Coalgebra meets Hybrid Systems

Renato Neves
joint work with Luis Barbosa, Sergey Goncharov, and José Proença
Some examples of how Coalgebra helps to provide

- syntax
- semantics
- and notions of equivalence

for hybrid systems
Introduction to Hybrid Systems

Overview

Hybrid Automata as Coalgebras

Hybrid While-Language

Conclusions and Future Work
The Essence of Hybrid Systems

Often found in the form of

- digital devices that closely interact with physical processes
- ‘impact-based’ physical systems

Described via classical methods of computation

Described via differential equations
Main Formalisms for Hybrid Systems

Hybrid Automata

\[
\begin{align*}
p' &= v \\
v' &= g \\
p &\geq 0
\end{align*}
\]

Hybrid While-Language

\[
\begin{align*}
\textbf{while} \; \textbf{true} \; \textbf{do} \{ & p' = v, v' = g \; \textbf{until} \; p = 0 \land v \leq 0 \\
& v := v \times -0.5 \}
\end{align*}
\]
Introduction to Hybrid Systems

Overview

Hybrid Automata as Coalgebras

Hybrid While-Language

Conclusions and Future Work
Talk’s Overview

We will focus on the two previous formalisms

- first hybrid automata;
- and then the hybrid while-language
Hybrid Automata and its Variants

The notion of a hybrid automaton has several variants

- deterministic
- non-deterministic
- probabilistic
- reactive
- weighted
- ...

Unfortunately: no uniform theory of hybrid automata

To be formally detailed later on
Coalgebra can help solve the aforementioned issue.

It provides a uniform theory of hybrid automata, which includes a

- notion of bisimulation
- notion of observational behaviour
- and a regular-expression-like language
Suitable semantics for **hybrid iteration** is difficult to establish.

Previous work crucially relies on nondeterminism and gives rise to problematic equations, e.g.

\[
\text{while true do } \{ p \} = 0
\]

Alternative (deterministic) semantics via final coalgebra + weak bisimilarity. It revolves around two monads for hybrid computation:

\[
\hat{H} \xrightarrow{\text{intensional to extensional}} H
\]

Abstracts away intermediate computational steps.
They extend non-deterministic finite automata with

- differential equations (for describing continuous dynamics)
- location invariants (for restricting the latter)
- assignments (for describing discrete dynamics)
- guards (for restricting the latter)

\[
\begin{align*}
\dot{p} &= v \\
\dot{v} &= g \\
p &\geq 0
\end{align*}
\]

\[
p = 0 \land v \leq 0,
\]

\[
v := v \times -0.5
\]
A hybrid automaton is a tuple \((L, E, X, dyn, inv, asg, grd)\) where

- \(L\) is a finite set of locations, \(E\) is a transition relation \(E \subseteq L \times L\), and \(X\) is a finite set of real-valued variables
- \(dyn\) is a function that associates to each location a system of differential equations over \(X\)
- \(inv\) is a function that associates to each location its invariant (a predicate over the variables in \(X\))
- \(asg\) is a function that given an edge returns an assignment command over \(X\). The function \(grd\) associates each edge with a guard (a predicate over the variables in \(X\))
Hybrid automata are nothing more than classical, non-deterministic automata with decorated states and edges, i.e.

\[ L \rightarrow P_\omega(L \times \text{Asg} \times \text{Grd}) \times \text{DiffEq} \times \text{Inv} \]

This immediately provides,

- a uniform notion of hybrid automata,
- a uniform regular-expression-like language

More details in [Neves and Barbosa, 2017]
A Zoo of Hybrid Automata

\[ M \rightarrow F(M \times Asg \times Grd) \times DifEq \times Inv \]

<table>
<thead>
<tr>
<th>Functor</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Deterministic</td>
</tr>
<tr>
<td>( P_\omega )</td>
<td>Classical</td>
</tr>
<tr>
<td>( D_\omega )</td>
<td>Markov</td>
</tr>
<tr>
<td>( P_\omega D_\omega )</td>
<td>Probabilistic</td>
</tr>
<tr>
<td>( W_\omega )</td>
<td>Weighted</td>
</tr>
</tbody>
</table>
Many variants of hybrid automata come equipped with their own semantics.

We can encode these uniformly and in functorial form

\[ [\cdot] : \text{HybAt}(F) \longrightarrow \text{Category of coalgebras} \]

Let us see how . . .
We adopt the following three assumptions (the last two used merely to simplify the presentation)

**Unique Solutions**

The function \( \text{dyn} \) only outputs systems of differential equations with exactly one solution. This induces a function

\[
\text{flow} : L \times \mathbb{R}^n \times [0, \infty) \to \mathbb{R}^n
\]

**Urgent Transitions**

As soon as an edge is enabled the current location must switch

**Non-restrictive Invariants**

The invariants of all locations are true
Uniform Semantics for Hybrid Automata – Rationale

\[ L \times \mathbb{R}^n \to F(L \times \text{Asg} \times \text{Grd}) \times \text{DifEq} \]

\[ \Rightarrow L \times \mathbb{R}^n \to F(L \times \text{Asg} \times \text{Grd}) \times (\mathbb{R}^n)[0,\infty) \]

\[ \Rightarrow L \times \mathbb{R}^n \to F\left(L \times \text{Asg} \times \text{Grd} \times (\mathbb{R}^n)[0,\infty)\right) \]

\[ \Rightarrow L \times \mathbb{R}^n \to F\left(L \times \text{Asg} \times \sqcup_{d \in [0,\infty)} (\mathbb{R}^n)[0,d) \times \mathbb{R}^n + (\mathbb{R}^n)[0,\infty)\right) \]

\[ \Rightarrow L \times \mathbb{R}^n \to F\left(L \times \sqcup_{d \in [0,\infty)} (\mathbb{R}^n)[0,d) \times \mathbb{R}^n + (\mathbb{R}^n)[0,\infty)\right) \]

\[ \cong L \times \mathbb{R}^n \to F\left(L \times \mathbb{R}^n \times \sqcup_{d \in [0,\infty)} (\mathbb{R}^n)[0,d) + (\mathbb{R}^n)[0,\infty)\right) \]

We obtain a coalgebra for \( F\left(- \times \sqcup_{d \in [0,\infty)} (\mathbb{R}^n)[0,d) + (\mathbb{R}^n)[0,\infty)\right) \)
Many variants of hybrid automata come equipped with their own notion of bisimulation.

The notion is placed at the semantic level.

The coalgebraic notion of bisimulation does not coincide with the notion of bisimulation for hybrid automata.

However...
Coalgebraic $\Phi$-Bisimulation

The Starting Point

Each equivalence relation $\Phi : (L \times \mathbb{R}^n) \times (L \times \mathbb{R}^n)$ induces a quotient map $q : L \times \mathbb{R}^n \to Q$

Denote $\bigsqcup_{d \in [0, \infty)} X^{[0, d)}$ by $Tr(X)$

And then ...

\[
\text{HybAt}(F) \downarrow \text{semantics} \quad \Rightarrow \quad \text{CoAlg}(F(- \times Tr(\mathbb{R}^n) + (\mathbb{R}^n)^{[0, \infty)})) \leftarrow \text{colour} \quad \Rightarrow \quad \text{CoAlg}(F(- \times Tr(\mathbb{R}^n \times L) + (\mathbb{R}^n \times L)^{[0, \infty)}))
\]

\[
\quad \downarrow \text{forget} \quad \Rightarrow \quad \text{CoAlg}(F(- \times Tr(Q) + Q^{[0, \infty)})) \quad \downarrow \text{quotient}
\]
Coalgebraic $\Phi$-bisimilarity covers the classic notions of bisimilarity for,

1. deterministic,
2. non-deterministic,
3. and probabilistic hybrid automata.
The generalised semantics yields coalgebras for

\[ G \cong F \left( - \times \coprod_{d \in [0, \infty)} (\mathbb{R}^n)^{[0, d]} + (\mathbb{R}^n)^{[0, \infty)} \right) \]

If \( F \) is bounded we obtain a notion of observable behaviour given by the corresponding universal map to the final coalgebra

\[
\begin{array}{ccc}
X & \xrightarrow{\text{beh}_{[ha]}} & \nu \gamma \cdot G \gamma \\
\downarrow \text{[ha]} & & \downarrow \cong \\
GX & \rightarrow & G(\nu \gamma \cdot G \gamma)
\end{array}
\]
The observable behaviour for \( F := \text{Id} \) is given by:

\[
\nu \gamma \cdot \left( \gamma \times \bigsqcup_{d \in [0, \infty)} (\mathbb{R}^n)[0, d^+) + (\mathbb{R}^n)[0, \infty) \right)
\]

\[
\simeq \left( \bigsqcup_{d \in [0, \infty)} (\mathbb{R}^n)[0, d^+) \right)^* \times (\mathbb{R}^n)[0, \infty) + \left( \bigsqcup_{d \in [0, \infty)} (\mathbb{R}^n)[0, d^+) \right)^\omega
\]

The case in which a guard never activates.
Via the semantics functor \([-\cdot]\) we obtain the following picture

\[
\begin{align*}
\mathbb{bb} &= \left\{ \begin{array}{ll}
p' &= v \\
v' &= g \\
p &\geq 0
\end{array} \right. \\
p = 0 \land v \leq 0,
\end{align*}
\]

\[
v := v \times -0.5
\]

\[
\text{beh}_{\mathbb{bb}}(*,(5,0)) = \ldots
\]

Position and velocity
Currently working on a coalgebraic notion of approximate bisimilarity for hybrid automata
Introduction to Hybrid Systems

Overview

Hybrid Automata as Coalgebras

Hybrid While-Language

Conclusions and Future Work
Fix a stock of variables \( X = \{ x_1, \ldots, x_n \} \). Then we have,

**Linear Terms**

\[
\text{LTerm}(X) \ni r \mid r \cdot x \mid t + s
\]

**Atomic Programs**

\[
\text{At}(X) \ni x := t \mid x'_1 = t_1, \ldots, x'_n = t_n \text{ for } t
\]

"run" the system of differential equations for \( t \) seconds

**Hybrid Programs**

\[
\text{Prog}(X) \ni a \mid p ; q \mid \text{if } b \text{ then } p \text{ else } q \mid \text{while } b \text{ do } \{ p \}
\]
Semantics – Key Aspects

How to interpret a hybrid program $p$?

$$[x' = 1 \text{ for } 1] : \mathbb{R} \rightarrow \text{(trajectories over } \mathbb{R})$$

i.e. functions from a time-domain into $\mathbb{R}$

How to interpret sequential composition?

The signature of the denotation suggests the use of monads
Recall our use of $\sum_{d \in [0, \infty)}(\mathbb{R}^n)[0,d)$ to interpret hybrid automata

Then consider the left adjoint $[\text{Set}, \text{Set}]_\omega \to \text{Mnd}_\omega(\text{Set})$

We use the latter and $\sum_{d \in [0, \infty)}(\mathbb{R}^n)[0,d) \times (-)$ to obtain the monad

$$X \mapsto \mu \gamma \cdot \left( \sum_{d \in [0, \infty)}(\mathbb{R}^n)[0,d) \times \gamma + X \right)$$

$$\simeq \left( \sum_{d \in [0, \infty)}(\mathbb{R}^n)[0,d) \right)^* \times X$$

Kleisli composition amounts to concatenation of lists of trajectories
Denotations \([p]\) become functions of the type

\[
[p] : \mathbb{R}^n \rightarrow \left(\sum_{d \in [0,\infty)} (\mathbb{R}^n)^{[0,d)}\right)^* \times \mathbb{R}^n
\]

**Example (with \(n = 1\))**

\[
[p] : \mathbb{R}^n \rightarrow \left(\sum_{d \in [0,\infty)} (\mathbb{R}^n)^{[0,d)}\right)^* \times \mathbb{R}^n
\]

\[
[x' = 1 \text{ for } 1](0) = ([\lambda t \in [0,1). 0 + t], 1)
\]

\[
[x' = 1 \text{ for } 1 ; x' = 1 \text{ for } 1](0)
\]

\[
= ([\lambda t \in [0,1). 0 + t, \lambda t \in [0,1). 1 + t], 2)
\]

\[
[\text{while } \text{true do } \{x' = 1 \text{ for } 1\}] = ?
\]
Semantics – Second Approach

Instead of using the least fixpoint we use the greatest

\[ X \mapsto \nu \gamma. \left( \sum_{d \in [0, \infty)} (\mathbb{R}^n)[0,d) \times \gamma + X \right) \]
\[ \simeq \left( \sum_{d \in [0, \infty)} (\mathbb{R}^n)[0,d) \right)^* \times X + \left( \sum_{d \in [0, \infty)} (\mathbb{R}^n)[0,d) \right)^\omega \]

This is an instance of a universal construction which tells that

- the functor above is also a monad (henceforth denoted by \( \hat{H} \))
- the monad supports a partial iteration operator

\[
\begin{align*}
f : X &\rightarrow \hat{H}(Y + X) \\
f^\#: X &\rightarrow \hat{H}(Y)
\end{align*}
\]
\[ f : X \rightarrow \hat{H}(Y + X) \]
\[ f^\# : X \rightarrow \hat{H}(Y) \]

\(f^\#\) iterates over \(f\) until the latter outputs a value of type \(Y\); and concatenates all lists of trajectories produced along the way.

Example (with \(n = 1\))

\[
\left[ \text{while true do \{x' = 1 for 1\}} \right](0) = [\lambda t \in [0, 1). \ 0 + t, \ \lambda t \in [0, 1). \ 1 + t, \ \lambda t \in [0, 1). \ 2 + t, \ldots ]
\]
The proposed semantics is intensional e.g.

\[(x' = 1 \text{ for } 1) ; (x' = 1 \text{ for } 1) \neq (x' = 1 \text{ for } 2)\]

We should abstract away from invisible intermediate steps, similarly to the case of weak bisimulation.

This amounts to ‘coherently’ turning a sequence of trajectories into a single trajectory.
From a Sequence of Trajectories into a Single Trajectory

**Concatenation of Trajectories**

\[(\lambda t \in [0, d_1). f_1(t)) + (\lambda t \in [0, d_2). f_2(t))\]

\[= \lambda t \in [0, d_1 + d_2). \text{if } t < d_1 \text{ then } f_1(t) \text{ else } f_2(t - d_1)\]

---

**Infinite Concatenation of Trajectories**

\[f_1 + f_2 + \cdots = \lambda t \in [0, \sum_{i \in \mathbb{N}} d_i). f_j(t - \sum_{i \leq j} d_i) \text{ where }\]

\[j \geq 1 \text{ is the smallest integer s.t. } t < \sum_{i \leq j} d_i\]
The previous operation induces a retraction

\[
\hat{H} \xrightarrow{\rho} \left( X \mapsto \sum_{d \in [0, \infty)} (\mathbb{R}^n)^{[0,d)} \times X + \sum_{d \in [0, \infty]} (\mathbb{R}^n)^{[0,d)} \right)
\]

where \( \rho \) resorts to concatenation of trajectories and \( \nu \) is defined as

\[
\begin{align*}
inl(f, x) &\mapsto \text{inl}([f], x) \\
inr(f) &\mapsto \text{inr}[f_{[0,1)}, f_{[1,2)}, \ldots] \text{ if duration of } f \text{ equals } \infty \\
inr(f) &\mapsto \text{inr}[f, !, !, \ldots] \text{ otherwise}
\end{align*}
\]

Let us denote the functor on the right-hand side by \( H \)...
An Extensional Hybrid Monad Appears

\[ \rho \text{ (to extensional)} \]

\[ \hat{H} \quad \leftrightarrow \quad H \]

\[ \nu \]

\( H \) inherits from the monad \( \hat{H} \) (through \( \nu \) and \( \rho \))

- Kleisli composition
- an iteration operator

\[ f : X \to H(Y + X) \]

\[ f^\dagger : X \to H(Y) \]
Interpretation via $H$ provides the desired aforementioned equality

$$(x' = 1 \text{ for } 1) ; (x' = 1 \text{ for } 1) = (x' = 1 \text{ for } 2)$$

and also other expected ones, such as

$$\text{while true do } \{x' = 1 \text{ for } 1\} = \text{while true do } \{x' = 1 \text{ for } 2\}$$
Thoughts about this Interpretation of While-Loops

- We did not use domain theory
- Instead we used the concept of final coalgebra to guide us
- Extensional
- Contrasts with previous works in the sense that
  - it is deterministic
  - does not collapse infinite while-loops into a single point of divergence, i.e. we do not necessarily obtain
    \[
    \text{while true do } \{ p \} = 0
    \]
  - In fact we get a \textit{continuum} of divergence points
## A Taxonomy of While Loops

<table>
<thead>
<tr>
<th></th>
<th>Non-progressive</th>
<th>Progressive</th>
<th>Zeno</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Divergent</strong></td>
<td>( \text{while} (\text{true}) { ) ( x := x + 1 } )</td>
<td>( \text{while} (\text{true}) { ) ( x := x + 1; (\text{wait} , \varepsilon) } )</td>
<td>( \varepsilon := 1 ) ( \text{while} (\text{true}) { ) ( x := x + 1; (\text{wait} , \varepsilon) ) ( \varepsilon := \frac{\varepsilon}{2} )</td>
</tr>
<tr>
<td><strong>Convergent</strong></td>
<td>( x := 0 ) ( \text{while} (x \leq 10) { ) ( x := x + 1 } )</td>
<td>( x := 0 ) ( \text{while} (x \leq 10) { ) ( x := x + 1; (\text{wait} , \varepsilon) } )</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
http://arcatools.org/assets/lince.html#fulllince
# Table of Contents

- Introduction to Hybrid Systems
- Overview
- Hybrid Automata as Coalgebras
- Hybrid While-Language
- Conclusions and Future Work
Conclusions and Future Work

- Hybrid systems in an object-oriented setting [Jacobs, 2000]
  - Seen as coalgebras $U \rightarrow A \times U^B \times U^{R\geq 0}$
- Approximate bisimulation coalgebraically
  e.g. [Sprunger et al., 2018, König and Mika-Michalski, 2018]
  - Seems particularly well-suited for systems with continuous state-spaces (such as those used in the semantics of hybrid automata)
This talk was financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 under the Portugal 2020 Partnership Agreement and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within project IBEX, with reference PTDC/CCI-COM/4280/2021.
