# Tools and Algorithms for Deciding Timed Relations

Mihir Mehta
Department of Computer Science and Engineering,
Indian Institute of Technology, Delhi.
`cs1090197@cse.iitd.ac.in`

December 2012

**Abstract**

This is a report summarising the author's project on their B Tech Project for the academic year 2012-2013.

## 1 Objectives

- To develop a software toolkit that would enable users to verify various timed relations specifications and implementations expressed as timed automata.
  - To gain an understanding of the theory related to labeled transition systems, CCS processes and timed automata by surveying relevant literature.
  - To study tools already built by researchers for similar purposes.
  - To develop the software in a modular way with modules for language specification and modules for implementations of utility algorithms.
  - To implement algorithms for determining timed relations.

## 2 CCS processes

- A CCS process is an automaton with state and interfaces for interaction.
- The interaction is in the form of *actions* over communication ports known as *channels*.
- Given a port name $a$ we refer to $a$ as the label for input on the port and $\overline{a}$ as the label for the output on the port.

- *Inaction*: This is the simplest CCS process, denoted by 0. No state transitions or communcation can occur, in other words, this represents a deadlock.

- *Prefixing*: This is the simplest constructor; if $P$ is a process and $a$ is a label (input or output) then $a.P$ is also a process which can perform the action $a$ in order to become the process P. Thus, $dot.dash.0$ is a CCS process capable of transmitting the Morse code letter $A$, and $\overline{dot}.\overline{dash}.0$ is a process capable of receiving it.

- *Naming*: We can give names to processes using syntax such as
  $M \overset{def}{=} dot.dash.0$
  $N \overset{def}{=} \overline{dot}.\overline{dash}.0$
  This gives us the ability to define CCS processes recursively, such as this one:
  $Repeater\ \overset{def}{d}\ ot.dash.Repeater$
  which is a process that continuously repeats the Morse code letter $A$.

- *Choice*: If $P$ and $Q$ are processes, then $P + Q$ is a process as well which has the initial capabilities of both P and Q. The deadlock process 0 is the identity element for this, that is, $P + 0 = P$ is an identity.
  Thus, the process $M + N$ could either transmit or receive the letter $A$.

- *Parallel Composition*: If $P$ and $Q$ are processes, then $P|Q$ is a process as well in which P and Q may proceed independently or communicate via complementary ports.
  Thus, the process $M|N$ could proceed with $M$ transmitting the letter $A$ to be received by $N$, or they could communicate with other processes using the *dot* and *dash* ports.

- *Restriction*: If $P$ is a process and $L$ is a set of channel names, then $P/L$ is a process in which the component processes of $P$ are the only processes which can communicate over channels from the set $L$.
  Thus, $(M|N)/\{dot, dash\}$ would be a process in which $M$ and $N$ can only communicate with each other.

- *Relabeling*: If $P$ is a process and $f$ is a function from labels to labels, then $P[f]$ is a process where each label from the domain of $f$ is replaced by its image under $f$. One application of relabelling is the idea of *generic* processes: By relabelling the generic ports of such a process with specific port names, one can generate specific processes.

It is evident that each CCS process can be replaced by a labeled transition system (LTS) with equivalent behaviour, therefore we will, in the rest of this discussion, freely use the properties of LTS when describing those of CCS.

# 3 Equivalences on CCS

## 3.1 Trace equivalence

- A *trace* of an LTS is a sequence of actions that the LTS can perform.
- For an LTS $P$, the set $Traces(P)$ represents the set of all possible traces of $P$.
- Trace equivalence is said to exist between two LTS $P$ and $Q$ when $Traces(P) = Traces(Q)$.
- However, this notion proves to have a significant limitation in the case of CCS processes: Two CCS processes can have trace equivalence between their corresponding LTS and yet behave differently in terms of when they deadlock while interacting with a third CCS process.

## 3.2 Strong bisimilarity

- *Strong bisimulation*: A binary relation $R$ is a strong bisimulation if and only if, for all $(s_1, s_2)\epsilon R$ and $a\epsilon Act$.
  $\forall s'_1(s_1 \xrightarrow{a} s'_1 \Rightarrow \exists s'_2.(s_2 \xrightarrow{a} s'_2 \wedge (s'_1, s'_2)\epsilon R))\wedge$
  $\forall s'_2(s_2 \xrightarrow{a} s'_2 \Rightarrow \exists s'_1.(s_1 \xrightarrow{a} s'_1 \wedge (s'_1, s'_2)\epsilon R))$
- *Strong bisimilarity*: It can be shown that the union of all strong bisimulations over the set of states is a strong bisimulation. This binary relation is called strong bisimilarity, denoted by $\sim$.
- This mitigates one of the failings of trace equivalence as an equivalence relation: strong bisimilarity between two CCS processes ensures identical deadlock behaviour while interacting with a third CCS process.
- However, another limitation soon becomes apparent: if two CCS processes are to be strongly bisimilar, they must coincide even on the number and position of $\tau$ transitions in their traces. This is contrary to the semantics of CCS processes, as a $\tau$ transition is supposed to be private to a process and invisible to all other processes in its environment.

## 3.3 Weak bisimilarity

- *Weak bisimulation*: A binary relation $R$ is a `weak bisimulation` if and only if, for all $(s_1, s_2)\epsilon R$ and $a\epsilon Act$.
  $\forall s'_1(s_1 \xrightarrow{a} s'_1 \Rightarrow \exists s'_2.(s_2 \overset{a}{\Rightarrow} s'_2 \wedge (s'_1, s'_2)\epsilon R))\wedge$
  $\forall s'_2(s_2 \xrightarrow{a} s'_2 \Rightarrow \exists s'_1.(s_1 \overset{a}{\Rightarrow} s'_1 \wedge (s'_1, s'_2)\epsilon R))$
- It can be shown that the union of all weak bisimulations over the set of states is a weak bisimulation. This binary relation is called `weak bisimilarity`, denoted by $\approx$.

- Better suited to CCS processes, as it ignores $\tau$ transitions, thus disregarding hidden behaviour within a process.

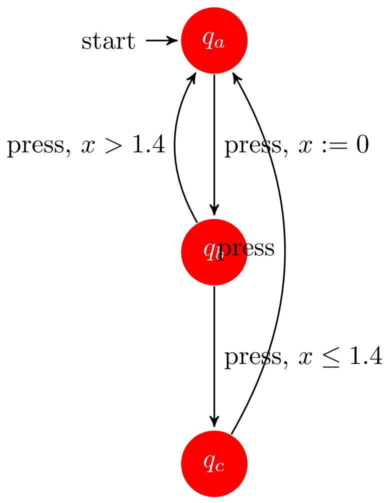
## 3.4 Kanellakis and Smolka's algorithm

- This is a naive algorithm for determining the bisimilarity relation for the set of processes in a labelled transition system.
- This relies on the notion of a splitter.
- Let $\pi = \{B_0, ..., B_k\}, k \geq 0$ be a partition of the set of states $Pr$ in a labeled transition system.
- A splitter for a block $B_i \epsilon \pi$ is a block $B_j \epsilon \pi$ such that for some action $a \epsilon Act$, some states in $B_i$ have $a$-labelled transitions whose targets lie in $B_j$ while other states in $B_i$ do not.
- This suggests a refinement of $\pi$: replace block $B_i$ with
  $B_i^1 = B_i \cap T_a^{-1}[B_j]$
  $B_i^2 = B_i - B_i^1$
- Refinements of this kind constitute the steps of this algorithm.
- The time complexity of this algorithm is $O(mn)$, since there can be at most n iterations, and all m edges are scanned in each iteration.

## 3.5 Fernandez' algorithm

- This is a more efficient algorithm for determining bisimilarity (O(m log n)).
- This relies on the technique of three-way splitting introduced by Paige and Tarjan.
- Splitters can now be 'simple' or 'compound'.
- Stability: A partition $\pi$ is said to be stable with respect to a compound block S if S is not a splitter for any block in $\pi$ for any action.
- For a compound block S, having a constituent simple block B satisfying $n(B) \leq 0.5 * n(S)$, and with respect to which $\pi$ is stable, we can split a block $B_i$ on an action $a$ as follows:
  $B_i^1 = (B_i \cap T_a^{-1}[B]) - T_a^{-1}[S - B]$
  $B_i^2 = (B_i \cap T_a^{-1}[S - B]) - T_a^{-1}[B]$
  $B_i^3 = B_i \cap T_a^{-1}[B] \cap T_a^{-1}[S - B]$

## 4 Timed automata



Timed automaton representing a light bulb with two brightness settings

- Formally, a timed automaton over a finite set of clocks $C$ and a finite set of actions $Act$ is a 4-tuple $(L, l_0, E, I)$.
- $L$ is a finite set of locations.
- $l_0$ is the initial location.
- $E \subseteq L \times B(C) \times Act \times 2^C \times L$ is a finite set of edges.
- $I : L \to B(C)$ assigns invariants to each edge location.
- $B(C)$ is the set of clock constraints over C. An element of $B(C)$ can be an equality, a slack inequality, or a strict inequality on $v(x)$ for some $x \epsilon C$, or an AND combination of such constraints.
- The state of the automaton at any particular instant is given by the ordered pair $(l, v)$ which gives the location and assigns a value to each clock.
- A transition is either a delay transition where the automaton stays at the same location while advancing each clock by the same time delay, or an action transaction where the automaton performs a state change while resetting some of its clocks.

## 5 Equivalences on Timed Automata

### 5.1 Timed bisimilarity

- *Timed bisimulation*: A binary relation $R$ is a timed bisimulation if and only if, for all $(s_1, s_2) \epsilon R$ , $a \epsilon Act$, $d \epsilon R_{\geq 0}$

$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{a} s_2' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{a} s_1' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_1'(s_1 \xrightarrow{d} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{d} s_2' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_2'(s_2 \xrightarrow{d} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{d} s_1' \wedge (s_1', s_2')\epsilon R))$

- It can be shown that the union of all timed bisimulations over the set of (location, valuation) pairs is a timed bisimulation. This binary relation is called `timed bisimilarity`, denoted by $\sim$.

## 5.2 Time abstracted bisimilarity

- *Time abstracted bisimulation*: A binary relation $R$ is a time abstracted bisimulation if and only if, for all $(s_1, s_2)\epsilon R$ , $a\epsilon Act$, $d\epsilon R_{\geq 0}$
$\forall s_1'(s_1 \xrightarrow{a} s_1' \Rightarrow \exists s_2'.(s_2 \xrightarrow{a} s_2' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_2'(s_2 \xrightarrow{a} s_2' \Rightarrow \exists s_1'.(s_1 \xrightarrow{a} s_1' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_1'(s_1 \xrightarrow{d} s_1' \Rightarrow \exists (s_2', d').(s_2 \xrightarrow{d} s_2' \wedge (s_1', s_2')\epsilon R))\wedge$
$\forall s_2'(s_2 \xrightarrow{d} s_2' \Rightarrow \exists (s_1', d').(s_1 \xrightarrow{d} s_1' \wedge (s_1', s_2')\epsilon R))$

- It can be shown that the union of all time abstracted bisimulations over the set of (location, valuation) pairs is a time abstracted bisimulation. This binary relation is called `time abstracted bisimilarity`, denoted by $\sim_u$.

## 5.3 Regions and region graphs

One problem with the notion of valuations is that since they map clock names to real values, it can be difficult to come up with finite-complexity terminating algorithms for determining properties such as reachability. However, by introducing the notion of equivalence between two valuations in the context of a timed automaton, we can divide the uncountably infinite set of valuations into a finite number of equivalence classes (called regions) which allows us to derive some useful notions.

### 5.3.1 Equivalence of valuations

Two valuations $v$ and $v'$ of a timed automaton are said to be equivalent ($v \equiv v'$) if and only if:

- For each $x\epsilon C$, either both $v(x)$ and $v'(x)$ are greater than $c_x$ or $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$
- For each $x\epsilon C$ such that $v(x) \leq c_x$, $frac(v(x)) = 0$ if and only if $frac(v'(x)) = 0$.

- For all $x, y \epsilon C$ such that $v(x) \leq c_x$ and $v(y) \leq c_y$, we have $frac(v(x)) \leq frac(v(y))$ if and only if $frac(v'(x)) \leq frac(v'(y))$.

Under this notion of equivalence, an equivalence class is known as a *region*. The equivalence class containing a valuation $v$ is denoted by $[v]_{\equiv}$.

### 5.3.2 Region graph

With these definitions, we can define the region graph of a timed automaton as a LTS representation of the automaton where the action transitions are labelled with the same actions and the delay self-transitions are labelled with $\varepsilon$. Formally, the region graph of a timed automaton $A$ with clock set $C$ and action set $Act$ is an LTS
$T_r(A) = (Proc, Act \cup \{\varepsilon\}, \{\xrightarrow{a} | a \epsilon Act \cup \{\varepsilon\}\})$
where $Proc = \{(l, [v]_{\equiv}) | l \epsilon L, v : C \to R_{\geq 0}\}$ (these states are called symbolic states)
The transitions are defined as follows:

- For each $a \epsilon A$, $(l, [v]_{\equiv}) \overset{a}{\Rightarrow} (l', [v']_{\equiv})$ iff $(l, v) \xrightarrow{a} (l', v')$
- $(l, [v]_{\equiv}) \xrightarrow{a} (l, [v']_{\equiv})$ if for some $d \epsilon R_{\geq 0}$, $(l, v) \xrightarrow{d} (l, v')$

## 5.4 Zones and reachability graphs

Though the region graph provides some useful results about time abstracted bisimilarity, its state space can grow very fast as a function of the number of clocks (the upper bound on the number of states happens to be exponential in the number of clocks). Using zones, we can represent the state space of a timed automaton in a more compact way and get some useful results about reachability.

### 5.4.1 Extended clock constraints

These are described by the grammar
$g ::= x \bowtie n | x - y \bowtie n | g_1 \wedge g_2$
The set of extended clock constraints is denoted by $B^+(C)$

### 5.4.2 Zones

A zone is a set of valuations represented by an extended clock constraint.
$Z = v | v \epsilon g_Z$

### 5.4.3 Symbolic state

For a timed automaton $A$, a symbolic state is an ordered pair of a location and a zone, that is, $(l, Z)$.

#### 5.4.4 Reachability graph

The reachability graph of a timed automaton is a directed graph in which the nodes are symbolic states and the edges are given by the *reachability relation*:

- $(l, Z')$ is reachable from $(l, Z)$ if there exists a valuation belonging $Z'$ which can be obtained from Z by applying a time delay which does not violate the invariant at $l$.

- $(l', Z')$ is reachable from $(l, Z)$ if $l$ has an action transition to $l'$ and there exists a valuation in $Z'$ which violates neither the guard of the transition nor the invariant at $l'$.

# 6 Code written so far

## 6.1 Fernandez' algorithm

We implemented Fernandez' algorithm in Ocaml to be used as a module in further applications.

## 6.2 Lexical analyser and Parser

Using a grammar very similar to that employed by the research tool Minim (Stavros Tripakis et al), we implemented an Ocaml program to parse a timed automaton from an input file.

The grammar specification we used generates a representation of the timed automaton which is similar to an adjacency list representation of a labelled graph.