EECE 396-1
Hybrid and Embedded Systems: Computation

T. John Koo, Ph.D.
Institute for Software Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University

300 Featheringill Hall
April 6-15, 2004

john.koo@vanderbilt.edu
http://www.vuse.vanderbilt.edu/~kootj
Analysis: Timed Automata
Topics

- Bisimular Transition Systems
  - Transition Systems
  - Forward and Backward Reachability Algorithms
  - Simulation and Bisimulation Relations
  - Transition Systems and Quotient Transition Systems
  - Bisimulation between Transition Systems
  - Bisimulation Algorithm

- Bisimulations of Timed Automata
  - Executions of Timed Automata
  - Region Equivalent Relation and Partition

Ref:
Analysis: Timed Automata
Bisimular Transition Systems
Transition System

Definition 1 Transition Systems
A transition system $T = (S, \Sigma, \rightarrow, S_I, S_F)$ consists of

- A set $S$ of states;
- An alphabet $\Sigma$ of events;
- A transition relation $\rightarrow \subseteq S \times \Sigma \times S$;
- A set $S_I \subseteq S$ of initial states;
- A set $S_F \subseteq S$ of final states;

Example: A nondeterministic finite automaton

Note:
1. It is customary to denote a transition $(s_1, \sigma, s_2) \in \rightarrow$ as $s_1 \xrightarrow{\sigma} s_2$ where $s_1, s_2 \in S$
2. The transition system is finite, if the cardinality of $S$, $|S|$, is finite, and it is infinite otherwise.
3. The transition system is deadlock-free, if for any state $s_1 \in S$, there exists a state $s_2 \in S$ and an event $\sigma \in \Sigma$ such that $s_1 \xrightarrow{\sigma} s_2$. 

Transition System

Example 1 Transition System

Consider a transition system $T = (S, \Sigma, \rightarrow, S_s, S_f)$ consisting of

- $S = \{q_0, q_1, \ldots, q_6\}$;
- $\Sigma = \{a, b, c\}$;
- $\rightarrow = \{(q_0, a, q_1), (q_0, a, q_2), (q_0, b, q_0), \ldots\}$;
- $S_s = \{q_0\}$;
- $S_f = \{q_3, q_6\}$;
Definition 1 Reachability Problem

Given a transition system $T = (S, \Sigma, \rightarrow, S_S, S_F)$. Is a state $s_2 \in S_F$ reachable from a state $s_1 \in S_S$ by a sequence of transitions?

In other words, we can check if

a) $Post^*(S_S) \cap S_F \neq \emptyset$, or

b) $S_S \cap Pre^*(S_F) \neq \emptyset$. 

Diagram: [Transition System Diagram]
Transition System

In general, one can use temporal logic formula to formulate more complicated reachability problem. Given transition system $T$, and temporal logic formula $\varphi$. The basic verification problem is

$$T \models \varphi$$

where $\varphi$ can be specified by using proper temporal logics such as LTL, CTL, or CTL*
Forward and Backward Reachability Algorithms

For checking the condition $Post^*(S_S) \cap S_F \neq \emptyset$, the Forward Reachability Algorithm can be used.

For checking the condition $S_S \cap Pre^*(S_F) \neq \emptyset$, the Backward Reachability Algorithm can be used.
Forward and Backward Reachability Algorithms

Algorithm (Forward Reachability Algorithm)
set $R := S_S$
while true do
  if $R \cap S_F \neq \emptyset$ return unsafe; stop
  if $Post(R) \subseteq R$ return safe; stop
  else $R := R \cup Post(R)$
end while
Forward and Backward Reachability Algorithms

Algorithm (Backward Reachability Algorithm)

set $R := S_F$

while true do

    if $R \cap S_S \neq \emptyset$ return unsafe; stop
    if $Pre(R) \subseteq R$ return safe; stop
    else $R := R \cup Pre(R)$

end while
Forward and Backward Reachability Algorithms

The algorithms can be implemented and always terminate in finite steps for the finite transisiton system. Hence, the reachability problem is decidable. This is because the numbers of states and transitions are finite. By enumeration, one can perform reachability analysis. For large scale transition systems, sets of states are not enumerated, but are represented more compactly by using Binary Decision Diagram (BDD).
Simulation and Bisimulation Relations

Consider two transition systems $T = (S, \Sigma, \rightarrow, S_S, S_F)$ and $T' = (S', \Sigma, \rightarrow', S'_S, S'_F)$ over the same sets of symbols.

What can we say about $T = (S, \Sigma, \rightarrow, S_S, S_F)$ and $T' = (S', \Sigma, \rightarrow', S'_S, S'_F)$?
Simulation and Bisimulation Relations

Definition 1 Simulation Relation

Given transition systems $T = (S, \Sigma, \rightarrow, S_S, S_F)$ and $T' = (S', \Sigma, \rightarrow', S_{S'}, S_{F'})$. A relation $\sim \subseteq S \times S'$ is called a simulation relation if:

1) $(s_1 \sim s'_1) \land (s_1 \in S_S) \Rightarrow (s'_1 \in S'_{S})$

2) $(s_1 \sim s'_1) \land (s_1 \in S_F) \Rightarrow (s'_1 \in S'_{F})$

3) $(s_1 \sim s'_1) \land ((s_1, \sigma, s_2) \in \rightarrow) \Rightarrow \exists s'_2$ s.t. $(s_2 \sim s'_2) \land ((s'_1, \sigma, s2') \in \rightarrow')$
Simulation and Bisimulation Relations

Define $\sim \subseteq S \times S'$ as:
$\sim = \{(q_0, p_0), (q_1, p_1), (q_2, p_1), (q_4, p_4), (q_5, p_4), (q_3, p_3), (q_6, p_3)\}$

It can be easily checked that $\sim$ is a simulation relation. If a simulation relation exists, then we can say that $T'$ can simulate $T$
or

$T'$ is similar to $T$. 

![Diagram](image-url)
Can you use the same $\sim \subseteq S \times S'$ to show that $T$ is similar to $T'$?
Simulation and Bisimulation Relations

If both statements

$T'$ is similar to $T$

and

$T$ is similar to $T'$

are true for the same simulation relation $\sim$, then we can say that

$T$ and $T'$ are bisimular.
Simulation and Bisimulation Relations

If

\[ T \text{ and } T' \text{ are bisimular} \]

then

\[ T \models \varphi \iff T' \models \varphi \]

and

\[ L(T) = L(T') \]

where \( \varphi \) can be specified in LTL, CTL or CTL*
Simulation and Bisimulation Relations

Since $T$ and $T'$ are bisimilar, checking $T' \models \varphi$ is equivalent to checking $T \models \varphi$. Therefore, the complexity of the problem is greatly reduced.

However, if one can only show that $T'$ is simular to $T$, the following result can be used:

If $T'$ is simular to $T$ then $T' \models \varphi \Rightarrow T \models \varphi$
Simulation and Bisimulation Relations

On the other hand, if one wants to solve the Language inclusion problem by checking whether $T$ is correct with respect to a specification $S$, i.e.

$$L(T) \subseteq L(S),$$

then the following problem

$$L(T') \subseteq L(S),$$

can be used instead of the original one.
Simulation and Bisimulation Relations

Given two transition systems $T$, $T'$, and an LTL formula. We have the following useful results:

Language equivalence
If $L(T) = L(T')$ then $T \models \varphi \iff T' \models \varphi$

Language inclusion
If $L(T) \subseteq L(T')$ then $T' \models \varphi \Rightarrow T \models \varphi$

However, the Language equivalence and inclusion problems are hard to check.
Simulation and Bisimulation Relations

However, Simulation implies language inclusion. Given two transition systems $T$, $T'$. We have the following useful result:

Simulation implies language inclusion

If $T'$ is similar to $T$ then $L(T) \subseteq L(T')$

Again, the following problem

$$L(T') \subseteq L(S),$$

can be used instead of the original one.
Bisimulation Algorithm

Given a transition system $T = (S, \Sigma, \rightarrow, S_S, S_F)$. Can we construct another transition system over the same $\Sigma$ such that it is bisimilar to $T$?

What we are interested here is to construct a bisimilar transition system over the same alphabet with less number of states and transitions.

How?
Bisimulation Algorithm

To define (state) partition, we can use the concept called Equivalence Relation.

**Definition 1** A (binary) relation $R$ on a set $S$ is a subset of $S \times S : R \subseteq S \times S$.

For each ordered pair $(a, b)$ of elements of $S$ the statement $(a, b) \in R$ symbolizes the assertion that “$a$ is related by $R$ to $b$.” Notice that $aRb$ is an equivalent way of writing $(a, b) \in R$.

**Definition 2** A relation $R$ on a set $S$ is called **reflexive** if $\forall a \in S, \ aRa$;

**symmetric** if for $a, b \in S, \ aRb \Rightarrow bRa$;

**transitive** if for $a, b, c \in S, \ aRb \land bRc \Rightarrow aRc$.

**Definition 3** A relation $R$ on a set $S$ is called an **equivalence relation** on $S$ if $R$ is reflexive, symmetric, and transitive.
**Bisimulation Algorithm**

**Definition 4** Let $R$ be an equivalence relation on $S$. For any $a \in S$, a subset of $S$ defined by $[a] = \{b \in S : (a, b) \in R\}$ is called the **equivalence class of** $a$.

There are two facts about equivalence classes:

**Lemma 1** Let $R$ be an equivalence relation on $S$. Every element $a \in S$ belongs to at least one equivalence class.

**Lemma 2** Let $R$ be an equivalence relation on $S$. No element $a \in S$ can belong to two different equivalence classes, i.e., for $a, b, c \in S$, $b \in [a] \land b \in [c] \Rightarrow [a] = [c]$. 
Bisimulation Algorithm

Definition 5 Let $\pi$ be a collection of nonempty subsets of $S$. $\pi$ is called a partition of $S$ if
1. $\forall S_i, S_j \in \pi, S_i \neq S_j \Rightarrow S_i \cap S_j = \emptyset$, and
2. $\bigcup_{S_i \in \pi} S_i = S$.

Theorem 1 If $R$ is an equivalence relation on a set $S$, then the equivalence classes of $R$ constitutes a partition of $S$.

On the other hand, given a partition of $S$, one can define an induced equivalence relation over $S$. 
Bisimulation Algorithm

Define an equivalence relation $\approx$ over $S$. $S$ is partitioned into a number of equivalence classes, $S_i$. Let $S/\approx = \{S_i\}$ denote the quotient space which is also called the partition of $S$. The quotient space consists of all equivalence classes.

Given a set $P \subseteq S$, let $P/\approx$ represent the part of the quotient space with which $P$ overlaps:

$$P/\approx = \{S_i \in S/\approx : S_i \cap P \neq \emptyset\}.$$ 

Hence, $P/\approx \subseteq S/\approx$.

$$S/\approx = \{S_1, S_2, S_3, S_4\} \quad P/\approx = \{S_1, S_3, S_4\}$$
Bisimulation Algorithm

Given a transition system $T = (S, \Sigma, \rightarrow, S_S, S_F)$ and an equivalent relation $\approx$ over $S$. We can define the quotient transition system as

$$T/ \approx = (S/ \approx, \Sigma, \rightarrow_\approx, S_S/ \approx, S_F/ \approx)$$

where $S_1, S_2 \in S/ \approx$, $(S_1, \sigma, S_2) \in \rightarrow_\approx$ if and only if there exists $s_1 \in S_1$ and $s_2 \in S_2$ such that $(s_1, \sigma, s_2) \in \rightarrow$. 

![Bisimulation Algorithm Diagram]
Bisimulation Algorithm

For $\sigma \in \Sigma$ define the $Pre_\sigma : 2^S \rightarrow 2^S$ operator as:

$$Pre_\sigma(P) = \{ s \in S : \exists s' \in P \text{ such that } (s, \sigma, s') \in \rightarrow \}$$

What is $Pre_a(S_1)$? \{q_0, q_1, q_2\}
What is $Pre_b(S_3)$? \{q_1\}
What is $Pre_c(S_3)$? \{q_1\}
Definition 1 Bisimulation
Given transition systems $T = (S, \Sigma, \rightarrow, S_S, S_F)$ and equivalence relation, $\approx$, over $S$, $\approx$ ia called a bisimulation if:
1) $S_S$ is a union of equivalence classes;
2) $S_F$ is a union of equivalence classes;
3) For all $\sigma \in \Sigma$, if $P$ is a union of equivalence classes, $Pre_\sigma(P)$ is also a union of equivalence classes.
Bisimulation Algorithm

If the equivalence relation \( \approx \) over \( S \) is a bisimulation, \( T \) and \( T/ \approx \) are bisimilar.

Notice that equality is a bisimulation.

Are the transition system and its quotient transition systems shown below similar? Why?
Bisimulation Algorithm

In this example, the transition system $T$ and its quotient transition system $T/\approx$ are NOT bisimular. This is because all the conditions are not satisfied.

Now, the question is:
Given a transition system, can we find an equivalence relation (or a partition) such that the transition and its quotient transition system are bisimular?
Bisimulation Algorithm

Algorithm 1 (Bisimulation)

Init: $S / \approx = \{S_S, S_F, S \setminus (S_S \cup S_F)\}$

while $\exists P_1, P_2 \in S / \approx$ and $\sigma \in \Sigma$ such that $P_1 \cap Pre_\sigma(P_2) \neq P_1$ and $P_1 \cap Pre_\sigma(P_2) \neq \emptyset$ do

begin

$R_1 = P_1 \cap Pre_\sigma(P_2)$

$R_2 = P_1 \setminus Pre_\sigma(P_2)$

$S / \approx = (S / \approx \setminus \{P_1\}) \cup \{R_1, R_2\}$

end
Bisimulation Algorithm

Algorithm 1 (Bisimulation)

Init: \( S/ \approx = \{S_S, S_F, S \setminus (S_S \cup S_F)\} \)

while \( \exists P_1, P_2 \in S/ \approx \) and \( \sigma \in \Sigma \) such that \( P_1 \cap Pre_\sigma(P_2) \neq P_1 \) and \( P_1 \cap Pre_\sigma(P_2) \neq \emptyset \) do

begin

\( R_1 = P_1 \cap Pre_\sigma(P_2) \)

\( R_2 = P_1 \setminus Pre_\sigma(P_2) \)

\( S/ \approx = (S/ \approx \ \setminus \ \{P_1\}) \cup \{R_1, R_2\} \)

end

---

Diagram: Transition system with states \( q_0, q_1, q_2, q_3, q_4, q_5, q_6 \) and transitions labeled with symbols \( a, b, c \). States \( q_0 \) is an initial state. States \( q_3, q_4, q_5, q_6 \) are final states. The diagram illustrates the structure of the states and transitions, colored to indicate different sets: \( S_S, S_F, S \setminus (S_S \cup S_F) \).
Algorithm 1 (Bisimulation)

Init: \( S/ \approx = \{S_S, S_F, S \setminus (S_S \cup S_F)\} \)

while \( \exists P_1, P_2 \in S/ \approx \) and \( \sigma \in \Sigma \) such that \( P_1 \cap \text{Pre}_\sigma(P_2) \neq P_1 \) and \( P_1 \cap \text{Pre}_\sigma(P_2) \neq \emptyset \) do

\[
R_1 = P_1 \cap \text{Pre}_\sigma(P_2)
\]

\[
R_2 = P_1 \setminus \text{Pre}_\sigma(P_2)
\]

\[S/ \approx = (S/ \approx \setminus \{P_1\}) \cup \{R_1, R_2\} \]

end
Bisimulation Algorithm

Algorithm 1 (Bisimulation)

Init: $S/ \approx = \{S_S, S_F, S \setminus (S_S \cup S_F)\}$

while $\exists P_1, P_2 \in S/ \approx$ and $\sigma \in \Sigma$ such that $P_1 \cap Pre_{\sigma}(P_2) \neq P_1$ and $P_1 \cap Pre_{\sigma}(P_2) \neq \emptyset$ do

begin

$R_1 = P_1 \cap Pre_{\sigma}(P_2)$

$R_2 = P_1 \setminus Pre_{\sigma}(P_2)$

$S/ \approx = (S/ \approx \setminus \{P_1\}) \cup \{R_1, R_2\}$

end

$Pre_{\sigma}(P_2)$

$SS = P_1$

$S \setminus (S_S \cup S_F)$

$SF = P_2$

$P_1 \cap Pre_{\sigma}(P_2) = \emptyset$
Bisimulation Algorithm

Algorithm 1 (Bisimulation)

Init: \( S/ \approx = \{ S_S, S_F, S \setminus (S_S \cup S_F) \} \)

while \( \exists P_1, P_2 \in S/ \approx \) and \( \sigma \in \Sigma \) such that \( P_1 \cap \text{Pre} \sigma (P_2) \neq P_1 \) and \( P_1 \cap \text{Pre} \sigma (P_2) \neq \emptyset \) do

begin

\( R_1 = P_1 \cap \text{Pre} \sigma (P_2) \)
\( R_2 = P_1 \setminus \text{Pre} \sigma (P_2) \)
\( S/ \approx = (S/ \approx \setminus \{ P_1 \}) \cup \{ R_1, R_2 \} \)

end
Bisimulation Algorithm

Algorithm 1 (Bisimulation)
Init: \( S/ \approx = \{S_S, S_F, S \setminus (S_S \cup S_F)\} \)
\[\text{while } \exists P_1, P_2 \in S/ \approx \text{ and } \sigma \in \Sigma \text{ such that } P_1 \cap \text{Pre}_\sigma(P_2) \neq P_1 \text{ and } P_1 \cap \text{Pre}_\sigma(P_2) \neq \emptyset \text{ do} \]
\[\text{begin} \]
\[R_1 = P_1 \cap \text{Pre}_\sigma(P_2) \]
\[R_2 = P_1 \setminus \text{Pre}_\sigma(P_2) \]
\[S/ \approx = (S/ \approx \setminus \{P_1\}) \cup \{R_1, R_2\} \]
\[\text{end} \]

Algorithm terminated!
A partition defined by \( \approx \) is generated.
Bisimulation Algorithm

Algorithm 1 (Bisimulation)

Init: \( S / \approx \ 
\begin{align*} & = \{ S_S, S_F, S \setminus (S_S \cup S_F) \} \\
& \text{while } \exists P_1, P_2 \in S / \approx \text{ and } \sigma \in \Sigma \text{ such that } P_1 \cap Pre_{\sigma}(P_2) \neq \emptyset \text{ and } P_1 \cap Pre_{\sigma}(P_2) \neq \emptyset \text{ do} \\
& \text{begin} \\
& \quad R_1 = P_1 \cap Pre_{\sigma}(P_2) \\
& \quad R_2 = P_1 \setminus Pre_{\sigma}(P_2) \\
& \quad S / \approx = (S / \approx \setminus \{P_1\}) \cup \{R_1, R_2\} \\
& \text{end} \\
\end{align*}

Given \( T \) and \( \approx \), by construction, we have \( T \setminus \approx \).
Bisimulation Algorithm

If the equivalence relation $\approx$ over $S$ is a bisimulation, $T$ and $T/ \approx$ are bisimular.

Furthermore, the simulation relation over $S \times S/ \approx$ is defined by

$$s_1 \sim S_1 \iff s_1 \in S_1$$

Namely, $\sim$ is the belonging relation $\in$. 
Bisimulation Algorithm

The equivalence relation $\approx$ over $S$ is a bisimulation. e.g. $q_0 \approx q_0$, $q_1 \approx q_1$, $q_1 \approx q_2$, $q_2 \approx q_1$, ...

The simulation relation $\sim$ over $S \times S/ \approx$ is defined by is the belonging relation $\in$. e.g. $q_0 \in S_1$, $q_1 \in S_2$, $q_2 \in S_2$, ...

\[\begin{array}{c}
\begin{array}{c}
q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \\
q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_2 \\
q_2 \xrightarrow{b} q_3 \\
q_3 \xrightarrow{b} q_4 \\
q_4 \xrightarrow{c} q_5 \\
q_5 \xrightarrow{c} q_6 \\
q_6 \xrightarrow{b} q_6
\end{array}
\end{array}\]

\[\begin{array}{c}
\begin{array}{c}
T \\
T/ \approx
\end{array}
\end{array}\]
Analysis: Timed Automata
Bisimulations of Timed Automata
Timed Automata

Timed automata are introduced for encoding timing constraints so that we can model temporal behaviors of resource management systems, Internet traffic, or even audio protocols. Formal methods can be applied for proving correctness of systems with respect to given specifications.

\[ x(t_0) = x_0 \]
\[ \forall t \geq t_0 : x(t) \in G(e_{12}) \]
\[ x := 0 \]
\[ \dot{x} = 1 \]
\[ x \in Dom(q_1) \]
\[ x \in G(e_{21}) \]
\[ q_1 \]

\[ q_2 \]
\[ \dot{x} = 1 \]
\[ x \in Dom(q_2) \]
\[ x := 0 \]
Timed Automata

Timed Automata are a class of hybrid systems that are extended from finite automata by involving simple continuous dynamics which basically have constants in the righ-hand side of the differential equations.

\[ x(t_0) = x_0 \]

\[ x \in G(e_{12}) \]

\[ x := 0 \]

\[ q_1 \]

\[ \dot{x} = 1 \]

\[ x \in Dom(q_1) \]

\[ x \in G(e_{21}) \]

\[ q_2 \]

\[ \dot{x} = 1 \]

\[ x \in Dom(q_2) \]

\[ x := 0 \]

\[ t_0, t_1, t_2, t_3, t_4 \text{ time} \]
Timed Automata

In each hybrid automaton, there are continuous state $x \in X(= \mathbb{R}^n)$ and discrete state $q \in Q$. The evolution of the continuous state are governed by an ordinary differential equation in the form $\dot{x} = [1, \ldots, 1]^T$. The initial condition is specified by a set $Init \subseteq Q \times X$.

\[ x(t_0) = x_0 \]
\[ x \in G(e_{12}) \]
\[ x := 0 \]
\[ q_1 \]
\[ \dot{x} = 1 \]
\[ x \in \text{Dom}(q_1) \]

\[ x \in G(e_{21}) \]
\[ x := 0 \]
\[ q_2 \]
\[ \dot{x} = 1 \]
\[ x \in \text{Dom}(q_2) \]

\[ t_0, t_1, t_2, t_3, t_4 \text{ time} \]
Timed Automata

There are different kinds of clocks:
Global Clock: $t$
Local Clocks: $x$ with $x \in \mathbb{R}$
Notice that each local clock can start at a specific time in the beginning of an execution.

$x(t_0) = x_0$

$q_1$
\[ \dot{x} = 1 \]
$x \in \text{Dom}(q_1)$

$q_2$
\[ \dot{x} = 1 \]
$x \in \text{Dom}(q_2)$

$x \in G(e_{12})$

$x := 0$

$x \in G(e_{21})$

$x := 0$
Timed Automata

Time limit is introduced in each discrete location by using the concept: **Domain**. The Domain is defined by $\text{Dom}(\cdot) : Q \rightarrow 2^{\mathbb{R}^n}$. Basically, a clock can run only if it is within the domain.

\[
x(t_0) = x_0 \\
x \in \text{Dom}(q_1) \quad \frac{\dot{x}}{\dot{x}} = 1 \\
x \in \text{G}(e_{12}) \quad x \leftarrow 0
\]

\[
x \in \text{Dom}(q_2) \quad \frac{\dot{x}}{\dot{x}} = 1 \\
x \in \text{G}(e_{21}) \quad x \leftarrow 0
\]
Timed Automata

Transition between discrete states depends on the **Guard** which is assigned for each edge $e \in E(\subseteq Q \times Q)$. The guard is defined by $G(\cdot) : E \to 2^{\mathbb{R}^n}$. Whenever a guard condition is satisfied, a transition between the discrete states can occur.

$x(t_0) = x_0$

$q_1$

$\dot{x} = 1$

$x \in Dom(q_1)$

$x := 0$

$q_2$

$\dot{x} = 1$

$x \in Dom(q_2)$

$x := 0$

$x \in G(e_{12})$

$x \in G(e_{21})$
Timed Automata

Each local clock can be reset or can remain unchanged after each transition. The **Reset** is defined by $R(\cdot) : E \times \mathbb{R}^n \rightarrow 2^{\mathbb{R}^n}$

$$x(t_0) = x_0 \quad \begin{array}{c}
q_1 \\
\dot{x} = 1 \\
x \in \text{Dom}(q_1) \\
x \in G(e_{12}) \\
x := 0
\end{array} \quad \begin{array}{c}
q_2 \\
\dot{x} = 1 \\
x \in \text{Dom}(q_2) \\
x \in G(e_{21}) \\
x := 0
\end{array}$$
Timed Automata

The (continuous) sets related to Timed Automata have to satisfy certain conditions in order to have the corresponding reachability problem decidable.

The sets involved in the definitions of the initial states, guards, domains and restes have to be rectangular sets.

\[ x(t_0) = x_0 \]

\[ \dot{x} = 1 \quad x \in Dom(q_1) \]

\[ x \in G(e_{12}) \]

\[ q_1 \]

\[ x := 0 \]

\[ q_2 \]

\[ \dot{x} = 1 \quad x \in Dom(q_2) \]

\[ x \in G(e_{21}) \]

\[ x := 0 \]
Timed Automata

Consider $x \in \mathbb{R}^n$. A subset of $\mathbb{R}^n$ set is called rectangular if it can be written as a finite boolean combination of constraints of the form $x_i \leq a$, $x_i < b$, $x_i = c$, $x_i \geq d$, and $x_i > e$, where $a, b, c, d, e$ are rational numbers.

e.g. in $\mathbb{R}^2$ the set $(x_1 \geq 0) \land (x_1 \leq 2) \land (x_2 \geq 1) \land (x_2 \leq 2)$ is rectangular

e.g. in $\mathbb{R}^2$ the set $((x_1 \geq 0) \land (x_2 = 0)) \lor ((x_1 = 0) \land (x_2 \geq 0))$ is rectangular

e.g. in $\mathbb{R}^2$ the empty set $(x_1 \geq 0) \land (x_1 \leq 0)$ is rectangular

e.g. in $\mathbb{R}^2$ the set $\{x \in \mathbb{R}^2 | x_1 = 2x_2\}$ is NOT rectangular
Timed Automata

Example An example of a timed automaton is given.

\[
x_1 := 0 \\
x_2 := 0 \\
x_1 \leq 3 \land x_2 \leq 2
\]

We have:

- \( Q = \{q_1, q_2\} \);
- \( X = \mathbb{R}^n \);
- \( f(q_1, x) = f(q_2, x) = [1 \ 1]^T \);
- \( Init = \{(q_1, (0, 0))\} \);
- \( Dom(q_1) = Dom(q_2) = \mathbb{R}^n_{\geq 0} \);
- \( E = \{(q_1, q_2), (q_2, q_1)\} \);
- \( G(q_1, q_2) = \{ x \in \mathbb{R}^2 | x_1 \leq 3 \land x_2 \leq 2 \}, G(q_2, q_1) = \{ x \in \mathbb{R}^2 | x_1 \leq 1 \} \);
- \( R(q_1, q_2, x) = \{(0, x_2)\}, R(q_2, q_1, x) = \{ x \} \).
Hybrid Automaton

An execution of a hybrid automaton is a hybrid trajectory, \((\tau, q, x)\), of its state trajectory.

**Definition 3 Execution** An execution of a hybrid automaton \(H\) is a hybrid trajectory, \((\tau, q, x)\), which satisfies the following conditions:

- **Initial condition:** \((q_0(0), x_0(0)) \in Init\)
- **Discrete evolution:** for all \(i\),
  \((q_i(\tau_i^l), q_{i+1}(\tau_{i+1})) \in E, x_i(\tau_i^l) \in G(q_i(\tau_i^l), q_{i+1}(\tau_{i+1})),\)
  and \(x_{i+1}(\tau_{i+1}) \in R(q_i(\tau_i^l), q_{i+1}(\tau_{i+1}), x_i(\tau_i^l))\)
- **Continuous evolution:** for all \(i\),
  1. \(q_i(\cdot) : I_i \rightarrow Q\) is constant over \(t \in I_i\), i.e. \(q_i(t) = q_i(\tau_i)\) for all \(t \in I_i\);
  2. \(x_i(\cdot) : I_i \rightarrow X\) is the solution to the differential equation \(\dot{x}_i(t) = f(q_i(t), x_i(t))\) over \(I_i\) starting at \(x_i(\tau_i)\); and,
  3. for all \(t \in [\tau_i, \tau_i^l]\), \(x_i(t) \in D(q_i(t))\).

Notice that a hybrid automaton can accept multiple executions for some initial states.
Definition 4 Classification of execution An execution \((\tau, q, x)\) is called:

- **Finite**: if \(\tau\) is a finite sequence and the last interval in \(\tau\) is closed.
- **Infinite**: if \(\tau\) is an infinite sequence, or if the sum of the time intervals in \(\tau\) is infinite, i.e. \(\sum_{i=0}^{N}(\tau_i' - \tau_i) = \infty\).
- **Zeno**: if it is infinite but \(\sum_{i=0}^{\infty}(\tau_i' - \tau_i) < \infty\).
- **Maximal**: if it is not a strict prefix of any other execution of \(H\).
Hybrid Automaton

**Definition 4 Classification of execution** An execution \((\tau, q, x)\) is called:

- **Finite**: if \(\tau\) is a finite sequence and the last interval in \(\tau\) is closed.
- **Infinite**: if \(\tau\) is an infinite sequence, or if the sum of the time intervals in \(\tau\) is infinite, i.e. \(\sum_{i=0}^{N}(\tau'_i - \tau_i) \equiv \infty\).
- **Zeno**: if it is infinite but \(\sum_{i=0}^{\infty}(\tau'_i - \tau_i) < \infty\).
- **Maximal**: if it is not a strict prefix of any other execution of \(H\).
Hybrid Automaton

Non-Determinism

- Multiple Executions for the same initial condition
- Sources of non-determinism
  - Non-Lipschitz continuous vectorfields, \( f \)
  - Multiple discrete transition destinations, \( E \) & \( G \)
  - Choice between discrete transition and continuous evolution, \( D \) & \( G \)
  - Non-unique continuous state assignment, \( R \)

**Definition:** A hybrid automaton \( H \) is deterministic if for all initial conditions there exists a unique maximal sequence.
Hybrid Automaton

Blocking

- No Infinite executions for some initial states
- Source of blocking
  - Cannot continue in domain due to reaching the boundary of the domain where no guard is defined
  - Have no place to make discrete transition to

Definition: A hybrid automaton H is non-blocking if for every initial condition there exists at least one infinite execution
Hybrid Automaton

- Zeno Executions
  - Infinite execution defined over finite time
    - Infinite number of transitions in finite time
    - Transition times converge

Definition: A hybrid automaton $H$ is zeno if there exists an initial condition for which all infinite executions are Zeno.
Transition System

Every hybrid automaton, $H$, generates a transition system $T = (S, \Sigma, \rightarrow, S_S, S_F)$ by setting

- $S = Q \times X$;
- $\Sigma = E \cup \{\tau\}$;
- $\rightarrow = (\bigcup_{e \in E} \xrightarrow{e}) \cup (\bigcup_{\tau \geq 0} \xrightarrow{\tau})$;
- $S_S = Init$;
- $S_F \subseteq Q \times X$;

where there are discrete transitions, $(q_1, x_1) \xrightarrow{e} (q_2, x_2)$, and continuous transitions, $(q_1, x_1) \xrightarrow{\tau} (q_2, x_2)$

For time automata, we are interested in **time-abstracted** continuous transitions in which time is being abstracted.
Transition System

The discrete transition $\xrightarrow{e}$ is defined as follow:

$$(q_1, x_1) \xrightarrow{e} (q_2, x_2) \text{ iff } e = (q_1, q_2) \in E \text{ and } x_2 \in R(e, x_1)$$

The time-abstracted continuous transition $\xrightarrow{\tau}$ is defined as follow:

$$(q_1, x_1) \xrightarrow{\tau} (q_2, x_2) \text{ iff } q_1 = q_2, \exists \tau \geq 0, \text{ and there exists a curve } \phi : [0, \tau] \times \mathbb{R}^n \rightarrow \mathbb{R}^n \text{ with } \phi(0, x_1) = x_1, \phi(\tau, x_1) = x_2$$

and for all $t \in [0, \tau]$ it satisfies $\frac{d}{dt}\phi(t, x_1) = f(q_1, \phi(t, x_1))$ and $\phi(t, x_1) \in D(q_1)$.

How much time it takes is not specified in time-abstracted continuous transition as long as there exists such a time.
Region Equivalence Relation

For $x_i \in \mathbb{R}$, define the integer part of $x_i$, $\lfloor x_i \rfloor \in \mathbb{Z}$, to be the largest integer $\leq x_i$ and define the fractional part of $x_i$, $\langle x_i \rangle \in [0, 1)$, such that $x_i = \lfloor x_i \rfloor + \langle x_i \rangle$.

For a given timed automaton, let $c_i$ be the largest constant with which $x_i$ is compared among all discrete states.

Consider the relation $\simeq$ over $S(= Q \times X)$, i.e. $(q, x) \simeq (q', x')$, if the following conditions are satisfied:
1. $q = q'$;
2. for all $x_i$, $\lfloor x_i \rfloor = \lfloor x'_i \rfloor$ or $(x_i > c_i) \land (x'_i > c_i)$;
3. for all $x_i, x_j$, with $x_i \leq c_i$ and $x_j \leq c_j$,
\[
(\langle x_i \rangle \leq \langle x_j \rangle) \Leftrightarrow (\langle x'_i \rangle \leq \langle x'_j \rangle);
\]
4. for all $x_i$ with $x_i \leq c_i$,
\[
(\langle x_i \rangle = 0) \Leftrightarrow (\langle x'_i \rangle = 0).
\]
Region Equivalence Relation

Consider the relation $\preceq$ over $S(= Q \times X)$, i.e. $(q, x) \preceq (q', x')$, if the following conditions are satisfied:

1. $q = q'$;
2. for all $x_i$, $[x_i] = [x_i']$ or $(x_i > c_i) \land (x_i' > c_i)$;
3. for all $x_i, x_j$, with $x_i \leq c_i$ and $x_j \leq c_j$,

   $$(\langle x_i \rangle \leq \langle x_j \rangle) \iff (\langle x_i' \rangle \leq \langle x_j' \rangle);$$

4. for all $x_i$ with $x_i \leq c_i$,

   $$(\langle x_i \rangle = 0) \iff (\langle x_i' \rangle = 0).$$

Condition 3 doesn’t apply in one dimensional case
Timed Automata

**Proposition** $\cong$ is an equivalence relation.

How to prove that??
Proposition \( \cong \) is an equivalence relation.

How to prove that??

Definition 5 Let \( \pi \) be a collection of nonempty subsets of \( S \). \( \pi \) is called a partition of \( S \) if
1. \( \forall S_i, S_j \in \pi, S_i \neq S_j \Rightarrow S_i \cap S_j = \emptyset \), and
2. \( \bigcup_{S_i \in \pi} S_i = S \).

Theorem 1 If \( R \) is an equivalence relation on a set \( S \), then the equivalence classes of \( R \) constitutes a partition of \( S \).

On the other hand, given a partition of \( S \), one can define an induced equivalence relation over \( S \).
Timed Automata

Consider the previous example of a timed automaton.

\[ x_1 := 0 \]
\[ x_2 := 0 \]
\[ x_1 \leq 3 \land x_2 \leq 2 \]

\[ \dot{x}_1 = 1 \]
\[ \dot{x}_2 = 1 \]
\[ x \in \mathbb{R}^2_{\geq 0} \]

\[ q_1 \]
\[ x := x \]
\[ x_1 \leq 1 \]

\[ \dot{x}_1 = 1 \]
\[ \dot{x}_2 = 1 \]
\[ x \in \mathbb{R}^2_{\geq 0} \]

\[ q_2 \]

In this example, what are the largest constants? What do the equivalence classes look like?
Region Equivalence Relation

In the example of a timed automaton, the largest constants are \( c_1 = 3 \) and \( c_2 = 2 \).
Region Equivalence Relation

Recall that the continuous sets involved in the definitions of initial states, guards, domains and rests are rectangular.

Hence, any rectangular sets can be represented by a union of equivalence classes.

For both $q_1, q_2$

Equivalence classes

- points
- open lines
- open sets
Region Equivalence Relation

**Proposition** $\cong$ is a bisimulation.

How to prove that??

For both $q_1, q_2$

Equivalence classes

- points
- open lines
- open sets
Region Equivalence Relation

**Proposition** \( \cong \) is a bisimulation.

How to prove that??

Recall that

**Definition 1 Bisimulation**

*Given transition systems \( T = (S, \Sigma, \rightarrow, S_S, S_F) \) and equivalence relation, \( \approx \), over \( S \), \( \approx \) ia called a bisimulation if:

1) \( S_S \) is a union of equivalence classes;
2) \( S_F \) is a union of equivalence classes;
3) For all \( \sigma \in \Sigma \), if \( P \) is a union of equivalence classes, \( Pre_\sigma(P) \) is also a union of equivalence classes.*
Region Equivalence Relation

**Proposition** \( \cong \) is a bisimulation.

Prove:

For the conditions
1) \( S_S \) is a union of equivalence classes and
2) \( S_F \) is a union of equivalence classes, since, by the definition of timed automata and by construction, \( S_S \) and \( S_F \) are rectangular sets, the conditions are satisfied.

For the condition
3) For all \( \sigma \in \Sigma \), if \( P \) is a union of equivalence classes, \( Pre_\sigma(P) \) is also a union of equivalence classes, we have to check both predecessors, \( Pred(P) \) and \( Pre_c(P) \).
Region Equivalence Relation

**Proposition** $\cong$ is a bisimulation.

Prove: (con’t)

Consider $\text{Pre}_d(P)$ with $e = (q, q')$ and $P$ being a union of equivalence classes. Let the inverse of the reset be

$$R^{-1}(q, q', P) = \{(q, x) \in Q \times X | \exists (q', x') \in P \text{ with } x' \in R(q, q', x)\}$$

Notice that

$$\text{Pre}_d(P) = R^{-1}(q, q', P) \cap (\{q\} \times G(q, q'))$$

It can be shown that $R^{-1}(q, q', P)$ is a union of equivalence classes. Hence, $\text{Pre}_d(P)$ is a union of equivalence classes.
**Proposition** $\cong$ is a bisimulation.

Prove: (con’t) \[ R(q_1, q_2, P) : \begin{align*}
    x_1 &\coloneqq 0 \\
    x_2 &\coloneqq x_2
\end{align*} \]
Proposition \( \cong \) is a bisimulation.

Prove: (con't)\[R(q_1, q_2, P) : x_1 \coloneqq 0 \]
\[x_2 \coloneqq x_2\]
\[G(q_1, q_2) : x_1 \leq 3 \land x_2 \leq 2\]
Region Equivalence Relation

Proposition $\cong$ is a bisimulation.

Prove: (con’t)

Consider $\text{Pre}_r(P)$ and $P$ being a union of equivalence classes. Notice that

$$\text{Pre}_r(P) = \{(q, x) \in Q \times X | \exists (q', x') \in P \exists t \geq 0 \text{ with } \dot{x} = x + t[1, \ldots, 1]^T\}$$

These are the points that if we move in the $[1, \ldots, 1]^T$ directions we will eventually reach $P$. If $P$ is a union of equivalence classes, this set is also a union of equivalence classes.
Region Equivalence Relation

**Proposition** $\cong$ is a bisimulation.

Prove: (con’t)
Region Equivalence Relation

In the example, how many equivalence classes are there?

Ans: \(2(12 \text{ points } + 30 \text{ open lines } + 18 \text{ open sets}) = 120\)

For both \(q_1, q_2\)

Equivalence classes

- Points
- Open lines
- Open sets
Region Equivalence Relation

In general, for an arbitrary systems there are

\[ m(n!)(2^n) \prod_{i=1}^{n} (2c_i + 2) \]

discrete states. The complexity of solving the reachability problem is linear in the number of discrete location \( m \), exponential in the number of clocks \( n \), and exponential in the encoding constants.

The reachability problem for timed automata can be answered on a finite quotient transition system and the reachability problem is decidable since the quotient space is finite. In computational complexity terminology, model checking for timed automata turns out to be \textit{PSPACE Complete}. 
Region Equivalence Relation

However, simple variants of timed automata do not have finite bisimulations. This is the case for example if \( \dot{x}_i = c_i \) for some constant \( c_i \neq 1 \) (skewed clocks, leading to multi-rate automata), or allow comparisons between clocks (terms of the form \( x_i \leq x_j \)), or resting one clock to another (\( x_i := x_j \)). The reachability question for such systems is \textit{undecidable}. 
End