A computable and compositional semantics for hybrid systems

Davide Bresolin[1] Pieter Collins[2] Luca Geretti[3]
Roberto Segala[3] Tiziano Villa[2] Sanja Živanović Gonzalez[4]

[1]University of Padova, Italy

[2]University of Maastricht, The Netherlands

[3]University of Verona, Italy

[4]AlphaStaff, USA

iFM^2 – Udine, 14 October 2024

Hybrid systems

Many real systems have a double nature:

- they evolve in a continuous fashion
- they are controlled by a discrete system



They are called hybrid systems and interact with the physical world via sensors and actuators, usually with feedback loops where physics affects computation and vice versa

What is this talk about?

- to define a mathematically precise semantics
- to describe the evolution of hybrid systems
- so that arbitrarily accurate approximations can be computed
- in a compositional way

The building blocks

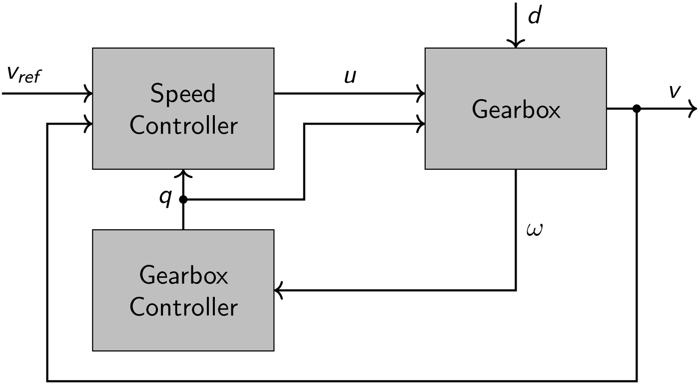
- the model of hybrid I/O automata
  - that gives a formalism to describe hybrid systems in a compositional way
- the theory of computable analysis
  - that gives the condition under which we can approximate the evolution of dynamical systems

Our contribution

We propose an automata-based formalism (HIOA) for hybrid system that

- is compositional
  - replacing a component with another one with the same I/O behaviour does not change the I/O behaviour of the composition
- is computable
  - if we can approximate the evolution of components, then we can approximate the evolution of the composition

Automatic Gearbox Example



reference speed v_{ref} road slope d gear q

rotational speed ω torque u speed v

Hybrid I/O Automata

Hybrid input/output automaton $H_{\mathrm{i/o}} = (I, O, U, Y, R_{\mathrm{i/o}}, F_{\mathrm{i/o}})$:

- I, O are input and output events. $A = I \cup O$;
- U, Y are input and output variables. $V = U \cup Y$;
- The discrete input/output behaviour is a multifunction

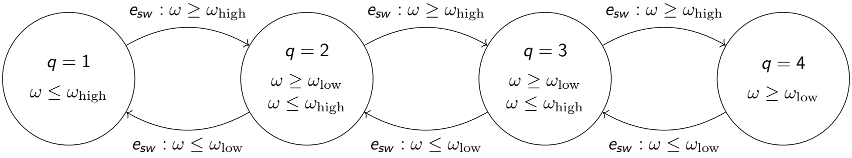
$$R_{\mathrm{i/o}} : \mathrm{Val}(V) \times (A \cup \{\tau\}) \times \mathrm{Val}(U) \rightrightarrows \mathrm{Val}(Y);$$

- The continuous input/output behaviour is a multifunction

$$F_{\mathrm{i/o}} : \mathrm{Val}(Y) \times \mathrm{CTrajs}(U) \rightrightarrows \mathrm{CTrajs}(Y)$$

respecting some natural prefix- and suffix- closure conditions.

Gearbox controller automaton



- Input variables: ω
- Output variables: q
- Input events: τ (not shown)
- Output events: e_{sw} (gear switch)

Gearbox automaton

Purely continuous component:

$$\dot{v}(t) = \frac{p_{\mathrm{r}}(q)u(t)}{m} - \frac{c}{m}v(t)^2 - g\sin(d(t)); \qquad v(0) = 0;$$
$$\omega(t) = p_{\mathrm{r}}(q)v(t);$$

(gear ratio ${}_{\mathrm{r}}(q)$, mass m, friction coefficient c, grav. accel. g)

- Input variables: q, u, d
- Output variables: v, ω
- Input events: τ (not shown)
- Output events: $\emptyset$

Speed controller automaton

PI-controller with control law:

$$u(t) = k_{\mathrm{r}}(q)(v_{\mathrm{ref}}(t) - v(t)) + u_{\mathrm{I}}(t)$$

$$\dot{u}_{\mathrm{I}}(t) = \frac{k_{\mathrm{r}}(q)}{t_{\mathrm{I}}}(v_{\mathrm{ref}}(t) - v(t)) \qquad u_{\mathrm{I}}(0) = 0;$$

(t_{I}: integration time constant, $k_{\mathrm{r}}(q)$: controller gain)

- Input variables: q, v, v_{ref}
- Output variables: u, u_I
- Input events: τ (not shown)
  e_{ref}: cruise speed is changed; resets u_I to 0
- Output events: $\emptyset$

Circular dependencies

- Key Challenge: Circular dependencies in hybrid systems
- Problem: Circular dependencies lead to non-converging loops, preventing system composition
- Solution Overview: Using interfaces and atomic components to resolve dependency issues

Introducing Interfaces

- Interfaces: Explicitly describe dependencies between variables
  - await relation $\succ$ between input and output variables
  - $y \succ x$: x is necessary to compute y
- Key Insight: Enforcing compatibility to prevent circular dependencies between components
- Benefit: Ensures computability of both discrete and continuous dynamics in composed systems

Atomic Interfaces

- Atomic Interfaces: Break down complex interfaces into simpler components
- Unidirectional Components: All output variables depend on all input variables
- Advantages: Simplifies dependency checks and ensures well-structured, computationally feasible compositions

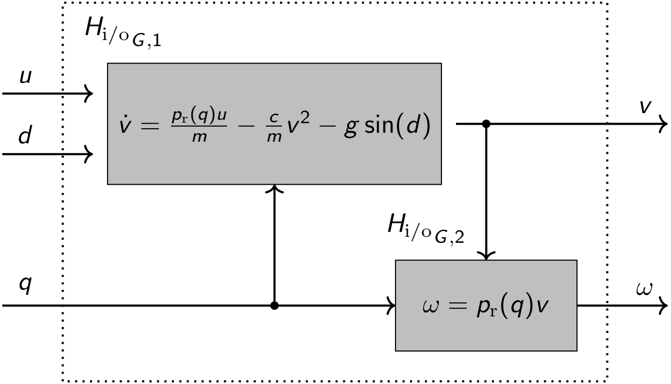
Composition and Dependent Product

- Definition of Compatibility: Two atomic components can be composed if no circular dependencies are introduced
- Dependent Product: Composition rule where outputs of one component serve as inputs to another
- Key Insight: Behavior of composed systems is computed step-by-step using the dependent product

Atomic Decomposition of Automata

- Atomic Decomposition: A hybrid automaton can be decomposed into smaller atomic components
- Result: The behavior of any complex automaton can be computed using dependent products of atomic decompositions
- Union Property: The atomic decompositions of two automata combine to form a decomposition of their composition

Decomposing the Gearbox



Handling Dependencies in Hybrid Systems

- Dependency Definition: A variable does not depend on others if changes in those variables don't affect it
- Challenge in Trajectories: Time dependencies make it complex to handle past, present, and future
- Distinction:
  - Algebraic Dependency: $x = f(y)$ (Deals with present)
  - Integral Dependency: $\dot{x} = f(x, y)$ (Deals with future)

Managing Algebraic and Integral Dependencies

- Circular Dependency Resolution: Only impose constraints on algebraic dependencies
- Integral Dependencies: Governed by differential equations, common in control theory
- Solution: Use fixpoint operators to compute dependent products under integral dependencies

Decomposing the Automatic Gearbox

Speed controller:

- u has algebraic dependence on v, v_{ref}

Gearbox Controller:

- no dependencies, q changes only at discrete events

Gearbox:

- ω has algebraic dependence on q
- v has integral dependence on u, d, q

No dependency loops, components are compatible

Computable analysis

- In computable analysis, computation is performed on infinite streams of data
- data streams encode a sequence of approximations to some quantity
- A function or operator is computable if:
  - from data streams encoding the inputs
  - it is possible to calculate a data streams encoding the outputs
- Finite computations are obtained by terminating when an accuracy criterion is satisfied.
- Computable analysis allows to determine whether a certain problem can be approximated to any accuracy

Example: testing a guard

- Consider the problem of testing whether a guard $p(x) \geq 0$ is true or not
- If x is a rational number, $p(x)$ can be computed exactly and the problem is solvable
- If the input is a sequence of approximations that converges to x, the problem becomes semi-decidable:
  - when $p(x) < 0$ or $p(x) > 0$, we can find an approximation $\tilde{x}$ of x that proves that the guard is false/true
  - when $p(x) = 0$, no matter how accurate $\tilde{x}$ is, we cannot give a definite answer

A fundamental theorem

Theorem

Only continuous functions and operators can be computable

- discontinuous functions and operators are uncomputable
- computability depends on the choice of representation and on the corresponding topology
- "naturally-defined" continuous operators are usually computable

Computability of Hybrid Systems

Theorem (P. Collins, 2011)

The finite-time evolution of a hybrid system is uncomputable.

Can we recover computability?

- By imposing restrictions on dynamics, reset functions, guards and invariants we can regularize the evolution to make it approximable either from above or from below

. . . however . . .

- the conditions for approximation of the reachable set from above are different from the ones for approximation from below
- we can only obtain a semi-decidable problem

Conditions for Computability

- Compactness: Functions are bounded and can be managed with finite computational resources
- Overtness: Logical dual of compactness
- Lipschitz continuity: Functions have a bounded rate of change
- Linear growth: Function growth is limited by a linear function
- Upper Semicontinuity: Prevents downward jumps in the function, but allows upward jumps
- Lower Semicontinuity: Prevents upward jumps in the function, but allows downward jumps

Combining the Conditions

- Key Insight: By focusing on specific combinations of these function classes, we can ensure:
  - Theoretical soundness
  - Computational tractability
- Outcome: These combinations identifies when the system can be approximates “from above” or “from below”

Main Theorem

Theorem

The composition $H_{\mathrm{i/o}_1} \parallel H_{\mathrm{i/o}_2}$ is computable for the following classes of systems:

1. *The input/output maps $R_{\mathrm{i/o}}, F_{\mathrm{i/o}}$ are upper-semicontinuous compact-valued, and $F_{\mathrm{i/o}}$ is an effectively compact operator with linear growth.*
2. *$R_{\mathrm{i/o}}, F_{\mathrm{i/o}}$ are lower-semicontinuous overt-valued, and $F_{\mathrm{i/o}}$ is effectively locally Lipschitzian.*
3. *$R_{\mathrm{i/o}}, F_{\mathrm{i/o}}$ are continuous single-valued over a closed (respectively, open) set, and $F_{\mathrm{i/o}}$ is effectively locally Lipschitzian with linear growth.*

Tool support

- The compositional framework has been implemented in ARIADNE
- The current version allows to model systems as HIOA
- The composed system must be closed in order to compute its evolution
- Composition is performed on the fly
- Work is under way on the modeling of inputs using differential inclusions

Conclusions and future work

- We provide a theory for computable composition of HIOA (missing in the literature)
- The theory is a prerequisite to develop a library that guarantees sound approximations

- It is our goal to extend the results to more general classes of Hybrid I/O Automata
- Implementation of reachability analysis routines for hybrid open systems is under way in ARIADNE

References

- Davide Bresolin, Pieter Collins, Luca Geretti, Roberto Segala, Tiziano Villa, Sanja Živanović Gonzalez. A computable and compositional semantics for hybrid systems. Information and Computation 300: 105189 (2024).
- Pieter Collins. Semantics and Computability of the Evolution of Hybrid Systems. SIAM Journal on Control and Optimization 49, 2 (2011), 890–925.
- Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. Information and Computation 185, 1 (2003), 105 – 157.
- Ariadne: an open library for formal verification of cyber-physical systems. `http://www.ariadne-cps.org`