Lecture 12: Algorithmic Verification of Hybrid Systems

A. Pnueli

We will start by showing that the general reachability problem for hybrid systems is undecidable. Then, we will identify a family of special systems for which reachability can be algorithmically decided. Both results will be demonstrated on a restricted class of hybrid systems, called Constant Slope Hybrid Systems (CSS).

These are systems which obey the following restrictions:

1. The time progress condition is trivially satisfied for any location.
2. The guard (enabling condition) of any transition is a conjunction of linear equalities/inequalities of the form $P_i x = a_i$, where $a_i \in \mathbb{Z}$ and $x \in C$.
3. The only allowed modification to a continuous variable $x_i$ in a discrete transition is a reset of the form $x_i := 0$.
4. The set of discrete variables $D$ consists of boolean variables $y_1, \ldots, y_b$ and a program counter which ranges over a finite set of locations $L$.

The guard for any terminal location $L_f \in L_f$ is $\emptyset$, and the only transition enabled on an $L_f$-location is the unique one leading to the location of the state.

Graphical Representation of CSS

Following is a graphical representation of a CSS corresponding to the Cat and Mouse System:

Initially $x_0 = 0, x = 0$.

The following is a graphical representation of the Cat and Mouse System:

- Initially $x_0 = 0, x = 0$.
- The time progress condition is trivially satisfied.
- For every location $L \in L_I$ and continuous variable $x \in C$, the associated activity is given by $a_i x = a_i$, where $a_i \in \mathbb{R}$.
- The only allowed modification to a continuous variable $x_i$ in a discrete transition is a reset of the form $x_i := 0$.

Example: Cat and Mouse System

Following is a graphical representation of a CSS corresponding to the Cat and Mouse System:

Initially $x_0 = 0, x = 0$.

The following is a graphical representation of the Cat and Mouse System:

Initially $x_0 = 0, x = 0$.

The time progress condition is trivially satisfied for any location.

The guard (enabling condition) of any transition is a conjunction of linear equalities/inequalities of the form $P_i x = a_i$, where $a_i \in \mathbb{Z}$ and $x \in C$.

The only allowed modification to a continuous variable $x_i$ in a discrete transition is a reset of the form $x_i := 0$.

The set of discrete variables $D$ consists of boolean variables $y_1, \ldots, y_b$ and a program counter which ranges over a finite set of locations $L$.

The guard for any terminal location $L_f \in L_f$ is $\emptyset$, and the only transition enabled on an $L_f$-location is the unique one leading to the location of the state.

The following is a graphical representation of the Cat and Mouse System:

Initially $x_0 = 0, x = 0$.
A typical reachability problem for this system is the mouse safety property, which can be formulated as follows:

A mouse wins if
\[
0 < x < X
\]
and a computer wins if
\[
0 = x
\]
Under the assumption of
\[
\frac{\partial y}{\partial x} + \nabla > \frac{\partial x}{\partial x}
\]
show that there exists no computation reaching location \( \text{bad} \).

Example: Gas Burner System

Another example is the Gas Burner system:

Initially
\[
x = y = z = 0
\]
\[
x > 0 < x
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
\[
0 = z
\]
\[
0 = x
\]
\[
0 = y
\]
Reachability is undecidable for CSS’s

We will show that the reachability problem for CSS is undecidable. This is proven by reduction to an -counter machine.

A CSS is said to be an integration graph if it satisfies the following requirements:

- An edge is called cyclic if it participates in some cycle in the graph.
- A CSS is called cyclic if it appears on a cycle edge.
- No irregular test appears in an integration graph.
- The reachability problem for CSS’s is undecidable. This is proven by reduction to an -counter machine.

### Integration Graphs

By the following two subgraphs:

- A continuous variable whose slope is 1 at every location is called a timer.
- All other continuous variables are called integrators. For example, in system GAS-BURNER, and are timers while is an integrator.

### Timers

The reachability problem for general CSS’s is undecidable.

#### The Reduction

Let be a counter machine with counters, program labels, and guard labels. The CSS will contain locational corresponding to .

With no loss of generality, assume that the labels and program labels .

Let be counter machine with counters, program labels, and guard labels.

The reduction is undecidable if is a CSS’s integration graph.

If then go to Else go to

**Conclusion**

A CSS is called cyclic if it appears on a cycle edge. A CSS is called cyclic if it satisfies the following requirements:

- An edge is called cyclic if it participates in some cycle in the graph.
- A CSS is called cyclic if it appears on a cycle edge.
- No irregular test appears in an integration graph.
- The reachability problem for CSS’s is undecidable. This is proven by reduction to an -counter machine.

By the following two subgraphs:

- A continuous variable whose slope is 1 at every location is called a timer.
- All other continuous variables are called integrators. For example, in system GAS-BURNER, and are timers while is an integrator.

### Timers

The reachability problem for general CSS’s is undecidable.

#### The Reduction

Let be a counter machine with counters, program labels, and guard labels. The CSS will contain locational corresponding to .

With no loss of generality, assume that the labels and program labels .

Let be counter machine with counters, program labels, and guard labels.

The reduction is undecidable if is a CSS’s integration graph.

If then go to Else go to

**Conclusion**

A CSS is called cyclic if it appears on a cycle edge. A CSS is called cyclic if it satisfies the following requirements:

- An edge is called cyclic if it participates in some cycle in the graph.
- A CSS is called cyclic if it appears on a cycle edge.
- No irregular test appears in an integration graph.
- The reachability problem for CSS’s is undecidable. This is proven by reduction to an -counter machine.

By the following two subgraphs:

- A continuous variable whose slope is 1 at every location is called a timer.
- All other continuous variables are called integrators. For example, in system GAS-BURNER, and are timers while is an integrator.
If variable $x$ appears in an irregular test which labels an edge entering location $L$, then the slope of $x$ is $0$ at all locations reachable from $L$. That is, after being irregularly tested, no variable can further change its value.

It is not difficult to see that if a system 

\[ \text{Cat and Mouse} \]

which has no cycles at all, and

\[ \text{GAS-BURNER} \]

are both integration graphs, then it is not difficult to see that each variable is not reachable in the other. There are two such systems which are equivalent to each other, but they are not equivalent to the system 

\[ \text{Cat and Mouse} \]

which has no cycles at all, and

\[ \text{GAS-BURNER} \]

are both integration graphs. The two systems are equivalent if and only if they have the same number of locations and the same number of edges.

A continuous variable is called irregular if it appears in an irregular test.

For example, the original GAS-BURNER system violates the requirement of partitionability with respect to the irregular variables $y$ and $z$. Without loss of generality, we assume that $L = L_f$, that is, all terminal locations belonging to $L_f$.

A continuous variable is called irregular if it appears in an irregular test.

The Cat and Mouse system contains no cycles. Therefore, it is decomposable into a looping and a testing part such that all the system locations $L_f \subseteq L$ and $L_f \cap R_x = \emptyset$, where $L_f$ is the set of non-terminal locations.

Every integration graph can be transformed into an equivalent partitioned graph.

A continuous variable is called irregular if it appears in an irregular test.

\[ R_x \]

The $R_x$-locations have no predecessors in the graph. Every edge leaving an $R_x$-location resets variable $x$. Every edge connecting an $R_x$-location to an $N_x$-location resets variable $x$.

An integration graph is said to be partitioned if for every irregular variable $x \in C$ the set of non-resetting locations $N_x$ and the set of non-resetting locations $R_x$ such that $N_x \cup R_x = L_f$.

\[ L_f \]

are both integration graphs. The two graphs are equivalent if and only if they have the same number of locations and the same number of edges.

For example, the original GAS-BURNER system violates the requirement of partitionability with respect to their regular variables $y$ and $z$. Without loss of generality, we assume that $L = L_f$, that is, all terminal locations belonging to $L_f$.

A continuous variable is called irregular if it appears in an irregular test.

The Cat and Mouse system contains no cycles. Therefore, it is decomposable into a looping and a testing part such that all the system locations $L_f \subseteq L$ and $L_f \cap R_x = \emptyset$, where $L_f$ is the set of non-terminal locations.

Every integration graph can be transformed into an equivalent partitioned graph.

A continuous variable is called irregular if it appears in an irregular test.

The $R_x$-locations have no predecessors in the graph. Every edge leaving an $R_x$-location resets variable $x$. Every edge connecting an $R_x$-location to an $N_x$-location resets variable $x$.

An integration graph is said to be partitioned if for every irregular variable $x \in C$ the set of non-resetting locations $N_x$ and the set of non-resetting locations $R_x$ such that $N_x \cup R_x = L_f$.

\[ L_f \]

are both integration graphs. The two graphs are equivalent if and only if they have the same number of locations and the same number of edges.
A Partioned Version of GAS-BURNER

Following is P-GAS-BURNER which is a partitioned version of GAS-BURNER:

\[ \begin{align*}
0 &= z = \beta = x \\
\text{Initially} &
\end{align*} \]

Finitary Timed Automata

We can obtain a version of a timed automaton by placing the following additional restrictions on a CSS:

1. All guards are of the form \( L \leq x \leq U \) for a timer \( x \) and natural constants \( L, U \in \mathbb{N} \).
2. The initial value of all timers is 0.
3. Variables are timers.
4. The slope of all continuous variables at all locations is 1. That is, all continuous variables are timers.

\[ \begin{align*}
0 &= z = \beta = x \\
\text{Initially} &
\end{align*} \]
We define a **computation segment** of an FTA to be a finite sequence of states which satisfies the requirements of initiation and consecution. A computation segment is called a **computation** if the location at $s_k$ is terminal. A computation segment $\sigma : s_0, \ldots, s_k$ of an FTA is associated with a **trail** $\tau : (l_0, d_0), (l_1, d_1), \ldots, (l_m, d_m)$ where $l_i, d_i, \forall i$ is the sequence of locations visited by $\sigma$ and $d_i$ is the length of time the computation spent at location $l_i$ on the current visit.