

# Incremental Computation of Succinct Abstractions of Mixed Discrete-Continuous Systems

Tomáš Dzetkulič and *Stefan Ratschan*

Institute of Computer Science  
Czech Academy of Sciences

July 14, 2011

# Mixed Discrete-Continuous Systems

State space: Cross product of

- ▶ **finite** (e.g., Boolean functions, floating point)
- ▶ **infinite**, but countable (e.g., integers, dynamical data structures)
- ▶ **uncountable** (reals)

component state spaces.

# Mixed Discrete-Continuous Systems

State space: Cross product of

- ▶ **finite** (e.g., Boolean functions, floating point)
- ▶ **infinite**, but countable (e.g., integers, dynamical data structures)
- ▶ **uncountable** (reals)

component state spaces.

Corresponding **dynamics** given by

- ▶ ODEs (at least linear)
- ▶ computer programs,
- ▶ ...

# Mixed Discrete-Continuous Systems

State space: Cross product of

- ▶ **finite** (e.g., Boolean functions, floating point)
- ▶ **infinite**, but countable (e.g., integers, dynamical data structures)
- ▶ **uncountable** (reals)

component state spaces.

Corresponding **dynamics** given by

- ▶ ODEs (at least linear)
- ▶ computer programs,
- ▶ ...

Formal details: open (see e.g., Bouissou et. al.: HybridFluctuat)

# Ultimate Goal: Safety verification

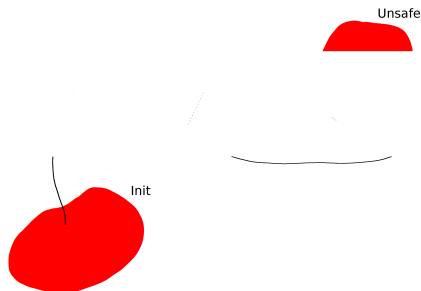
For

- ▶ **system** with
- ▶ set of **initial** states
- ▶ set of **unsafe** states

# Ultimate Goal: Safety verification

For

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



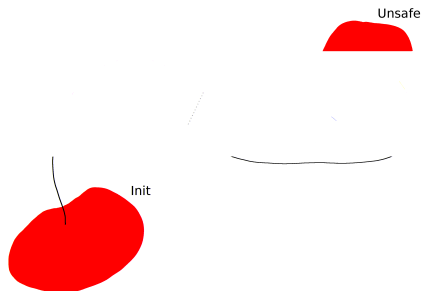
do:

- ▶ If system has an error trajectory (i.e., trajectory from initial to unsafe states), find it,

# Ultimate Goal: Safety verification

For

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



do:

- ▶ If system has an error trajectory (i.e., trajectory from initial to unsafe states), find it,
- ▶ otherwise (the system is safe), detect this.

## Observation 1

Unlike for purely discrete systems,  
for non-trivial continuous dynamics one **has to over-approximate**  
when computing reachability information



# Observation 1

Unlike for purely discrete systems,  
for non-trivial continuous dynamics one **has to over-approximate**  
when computing reachability information

It is a-priori not clear  
**how much** and **where**  
**to over-approximate** for a given safety property.

# Observation 1

Unlike for purely discrete systems,  
for non-trivial continuous dynamics one **has to over-approximate**  
when computing reachability information

It is a-priori not clear  
**how much** and **where**  
**to over-approximate** for a given safety property.

⇒:

*incrementally improve **abstraction***  
*(abstraction refinement)*



## Observation 2

Unlike for finite state systems,  
there is **no bound** on the **size** of **abstractions**  
(uncountable state space!)

For hybrid systems,  
removing single error trajectories from abstractions (**CEGAR**)  
tends to rapidly **increase** the **number of abstract states**.

## Observation 2

Unlike for finite state systems,  
there is **no bound** on the **size of abstractions**  
(uncountable state space!)

For hybrid systems,  
removing single error trajectories from abstractions (**CEGAR**)  
tends to rapidly **increase** the **number of abstract states**.

⇒:

*compute as much information as possible  
without increasing the abstraction size*

## Observation 3

The ability to handle **high-dimensional systems** is important

## Observation 3

The ability to handle **high-dimensional systems** is important

Even if **safety verification fails**,

we want at least to be able to **compute abstractions**  
(guidance for simulation based approaches)

## Observation 3

The ability to handle **high-dimensional systems** is important

Even if **safety verification fails**,

we want at least to be able to **compute abstractions**  
(guidance for simulation based approaches)

⇒:

*high-dimensional abstractions should be efficiently computable*  
(even if the refinement loop may not terminate)



# Summary of Resulting Design Principles

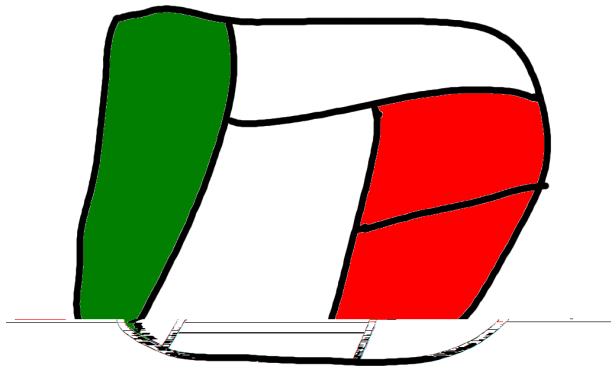
- ▶ Use some form of **abstraction refinement**
- ▶ **Compute** as much **information** as possible **without increasing** the **abstraction** size
- ▶ Compute **single abstractions** in such a way that this scales with problem dimension

# Abstractions

Based on decomposition of state space  
into finite set of **regions**

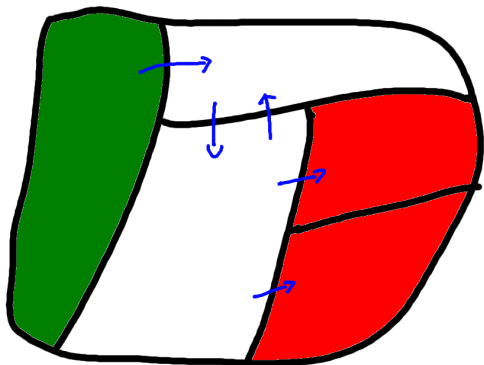
# Abstractions

Based on decomposition of state space  
into finite set of **regions**



# Abstractions

Based on decomposition of state space  
into finite set of **regions**



with **marks** and **transitions**.

# Abstraction Pruning

Reflect **more information** in abstraction,  
**without** creating **more abstract states**

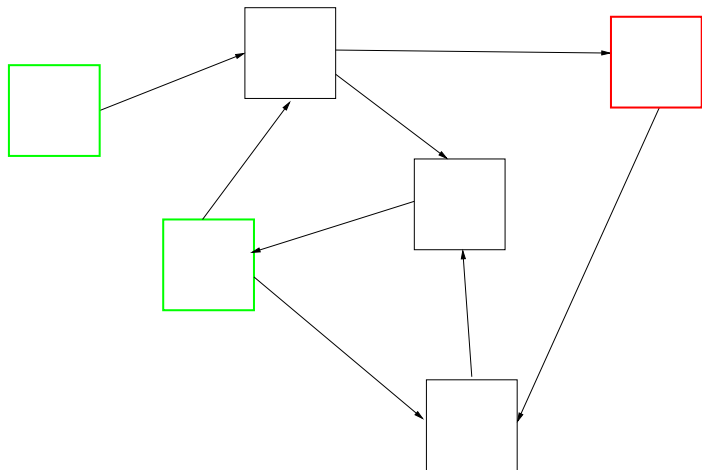
# Abstraction Pruning

Reflect **more information** in abstraction,  
**without** creating **more abstract states**

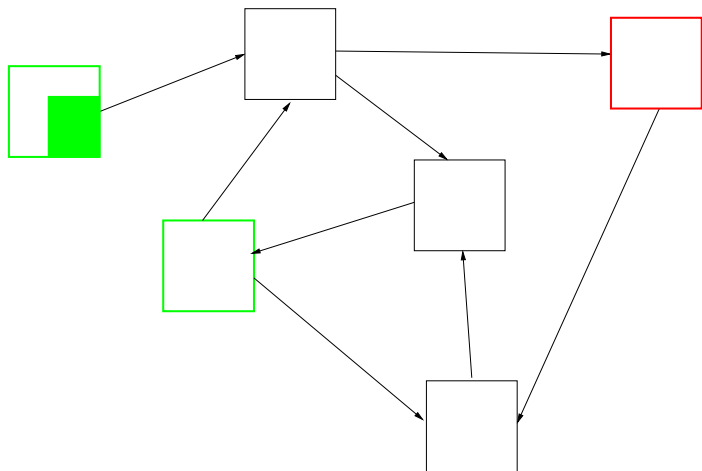
Observation:

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

## Algorithm for Abstraction Pruning



## Algorithm for Abstraction Pruning

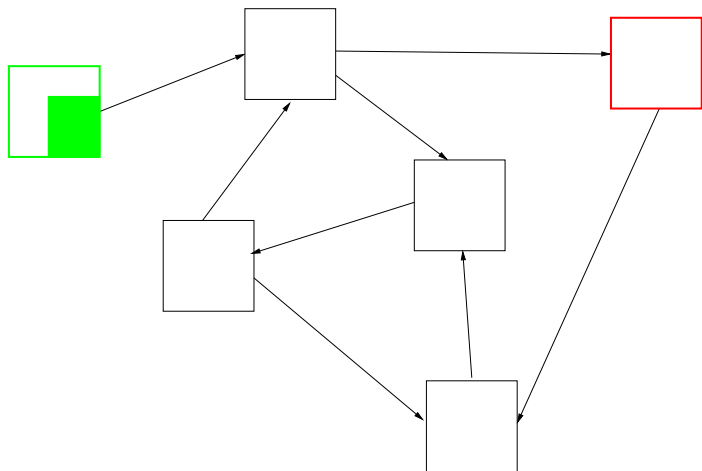


For each region marked as initial:

**over-approximate** set of states **reachable from** an **initial** state

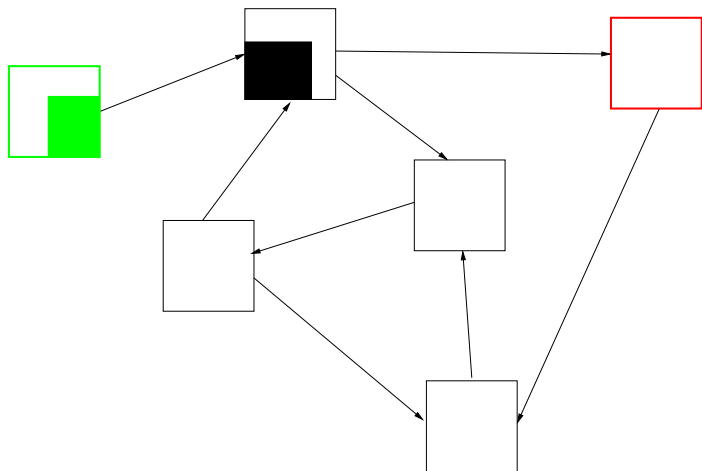


## Algorithm for Abstraction Pruning



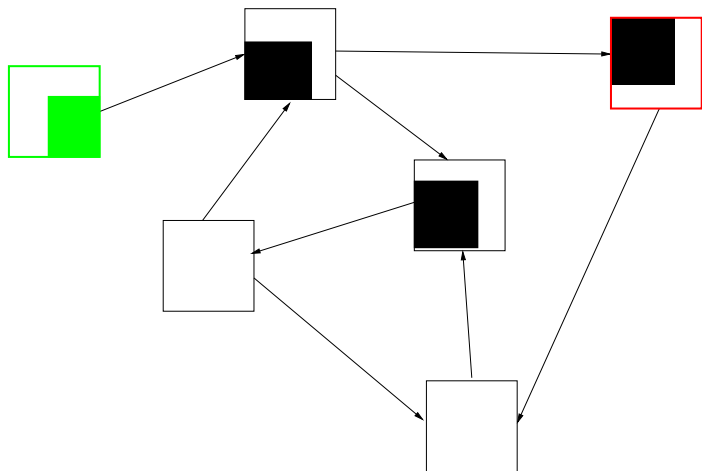
If **empty** set, **remove** initiality mark

## Algorithm for Abstraction Pruning



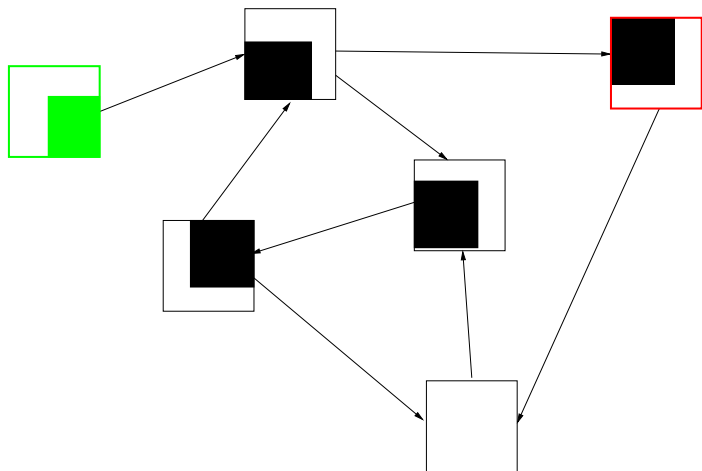
while some **new states reachable** through a transition,  
**add** them

## Algorithm for Abstraction Pruning



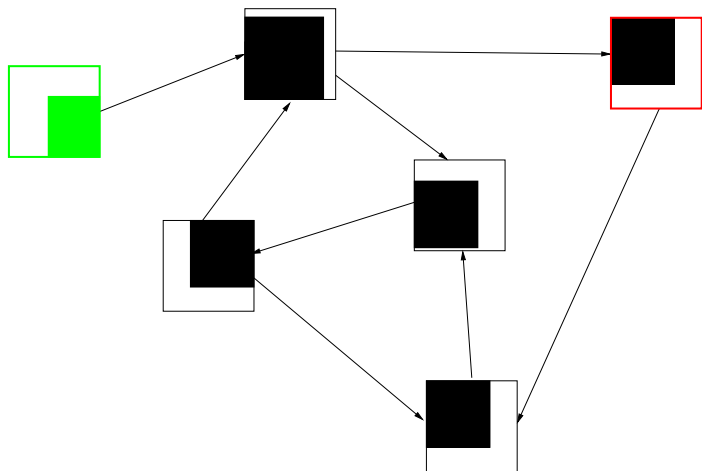
while some **new states reachable** through a transition,  
**add** them

## Algorithm for Abstraction Pruning



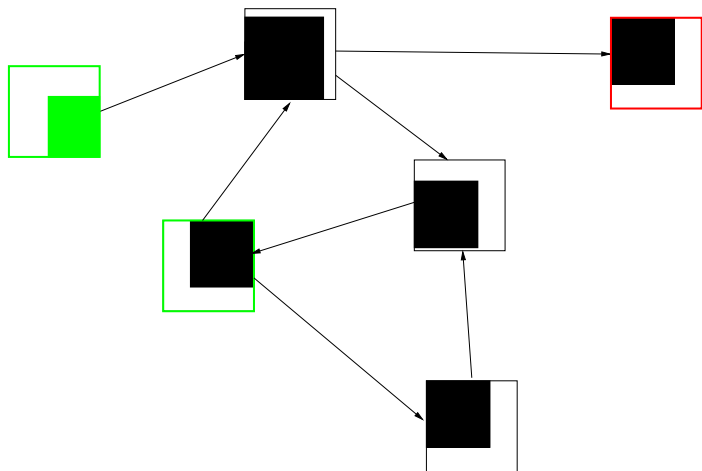
while some **new states reachable** through a transition,  
**add** them

## Algorithm for Abstraction Pruning



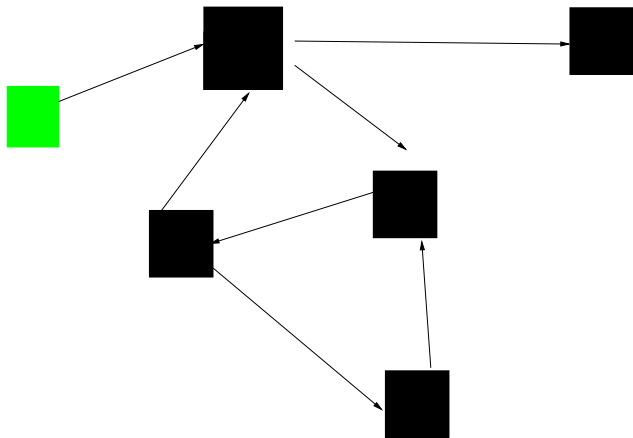
while some **new states reachable** through a transition,  
**add** them

## Algorithm for Abstraction Pruning



remove unconfirmed transitions

## Algorithm for Abstraction Pruning



Replace regions by new ones

# Discussion

Termination?



# Discussion

Termination?

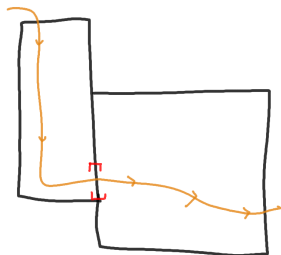
Backward in time

# Discussion

Termination?

Backward in time

Exit regions (exploit continuous time nature):



# Splitting

Only if pruning does **not** compute **relevant information** any more.

**Split** a region into into two, creating two abstract states.

# Splitting

Only if pruning does **not** compute **relevant information** any more.

**Split** a region into into two, creating two abstract states.

**Incrementality** between splits, forward/backward phases

# Splitting

Only if pruning does **not** compute **relevant information** any more.

**Split** a region into two, creating two abstract states.

**Incrementality** between splits, forward/backward phases

Method can be instantiated

with **arbitrary reachability computation** algorithm  
(it should **scale** in problem dimension).

# Splitting

Only if pruning does **not** compute **relevant information** any more.

**Split** a region into two, creating two abstract states.

**Incrementality** between splits, forward/backward phases

Method can be instantiated

with **arbitrary reachability computation** algorithm  
(it should **scale** in problem dimension).

**Experiments** with prototypical implementation  
for classical hybrid systems **promising**

# Splitting

Only if pruning does **not** compute **relevant information** any more.

**Split** a region into into two, creating two abstract states.

**Incrementality** between splits, forward/backward phases

Method can be instantiated

with **arbitrary reachability computation** algorithm  
(it should **scale** in problem dimension).

**Experiments** with prototypical implementation  
for classical hybrid systems **promising**

See our tool **HSOLVER** (<http://hsolver.sourceforge.net>)

# Conclusion

Method for **incremental computation of abstractions.**



## Conclusion

Method for **incremental computation of abstractions**.

Pruning for keeping number of abstract states **small**.

# Conclusion

Method for **incremental computation of abstractions**.

Pruning for keeping number of abstract states **small**.

**Instantiations** in various **domains**  
and with various **reachability algorithms** possible.

# Conclusion

Method for **incremental computation of abstractions**.

Pruning for keeping number of abstract states **small**.

**Instantiations** in various **domains**  
and with various **reachability algorithms** possible.

Computed abstractions can be used for

- ▶ **verification**
- ▶ **falsification/testing**

# Conclusion

Method for **incremental computation of abstractions**.

Pruning for keeping number of abstract states **small**.

**Instantiations** in various **domains**  
and with various **reachability algorithms** possible.

Computed abstractions can be used for

- ▶ **verification**
- ▶ **falsification/testing**

**Implementation** based on boxes available (open source) from

`http://hsolver.sourceforge.net`