Model-Checking in Systems Biology — From Micro to Macro

(and from Discrete and Precise to Continuous and Uncertain)

Pieter Collins

Department of Knowledge Engineering

Maastricht University

Collaborators

Michael Clerx

DKE & CARIM, Maastricht

Jan H. van Schuppen, Ivan Zapreev

Sanja Gonzalez Zivanovic

CWI, Amsterdam

Davide Bresolin, Luca Geretti, Tiziano Villa

Università degli Studi di Verona

First International Conference on Formal Methods in Macro-Biology, 22 September 2014

Introduction

Formal methods in computer science

Formal methods for systems analysis

- use precise logical descriptions and reasoning
- to obtain rigorous conclusions about the operation.

They have a long and successful history in the study of computer systems.

- Computer systems are discrete, so can be described and studied exactly.
- Software code is a precise, deterministic formulation of the behaviour.
  - Though parallel processing and interactions with the environment may introduce nondeterminism.
- Large software systems are are complex, with many subtle (and possibly unwanted) interactions.
  - But are (hopefully) designed to be as simple as possible.

Formal methods in biology—from discrete to continuous

Biological systems have continuous time- and state-spaces.

New challenges not encountered in discrete systems!

— differential equations, uncountably infinite state spaces, . . .

Even unclear *whether* we perform rigorous model checking of continuous systems on a digital computer.

Formal methods in biology—from precise to uncertain

Biological systems models are usually not known to a high degree of accuracy.

— *Uncertainty:* Precise measurement of rate constants almost impossible, even reaction pathways may be incomplete.

— *Variability:* Cells and organisms of the same type are not identical.

Formal methods in biology—from micro to macro

Macroscopic biological systems depend on interaction of many microscopic subsystems (reaction networks, cells, organisms).

— *Complexity:* Systems are *hugely* complex!

— *Emergence:* Behaviour not always clear even if individual processes are known.

Brute-force analysis infeasible in practice!

Use structural features to mitigate complexity:

— *Robustness:* Subsystems perform reliably under fluctuating external conditions.

— *Averaging:* Large numbers of individuals performing a similar task.

Expect rigorous and efficient analysis to require a modular approach utilising robustness and averaging.

Why formal methods?

If in-silico predictions are easily verified by experiment, non-rigorous computational methods probably suffice.

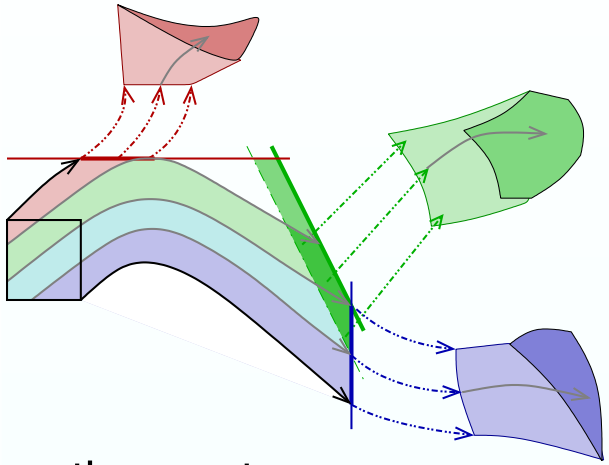
— Rigorous methods may still be useful if numerics is ill-conditioned.

Formal methods come into their own in cases where safety is critical and experimental validation of the prediction of a mathematical model is infeasible.

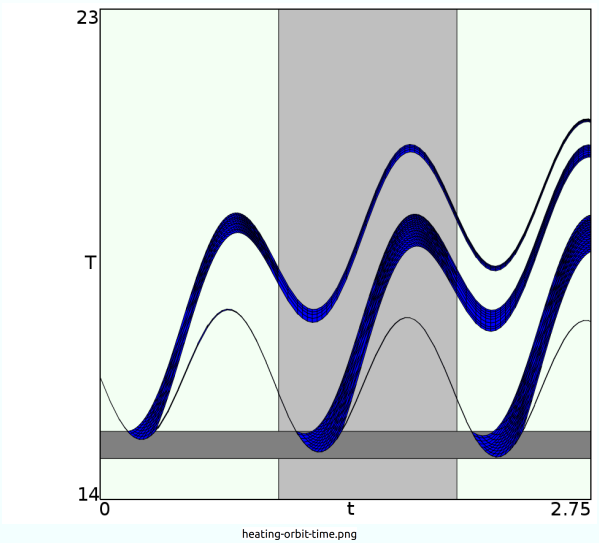
— advanced medical devices, personalised medicine.

ARIADNE—Verification of hybrid systems

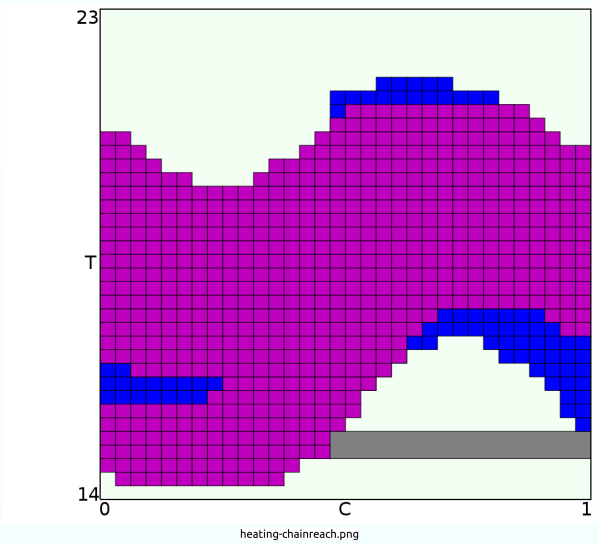
I am working on a tool ARIADNE for verification of *hybrid systems*.



Reachable set of a simple heating system.



heating-orbit-time.png



heating-chainreach.png

ARIADNE—Rigorous numerics

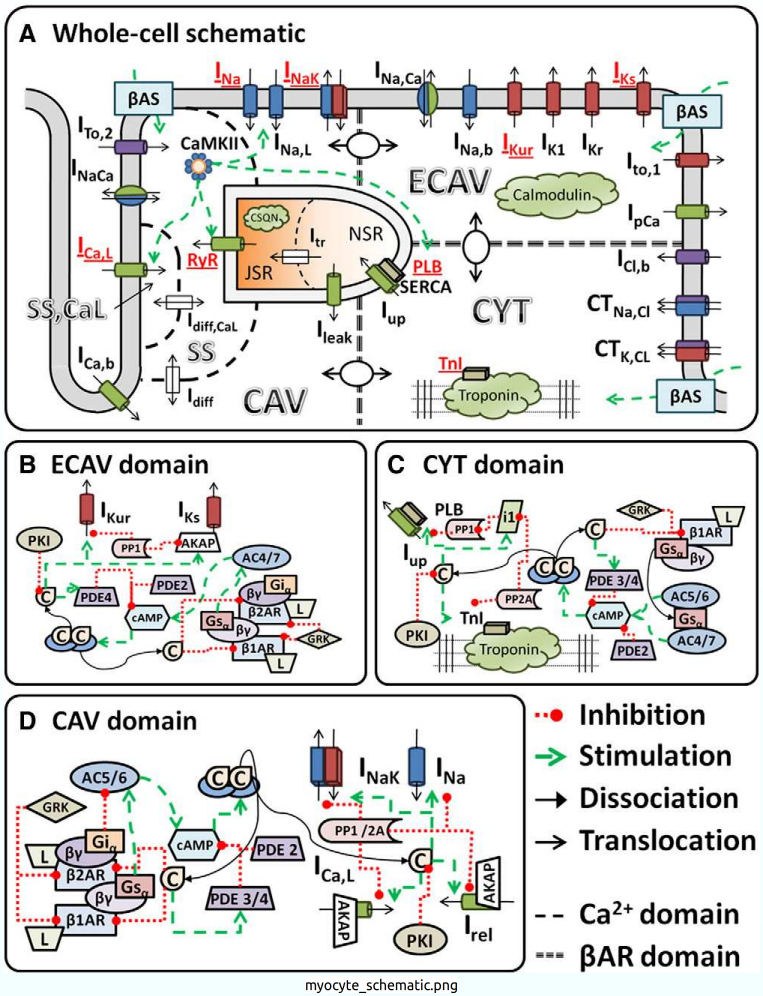
More generally, ARIADNE is aimed at being a general-purpose tool for rigorous numerics.

It includes rigorous calculi for real numbers, continuous functions, Euclidean geometry and dynamical systems.

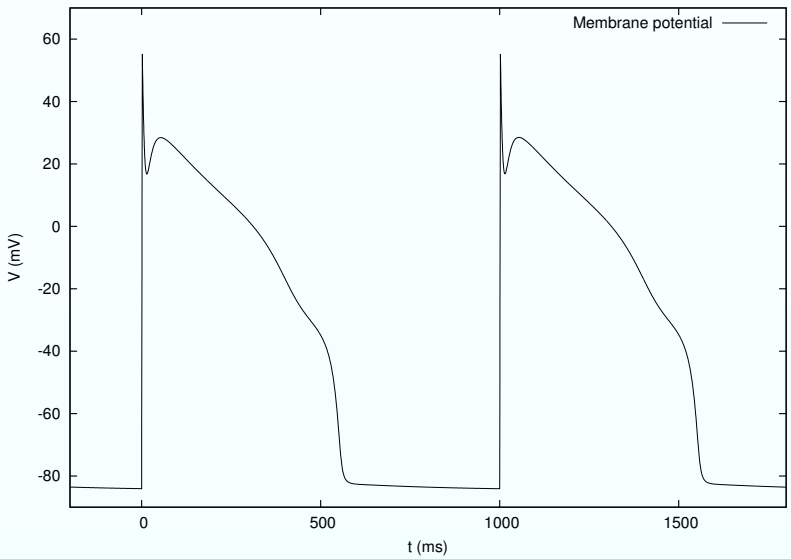
The methods are based on interval arithmetic, automatic differentiation, polynomial models and constraint sets.

Cardiac myocytes

Goal is to perform rigorous analyses in cardiac electrophysiology, from myocyte models to whole-heart models.



myocyte_schematic.png



Outline

- Introduction
- Computable Analysis
- Model-Checking Continuous Systems
- Variable and Uncertain Systems
- Complex and Compositional Systems
- Implementation in ARIADNE
- Myocyte Models
- Conclusions

Computable Analysis

Theory of computation for continuous mathematics

We need an appropriate theory of computation for continuous mathematics.

Mathematical objects lie in *uncountable* uncountable sets, and so require an *infinite* amount of information to specify exactly.

— For practical usefulness, computations must be finite, so we can only work with countable subsets.

— For simple theory, it would be *really handy* to be able to work with natural mathematical objects directly!

Reason *formally* on infinite data *streams*, but ensure that *useful* information can be obtained from finite pieces of data, so the theory implementable.

A simple approach is to use *type-two effectivity*, in which computation is performed by Turing machines working on data stream as a basis for a *computable type theory* via *realisability*.

— Combines an explicit machine representation with mathematical elegance.

— ω*-Scott domain theory* is equivalent, but technically more complicated.

Real numbers—Decimal representation

Since the real numbers are uncountable, any representation capable of describing *all* real numbers must necessarily use an infinite amount of data.

The decimal representation uses sequences of symbols `- . 0 1 2 3 4 5 6 7 8 9`

Given a partial decimal expansion, useful information can be extracted.

e.g. From $\pi = 3.14159\cdots$, we can deduce

$\pi \in [3.14159, 3.14160]$; equivalently $\mid \pi - 3.141595 \mid \leq 5 \times 10^{-6}$.

But the decimal expansion requires *too much* information to perform arithmetic!

$$3 \times 0.33333\cdots = \begin{cases} 0.99999\cdots? \\ 1.0000\cdots? \end{cases}$$

We cannot even output the first digit of the result!!

The decimal expansion contains *too little* information to perform comparisons!

$0.33333\cdots \geq \frac{1}{3}?$ $\quad$ $\pi = 3.14159\cdots?$ $\quad$ $0.00000\cdots > 0?$

Real numbers—Signed digit representation

A *signed-digit* representation includes symbols representing negative digits.

e.g. Using $\bar{4}$ for -4, write $3 \times 0.3333320\cdots = 1.0000\bar{4}0\cdots$.

The information given by signed-digit representations *is* appropriate for arithmetic.

The information is still *not* sufficient to decide whether a number x is positive!

— Tests $x > 0$ and $x \geq 0$ cannot be considered as functions $\mathbb{R} \to \mathbb{B}$.

— If $x \neq 0$, then we can prove $x \gtrless 0$ given enough signed digits.

Use *three-valued* Kleene logic $\mathbb{B}_\perp = \{\mathsf{T}, \mathsf{F}, \perp\}$. Testing $x \gtrsim 0$ is a predicate $\mathbb{R} \to \mathbb{B}_\perp$ with value T if $x > 0$, F if $x < 0$, and $\perp$ if $x = 0$.

Limits of *effectively convergent* Cauchy sequences $|x_n - x_m| \leq \epsilon_{\min(m,n)}$ can also be computed using the signed digit representation!

Many *equivalent* representations (based on metric/order completions).

Real numbers—Symbolic, computable and floating-point

Real numbers can also be described by symbolic formulae e.g. $\sin(2/3)$.

More generally, any program outputting the symbols of the signed digit representation described a *computable* real.

Note that we *absolutely do not* use (double-precision) floating-point numbers as a real number type!

— Roundoff errors cause computations to be unreliable.

— Binary floating-point numbers are insufficient even to represent decimals exactly e.g. 3.1 is rounded to $\frac{31}{10} + \frac{1}{5 \cdot 2^{51}}$.

Floating-point numbers *are* invaluable as an efficient data type for calculations!

— But should never be directly available to users as a specification type.

Real numbers—implementation in ARIADNE

In C++, mathematical objects can be represented by abstract classes.

In ARIADNE, we have a type `Real` from which we can extract rational and double-precision approximations:

`Real::get(Accuracy n) -> Rational;`
: computes x to an accuracy of 2^{-n}.

`Real::get(DoublePrecision) -> ApproximateDouble;`
: computes x using double-precision.

The `Real` type supports arithmetic, comparisons and elementary functions:

```
operator+(Real,Real) -> Real;
operator>=(Real,Real) -> Tribool;
exp(Real) -> Real;
```

Internally, computations are preformed using *interval* representations.

Representations of general types

Represent general mathematic objects by data *streams* Σ^ω.

Data streams are mapped to mathematical objects by *representations*, which are partial surjective functions $\delta : \Sigma^\omega \twoheadrightarrow X$.

— A δ-*name* of $x \in X$ is a sequence $\vec{w}$ such that $\delta(\vec{w}) = x$.

Representations induce a quotient *topology* on the represented space,

— or must be *admissible* with respect to an existing topology.

The information provided by a finite piece of a name of x is equivalent specifying a neighbourhood of x.

Computable operations

Computations are performed by non-halting Turing machines, and define partial functions $\eta : \Sigma^\omega \rightarrow\!\!\!\!\!\rightarrow \Sigma^\omega$.

A function $f : X \to Y$ is *computable* with respect to representations δ_X, δ_Y, if there is a machine-computable function $\eta : \Sigma^\omega \rightarrow\!\!\!\!\!\rightarrow \Sigma^\omega$ such that

$$\forall\, \vec{w} \in \mathrm{dom}(\delta_X), \quad \delta_Y(\eta(\vec{w})) = f(\delta_X(\vec{w})).$$

Theorem. *If $f : X \to Y$ is computable with respect to δ_X, δ_Y, then it is continuous with respect to the induced topologies.*

A *computable type* $\mathbb{X}$ is a pair $(X, [\delta])$, where $[\delta]$ is an equivalence-class of representations under machine-computable conversions.

Computable type theory

Computable types form a *Cartesian closed category*, which means they are a model of intuitionistic/constructive type theory:

There is a *terminal* type $\mathbb{I}$; objects of $\mathbb{X}$ are identified with morphisms $\mathbb{I} \to \mathbb{X}$.

There are *product* types $\mathbb{X}_1 \times \mathbb{X}_2$ with computable projections $\pi_i : \mathbb{X}_1 \times \mathbb{X}_2 \to \mathbb{X}_i$.

There are *exponential* types $\mathbb{Y}^{\mathbb{X}}$ with computable *evaluation* $\varepsilon : \mathbb{Y}^{\mathbb{X}} \times \mathbb{X} \to \mathbb{Y}$.

— The type $\mathbb{Y}^{\mathbb{X}}$ consists of the (sequentially) continuous functions from $\mathbb{X}$ to $\mathbb{Y}$, so we write $\mathbb{Y}^{\mathbb{X}} \equiv \mathbb{X} \to \mathbb{Y} \equiv \mathcal{C}(\mathbb{X};\ \mathbb{Y})$.

The natural equivalence (*Currying*) between $\mathbb{X} \times \mathbb{Y} \to \mathbb{Z}$ and $\mathbb{X} \to (\mathbb{Z}^{\mathbb{Y}})$ given by $\tilde{f}(x, y) = \varepsilon(\hat{f}(x), y)$ is computable.

— Computability of many operators can be *easily* proved by type-theoretic manipulations using the Curry equivalence!!

Summary—Fundamental types and operations

Natural numbers $\mathbb{N}$. Rationals $\mathbb{Q}$.

Kleene's logical type $\mathbb{T} = \{\mathsf{T}, \mathsf{F}, \bot\}$.

negation $\neg$, finite conjunction $\wedge$, disjunction $\vee$.

Boolean subtype $\mathbb{B} = \{\mathsf{T}, \mathsf{F}\}$.

Sierpinski type $\mathbb{S} = \{\mathsf{T}, \bot\}$.

countable disjunction $\bigvee$.

Real number type $\mathbb{R}$.

constants $\mathbb{Q} \hookrightarrow \mathbb{R}$

arithmetic $+, -, \times, \div : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$

comparison $\gtrsim\colon \mathbb{R} \times \mathbb{R} \to \mathbb{T}$

effective limit $\mathbb{R}^\omega \twoheadrightarrow \mathbb{R}$.

Continuous function types $\mathcal{C}(\mathbb{R}^n;\ \mathbb{R}^m)$.

evaluation $\mathcal{C}(\mathbb{R}^n;\ \mathbb{R}^m) \times \mathbb{R}^n \to \mathbb{R}^m : (f, x) \mapsto f(x)$.

composition $\mathcal{C}(\mathbb{R}^m;\ \mathbb{R}^l) \times \mathcal{C}(\mathbb{R}^n;\ \mathbb{R}^m) \to \mathcal{C}(\mathbb{R}^n;\ \mathbb{R}^l) : (f, g) \mapsto f \circ g$.

Predicates and open/closed sets

Since the power set of $\mathbb{R}$ has higher than continuum cardinality, we cannot handle arbitrary sets.

The usual classes of sets can be handled via the Sierpinski type since *open* sets have *characteristic functions* which are continuous $\mathbb{X} \to \mathbb{S}$.

Identify the type $\mathcal{O}(\mathbb{X})$ of open sets with $\mathbb{S}^{\mathbb{X}}$.

— Verifyable predicates are true on computable open sets.

Identify closed sets $\mathcal{A}(\mathbb{X})$ with $\mathbb{X} \to \{\mathsf{F}, \bot\}$ for which membership is falsifyable.

Quantifiers and compact/overt sets

Natural identification of *compact* sets with universal quantifiers.

$C \subset U \Leftrightarrow \forall\, x \in C, \chi_U(x)$ so identify $\mathcal{K}(\mathbb{X})$ with a subtype of $\mathbb{S}^{\mathcal{O}(\mathbb{X})}$.

Similarly, existential quantifiers give rise to *overt sets* $\mathcal{V}(\mathbb{X})$.

$S \bowtie U \Leftrightarrow \exists\, x \in S, \chi_U(x)$.

A set is *located* if we know it as a compact overt set. Located sets correspond to the Hausdorff metric.

Measures and probability distributions

Traditional probability theory is based on σ-algebras and measurable functions which are highly uncomputable.

Open sets can be approximated from below, and it is therefore reasonable to also be able to approximate the measure of such sets from below.

A *valuation* on $\mathbb{X}$ is a continuous function $\nu : \mathcal{O}(\mathbb{X}) \to \mathbb{R}^{+,\infty}_{<}$ satisfying

$$\nu(U_1 \cup U_2) + \nu(U_1 \cap U_2) = \nu(U_1) + \nu(U_2),$$

In other words, valuations only give lower-measures, and only of open sets.

A probability measure on $\mathbb{X}$ is a valuation P such that $P(\mathbb{X}) = 1$.

Valuations are naturally equivalent to integrals of lower-semicontinuous functions:

$$\int_{\mathbb{X}} \psi \, d\nu = \sup\left\{ \sum_{m=1}^{n} (p_m - p_{m-1}) \, \nu(\psi^{-1}(p_m, \infty]) \mid (0 = p_0 < \cdots < p_n) \in \mathbb{Q}^* \right\}$$

Random variables

Random variables can be defined as measurable functions a base probability space (Ω, P).

The space of random variables in $\mathbb{X}$ is the completion of $\mathcal{C}(\Omega, \mathbb{X})$ under the *Fan metric*

$$d(X, Y) = \inf\{\epsilon > 0 \mid P(d(X(\omega), Y(\omega)) > \epsilon) < \epsilon\}.$$

The *distribution* of a random variable $\mathbb{P}(X \in U)$ is a computable valuation.

The *expectation* of a real-valued random variable is computable since

$$\mathbb{E}(X) = \int_\Omega X(\omega)\, dP(\omega) = \int_0^\infty \Pr(X > \lambda)\, d\lambda\,.$$

Summary—Derived set and probabilistic type

Open sets $\mathcal{O}(\mathbb{X}) \equiv (\mathbb{X} \to \mathbb{S})$.

preimage $\mathcal{C}(\mathbb{X};\ \mathbb{Y}) \times \mathcal{O}(\mathbb{Y}) \to \mathcal{O}(\mathbb{X}) : (f, V) \mapsto f^{-1}(V)$.

intersection $\cap$, countable union $\bigcup$, complement $\mathcal{O}(\mathbb{X}) \leftrightarrow \mathcal{A}(\mathbb{X})$,

Compact sets $\mathcal{K}(\mathbb{X}) \subset (\mathcal{O}(\mathbb{X}) \to \mathbb{S})$;

$$C \subset (U_1 \cap U_2) \Leftrightarrow C \subset U_1 \wedge C \subset U_2$$

union $\cup : \mathcal{K} \times \mathcal{K} \to \mathcal{K}$, intersection $\cap : \mathcal{K} \times \mathcal{A} \to \mathcal{K}$.

Overt sets $\mathcal{V}(\mathbb{X}) \subset (\mathcal{O}(\mathbb{X}) \to \mathbb{S})$;

$$B \bowtie (U_1 \cup U_2) \Leftrightarrow B \bowtie U_1 \vee C \bowtie U_2$$

countable union $\bigcup : \mathcal{V}^\omega \to \mathcal{V}$, intersection $\cap : \mathcal{V} \times \mathcal{O} \to \mathcal{O}$.

Probability distributions $\mathcal{P}(\mathbb{X}) \subset \mathcal{O}(\mathbb{X}) \to [0,1]_<$

$$P(U_1 \cup U_2) + P(U_1 \cap U_2) = P(U_1) + P(U_2).$$

Compact/overt sets, probability distributions and random variables are *monads*:

unit: $\mathbb{X} \to \mathcal{M}(\mathbb{X})$, bind: $\mathcal{M}(\mathbb{X}) \times (X \to \mathcal{M}(\mathbb{Y})) \to (\mathcal{M}(\mathbb{X}) \to \mathcal{M}(\mathbb{Y}))$.

Model-Checking Continuous Systems

Joint work with Ivan Zapreev

Continuous-time systems

Problem: Verify properties of continuous-time systems described by a *linear temporal logic*.

Since we are working with continuous problems, *non-robust* instances cannot be verified or falsified.

Separate the problem of *solving* a *differential* equation from checking an explicity-given dynamic.

Differential equations

Consider the ordinary differential equation

$$\dot{x} = f(x); \quad x \in \mathbb{R}^n.$$

Under mild assumptions on f, the solution $\xi(t)$ starting at point x_0 exists and is unique for all $t \in \mathbb{R}^+ := [0, \infty)$.

Theorem (Ruohonen, 1996)**.** *Suppose the solution ξ of the differential equation $\dot{x} = f(x)$ starting at x_0 is unique. Then $\xi : \mathbb{R}^+ \to \mathbb{R}^n$ is computable given $f : \mathbb{R}^n \to \mathbb{R}^n$ and $x_0 : \mathbb{R}^n$.*

Solutions are computed in practise by rigorous numerical methods based on the Picard operator or Taylor series expansions.

Differential equations

The *flow* of the differential equation $\dot{x} = f(x)$ is the function

$$\phi : \mathbb{R}^n \times \mathbb{R}^+ \to \mathbb{R}^n$$

satisfying

$$\dot{\phi}(x,t) = f(\phi(x,t)) \quad \text{and} \quad \phi(x,0) = x.$$

Equivalently, we can think of ϕ as taking

$$\phi : \mathbb{R}^n \to (\mathbb{R}^+ \to \mathbb{R}^n)$$

so $\phi(x)$ gives the trajectory starting at $x \in \mathbb{R}^n$.

By Ruohonen's theorem, ϕ is computable given f if solutions are unique

Model checking

Validate system properties expressed in terms of *linear temporal logic*.

We make statements about *all* trajectories defined in terms of temporal operators including $\bigcirc$ (next), $\Diamond$ (eventually) and $\Box$ (always).

For simplicity, we restrict here to properties which can be expressed using the modalities *always* $\Box$ and *eventually* $\Diamond$, notably

- The *target* property $\forall \Diamond \Psi$.
- The *safety* property $\forall \Box \Psi$.
- The *recurrence* property $\forall \Box \Diamond \Psi$.
- The *stabilisation* property $\forall \Diamond \Box \Psi$.

Require $\Psi : \mathbb{X} \to \mathbb{S}$ to be a verifyable atomic proposition.

Continuous systems

Study deterministic continuous systems S given by an initial set

$$X_0 : \mathcal{K}(\mathbb{X})$$

and a flow function

$$\phi : \mathbb{X} \times \mathbb{R}^+ \to \mathbb{X} \equiv \mathbb{X} \to (\mathbb{R}^+ \to \mathbb{X})$$

satisfying

$$\phi(x, t_1 + t_2) = \phi(\phi(x, t_1), t_2) \text{ and } \phi(x, 0) = x.$$

Note that the results generalise easily to a nondeterministic flow:

$$\phi : \mathbb{X} \to \mathcal{K}(\mathbb{R}^+ \to \mathbb{X}).$$

Model checking–Target properties

The *target property* $S \models \forall \Diamond \Psi$ translates to

$$\forall\, x_0 \in X_0,\ \forall\, \xi \in \phi(x_0),\ \exists\, t \in \mathbb{R}^+,\ \Psi(\xi(t)).$$

The set of all trajectories $\phi(X_0) = \{\phi(x_0) \mid x_0 \in X_0\}$ is a *compact* subset of $\mathbb{X}^{[0,\infty)}$.

The *path formula* $\Diamond \Psi$ defines an *open* set of valid trajectories

$$\{\eta \in \mathbb{X}^{[0,\infty)} \mid \exists\, t \in \mathbb{R}^+,\ \Psi(\eta(t))\}$$

By the definition of compactness, testing inclusion of a compact set in an open set is verifyable!

Model checking—Safety properties

For the safety property $S \models \forall\, \Box \Psi$ we have a problem, since the the path formula defines a set of paths

$$\Box \Psi = \{\eta : \mathbb{R}^+ \to \mathbb{X} \mid \forall\, t \in [0, \infty), \Psi(\eta(t))\}$$

which is not open due to the universal quantification over a non-compact set.

To prove safety, we need to use the flow property $\phi(x, t_1 + t_2) = \phi(\phi(x, t_1), t_2)$.

Safety is verified if we can choose $T > 0$, open U and compact K with $U \subset K$ and check that

$$X_0 \subset U \ \wedge\ \phi(K, T) \subset U \ \wedge\ \phi(K, [0, T]) \subset \Psi^{-1}(\mathsf{T}).$$

The condition $U \subset K$ cannot be verified, but must be valid by construction of U and K; this is possible as long as the type $\mathbb{X}$ is *effectively locally compact*.

Conversely [Col07], if safety is *robust*, then it may be verified as above.

Model checking—Recurrence and Stabilisation

For the recurrence property, we have $\forall\Box\Diamond\Phi \equiv \forall\Box\forall\Diamond\Psi$, so recurrence is verifyable if we can verify safety and target properties.

For the stabilisation property $\forall\Diamond\Box\Psi$, we showed in [CZ11] that

$$\forall\Diamond\Box\Psi \equiv \forall\Diamond\forall\Box(\Psi \vee \forall\Diamond\Box\Psi).$$

The set of robustly stabilisable initial points can be computed as

$$\bigvee_{n=0}^{\infty}\Theta_n \quad \text{where} \quad \Theta_0 = \forall\Diamond\forall\Box\Psi; \qquad \Theta_{n+1} = \forall\Diamond\forall\Box(\Psi \vee \Theta_n).$$

The operator $\forall\Box(\Psi \wedge \Theta_n)$ denotes the *robust* safety operator as before.

Summary

We have seen:

— Under mild conditions on f, the differential equation $\dot{x} = f(x)$ can be integrated to give the *flow*

$$\phi : X \times \mathbb{R}^+ \to X \equiv X \to (\mathbb{R}^+ \to X).$$

— From the flow we can verify *robust* instances of infinite-time temporal logic properaties.

Hence the model-checking problem for continuous-time systems is *solvable* for robust instances!

Finding efficient algorithms is another matter...

— A combination of affine/polynomial calculus, (non)linear programming and counterexample-guided abstraction refinement is promising for small systems.

Uncertain Systems

Joint work with Sanja Gonzalez Zivanovic and Daniel Gra ca

Classes of uncertain systems

Uncertain differential systems have the form

$$\dot{x}(t) = f(x(t), v(t)); \quad v(\cdot) \in \mathcal{V} \subset \mathbb{V}^{[0,\infty)}$$

where $\mathcal{V}$ is a space of *noise* signals.

In a *nondeterministic* system, the noise is constrained to lie in some set V, yielding a *differential inclusion*

$$\dot{x} \in F(x) := \overline{\mathrm{conv}}\{f(x, v) \mid v \in V\}$$

In a *stochastic* system, the noise is a random variable. For a *stochastic differential equation*,

$$dx(t) = f(x(t))dt + G(x(t))\, dW(t),$$

the noise $v(t) = dW(t)/dt$ is a (formal) derivative of Brownian motion.

Aim to find an analogue of the flow of an ordinary differential equation, and show that it is computable.

Use of stochastic systems

Although stochastic differential equations are currently more commonly used as models of uncertain systems,

- they in principle are based on an *exact* knowledge of the noise distribution which is unavailable in practise, and
- they require that the noise signals is uncorrelated with scale-free increments:

  $W(t_1) - W(t_0) \sim N(0, t_1 - t_0)$ and is independent of W_0 for $t_1 > t_0$.

so may not be the most appropriate model.

Differential inclusions arise naturally in model-order reduction, since the uncertainties are not independent but can be bounded.

Other models of nondeterministic systems are possible, such as *imprecise* stochastic models, where the noise distribution is not known exactly.

Differential inclusions

Model the noisy system $\dot{x} = f(x, v)$ by the *differential inclusion*

$$\dot{x} \in F(x); \qquad F : \mathbb{R}^n \rightrightarrows \mathbb{R}^n.$$

A function $\xi : \mathbb{R}^+ \to \mathbb{R}^n$ is a solution if $\dot{\xi}(t) \in F(\xi(t))$ at *almost* every t.

If $f(x, V)$ is convex for all x, then solutions of $\dot{x} \in F(x)$ coincide with those of $\dot{x}(t) = f(x(t), v(t))$ with measurable $v(t)$.

Ideally, we wish to compute the set of trajectories $\Xi_F : \mathbb{R}^n \rightrightarrows (\mathbb{R}^+ \to \mathbb{R}^n)$

$$x_0 \mapsto \{\xi : \mathbb{R}^+ \to \mathbb{R}^n \mid \xi(0) = x_0 \wedge \dot{\xi}(t) \in F(\xi(t)) \text{ a.e.}\}$$

which takes x_0 to the set of trajectories starting at x_0.

For many applications it often suffices to compute the *flow tube*
$\Phi_F : \mathbb{R}^n \times \mathbb{R}^+ \rightrightarrows \mathbb{R}^n$

$$(x_0, t) \mapsto \{\xi(t) : \mathbb{R}^+ \to \mathbb{R}^n \mid \xi(0) = x_0 \wedge \dot{\xi}(\tau) \in F(\xi(\tau)) \text{ a.e.}\}.$$

Differential inclusions

To compute properties of trajectories, need differential inclusions taking compact overt values,

$$F : \mathbb{R}^n \to \mathcal{KV}(\mathbb{R}^n).$$

To ensure trajectories do not escape to infinity in finite time, we require the *bounded growth* condition:

$$\exists\, K, y \in F(x) \Rightarrow \mid y \mid \leq K(1 + \mid x \mid).$$

In order to prove existential properties, we need $F : \mathbb{R}^n \to \mathcal{V}(\mathbb{R}^n)$, and a (local) *one-sided Lipschitz* condition:

$$y_1 \in F(x_1) \;\Rightarrow\; \exists\, y_2 \in F(x_2),\ (x_1 - x_2)\cdot(y_1 - y_2) \leq L\,|x_1 - x_2|^2\ .$$

Theorem. *Let $F : \mathbb{R}^n \to \mathcal{KV}(\mathbb{R}^n)$ be a continuous multivalued function with compact convex values, bounded growth, and satisfying the one-sided Lipschitz condition. Then the solution of the initial value problem $\dot{x} \in F(x);\ \ x(0) = x_0$ is computable as a function*

$$\Xi_F : \mathbb{R}^n \to \mathcal{KV}(\mathcal{C}(\mathbb{R}^+, \mathbb{R}^n)).$$

Stochastic differential equations

Stochastic differential equations can be solved by effectivising the usual existence and uniqueness results for stochastic differential equations.

Theorem. *If $f : \mathbb{R}^n \to \mathbb{R}^n$ and $G : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ are Lipschitz function, then the solution of the stochastic differential equation*

$$dX(t) = f(X(t))dt + G(X(t))dW(t)$$

is computable as a random variable X taking values in $\mathcal{C}(\mathbb{R}^+, \mathbb{R}^n)$.

The stochastic differential equation may be interpreted using either the Itō or Stratonovich stochastic calculus.

— There is some evidence to suggest that the Stratonivich interpretation is more suitable for physical systems.

— If V_n is a sequence of smooth processes such that $V_n \to W$ almost everywhere, in the limit, we obtain the *Stratonovich* form:

$$dx(t) = f(x(t))dt + G(x(t)) \circ dW(t).$$

Summary

We have seen that the solution of noisy systems is computable for either

- bounded deterministic noise, or with
- stochastic noise modelled by a Wiener process.

Obtaining *efficient* rigorous solutions of differential inclusions is extremely difficult!

As far as I know, there are no algorithms giving rigorous solutions of stochastic differential equations...

Maybe other models of nondeterminism should be considered...

Compositional Systems

Multi-component systems

A macroscopic system is composed of many component subsystems.

Each subsystem S_i has its own *internal* state x_i, and interacts with other subsystems only via its *inputs* u_i and *outputs* y_i.

Subsystems interact via relationships of the form

$$u_i = g_i(y_1, \ldots, y_m).$$

Properties of interest are typically *global* quantities depending only on the external behaviour of the individual components.

The internal dynamics is unobservable and the details are not of experimental interest.

Aim to find methods for utilising the compositional structure.

— Focus on abstracting and/or simplifying component models, such that

— the recombination can be analysed and gives rigorous results.

State-space and behavioural models

A standard state-space model S of an input-output system is

$$\dot{x} = f(x, u), \quad y = h(x), \quad x(0) \in X_0.$$

The *external* behaviour is defined by the *input-output map*

$$\mathcal{IO}[S] : \mathbb{U}^{[0,\infty)} \rightrightarrows \mathbb{Y}^{[0,\infty)}$$

given by

$$\mu \mapsto \{\eta \mid \exists\, \xi \in \mathbb{X}^{[0,\infty)},\ \dot{\xi} = f(\xi, \mu) \ \wedge\ \eta = h(\xi) \ \wedge\ \xi(0) \in X_0\}.$$

Parallel composition

Systems S_i, $i = 1, 2$ with input spaces $\mathbb{U}_0 \times \mathbb{U}_i$ and output spaces $\mathbb{Y}_i$ such that $\mathbb{U}_1 = \mathbb{Y}_2$ and $\mathbb{U}_2 = \mathbb{Y}_1$ can be run in composition.

The *parallel composition* $S_1 \mid\mid S_2$ to have input space $\mathbb{U}_0$, state space $\mathbb{X}_1 \times \mathbb{X}_2$ and output space $\mathbb{Y}_1 \times \mathbb{Y}_2$, and dynamics:

$$(\dot{x}_1, \dot{x}_2) = (f_1(x_1, (u_0, h_2(x_2))), f_2(x_2, (u_0, h_1(x_1)));$$
$$(y_1, y_2) = (h_1(x_1), h_2(x_2)).$$

For input-output maps M_1, M_2, define parallel composition as the restriction of behaviours:

$$M_1 \mid\mid M_2 : \mu_0 \mapsto \{(\eta_1, \eta_2) \mid \eta_1 \in M_1(\mu_0, \eta_2) \wedge \eta_2 \in M_2(\mu_0, \eta_1)\}.$$

These definitions satisfy the natural compatibility condition:

$$\mathcal{IO}[S_1 \mid\mid S_2] = \mathcal{IO}[S_1] \mid\mid \mathcal{IO}[S_2].$$

Abstraction and compositional reasoning

Ideally, we would like to analyse systems component-by-component in a state-space independent way.

The information provided by the input-output map as an element of the type

$$M : (\mathbb{R}^{[0,\infty)} \to \mathbb{U}) \to \mathcal{K}(\mathbb{R}^{[0,\infty)} \to \mathbb{Y})$$

is *insufficient* to prove safety and stabilisation properties!

The *abstraction* problem is to find a suitable representation of the input-output behaviour which

— is *independent* of the state space $\mathbb{X}$, and

— allows any (robust) temporal logic property to be verified.

The *reduction* problem is to find simplified state-space representations approximately preserving the input-output behaviour.

— To be useful for rigorous model-checking, error bounds for the approximation must be given.

— These should give better-quality approximations than general abstractions.

Regular expressions

For discrete-time finite input-output automata, the external behaviour is a set of infinite sequences of interleaved inputs and outputs $(y_0, u_0, y_1, u_1, y_2, \ldots)$.

The set of possible input/output sequences is an *ω-regular language* which can be described by a *regular expression*.

Conversely, an equivalent finite-state automaton can be reconstructed from the regular expression describing the external behaviour.

Regular expressions provide a canonical state-space independent way of describing the external behaviour.

Discretising systems

For continuous-time systems, we obtain a finite-state automaton description by discretising the state space, and spaces of input and output *functions*.

The state space $\mathbb{X}$ is discretised into finitely many (possibly overlapping) closed compact sets $\widehat{X}$.

Spaces of input and output signals $\mathbb{U}^{[0,T]}$ and $\mathbb{Y}^{[0,T]}$ are discretised into closed compact sets $\widehat{\mathcal{U}}$ and $\widehat{\mathcal{Y}}$ of Lipschitz functions.

Given $\hat{x}$ and $\hat{\mu}$, we can compute the set of solutions

$$\Xi(\hat{x},\hat{\mu}) = \{\xi \in \mathbb{X}^{[0,T]} \mid \exists\, \mu \in \hat{\mu},\ \xi \in \mathbb{X}^{[0,T]} :\ \dot{\xi} = f(\xi,\mu) \wedge \xi(0) \in \hat{x}\}.$$

We can then compute discretisations to

$$\widehat{H}(\hat{x},\hat{\mu}) = \{\hat{\eta} \in \widehat{\mathcal{Y}} \mid \exists\, \xi \in \Xi(\hat{x},\hat{\mu}) : h(\xi) \in \hat{\eta}\},$$
$$\widehat{F}(\hat{x},\hat{\mu}) = \{\hat{x} \in \widehat{X} \mid \exists\, \xi \in \Xi(\hat{x},\hat{\mu}) : \xi(T) \in \hat{x}\}.$$

The resulting system is a discrete automaton with states $\widehat{X}$, input space $\widehat{\mathcal{U}}$ and output space $\widehat{\mathcal{Y}}$, so can be described by a regular expression over $\widehat{\mathcal{U}}, \widehat{\mathcal{Y}}$!

Regular expression representation

The *regular expression representation* of an input-output system is a sequence of regular expressions R_i computed by successively finer discretisations $\widehat{\mathcal{U}}_i, \widehat{\mathcal{Y}}_i$ of $\mathbb{U}^{[0,T]}$ and $\mathbb{Y}^{[0,T]}$.

Theorem. *Let $S = (F, H, X_0)$ be an input-output system such that $F : \mathbb{X} \times \mathbb{U} \to \mathcal{K}(\mathbb{X})$, $H : \mathbb{X} \to \mathcal{K}(\mathbb{Y})$ and $X_0 \in \mathcal{K}(\mathbb{X})$. Then* any *property of S verifyable from (F, H, X_0) is also verifyable from a regular expression representation of the input-output map $\mathcal{IO}[S]$.*

In practise, we may not actually wish to use the regular expression representation to express the behaviour of a subsystem directly, but instead use a more expressive class of system such as timed automata.

System reduction

A detailed approach to subsystem modelling often yields a model containing effects which have little impact on the external behaviour.

The goal of system reduction is to create a simplified model with approximately the same input-output behaviour.

There are many ways of constructing approximate *reduced order* component models, which usually take into account special structures of the system.

All approximate the input-output system $\dot{x} = f(x, u), \quad y = h(x)$ by system with the same external variables, but state variables $\tilde{x} \in \widetilde{\mathbb{X}}$

$$\dot{\tilde{x}} = \tilde{f}(\tilde{x}, u); \qquad y = \tilde{h}(\tilde{x}).$$

Typically, x and $\tilde{x}$ are related by a projection $\tilde{x} = \pi(x)$ for $\pi : \mathbb{X} \to \widetilde{\mathbb{X}}$.

System reduction

For rigorous formal methods, we need to take into accout the errors introduced in the reduction.

Taking derivatives of $\tilde{x} = \pi(x)$ we find

$$\dot{\tilde{x}} \in \{\pi'(x) f(x,u) \mid x \in \pi^{-1}(\tilde{x})\}.$$

This gives a potentially unbounded set for $\dot{\tilde{x}}$!

To proceed, we need to *assume* that the actualy state lies in a subset A_X of X with compact fibres over $\tilde{x}$. Then

$$\dot{\tilde{x}} \in \{\pi'(x) f(x,u) \mid x \in \pi^{-1}(\tilde{x}) \cap A_x\} \in \mathcal{K}(\mathbb{R}^m).$$

The output is given by

$$y \in \widetilde{H}(\tilde{x}) := \{h(x) \mid x \in \pi^{-1}(\tilde{x}) \cap A_X\}.$$

The resulting system is a *differential inclusion* for the projection $\tilde{x}$.

System reduction

The main challenge is in finding an an appropriate reduction $\pi : \mathbb{X} \to \widetilde{\mathbb{X}}$.

- Finding a good reduction is highly dependent on the class of system,
- but can be done using existing approximative methods.

A typical approach is to find an (almost) invariant submanifold $\widetilde{X} = M \subset \mathbb{X}$.

- It is more convenient to work with a parametrisation of the manifold rather than a grid-based representation.
- It usually suffices to compute an *approximate* functional parametrisation.

It may be necessary to use different reductions for different parts of the state space X.

System reduction

A second major difficulty is that an attractor A_X for the unreduced state x needs to be found.

Whether a set A_X is attracting also depends on the input, so we typically need to assume that u lies in some compact set B_U.

If $A_X = \{x \mid a(x) \leq 0\}$, then A_X is invariant if $\nabla a(x) \cdot f(x, u) \leq 0$ whenever $a(x) = 0$ and $u \in B_U$.

Since $\widetilde{E}$ depends on A_X, the tighter A_X, the better the approximation.

An *assume-guarantee* approach may be used, where successively narrowing B_U allows for smaller errors in the reduced system.

- However, this introduces extra coupling between the subsystems.

System reduction

A third major problem is that although the reduced system may have fewer variables, the reduced system may be in a more challenging class.

Due to the need to consider error bounds, the new system is no longer a differential equation, but a nondeterministic *differential inclusion* $\dot{\tilde{x}} \in \widetilde{F}(\tilde{x}, u)$.

— The practical solution of differential inclusions is very difficult!

If different reductions are used, the resulting system is *hybrid*.

— Switching occurs if the state estimate $\tilde{x}$ leaves the domain of validity of a given reduction

— and possibly requiring re-inflation of the reduced state to full dimension.

— Switching from a reduced mode is also necessary if the input leaves its pre-specified bounds.

Implementation in ARIADNE

Information / Paradigms

Distinguish *information* levels or computational *paradigms*

`Exact`, `Effective`, `Validated`, `Approximate`.

`Effective` objects correspond to computable analysis.

— Defined by abstract *interfaces*.

— Main types for user inputs.

`Validated` objects are approximations with guaranteed error bound.

— Concrete classes used for rigorous numerical computation.

— Often either *balls* in a metric space or *intervals* in a partially-ordered space.

— User inputs which are subject to (experimental) errors.

`Approximate` objects have no accuracy guarantees.

— Concrete classes used for approximate or scratch computation.

— Useful in *preconditioning* or *hotstarting* rigorous methods.

`Exact` objects are discrete types or countable subtypes of continuous types.

— Used for algebraic computation where exact results are needed.

Numeric types

`Integer`, `Rational`: number types for exact arithmetic.

`Real`: number type for user input.

`Float`: raw data type as a base for numerical computation.

Supports only *rounded* arithmetic `add_down`, `add_up`, `add_near` etc.

- `ExactFloat`: Exactly representable number.
- `ValidatedFloat`: Guaranteed bounds (using interval arithmetic).
- `Lower/UpperFloat`: Guaranteed lower/upper bounds.
- `ApproximateFloat`: Approximation with no accuracy guarantees.

Double-precision `Float64`; also multiple-precision `FloatMP` in some versions.

Extraction of computational numbers:

- `Real::get(Validated)` -> `ValidatedFloat`.
- `Real::get(Approximate)` -> `ApproximateFloat`.

Functional calculus

Euclidean `Function` classes with `Effective`, `Validated` and `Approximate` information levels.

Arbitrary-order derivatives computed by automatic differentiation.

```
Function::derivatives(Number, Degree) -> Differential<Number>
```

Computations using scaled `PolynomialModels` on box domains.

```
PolynomialModel::evaluate(Number x) {
    return polynomial(unscale(x)) ± error; }
```

`Solver` classes for algebraic and differential equations, nonlinear programming and constraint propagation.

— `Integrator::flow(VectorFunction f, Box X0, Real T) -> VectorFunction;`

Geometric calculus

Fundamental `Interval` and `Box` sets.

Abstract classes `OpenSet`, `CompactSet` etc.

— `OpenSet::covers(Box) -> Sierpinski`.

Concrete sets represented via functions and boxes:

— `ConstraintSet(VectorFunction g, Box c)` is $\{x \mid g(x) \in C\}$

— `ConstrainedImageSet(VectorFunction f, Box d, VectorFunction g, Box c)` is $\{f(x) \mid x \in D \wedge g(x) \in C\}$

Geometric predicates `intersection`, `subset`/`inside` (using nonlinear programming and constraint propagation).

Discretised sets such as `GridTreeSet` and abstract `Pavings`.

Open issues

Hybrid system evolution very complicated; needs simplification.

Slow compile times: Clean-up headers and use precompilation.

Much code duplication: refactor analytic functions using of `Field` and `Algebra` concepts.

Testing is a major challenge since semantics is nondeterministic.

- The specificion of a validated flow tube $\hat{\phi}$ is $\hat{\phi}(x,t) \ni \phi(x,t)$ for all x, t.
- A good algorithm will compute $\hat{\phi}$ tight to machine precision.
- How can we verify the enclosure property??
  Test should use a simple algorithm which is easily seen to be correct, but also be high-precision and efficient.

Maybe need to use theorem-provers and/or code-validaters to demonstrate correctness...

Cardiac Myocyte Models

Electrophysiology of cardiac myocytes

The electrophysiology of cardiac myocytes provides an interesting testing ground for the development of numerical implementations of the operations described in this paper.

The *membrane potential* of a cell is determined by the transport of charged Na^+, K^+, Ca^{2+} and (to a lesser extent) Cl^- ions through *channels* in the cell membrane.

The opening and closing of channels is governed by a number of factors, notably the membrane potential itself, and by signalling molecules.

We would like to prove that the membrane potential of a myocite model does not exhibit any dangerous arrhythmias, such as *early afterdepolarisations*.

Hodgkin-Huxley equations and Markov models

The basic equation for the current through a channel of type i is

$$I_i(t) = g_i(t)\ (V(t) - E_i)$$

where g_i is a *gating* variable and E_i is the reverse potential of the channel.

A simple gating model is the *voltage-gating* model of Hodgkin and Huxley (1952).

A similar model is the *slow-inward sodium current* used in a simple model of Tran et al. (2009), and is given by

$$I(t) = \bar{g} \cdot d(t) \cdot f(t) \cdot (V(t) - E)$$

where the gating variables satisfy

$$\dot{d} = \alpha_d(V) \cdot (1 - d) - \beta_d(V) \cdot d; \qquad \dot{f} = \alpha_f(V) \cdot (1 - f) - \beta_f(V) \cdot f$$

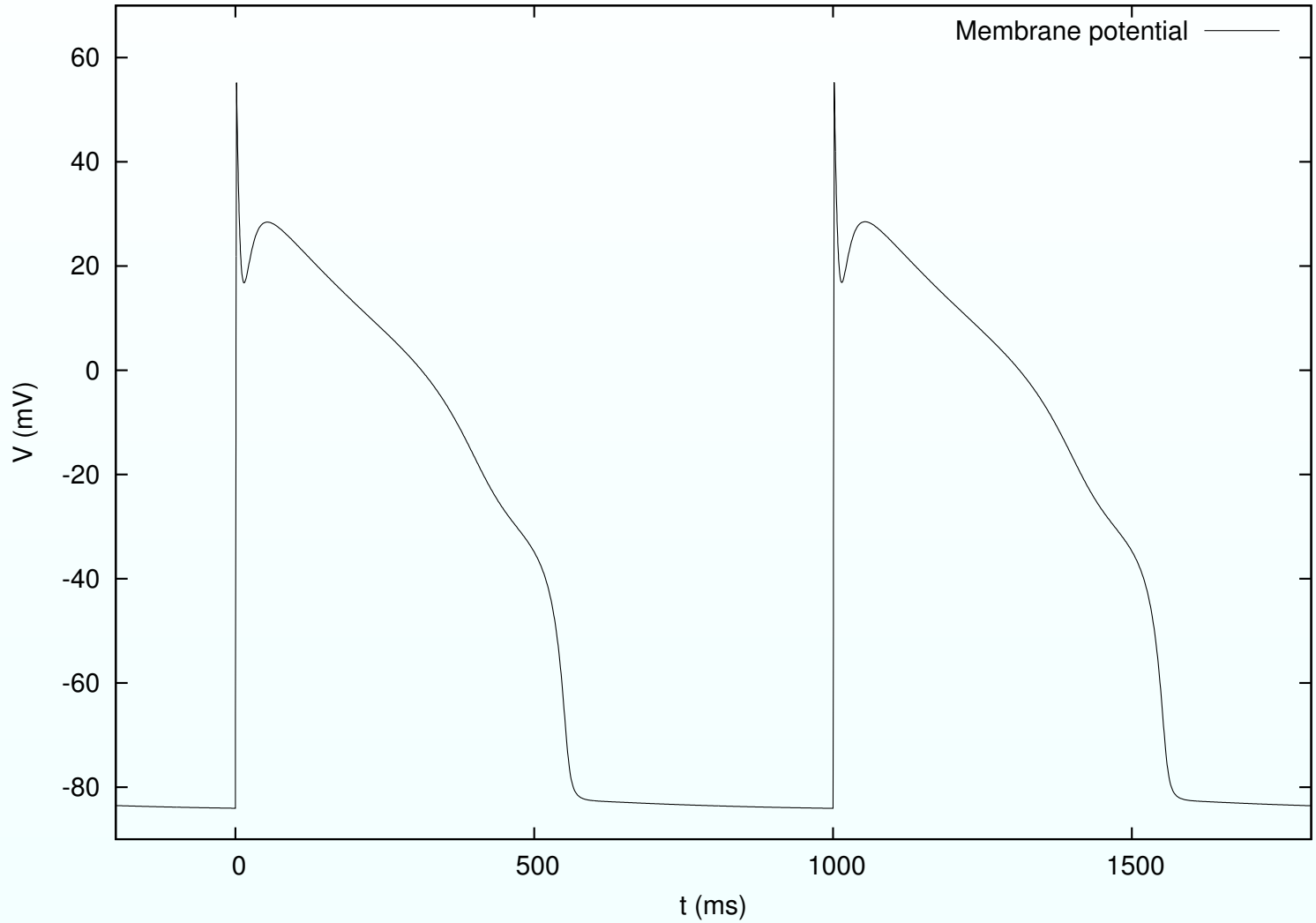
and the functions α_d, β_d, α_f and β_f are sigmoidal functions depending only on V.

More complicated models consider the mechanism of channel opening and closing, described by a Markov model with multiple state variables.

A detailed electrophysiological model is given by Heijman et al. (2011).

Membrane potential

The membrane potential for the Tran (2009) model is shown below computed using the Maastrict Mycyte Toolkit (Clerx).



Rigorous analysis

Even though the single-cell model of Tran (2009) only has four state variable, analysis by rigorous methods is extremely challenging.

The system exhibits multiple time-scales, with rapid depolarisation due to the action potential followed by a plateau, and later depolarisation.

- Explicit methods for differential equations need to take small steps,
- We were unable to integrate the system using Taylor-series based integration methods with our tool ARIADNE.

Dealing with the multiple time scales is likely to be a major challenge, and require specialised methods for stiff systems.

- The usual approach is to use implicit methods, which are currently not available for rigorous integrators.
- An alternative approach is to (rigorously) approximate the system by a hybrid system.

Challenges

Provide characterisations of arrhythmias in terms of temporal logic.

— This will require close collaboration with medical experts.

The form of the gating functions $\alpha_{d,f}, \beta_{d,f}$ in the model are complicated and make extensive use of exponential functions.

— We have considered simplifying the sigmoidal functions used for the gating by using splines.

Model-order reduction of more complicated Markov gating models:

— We expect the behaviour of these models to be very well-described by a model with many fewer variables.

Challenges

Comparison of different models for the same ion channel.

— Since many different models exist in the literature for the same ion channel, we would like to compare the input-output behaviour, ideally as described by discretisation and regular expressions.

Sensitivity analysis with respect to parameter uncertainty.

— Models parameters are hard to determine experimentally, many models are over-determined, and different cells have different characteristics. Careful analysis should take into account uncertainties in the model.

Moving from single-cell analysis to tissue and whole-heart analysis

— Will require a combination of reduction and abstraction techniques,

— Possibly including rigorous derivation of partial differential equations models.

Conclusions

Summary

We have described a framework for model-checking the complex, multi-component, highly uncertain systems arising in systems biology.

We have shown that temporal logic properties *are* checkable for a wide class of systems!

— Computability requires appropriate data representations and semantics of operations.

— Nondeterministic and stochastic systems can be studied.

We have sketched some ideas for the rigorous analysis of large-scale systems based on abstraction and model-order reduction.

Lots to do!

The development of practical general and efficient implementations of most of the methods described here is a much greater challenge.
While basic methods for the analysis of well-conditioned deterministic systems exist, tooling for many of the more sophisticated algorithms still needs to be developed.
A good base for this work are existing packages for rigorous numerical methods and model checking, notably GAIO, d/dt, Cosy Infinity, VNode, CAPD Library, HSOLVER, ARIADNE, Phaver, FlowStar.

References

[Col07] Pieter Collins, *Optimal semicomputable approximations to reachable and invariant sets*, Theory Comput. Syst. **41** (2007), no. 1, 33–48.

[CZ11] Pieter Collins and Ivan S. Zapreev, *Computable semantics for CTL* on discrete-time and continuous-space dynamic systems*, Int. J. Found. Comput. Sci. **22** (2011), no. 4, 801–821.