Verification of Mixed Discrete-Continuous Systems

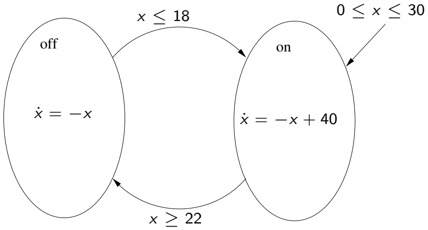
Stefan Ratschan

Institute of Computer Science
Czech Academy of Sciences

July 21, 2010

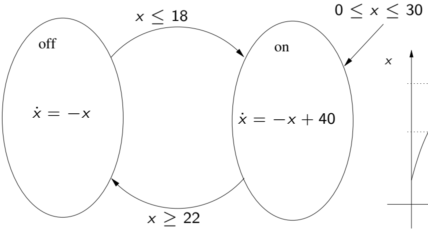
Hybrid (Dynamical) Systems

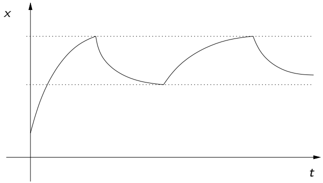
Example: Thermostat



Hybrid (Dynamical) Systems

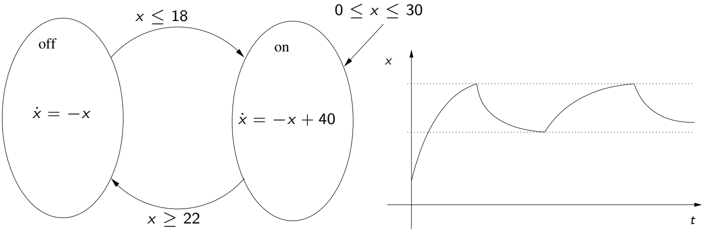
Example: Thermostat





Hybrid (Dynamical) Systems

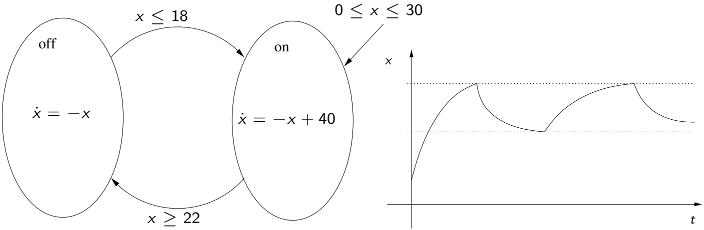
Example: Thermostat



in addition: updates

Hybrid (Dynamical) Systems

Example: Thermostat

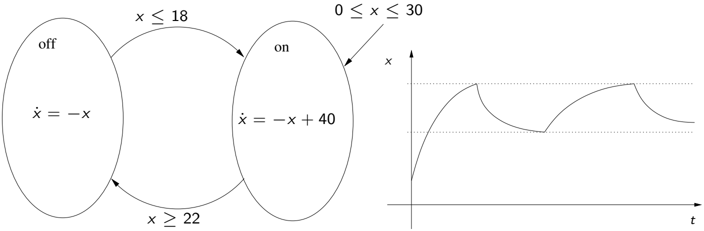


in addition: updates

General motivation: software interacting with physical environment (cyber-physical systems)

Hybrid (Dynamical) Systems

Example: Thermostat



in addition: updates

General motivation: software interacting with physical environment (cyber-physical systems)

But: software more complex than finitely many modes

Talk Outline

- Our method for safety verification of hybrid systems

Talk Outline

- Our method for safety verification of hybrid systems
- Speculation: More interesting discrete behavior?

HSolver

Tool for safety verification of hybrid systems (hsolver.sourceforge.net,[Ratschan and She, 2007])

HSolver

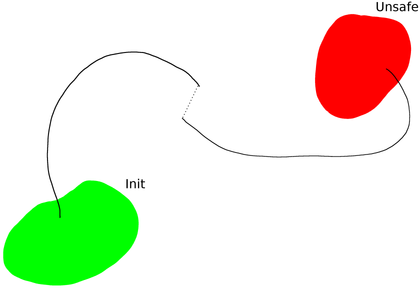
Tool for safety verification of hybrid systems (`hsolver.sourceforge.net`,[Ratschan and She, 2007])

Input:

- Hybrid system with
- set of initial states
- set of unsafe states

HSolver

Tool for safety verification of hybrid systems (`hsolver.sourceforge.net`,[Ratschan and She, 2007])



Input:

- Hybrid system with
- set of initial states
- set of unsafe states

Output:

- If terminates (printing "safe") then there is no *error trajectory* (i.e., trajectory from initial to unsafe states).
- Might run forever.

Characteristics of HSolver

Highlights:

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length
- Correctness: No incorrect result due to floating point rounding or convergence to non-global optima

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length
- Correctness: No incorrect result due to floating point rounding or convergence to non-global optima
- Scalability: Even if verification fails (e.g., for high-dimensional examples), can compute abstractions

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length
- Correctness: No incorrect result due to floating point rounding or convergence to non-global optima
- Scalability: Even if verification fails (e.g., for high-dimensional examples), can compute abstractions

Weaknesses:

- Does not yet exploit special cases (ongoing work, experimental features [Dzetkulič and Ratschan, 2009])

Characteristics of HSolver

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length
- Correctness: No incorrect result due to floating point rounding or convergence to non-global optima
- Scalability: Even if verification fails (e.g., for high-dimensional examples), can compute abstractions

Weaknesses:

- Does not yet exploit special cases (ongoing work, experimental features [Dzetkulič and Ratschan, 2009])
- Only restricted method for finding error trajectories [Ratschan and Smaus, 2009]

Characteristics of HSOLVER

Highlights:

- General inputs: non-linear differential equations/inequalitions, non-linear jumps
  (⚠ linear ODEs vs. non-linear hybrid automaton!)
- Unbounded verification: formally verifies absence of error trajectory of arbitrary length
- Correctness: No incorrect result due to floating point rounding or convergence to non-global optima
- Scalability: Even if verification fails (e.g., for high-dimensional examples), can compute abstractions

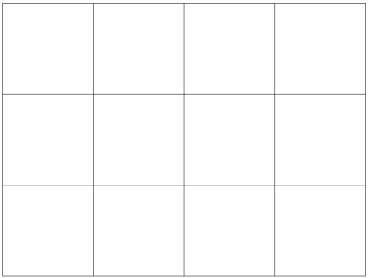
Weaknesses:

- Does not yet exploit special cases (ongoing work, experimental features [Dzetkulič and Ratschan, 2009])
- Only restricted method for finding error trajectories [Ratschan and Smaus, 2009]
- Current implementation: does not scale wrt. number of discrete modes

Abstractions in HSolver

Box grid [Kuipers, 1995]

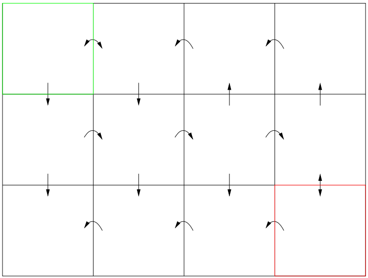
2-dimensional, 1 mode:



Abstractions in HSolver

Box grid [Kuipers, 1995]

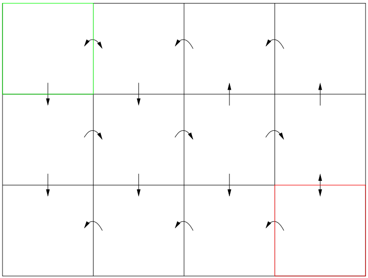
2-dimensional, 1 mode:



Abstractions in HSolver

Box grid [Kuipers, 1995]

2-dimensional, 1 mode:

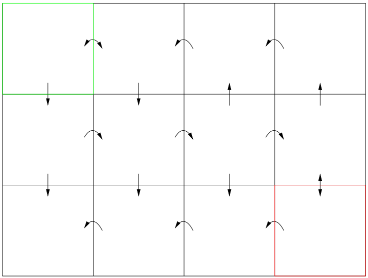


Safety of abstraction implies safety of original system

Abstractions in HSOLVER

Box grid [Kuipers, 1995]

2-dimensional, 1 mode:



Safety of abstraction implies safety of original system

refinement: split a box into two
increases abstraction size: only use as last resort!

Abstraction Pruning

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

Abstraction Pruning

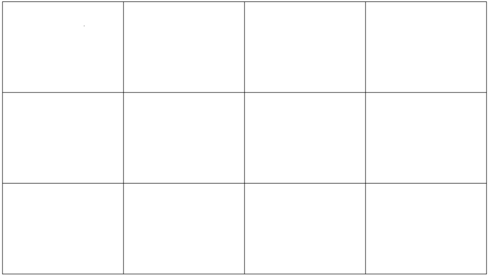
Reflect more information in abstraction,
without creating more boxes by splitting

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

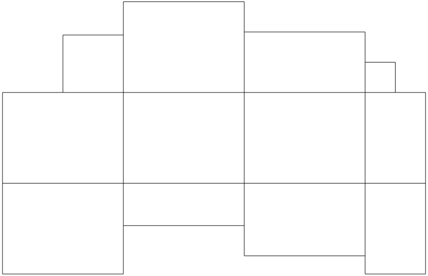
Observation: parts of state space not lying on an error trajectory not needed, remove such parts from boxes



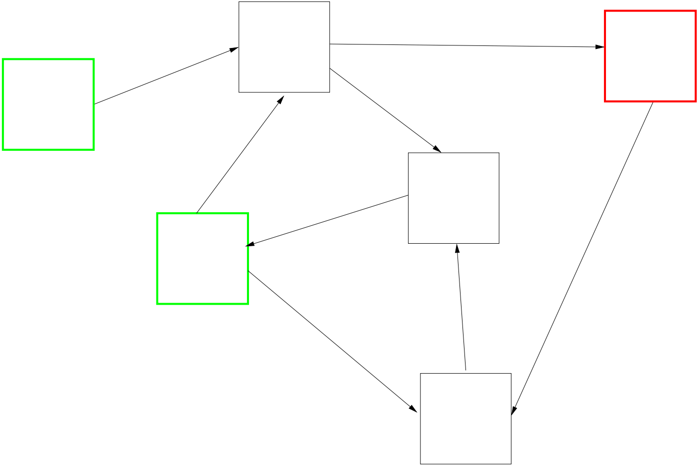
Abstraction Pruning

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

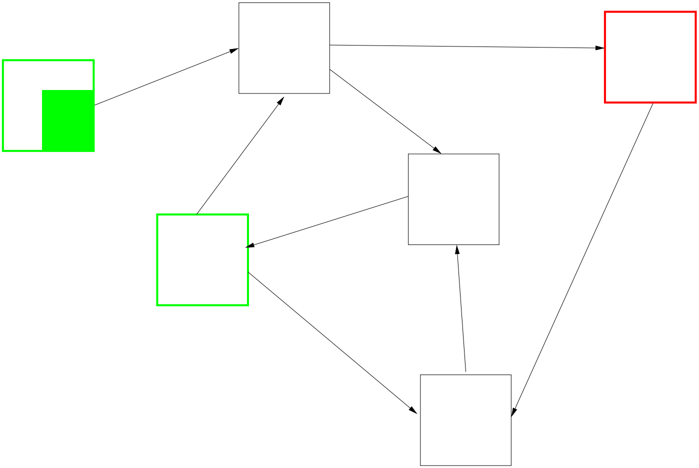
Observation: parts of state space not lying on an error trajectory not needed, remove such parts from boxes



Algorithm for Abstraction Pruning

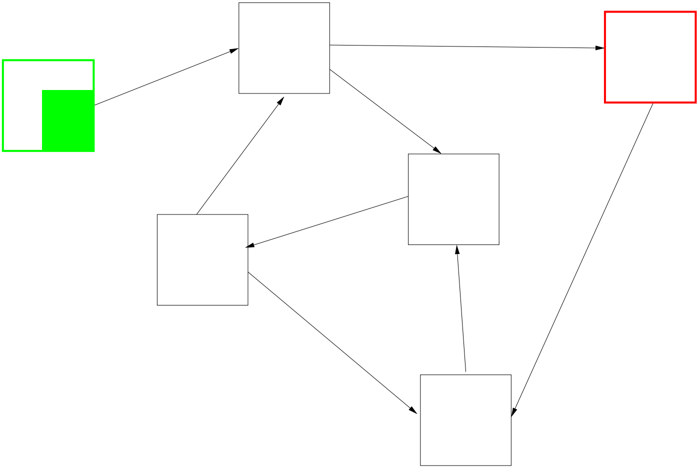


Algorithm for Abstraction Pruning



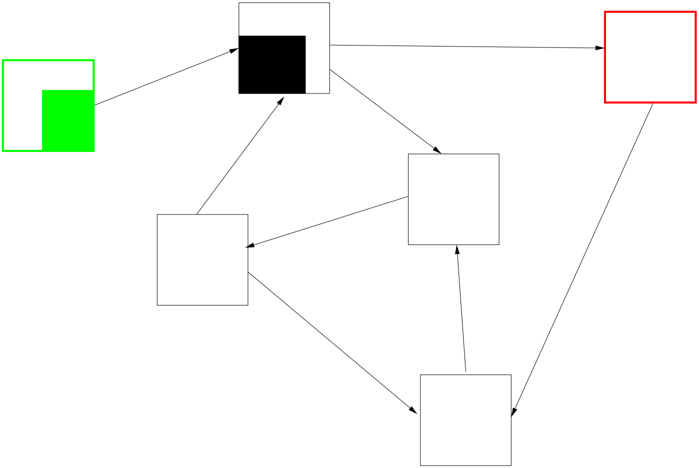
For each box marked as initial:
  over-approximate set of states reachable from an initial state

Algorithm for Abstraction Pruning



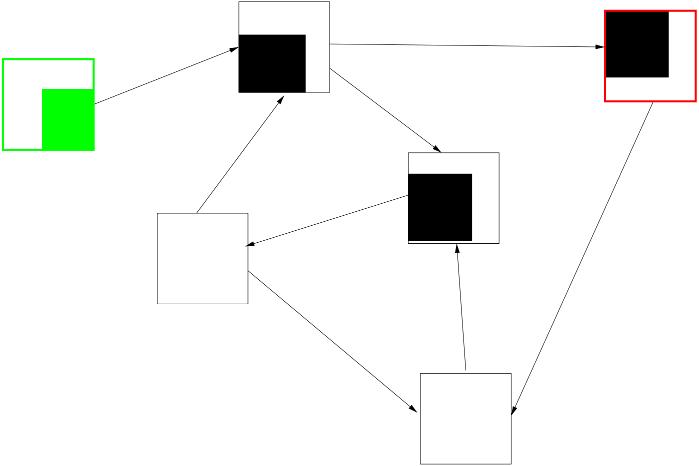
If empty set, remove initiality mark

Algorithm for Abstraction Pruning



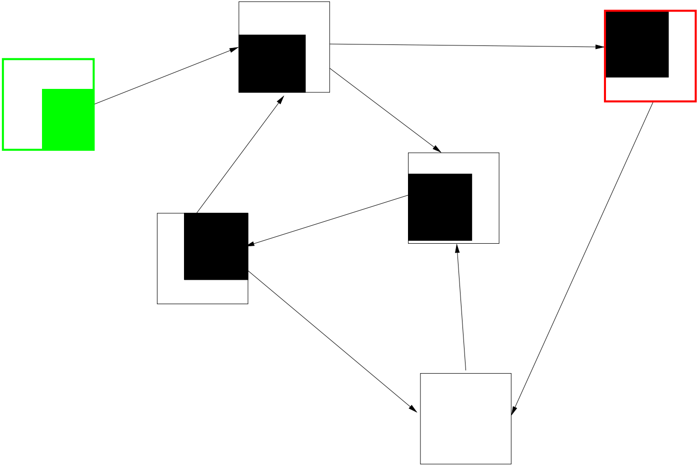
while some new states reachable through a transition,
add them

Algorithm for Abstraction Pruning



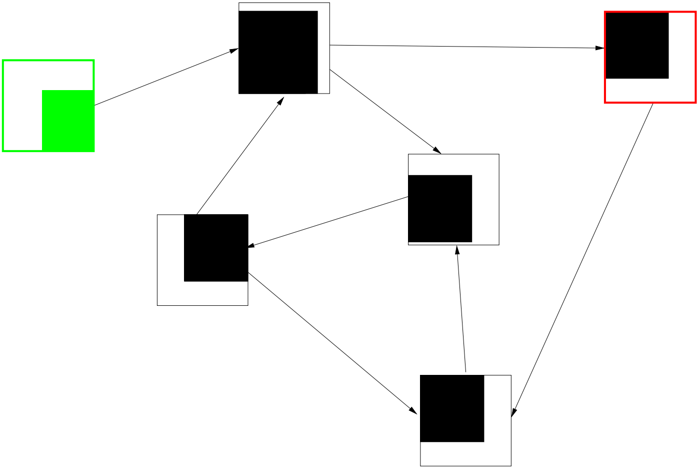
while some new states reachable through a transition,
add them

Algorithm for Abstraction Pruning



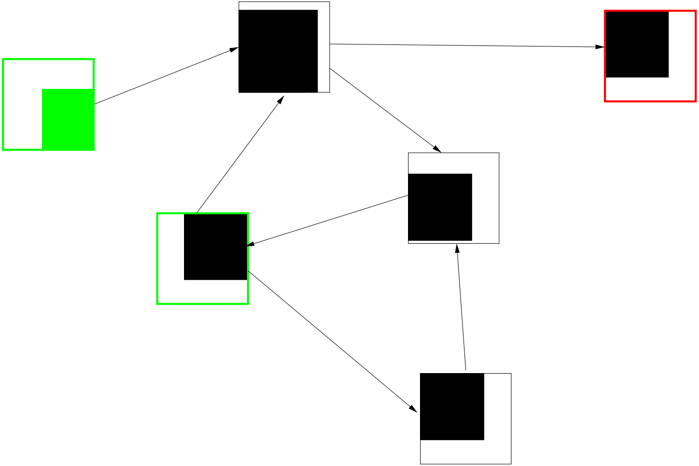
while some new states reachable through a transition,
  add them

Algorithm for Abstraction Pruning



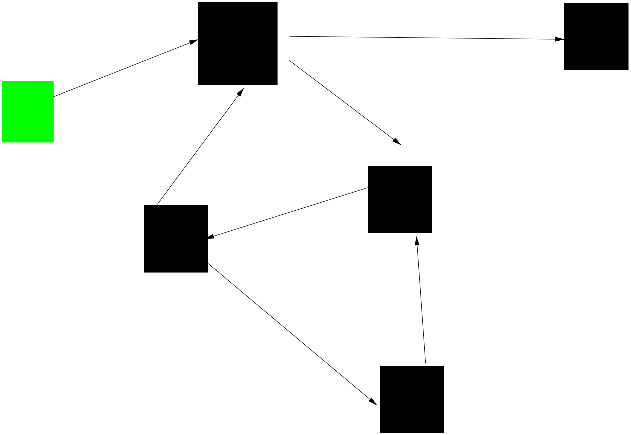
while some new states reachable through a transition,
add them

Algorithm for Abstraction Pruning



remove unconfirmed transitions

Algorithm for Abstraction Pruning



Replace boxes by new ones

Further Issues

Error trajectory: starts in initial states, leads to unsafe states

Hence: apply pruning algorithm also backward in time

Further Issues

Error trajectory: starts in initial states, leads to unsafe states

Hence: apply pruning algorithm also backward in time

Incrementality between splits, forward/backward phases

Further Issues

Error trajectory: starts in initial states, leads to unsafe states

Hence: apply pruning algorithm also backward in time

Incrementality between splits, forward/backward phases

Many possibilities concerning heuristics/widening strategies.

Further Issues

Error trajectory: starts in initial states, leads to unsafe states

Hence: apply pruning algorithm also backward in time

Incrementality between splits, forward/backward phases

Many possibilities concerning heuristics/widening strategies.

Method can be instantiated
with arbitrary reachability computation algorithm

Further Issues

Error trajectory: starts in initial states, leads to unsafe states

Hence: apply pruning algorithm also backward in time

Incrementality between splits, forward/backward phases

Many possibilities concerning heuristics/widening strategies.

Method can be instantiated
with arbitrary reachability computation algorithm

Our implementation:

- Interval constraint propagation [Benhamou and Granvilliers, 2006] +
- algebraization of ODEs [Hickey, 2000, Ratschan and She, 2007]

How to Get Rich?

How to Get Rich?

What about more interesting discrete behavior?

How to Get Rich?

What about more interesting discrete behavior?

For example:

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

Communication between program and differential (in)equations?

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

Communication between program and differential (in)equations?

Program can explicitly switch differential (in)equations

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

Communication between program and differential (in)equations?

Program can explicitly switch differential (in)equations

Program can read and write continuous variables
(abstracts from A/D conversion)

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

Communication between program and differential (in)equations?

Program can explicitly switch differential (in)equations

Program can read and write continuous variables
(abstracts from A/D conversion)

How to match discrete to continuous time?

How to Get Rich?

What about more interesting discrete behavior?

For example:
Instead of finitely many modes,
variables over various data types,
evolving according to computer program?

Communication between program and differential (in)equations?

Program can explicitly switch differential (in)equations

Program can read and write continuous variables
(abstracts from A/D conversion)

How to match discrete to continuous time?

For each program statement interval bound on duration?
(may be $[0, 0]$)

Previous Work

HybridFluctuat (Bouissou, Goubault, Putot et. al.)

Similar model for interaction software - environment.

Previous Work

HybridFluctuat (Bouissou, Goubault, Putot et. al.)

Similar model for interaction software - environment.

Verification by symbolic co-simulation

Interval based ODE integrator (i.e., good for deterministic systems, only limited non-determism)

Bounded time

Does not exploit property at hand

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

- Split(a)$= (a_1, \ldots, a_k)$ s.t. $\bigcup_{i \in \{1,\ldots,k\}} a_i = a$

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

- Split(a)$= (a_1, \ldots, a_k)$ s.t. $\bigcup_{i \in \{1,\ldots,k\}} a_i = a$
- InitReach(a) s.t. $InitReach(a) \supseteq$ $\{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \text{ initial}, \sigma_1 \in a, \ldots, \sigma_k \in a\}$

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

- Split(a)$= (a_1, \ldots, a_k)$ s.t. $\bigcup_{i \in \{1,\ldots,k\}} a_i = a$
- InitReach(a) s.t. $\mathit{InitReach}(a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \text{ initial}, \sigma_1 \in a, \ldots, \sigma_k \in a\}$
- Reach(a_{in}, a): $\mathit{Reach}(a_{in}, a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \in a_{in}, \sigma_2 \in a, \ldots, \sigma_k \in a\}$

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

- Split(a)$=(a_1, \ldots, a_k)$ s.t. $\bigcup_{i \in \{1,\ldots,k\}} a_i = a$
- InitReach(a) s.t. $\mathit{InitReach}(a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \text{ initial}, \sigma_1 \in a, \ldots, \sigma_k \in a\}$
- Reach(a_{in}, a): $\mathit{Reach}(a_{in}, a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \in a_{in}, \sigma_2 \in a, \ldots, \sigma_k \in a\}$
- $\uplus_b$ s.t. $(a_1 \cup a_2) \cap b \subseteq a_1 \uplus_b a_2 \subseteq b$

Thought Experiment: Software Analogon of Verif. Alg.

Naive translation from hybrid to software.

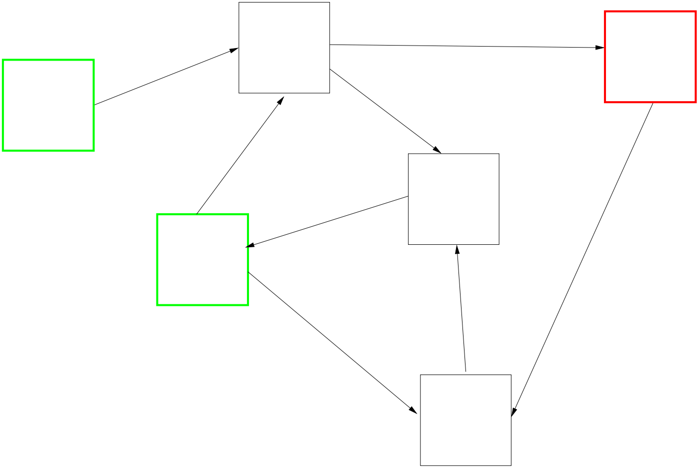
Result might be well-known, might be completely useless etc.

State space: Ω (program counter + values of variables)

Abstract states: $\mathcal{A} \subseteq 2^{\Omega}$

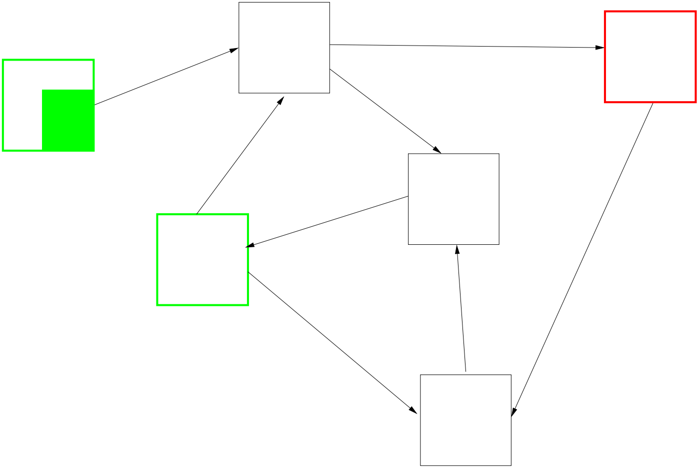
- Split(a)$= (a_1, \ldots, a_k)$ s.t. $\bigcup_{i \in \{1,\ldots,k\}} a_i = a$
- InitReach(a) s.t. $InitReach(a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \text{ initial}, \sigma_1 \in a, \ldots, \sigma_k \in a\}$
- Reach(a_{in}, a): $Reach(a_{in}, a) \supseteq \{x \in a \mid \sigma_1 \to \cdots \to \sigma_k = x, \sigma_1 \in a_{in}, \sigma_2 \in a, \ldots, \sigma_k \in a\}$
- $\uplus_b$ s.t. $(a_1 \cup a_2) \cap b \subseteq a_1 \uplus_b a_2 \subseteq b$
- $\sqsubseteq$ s.t. $a_1 \sqsubseteq a_2$ implies $a_1 \subseteq a_2$

Algorithm for Abstraction Pruning



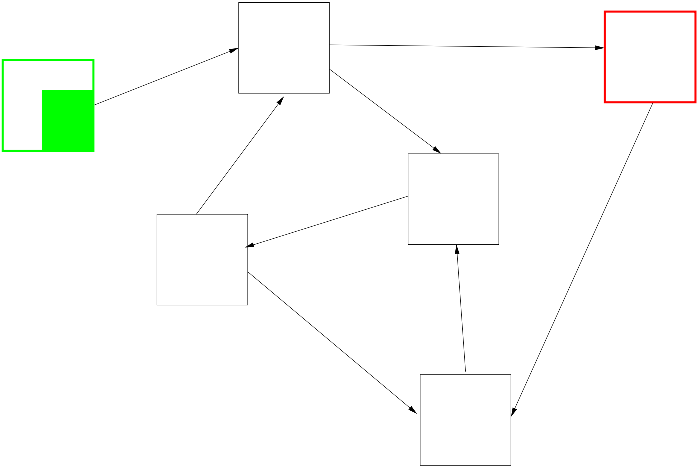
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



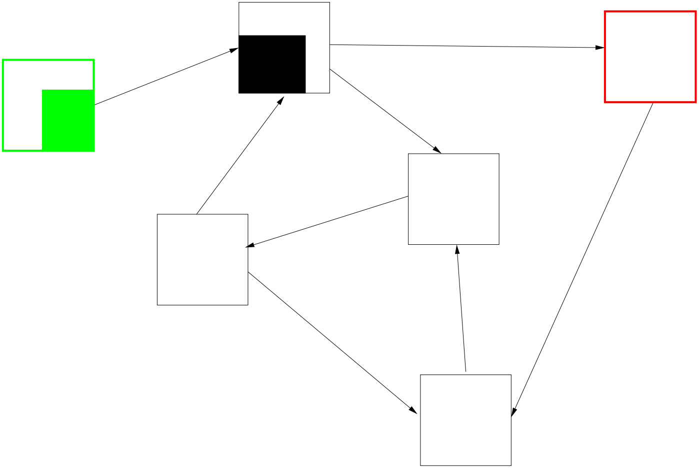
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



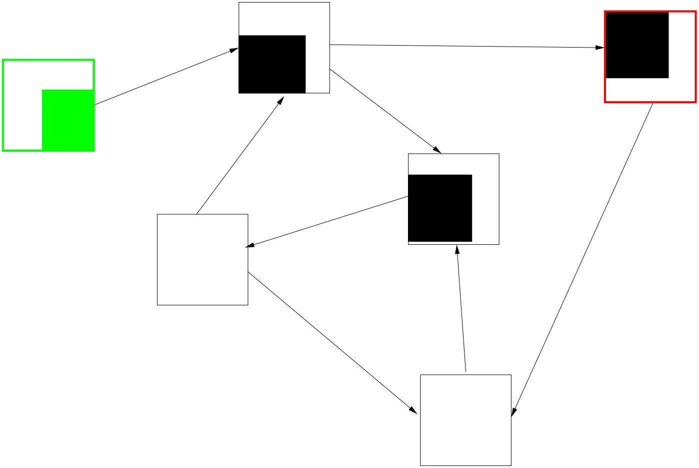
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



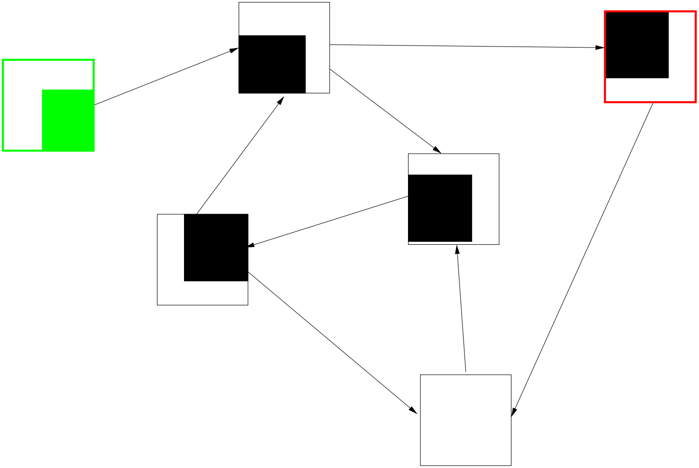
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



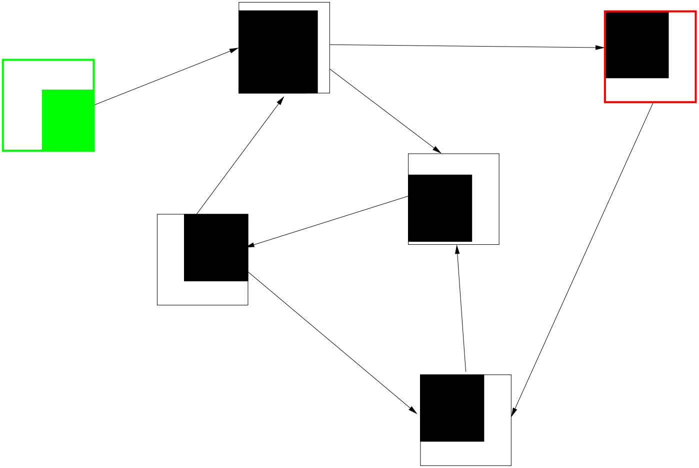
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



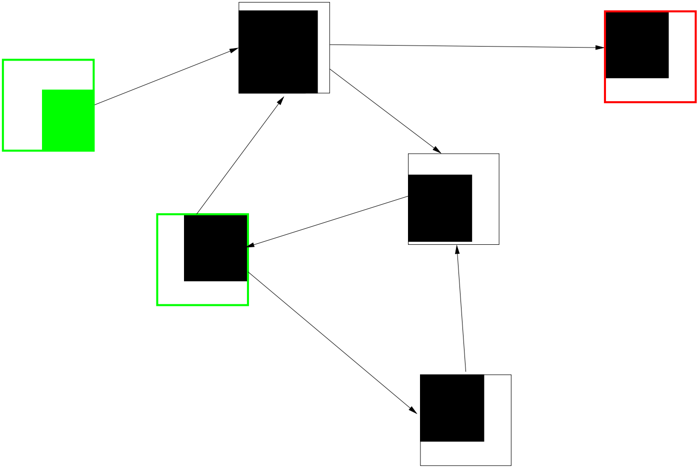
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



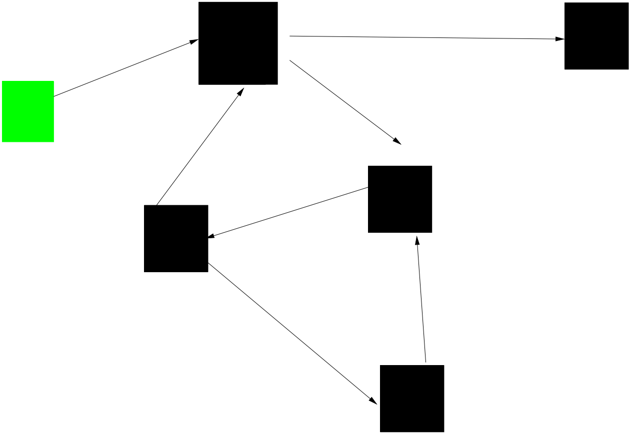
Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Algorithm for Abstraction Pruning



Boxes now represent elements of a chosen class $\mathcal{A}$ of subsets of Ω

Conclusion

Conclusion

- Combined algorithm?

Conclusion

- Combined algorithm?
- Which class of abstract states $\mathcal{A}$?

Conclusion

- Combined algorithm?
- Which class of abstract states $\mathcal{A}$?
- Is all of this anyway completely nonsense?

Literature I

F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16, pages 571–603. Elsevier, Amsterdam, 2006.

Olivier Bouissou, Eric Goubault, Sylvie Putot, Karim Tekkal, and Franck Védrine. HybridFluctuat: A static analyzer of numerical programs within a continuous environment. In *Proceedings of Computer Aided Verification CAV'09*, volume 5649 of *LNCS*, pages 620–626. Springer, 2009.

Tomáš Dzetkulič and Stefan Ratschan. How to capture hybrid systems evolution into slices of parallel hyperplanes. In *ADHS'09: 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, pages 274–279, 2009.

Literature II

Timothy J. Hickey. Analytic constraint solving and interval arithmetic. In *Proc. of the 27th ACM SIGACT-SIGPLAN Symp. on Principles of Progr. Lang.*, pages 338–351. ACM Press, 2000.

B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29: 290–338, 1995.

Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.

Stefan Ratschan and Jan-Georg Smaus. Finding errors of hybrid systems by optimising an abstraction-based quality estimate. In Catherine Dubois, editor, *Tests and Proofs*, volume 5668 of *LNCS*, pages 153–168. Springer, 2009.