

# Open systems and modularity

Luca Geretti

University of Verona, Italy



# Outline



- 1 Introduction
- 2 Implementation of time-varying inputs
- 3 Decomposition
- 4 Assume-guarantee reasoning
- 5 Hands-on

# Design by contract



A formal approach used to handle complex systems:

- The system is specified as a set of **components**
- Each component is annotated with **assumptions** and **guarantees** that represent a **contract**
  - ▶ An assumption can be seen as a set of conditions on the inputs to the component (i.e., the set of allowed inputs)
  - ▶ A guarantee can be seen as a set of required outputs by the component (i.e., the set of expected outputs)
- Contracts can be composed/decomposed horizontally to handle complex hierarchical systems
  - ▶ E.g., a contract on the full system can be **decomposed** into a set of simpler contracts for each component that, if **satisfied separately**, guarantee that the original contract is satisfied
- Also referred to as **assume-guarantee reasoning**

# The case for decomposition



## Motivation

Validated reachability for nonlinear systems can be effectively computed for a **small number of coupled variables**.

# The case for decomposition



## Motivation

Validated reachability for nonlinear systems can be effectively computed for a **small number of coupled variables**.

## Approach

Instead of analysing a full system, we **decouple** it into subsystems where we replace the coupling variables with proper **time-varying inputs**.

- We settle on **partial information** over the dynamics of some variables
- Inputs equivalently represent noise sources with bounded values

# Continuous vs hybrid decomposition



## Continuous case

We decompose the dynamics on a specific location of the composed hybrid system.

- We can handle a high-dimensional system in a flexible way, where optimality depends on the decomposition criterion.

# Continuous vs hybrid decomposition



## Continuous case

We decompose the dynamics on a specific location of the composed hybrid system.

- We can handle a high-dimensional system in a flexible way, where optimality depends on the decomposition criterion.

## Hybrid case

We directly decompose the hybrid system, according to the defined components.

- We can use a contract-based approach, focusing on abstraction and refinement of the components of interest.

These are separate problems whose solutions can be combined.

# Openness of components



- Unless we heavily abstract the original system, decomposition introduces some **decoupling** in the continuous space.



# Openness of components



- Unless we heavily abstract the original system, decomposition introduces some **decoupling** in the continuous space.
- Decoupled components consequently have continuous **time-varying inputs** that we must represent.

# Openness of components



- Unless we heavily abstract the original system, decomposition introduces some **decoupling** in the continuous space.
- Decoupled components consequently have continuous **time-varying inputs** that we must represent.
- These inputs should be representable by **a generic set** to cover tightly the original behaviors of the variables.

# Model the behavior of time-varying inputs



1. Option 1: use **interval coefficients** in the dynamics:
  - ▶ If we do not split the intervals, we can integrate the dynamics as usual, obtaining an overapproximation of low order.

# Model the behavior of time-varying inputs



1. Option 1: use **interval coefficients** in the dynamics:
  - ▶ If we do not split the intervals, we can integrate the dynamics as usual, obtaining an overapproximation of low order.
2. Option 2: use a **differential inclusion**:
  - ▶ Higher order approximating functions can be used to represent the behaviors of the inputs.

# Model the behavior of time-varying inputs



1. Option 1: use **interval coefficients** in the dynamics:
  - ▶ If we do not split the intervals, we can integrate the dynamics as usual, obtaining an overapproximation of low order.
2. Option 2: use a **differential inclusion**:
  - ▶ Higher order approximating functions can be used to represent the behaviors of the inputs.

In the following we discuss the less common approach of differential inclusions.

# Outline



- 1 Introduction
- 2 Implementation of time-varying inputs
- 3 Decomposition
- 4 Assume-guarantee reasoning
- 5 Hands-on

# From differential equations to differential inclusions



We use **differential inclusions** (DI) to extend differential equations to the presence of time-varying inputs.

The dynamics of the continuous state  $x \in \mathbb{R}^n$  in a DI is expressed as

$$\dot{x} \in F(x, v)$$

where  $v \in V \subset \mathbb{R}^m$  are the inputs that represent the sources of **uncertainty** in the dynamics, with  $V$  a compact bounding set.

# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we approximate each  $v_i$  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .



# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we approximate each  $v_i$  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .
- In order for the approximation to be tight, we perform it at each integration step  $k$ .
  - ▶ We call the resulting approximate solution  $y(t_k)$ .

# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we approximate each  $v_i$  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .
- In order for the approximation to be tight, we perform it at each integration step  $k$ .
  - ▶ We call the resulting approximate solution  $y(t_k)$ .
- On each integration step  $k$ :
  1. We identify proper  $w_{i,k}$ ,  $\forall i$  and compute  $y(t_k)$  from  $x(t_{k-1})$ .

# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we **approximate each  $v_i$**  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .
- In order for the approximation to be tight, we perform it **at each integration step  $k$** .
  - ▶ We call the resulting approximate solution  $y(t_k)$ .
- On each integration step  $k$ :
  1. We identify proper  $w_{i,k}$ ,  $\forall i$  and compute  $y(t_k)$  from  $x(t_{k-1})$ .
  2. We **compute an approximation error  $\varepsilon_k$**  such that
$$|x(t_k) - y(t_k)| \leq \varepsilon_k$$

# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we **approximate each  $v_i$**  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .
- In order for the approximation to be tight, we perform it **at each integration step  $k$** .
  - ▶ We call the resulting approximate solution  $y(t_k)$ .
- On each integration step  $k$ :
  1. We identify proper  $w_{i,k}$ ,  $\forall i$  and compute  $y(t_k)$  from  $x(t_{k-1})$ .
  2. We **compute an approximation error  $\varepsilon_k$**  such that
$$|x(t_k) - y(t_k)| \leq \varepsilon_k$$
  3. We **over-approximate** the exact solution with
$$\tilde{x}(t_k) = y(t_k) \pm \varepsilon_k.$$

# The general approach



- To compute the solution of  $\dot{x}(t) = F(x(t), v(t))$ , we **approximate each  $v_i$**  with a function we call  $w_i$ ,  $i = 1, \dots, m$ .
- In order for the approximation to be tight, we perform it **at each integration step  $k$** .
  - ▶ We call the resulting approximate solution  $y(t_k)$ .
- On each integration step  $k$ :
  1. We identify proper  $w_{i,k}$ ,  $\forall i$  and compute  $y(t_k)$  from  $x(t_{k-1})$ .
  2. We **compute an approximation error  $\varepsilon_k$**  such that
$$|x(t_k) - y(t_k)| \leq \varepsilon_k$$
  3. We **over-approximate** the exact solution with
$$\tilde{x}(t_k) = y(t_k) \pm \varepsilon_k.$$
  4. We use  $\tilde{x}(t_k)$  in place of  $x(t_k)$  on the  $(k + 1)$ -th step.

## Restrictions currently used in ARIADNE



In order to identify tight analytical expressions for  $\varepsilon$ , some restrictions are currently in place:

# Restrictions currently used in ARIADNE



In order to identify tight analytical expressions for  $\varepsilon$ , some restrictions are currently in place:

- We focus on **input-affine** dynamics of the form

$$\dot{x}(t) = f(x(t)) + \sum_{i=1}^m g_i(x(t))v_i(t)$$

with  $\{f, g_i\}$  non-linear.

- ▶ This represents a limitation in terms of ability to substitute variables with inputs.

# Restrictions currently used in ARIADNE



In order to identify tight analytical expressions for  $\varepsilon$ , some restrictions are currently in place:

- We focus on **input-affine** dynamics of the form

$$\dot{x}(t) = f(x(t)) + \sum_{i=1}^m g_i(x(t))v_i(t)$$

with  $\{f, g_i\}$  non-linear.

- ▶ This represents a limitation in terms of ability to substitute variables with inputs.
- We settle on a **box**  $V = \prod_i [-V_i, V_i]$ , with  $|v_i(t)| \leq V_i$ 
  - ▶ This is quite coarse when input values represent sets, as usually required by hybrid decomposition.



# The auxiliary functions $w_{i,k}$



The auxiliary system we construct relies on replacing  $v_i$  at each step  $k$  with a function  $w_{i,k}$  in a number of **parameters**  $a_{i,k}$  bounded by  $V_i$ . We currently can use the following auxiliary functions:

- 0) Zero:  $w_{i,k}(t) = 0$ ,
- 1) Constant:  $w_{i,k}(t) = a_{i,k}^{(0)}$ ,
- 2a) Affine:  $w_{i,k}(t) = a_{i,k}^{(0)} + a_{i,k}^{(1)}(t - t_{k+1/2})/h_k$ ,
- 2b) Sinusoidal:  $w_{i,k}(t) = a_{i,k}^{(0)} + a_{i,k}^{(1)} \sin(\gamma(t - t_{k+1/2})/h_k)$ ,
- 2c) Piecewise:  $w_{i,k}(t) = \begin{cases} a_{i,k}^{(0)} & \text{if } t \in [t_k, t_{k+1/2}) \\ a_{i,k}^{(1)} & \text{if } t \in [t_{k+1/2}, t_{k+1}) \end{cases}$ ,

where  $h_k = t_{k+1} - t_k$  is the integration step size and  $\gamma \approx 4.163152$ .

# The approximation error $\varepsilon_k$



- In the most general case  $\varepsilon_k$  is a function of the  $C^2$  norms of  $f$ ,  $g_i$  on the domain  $D_k$ , and of the step size  $h_k$ ;
- The error formula depends on the auxiliary functions chosen: the more the parameters used, the tighter the result;
- For two-parameter auxiliary functions, it is specialised for the cases of **additive inputs** or **single input**.

# The approximation error $\varepsilon_k$



- In the most general case  $\varepsilon_k$  is a function of the  $C^2$  norms of  $f$ ,  $g_i$  on the domain  $D_k$ , and of the step size  $h_k$ ;
- The error formula depends on the auxiliary functions chosen: the more the parameters used, the tighter the result;
- For two-parameter auxiliary functions, it is specialised for the cases of **additive inputs** or **single input**.

Order of the error based on the number of parameters:

- 0:  $O(h_k)$ ,
- 1:  $O(h_k^2)$ ,
- 2:  $\begin{cases} O(h_k^2) + O(h_k^3) & \text{but can get to } O(h_k^3), \\ O(h_k^3) & \text{if additive inputs or 1-2 inputs.} \end{cases}$

# Number of parameters to get to $O(h_k^3)$



Assumptions: system is input-affine, inputs are non-additive.

n. of inputs = $m$	n. of equations = total n. of parameters = $m(m+3)/2$	degree $d$ for at least one $w_i$ = $\lceil (m+1)/2 \rceil$
1	2	1
2	5	2
3	9	2
4	14	3
5	20	3
6	27	4
10	65	5

# The algorithm for a single continuous step



Let  $S_k = \{q_k(p) \pm e_k \mid p \in [-1, 1]^{P_k}\}$  be an over-approximation of the set  $S(x_0, t_k)$ .

1. Choose auxiliary functions  $w_{i,k}(t, a_{i,k})$ ;
2. Compute the flow  $\tilde{\phi}_k(x_k, a_k)$  of

$$\dot{x}(t) = f(x(t)) + \sum_{i=1}^m g_i(x(t)) w_{i,k}(t, a_{i,k})$$

for  $t \in [t_k, t_{k+1}]$ ,  $x_k = x(t_k) \in S_k$ ;

3. Add the uniform error bound  $\varepsilon_k$ ;
4. Compute the set  $S_{k+1}$  which approximates  $S(x_0, t_{k+1})$ .

# Implementation aspects



## Reconditioning the set

At each step the set  $S$  will increase its number of parameters

- For efficiency purposes, we need to periodically simplify the set by reducing such number at the cost of accuracy.

# Implementation aspects



## Reconditioning the set

At each step the set  $S$  will increase its number of parameters

- For efficiency purposes, we need to periodically simplify the set by reducing such number at the cost of accuracy.

## Choosing the auxiliary function dynamically

Finite accuracy in the representation of a set has an additional effect on the approximation quality

- Using a high-order  $w_{i,k}$ , while yielding a lower  $\varepsilon$ , may not result in the tightest  $S_{k+1}$
- Instead of using a fixed  $w_{i,k}$ , we dynamically evaluate each auxiliary function and choose the one giving the tightest set.

# Outline



- 1 Introduction
- 2 Implementation of time-varying inputs
- 3 Decomposition**
- 4 Assume-guarantee reasoning
- 5 Hands-on



# Decomposition in the continuous space

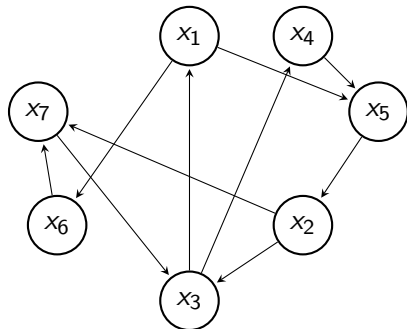
An example of a system



Source: *M. T. Laub and W. F. Loomis. A molecular network that produces spontaneous oscillations in excitable cells of dictyostelium. Molecular Biology of the Cell, 9:3521–3532, 1998.*

$$\left\{ \begin{array}{l} \dot{x}_1 = 1.4x_3 - 0.9x_1 \\ \dot{x}_2 = 2.5x_5 - 1.5x_2 \\ \dot{x}_3 = 0.6x_7 - 0.8x_2x_3 \\ \dot{x}_4 = 2 - 1.3x_3x_4 \\ \dot{x}_5 = 0.7x_1 - x_4x_5 \\ \dot{x}_6 = 0.3x_1 - 3.1x_6 \\ \dot{x}_7 = 1.8x_6 - 1.5x_2x_7 \end{array} \right.$$

Dynamics



Static dependency graph

# Decomposition in the continuous space

A partially distributed approach



On each continuous integration step  $k$ :

1. Compute the bounding box of the flow  $B_k$ 
  - ▶ Gives the bounds for an input that replaces a variable
  - ▶ May not converge, as usual

# Decomposition in the continuous space

A partially distributed approach



On each continuous integration step  $k$ :

1. Compute the bounding box of the flow  $B_k$ 
  - ▶ Gives the bounds for an input that replaces a variable
  - ▶ May not converge, as usual
2. Choose a decomposition of the variables into  $Q$  disjoint sets that minimises a chosen cost function
  - ▶ Static decomposition, valid  $\forall k$ : minimise the number of arcs removed from the dependency graph
  - ▶ Dynamic decomposition: weight each arc with the impact on the dynamics in terms of range width (using  $B_k$ ), minimize the sum of the weights

# Decomposition in the continuous space

A partially distributed approach



On each continuous integration step  $k$ :

1. Compute the bounding box of the flow  $B_k$ 
  - ▶ Gives the bounds for an input that replaces a variable
  - ▶ May not converge, as usual
2. Choose a decomposition of the variables into  $Q$  disjoint sets that minimises a chosen cost function
  - ▶ Static decomposition, valid  $\forall k$ : minimise the number of arcs removed from the dependency graph
  - ▶ Dynamic decomposition: weight each arc with the impact on the dynamics in terms of range width (using  $B_k$ ), minimize the sum of the weights
3. For each set of variables:
  - 3.1 Replace the external variables with inputs
  - 3.2 Compute the flow function

# Decomposition in the continuous space

A partially distributed approach



On each continuous integration step  $k$ :

1. Compute the bounding box of the flow  $B_k$ 
  - ▶ Gives the bounds for an input that replaces a variable
  - ▶ May not converge, as usual
2. Choose a decomposition of the variables into  $Q$  disjoint sets that minimises a chosen cost function
  - ▶ Static decomposition, valid  $\forall k$ : minimise the number of arcs removed from the dependency graph
  - ▶ Dynamic decomposition: weight each arc with the impact on the dynamics in terms of range width (using  $B_k$ ), minimize the sum of the weights
3. For each set of variables:
  - 3.1 Replace the external variables with inputs
  - 3.2 Compute the flow function
4. Combine the flow functions

# Decomposition in the continuous space

A partially distributed approach



On each continuous integration step  $k$ :

1. Compute the bounding box of the flow  $B_k$ 
  - ▶ Gives the bounds for an input that replaces a variable
  - ▶ May not converge, as usual
2. Choose a decomposition of the variables into  $Q$  disjoint sets that minimises a chosen cost function
  - ▶ Static decomposition, valid  $\forall k$ : minimise the number of arcs removed from the dependency graph
  - ▶ Dynamic decomposition: weight each arc with the impact on the dynamics in terms of range width (using  $B_k$ ), minimize the sum of the weights
3. For each set of variables:
  - 3.1 Replace the external variables with inputs
  - 3.2 Compute the flow function
4. Combine the flow functions
5. Evaluate the set at the end of the continuous step

# Decomposition in the continuous space

A totally distributed approach



## Differences

- We don't compute a global bounding box
- We introduce initial assumptions on the range of the inputs along the flow
  - ▶ If the input ranges obtained from the flow functions do not refine the assumptions, we relax those assumptions
  - ▶ Using the obtained refinement, we can subsequently refine the flow functions

# Decomposition in the continuous space

A totally distributed approach



## Differences

- We don't compute a global bounding box
- We introduce initial assumptions on the range of the inputs along the flow
  - ▶ If the input ranges obtained from the flow functions do not refine the assumptions, we relax those assumptions
  - ▶ Using the obtained refinement, we can subsequently refine the flow functions

## Issues

- Numerical convergence is slightly more delicate, since we need to guess input ranges
  - Multiple iterations to refine the input ranges is more costly
- Try this approach only if  $B_k$  cannot be obtained.



# Decomposition in the continuous space

Our example in two subsets



Static decomposition for  $Q = 2$  suggests  $\{x_1, x_4, x_5\}$  and  $\{x_2, x_3, x_6, x_7\}$  sets, yielding

$$C_1 : \begin{cases} \dot{x}_1 = 1.4v_3 - 0.9x_1 \\ \dot{x}_4 = 2 - 1.3v_3x_4 \\ \dot{x}_5 = 0.7x_1 - x_4x_5 \end{cases} \quad C_2 : \begin{cases} \dot{x}_2 = 2.5v_5 - 1.5x_2 \\ \dot{x}_3 = 0.6x_7 - 0.8x_2x_3 \\ \dot{x}_6 = 0.3v_1 - 3.1x_6 \\ \dot{x}_7 = 1.8x_6 - 1.5x_2x_7 \end{cases}$$

Hence we have  $\{n = 3, m = 1\}$  and  $\{n = 4, m = 2\}$  components.

The resulting solution will be coarser but faster to calculate.

# Decomposition in the continuous space

Our example in three subsets



Static decomposition for  $Q = 3$  suggests  $\{x_1, x_2, x_3\}$ ,  $\{x_4, x_5\}$  and  $\{x_6, x_7\}$  sets, yielding

$$C_1 : \begin{cases} \dot{x}_1 = 1.4x_3 - 0.9x_1 \\ \dot{x}_2 = 2.5v_5 - 1.5x_2 \\ \dot{x}_3 = 0.6v_7 - 0.8x_2x_3 \end{cases} \quad C_2 : \begin{cases} \dot{x}_4 = 2 - 1.3v_3x_4 \\ \dot{x}_5 = 0.7v_1 - x_4x_5 \end{cases}$$

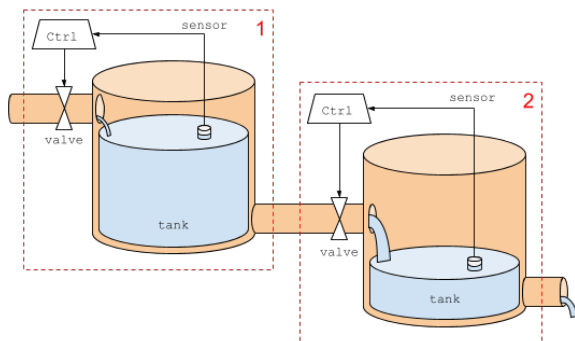
$$C_3 : \begin{cases} \dot{x}_6 = 0.3v_1 - 3.1x_6 \\ \dot{x}_7 = 1.8x_6 - 1.5v_2x_7 \end{cases}$$

Hence we have  $\{n = 3, m = 2\}$ ,  $\{n = 2, m = 2\}$  and  $\{n = 2, m = 2\}$  components.

Note that  $v_1$  is a shared parameter for  $C_2$  and  $C_3$ , hence we have only 5 actual inputs.

# Decomposition in the hybrid space

An open loop system



## Objective

Keep the water level  $x$  in each tank within a required range, using a valve aperture control  $u$ .

- a. The first component  $C_1$  can be analyzed in isolation
- b. The second component  $C_2$  can be analyzed in isolation if we replace its dependency from  $C_1$  with a time-varying input.

# Decomposition in the hybrid space

## Model details



- Each controller 1,2 switches between several operating modes/locations  $\ell_1, \ell_2$  as a function of the water level.
- For the two components and a given composed discrete location  $\ell_1 \oplus \ell_2$ , the dynamics are:

$$\dot{x}_1 = \alpha_1 u_1 - \beta_1 \sqrt{x_1}$$

$$\dot{x}_2 = \alpha_2 u_2 - \beta_2 \sqrt{x_2}$$

$$\dot{u}_1 = \psi_{1,\ell_1}(x_1)$$

$$\dot{u}_2 = \psi_{2,\ell_2}(x_2)$$

where the constants  $\alpha, \beta$  depend on the tank and pipe sections, and  $\psi$  depends on the specific controller (e.g., hysteretic, proportional) and the location/mode.

- To remove the dependency from  $x_1$ , we introduce an input  $v$  that overapproximates the behavior of  $C_1$  in respect to  $x_1$ .

# Decomposition in the hybrid space

Analyzing the decoupled component (basic approach)



1. First, we analyze  $C_1$  separately, obtaining the reachable set  $Re_1$ .
2. Then, we analyze the decoupled  $C_2$  on each location  $\ell_2$ :

$$\dot{x}_2 = \alpha_2 u_2 v - \beta_2 \sqrt{x_2}$$

$$\dot{u}_2 = \psi_{2,\ell_2}(x_2)$$

where we set  $v \in Re_1 \downarrow_{x_1}$  as an input.

By decoupling we discarded all discrete synchronisation information between  $C_1$  and  $C_2$ : hence we need to restrict  $v$  to the whole  $Re_1$ .

# Decomposition in the hybrid space

Analyzing the decoupled component (improved approach)



1. First, we analyze  $C_1$  separately, obtaining the reachable set  $Re_1$ .
2. Then, we analyze the decoupled  $C_2$ :

$$\begin{aligned}\dot{v} &= z \\ \dot{x}_2 &= \alpha_2 u_2 v - \beta_2 \sqrt{x_2} \\ \dot{u}_2 &= \psi_{2,\ell}(x_2)\end{aligned}$$

where we set  $z \in \{\dot{x}_1(x_1, u_1)\}_{(x_1, u_1) \in Re_1}$  as an input.

While  $v$  becomes an extra variable, we can set an **initial value**  $v(0) = x_1(0)$  along with an **invariant**  $v \in Re_1 \downarrow_{x_1}$ .

# Decomposition in the hybrid space

## General issues



Reachable sets might be significantly large

We need an appropriate representation for a reachable set to minimise the corresponding ranges of the introduced inputs.

→ We need to supply traces rather than time-invariant sets.

# Decomposition in the hybrid space

## General issues



### Reachable sets might be significantly large

We need an appropriate representation for a reachable set to minimise the corresponding ranges of the introduced inputs.

→ We need to supply traces rather than time-invariant sets.

### Closed loop systems require initial assumptions on inputs

This translates into successive refinements of the inputs until convergence, if numerically possible, is obtained.

→ We need to identify abstractions of components providing such inputs.



# Outline



- 1 Introduction
- 2 Implementation of time-varying inputs
- 3 Decomposition
- 4 Assume-guarantee reasoning**
- 5 Hands-on

# Assume-guarantee system specification



- The system is specified as a set of **components**.
- Every component is annotated with a pair  $(A_i, G_i)$  of **assumptions** and **guarantees**.
- The requirements  $(A, G)$  of the whole system are decomposed into a set of simpler requirements  $(A_i, G_i)$  that, if satisfied, guarantee that the overall requirements  $(A, G)$  are satisfied.

# Assume-guarantee system specification



- The system is specified as a set of **components**.
- Every component is annotated with a pair  $(A_i, G_i)$  of **assumptions** and **guarantees**.
- The requirements  $(A, G)$  of the whole system are decomposed into a set of simpler requirements  $(A_i, G_i)$  that, if satisfied, guarantee that the overall requirements  $(A, G)$  are satisfied.

We will evaluate two problems in terms of assume-guarantee:  
**safety** and **dominance**.

# Safety checking



Let  $C$  be a component of the system, annotated with assumptions  $A$  and guarantees  $G$ . We can verify whether the component  $C$  respects the safety guarantees  $G$  or not given the assumptions  $A$ .

- Represent the component  $C$  by a hybrid automaton  $H$  with inputs and outputs.
- Assumptions  $A$  are represented by a hybrid automaton  $H_A$  that specifies the possible inputs  $U$  for  $H$ .
- Guarantees  $G$  specify the possible outputs  $Y$  of automaton  $H$ .

This is a **reachability analysis** problem:

$$\text{Reach}(H \parallel H_A) \subseteq \text{Sat}(G).$$

# Safety checking by grid refinement

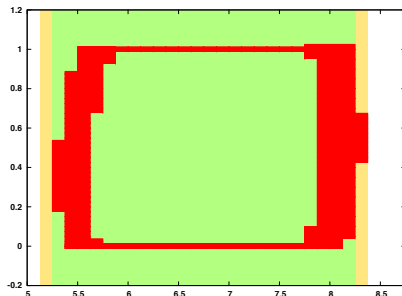


1. Compute an outer-approximation  $O$  of  $Reach(H\|H_A)$  using a grid of a given size.
2. If  $O \subseteq Sat(G)$ , the system is verified to be safe. Exit with success.
3. Otherwise, compute an  $\varepsilon$ -lower approximation  $L_\varepsilon$  of  $Reach(H\|H_A)$ . The value of  $\varepsilon$  depends on the size of the grid (typically,  $\varepsilon$  is a small multiple of the size of a grid cell).
4. If there exists at least a point in  $L_\varepsilon$  that is outside  $Sat(G)$  by more than  $\varepsilon$ , the system is verified to be unsafe. Exit with failure.
5. Otherwise, set the grid to a finer size and restart from point 1.

# Verifying the water tank



**Safety property:** the water level between 5.25 and 8.25 meters.



Green: safe set

Orange:  $\epsilon$ -tolerance

Red: computed set

First iteration:

grid  $1/8 \times 1/80$

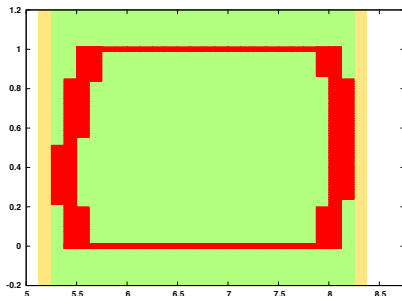
(x-axis:  $x(t)$ , y-axis:  $\alpha(t)$ ).

Outer reach is not safe, try lower reach.

# Verifying the water tank



**Safety property:** the water level between 5.25 and 8.25 meters.



Green: safe set

Orange:  $\epsilon$ -tolerance

Red: computed set

First iteration:

grid  $1/8 \times 1/80$

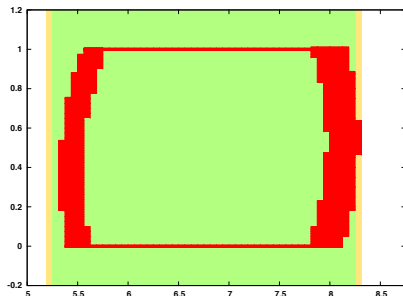
(x-axis:  $x(t)$ , y-axis:  $\alpha(t)$ ).

Lower reach is not unsafe,  
refine grid.

# Verifying the water tank



**Safety property:** the water level between 5.25 and 8.25 meters.



Second iteration:  
grid  $1/16 \times 1/160$   
(x-axis:  $x(t)$ , y-axis:  
 $\alpha(t)$ ).

Outer reach is not safe, try  
lower reach.

Green: safe set

Orange:  $\varepsilon$ -tolerance

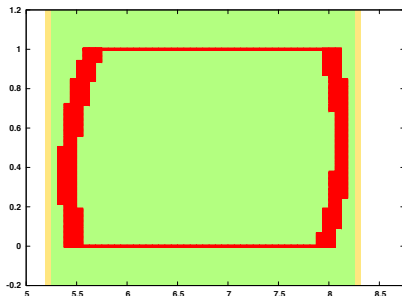
Red: computed set



# Verifying the water tank



**Safety property:** the water level between 5.25 and 8.25 meters.



Green: safe set

Orange:  $\epsilon$ -tolerance

Red: computed set

Second iteration:

grid  $1/16 \times 1/160$

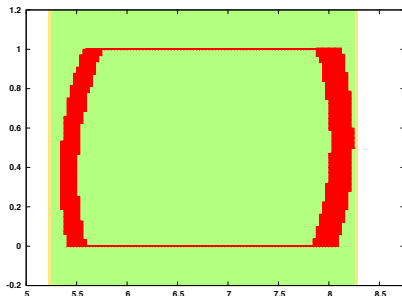
(x-axis:  $x(t)$ , y-axis:  $\alpha(t)$ ).

Lower reach is not unsafe,  
refine grid.

# Verifying the water tank



**Safety property:** the water level between 5.25 and 8.25 meters.



Third iteration:  
grid  $1/32 \times 1/320$   
(x-axis:  $x(t)$ , y-axis:  
 $\alpha(t)$ ).

Outer reach is safe, **system**  
**is proved safe.**

**Green:** safe set

**Orange:**  $\epsilon$ -tolerance

**Red:** computed set

# Verifying the water tank



1. In this example, we could prove safety by outer reach.
2. Variations of the parameters could yield systems where lower reach would prove unsafety or where no conclusions could be drawn (smallest precision of the parameters reached without proving safety or unsafety).

# Dominance checking



## Definition

Given two components  $C_1$  and  $C_2$ , with assumptions and guarantees  $(A_1, G_1)$  and  $(A_2, G_2)$ , we say that  $C_1$  **dominates**  $C_2$  if and only if under **weaker assumptions** ( $A_2 \subseteq A_1$ ), **stronger promises** are guaranteed ( $G_1 \subseteq G_2$ ).

If this is the case, the component  $C_2$  can be replaced with  $C_1$  in the system without affecting the whole system behaviour.

Intuitively, the component  $C_1$  dominates  $C_2$  if it issues sharper outputs ( $G_1 \subseteq G_2$ ) with looser inputs ( $A_2 \subseteq A_1$ ), e.g., a dominating controller can issue a subset of the control commands to cope with an environment which is allowed more freedom.

# Dominance checking by reachability analysis



1. Represent the two components by two hybrid automata  $H_1$  and  $H_2$  with inputs and outputs.
2. Assumptions  $A_1$  and  $A_2$  are represented by hybrid automata  $H_{A_1}$  and  $H_{A_2}$  that specify the possible inputs  $U_1, U_2$  for the components.
3. Guarantees  $G_1$  and  $G_2$  specify the possible outputs  $Y_1, Y_2$  of the automata  $H_1$  and  $H_2$ .
4.  $H_1$  dominates  $H_2$  if and only if  $G_1 \subseteq G_2$  and  $A_2 \subseteq A_1$ .

This is a **reachability analysis** problem:

$$Reach(H_{A_1} \parallel H_1)|_{Y_1} \subseteq Reach(H_{A_2} \parallel H_2)|_{Y_2}.$$

# Dominance checking using approximations



Approximate reachability routines can be used to test dominance of components:

1. Compute an  $\varepsilon$ -lower approximation  $L_2^\varepsilon$  of  $Reach(H_{A_2} \| H_2)|_{Y_2}$ .
2. Remove a border of size  $\varepsilon$  from  $L_2^\varepsilon$ .
3. Compute an outer approximation  $O_1$  of  $Reach(H_{A_1} \| H_1)|_{Y_1}$ .
4. If  $O_1 \subseteq L_2^\varepsilon - \varepsilon$  then  $Reach(H_{A_1} \| H_1)|_{Y_1} \subseteq Reach(H_{A_2} \| H_2)|_{Y_2}$  and thus  $H_1$  **dominates**  $H_2$ .
5. If not, we cannot say anything about  $H_1$  and  $H_2$ , and we retry with a finer approximation.

# Correctness aspects



The proof of correctness of the procedure relies on the following steps:

1.  $Reach(H_{A_1} \| H_1)|_{Y_1} \subseteq O_1$  by definition.
2.  $O_1 \subseteq L_2^\varepsilon - \varepsilon$  to be verified.
3.  $L_2^\varepsilon - \varepsilon \subseteq Inner_2$  under suitable hypotheses.
4.  $Inner_2 \subseteq Reach(H_{A_2} \| H_2)|_{Y_2}$  by definition.

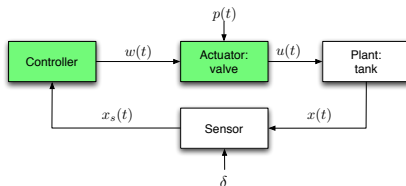
Therefore  $Reach(H_{A_1} \| H_1)|_{Y_1} \subseteq Reach(H_{A_2} \| H_2)|_{Y_2}$  and thus  $H_1$  **dominates**  $H_2$ .

A sufficient hypothesis to guarantee that  $L_2^\varepsilon - \varepsilon \subseteq Inner_2$  is that  $Reach(H_{A_2} \| H_2)|_{Y_2}$  is a  **$\varepsilon$ -regular** set, i.e., there are no holes “smaller than  $\varepsilon$ ” in the set.

# The water tank again



We want to replace the controller and the valve.



- The valve is **slower** than the previous one
- The controller is **smarter** and can fix the valve aperture to any value  $w(t) \in [0, 1]$

Does the system still operate correctly?



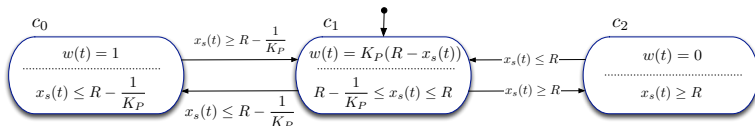
# The water tank again



Application of dominance relation in this example:

1. The automaton  $H_1$  represents the whole system with new components (proportional controller, slower valve, sensor, plant).
2. The automaton  $H_2$  represents the whole system with old components (hysteresis controller, original faster valve, sensor, plant).
3.  $A_1$  and  $A_2$  specify the same external input  $U_1 = U_2 = p(t)$ , i.e. the pressure on the valve, so it is  $A_2 = A_1$ .
4.  $G_1$  and  $G_2$  specify the same output  $Y_1 = Y_2 = x(t)$ , i.e., the water level of the tank, for which it is requested  $G_1 \subseteq G_2$ .

# A proportional controller



- The input is the measured water level  $x_s(t)$  provided by the sensor.
- The output is a command signal  $w(t) \in [0, 1]$  for the valve position regulation.
- The controller computes the output  $w(t)$  from the measured level  $x_s(t)$  and the water level reference  $R$ .
- In response to a command  $w(t)$  the valve aperture  $a(t)$  varies with the first-order linear dynamics  $\dot{a}(t) = \frac{1}{\tau}(w(t) - a(t))$ .

# A proportional controller

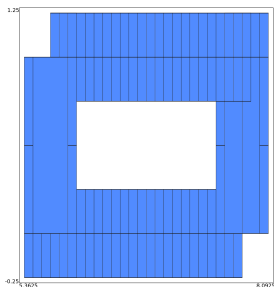


1. Location  $c_0$  models when the controller saturates the opening valve command to  $w(t) = 1$ .
2. Location  $c_1$  models when the controller tracks the water reference level  $R$ .
3. Location  $c_2$  models when the controller saturates the closing valve command to  $w(t) = 0$ .

# Results



$\varepsilon$ -lower approximation of the reachable set of the hysteresis controller:



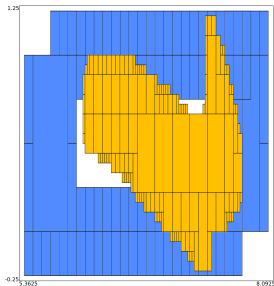
## Assumptions:

- Inlet pressure  $p$  between 50 and 60  $KPa$  (KiloPascal)
- Sensor's error between  $-0.05$  and  $0.05$   $m$

# Results



Outer approximation of the reachable set of the proportional controller:



## Assumptions:

- Inlet pressure  $p$  between 50 and 60  $KPa$  (KiloPascal)
- Sensor's error between  $-0.05$  and  $0.05$   $m$

The **proportional controller** dominates the **hysteresis controller**.

# References



## ■ Continuous decomposition

Chen, X. and Sankaranarayanan, S. "Decomposed Reachability Analysis for Nonlinear Systems," 2016 IEEE Real-Time Systems Symposium (RTSS), Porto, 2016, pp. 13-24

## ■ Differential inclusions

Gonzalez, S.; Collins, P.; Geretti, L.; Bresolin, D.; Villa, T. "Higher Order Method for Differential Inclusions", arXiv preprint arXiv:2001.11330

## ■ Assume-guarantee verification

Benvenuti, L.; Bresolin, D.; Collins, P.; Ferrari, A.; Geretti, L.; Villa, T. "Assume-guarantee verification of nonlinear hybrid systems with Ariadne", International Journal of Robust and Nonlinear Control, Volume 24, Issue 4, Mar. 2014, pg. 699-724, ISSN: 1049-8923, DOI: 10.1002/RNC.2914

# Outline



- 1 Introduction
- 2 Implementation of time-varying inputs
- 3 Decomposition
- 4 Assume-guarantee reasoning
- 5 Hands-on

# Hands-on



We focus on differential inclusions only.

- We use `examples/continuous/noisy/higgins-selkov.*pp`  
(though any example in `noisy/` would work)
  1. set `draw` to `true` in `noisy-utilities.hpp`
  2. Run with verbosity 2
  3. Run with `-v 4 -c max -g none -d none`
  4. Try with verbosity 6
  5. Enlarge the initial set
  6. (alternatively) Enlarge the input set
- Try with other examples.