Computable Analysis and Nonlinear Dynamics

Pieter Collins
Centrum voor Wiskunde en Informatica
Amsterdam, Netherlands
Pieter.Collins@cwi.nl

Dynamics and Computation
8 - 12 February 2010
Outline

- Introduction
- Hybrid Systems
- Reachability Analysis and Verification
- Computable Types
- Implementation in Ariadne
- Examples
- Open problems
- Conclusions
Introduction

Outline

Introduction
- Analysis of Nonlinear Hybrid Systems
- Automatic analysis
- The reachability problem
- Ariadne
- Existing tools

Modeling Hybrid Systems

Evolution and Reachability Analysis

Hybrid Automata

Computable Types

Implementation

Hybrid Components

Case Study

Additional work

Open problems

Conclusions
Analysis of Nonlinear Hybrid Systems

- Want to solve *verification problems* for *hybrid systems*:
  - Safe control-to-target: Starting from an *initial set* $X_0$, remain in the *safe set* $S$, and eventually reach the *target set* $T$ and stay there.

- Hybrid evolution comprises:
  - Continuous dynamics in each *discrete mode*.
  - Instantaneous *transitions* between modes.
  - Switching is controlled by *guard* conditions.
Automatic Systems Analysis

- We would like to develop an automated tool to validate the behaviour of a hybrid system.
  - Write the system description in a *modelling language* with a well-defined *operational semantics*.
  - Write the specification in a *temporal logic*.
  - Combine the system and specification to a single *dynamical model* with well-defined *computational semantics* of evolution.
  - Perform *reachability analysis* on the dynamical model to check that the system satisfies its specifications.
  - Output a proof of *correctness*, or a trace analysis to show any detected *failures*. 
The reachability problem

- A hybrid state \((q, r)\) is \textit{reachable} from \((q_0, x_0)\) if there is a sequence of transitions \((q_{i-1}, x_{i-1}) \rightarrow (q_i, x_i)\) with \((q, x) = (q_i, x_i)\) for some \(i\).
- Given a set of points \(X_0 \subset Q \times \mathcal{X}\), the \textit{reachable set} is the set of points reachable from some \((q_0, x_0) \in X_0\).
- Reachability analysis can be used to verify
  - controllability properties \(\text{Reach}(H, x_0) \cap T \neq \emptyset\).
  - safety properties \(\text{Reach}(H, X_0) \subset S\).
  - The reachable set is \textit{uncomputable} for all but the simplest systems.
Ariadne Reachability Analyser—Main Goals

- Reachability analysis and verification of hybrid automata.
  - Modular analysis with assume/guarantee reasoning.
  - Discrete abstractions (simulation relations and symbolic dynamics)

- Focus on:
  - Small subsystems with nontrivial continuous dynamics.
  - *Numerical* tools computing *approximations* to the reachable set with *guarantees* on the errors.
Existing tools

- Dynamical systems
  - GAIO (Dellnitz, Froyland & Junge, 2001)
  - CAPD Library (Mrozek, Zgliczynski et al., 2005)
- Differential equations:
  - VNODE (Nedialkov, 2006)
  - COSY INFINITY (Berz, Makino, 2005)
- Timed automata:
  - Kronos (Yovine, 1997)
  - UPPAAL (Larsen et al., 1997)
- Affine hybrid automata:
  - d/dt (Asarin, Dang & Maler, 2001)
  - VeriShift (Botchkarev & Tripakis, 2000)
  - Hy(per)Tech (Henzinger, Ho & Wong-Toi, 1997)
  - PHAVer (Frehse, 2005)
- Nonlinear hybrid automata:
  - CheckMate (Krogh, Silva et al., 2000)
  - HSolver (Ratschan, She & Dzetkulič, 2008)
Modeling Hybrid Systems
Watertank – System

The water level $h$ in a tank with continuous outflow and a valve-restricted inflow needs to be controlled to between $h_{\text{min}}$ and $h_{\text{max}}$.

Model as a hybrid system with three components, the tank itself, the valve, and the controller.

- Use **dotted variables** $\dot{x}$ or $\text{dot}(x)$ to define differential equations.
- Use **primed variables** $x'$ or $\text{prime}(x)$ to define resets.
Watertank Example – Tank Model

// Create the tank object
AtomicHybridAutomaton tank("tank");

// Declare the system variables
RealVariable height("height");
RealVariable aperture("aperture");

// Declare the system constants
RealConstant lambda("lambda",0.02);
RealConstant rate("rate",0.3);

// Declare the system dynamic
tank.new_mode((dot(height)=rate*aperture-lambda*height));
Watertank Example – Valve Model

AtomicHybridAutomaton valve("valve");

RealConstant T("T",4.0);
RealVariable aperture("aperture");

valve.new_mode(open,(aperture=+1.0));
valve.new_mode(closed,(aperture=-1.0));
valve.new_mode(opening,(dot(aperture)=+1.0/T));
valve.new_mode(closing,(dot(aperture)=-1.0/T));

valve.new_reset(closed,start_opening,opening,(prime(alpha)=alpha));
valve.new_reset(open,start_closing,closing,(prime(alpha)=alpha));
valve.new_reset(closing,start_opening,opening,(prime(alpha)=alpha));
valve.new_reset(opening,start_closing,closing,(prime(alpha)=alpha));

valve.new_urgent_transition(opening,finish_opening,open,alpha>=1.0);
valve.new_urgent_transition(closing,finish_closing,closed,alpha<=0.0);
Watertank Example – Controller Model

```java
AtomicHybridAutomaton controller("controller");

RealConstant hmax("hmax",8.0);
RealConstant hmin("hmin",5.5);

controller.new_mode(requested_close);
controller.new_mode(requested_open);

ccontroller.new_urgent_transition(
    requested_open, start_closing, requested_close, height>=hmax);
ccontroller.new_urgent_transition(
    requested_close, start_opening, requested_open, height<=hmin);```
Watertank Example – Composed System

CompositeHybridAutomaton system((tank, valve, controller));
Watertank Example—Evolution

Outline

Introduction

Modeling Hybrid Systems
- Watertank – System
- Watertank Example – Tank Model
- Watertank Example – Valve Model
- Watertank Example – Controller Model
- Watertank Example – Composed System

Watertank Example—Evolution

System Validity

Evolution and Reachability Analysis

Hybrid Automata

Computable Types

Implementation

Hybrid Components

Case Study

Additional work

Open problems

Conclusions
System Validity

- A component may only change its own state variables in a reset. This is to ensure that the behaviour of a component cannot be changed in composition.
  \[
  \text{logistic: } \dot{x} = x(1 - x) \\
  \text{break_everything: } x' = x - 1.
  \]

- The composed system model is **valid** if every real variable has **exactly one** defining equation in any mode or transition and there are no **algebraic loops**.
  
  \begin{align*}
  \checkmark & \quad x = 2y, \quad \dot{y} = x; \\
  \checkmark & \quad x = 2y, \quad y' = x; \\
  \times & \quad x = 1, \quad \dot{x} = 0; \\
  \checkmark & \quad x' = 1, \quad \dot{x} = 0; \\
  \times & \quad x = y^2, \quad y = x + 1;
  \end{align*}
Evolution and Reachability Analysis
A trajectory of a hybrid system comprises continuous evolution interspersed with discrete events.
### Evolution of a hybrid system

- A trajectory of a hybrid system comprises continuous evolution interspersed with discrete events.

- The continuous dynamics is governed by a differential equation/inclusion \( \dot{x} = f(x) \).
Evolution of a hybrid system

- A trajectory of a hybrid system comprises continuous evolution interspersed with discrete events.

- The continuous dynamics is governed by a differential equation/inclusion \( \dot{x} = f(x) \).
- Switching is controlled by guard conditions \( g_e(x) = 0 \).
Evolution of a hybrid system

- A trajectory of a hybrid system comprises continuous evolution interspersed with discrete events.

- The continuous dynamics is governed by a differential equation/inclusion \( \dot{x} = f(x) \).

- Switching is controlled by guard conditions \( g_e(x) = 0 \).

- The discrete dynamics is governed by an update equation \( x' \in r_e(x) \).
A trajectory of a hybrid system comprises continuous evolution interspersed with discrete events.

The continuous dynamics is governed by a differential equation/inclusion $\dot{x} = f(x)$.

Switching is controlled by guard conditions $g_e(x) = 0$.

The discrete dynamics is governed by an update equation $x' \in r_e(x)$. 
Model checking

- To verify system, construct a *discretisation* of the reachable set.
Model checking

- To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
Model checking

- To verify system, construct a *discretisation* of the reachable set.
  
  ![Discretisation Diagram]

- Outer-approximate the initial set on a grid.
- For each cell:
Model checking

- To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
- For each cell:
  - Compute an over-approximation of the flow tube and final set for a time step \( h \).
Model checking

- To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
- For each cell:
  - Compute an over-approximation of the flow tube and final set for a time step $h$.
  - Outer-approximate the evolved sets on the grid.
Model checking

- To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
- For each cell:
  - Compute an over-approximation of the flow tube and final set for a time step $h$.
  - Outer-approximate the evolved sets on the grid.
  - Refine the grid as necessary to obtain a reasonable approximation.
Model checking

- To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
- For each cell:
  - Compute an over-approximation of the flow tube and final set for a time step $h$.
  - Outer-approximate the evolved sets on the grid.
  - Refine the grid as necessary to obtain a reasonable approximation.
To verify system, construct a *discretisation* of the reachable set.

- Outer-approximate the initial set on a grid.
- For each cell:
  - Compute an over-approximation of the flow tube and final set for a time step $h$.
  - Outer-approximate the evolved sets on the grid.
  - Refine the grid as necessary to obtain a reasonable approximation.
- Repeat recursively until no new cells are found.
Hybrid Automata

Introduction

Modeling Hybrid Systems

Evolution and Reachability Analysis

Hybrid Automata

- Hybrid automaton
- Hybrid trajectories
- Standard semantics
- Upper semantics
- Lower semantics
- Fundamental operations
- Enclosure Sets
- Representation of enclosure sets

Computable Types

Implementation

Hybrid Components

Case Study

Additional work

Open problems

Conclusions
A hybrid automaton is a tuple \((\mathcal{E}, \mathcal{X}, F, P, R, G)\) where

- \(\mathcal{E}\) and \(\mathcal{X}\) are types.
- \(F : \mathcal{X} \Rightarrow \mathbb{C}(\mathbb{R}^+; \mathcal{X})\) is a multflow.
- \(P : \mathcal{X} \rightarrow \mathcal{T}\) is a progress predicate.
- \(R : \mathcal{X} \times \mathcal{E} \Rightarrow \mathcal{X}\) is a reset relation.
- \(G : \mathcal{X} \times \mathcal{E} \rightarrow \mathcal{T}\) is a guard predicate.

- Often, the flow \(F\) is defined by a (nondeterministic) differential equation \(\dot{x} = f(x, v)\) or differential inclusion \(\dot{x} \in f(x)\).

- Any predicate \(P\) on \(X\) can be described by \(p(x) \geq 0\) for some \(p : X \rightarrow \mathbb{R}\), or as a pair \((P^\circ, \overline{P}) \in \mathcal{O}(X) \times \mathcal{A}(X)\).
Hybrid trajectories

- A hybrid trajectory is an alternating sequence of discrete events $e_n \in \mathcal{E}$ and continuous transitions $\xi_n : [t_n, t_{n+1}] \to \mathcal{X}$.

- A variable may only jump discontinuously during an event.

- The graph of a hybrid trajectory $\xi$ is defined by:
  \[ \text{Gr}(\eta) = \{(t, n, x, e) \in \mathbb{R}^+ \times \mathbb{Z}^+ \times X \times E \mid t \in [t_n, t_{n+1}], x = \xi_n(t), e = e_n\}. \]

- The set of hybrid trajectories can be given a type via the Fell topology on the graphs.
Standard semantics of evolution

- A hybrid trajectory is a solution of the hybrid system
  \[ H = (\mathcal{E}, \mathcal{X}, F, P, R, G) \]
  if for all \( n \),
  
  i) \( \xi(\cdot, n) \in F(\xi(t_n, n)) \).
  
  ii) \( \forall t \in [t_n, t_{n+1}), \xi(t, n) \in P \).
  
  iii) \( \xi(t_n, n) \in R_{e_n}(\xi(t_n, n-1)) \).
  
  iv) \( \xi(t_n, n-1) \in G_{e_n} \).

- With the standard semantics, it may not be possible to compute the evolution due to discontinuities arising at crossings with the guard sets.
Upper semantics

- In order to prove safety properties, we need to compute a *compact* set of hybrid trajectories from a given initial point which *over-approximates* the trajectories using the standard semantics.

- In order to compute a compact set of hybrid trajectories:
  - The functions $F$ and $R$ must be computable compact-valued maps.
  - The sets $P$ and $G$ must be computable closed sets.

- In most system models, the functions $F$ and $R$ are continuous and computable.

- To ensure that the sets $P$ and $G$ are closed, we need to replace predicates $p(x) < 0$ with $p(x) \leq 0$.
  - This may introduce additional nondeterminism, especially during grazing contact with guard sets.
Non-blocking lower semantics

- In order to prove non-blockingness and to find counterexamples, we need to compute a *separable* set of hybrid trajectories which *under-approximates* the standard semantics.
  - The functions $F$ and $R$ must be computable separable-valued maps.
  - The sets $P$ and $G$ must be computable open sets.
- Often, we want to specify an *urgent* action with progress predicate $p(x) < a$ and guard $g(x) \geq a$.
  - If $P$ and $G$ do not overlap, taking interiors means no trajectory exists.
  - Postulate a *non-blocking* conditions that $P \cap \overline{G} = X$.

In order to compute good approximations to the evolution using lower semantics, explicit information about urgent transitions is needed.
The fundamental operations for computing the evolution of a hybrid system are:

- Computing the flow $\phi$ of a differential equations $\dot{x} = f(x)$:
  $$\dot{\phi}(x, t) = f(\phi(x, t)); \phi(x, 0) = 0.$$  
- Computing the crossing time of a guard set $g(x) = 0$:
  $$g(\phi(x_0, \tau(x))) = 0.$$  
- Applying the flow or a reset to an enclosure set:
  $$S' = \{r(x) | x \in S\} \text{ or } S' = \{\phi(x, t) | x \in S, t \in [t_0, t_1]\}.$$  

The fundamental operation for computing a discretisation of the evolution is:

- Computing an outer-approximation of an enclosure on a grid.

All these operations must be performed rigorously and efficiently!
The point reached at time $t_f \in [0, h]$ with a single event at time $t_1$ starting at time $t_0$ from point $x_0 \in X_0$ is:

$$\phi_2(r(\circ \phi_1(x_0, t_1 - t_0)), t_f - t_1).$$

- If $a(x) \geq 0$ is an *permissive guard* condition for the event, then:
  $$a(\phi_1(x_0, t_1 - t_0)) \geq 0.$$

- If $c(x) \leq 0$ is an *invariant* in the first mode:
  $$\max_{t \in [t_0, t_1]} c(\phi_1(x_0, t - t_0)) \leq 0.$$

- If $c$ is increasing during the evolution, the invariant simplifies to:
  $$c(\phi_1(x_0, t_1 - t_0)) \leq 0.$$
The point reached at time $t_f$ with a single event at time $t_1$ starting at time $t_0$ from point $x_0$ is:

$$\phi_2(r(\circ \phi_1(x_0, t_1 - t_0)), t_f - t_1).$$

If $g(x) \geq 0$ is an urgent guard condition for the event, then:

$$\max_{t \in [0, t_1]} g(\phi_1(x_0, t - t_0)) \leq 0 \land g(\phi_1(x_0, t_1 - t_0)) = 0.$$
The point reached at time $t_f$ with a single event at time $t_1$ starting at time $t_0$ from point $x_0$ is:

$$\phi_2(r(\circ \phi_1(x_0, t_1 - t_0)), t_f - t_1).$$

If $g(x) \geq 0$ is an urgent guard condition for the event, then:

$$\max_{t \in [0, t_1]} g(\phi_1(x_0, t - t_0)) \leq 0 \land g(\phi_1(x_0, t_1 - t_0)) = 0.$$ 

If $g$ is increasing during the evolution, solve $t_1 = \gamma_1(x_0)$, so

$$x_f = \phi_2(r \circ \phi_1(x_0, \gamma_1(x_0) - t_0)), t_f - \gamma_1(x_0)).$$
Computable Types
Computable Topology and Analysis

- Need a formal theory of computation to:
  - Give suitable *representations* of points, sets and maps.
  - Analyse which operations are *computable* with respect to the given representations.

- Develop a theory of *computable types* with
  - natural “computable” operations, and
  - well-defined representations via concrete approximations.
Base types

- Countable numerical types $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{A}$ with natural arithmetic operations.
- Three-valued logical type $\mathcal{T} = \{\top, \bot, \uparrow\}$ with subtypes $\mathcal{S} = \{\top, \uparrow\}$ and $\mathcal{B} = \{\top, \bot\}$.
- Canonical real type $\mathcal{R}$ with operations:
  - Constant 1,
  - Arithmetical operations $+,-,\times,\div$,
  - Comparison $<$, which is only semidecidable.
  - Limit of effective Cauchy sequences $|x_i - x_j| < 2^{-\min(i,j)}$.
  - Standard functions including $\max$, $\min$, $\abs$, $\neg$, $\rec$, $\sqrt{x}$, $\exp$, $\log$, $\sin$, $\cos$, $\tan$, $\atan$.
- Equality for any uncountable type (including $\mathcal{R}$) is undecidable!
Product, Function and Sub-Types

- Product types $\mathcal{X}_1 \times \mathcal{X}_2$ with computable *projections*: 
  $$\pi_1 : \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{X}_1, \quad \pi_2 : \mathcal{X}_1 \times \mathcal{X}_2 \to \mathcal{X}_2.$$ 
- Function types $\mathcal{Y}^{\mathcal{X}} \cong C(\mathcal{X}; \mathcal{Y})$ with computable *evaluation*: 
  $$\varepsilon : C(\mathcal{X}; \mathcal{Y}) \times \mathcal{X} \to \mathcal{Y}, \quad \varepsilon(f, x) = f(x).$$ 
- Isomorphism between $\mathcal{A} \times \mathcal{X} \to \mathcal{Y}$ and $\mathcal{A} \to C(\mathcal{X}; \mathcal{Y})$. 
- Arbitrary subtypes $\{x \in \mathcal{X} \mid p(x)\}$. 
Standard types

- If \((X, \tau)\) is a topological space with basis \((I_0, I_1, I_2, \ldots)\), define a representation by
  \[
  \delta(n_0, n_1, \ldots) = x \iff \{I_{n_j} \mid j \in \mathbb{N}\} = \{n \mid x \in I_n\}.
  \]

- If \((X, d)\) is a complete metric space and \(A = (a_0, a_1, \ldots)\) is a countable dense subset such that \(d : A \times A \to \mathbb{R}\) is computable, define the Cauchy representation by
  \[
  \gamma(n_0, n_1, \ldots) = x \iff d(a_{n_i}, a_{n_j}) < 2^{-\min(i,j)} \land x = \lim a_{n_j}.
  \]

- If \((X, \leq)\) is a directed complete partial order with a countable order-base \((p_0, p_1, \ldots)\), then define representations by
  \[
  \omega_{\leq}(n_0, n_1, \ldots) = x \iff \{p_{n_j} \mid j \in \mathbb{N}\} = \{n \mid x \leq p_n\}.
  \]
Set Types

- Characterise open sets as the preimage of the Sierpinski space under the characteristic function.
  \[ \mathcal{O}(\mathcal{X}) \cong S^{\mathcal{X}} \text{ by } U \cong \chi_U. \]
- Characterise closed sets by the characteristic of the complement.
  \[ \mathcal{A}(\mathcal{X}) \cong S^{\mathcal{X}} \text{ by } A \cong \chi_{\mathcal{X}\setminus A}. \]
- Characterise compact sets by subset relation \( \mathcal{O}(\mathcal{X}) \rightarrow S \).
  \[ C \subset U_1 \cap U_1 \iff V \subset U_1 \land V \subset U_2. \]
- Characterise separable sets by intersection relation \( \mathcal{O}(\mathcal{X}) \rightarrow S \).
  \[ V \Join U_1 \cup U_1 \iff V \Join U_1 \lor V \Join U_2. \]
- Represent open sets as continuous preimages of \((0, \infty)\).
- Represent compact sets as continuous images of the Cantor set.
- Represent separable sets as (the closure of) the continuous image of \( \mathbb{N} \).
### Computable operations on sets

- Finite intersection $\emptyset \times \emptyset \rightarrow \emptyset$.
- Countable union $\emptyset^\mathbb{N} \rightarrow \emptyset$.
- Complement $\emptyset \leftrightarrow \mathcal{A}$.
- Conversions $\mathcal{K} \rightarrow \mathcal{A}$ and $\emptyset \rightarrow \mathcal{V}$.
- Intersection $\emptyset \times \mathcal{V} \rightarrow \mathcal{V}$, $\mathcal{A} \times \mathcal{K} \rightarrow \mathcal{K}$.
  - The intersection of two separable sets is not computable.
- Singleton $\mathcal{X} \rightarrow \mathcal{V}(\mathcal{X})$ and $\mathcal{X} \rightarrow \mathcal{K}(\mathcal{X})$.
- Preimage $\mathcal{C} \times \emptyset \rightarrow \emptyset$ and $\mathcal{C} \times \mathcal{A} \rightarrow \mathcal{A}$.
- Image $\mathcal{C} \times \mathcal{V} \rightarrow \mathcal{V}$ and $\mathcal{C} \times \mathcal{K} \rightarrow \mathcal{K}$.
  - The image of a closed set is not computable.
Probability and measure

- Define the type of measures on a space $X$ as $\mathcal{M}(X) \equiv \mathcal{PL}(\mathcal{C}(X; \mathbb{R}); \mathbb{R})$.
  - Push-forward is computable:
    
    $$ f_* (\mu) (\phi) = \mu (\phi \circ f) $$

- Corresponds to weak topology on measures.

- Define Lebesgue space types $L^p$ as completions of rational step functions under the $p$-norms:
  
  $$ \| f \|_p = \int_X |f(x)|^p \, dx $$

  - Indefinite integral is computable $L^p (\mathbb{R}) \rightarrow C(\mathbb{R})$.
  - Image under a continuous function $f : \mathbb{R} \rightarrow \mathbb{R}$ is computable $L^\infty (\mathbb{R}) \rightarrow L^\infty (\mathbb{R})$.
Implementation – Low-Level Operations
Interval Arithmetic

- Due to roundoff and truncation errors, cannot exactly compute the results of real-number computations.

- Represent an approximation to $x \in \mathbb{R}$ by an interval $[x] = [x, \overline{x}]$ with $x, \overline{x} \in \mathbb{F}$.

- Perform computations using **rounded interval arithmetic**:
  - $[x, \overline{x}] + [y, \overline{y}] = [x + d y, \overline{x} + u \overline{y}]$

- Some computations can be performed exactly:
  - $-[x, \overline{x}] = [-\overline{x}, -x]$; $2^n [x, \overline{x}] = [2^n x, 2^n \overline{x}]$. 

- Error bounds can easily be computed:
  - $2 |(x + y) - (x + n y)| \leq (x + u y) - u (x + d y)$. 

Care must be taken with some compilers!

- GCC sometimes reuses results of arithmetical computations after a rounding mode change:
  ```
  set_rounding_downward();
  double l = x + y;
  set_rounding_upward();
  double u = x + y;
  // Error: Compiler uses x+y computed for l!!
  ```

- GCC sometimes simplifies arithmetic expressions, even if this changes the rounding:
  ```
  set_rounding_upward();
  double l = - ( (-x) - y );
  // Error: Compiler simplifies to x+y!
  ```

- Solution (a trick) is to use the `volatile` keyword for intermediate results.
Interval Arithmetic

- Transcendental functions can be computed using power-series expansions.
  - Use Horner's rule to evaluate polynomials:
    \[ \sum_{i=0}^{n} a_i x^i = a_0 + x \times (a_1 + x \times (a_2 + \cdots (a_{n-1} + x \times a_n) \cdots)) \]
  - Use standard reductions to reduce radius of convergence.
  - Take care with negative terms!

- Example \( \cos_u(x) \):
  - Reduce \( |x| \) using \( \cos(x) = (-1)^n \cos(x + n\pi), \)
    \( \cos(x) = \cos(-x) \) and \( \cos(x) = 2 \cos^2(x/2) - 1 \).
  - Set \( s_u = x \times_u x \), and use
    \[ \cos_u(x) = 1 + u s_u \times_u (-\frac{1}{2} u + u s_u \times_u (\frac{1}{24} u + u \cdots)) \]
  - If \( s_u < 0.5 \), truncation error after 8 terms is less than round-off error.
A *Taylor model* is a scaled polynomial approximation to a function with a uniform error.

Formally, a Taylor model is a triple $T(s, p, e)$ where:
- $s : [-1, +1]^n \rightarrow \prod_{i=1}^n [a_i, b_i]$ is a scaling function.
- $p : [-1, +1]^n \rightarrow \mathbb{R}$ is a polynomial.
- $e \in [0, \infty)$ is an error bound.

A Taylor model $T(s, p, e)$ is an approximation to $f : \mathbb{R}^n \rightarrow \mathbb{R}$ on $D = [a_1, b_1] \times \cdots \times [a_n, b_n] = \text{range}(s)$ if:
$$||f|_D - p \circ s^{-1}||_{\infty} \leq e.$$
Taylor Models

- Store polynomial data for \( \sum_{\alpha} c_{\alpha} x^\alpha \) in an ordered sparse format with contiguous data array:

| \( \alpha_1 \) | \( \alpha_2 \) | \( \alpha_3 \) | \( \alpha_4 \) | \( \alpha_5 \) | | \( |\alpha| \) | \( c_\alpha \) | \( \beta_1 \) | \( \beta_2 \) | \( \beta_3 \) | \( \beta_4 \) | \( \beta_5 \) | | \( |\beta| \) | \( c_\beta \) |

- Fundamental operations can be implemented very efficiently:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
<th>Allocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \leftarrow r + c )</td>
<td>( O(1) )</td>
<td>0</td>
</tr>
<tr>
<td>( r \leftarrow r \times c \cdot x^\alpha )</td>
<td>( O(</td>
<td>r</td>
</tr>
<tr>
<td>( r \leftarrow p + c \cdot x^\alpha q )</td>
<td>( O(</td>
<td>p</td>
</tr>
<tr>
<td>( r \leftarrow r + p \times q )</td>
<td>( O(</td>
<td>r</td>
</tr>
</tbody>
</table>

- There is no speedup for in-place addition \( r \leftarrow r + p \).
- Multiplication is efficiently implemented in terms of addition and monomial multiplication.
- Analytic functions are efficiently implemented using the fundamental operations on power-series expansions.
Taylor Models – Scaling

- operator*=(TaylorModel& x, Interval c) {
  Float roundoff_error=0.0; // Accumulated roundoff error
  for(Term term in x) {
    if(term.value >= 0) {
      Float u = mul_up(term.value, c.upper);
      Float l = mul_down(term.value, c.lower);
    } else {
      Float u = mul_up(term.value, c.lower);
      Float l = mul_down(term.value, c.upper);
    }
    roundoff_error = add_up(roundoff_error, sub_up(u, l)/2);
    term.value = mul_near(term.value, c.midpoint);
  }
  x.error = mul_up(x.error, max(abs(c.lower), abs(c.upper)));
  x.error = add_up(x.error, roundoff_error);
}
Taylor Models – Addition

- TaylorModel operator+(TaylorModel x, TaylorModel y) {
  TaylorModel r(x.argument.size());
  r.reserve(x.number_of_nonzeros()+y.number_of_nonzeros());
  Term xterm=x.begin(); Term yterm=y.begin();
  while (xterm!=x.end() && yterm!=y.end()) {
    if (xterm.key == yterm.key) {
      Float u = add_up(xterm.value, yterm.value);
      Float l = add_down(xterm.value, yterm.value);
      r.error = add_up(r.error, sub_up(u, l)/2);
      r.new_term(xterm.key,
          add_near(xterm.value, yterm.value));
      ++xterm; ++yterm;
    } else if(xterm.key<yterm.key) {
      r.new_term(xterm); ++xterm;
    } else if(yterm.key<xterm.key) {
      r.new_term(yterm); ++yterm;
    }
  }
  r->error = add_up(r.error, x.error, y.error);
  return r;
}
To avoid growth of the number of coefficients, sweep small coefficients into the uniform error:

\[ (\sum_{\alpha} c_{\alpha} x^{\alpha} + c_{\beta} x^{\beta}) + e \mapsto \sum_{\alpha} c_{\alpha} + (|c_{\beta}| + u \epsilon). \]

Here, the fact that the domain of \( p \) is \([-1, +1]^n\) is a big advantage!
Taylor Models–Refinement/Restriction

- A Taylor model $T_1 = (s, p_1, e_1)$ refines $T_0 = T(s, p_0, e_0)$ if any function represented by $T_1$ is also represented by $T_0$.
  - A necessary and sufficient conditions for refinement is
    $\|p_1 - p_0\|_\infty + e_1 \leq e_0$.

- Coefficients of higher-order terms can be reduced by restricting to a subdomain:
  - e.g. If $p(x) = x^3$ on $[-1, +1]$, then we can restrict to $[0, 1]$ by setting $p = q \circ s^{-1}$ where $s(z) = (z + 1)/2$ and
    $$q(z) = (p \circ s)(z) = \frac{1}{8} + \frac{3}{8}z + \frac{3}{8}z^2 + \frac{1}{8}z^3.$$  
    The cubic term has decrease by a factor $2^3$.  

Taylor Models — (Anti)differentiation

- A Taylor model $T = (s, p, e)$ can be antidifferentiated with respect to a variable $x_j$ by
  $$\int f(x) \, dx_j = r_j \left( e + \sum_\alpha \frac{c_\alpha}{\alpha_j+1} x^{\alpha+\epsilon_j} \right)$$
  where $r_j = \text{rad}(\text{dom}(s_j)) = (b_j - a_j)/2$.

- In general, Taylor models cannot be differentiated since the error is in the uniform norm.

- If $e = 0$, then the polynomial can be differentiated term-by-term:
  $$\frac{df}{dx_j} = \frac{1}{r_j} \sum_\alpha \alpha_j c_\alpha x^{\alpha-\epsilon_j}.$$
Function Evaluation using Interval Extensions

- Use interval arithmetic to rigorously evaluate a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at a point $x$ or over a range of values $[x]$.
- An *interval extension* of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a function $[f] : \mathbb{I} \mathbb{R}^n \rightarrow \mathbb{I} \mathbb{R}$ satisfying $[f](\lfloor x \rfloor) \supset f(\lfloor x \rfloor)$.
- The *natural interval extension* of an expression is obtained by using interval operations at each stage.
  - e.g. If $f(x) = x(1 - x)$, then
    
    $f([0, 1]) = [0, 1] (1 - [0, 1]) = [0, 1] \times [0, 1] = [0, 1]$.

- Different forms for the same function give rise to different interval extensions.
  - e.g. $f(x) = x(1 - x) = \frac{1}{4} - (\frac{1}{2} - x)^2$, so
    
    $f([0, 1]) = \frac{1}{4} - (\frac{1}{2} - [0, 1])^2 = \frac{1}{4} - [-\frac{1}{2}, \frac{1}{2}]^2 = [0, \frac{1}{4}]$, which is sharp.
Sharp Interval Extensions

- If the interval \([x]\) is very narrow (order of machine precision), then the natural interval extension usually gives good results.

- If a variable \(x\) occurs only once in the expression for a function \(f\), then the natural interval extension is sharp (using exact arithmetic).

- If \(f\) is monotone (increasing) in \(x\) on the interval \([a, b]\), then an interval extension is given by
  \[
  f([a, b]) = [f(a), f(b)].
  \]
Evaluating Polynomial Ranges

- Compute the range of a polynomial
  \[ p : [-1, +1] \to \mathbb{R}, \quad p(x) = \sum_{i=0}^{n} a_i x^i. \]

- Simple range:
  \[ \text{rng}(p) \subset a_0 + (|a_1| + u |a_2| + u \cdots + u |a_n|) [-1, +1]. \]
  Can be improved by taking \( a_{2j}[-1, +1]^{2j} = a_{2j}[0, 1]. \)

- Monotone range:
  \[ \text{rng}(p) \subset \text{hull}(p([-1]), p([+1])). \]

- Bernstein range: Write \( p(x) = \sum_{j=0}^{m} c_{i}^{m} B_{j}^{m}(x) \) where
  \[ B_{j}^{m}(x) = \frac{1}{2^m} \binom{m}{j} (1 - x)^j (1 + x)^{m-j}. \]

**Theorem**
- \( \text{rng}(p) \subset \text{hull}\{c_{j}^{m} \mid j = 0, \ldots, m\}. \)
- If \( m \geq n \) and \( p \) is monotonic, the Bernstein range is exact.
- The Bernstein range converges to \( \text{rng}(p) \) as \( m \to \infty. \)
Implementation – High-Level Operations
Composition of functions

- If $F = T(s, q, d)$ and $G_i = T(r, p_i, e_i)$ are Taylor models, then the composition $F \circ G$ can be computed by:
  - Unscaling each $(p_i, e_i)$ by $s_i^{-1}$, and
  - Applying the polynomial expression $q$ to each $s_i^{-1} \circ p_i$.

- We obtain
  $$F \circ G = q(s_1^{-1} \circ (p_1 \pm e_1), \ldots, s_n^{-1} \circ (p_n \pm e_n)) \pm d.$$ 

- The polynomial $q$ can be evaluated directly, or converted into Horner or Bernstein form.

- If $s_i : [-1, +1] \rightarrow [c_i - r_i, c_i + r_i]$, then $s_i^{-1}(x) = (x - c_i)/r_i$, so
  is a scalar addition followed by a scaling.
Solving differential equations

- The flow $\phi(a, x_0, t)$ of the differential equation $\dot{x} = f(a, x)$ is
  \[ \dot{\phi}(a, x_0, t) = f(a, \phi(a, x_0, t)); \quad \phi(a, x_0, 0) = x_0. \]
- A \textbf{bound} for the flow starting in a box $D$ with time step $h$ is any set $B$ satisfying $D + hf(B) \subset B$.
- Picard’s iteration for computing the solution is
  \[ \phi_{n+1}(a, x_0, t) = x_0 + \int_0^t f(a, \phi_n(a, x_0, \tau)) \, d\tau. \]
- Picard’s iterations is a contraction for $t \leq h = 1/2LK$.
- Since the only operations used are \textit{compose} and \textit{antidifferentiate}, which are computable on Taylor models, we can apply Picard’s iteration to Taylor models.
- If $\phi_{n+1}$ refines $\phi_n$, then $\phi_{n+1}$ is a Taylor model for the flow, as is any further iteration.
- If $\phi_0(a, x_0, t)$ is the constant box $B$, then $\phi_1$ refines $\phi_0$. 
Solving algebraic equations

- Solve the system of algebraic equations \( f(x) = 0 \).
- Use the parameterised *Krawczyk* operator
  \[
  [x]_{n+1} = x_n - J f(x_n) + (I - JD f([x]_n))( [x]_n - x_n)
  \]
  where \( J \) is a fixed matrix, typically \( D f^{-1}(x_n) \), and \( x_n \in [x]_n \).
- Using Taylor models, can compute the solution \( x = h(a) \) in box \( X \) to \( f(a, x) = 0 \) on box \( A \), as long as \( D f(A, X) \) is nonsingular.
- For \( f : \mathbb{R}^P \times \mathbb{R} \to \mathbb{R} \), only need to compute a reciprocal, no matrix inversion is required.
Computing crossings with guard sets

- Crossings of the flow $\phi(x_0, t)$ with the guard $g(x) = 0$ can be computed by solving
  $$g \circ \phi(x_0, t) = 0.$$

- The time derivative of $g \circ \phi$ is
  $$\frac{d}{dt} (g \circ \phi) = \nabla g \cdot \dot{\phi} = (\nabla g \cdot f) \circ \phi.$$
  Hence transversality can be detected without computing the flow.

- The crossing time $\tau(x)$ can be computed using the parameterised Krawczyk operator:
  $$\tau_{n+1}(x) = \tau_n(x) - \frac{g(\phi(x, \tau_n(x)))}{(\nabla g \cdot f)(\phi(\hat{x}, \tau_n(\hat{x})))}$$
  with error
  $$\epsilon_{n+1} = \left(1 - \frac{(\nabla g \cdot f)(\phi(x, \lfloor \tau(x) \rfloor))}{(\nabla g \cdot f)(\phi(\hat{x}, \tau(\hat{x})))}\right)\epsilon_n.$$

- The convergence rate depends on the size of $(\nabla g \cdot f)(\phi(x, \lfloor \tau(x) \rfloor))$. 

### Outline
- Introduction
- Modeling Hybrid Systems
  - Evolution and Reachability Analysis
- Hybrid Automata
- Computable Types
- Implementation
- Hybrid Components
  - Input-output behaviours
  - Composition of behaviours
  - Behavioural or input/output approach?
  - Algebraic dependencies
  - Discrete-event systems
  - Continuous event sets
  - Duplicated events
  - Hybrid behaviour
  - Hybrid behaviour
- Case Study
- Additional work
- Open problems
- Conclusions
Geometric Calculus

- Constraint set \( S = g^{-1}(C') = \{ x \in \mathbb{R}^n \mid f(x) \in C \} \).
  - Open \( \{ x \in \mathbb{R}^n \mid g_i(x) < c_i \} \) or closed \( \{ x \in \mathbb{R}^n \mid g_i(x) \leq c_i \} \).
  - Preimage \( f^{-1}(S) = (g \circ f)^{-1}(C') \).

- Image set \( S = h(D) = \{ x \in \mathbb{R}^n \mid \exists y \in D, \ x = h(y) \} \).
  - Image \( f(S') = (f \circ h)(D) \).

- Constrained image set \( S = h(D \cap g^{-1}(C')) = \{ x \in \mathbb{R}^n \mid \exists y \in D, \ x = h(y) \land h(y) \in C' \} \).
  - Image \( f(S') = (f \circ h)(D \cap g^{-1}(C')) \).
  - Intersection \( S \cap f^{-1}(B) = h(D \cap g^{-1}(C') \cap (f \circ h)^{-1}(B)) \).
Computing discretisations of sets

- To discretise the evolved sets, we need to test whether
  \[ S = \{ h(y) \mid y \in D \land g(y) \in C \} \]
  intersects the box \( B \).
- More interested in proving disjointness than finding an intersection point.
- Possibility to use intermediate results for a box \( B_i \) to help test a nearby box \( B_j \).
- Many possible approaches:
  - Interval evaluation and subdivision:
    - ✔ Reliable and easy to implement. ✗ Slowww...
  - Constraint propagation:
    - ✔ Can handle arbitrary constraints. ✗ Not too nippy...
  - Linear programming:
    - ✔ Fast, easy to verify disjointness. ✗ Linear.
  - Nonlinear programming.
Subdivision and linearisation

- Formulate as a constraint satisfaction problem \( f(y) \leq c \).
- No solution if there exists a function \( \lambda \) such that \( \lambda(y) > 0 \) and \( \lambda(y) \cdot (c - f(y)) < 0 \) on \( D \).
- Use constraint propagation or naive splitting to subdivide \( D \) into pieces on which each \( f_i \) is nearly linear: \( f(y) = a + Ay \pm e \) on \( \tilde{D} \).
- Solve the linear optimization problem
  \[
  \max v \quad \text{s.t.} \quad Ay + v \leq c + e - a; \quad d_l \leq y \leq d_u.
  \]
  - The problem is infeasible if \( v_{\text{max}} < 0 \).
  - Primal-dual interior point methods seem to work well...
Subdivision and linearisation

- Formulate as a constraint satisfaction problem $f(y) \leq c$.
- No solution if there exists a function $\lambda$ such that $\lambda(y) > 0$ and $\lambda(y) \cdot (c - f(y)) < 0$ on $D$.
- Use constraint propagation or naive splitting to subdivide $D$ into pieces on which each $f_i$ is nearly linear:
  $$f(y) = a + Ay \pm e \text{ on } \tilde{D}$$
- Solve the linear optimization problem
  $$\max v \quad \text{s.t.} \quad A y + v \leq c + e - a; \quad d_l \leq y \leq d_u.$$  
  - The problem is infeasible if $v_{\text{max}} < 0$.
  - Primal-dual interior point methods seem to work well...
- Implementation still in progress (currently use naive subdivision).
Integration of Ariadne with the CIF

- The Compositional Interchange Format (CIF) is a hybrid systems description language.

  - There is a one-to-one mapping from a subset of CIF to Ariadne.
    - The mapping preserves the operational semantics.
    - An automatic translation tool is in development.
Input-output behaviours

- A system $S$ be considered as a procedure which takes an input $u \in U$ and outputs $y \in Y$.
  - The output may depend on an initial state $x_0 \in X$ and any disturbance $v \in V$.
  - The system can be described by a multi-valued input/output function $M : U \Rightarrow Y$.

- Abstract away the distinction between input and output to obtain the external behaviour.
  - The system can be described by a set $B \subset W = U \times Y$.

- Would like the representation of our components to only depend on the external behaviour and not on any state-space model.
Composition of behaviours

- The simplest way of composing systems is to intersect their external behaviours.
- Let $S_1$ and $S_2$ be systems with behaviours $B_j \subset V \times W_j$. Then the behavior $C$ of the composed system $S_1 || S_2$ is the subset of $V \times W_1 \times W_2$ defined by
  \[ \pi_{V \times W_j}(C) = B_j \text{ for } j = 1, 2. \]
The behavioural and input/output approaches suggest different types for representing a system:

- \( O(U \times Y) \leftrightarrow C(U; O(Y)) \rightarrow C(U; V(Y)) \rightarrow V(U \times Y) \).
- \( A(U \times Y) \leftrightarrow C(U; A(Y)) \leftarrow C(U; K(Y)) \leftarrow K(U \times Y) \).

If \( u \) represents an input, then we need to compute the set of possible outputs \( y \) from \( u \), so we need a “functional” description.

In order to compute a forwards-time evolution, we need to use separable and compact set types \( V \) and \( K \).

We cannot use the symmetric description preferred by a behavioural approach.
Algebraic dependencies

- When using a differential framework, two systems can only be combined if there are no *algebraic loops*.

\[ S_1 : \dot{x}_1 = f_1(x_1, u_1), \quad y_1 = h_1(x_1, u_1); \quad S_2 : y_2 = h_2(u_2). \]

Compose using \( y_1 = u_2 \) and \( u_1 = y_2 \) to obtain
\[ y_1 = h_1(x_1, h_2(y_1)), \]
an algebraic equation which need not have any solutions.

\[ S_1 : \dot{x}_1 = f_1(x_1, u_1), \quad y_1 = h(x_1); \quad S_2 : y_2 = h_2(u_2). \]

Composing yields
\[ \dot{x}_1 = f_1(x_1, h_2(h_1(x_1))), \]
which always has solution if \( f_1, h_1, h_2 \) are Lipschitz.

- Similar considerations hold in discrete-time:
\[ x'_1 = f_1(x_1, u_2), \quad y'_1 = h_1(x_1, u_1); \quad S_2 : y'_2 = h_2(u_2). \]
Algebraic dependencies

- Define a *dependency relation* between external variables, with values \{none, diff, alg\}.
  - A variable is an *input* if it has no dependencies.
  - A composed system is well-formed if there are no “algebraic loops”.

- **Theorem** If there are no algebraic dependencies between variables in a composed continuous-time system $S_1 || S_2$, then
  - The behaviour can be computed in $\mathcal{V}(\mathcal{C}(\mathbb{R}^+, \mathcal{E}))$
    (by using the Banach fixed-point theorem).
  - The behaviour can be computed in $\mathcal{K}(\mathcal{C}(\mathbb{R}^+, \mathcal{E}))$
    (by computing a decreasing sequence of compact sets).

The signature of a component must include the dependencies between variables, including which dependencies are algebraic.
Discrete-event systems

- A component of a discrete event system admits a language of external behaviours $L \subset E^*$. 

- If $E = A \sqcup O$ where $A$ are input events and $O$ output events, we can recover an input-output map $L : A^* \Rightarrow O^*$ by projecting:
  
  $L(\vec{a}) = \{ \vec{o} \in O^* \mid \exists \vec{e} \in E, \pi_A(\vec{e}) = \vec{a} \land \pi_O(\vec{e}) = \vec{o} \}$.

- We cannot recover $\vec{e}$ from $\vec{a}$ and $\vec{o}$ since we do not know in which order the events are interleaved.

The behaviour of a discrete-event system may be given as a language, but not just as an input-output map.
Continuous event sets

- The event type $E$ may not have decidable equality, or may not be compact.
  - If $E$ does not have decidable equality, then we cannot find an output for input sequence $\vec{a}$ given $L \in \mathcal{V}(E^*)$, and cannot intersect two languages.
  - If $E$ is not compact, then we cannot bound the set of possible outputs.

The behaviour of a discrete-time automaton with events from an indiscrete type must be described in an input-output setting.
Duplicated events

- When combining two event sequences, it is unclear whether a duplicated event represents the same transition.

- Suppose $S'$ receives an input sequence $\vec{a} = \ldots, (a_i, s_i), \ldots$ and $\vec{b} = \ldots, (b_j, t_j), \ldots$ with $(a_i, t_i) = (b_j, t_j)$.
  - Should $(a_i, s_i)$ and $(b_j, t_j)$ be regarded as duplicate records of the same event, and executed only once?
  - Should $(a_i, s_i)$ and $(b_j, t_j)$ be regarded as different events from different components, and both be handled?

- Since equality of $s_i$ and $t_j$ is undecidable, the solution cannot compare $s_i$ and $t_j$.

Easiest solution is to assume that different components generate distinguishable events.
Hybrid behaviour

- Even when an input event does not cause any change in the output, the ordering of events is still important.

\[
\begin{align*}
\dot{x}_1 &= 1, \quad \dot{x}_2 = 0; \quad y = x_2 \\
ra : x'_{1} &= x_1 + 1 \\
ro : x'_{2} + &= x_2 + 1 + x_1^2
\end{align*}
\]

- Since \(a\) does not change the output \(y\), it need not occur in the output signal \((O, Y)\).
- However, the relative ordering of \(a\) and \(o\) changes the state after both have occurred.
  - If \(a\) occurs before \(o\), the new state has \(x_2 = 3\).
  - If \(o\) occurs before \(a\), the new state has \(x_2 = 2\).
A hybrid component is a tuple \((A, O, U, Y, D, B)\) where

- \(A\) is a set of input events.
- \(O\) is a set of output events, disjoint from \(O\).
- \(U_1, \ldots, U_k\) are subspaces of input variables.
- \(Y_1, \ldots, Y_l\) are subspaces of output variables.
- \(D\) is a set of pairs \((i, j)\) for which \(Y_j\) depends algebraically on \(U_i\).
- \(B : \mathcal{H}(A, U) \ni \mathcal{H}(A \cup O, Y)\) is an input-output map, which also records the input events in the output.

The information \((A, O, U, Y, D)\) is sufficient to determine whether two components can be composed.

The information given by \(B\) is sufficient to compute the composition.
Case Study
Boost power converter

- **State:** inductor current $I$ and capacitor charge $Q$.
- **Switch open:**
  
  \[ I \geq 0: \quad L \frac{dI}{dt} = E - \frac{Q}{C}, \quad \frac{dQ}{dt} = I - \frac{Q}{RC}. \]
  
  or \[ Q \geq EC: \quad I = 0, \quad \frac{dQ}{dt} = -\frac{Q}{RC}. \]

- **Switch closed:** $Q \geq 0$
  
  \[ L \frac{dI}{dt} = E; \quad \frac{dQ}{dt} = -\frac{Q}{RC}. \]
Boost power converter—Python code

```python
bp = HybridAutomaton("Boost Power Converter")
bp.new_mode(mode_id="Switch open, current flowing",
             dynamic=AffineVectorField([[0,-1/(L*C)],[1,-1/(Q*C)]),
             [E/L,0],
             invariant=PolyhedralSet([-1,0,0]))
bp.new_transition(event_id="Diode blocks",
                  source_id="Switch open, current flowing",
                  target_id="Switch open, no current",
                  reset=AffineMap(Matrix([[0,1]]),Vector([0]),
                  activation=PolyhedralSet(Matrix([[1,0]]),Vector([0])))

initial_set=HybridSet(bp.state_space())
initial_set["Switch open, current flowing"]=RectangularSet("[I0,I0]x[Q0,Q0]")
safe_set=HybridSet(bp.locations()),PolyhedralSet([1,0,Qmax])

evolver.verify(ha,initial_set,safe_set)
```
Additional work

- Stochastic systems
- Computability of invariant sets
- Symbolic dynamics
- Topological entropy
- Complexity of reachability
- Control synthesis
- Behavioural approach
- Approximation of sets

Conclusions
Symbolic dynamics

- Given a continuous map \( f : X \rightarrow X \), and a collection of sets \( \mathcal{R} = \{R_1, \ldots, R_k\} \), define an itinerary of an orbit \( \vec{x} = (x_0, x_1, \ldots) \) as
  \[
  \{\vec{s} \mid x_n \in R_{s_n}\}
  \]
- Aim to compute under- and over-approximations to the set of itineraries.
- Over-approximations can be computed by considering
  \[\bigcap_{n=0}^{n-1} f^{-n}(\overline{R_{s_n}}) = \emptyset\]
- Under-approximations which converge in entropy need algebraic topology.

Joint work with Lorenzo Sella.
Model checking

- Specify temporal properties using *temporal logic*
  
  State formulae \( \Phi := AP \mid \neg \Phi \mid \Phi \land \Psi \mid \forall \phi \mid \exists \phi. \)
  
  Path formulae \( \phi := \Phi \mid \phi \land \psi \mid \neg \phi \mid X\phi \mid \phi U \psi \mid \phi R \phi. \)

- Universal formula \( \forall \Box \Diamond \Phi \) and negation \( \exists \Diamond \Box \neg \Phi. \)

- Give a verifyable semantics for formulae \( \forall \Box \Diamond \Phi \) and negation \( \exists \Diamond \Box \neg \Phi. \)

Joint work with Ivan Zapreev.
Differential inclusions

- Consider differential inclusions of the form
  \[
  \dot{x}(t) = f(x(t), v(t)) \text{ with } ||v(t)||_\infty \leq 1.
  \]

- Aim to find single-step solutions of the form
  \[
  x(h) = \phi(x(0), h, a) \pm \varepsilon
  \]
  with \(a \in [-1, +1]^d\) and error \(\varepsilon = O(h^{d+1})\).

Work in progress with Sanja Zivanovic.
Open problems
Stochastic systems

- Stochastic differential equation $dx(t) = f(x)dt + \sigma(x)dW$.
- **Problem** Given an initial probability distribution $\mu(0)$, compute the probability distribution $\mu(t)$ at time $t$.
- Should follow fairly straightforwardly from numerics of Ito integration.
- For a random process with bounded values $\nu : X \to L^\infty(\mathbb{R}; \mathbb{R}^n)$, maybe can compute directly.
Computability of invariant sets

- **Problem** Given a continuous function $f : X \rightarrow X$ and open $U \subset X$, is there an $f$-invariant subset $S \subset U$?

- The **fixed-point index** gives necessary and sufficient conditions for $f$ to have a *robust* fixed point in $U$.
- If $S = \text{Inv}(U) \subset U$, then the **Conley index** gives (necessary and?) sufficient conditions for $S \neq \emptyset$.
- If we allow perturbations in the class of *lower-semicontinuous multimaps* then $f^{-1}(\text{cl}(V)) \subset V$ for some $\text{cl}(V) \subset U$ is an optimal condition.
- The method must be *robust* in the sense that any sufficiently small perturbation of $f$ and $U$ also has an invariant set.
Symbolic dynamics

- Let \( f : X \rightarrow X \) and \( \mathcal{R} = (R_1, \ldots, R_k) \) a collection of subsets of \( f \).
- An \( \mathcal{R} \)-itinerary of a sequence \( (x_0, x_1, \ldots) \) is a sequence \( (s_0, s_1, \ldots) \) such that \( x_i \in R_{s_i} \).

**Problem** Compute over- and under-approximations to the set of itineraries of orbits of \( f \).

- The approximations may depend on the function space used.
Topological entropy

- **Problem** Find necessary and sufficient conditions under which the topological entropy can be computed, and implement an algorithm.

- For continuous maps $f : [0, 1] \to [0, 1]$, lower-semicomputability (Misieurewicz).
- For smooth maps $f : [0, 1]^n \to (0, 1)^n$, upper-semicontinuity (Yomdin).
- For smooth maps $f : [0, 1]^2 \to (0, 1)^2$, not lower-semicomputable.

- A related problem is to compute the *control entropy*, which is the average amount of information which needs to be exchanged to perform a control task.
Complexity of reachability

- Suppose $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is polynomial-time computable, $x_0 \in \mathbb{R}^n$ is an initial point and $S \subset \mathbb{R}^n$ is a safe set.

- **Problem** What is the complexity of deciding safety assuming the problem is robust?

- Find the complexity in $d \log(r/\epsilon)$ where $d$ dimension of the space, $r$ the radius and $\epsilon$ the sensitivity.

- Robust reachability is EXP-easy (convert to a graph) and NP-hard (encode SAT in the dynamics).
Control synthesis

- A discrete-time control system is defined by

\[ x_{n+1} = f(x_n, u_n, v_n); \quad y_{n+1} = h(x_n, w_n). \]

A control law is a (lower-semicontinuous) function \( g : Y \rightarrow U \)
defining a closed-loop system by \( u_{n+1} = g(y_n) \).

- **Problem** Implement an *efficient* correct-by-design control synthesis procedure.

- Expect the general problem to be intractible, but for practical problems can use system structure.
  - For near-linear problems, use Ricatti equation.
  - For nonlinear problems, use Hamilton-Jacobi-Bellmann equation and/or grid discretisation methods.
  - For composite systems, attempt to decompose control synthesis problem.
**Behavioural approach**

- The *input-output behaviour* of a system $S = \langle U; Y \rangle$ with time $T$ is a function $U^T \rightarrow Y^T$ giving the possible outputs $y(\cdot)$ on input $u(\cdot)$.
- The nondeterminism is due to incomplete knowledge of the initial state and to noise.

**Problem** What information do we need about the input-output behaviour to compose systems $S_1 = \langle U_0 \times U_1; Y_1 \rangle$ and $S_2 = \langle U_0 \times U_2; Y_2 \rangle$ by the identification $u_2 = f_{12}(y_1)$ and $u_1 = f_{21}(y_2)$?

- Expect compactness and the contraction-mapping theorem to be useful.
Approximation of sets

- **Problem** Let $p : [0, 1]^m \rightarrow \mathbb{R}^n$ be a polynomial. Find a method of computing a polynomial $q : [0, 1]^k \rightarrow \mathbb{R}^n$ with $\dim(k) < \dim(m)$ such that $\text{rng}(p) \subset \text{rng}(q)$.

- In the linear case, we have the following results:
  - If $A = OR$ where $\sup_i \sum_j |R_{ij}| \leq 1$, then $AE \subset OE$.
  - If $A = (B \ C)$ and $\sum_j |C_{ij}| \leq d_i$, then $AE \subset (B \ D)E$ where $D = \text{diag}(d_1, \ldots, d_m)$. 
Conclusions
Conclusions

- Ariadne is an open-source tool for reachability analysis of hybrid automata.
- Ariadne’s operations are based on a rigorous theory of computable analysis, have well-defined semantics and give optimal results.
- Ariadne is designed to be easy-to-use, with scripting in MATLAB and Python, and easy to extend, with a well-defined API.
Extensions

- Evolution of hybrid systems with inputs and noise.
- Verification of Linear Temporal Logic (LTL) formulae.
- Analysis of stochastic systems and dynamical games.
- Computation of optimal controllers.
- System reduction, including time-scale decomposition and assume-guarantee reasoning.
- A general-purpose tool for verified numerics?