An Algorithm for Formal Safety Verification of Complex Heterogeneous Systems

Stefan Ratschan

Institute of Computer Science
Czech Academy of Sciences

Motivation

Continuous physical systems more and more
intertwined with digital systems/software
(keyword: cyber-physical systems).

Motivation

Continuous physical systems more and more intertwined with digital systems/software (keyword: cyber-physical systems).

Question: Do such systems behave correctly?

Motivation

Continuous physical systems more and more intertwined with digital systems/software (keyword: cyber-physical systems).

Question: Do such systems behave correctly?

Relevant fields:

- Modeling and simulation of (uncertain) physical systems
- Software verification:
  Software with all its complexities, for example, data structures such as integers, lists, arrays,

Motivation

Continuous physical systems more and more
intertwined with digital systems/software
(keyword: cyber-physical systems).

Question: Do such systems behave correctly?

Relevant fields:

- Modeling and simulation of (uncertain) physical systems
- Software verification:
  Software with all its complexities, for example,
  data structures such as integers, lists, arrays,

Combination?

Motivation

Continuous physical systems more and more
intertwined with digital systems/software
(keyword: cyber-physical systems).

Question: Do such systems behave correctly?

Relevant fields:

- Modeling and simulation of (uncertain) physical systems
- Software verification:
  Software with all its complexities, for example,
  data structures such as integers, lists, arrays,

Combination?

Existing combination: Hybrid systems:
Ordinary differention equations + a finite set of modes.

Motivation

Continuous physical systems more and more intertwined with digital systems/software (keyword: cyber-physical systems).

Question: Do such systems behave correctly?

Relevant fields:

- Modeling and simulation of (uncertain) physical systems
- Software verification:
  Software with all its complexities, for example, data structures such as integers, lists, arrays,

Combination?

Existing combination: Hybrid systems:
Ordinary differention equations + a finite set of modes.

Finite set often not enough for modeling software.

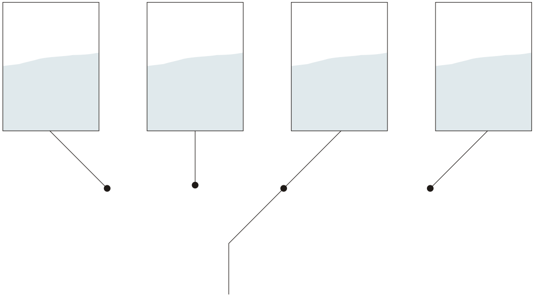
Plan for Talk

- Modeling formalism for physical/software systems encompassing ODEs and data structures
- Safety verification algorithm

Illustrating Example

Illustrating Example

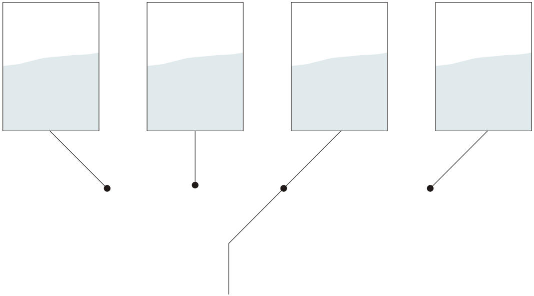
4 tanks T_1, T_2, T_3, T_4 with chemical reactions:



Tanks send cooling requests to central control unit

Illustrating Example

4 tanks T_1, T_2, T_3, T_4 with chemical reactions:

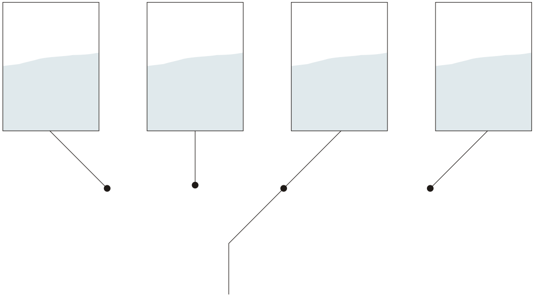


Tanks send cooling requests to central control unit

Only one cooling line available

Illustrating Example

4 tanks T_1, T_2, T_3, T_4 with chemical reactions:



Tanks send cooling requests to central control unit

Only one cooling line available

Central control unit keeps queue of cooling requests,
e.g. $< T_3, T_1 >$

Formalism

- k variables ranging over data types $D_1, \ldots, D_k$
  (e.g., queue of cooling requests)

Formalism

- k variables ranging over data types $D_1, \ldots, D_k$
  (e.g., queue of cooling requests)
- n variables ranging over the reals
  (e.g., tank temperatures)

Formalism

- k variables ranging over data types $D_1, \ldots, D_k$
  (e.g., queue of cooling requests)
- n variables ranging over the reals
  (e.g., tank temperatures)

Resulting state space: $\Phi \doteq D_1 \times \cdots \times D_k \times \mathbb{R}^n$

Formalism

- k variables ranging over data types $D_1, \ldots, D_k$
  (e.g., queue of cooling requests)
- n variables ranging over the reals
  (e.g., tank temperatures)

Resulting state space: $\Phi \doteq D_1 \times \cdots \times D_k \times \mathbb{R}^n$

Specification of system behavior:

- Continuous: Extended vector field $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$
  - Continuous time change of $\mathbb{R}^n$ according to derivative,
  - $D_1 \times \cdots \times D_k$ stay constant,
- Discrete: $\texttt{Jump} \subseteq \Phi \times \Phi$

Formalism

- k variables ranging over data types $D_1, \ldots, D_k$
  (e.g., queue of cooling requests)
- n variables ranging over the reals
  (e.g., tank temperatures)

Resulting state space: $\Phi \doteq D_1 \times \cdots \times D_k \times \mathbb{R}^n$

Specification of system behavior:

- Continuous: Extended vector field $\texttt{Flow} : \Phi \to \mathbb{R}^n$
  - Continuous time change of $\mathbb{R}^n$ according to derivative,
  - $D_1 \times \cdots \times D_k$ stay constant,
- Discrete: $\texttt{Jump} : \Phi \to \Phi$

Non-determinism:

Formalism

- k variables ranging over data types $D_1, \dots, D_k$
  (e.g., queue of cooling requests)
- n variables ranging over the reals
  (e.g., tank temperatures)

Resulting state space: $\Phi \doteq D_1 \times \cdots \times D_k \times \mathbb{R}^n$

Specification of system behavior:

- Continuous: Extended vector field $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$
  - Continuous time change of $\mathbb{R}^n$ according to derivative,
  - $D_1 \times \cdots \times D_k$ stay constant,
- Discrete: $\texttt{Jump} \subseteq \Phi \times \Phi$

Non-determinism: Relations instead of functions

Formalism: Modeling Language

Many possibilities for denoting sets `Flow`, `Jump`

Formalism: Modeling Language

Many possibilities for denoting sets `Flow`, `Jump`

For example:

- `Flow` $\subseteq \Phi \times \mathbb{R}^n$:
  ODE, differential inequalities, ODEs with interval uncertainties
- `Jump` $\subseteq \Phi \times \Phi$: computer program

Formalism: Modeling Language

Many possibilities for denoting sets `Flow`, `Jump`

For example:

- $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$:
  ODE, differential inequalities, ODEs with interval uncertainties
- $\texttt{Jump} \subseteq \Phi \times \Phi$: computer program

Specific examples:

$$[\texttt{empty}(Q) \Rightarrow \dot{x} = f(x)] \wedge [\neg\texttt{empty}(Q) \Rightarrow \dot{x} = g(x)]$$

Formalism: Modeling Language

Many possibilities for denoting sets `Flow`, `Jump`

For example:

- $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$:
  ODE, differential inequalities, ODEs with interval uncertainties
- $\texttt{Jump} \subseteq \Phi \times \Phi$: computer program

Specific examples:

$$[\texttt{empty}(Q) \Rightarrow \dot{x} = f(x)] \wedge [\neg\texttt{empty}(Q) \Rightarrow \dot{x} = g(x)]$$

$$t_1 \geq 100 \Rightarrow Q' = Q + T_1$$

System Evolution

- Continuous: Relation `Flow` $\subseteq \Phi \times \mathbb{R}^n$
  - Results in flows: functions from time to Φ ($\sim$ classical trajectories).
  - $D_1 \times \cdots \times D_k$ stay constant,

Example: Chemical reaction

System Evolution

- Continuous: Relation $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$
  - Results in flows: functions from time to Φ ($\sim$ classical trajectories).
  - $D_1 \times \cdots \times D_k$ stay constant,

  Example: Chemical reaction
- Discrete: Relation $\texttt{Jump} \subseteq \Phi \times \Phi$

  Example: Cooling request, switching of cooling system

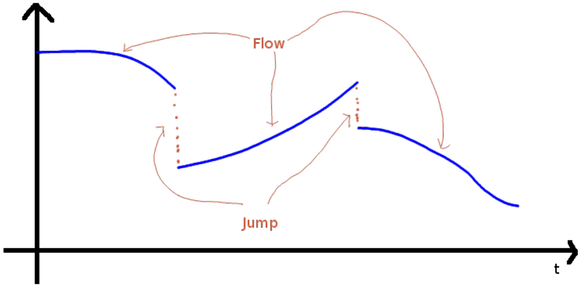
System Evolution

- Continuous: Relation $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^n$
  - Results in flows: functions from time to Φ ($\sim$ classical trajectories).
  - $D_1 \times \cdots \times D_k$ stay constant,

  Example: Chemical reaction
- Discrete: Relation $\texttt{Jump} \subseteq \Phi \times \Phi$

  Example: Cooling request, switching of cooling system

Evolution: Sequence of flows connected by jumps:



Problem: Safety Verification

Given: System +

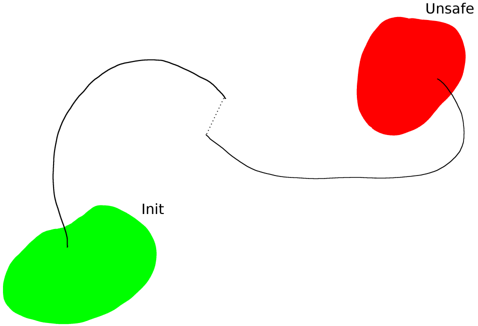
- set of states `Init` $\subseteq \Phi$ that we consider initial
- set of states `Unsafe` $\subseteq \Phi$ that we consider unsafe

Problem: Safety Verification

Given: System +

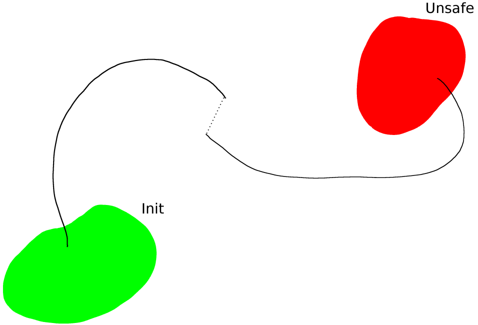
- set of states `Init` $\subseteq \Phi$ that we consider initial
- set of states `Unsafe` $\subseteq \Phi$ that we consider unsafe

Error: System evolves from a state in `Init` to a state in `Unsafe`



Problem: Safety Verification

Error: System evolves from a state in `Init` to a state in `Unsafe`

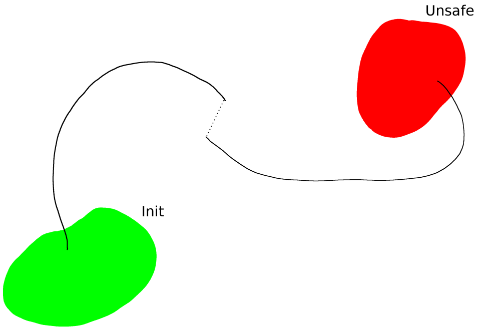


Goal: Algorithm that automatically either

- finds an error, or
- proves that none exists.

Problem: Safety Verification

Error: System evolves from a state in `Init` to a state in `Unsafe`



Goal: Algorithm that automatically either

- finds an error, or
- proves that none exists.

Rest of talk: Prove non-existence of error

Related Area: Software Model Checking

Related Area: Software Model Checking

No continuous (`Flow`), only discrete dynamics (`Jump`)

Related Area: Software Model Checking

No continuous (`Flow`), only discrete dynamics (`Jump`)

Approaches to automatically prove non-existence of errors:

- Bounded Model Checking:
  1. Fix upper bound n on time,
  2. prove that `Unsafe` not reachable in $1, 2, \ldots, n$ steps.

Related Area: Software Model Checking

No continuous (`Flow`), only discrete dynamics (`Jump`)

Approaches to automatically prove non-existence of errors:

- Bounded Model Checking:
  1. Fix upper bound n on time,
  2. prove that `Unsafe` not reachable in $1, 2, \ldots, n$ steps.
- Unbounded Model Checking:
  Prove non-existence of errors over unbounded time
  - Approach 1: Induction
  - Approach 2: Abstraction refinement

Overall Algorithm

Divide state space Φ into finitely many (non-overlapping) regions

Overall Algorithm

Divide state space Φ into finitely many (non-overlapping) regions
while over-approximation based on region not safe

Overall Algorithm

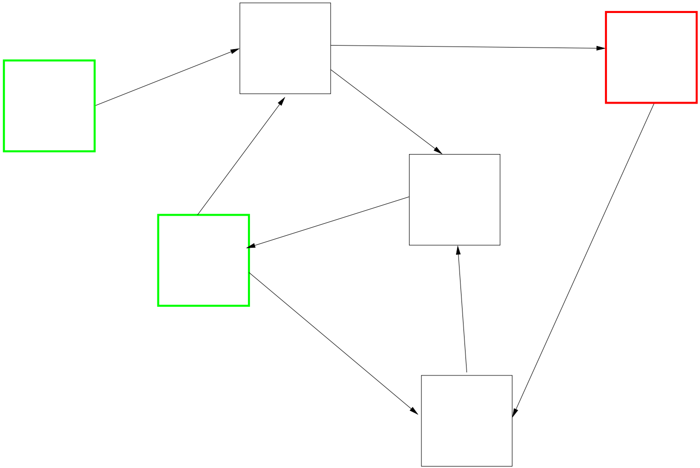
Divide state space Φ into finitely many (non-overlapping) regions
while over-approximation based on region not safe
    Refine division by further dividing regions

Overall Algorithm

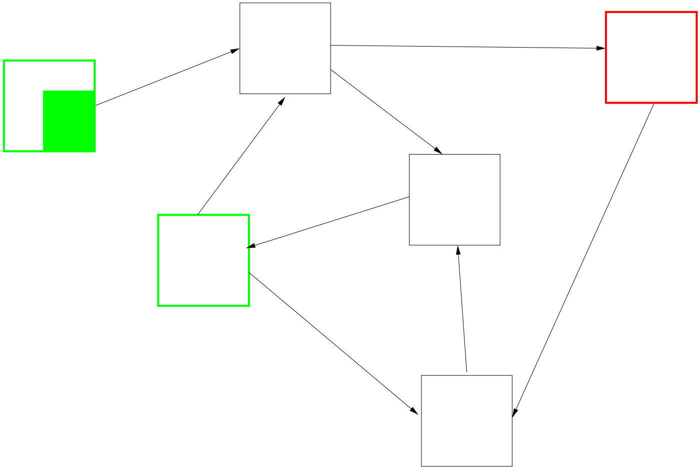
Divide state space Φ into finitely many (non-overlapping) regions
while over-approximation based on region not safe
  Refine division by further dividing regions

Abstraction:
 Set of finitely many regions
  together with (conservative) transitions

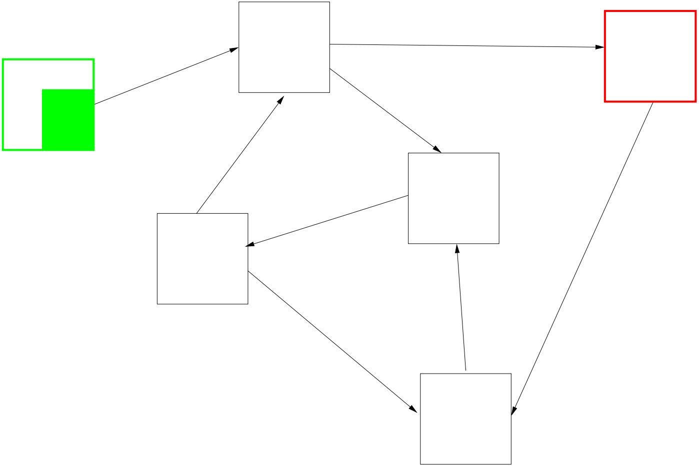
Analysis of Abstraction



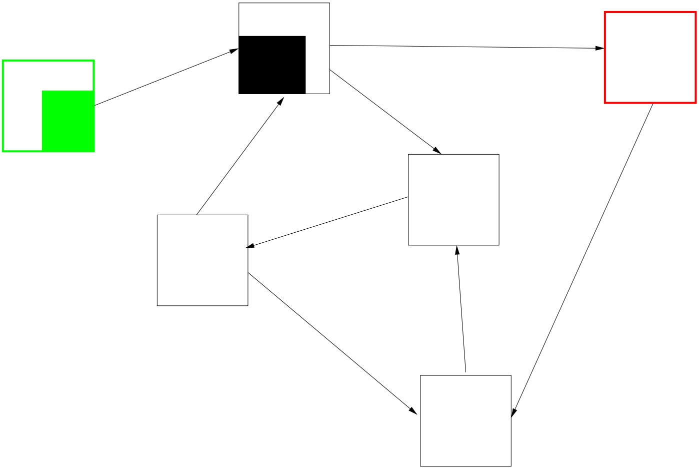
Analysis of Abstraction



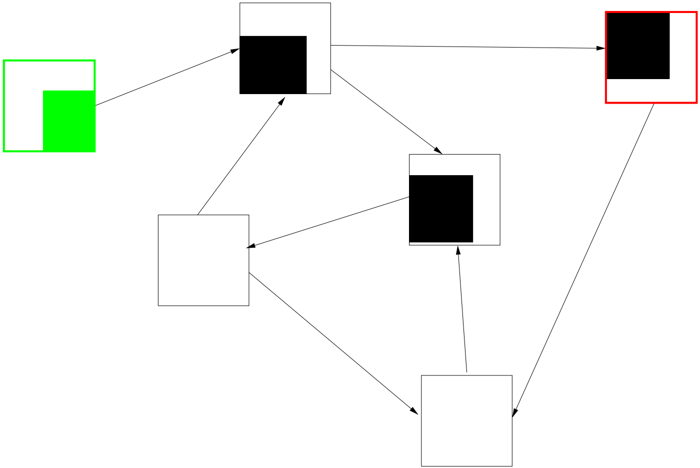
Analysis of Abstraction



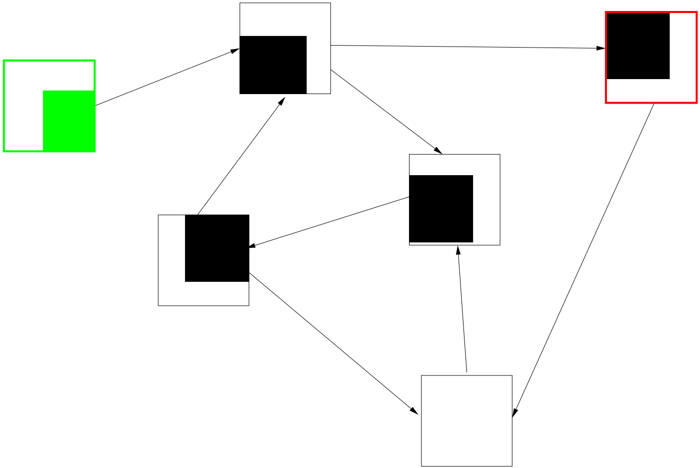
Analysis of Abstraction



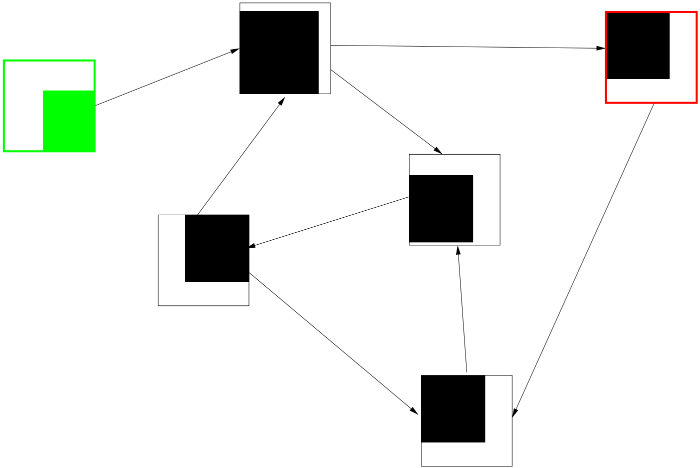
Analysis of Abstraction



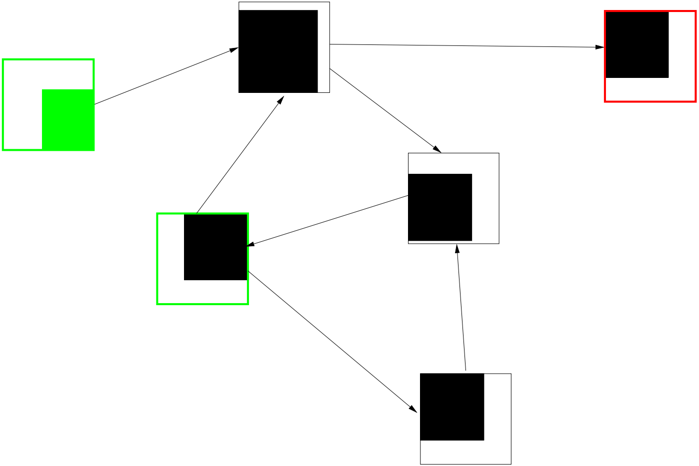
Analysis of Abstraction



Analysis of Abstraction

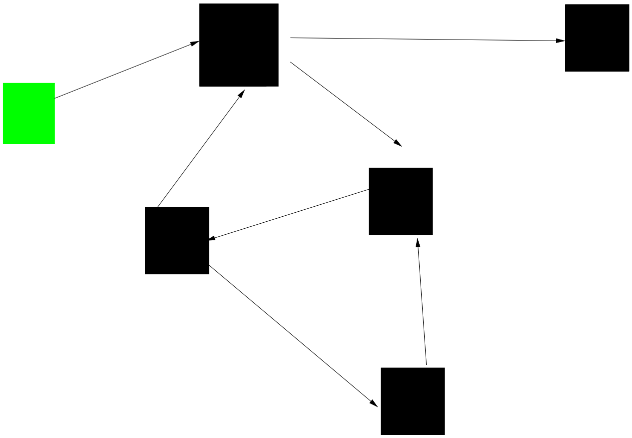


Analysis of Abstraction



remove unconfirmed transitions (initial/unsafe markings)

Analysis of Abstraction



Replace regions by new ones

Overall Algorithm

Divide state space Φ into finitely many (non-overlapping) regions
while over-approximation based on region not safe
    Refine division by further dividing regions

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.\mathtt{on}, \mathtt{off}$)

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.\mathtt{on}, \mathtt{off}$)

Continuous dynamics: Non-linear ODEs

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.\mathtt{on}, \mathtt{off}$)

Continuous dynamics: Non-linear ODEs

That is: Hybrid (dynamical) systems.

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.$ `on`, `off`)

Continuous dynamics: Non-linear ODEs

That is: Hybrid (dynamical) systems.

Algorithm: Regions: Boxes (i.e., Cartesian product of intervals)

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.\texttt{on}, \texttt{off}$)

Continuous dynamics: Non-linear ODEs

That is: Hybrid (dynamical) systems.

Algorithm: Regions: Boxes (i.e., Cartesian product of intervals)

Reachability between regions: interval methods

Algorithm Instantiation and Implementation

`http://hsolver.sourceforge.net`

State space:

- Bounded
- Finitely many discrete states ($e.g.$on, off)

Continuous dynamics: Non-linear ODEs

That is: Hybrid (dynamical) systems.

Algorithm: Regions: Boxes (i.e., Cartesian product of intervals)

Reachability between regions: interval methods

Competitive in hybrid systems area

Conclusion

Modeling formalism and safety verification algorithm
for physical/software systems

Conclusion

Modeling formalism and safety verification algorithm
for physical/software systems

Next steps:

- Parametric implementation:
  user-provided solvers for data types.
- More efficient handling of continuous evolution