Has a physical clock, which may drift from the real time with a drift rate bounded by \( r \)
  - Generates a logical clock

- The goal of the algorithm is to achieve:
  - Agreement i.e. that the logical clocks are close to one another
  - Validity i.e. that the logical clocks are within the range of the physical clocks

- Idea is to periodically exchange the physical clock values between the different processes and set the logical clock to the maximum value of all the physical clock values
  - The logical clock, \( \text{logclock} \) is a derived variable, which is a function whose value is defined in terms of the state variables
TIOA Example 4.5: Clock Synchronization (2/2)

automaton ClockSync( \( u, r : \text{Real} \), \( i : \text{Index} \) )

signature

input receive( \( m : \text{Real} \), \( j : \text{Index} \), \( \text{const } i : \text{Index} \) ) where \( j \neq i \)

output send( \( m : \text{Real} \), \( \text{const } i : \text{Index} \) ),

states

nextsend: discrete Real := 0, 
maxother: discrete Real := 0, 
physclock: Real := 0, 
initially \( u \geq 0 \land (0 \leq r < 1) \)

derived variables

logclock = \max(\text{maxother, physclock})

transitions

output send( \( m, i \) )

pre

\( m = \text{physclock} \land \text{physclock} = \text{nextsend} \)

eff

\( \text{nextsend} := \text{nextsend} + u \)

input receive( \( m, j, i \) )

eff

\( \text{maxother} := \max(\text{maxother, } m) \)

trajectories

stop when \( \text{physclock} = \text{nextsend} \)

evolve \( (1-r) \leq d(\text{physclock}) \leq (1 + r) \)
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
TIOA Behavior: Executions

- Like with (non-timed) I/O automata, executions record what happens during a particular run of a system. In the case of a timed system this means:
  - all the instantaneous, discrete state changes
  - all the changes to the state that occur while time advances

**Execution** of a timed automata $\mathcal{A}$ is an alternating sequence

$\alpha = \tau_0 a_1 \tau_1 a_2 \ldots$ where:

1. each $\tau_i$ is a trajectory in $\mathcal{T}$
2. $\tau_0.fstate$ is a start state
3. if $\tau_i$ is not the last trajectory in $\alpha$ then $\tau_i.lstate \xrightarrow{a_{i+1}} \tau_{i+1}.fstate$

**Note** To allow for simultaneous actions, i.e. actions occurring at the same time instant, a special **point trajectory**, $\varphi(v)$, whose domain is the interval $[0,0]$ is defined

**Reachable state** is a last state of a closed execution, i.e. of an execution whose last trajectory is closed

**Invariant (assertion)** is a predicate that is true for all the reachable states of a TIOA
TIOA Behavior: Traces

The trace of an execution of a TIOA captures its external behavior. It consists of a sequence of alternating

External actions

- By definition, internal actions are not externally observable

Trajectories over the empty set of variables, $\emptyset$ – they capture the amount of time that elapses between external actions

- Trajectories describe the evolution in time of state variables
- State variables are internal, i.e. they are not externally visible
TIOA Behavior Ex. 4.9: Periodic Sending Process (1/2)

Consider the TA of Example 4.2 where,

- \( u \) is instantiated to the real number 3
- \( M \) is instantiated to the set \( \{m_1, m_2, \ldots\} \)

Then the following sequence is an execution of the automaton:

\[
\alpha = \tau \text{ send}(m_1) \tau \text{ send}(m_2) \tau \text{ send}(m_3) \tau \ldots
\]

where: \( \tau : [0, 3] \rightarrow \text{val}(\{\text{clock}\}) \) is defined such that 
\( \tau(t)(\text{clock}) = t \) for all \( t \in [0, 3] \)

- The function \( \tau \) is defined for closed intervals of length 3, starting at time 0
- It describes the evolution of the variable \( \text{clock} \), which is 0 at the start of \( \tau \) and increases with rate 1 for 3 time units
- The discrete \text{send} events occur periodically, every 3 time units and reset the \( \text{clock} \) variable to 0
TIOA Behavior Ex. 4.9: Periodic Sending Process (2/2)

The trace of the above execution is the sequence:

$$\text{trace}(\alpha) = \alpha' = \tau' \text{ send}(m_1) \tau' \text{ send}(m_2) \tau' \text{ send}(m_3) \tau' \ldots$$

where $\tau' : [0, 3] \rightarrow \text{val}(\emptyset)$

- $\text{trace}(\alpha)$ does not contain any information about what happens to the value of clock as time progresses
  - The range of function $\tau'$ contains only the function with the empty domain
- $\alpha$ and $\alpha'$ express the same information about the amount of time that elapses between discrete steps.
  - The domains of $\tau$ and $\tau'$ are identical,
TIOA Behavior Example 4.10: Timeout Process

Consider the TIOA of Example 4.4 where

- the maximum waiting time for a message \( u \) is 5
- the message alphabet \( M \) is the set \( \{m_1, m_2\} \)

Then the following sequence is an execution of the automaton:

\[
\alpha = \tau_0 \text{ receive}(m_1) \tau_1 \text{ timeout} \tau_2 \text{ send}(m_2) \tau_3 \text{ timeout} \tau_4
\]

where: \( \text{Val} = \text{val}(\{\text{suspected, clock}\}) \) and the trajectories \( \tau_0, \tau_1, \tau_2, \tau_3, \tau_4 \) are defined as follows:

\[
\begin{align*}
\tau_0 &: [0, 2] \rightarrow \text{Val} \text{ where } \tau_0(t)(\text{suspected}) = \text{false and } \tau_0(t)(\text{clock}) = t \text{ for all } t \in [0, 2] \\
\tau_1 &: [0, 5] \rightarrow \text{Val} \text{ where } \tau_1(t)(\text{suspected}) = \text{false and } \tau_1(t)(\text{clock}) = t \text{ for all } t \in [0, 5] \\
\tau_2 &: [0, 1] \rightarrow \text{Val} \text{ where } \tau_2(t)(\text{suspected}) = \text{true and } \tau_2(t)(\text{clock}) = 5 + t \text{ for all } t \in [0, 1] \\
\tau_3 &: [0, 5] \rightarrow \text{Val} \text{ where } \tau_3(t)(\text{suspected}) = \text{false and } \tau_3(t)(\text{clock}) = t \text{ for all } t \in [0, 5] \\
\tau_4 &: [0, \infty) \rightarrow \text{Val} \text{ where } \tau_4(t)(\text{suspected}) = \text{true and } \tau_4(t)(\text{clock}) = 5 + t \text{ for all } t \in [0, \infty)
\end{align*}
\]

- The automaton Timeout can perform multiple timeout transitions
- This execution is a finite alternating sequence ending with a trajectory
Consider the time-bounded channel from Example 4.1. Clearly,

- Time cannot pass beyond any delivery deadline recorded in the message queue
- Each deadline in the queue is less than or equal to the sum of the current time and bound b

We can state this property as an invariant (assertion) as follows:

**Invariant:** In any reachable state \( x \) of automaton `timedChannel`, for all\( p \in x(queue), x(now) \leq p.deadline \leq x(now) + b \)

Alternatively, we could write \( 0 \leq x(now) - p.deadline \leq b \)
Note that:

- Reachable states are the final states of closed executions
- Any closed execution is the concatenation of closed execution fragments, $\alpha_0 \bowtie \alpha_1 \bowtie \ldots \bowtie \alpha_k$, where every $\alpha_i$ is
  - either a closed trajectory
  - or a discrete action surrounded by point trajectories

and where $\alpha_i.lstate = \alpha_{i+1}.fstate$ for $0 \leq i \leq k - 1$.

Thus the invariant can be proved using induction on the length $k$ of the sequence of execution fragments $\alpha_i$. 
TIOA Behavior Ex. 4.11: Time-Bounded Channel (3/6)

**Invariant:** In any reachable state \( x \) of automaton `timedChannel`, for all \( p \in x(queue), x(now) \leq p.deadline \leq x(now) + b \)

**Proof** By induction on the length \( k \) of the sequence of execution fragments \( \alpha_i \):

**Base case:** \( k=1 \) In the initial state, \( x = \alpha_0.fstate \)

\[
x(queue) = \{}\]

Consider 2 cases:

1. \( \alpha_0 \) is an action surrounded by point trajectories
   Clearly the only action that may occur is a `send(m)`. Thus, for \( x = \alpha_0.lstate \) we have \( x(queue) = \{[m,b]\} \) and \( x(now) = 0 \), hence the invariant is satisfied

2. \( \alpha_0 \) is a trajectory
   Thus \( queue = \{} \), for all states in the trajectory and the invariant is trivially satisfied
TIOA Behavior Ex. 4.11: Time-Bounded Channel (4/6)

Invariant: In any reachable state \( x \) of automaton \( \text{timedChannel} \), for all \( p \in x(\text{queue}) \), \( x(\text{now}) \leq p.\text{deadline} \leq x(\text{now}) + b \)

Induction step Let’s assume that the invariant is true for an execution with \( k \) trajectories \( \alpha_0 \sim \alpha_1 \cdots \sim \alpha_{k-1} \). Consider 2 cases:

1. \( \alpha_k \) is an action surrounded by point trajectories First note that \( \text{now} \) does not change (time does not advance) in \( \alpha_k \), i.e.
\[
\alpha_{k-1}.\text{lstate}(\text{now}) = \alpha_k.\text{fstate}(\text{now}) = \alpha_k.\text{lstate}(\text{now}).
\]
Now, the action can be either:

receive\((m)\) removes the packet at the head of \( \text{queue} \), and leaves the remaining packets in the queue. By the induction hypothesis, these packets, if any, satisfy the invariant

send\((m)\) appends a packet \([m,\alpha_k.\text{fstate}(\text{now})+b]\) to the \( \text{queue} \). Clearly, at \( \alpha_k.\text{lstate} \) this packet satisfies the invariant. The remaining packets, if any, were already in the queue, and by the induction hypothesis satisfy the invariant
TIOA Behavior Ex. 4.11: Time-Bounded Channel (5/6)

Invariant: In any reachable state $x$ of automaton `timedChannel`, for all $p \in x(queue)$, $x(now) \leq p.deadline \leq x(now) + b$

Induction step

2. $\alpha_k$ is a trajectory In this case the state of `queue` does not change, and `now` increases at the same rate as time. We consider two cases:
   i. `queue` is empty then the invariant is trivially true
   ii. otherwise we consider the two inequalities separately:
      $p.deadline \leq x(now) + b$ this results directly from the induction hypothesis and that `now` increases monotonically

$x(now) \leq p.deadline$ by the stopping clause, the predicate

$\exists p: \text{Packet } p \in \text{queue} \land (now = p.deadline)$

cannot be true except in $\alpha_k.lstate$.

Therefore for all states $x$ but the last in $\alpha_k$, for all packets $p$ in $x(queue)$ we have: $x(now) < p.deadline$

For $\alpha_k.lstate$ there may be some packet $p$ such that $x(now) = p.deadline$
TIOA Behavior Ex. 4.11: Time-Bounded Channel (6/6)

- In the `timedChannel` automaton, if instead of specifying the trajectory as:

  ```plaintext
  stop when
  ∃p: Packet p ∈ queue ∧ (now = p.deadline)
  evolve
d(now) = 1
  ```

  we had specified it as:

  ```plaintext
  stop when
  queue ≠ ∅ ∧ (now = head(queue).deadline)
  evolve
d(now) = 1
  ```

  we would have had some more work to prove the invariant

- The “low level” of TIOA is a mixed blessing:

  - On one hand, it forces us to consider every detail, making it hard to “prove” something that is not true
  - On the other hand, proving even an “obvious assertion” requires a lot of work
Outline

1. Introduction

2. Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3. Further Reading
Composition of TIOA: Introduction

- Allows an automaton representing a complex system to be constructed by composing automata representing individual system components.
- The composed automaton is built by matching:
  - each output action of the component automata
  - with input actions with the same name in different component automata
- When any component automaton performs a discrete step involving an action $a$, so do all component automata that have $a$ as an external action:
  - I.e. automata in a composed automaton synchronize on external actions with the same name.
Composition of TIOA: Definition

Compatible Automata  TIOA $A_1$ and $A_2$ are compatible if:

1. $X_1 \cap X_2 = \emptyset$, i.e. their (internal) variables are disjoint
2. $H_1 \cap A_2 = H_2 \cap A_1 = \emptyset$, i.e. the internal actions of one TA is disjoint from the actions of the other TA
3. $O_1 \cap O_2 = \emptyset$, i.e. their output actions are disjoint

Composition  If $A_1$ and $A_2$ are compatible then their composition $A_1 || A_2$ is defined to be the TA $A = (X, Q, \times, I, O, H, D, T)$ where:

- $X = X_1 \cup X_2$
- $Q = \{x \in val(X) \mid x[i] \in Q_i, i \in \{1, 2\}\}$, i.e. $Q = Q_1 \times Q_2$
- $\Theta = \{x \in Q \mid x[i] \in \Theta_i, i \in \{1, 2\}\}$, i.e. $\Theta = \Theta_1 \times \Theta_2$
- $O = O_1 \cup O_2$, $I = (I_1 \cup I_2) - O$ and $H = H_1 \cup H_2$
- For each $x, x' \in Q$ and each $a \in A$, $x \xrightarrow{a} A x'$ iff for $i \in \{1, 2\}$ either (1) $a \in A_i$ and $x[i] \xrightarrow{a} A_i x'[i]$, or (2) $a \notin A_i$ and $x[i] = x'[i]$
- $T \subseteq \text{trajs}(Q)$ is given by $\tau \in T \iff \tau \downarrow X_i \in T_i$, $i \in 1, 2$
Composition: Fundamental Properties

Composition  The composition of TIOAs is a TIOA

Theorem 7.2 If $A_1$ and $A_2$ are TIOAs, then $A_1||A_2$ is a TIOA

Executions  the execution fragments of a composition of TIOA project to give execution fragments of the component automata.

Lemma 5.2 Let $A = A_1||A_2$ and let $\alpha$ be an execution fragment of $A$. Then $\alpha[(A_1, X_1)]$ and $\alpha[(A_2, X_2)]$ are execution fragments of $A_1$ and $A_2$, respectively.

Traces  satisfy the following projection and pasting result:

Theorem 7.3 Let $A = A_1||A_2$. Then $\text{traces}_A$ is exactly the set of $(E, \emptyset)$-sequences whose restrictions to $A_1$ and $A_2$ are traces of $A_1$ and $A_2$, respectively. That is,

$$\text{traces}_A = \{\beta|\beta \text{ is an } (E, \emptyset)\text{-sequence and } \beta[(E_i, \emptyset)] \in \text{traces}_{A_i}, i \in \{1, 2\}\}$$
Composition Ex. 5.5: Periodic Process w/ Timeouts (1/8)

**Notation** To avoid name clashes, when necessary, we refer to an internal variable $v$ of TA $A$ in the composite TA as $A.v$.

Let $C$ be the composition of three automata from examples 4.1, 4.2, 4.4

$$C = \text{PeriodicSend} \ || \ \text{TimedChannel} \ || \ \text{Timeout}$$

where $M = \{m_1, \ldots, m_n\}$ and $b + \text{PeriodicSend}.u < \text{Timeout}.u$.

If $b < u_1$, where $u_1 = \text{PeriodicSend}.u$, the following sequence is a trace of $C$:

$$\alpha = u_1 \ \text{send}(m_1) \ \bar{b} \ \text{receive}(m_1) \ u_1-b \ \text{send}(m_2) \ \bar{b} \ \text{receive}(m_2) \ u_1-b \ldots$$

where $\bar{t}$ denotes the trace with domain $[0, t]$ and as range the set consisting of the function with the empty domain.
Composition Ex. 5.5: Periodic Process w/ Timeouts (2/8)

**Invariant 1** In any reachable state $x$ of $C$, $x($suspected$) = false$

Given that suspected is set to true upon occurrence of a timeout action, we will prove the following invariant:

**Invariant 2** In any reachable state $x$ of $C$,

1. if $x(queue)$ is not empty then there is a packet $p$ such that $p \in x(queue)$ and $p.deadline - x(now) < u2 - x(Timeout.clock)$
2. if $x(queue)$ is empty then $u1 - x(PeriodicSend.clock) + b < u2 - x(Timeout.clock)$

where $u1 = PeriodicSend.u$ and $u2 = Timeout.u$

which states that the variable Timeout.clock never reaches the point at which a timeout action occurs:

1. ensures that if there is any message in transit it will be delivered before there is a timeout
2. ensures that if there is no message in transit, a send action will occur early enough, so that no timeout will occur
Invariant 2 In any reachable state $x$ of $\mathcal{C}$,

1. if $x(queue)$ is not empty then there is a packet $p$ such that $p \in x(queue)$ and $p.deadline - x(now) < u_2 - x(Timeout.clock)$

2. if $x(queue)$ is empty then $u_1 - x(PeriodicSend.clock) + b < u_2 - x(Timeout.clock)$

where $u_1 = PeriodicSend.u$ and $u_2 = Timeout.u$

To prove this invariant we will follow the same approach as in the proof of invariant in Example 4.11

- I.e., we’ll use induction on the number of **elementary** execution fragments, i.e.
  - either a closed trajectory
  - or a discrete action surrounded by point trajectories
Invariant 2  In any reachable state $x$ of $C$,

1. if $x(queue)$ is not empty then there is a packet $p$ such that $p \in x(queue)$ and $p.deadline - x(now) < u2 - x(Timeout.clock)$

2. if $x(queue)$ is empty then $u1 - x(PeriodicSend.clock) + b < u2 - x(Timeout.clock)$

where $u1 = PeriodicSend.u$ and $u2 = Timeout.u$

Base case, $k = 1$  In this case, $\alpha_0$ must be a trajectory, and throughout this trajectory we have:

\[
\begin{align*}
  x(queue) &= \emptyset \\
  x(Pe\text{r}iodicSend.\text{clock}) &= x(Timeout.\text{clock})
\end{align*}
\]

Given that $u1 + b < u2$, it follows that $u1 - x(PeriodicSend.clock) + b < u2 - x(Timeout.clock)$
Invariant 2 In any reachable state $x$ of $C$, 

1. if $x$(queue) is not empty then there is a packet $p$ such that $p \in x$(queue) and $p$.deadline $- x$(now) $< u2 - x$(Timeout.clock) 

2. if $x$(queue) is empty then $u1 - x$(PeriodicSend.clock) $+ b < u2 - x$(Timeout.clock) 

where $u1 = $ PeriodicSend.u and $u2 = $ Timeout.u 

Induction step We consider two cases: 

$\alpha_{k-1}.lstate(queue) = \{\}$ 

$\alpha_{k-1}.lstate(queue) \neq \{\}$ 

and for each of these two cases we need to consider the two possible types of execution fragments: 

- either a closed trajectory 
- or a discrete action surrounded by point trajectories
Composition Ex. 5.5: Periodic Process w/ Timeouts (6/8)

Induction step: \( \alpha_{k-1}.lstate(queue) = \emptyset \) By the inductive hypothesis:

\[
u_1 - x(\text{PeriodicSend.clock}) + b < u_2 - x(\text{Timeout.clock})\]

\( \alpha_k \) is a closed trajectory whose initial state satisfies this inequality.

Given that the derivatives of both PeriodicSend.clock and Timeout.clock are 1, it holds true for all states in \( \alpha_k \).

\( \alpha_k \) is an action surrounded by point trajectories In this case the only possible action is a \( \text{send}(m) \). As a result, for \( x = \alpha_k.lstate \) we have:

\[
x(queue) = \{[m, x(now)+b]\}
\]

Thus \( p.\text{deadline} - x(now) = b \) for this packet. Given that time does not advance in \( \alpha_k \), and that \( u_1 - x(\text{PeriodicSend.clock}) \geq 0 \), from the induction hypothesis it follows that

\[
p.\text{deadline} - x(now) < u_2 - x(\text{Timeout.clock})
\]

is satisfied by the only packet in the queue at \( x = \alpha_k.lstate \).
Composition Ex. 5.5: Periodic Process w/ Timeouts (7/8)

Induction step: $\alpha_{k-1}.\lt{\text{state}(\text{queue}) \neq \{\}}$ Thus, by the inductive hypothesis at $\alpha_k.\lt{\text{state}}$ there is a packet $p \in x(\text{queue})$ such that $p.\text{deadline} - x(\text{now}) < u2 - x(\text{Timeout.clock})$

$\alpha_k$ is a closed trajectory then given that the derivatives of both $\text{now}$ and $\text{Timeout.clock}$ are 1, the inequality above holds true in all states of $\alpha_k$ for that packet

$\alpha_k$ is an action surrounded by point trajectories In this case the action can be either

send($m$) in this case, given that the time does not advance, the inequality will continue to hold true for that packet at $x = \alpha_k.\lt{\text{state}}$

receive($m$) in this case we need to consider two cases, either the queue becomes empty or it does not.
Composition Ex. 5.5: Periodic Process w/ Timeouts (8/8)

Induction step: $\alpha_{k-1}.lstate(queue) \neq \{\}$ Thus, by the inductive hypothesis at $\alpha_k.fstate$ there is a packet $p \in x(queue)$ such that $p.deadline - x(now) < u2 - x(Timeout.clock)$

$\alpha_k$ is receive(m) surrounded by point trajectories Either:

queue becomes empty From the parameter assumptions, we have: $u1 + b < u2$. Furthermore, at $x = \alpha_k.lstate$, $x(Timeout.clock) = 0$, and for all states $x$ we have and $x(PeriodicSend.clock) \geq 0$, thus it follows that at $x = \alpha_k.lstate$:

$$u1 - x(PeriodicSend.clock) + b < u2 - x(Timeout.clock)$$

otherwise in this case from invariant in Example 4.11 we have that for all packets in queue and all sates:: $x(now) \leq p.deadline \leq x(now) + b$

Thus, $p.deadline - x(now) \leq b$

From, the parameters assumption, and given that $u1 \geq 0$ it follows that: $p.deadline - x(now) \leq b \leq u1 + b < u2$

Finally, for $x = \alpha_k.lstate$ given that $x(Timeout.clock) = 0$ it follows that $p.deadline - x(now) < u2 - x(Timeout.clock)$
Outline

1 Introduction

2 Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3 Further Reading
Timed I/O Automata with Bounds: Rationale

- **TIOA with bounds** are a new class of TIOA that extends TIOA with:
  - Tasks which are sets of locally controlled actions
  - Bounds which impose constraints on the time when an action may be performed
- This class makes it easier to present many results on the partially synchronous model that assume that there are bounds (both lower and upper) on the time processes take to perform a step of an algorithm
- We’ll restrict our attention to a class of automata where every action:
  1. either is enabled/disabled throughout an entire trajectory
  2. or becomes enabled once during a trajectory and remains so until the end of that trajectory

A trajectory $T$ that satisfies this property wrt to a set of actions $C$ is said to be **well-formed** wrt $C$
Time I/O Automata with Bounds: Definition

A TIOA with bounds \( \mathcal{A} = (X, Q, \Theta, I, O, H, D, \mathcal{T}, C, l, u) \) consists of:

\[
(X, Q, \Theta, I, O, H, D, \mathcal{T}) \quad \text{a TIOA}
\]

\( C \subseteq I \cup O \cup H \) i.e. a set of actions

- \( C \) is called a task
- \( \mathcal{T} \) is well-formed wrt \( C \)

\( l \) a lower bound \( l \in R^{\geq 0} \)

\( u \) an upper bound \( u \in R^{\geq 0} \cup \{\infty\} \), with \( l \leq u \)

- Lower and upper bounds are used to specify how much time is allowed to pass between the enabling and the performance of an action:
  - **Lower bound** \( l \) is the minimum time that an action must be enabled before it is performed
  - **Upper bound** \( u \) is the maximum time that an action may be enabled without being performed:
    - i.e. it must either be performed or become disabled after \( u \) time units
TIOA with Bounds Example: Timeout Process (1/2)

- $P_2$ 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 $\ell_1, \ell_2$, $0 < \ell_1 \leq \ell_2 < \infty$:
  - In Example 4.4 above, we used a local clock
- Its *timeout* action is performed at most time $\ell$ after its *count* reaches 0.

Note that the definition of the TIOA with bounds assumes the existence of only one task per TIOA, but it can be easily generalized to TIOA with an arbitrary number of tasks.
Timed I/O Automata

Automaton Timeout (k : Int, M: Type)

Signature
- Input: receive (m: M)
- Internal: decrement
- Output: timeout

States
- Suspected: Bool := false,
- Counter: Int := k,
- Initially k ≥ 1

Transitions
- Input: receive (m, j, i)
  - Eff
    - Counter := k
    - Suspected := false
- Output: timeout
  - Pre
    - Suspected = false ∧ Counter = 0
  - Eff
    - Suspected = true

Internal decrement
- Pre
  - Counter ≠ 0
- Eff
  - Counter := counter − 1

Tasks
- Decr = \{decrement\};
- Susp = \{timeout\}

Bounds
- Decr = [ℓ₁, ℓ₂];
- Susp = [0, ℓ]
Time I/O Automata with Bounds: The **Extend** Operation

The **Extend** operation transforms a TIOA $A = (B, C, l, u)$ with bounds to another TIOA $A' = (X', Q', \Theta', I, O, H, D', T')$ that incorporates the bounds in addition to the timing constraints already present in $B$.

Basically:

- $X' = X \cup \{\text{now}, \text{first}, \text{last}\}$, where
  - $\text{now}$ is an analog variable such that $\text{type}(\text{now}) = R$
  - $\text{first}$ and $\text{last}$ are discrete variables where $\text{type}(\text{first}) = R$ and $\text{type}(\text{last}) = R \cup \{\infty\}$

Variables $\text{now}, \text{first}, \text{last}$ are new variables that do not appear in $X$

- $Q' = Q \times \text{val}(\text{now}, \text{first}, \text{last})$

- $\Theta'$ is obtained from $\Theta$ by assigning the following valuations to $x \in \Theta$:
  - $x(\text{now}) = 0$
  - $x(\text{first}) = \begin{cases} l & \text{if } C \text{ is enabled in } x \\ 0 & \text{otherwise} \end{cases}$
  - $x(\text{last}) = \begin{cases} u & \text{if } C \text{ is enabled in } x \\ \infty & \text{otherwise} \end{cases}$
Time I/O Automata with Bounds: The **Extend** Operation

- $D'$ is obtained from $D$ by adding the predicate $x(first) \leq x(now)$ for transitions $x \xrightarrow{a} x'$, when $a \in C$
- $T'$ is obtained from $T$ by adding the following trajectory:
  
  ```
  stop when now \leq last
  evolve d(now) = 1
  ```

Note that $now$, $first$ and $last$ all represent absolute time:

- A full formal definition requires that the values of $first$ and $last$ be updated whenever actions in $C$ are enabled, disabled or performed
  
  - I.e. the “definition” above omitted details on how $D'$ and $T'$ are obtained from $D$ and $T$
TIOA with Bounds and the Extend Operation

**Theorem 5.18** Suppose that $\mathcal{A}$ is a TIOA with bounds. Then

$$traces_{\text{Extend}(\mathcal{A})} \subseteq traces_{\mathcal{A}}$$

I.e. it is possible to implement a TIOA with bounds with a TIOA without bounds.

**Note** the definition of the Extend operation assumes the existence of only one task per TIOA, but, like the definition of the TIOA with bounds, it can be easily generalized to TIOA with an arbitrary number of tasks.
Outline

1. Introduction

2. Timed I/O Automata
   - Preliminaries
   - Definition
   - Examples
   - TIOA Behavior
   - Composition of TIOA
   - Timed I/O Automata with Bounds

3. Further Reading
Further Reading


