EECS 291E: Lecture Notes 5. Sequence Properties and Verification.

Claire J. Tomlin

February 1, 2018

One of our goals in this course is to develop tools for analyzing whether or not a hybrid system satisfies certain desirable properties. A second goal is to design controllers for hybrid systems, such that the closed loop hybrid system satisfies such properties. Thus, we would like to be able to answer the questions of:

- **Verification:** Does the hybrid automaton meet the specification?
- **Synthesis:** Can a controller be designed such that the closed loop hybrid system satisfies the specification?

In this chapter, we will discuss the **safety** property; in the next chapter, we will discuss the **stability** property.

1 Reachability and Sequence Properties

- The problem we will address is, given a hybrid automaton H compute Reach(H).
- If we can solve this problem we can also answer questions about safety properties. A safety property can generally be posed as:
  Does the state (q, x) always remain in a set of states $G \subseteq Q \times X$?
  The set G can be used to encode "good" or "safe" states. For example, G could be the set of states for which two aircraft always maintain the minimum required separation distance. Using notation from temporal logic, the above safety property can be written as
  $$\Box((q, x) \in G)$$

Here, $\Box$ stands for "always", and the way to read the above formula is "the property that (q, x) is in G is always satisfied". We will also use the notation $(Q \cup X, \Box G)$ to indicate that for the hybrid automaton H with states $Q \cup X$, it is always the case that the state stays in G.

- The dual property of safety is known as **liveness**. A liveness property can be posed as:
  Does the state (q, x) eventually reach a set of states $G \subseteq Q \times X$?,
  This reflects the fact that something good should eventually happen to our system. In temporal logic notation this property can be written as

$$\Diamond((q, x) \in G)$$

  Here, $\Diamond$ stands for "eventually", and the way to read the above formula is "the property that (q, x) eventually reaches G is satisfied".

- Using concepts of "always" and "eventually", one can encode arbitrarily complex properties, such as:

$$\Box\Diamond((q, x) \in G)$$

  which means that the state visits the set G "infinitely often", and

$$\Diamond\Box((q, x) \in G)$$

  which means that the state reaches G at some point and stays there for ever after.

- In this course, we will focus on safety properties, and not liveness properties. This is because, for most of the practical systems we deal with, we are either interested in safety properties directly, or in the property that the state reaches a given set in a finite amount of time. *How would you pose this second property as a safety property?*

Proposition 1 *H satisfies $(Q \cup X, \Box G)$ for $G \subseteq Q \times X$ if and only if* $\mathrm{Reach}(H) \subseteq G$.

- Different methods have been proposed for solving the reachability problem:
  1. **Optimal Control**: The role of the "control" is often played by the non-determinism of the system.
  2. **Deductive Techniques**: Establish invariants to bound $\mathrm{Reach}(H)$.
  3. **Model Checking Techniques**: Automatically compute $\mathrm{Reach}(H)$. Requires one to be able to "compute" with sets of states. Class of systems to which it applies is inherently limited.
  4. **Approximation**: Works with all of the above
     - For optimal control, approximate by sets and dynamics for which optimal control problems are easier to solve (e.g. ellipsoidal sets and linear dynamics)
     - Deductive techniques are inherently based on over-approximation

  - For model checking, approximate by a system you can compute with.
  - Also possible to do "brute force" over-approximation, based, for example, on gridding.

- In all cases you stop once your question has been answered. For example, if your question was "does H satisfy $(Q \cup X, \Box G)$", you could stop if you found a reachable state outside G.

- For approximation, you typically "over-approximate".

- All methods are supported by computational tools:

  1. typically uses optimal control and convex optimization tools
  2. typically uses theorem provers
  3. typically uses model checkers
  4. done using "gridding" or polynomial manipulation packages.

- **Simulation** can also be used for reachability investigations. It is not a formal method however since:

  - It is a shot in the dark: we simulate. If G^C, the complement of G, is reached, fine, else we have to choose another initial condition and try again.
  - No termination guarantee.

  Still it is the best we can do for many classes of hybrid systems.

2 General Transition Systems

Definition 2 (Transition System) *A transition system is a collection* $T = (S, \Sigma, \rightarrow, S_0, S_F)$*, where*

- S *is a set of states;*
- Σ *is an alphabet of events;*
- $\rightarrow$: $S \times \Sigma \rightarrow 2^S$ *is a transition relation;*
- $S_0 \subseteq S$ *is a set of initial states; and,*
- $S_F \subseteq S$ *is a set of final states.*

Example: A finite automaton $M = (Q, \Sigma, \text{Init}, R)$, with final states F specified, is a transition system with:

- $S = Q$

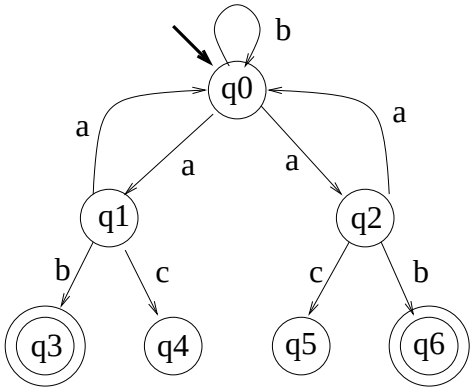

Figure 1: Reachability example for finite automata

- Σ the same
- $\rightarrow = R$
- $S_0 = \text{Init}$
- $S_F = F$

Example: An autonomous hybrid automaton $H = (Q,\ X,\ \text{Init},\ f,\ \text{Dom},\ R)$ and a safety property $(Q \cup X, \Box G)$ form a transition system with:

- $S = Q \times X$
- Σ not specified
- $\rightarrow = \{$ discrete transitions $\} \cup \{$ continuous evolution$\}$, all of which are characterized by f, Dom, and R
- $S_0 = \text{Init}$
- $S_F = G^C$

Problem 1 (Reachability) *Given a transition system T, is any state $s_f \in S_F$ reachable from a state $s_0 \in S_0$ by a sequence of transitions?*

Remark: For finite automata we can always "decide" reachability problems by brute force.

Example: Consider for example the finite automaton of Figure 1. We can start "exploring" from q_0 and keep going until we either visit a state in F or have nowhere else to go (have visited all reachable states).

More formally, we can approach this problem using the **predecessor operator**:

$$\text{Pre} : 2^S \rightarrow 2^S$$

which is defined as follows (for a set $S' \subseteq S$):

$$\text{Pre}(S') = \{s \in S : \exists s' \in S' \exists \sigma \in \Sigma \text{ such that } s \rightarrow_\sigma s'\}$$

In the above, $\rightarrow_\sigma$ indicates that $(s, \sigma) \rightarrow s'$. Thus, the **predecessor** of a set of states S' is the set of states that can reach S' in one transition. The set of reachable states for a transition system can be computed using the following algorithm:

Algorithm 1 (Reachability)

Initialization:
    $W_0 = S_F$, $i = 0$
repeat
    if $W_i \cap S_0 \neq \emptyset$
        return "S_F reachable"
    end if
    $W_{i+1} = \text{Pre}(W_i) \cup W_i$
    $i = i + 1$
until $W_i = W_{i-1}$
return "S_F not reachable"

- For the example of Figure 1, if event a happens at iteration 1, and events b, c happen at iteration 2, then $W_0 = \{q_0\}$, $W_1 = \{q_0, q_1, q_2\}$ and $W_2 = Q$.
- For finite automata the algorithm *can be implemented* and *always terminates*.
- What properties of finite automata allow us to do this?
    1. We can represent states in a finite way (by enumeration).
    2. We can represent transitions among states in a finite way (by enumeration)
    3. We are guaranteed that if we start exploring a particular execution, after a finite number of steps we will either reach a state we visited already, or a state from which we have nowhere else to go, or a final state.
- Enumeration is a fairly naive way of going about reachability analysis. In practice, sets of states are not enumerated, but are represented more compactly, using, for example, Binary Decision Diagrams, or BDDs.
- For general transition systems, this algorithm is conceptually useful, but not directly implementable on a computer. To effectively implement this algorithm, on has to devise effective ways to:
    - store sets of states
    - compute the Pre of a set of states
    - take the union and intersection of sets of states
    - check whether a set of states is empty

– check whether two sets of states are equal

If the number of states is finite, one can do all of these relatively easy using enumeration; for non-finite state spaces, these are all difficult problems, which will be addressed in theis chapter and later chapters. One way of addressing this is through **bisimulation**.

3 Bisimulation

- In the example of Figure 1, q_1 and q_2 have very similar properties, since they can both be reached from q_0 by a and all subsequent executions look similar.
- Suggests that these states are in some sense "equivalent".
- Try to make this statement more precise by introducing the notion of bisimulation.
- Consider the set of states S. A relation on S is a subset of $S \times S$.

Definition 3 (Equivalence Relation) *A relation $\sim \subseteq S \times S$ is called an equivalence relation if it is:*

1. *Reflexive: $(s, s) \in \sim$ for all $s \in S$;*
2. *Symmetric: $(s, s') \in \sim$ implies that $(s', s) \in \sim$; and,*
3. *Transitive: $(s, s') \in \sim$ and $(s', s'') \in \sim$ imply $(s, s'') \in \sim$.*

- For simplicity we write $s \sim s'$ instead of $(s, s') \in \sim$ and say s is equivalent to s'.
- An example of an equivalence relation: Equality.
- An equivalence relation partitions S to a number of *equivalence classes*:

$$S = \bigcup_i S_i$$

such that for all $s, s' \in S$, $s, s' \in S_i$ if and only if $s \sim s'$.

- Equivalence classes cover S (by symmetry)
- Equivalence classes are disjoint (by transitivity)
- For equality the equivalence classes are the singletons, for $S \times S$ there is only one equivalence class, S itself.
- Given an equivalence relation $\sim$, let $S/\sim = \{S_i\}$ denote the quotient space, i.e. the set consisting of all equivalence classes.

- Given a set $P \subseteq S$, let $P/\sim$ represent the part of the quotient space with which P overlaps:
$$P/\sim = \{S_i \ : \ S_i \cap P \neq \emptyset\} \subseteq S/\sim$$

- If S are the states of a transition system, $T = (S, \Sigma, \rightarrow, S_0, S_F)$, define the *quotient transition system* as
$$T/\sim = (S/\sim, \Sigma, \rightarrow_\sim, S_0/\sim, S_F/\sim)$$
where for $S_1, S_2 \in S/\sim$, $S_1 \times \sigma \rightarrow_\sim S_2$ if and only if there exist $s_1 \in S_1$ and $s_2 \in S_2$ such that $(s_1 \times \sigma \rightarrow s_2)$.

- Notice that the quotient transition system may be "non-deterministic", even if the original system is.

- Finally, we denote the predecessor under an event σ as Pre_σ:
$$\mathrm{Pre}_\sigma(P) = \{s \in S : \exists s' \in P \text{ such that } (s, \sigma) \rightarrow s'\}$$

Definition 4 (Bisimulation) *Given $T = (S, \Sigma, \rightarrow, S_0, S_F)$, and $\sim$ an equivalence relation over S, $\sim$ is called a bisimulation if:*

1. *S_0 is a union of equivalence classes;*
2. *S_F is a union of equivalence classes;*
3. *if one state (say s) in one equivalence class (say S_i) can transition to another equivalence class (say S_j), then all other states, $s' \in S_i$ must be able to transition to some state in S_j. More formally, for all i, j and for all states $s, s' \in S_i$, if $s \rightarrow S_j$, then $s' \rightarrow S_j$.*

Note that the third condition above is equivalent to "for all $\sigma \in \Sigma$, if P is a union of equivalence classes $\mathrm{Pre}_\sigma(P)$ is also a union of equivalence classes".

- If $\sim$ is a bisimulation, T and $T/\sim$ are called bisimilar.
- Equality is a bisimulation.
- In a sense bisimilar transition systems generate the same sequences of transitions (language).
- Therefore, if $\sim$ is a bisimulation, we need not distinguish between the elements of an equivalence class.
- More specifically, if a state in S_F is reachable from a state in S_0, a state in $S_F/\sim$ is reachable from a state in $S_0/\sim$.

Proposition 5 *$\sim$ is a bisimulation if and only if:*

1. $(s_1 \sim s_2) \wedge (s_1 \in S_0) \Rightarrow (s_2 \in S_0)$;

2. $(s_1 \sim s_2) \wedge (s_1 \in S_F) \Rightarrow (s_2 \in S_F)$; *and,*

3. $(s_1 \sim s_2) \wedge ((s_1, \sigma) \rightarrow s_1') \Rightarrow \exists s_2'$ *such that* $(s_1' \sim s_2') \wedge ((s_2, \sigma) \rightarrow s_2')$.

Proof: ($\Rightarrow$):

1. If for all $s_1 \in S_0$, $s_1 \sim s_2$ implies $s_2 \in S_0$, S_0 must be a union of equivalence classes.

2. Similarly for S_F.

3. If P is an equivalence class and $s_1 \in \mathrm{Pre}_\sigma(P)$, then $s_2 \in \mathrm{Pre}_\sigma(P)$ for all $s_2 \sim s_1$. Hence $\mathrm{Pre}_\sigma(P)$ must be a union of equivalence classes.

($\Leftarrow$): similar ∎

Aside: More generally, two transition systems, $T = (S, \Sigma, \rightarrow, S_0, S_F)$ and $T' = (S', \Sigma, \rightarrow', S_0', S_F')$ are called *bisimilar* if there exists a relation $\sim \subseteq S \times S'$ such that:

1. $(s_1 \sim s_2) \wedge (s_1 \in S_0) \Rightarrow (s_2 \in S_0')$;

2. $(s_1 \sim s_2) \wedge (s_2 \in S_0') \Rightarrow (s_1 \in S_0)$;

3. $(s_1 \sim s_2) \wedge (s_1 \in S_F) \Rightarrow (s_2 \in S_F')$;

4. $(s_1 \sim s_2) \wedge (s_2 \in S_F') \Rightarrow (s_1 \in S_F)$;

5. $(s_1 \sim s_2) \wedge ((s_1, \sigma) \rightarrow s_1') \Rightarrow \exists s_2'$ such that $(s_1' \sim s_2') \wedge ((s_2, \sigma) \rightarrow' s_2')$.

4 Computing Bisimulations

- Bisimulations look useful since they preserve the language of the transition system.
- How does one find a bisimulation?

Algorithm 2 (Bisimulation)

Initialization:
$S/\sim = \{S_0, S_F, S \setminus (S_0 \cup S_F)\}$
while $\exists P, P' \in S/\sim$ and $\sigma \in \Sigma$ such that $P \cap \mathrm{Pre}_\sigma(P') \neq P$ and $P \cap \mathrm{Pre}_\sigma(P') \neq \emptyset$ **do**
begin
$P_1 = P \cap \mathrm{Pre}_\sigma(P')$
$P_2 = P \setminus \mathrm{Pre}_\sigma(P')$
$S/\sim = (S/\sim \setminus \{P\}) \cup \{P_1, P_2\}$
end

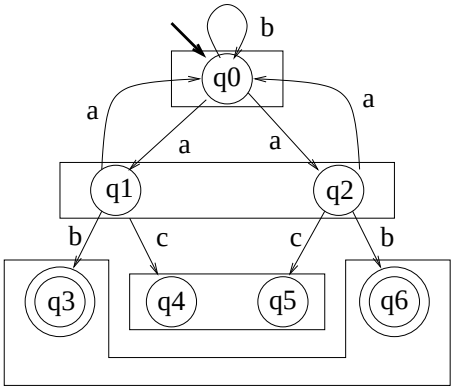

Figure 2: Bisimulation of previous example

- If the algorithm terminates, $\sim$ is a bisimulation, since:
  1. S_0 is a union of equivalence classes (note that $S_0 \in S/\sim$ initially, and we only split sets)
  2. S_F is a union of equivalence classes (for the same reason).
  3. Termination implies that for all $P' \in S/\sim$ and for all σ, $P \cap \mathrm{Pre}_\sigma(P')$ is either equal to P or equal to $\emptyset$, therefore, $\mathrm{Pre}_\sigma(P')$ is a union of equivalence classes.
- Again, implementation and termination of this algorithm for general transition systems are not obvious. For finite state systems we can implement the algorithm and guarantee that it terminates because we can enumerate the states for the finite state system.
- Figure 2 shows the results of applying this algorithm to our previous finite state example.
- Why is this an improvement?
  1. No need to enumerate all the states, therefore may have a computational advantage.
  2. Extends to systems with infinite states. If the bisimulation quotient can be computed and is finite, then the reachability computation is decidable.

5 Bisimulations of Timed Automata

- Consider $X = \{x_1, \ldots, x_n\}$ a finite collection of variables, each of which takes values in $\mathbb{R}$. and let $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$

Definition 6 (Clock Constraints) *The set, $\Phi(X)$, of clock constraints of X, is a set of logical expressions defined inductively by $\delta \in \Phi(X)$ if:*

$$\delta := (x_i \leq c) | (x_i \geq c) | \neg\delta | \delta_1 \wedge \delta_2$$

where $x_i \in X$ and $c \geq 0$ is a rational number.

- Examples: let $X = \{x_1, x_2\}$.
  - $(x_1 \leq 1) \in \Phi(X)$
  - $(0 \leq x_1 \leq 1) \in \Phi(X)$, since $(0 \leq x_1 \leq 1) \Leftrightarrow (x_1 \geq 0) \wedge (x_1 \leq 1)$
  - $(x_1 = 1) \in \Phi(X)$, since $(x_1 = 1) \Leftrightarrow (x_1 \geq 1) \wedge (x_1 \leq 1)$
  - $(x_1 < 1) \in \Phi(X)$, since $(x_1 < 1) \Leftrightarrow (x_1 \leq 1) \wedge \neg(x_1 \geq 1)$
  - $\text{True} \in \Phi(X)$, since $\text{True} \Leftrightarrow \neg((x_1 = 1) \wedge \neg(x_1 = 1))$
  - $(x_1 \leq x_2) \notin \Phi(X)$
- Given $\delta \in \Phi(X)$, we say $x \in \mathbf{X}$ *satisfies* δ if $\delta(x) = \text{True}$.
- To each $\delta \in \Phi(X)$ we can associate a set:

$$\hat{\delta} = \{x \in \mathbf{X} \ : \ \delta(x) = \text{True}\}$$

- The original definition of a timed automaton, found in [1] is given below.

Definition 7 (Timed Automaton) *A timed automaton is a hybrid automaton $H = (Q, X, \text{Init}, f, \text{Dom}, R)$, where*

- *Q is a set of discrete variables, $Q = \{q_1, \ldots, q_m\}$;*
- *$X = \{x_1, \ldots, x_n\}$, $X = \mathbb{R}^n$;*
- *$\text{Init} = \{\{q_i\} \times \widehat{\text{Init}_{q_i}}\}_{i=1}^m$ where $\text{Init}_{q_i} \in \Phi(X)$;*
- *$f(q, x) = (1, \ldots, 1)$ for all (q, x);*
- *$\text{Dom}(q) = X$ for all $q \in Q$;*
- *$R : Q \times \Phi(X) \to Q \times X$ where $R(q, x)$ either leaves x_i unaffected or resets it to 0 (notice that R is single valued).*

Example: As an example consider the timed automaton of Figure 3.

- $Q = \{q_1, q_2\}$;
- $X = \{x_1, x_2\}$, $\mathbf{X} = \mathbb{R}^2$;
- $\text{Init} = \{(q_1, 0, 0)\}$;
- $f(q, x) = (1, 1)$ for all (q, x);
- $\text{Dom}(q) = \mathbb{R}^2$ for all $q \in Q$;

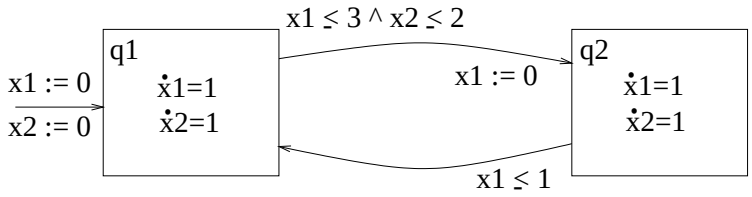

Figure 3: Example of a timed automaton

- $E = \{(q_1, q_2), (q_2, q_1)\}$;
- $G(q_1, q_2) = \{x \in \mathbb{R}^2 \ : \ (x_1 \leq 3) \wedge (x_2 \geq 2)\}$, $G(q_2, q_1) = \{x \in \mathbb{R}^2 \ : \ (x_1 \leq 1)\}$;
- $R(q_1, q_2, x) = \{(0, x_2)\}$, $R(q_2, q_1, x) = \{(x_1, 0)\}$

where the graphical notation for hybrid automata has been used (E edge, G guard).

6 Timed Automata are Bisimilar to Finite Systems

- Without loss of generality, all constants can be assumed to be integers.
- Let T be the transition system defined by a timed automaton, H.
- Consider an arbitrary $\lambda > 0$, rational.
- Let H_λ denote the timed automaton obtained by replacing all constants, c, in H by λc.
- Let T_λ denote the transition system associated with H_λ.

Proposition 8 *T and T_λ are bisimilar.*

Proof: Consider the relation $(q, x) \sim (q, \lambda x)$. Note that, since $\lambda > 0$, $(x_i \leq c) \Leftrightarrow (\lambda x \leq \lambda c)$ and $(x_i \geq c) \Leftrightarrow (\lambda x \geq \lambda c)$. Therefore:

$$
\begin{aligned}
(q, x) \in \text{Init} &\Leftrightarrow (q, \lambda x) \in \text{Init}_\lambda && (1)\\
(q, x) \in F &\Leftrightarrow (q, \lambda x) \in F_\lambda && (2)\\
(q, x) \xrightarrow{e} (q', x') &\Leftrightarrow (q, \lambda x) \xrightarrow{e} (q', \lambda x') && (3)\\
(q, x) \xrightarrow{\tau} (q', x') &\Leftrightarrow (q, \lambda x) \xrightarrow{\tau} (q', \lambda x') && (4)
\end{aligned}
$$

For the discrete transition, $(q, x) \xrightarrow{e} (q', x')$ if $e = (q, q') \in E$, $Ge(x) = \text{True}$ and $x' \in R(e, x)$. Therefore, $(q, \lambda x) \xrightarrow{e} (q', \lambda x')$, since $e = (q, q') \in E$, $G_{e_\lambda}(\lambda x) = \text{True}$ and $\lambda x' \in R_\lambda(e, \lambda x)$. For the continuous transition, recall that $(q, x) \xrightarrow{\tau} (q', x')$ if $q = q'$, and there exists $t \geq 0$ such that $x' = x + t(1, \ldots, 1)$. Therefore, $(q, \lambda x) \xrightarrow{\tau} (q', \lambda x')$ since $q = q'$ and $\lambda x' = \lambda x + \lambda t(1, \ldots, 1)$. ∎

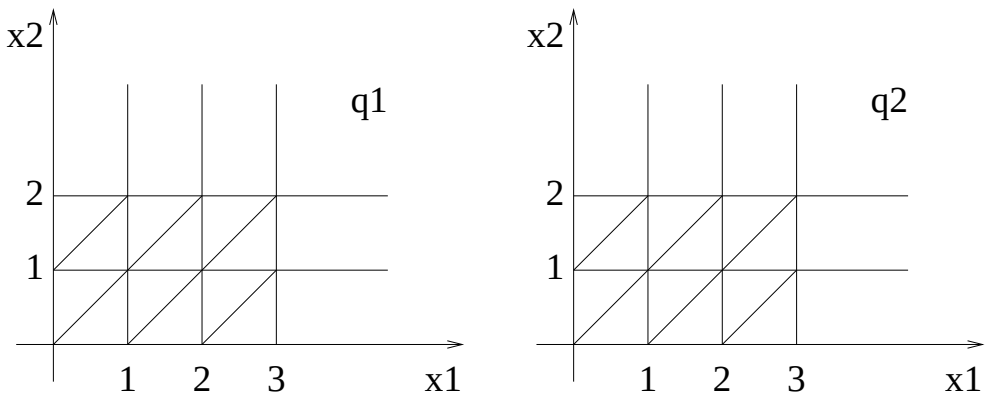

Figure 4: Equivalence classes for the example

- We can therefore assume all constants are integers. If they are not, we let λ be a common multiple of their denominators and consider the bisimilar system T_λ.
- Let c_i denote the largest constant with which x_i is compared.
- In the above example, $c_1 = 3$ and $c_2 = 2$.
- Let $\lfloor x_i \rfloor$ denote the integer part of x_i and $\langle x_i \rangle$ denote the fractional part of x_i. In other words, $x_i = \lfloor x_i \rfloor + \langle x_i \rangle$, $\lfloor x_i \rfloor \in \mathbb{Z}$ and $\langle x_i \rangle \in [0, 1)$.
- Consider the relation $\sim \subseteq \mathbf{Q} \times \mathbf{X}$ with $(q, x) \sim (q', x')$ if:
  1. $q = q'$;
  2. for all x_i, $\lfloor x_i \rfloor = \lfloor x'_i \rfloor$ or $(\lfloor x_i \rfloor > c_i) \vee (\lfloor x'_i \rfloor > c_i)$
  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 $x_i \leq c_i$,
$$(\langle x_i \rangle = 0) \Leftrightarrow (\langle x'_i \rangle = 0)$$

Proposition 9 $\sim$ *is an equivalence relation.*

- What do the equivalence classes look like?
- The equivalence classes are either open triangles, open line segments, open parallelograms or points.
- For the example introduced above, they are shown in Figure 4.
- Notice that the number of classes is $2 \times (12 \text{ points} + 30 \text{ lines} + 18 \text{ open sets})$. Quite a few, but definitely finite!

Proposition 10 $\sim$ *is a bisimulation.*

Proof: We need to show:

1. Init is a union of equivalence classes.
2. F is a union of equivalence classes.
3. If P is an equivalence class and $e \in E$, $\text{Pre}_e(P)$ is a union of equivalence classes.
4. If P is an equivalence class, $\text{Pre}_\tau(P)$ is a union of equivalence classes.

The proof will be somewhat informal. ∎

- First, note that if $\delta \in \Phi(X)$,
$$\hat{\delta} = \{x \in \mathbf{X} \ : \ \delta(x) = \text{True}\}$$
is "a union of equivalence classes" (for the X variables only). Recall that $\hat{\delta}$ can be written as the product of unions and intersections of sets of the form:
$$\{x_i \geq c\}, \{x_i \leq c\}, \{x_i < c\}, \{x_i > c\}, \{x_i = c\}$$
where c is an integer constant. All these sets are unions of equivalence classes for the X variables.
- This takes care of the requirements on Init and F.
- To deal with $\text{Pre}_e(P)$, let:
$$R^{-1}(e, P) = \{(q, x) \in \mathbf{Q} \times \mathbf{X} \ : \ \exists (q', x') \in P \text{ with } e = (q, q'), x' \in R(e, x)\}$$
- Notice that:
$$\text{Pre}_{(q,q')}(P) = R^{-1}(q, q', P) \cap (\{q\} \times G(q, q'))$$
Proposition 11 *If P is an equivalence class, $R^{-1}(e, P)$ is a union of equivalence classes. Hence* $\text{Pre}_e(P)$ *is a union of equivalence classes.*
- Easier to demonstrate by examples. For the timed automaton of Figure 3, consider the equivalence classes $P_1, \ldots, P_4$ shown in Figure 5. Notice that:
$$\begin{aligned}
\text{Pre}_{e_1}(P_1) &= \emptyset, \\
\text{Pre}_{e_2}(P_1) &= Q_1 \cap (\{q_2\} \times \{x_1 \leq 1\}) = \emptyset \\
\text{Pre}_{e_1}(P_2) &= \emptyset, \\
\text{Pre}_{e_2}(P_2) &= Q_2 \cap (\{q_2\} \times \{x_1 \leq 1\}) = Q_2 \\
\text{Pre}_{e_1}(P_3) &= \emptyset \cap (\{q_1\} \times \{x_1 \leq 1 \wedge x_2 \leq 2\}) = \emptyset, \\
\text{Pre}_{e_2}(P_3) &= \emptyset \\
\text{Pre}_{e_1}(P_4) &= \{q_1\} \times (\{x_1 \geq 0 \wedge 1 < x_2 < 2\} \cap \{x_1 \leq 1 \wedge x_2 \leq 2\}) \\
&= \{q_1\} \times \{0 \leq x_1 \leq 3 \wedge 1 < x_2 < 2\}, \\
\text{Pre}_{e_2}(P_4) &= \emptyset
\end{aligned}$$

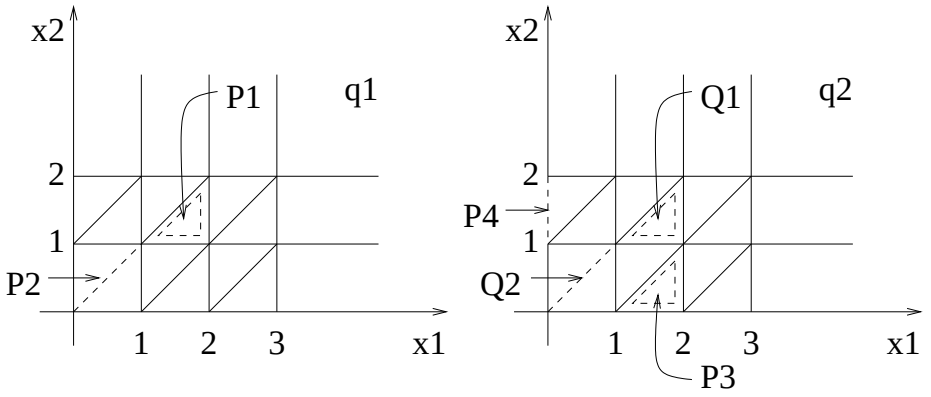

Figure 5: Examples of Pre_e computation

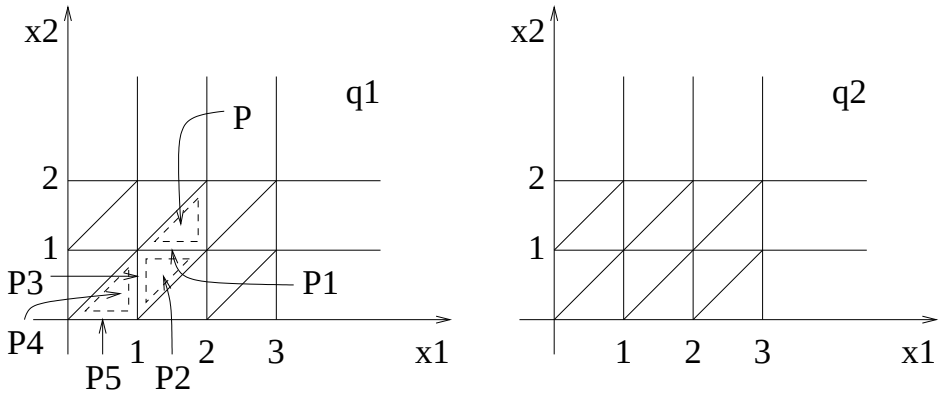

Figure 6: Examples of Pre_τ computation

In all cases the result is a union of equivalence classes.

- Finally, notice that

$$\mathrm{Pre}_\tau(P) = \{(q,x) \in \mathbf{Q} \times \mathbf{X} \ : \ \exists (q',x') \in P, t \geq 0 \text{ with } q = q', x' = x + t(1,\ldots,1)\}$$

These are all points that if we move in the $(1,\ldots,1)$ direction we will eventually reach P. If P is an equivalence class, this set is also a union of equivalence classes.

- For example, in Figure 6, $\mathrm{Pre}_\tau(P) = P \cup P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5$

7 Complexity Estimates and Generalizations

- The above discussion indicates that reachability questions for timed automata can be answered on a finite state system, defined on the quotient space of the bisimulation $\sim$ (also known as the *region graph*).
- What is the "size" of this finite state system?

- For the example, the number of equivalence classes is 2(12points+30lines+18open sets) = 120. Quite a few!
- For an arbitrary system, one can expect up to $m(n!)(2^n)\prod_{i=1}^{n}(2c_i+2)$ discrete states.
- Of course, in general, one need not construct the entire region graph:
  1. On the fly reachability: run the reachability algorithm. Typically it will terminate, without constructing the entire region graph.
  2. Construct coarser bisimulations: run the bisimulation algorithm. Typically the bisimulation generated will have fewer equivalence classes than the region graph.
- Still the problem is PSPACE complete!
- The finite bisimulation result is preserved under some simple extensions:
  1. $I(q) = \hat{I}_q$ for some $I_q \in \Phi(X)$.
  2. $f(q,x) = (k_1, \ldots, k_n)$ for some rational $k_1, \ldots, k_n$ and all (q,x).
  3. R mapping equivalence classes to equivalence classes.

8 Rectangular Hybrid Automata

- The largest class of systems of this form that is known to be decidable is the class of *initialized rectangular automata* [2, 3, 4, 5].
- A set $R \subset \mathbb{R}^n$ is called a rectangle if $R = \prod_{i=1}^{n} R_i$ where R_i are intervals whose finite end points are rational.

  Definition 12 (Rectangular Automaton) *A rectangular automaton is a hybrid automaton* $H = (Q, X, \mathrm{Init}, f, \mathrm{Dom}, R)$, *where*
  - Q *is a set of discrete variables,* $\mathbf{Q} = \{q_1, \ldots, q_m\}$;
  - $X = \{x_1, \ldots, x_n\}$, $X = \mathbb{R}^n$;
  - $\mathrm{Init} = \cup_{i=1}^{m}\{q_i\} \times \mathrm{Init}(q_i)$ *where* $\mathrm{Init}(q_i)$ *is a rectangle;*
  - $f(q,x) = F(q)$ *for all* (q,x), *where* $F(q)$ *is a rectangle;*
  - $\mathrm{Dom}(q)$ *is a rectangle for all* $q \in \mathbf{Q}$;
  - $R(q,x)$ *either leaves* x_i *unaffected or resets to an arbitrary value in a rectangle.*

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[2] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In G. Cohen and J.-P. Quadrat, editors, *Proceedings of the 11th International Conference on Analysis and Optimization of Systems: Discrete-event Systems*, number 199 in LNCS, pages 331–351. Springer Verlag, 1994.

[3] T.A. Henzinger. Hybrid automata with finite bisimulations. In Z. Fülöp and F. Gécseg, editors, *ICALP 95: Automata, Languages, and Programming*, number 944 in LNCS, pages 324–335. Springer Verlag, 1995.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, et al. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, Vol.138(1):3–4, 1995.

[5] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, LNCS, pages 366–392. Springer Verlag, New York, 1993.