Interval Methods for the Analysis of Hybrid Dynamical Systems

Stefan Ratschan
Tomáš Dzetkulič, Zhikun She, Jan-Georg Smaus et. al.

June 4, 2009
Hybrid Systems

Thermostat:

\[ 0 \leq x \leq 30 \]

\[ \dot{x} = -x \] (off)

\[ x \leq 18 \]

\[ x \geq 22 \]

\[ \dot{x} = -x + 40 \] (on)
Hybrid Systems

Thermostat:

- $0 \leq x \leq 30$: 
  - $\dot{x} = -x$

- $x \leq 18$: 
  - $\dot{x} = -x + 40$

- $x \geq 22$: 
  - $\dot{x} = -x$

Dynamical system with both continuous and discrete state and evolution. The continuous state can change discontinuously non-deterministically, i.e., non-forced jumps. The state is bounded in the interval $[2, 18]$. 

Graph showing the evolution of $x$ over time $t$. The graph indicates the behavior of the system under different conditions.
Hybrid Systems

Thermostat:

Dynamical system with both continuous and discrete state and evolution.
Hybrid Systems

Thermostat:

\[ x < 18 \implies \dot{x} = -x \]
\[ x \geq 22 \implies \dot{x} = -x + 40 \]

Dynamical system with both continuous and discrete state and evolution.

also continuous state can change discontinuously
Hybrid Systems

Thermostat:

Dynamical system with **both continuous and discrete** state and evolution.

also continuous state can change discontinuously

non-determinism: non-forced jumps
Hybrid Systems

Thermostat:

Dynamical system with both continuous and discrete state and evolution.

also continuous state can change discontinuously

non-determinism: non-forced jumps, $x - 1 \leq \dot{x} \land \dot{x} \leq x + 1$
Hybrid Systems

Thermostat:

\[ 0 \leq x \leq 30 \quad \text{off} \]

\[ x \leq 18 \quad \dot{x} = -x \]

\[ x \geq 22 \quad \dot{x} = -x + 40 \]

\[ 0 \leq x \leq 30 \quad \text{on} \]

Dynamical system with both continuous and discrete state and evolution.

also continuous state can change discontinuously

non-determinism: non-forced jumps, \( x - 1 \leq \dot{x} \land \dot{x} \leq x + 1 \)

non-linearity
Hybrid Systems

Thermostat:

\[ \begin{align*}
0 \leq x &\leq 30 & \Rightarrow & \text{off} \\
0 \leq x &\leq 18 & \Rightarrow & \dot{x} = -x \\
x &\geq 22 & \Rightarrow & \dot{x} = -x + 40 \\
\end{align*} \]

Dynamical system with **both continuous and discrete** state and evolution.

also continuous state can change discontinuously

non-determinism: non-forced jumps, \( x - 1 \leq \dot{x} \land \dot{x} \leq x + 1 \)

non-linearity

bounded state space
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equalities, e.g., $s = \text{off}$, $s \neq \text{firstgear}$
- arithmetical (in)equalities, e.g., $x^2 \leq 1$
- Init$(s, \vec{x})$ (e.g., $s = \text{firstgear} \land 0 \leq x \land x \leq 10$)
- Unsafe$(s, \vec{x})$ (e.g., $x \geq 8000$)
- Flow$(s, \vec{x}, \dot{\vec{x}})$ (e.g., $s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow ...$
- even implicit and algebraic!
- Jump$(s, \vec{x}, s', \vec{x}')$ (e.g., $(s = \text{off} \land x \geq 10) \rightarrow (s' = \text{on} \land x' = 0)$)
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equalities, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equalities, e.g., \( s = \text{off} \), \( s \neq \text{firstgear} \)
- arithmetical (in)equalities, e.g., \( x^2 \leq 1 \)
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equality, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
- arithmetical (in)equality, e.g., \( x^2 \leq 1 \)

- \( \text{Init}(s, \vec{x}) \) (e.g., \( s = \text{firstgear} \land 0 \leq x \land x \leq 10 \))
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equalities, e.g., $s = \text{off}$, $s \neq \text{firstgear}$
- arithmetical (in)equalities, e.g., $x^2 \leq 1$

- $\text{Init}(s, \vec{x})$ (e.g., $s = \text{firstgear} \land 0 \leq x \land x \leq 10$)
- $\text{Unsafe}(s, \vec{x})$ (e.g., $x \geq 8000$)

- $\text{Flow}(s, \vec{x}, \dot{\vec{x}})$ (e.g., $s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow ...$)

- for algorithms, $\dot{x}$ purely syntactic!

- even implicit and algebraic!

- $\text{Jump}(s, \vec{x}, s', \vec{x}')$ (e.g., $(s = \text{off} \land x \geq 10) \rightarrow (s' = \text{on} \land x' = 0)$)
Specification using Constraints

Constraint: Boolean combination of

- mode (dis)equalities, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
- arithmetical (in)equalities, e.g., \( x^2 \leq 1 \)

- \( \text{Init}(s, \vec{x}) \) (e.g., \( s = \text{firstgear} \land 0 \leq x \land x \leq 10 \))
- \( \text{Unsafe}(s, \vec{x}) \) (e.g., \( x \geq 8000 \))
- \( \text{Flow}(s, \vec{x}, \dot{\vec{x}}) \) (e.g.,
  \( s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow \ldots \))
Constraint: Boolean combination of

- mode (dis)equalities, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
- arithmetical (in)equalities, e.g., \( x^2 \leq 1 \)

- \( \text{Init}(s, \vec{x}) \) (e.g., \( s = \text{firstgear} \land 0 \leq x \land x \leq 10 \))
- \( \text{Unsafe}(s, \vec{x}) \) (e.g., \( x \geq 8000 \))
- \( \text{Flow}(s, \vec{x}, \dot{\vec{x}}) \) (e.g.,
  \( s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow \ldots \))
  - for algorithms, \* purely syntactic! \*
Constraint: Boolean combination of

- mode (dis)equalities, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
- arithmetical (in)equalities, e.g., \( x^2 \leq 1 \)

- \( \text{Init}(s, \vec{x}) \) (e.g., \( s = \text{firstgear} \land 0 \leq x \land x \leq 10 \))
- \( \text{Unsafe}(s, \vec{x}) \) (e.g., \( x \geq 8000 \))
- \( \text{Flow}(s, \vec{x}, \dot{\vec{x}}) \) (e.g.,
  \( s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow \ldots \))
  - for algorithms, purely syntactic!
  - even implicit and algebraic!
Specification using Constraints

Constraint: Boolean combination of
- mode (dis)equalities, e.g., \( s = \text{off}, s \neq \text{firstgear} \)
- arithmetical (in)equalities, e.g., \( x^2 \leq 1 \)
- \( \text{Init}(s, \vec{x}) \) (e.g., \( s = \text{firstgear} \land 0 \leq x \land x \leq 10 \))
- \( \text{Unsafe}(s, \vec{x}) \) (e.g., \( x \geq 8000 \))
- \( \text{Flow}(s, \vec{x}, \dot{\vec{x}}) \) (e.g.,
  \[ s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \land s = \text{on} \rightarrow \ldots \]
  - for algorithms, \`purely syntactic!\`
  - even implicit and algebraic!
- \( \text{Jump}(s, \vec{x}, s', \vec{x}') \) (e.g.,
  \( (s = \text{off} \land x \geq 10) \rightarrow (s' = \text{on} \land x' = 0) \))
Goal

Automatically **verify** that a given hybrid system is **safe**:

There is no trajectory that

- starts in an initial state,
- evolves according to *Flow*, *Jump*, and
- reaches an unsafe state.

That is, there is no *error trajectory*. 
Interval Grid Method

Stursberg/Kowalewski et. al., illustration: one mode, two dim.

\[
\dot{x} = f(x)
\]

\[\dot{x} \in [-5, 1]\]

- put transitions between all neighboring hyper-rectangles (boxes), mark all as initial/unsafe
- remove impossible transitions/marks (interval arithmetic check on boundaries/boxes)

Result: finite abstraction (over-approximates, finite)
Interval Grid Method

Stursberg/Kowalewski et. al., illustration: one mode, two dim.

- put transitions between all neighboring hyper-rectangles (boxes), mark all as initial/unsafe
Interval Grid Method

Stursberg/Kowalewski et. al., illustration: one mode, two dim.

\[ \dot{x} = f(x) \]
\[ \dot{x} \in [-5, -1] \]

- put transitions between all neighboring hyper-rectangles (boxes), mark all as initial/unsafe
- remove impossible transitions/marks (interval arithmetic check on boundaries/boxes)
Interval Grid Method

Stursberg/Kowalewski et. al., illustration: one mode, two dim.

\[ \dot{x} = f(x) \]
\[ \dot{x} \in [-5, 1] \]

- put transitions between all neighboring hyper-rectangles (boxes), mark all as initial/unsafe
- remove impossible transitions/marks (interval arithmetic check on boundaries/boxes)
Interval Grid Method

Stursberg/Kowalewski et. al., illustration: one mode, two dim.

\[ \dot{x} = f(x) \]
\[ \dot{x} \in [-5, 1] \]

- put transitions between all neighboring hyper-rectangles (boxes), mark all as initial/unsafe
- remove impossible transitions/marks (interval arithmetic check on boundaries/boxes)

Result: finite abstraction (over-approximates, finite)
Interval Grid Method II

Check safety on resulting finite abstraction

if safe: finished, otherwise: refine grid;
continue until success
Analysis

Advantages:

▶ general
▶ can do verification instead of verification modulo rounding errors
▶ interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:

▶ may require a very fine grid to provide an affirmative answer (curse of dimensionality)
▶ ignores the continuous behavior within the grid elements

Let's remove them!
Analysis

Advantages:

- general

Disadvantages:

- may require a very fine grid to provide an affirmative answer (curse of dimensionality)
- ignores the continuous behavior within the grid elements
Analysis

Advantages:

▶ general

▶ can do verification instead of verification modulo rounding errors
Analysis

Advantages:

- general
- can do verification instead of verification modulo rounding errors
- interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)
Analysis

Advantages:

- general
- can do verification instead of verification modulo rounding errors
- interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:
Analysis

Advantages:

- general
- can do verification instead of verification modulo rounding errors
- interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:

- may require a very fine grid to provide an affirmative answer (curse of dimensionality)
Analysis

Advantages:

- general
- can do verification instead of verification modulo rounding errors
- interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:

- may require a very fine grid to provide an affirmative answer (curse of dimensionality)
- ignores the continuous behavior within the grid elements
Analysis

Advantages:

- general
- can do verification instead of verification modulo rounding errors
- interval tests cheap (e.g., compared to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:

- may require a very fine grid to provide an affirmative answer (curse of dimensionality)
- ignores the continuous behavior within the grid elements

Let’s remove them!
Abstraction Pruning

Reflect more information in abstraction, without creating more boxes by splitting
Abstraction Pruning

Reflect more information in abstraction, without creating more boxes by splitting

Observation: parts of state space not lying on an error trajectory not needed, remove such parts from boxes
Abstraction Pruning

Reflect more information in abstraction, without creating more boxes by splitting

Observation: parts of state space not lying on an error trajectory not needed, remove such parts from boxes
Abstraction Pruning

Reflect more information in abstraction, without creating more boxes by splitting

Observation: parts of state space not lying on an error trajectory not needed, remove such parts from boxes
Abstraction Pruning

Reflect **more information** in abstraction, **without** creating **more boxes** by splitting

Observation: parts of state space not lying on an error trajectory not needed, **remove** such parts from boxes

Method: form **constraints** that hold on error trajectories, **remove** non-solutions.
Constraints

A point on an error trajectory is reachable from an initial state, and leads to an unsafe state.
Constraints

A point on an error trajectory is **reachable** from an initial state, and **leads** to an unsafe state.

A point in a box $B$ can be reachable

- from the **initial set** via a flow in $B$
- from a **jump** via a flow in $B$
- from a **neighboring box** via a flow in $B$
Constraints

A point on an error trajectory is reachable from an initial state, and leads to an unsafe state.

A point in a box $B$ can be reachable

- from the initial set via a flow in $B$
- from a jump via a flow in $B$
- from a neighboring box via a flow in $B$

formulate corresponding constraints, remove non-solutions
Example of Constraint

If $\vec{y} \in B$ is reachable from the initial set via a flow in $B$ then $\exists \vec{x} \in B \left[ \text{Init}(\vec{x}) \land \text{flow}_B(\vec{x}, \vec{y}) \right]$

In all three cases we need: $\text{flow}_B(\vec{x}, \vec{y})$: there exists a flow in $B$ from $\vec{x}$ to $\vec{y}$
Example of Constraint

If $\vec{y} \in B$ is reachable from the initial set via a flow in $B$ then

$$\exists \vec{x} \in B [Init(\vec{x}) \land \text{flow}_B(\vec{x}, \vec{y})]$$
Example of Constraint

If $\vec{y} \in B$ is reachable from the initial set via a flow in $B$ then

$$\exists \vec{x} \in B \left[ \text{Init}(\vec{x}) \wedge \text{flow}_B(\vec{x}, \vec{y}) \right]$$

In all three cases we need:

$flow_B(\vec{x}, \vec{y})$: there exists a flow in $B$ from $\vec{x}$ to $\vec{y}$
Flow Constraint

A flow is a smooth function \( u : [0, t] \rightarrow \mathbb{R}^n \), such that for all \( t' \in [0, t] \), \( \text{Flow}(u(t'), \dot{u}(t')) \)
Flow Constraint

A *flow* is a smooth function \( u : [0, t] \rightarrow \mathbb{R}^n \), such that for all \( t' \in [0, t] \), \( \text{Flow}(u(t'), \dot{u}(t')) \)

What do we *know* for flows?
Flow Constraint

A flow is a smooth function $u : [0, t] \rightarrow \mathbb{R}^n$, such that for all $t' \in [0, t]$, $\text{Flow}(u(t'), \dot{u}(t'))$

What do we know for flows?

- case $u : [0, t] \rightarrow \mathbb{R}$
A flow is a smooth function $u : [0, t] \to \mathbb{R}^n$, such that for all $t' \in [0, t]$, $Flow(u(t'), \dot{u}(t'))$

What do we know for flows?

Case $u : [0, t] \to \mathbb{R}$, s.t. $u(0) = x$, $u(t) = y$
Flow Constraint

A flow is a smooth function $u : [0, t] \to \mathbb{R}^n$, such that for all $t' \in [0, t]$, $\text{Flow}(u(t'), \dot{u}(t'))$

What do we know for flows?

- case $u : [0, t] \to \mathbb{R}$, s.t. $u(0) = x$, $u(t) = y$

\[ \exists t \geq 0 \exists t' \in [0, t] \left[ x = u(0) \land y = u(t) \land u(t) = T_{u,k}([0, t], t') \right], \]

where $T_{u,k}$ Taylor polynomial + remainder term
Remaining Problems:

- $T_{u,k}([0, t], t')$ contains derivatives, only implicitly given
Flow Constraint

Remaining Problems:

- $T_{u,k}([0, t], t')$ contains derivatives, only implicitly given
- flows $\mathbb{R} \rightarrow \mathbb{R}^n$?
Flow Constraint

Remaining Problems:

- $T_{u,k}([0, t], t')$ contains derivatives, only implicitly given
- flows $\mathbb{R} \rightarrow \mathbb{R}^n$?
- solving the constraints (i.e., remove non-solutions)
Flow Constraint

Remaining Problems:

- $T_{u,k}([0, t], t')$ contains derivatives, only implicitly given
- flows $\mathbb{R} \to \mathbb{R}^n$?
- solving the constraints (i.e., remove non-solutions)

Assumption: constraint $Flow^{(k)}(x, \dot{x})$. 
Flow Constraint

Remaining Problems:

- \( T_{u,k}([0, t], t') \) contains derivatives, only implicitly given
- flows \( \mathbb{R} \rightarrow \mathbb{R}^n \)?
- solving the constraints (i.e., remove non-solutions)

Assumption: constraint \( \text{Flow}^{(k)}(x, \dot{x}) \).

In addition, we know: flow in \( B \).
Flow Constraint

Remaining Problems:

- $T_{u,k}([0, t], t')$ contains derivatives, only implicitly given
- flows $\mathbb{R} \rightarrow \mathbb{R}^n$?
- solving the constraints (i.e., remove non-solutions)

Assumption: constraint $Flow^{(k)}(x, \dot{x})$.

In addition, we know: flow in $B$.

So: for $u^{(k)}(t')$, extend constraint with

$$\exists u(t') \exists u^{(k)}(t') \left[ u(t') \in B \land Flow^{(k)}(u(t'), u^{(k)}(t')) \land \ldots \right],$$

where $u(t')$, $u^{(k)}(t')$ are fresh variables.
Multi-dimensional Flows

Given: \( u(t) : [0, t] \rightarrow \mathbb{R}^n \)

To get to one-dimensional flow:

For whatever choice of \( P \), one gets a corresponding constraint.

\[ P(x) = P(u(0)) \land P(y) = P(u(t)) \land P(u(t)) = T \]

For example: Use axis projections \( P_i(a_1, \ldots, a_n) = a_i, i = 1, \ldots, n \).
Multi-dimensional Flows

Given: \( u(t) : [0, t] \rightarrow \mathbb{R}^n \)

To get to one-dimensional flow: \( P(u(t)) \), where \( P : \mathbb{R}^n \rightarrow \mathbb{R} \)
Multi-dimensional Flows

Given: \( u(t) : [0, t] \rightarrow \mathbb{R}^n \)

To get to one-dimensional flow: \( P(u(t)) \), where \( P : \mathbb{R}^n \rightarrow \mathbb{R} \)

For whatever choice of \( P \), one gets a corresponding constraint.

\[
P(x) = P(u(0)) \land P(y) = P(u(t)) \land P(u(t)) = T_{P(u),k}([0, t], t')
\]
Multi-dimensional Flows

Given: \( u(t) : [0, t] \rightarrow \mathbb{R}^n \)

To get to one-dimensional flow: \( P(u(t)) \), where \( P : \mathbb{R}^n \rightarrow \mathbb{R} \)

For whatever choice of \( P \), one gets a corresponding constraint.

\[
P(x) = P(u(0)) \land P(y) = P(u(t)) \land P(u(t)) = T_{P(u), k}([0, t], t')
\]

For example: Use axis projections \( P_i(a_1, \ldots, a_n) = a_i, i = 1, \ldots, n \),

\[
flow_B(x_1, \ldots, x_n, y_1, \ldots, y_n) \doteq \exists t \geq 0 \bigwedge_{i \in 1, \ldots, n} \exists t' \in [0, t] \bigwedge \[ x_i = P_i(u(0)) \land y_i = P_i(u(t)) \land P_i(u(t)) = T_{P_i(u), k}([0, t], t') \]

Note: all projected flows have same length, so: \( t \) shared!
Multi-dimensional Flows

Given: \( u(t) : [0, t] \to \mathbb{R}^n \)

To get to one-dimensional flow: \( P(u(t)) \), where \( P : \mathbb{R}^n \to \mathbb{R} \)

For whatever choice of \( P \), one gets a corresponding constraint.

\[
P(x) = P(u(0)) \land P(y) = P(u(t)) \land P(u(t)) = T_{P(u),k}([0, t], t')
\]

For example: Use axis projections \( P_i(a_1, \ldots, a_n) = a_i, i = 1, \ldots, n, \)

\[
flow_B(x_1, \ldots, x_n, y_1, \ldots, y_n) \overset{\text{def}}{=} \exists t \geq 0 \bigwedge_{i \in 1, \ldots, n} \exists t' \in [0, t] \[ x_i = P_i(u(0)) \land y_i = P_i(u(t)) \land P_i(u(t)) = T_{P_i(u),k}([0, t], t') \]

Note: all projected flows have same length, so: \( t \) shared!
Choice of Projection Functions?

Assumption: \( \dot{u} = Au, \ A \in \mathbb{R}^{n \times n} \)
Choice of Projection Functions?

Assumption: \( \dot{u} = Au, \ A \in \mathbb{R}^{n \times n} \)

Then: \( P_i(u(t)) \) can be written analytically, but (in general) is a sum of \( n \) terms of the form \( ce^{\lambda t}, ae^{\alpha t} \cos \beta t + be^{\alpha t} \sin \beta t \).
Choice of Projection Functions?

Assumption: $\dot{u} = Au$, $A \in \mathbb{R}^{n \times n}$

Then: $P_i(u(t))$ can be written analytically, but (in general) is a sum of $n$ terms of the form $ce^{\lambda t}$, $ae^{\alpha t} \cos \beta t + be^{\alpha t} \sin \beta t$.

But: if $P(x) = p^T x$, where $p$ is a right eigenvector of $A$, then analytic solution only one of above terms!
Choice of Projection Functions?

Assumption: $\dot{u} = Au$, $A \in \mathbb{R}^{n \times n}$

Then: $P_i(u(t))$ can be written analytically, but (in general) is a sum of $n$ terms of the form $ce^{\lambda t}$, $ae^{\alpha t} \cos \beta t + be^{\alpha t} \sin \beta t$.

But: if $P(x) = p^T x$, where $p$ is a right eigenvector of $A$, then analytic solution only one of above terms!

So, in general: eigenvectors of linearization seem to be a good choice (ongoing research).
Assumption: \( \dot{u} = Au, \ A \in \mathbb{R}^{n \times n} \)

Then: \( P_i(u(t)) \) can be written analytically, but (in general) is a sum of \( n \) terms of the form \( ce^{\lambda t}, \ ae^{\alpha t} \cos \beta t + be^{\alpha t} \sin \beta t \).

But: if \( P(x) = p^T x \), where \( p \) is a right eigenvector of \( A \), then analytic solution only one of above terms!

So, in general: eigenvectors of linearization seem to be a good choice (ongoing research).

Note: approximate eigenvectors suffice!
"Solving": some simple, over-approximating description of solutions of a given constraint (e.g., a box).
"Solving": some simple, over-approximating description of solutions of a given constraint (e.g., a box).

Three solvers:

- Interval Constraint Propagation
"Solving": some simple, over-approximating description of solutions of a given constraint (e.g., a box).

Three solvers:

- **Interval Constraint Propagation** (for linear ODEs we can also use explicit solutions of projection to eigenvectors, in case with purely real eigenvalues, pruning tight!)
Solving the Constraint

"Solving": some simple, over-approximating description of solutions of a given constraint (e.g., a box).

Three solvers:

- **Interval Constraint Propagation** (for linear ODEs we can also use explicit solutions of projection to eigenvectors, in case with purely real eigenvalues, pruning tight!)

- **Barrier Computation** (in each box, compute hyperplane such that solutions of constraint only on one side): joint work with Tomáš Dzetkulič, submitted.
Solving the Constraint

"Solving": some simple, over-approximating description of solutions of a given constraint (e.g., a box).

Three solvers:

- **Interval Constraint Propagation** (for linear ODEs we can also use explicit solutions of projection to eigenvectors, in case with purely real eigenvalues, pruning tight!)

- **Barrier Computation** (in each box, compute hyperplane such that solutions of constraint only on one side): joint work with Tomáš Dzetkulič, submitted.

- **Polyhedral Quantifier Elimination** (relax non-linear constraint to linear one, apply polyhedral algorithm a’la HyTech)
Implementation

http://hsolver.sourceforge.net
Implementation

http://hsolver.sourceforge.net

Currently only mean-value theorem, projection to axes.
Implementation

http://hsolver.sourceforge.net

Currently only mean-value theorem, projection to axes.

Visualization: output as graph, use arbitrary graph visualization tool:
Visualization of Abstraction: Projection

![Diagram of abstraction projection with labels "m1" and "m2" and options VAR 0 and VAR 1.]
Conclusion

Abstraction refinement based on interval constraint propagation is useful for verification of hybrid systems.
Conclusion

Abstraction refinement based on interval constraint propagation is useful for verification of hybrid systems.

Pros:
- Even in non-linear case, no problem with rounding errors.
- In some cases, the boxes resulting from one reasoning step are tight.

Cons:
- Resulting boxes, even if tight, still rough over-approximation (cf. wrapping effect)
- Currently only reasoning over pairs of boxes, but $B_1 \rightarrow B_2$ and $B_2 \rightarrow B_3$ does not necessarily imply $B_1 \rightarrow B_3$
Conclusion

Abstraction refinement based on interval constraint propagation is useful for verification of hybrid systems.

Pros:
- Even in non-linear case, no problem with rounding errors.
- In some cases, the boxes resulting from one reasoning step are tight.

Cons:
- Resulting boxes, even if tight, still rough over-approximation (cf. wrapping effect)
Conclusion

Abstraction refinement based on interval constraint propagation is useful for verification of hybrid systems.

Pros:

- Even in non-linear case, no problem with rounding errors.
- In some cases, the boxes resulting from one reasoning step are tight.

Cons:

- Resulting boxes, even if tight, still rough over-approximation (cf. wrapping effect)
- Currently only reasoning over pairs of boxes, but $B_1 \rightarrow B_2$ and $B_2 \rightarrow B_3$ does not necessarily imply $B_1 \rightarrow B_3$