Safety Verification of Hybrid Systems

Tomáš Dzetkulič
Supervisor: Dr. Stefan Ratschan

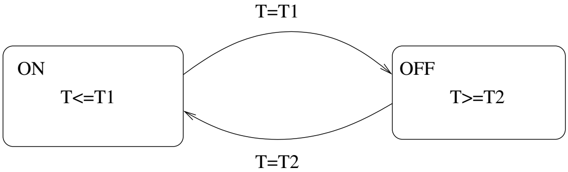
November 30, 2009

Hybrid System

- Dynamic system that exhibits both continuous and discrete behavior

Hybrid System

- Dynamic system that exhibits both continuous and discrete behavior
- Example: Thermostat



$$\dot{T}_{on} = c_1(T - T_0) + c_2$$
$$\dot{T}_{off} = c_1(T - T_0)$$

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

Definition of Hybrid System

Hybrid System: $H = (\mathit{Modes}, \mathit{Variables}, \mathit{Flow}, \mathit{Init}, \mathit{UnSafe}, \mathit{Jump})$

- Modes: Finite set of discrete modes

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables

Definition of Hybrid System

Hybrid System: $H = (\mathit{Modes}, \mathit{Variables}, \mathit{Flow}, \mathit{Init}, \mathit{UnSafe}, \mathit{Jump})$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables
- State: pair of mode and a continuous state: real valuation of all variables

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables
- State: pair of mode and a continuous state: real valuation of all variables
- Flow: Function $Modes$ x $R^n \rightarrow R^n$ assigns a vector field to each mode. The dynamics in mode m is $\dot{x} = Flow(m, x)$

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables
- State: pair of mode and a continuous state: real valuation of all variables
- Flow: Function $Modes\ x\ R^n \rightarrow R^n$ assigns a vector field to each mode. The dynamics in mode m is $\dot{x} = Flow(m, x)$
- Init: Set of admissible initial states of H

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables
- State: pair of mode and a continuous state: real valuation of all variables
- Flow: Function $Modes\ x\ R^n \rightarrow R^n$ assigns a vector field to each mode. The dynamics in mode m is $\dot{x} = Flow(m, x)$
- Init: Set of admissible initial states of H
- UnSafe: Set of states that should not be reached

Definition of Hybrid System

Hybrid System: $H = (Modes, Variables, Flow, Init, UnSafe, Jump)$

- Modes: Finite set of discrete modes
- Variables: Finite set of n continuous variables
- State: pair of mode and a continuous state: real valuation of all variables
- Flow: Function $Modes\ x\ R^n \rightarrow R^n$ assigns a vector field to each mode. The dynamics in mode m is $\dot{x} = Flow(m, x)$
- Init: Set of admissible initial states of H
- UnSafe: Set of states that should not be reached
- Jump: Relation on State pairs. If two states are in jump relation, there is possible discrete transition between them

What can we model with Hybrid Systems?

- Various traffic protocols within project AVACS `http://avacs.org`
  - Aircraft collision avoidance protocol
  - ETCS European Train Control System
- Embedded systems
  - Consumer and household products
  - Office, telecommunications devices
  - Medicine measurement and treatment devices
- Networking and locking protocols
- Physical systems with impact

Verification of Hybrid System

Evolution: Sequence of states, where two successive states are either:

- In Jump
- There is continuous trajectory given in *Flow* between these states

Verification of Hybrid System

Evolution: Sequence of states, where two successive states are either:

- In Jump
- There is continuous trajectory given in *Flow* between these states

Safety property: there is no evolution from initial state to unsafe state

Verification of Hybrid System

Evolution: Sequence of states, where two successive states are either:

- In Jump
- There is continuous trajectory given in *Flow* between these states

Safety property: there is no evolution from initial state to unsafe state

Verification: Algorithm that for each safe Hybrid System finishes in finite time and proves its safety

Verification of Hybrid System

Evolution: Sequence of states, where two successive states are either:

- In Jump
- There is continuous trajectory given in *Flow* between these states

Safety property: there is no evolution from initial state to unsafe state

Verification: Algorithm that for each safe Hybrid System finishes in finite time and proves its safety

Correctness of algorithm should not be hampered by floating-point rounding errors (reliable computing)

Basic Verification Algorithm

- We split continuous statespace into multidimensional intervals - boxes
- Box and Mode form abstract state of the Hybrid System. We explore possible transitions between abstract states. This involves solving quantified constraints.
- If there is a route from initial to unsafe abstract state, we refine abstraction by splitting one of the boxes

Basic Verification Algorithm

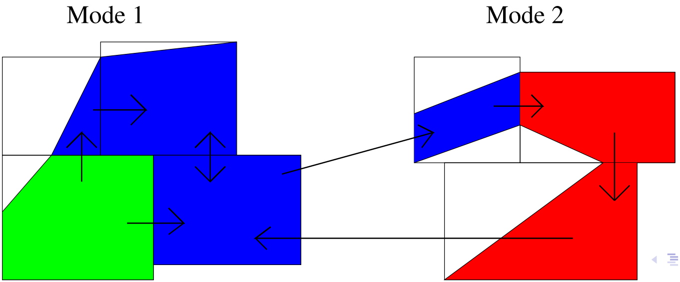
- We split continuous statespace into multidimensional intervals - boxes
- Box and Mode form abstract state of the Hybrid System. We explore possible transitions between abstract states. This involves solving quantified constraints.
- If there is a route from initial to unsafe abstract state, we refine abstraction by splitting one of the boxes

Invariant: Abstraction covers all error-trajectories in input system.

Basic Verification Algorithm

- We split continuous statespace into multidimensional intervals - boxes
- Box and Mode form abstract state of the Hybrid System. We explore possible transitions between abstract states. This involves solving quantified constraints.
- If there is a route from initial to unsafe abstract state, we refine abstraction by splitting one of the boxes

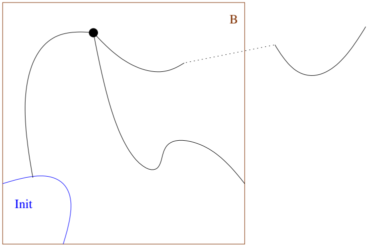
Invariant: Abstraction covers all error-trajectories in input system.



Reach Set Constraint

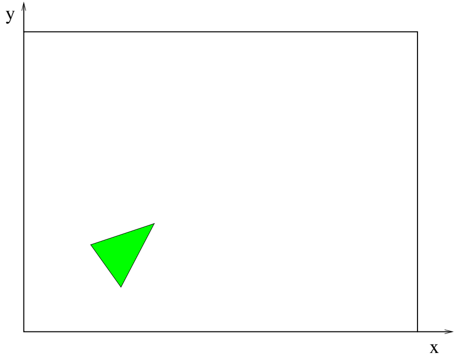
A point in a box B can be reachable

- 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



Reachability Analysis Using Constraints

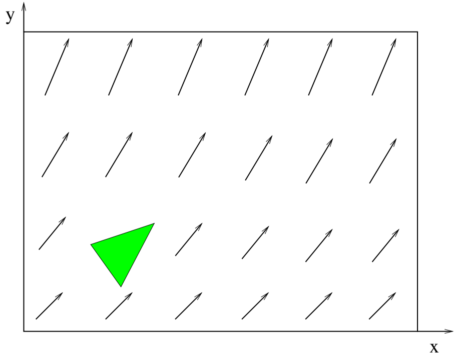
Single box and initial states



$Init(x_0, y_0)$

Reachability Analysis Using Constraints

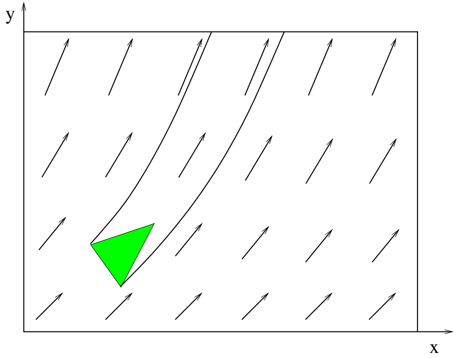
Vector field for derivatives



$Init(x_0, y_0) \wedge Flow(x_d, y_d, x_a, y_a)$

Reachability Analysis Using Constraints

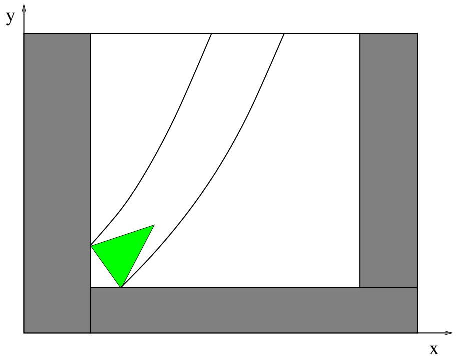
Reachable states for this example



$Init(x_0, y_0) \wedge Flow(x_d, y_d, x_a, y_a)$

Reachability Analysis Using Constraints

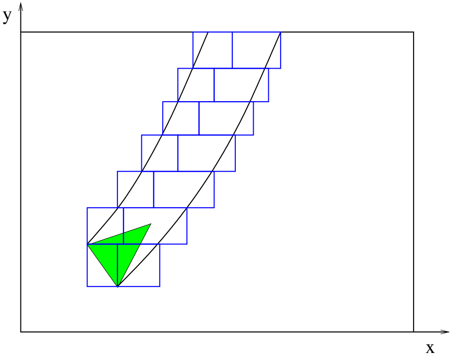
We create a constraint that overapproximates the reachable set and solve it, using interval methods



$\exists x_0, y_0, x_a, y_a, t :$
$Init(x_0, y_0) \land Flow(x_d, y_d, x_a, y_a) \land$
$x = x_0 + t * x_d \land y = y_0 + t * y_d$

Reachability Analysis Using Constraints

Splitting reduces overapproximation



$\exists x_0, y_0, x_a, y_a, t$:
$Init(x_0, y_0) \wedge Flow(x_d, y_d, x_a, y_a) \wedge$
$x = x_0 + t * x_d \wedge y = y_0 + t * y_d$

Method Properties

Advantages of this method:

- Possible non-linear `Init`, `Flow`, `Unsafe` and `Jump` constraint
- Speed of one solving step
- Simple hyper-rectangle representation
- Safe rounding

Method Properties

Advantages of this method:

- Possible non-linear `Init`, `Flow`, `Unsafe` and `Jump` constraint
- Speed of one solving step
- Simple hyper-rectangle representation
- Safe rounding

Disadvantage of the method:

- Potentially big amount of overapproximation

Tools We Use for Verification

RSolver - algorithm for solving quantified inequality constraints

- Based on constraint propagation

Tools We Use for Verification

RSolver - algorithm for solving quantified inequality constraints

- Based on constraint propagation

All computations have to be rounding-safe

- Use of interval arithmetic library

Future Work

Improving abstraction: Use polyhedrons or Taylor Models instead of boxes

Future Work

Improving abstraction: Use polyhedrons or Taylor Models instead of boxes

Improving verification algorithm

- Exact solution for certain types of ODEs
- Polyhedral Quantifier Elimination
- Reduce Wrapping Effect
- Increase degree of Taylor constraint
- SAT modulo ODE
- LP Solver on linear jumps/flows etc.

References

- T. Dzetkulič, S. Ratschan: How to Capture Hybrid Systems Evolution into Slices of Parallel Hyperplanes, ADHS 2009
- S. Ratschan and Z. She: Safety verification of hybrid systems by constraint propagation based abstraction refinement, HSCC 2005
- `http://hsolver.sourceforge.net/`