EECS 144/244: System Modeling, Analysis, and Optimization

Continuous Systems

Lecture: Nonlinear Systems

Alexandre Donzé

University of California, Berkeley

April 5, 2013

Nonlinear Systems Definition (?)

- Formally, a system which is not linear.

Nonlinear Systems Definition (?)

- Formally, a system which is not linear.
  Go figure. About everything in the world is nonlinear somehow

Nonlinear Systems Definition (?)

- Formally, a system which is not linear.
  Go figure. About everything in the world is nonlinear somehow

- Implicitly, a difficult, complex problem/system for which linear tools do not apply

Nonlinear Systems Definition (?)

- Formally, a system which is not linear.
  Go figure. About everything in the world is nonlinear somehow

- Implicitly, a difficult, complex problem/system for which linear tools do not apply

"The technique works for nonlinear systems"

Nonlinear Systems Definition (?)

- Formally, a system which is not linear.
  Go figure. About everything in the world is nonlinear somehow

- Implicitly, a difficult, complex problem/system for which linear tools do not apply

"The technique works for nonlinear systems"

often reads

"The technique has nothing special, but it kind of works for this specific/trivial/artificial yet **nonlinear** *system"'*

Nonlinear is Cool

Nonlinear is Cool

- It's difficult and challenging !

Nonlinear is Cool

- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)

Nonlinear is Cool

- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)
- It's chaos !

Nonlinear is Cool

- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)
- It's chaos ! (yeah this neither is not at all well defined...)

Nonlinear is Cool

- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)
- It's chaos ! (yeah this neither is not at all well defined...)
- It's beautiful !

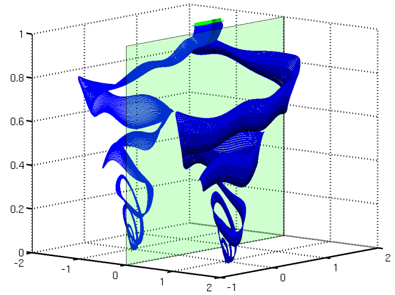
Nonlinear is Cool

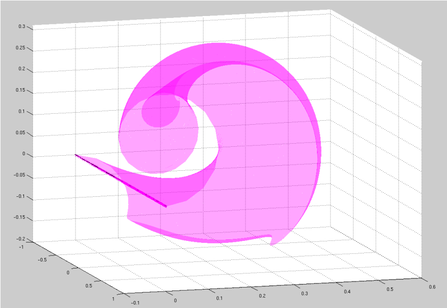
- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)
- It's chaos ! (yeah this neither is not at all well defined...)
- It's beautiful ! (although often beautif-useless)

Nonlinear is Cool

- It's difficult and challenging ! (even when it's not, you can pretend it is, because it is **nonlinear**)
- It's chaos ! (yeah this neither is not at all well defined...)
- It's beautiful ! (although often beautif-useless)

Here are two contributions of my own:





Definition (?)

A slightly more useful still implicit definition:

a system with "nonlinearities"

Definition (?)

A slightly more useful still implicit definition:

a system with "nonlinearities"

I.e., a system which main dynamics is linear but with additional nonlinear features, e.g.:

- saturations
- discontinuities (e.g.: switch in circuits),
- A delay $x(t) \to x(t - \tau)$
- An additive term, e.g. : $\dot{\mathbf{x}} = A\mathbf{x} + \psi(\mathbf{x})$ for some nonlinear function $\psi(\mathbf{x})$

Definition (?)

A slightly more useful still implicit definition:

a system with "nonlinearities"

I.e., a system which main dynamics is linear but with additional nonlinear features, e.g.:

- saturations
- discontinuities (e.g.: switch in circuits),
- A delay $x(t) \to x(t - \tau)$
- An additive term, e.g. : $\dot{\mathbf{x}} = A\mathbf{x} + \psi(\mathbf{x})$ for some nonlinear function $\psi(\mathbf{x})$

Note

- In the next two lectures, we will discuss a special class of nonlinear systems: hybrid systems.
- When the above does not apply, some talk of *highly nonlinear systems* ...

Nonlinear Differential Equations

A “definition” I’ve seen recently:

Nonlinear Differential Equations

A “definition” I’ve seen recently:

A nonlinear system is a system of the form $\dot{\mathbf{x}} = \varphi(t, \mathbf{x})$ *where* f *is Lipshitz continuous.*

Nonlinear Differential Equations

A “definition” I’ve seen recently:

A nonlinear system is a system of the form $\dot{\mathbf{x}} = \varphi(t, \mathbf{x})$ *where* f *is Lipshitz continuous.*

But wait, then technically, linear systems are nonlinear systems ... !

Not (completely) as silly as it sounds: allows to test/benchmark a general method against linear methods

Nonlinear Differential Equations

A “definition” I’ve seen recently:

A nonlinear system is a system of the form $\dot{\mathbf{x}} = \varphi(t, \mathbf{x})$ *where* f *is Lipshitz continuous.*

But wait, then technically, linear systems are nonlinear systems ... !

Not (completely) as silly as it sounds: allows to test/benchmark a general method against linear methods

Also, I would agree that this is the most commonly understood **informal** definition in dynamical systems theory.

Steady-State Analysis

Assume time-invariant dynamics $\dot{\mathbf{x}} = f(\mathbf{x})$.

A state $\mathbf{x}_e$ such that $f(\mathbf{x}_e) = 0$ is call an equilbrium state.

Linear systems $\dot{\mathbf{x}} = A\mathbf{x}$ have only equilibrium point, $\mathbf{x}_e = 0$, and the behaviors around it are well characterized

Nonlinear systems may have an arbitrary number of equilibriums.

Finding Steady States

To find potential equilibrium states, one has to solve the nonlinear equation:

$$f(\mathbf{x}) = 0$$

Finding Steady States

To find potential equilibrium states, one has to solve the nonlinear equation:

$$f(\mathbf{x}) = 0$$

(Note again that the fact that f is nonlinear is almost completely irrelevant to assess the difficulty of this problem.)

The Newton-Raphson Method

Iterative numerical algorithm to solve $f(\mathbf{x}) = 0$

1. Start with some guess of the solution
2. Repeat
   - Check if current guess is good enough
   - If not, improve it

The Newton-Raphson Method

Iterative numerical algorithm to solve $f(\mathbf{x}) = 0$

1. Start with some guess of the solution
2. Repeat
   - Check if current guess is good enough
   - If not, improve it

To improve the estimate, the algorithm make use of:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

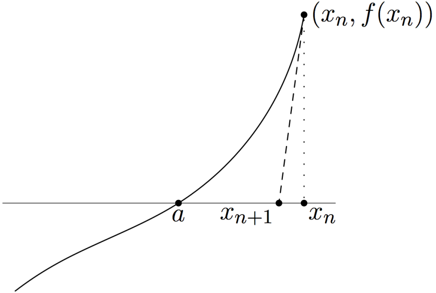
The Newton-Raphson Method

Iterative numerical algorithm to solve $f(\mathbf{x}) = 0$

1. Start with some guess of the solution
2. Repeat
   - Check if current guess is good enough
   - If not, improve it

To improve the estimate, the algorithm make use of:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



The Newton-Raphson Algorithm

The method generalizes to vector functions using Jacobian :

$$J_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

The Newton-Raphson Algorithm

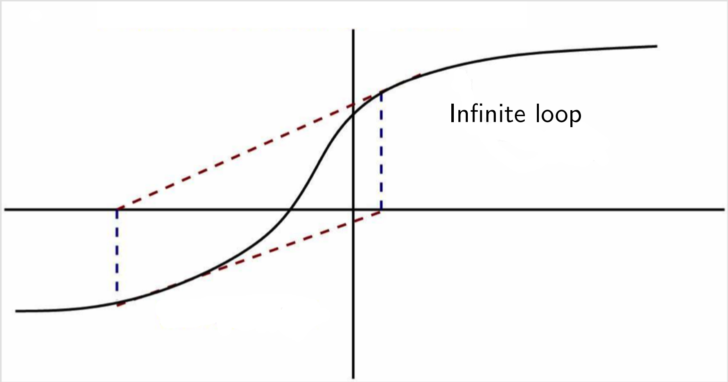
The method generalizes to vector functions using Jacobian :

$$J_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Start with initial guess $\mathbf{x}_0$, $i = 0$
repeat
  Compute jacobian $J_i = J_f(\mathbf{x}_i)$
  Let $\delta\mathbf{x} = J_i^{-1} f(\mathbf{x})$
  Update guess $\mathbf{x}_{i+1} = \mathbf{x}_i + \delta\mathbf{x}$
until Convergence or max number iterations

Convergence

Not guaranteed :



Conditions for convergence (unfortunately, no general method to enforce them):

- f has to be smooth (continuous, differentiable)
- initial guess has to be "close" to a solution

Convergence rate

When it converges, *quadratic*.

Let $f(\mathbf{x}^*) = 0$ and $err_i = \|\mathbf{x}_i - \mathbf{x}^*\|$. If

1. f is smooth
2. $J_f(\mathbf{x}^*) \neq 0$
3. $\|\mathbf{x}_0 - \mathbf{x}^*\|$ is small enough

Then there is a constant C such that $err_{i+1} \leq C err_i^2$

Stopping criterions

- On $\mathbf{x}_i$ using relative and absolute tolerances :

$$\text{Stops when } \|\mathbf{x}_{i+1} - \mathbf{x}_i\| \leq \varepsilon_{\mathsf{abs}} + \varepsilon_{\mathsf{rel}} \|\mathbf{x}_i\|$$

- or on the residual $\|f(\mathbf{x}_i)\|$.

$$\text{Stops when } \|f(\mathbf{x}_i)\| \leq \varepsilon_{\mathsf{abs}}$$

- or some combination... Ultimately, specific tuning to your function

Note

- $\varepsilon_{\mathsf{rel}}$ is typically between 10^{-3} and 10^{-6}
- $\varepsilon_{\mathsf{abs}}$ between 10^{-9} and 10^{-12} or **small with respect to typical values of** x
- This applies to tolerances for ODE solver as well

Linearization

Once an equilibrium state has been found, one can study locally the behavior of the system

Linearization

Once an equilibrium state has been found, one can study locally the behavior of the system

$$\dot{\mathbf{x}} = F(\mathbf{x}_e) + \left.\frac{\partial F}{\partial \mathbf{x}}\right|_{\mathbf{x}_e} (\mathbf{x} - \mathbf{x}_e) + \text{higher order terms in } (\mathbf{x} - \mathbf{x}_e).$$

Linearization

Once an equilibrium state has been found, one can study locally the behavior of the system

$$\dot{\mathbf{x}} = F(\mathbf{x}_e) + \left.\frac{\partial F}{\partial \mathbf{x}}\right|_{\mathbf{x}_e} (\mathbf{x} - \mathbf{x}_e) + \text{higher order terms in } (\mathbf{x} - \mathbf{x}_e).$$

More generally, the system

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) \qquad && \mathbf{x} \in R^n, \mathbf{u} \in \mathcal{R} \\ \mathbf{y} &= h(\mathbf{x}, \mathbf{u}) \qquad && \mathbf{y} \in \mathcal{R} \end{aligned}$$

can be linearized about an equibrium point $\mathbf{x} = \mathbf{x}_e$,$\mathbf{u} = \mathbf{u}_e$,$\mathbf{y} = \mathbf{y}_e$, by defining new variables:

$$\mathbf{z} = \mathbf{x} - \mathbf{x}_e \qquad \mathbf{v} = \mathbf{u} - \mathbf{u}_e \qquad \mathbf{w} = \mathbf{y} - h(\mathbf{x}_e, \mathbf{u}_e)$$

Linearization (cont'd)

The dynamics of the system near the equilibrium point can then be approximated by the linear system

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$$
$$y = C\mathbf{x} + D\mathbf{u}$$

where

$$A = \left.\frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right|_{\mathbf{x}_e,\mathbf{u}_e} \qquad B = \left.\frac{\partial f(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}_e,\mathbf{u}_e}$$
$$C = \left.\frac{\partial h(\mathbf{x},\mathbf{u})}{\partial \mathbf{x}}\right|_{\mathbf{x}_e,\mathbf{u}_e} \qquad D = \left.\frac{\partial y(\mathbf{x},\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}_e,\mathbf{u}_e}$$

Stability Analysis

An equilibrium point is (locally) stable if initial conditions that start near an equilibrium point stay near that equilibrium point.

A equilibrium point is (locally) asymptotically stable if it is stable and the state of the system converges to the equilibrium point as time increases.

Note

- Stability for nonlinear systems is a *local* property
- Depending on initial conditions, the system can converge to different equilibriums (see "Bi-stability", regions of attraction, etc)

Lyapunov Stability

A Lyapunov function is an energy-like function $V : \mathcal{R}^n \rightarrow \mathcal{R}$ that can be used to reason about the stability of an equilibrium point.

Lyapunov Stability

A Lyapunov function is an energy-like function $V : \mathcal{R}^n \rightarrow \mathcal{R}$ that can be used to reason about the stability of an equilibrium point.

We define the derivative of V along the trajectory of the system as

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}}\dot{\mathbf{x}} = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x})$$

Lyapunov Stability

A Lyapunov function is an energy-like function $V : \mathcal{R}^n \to \mathcal{R}$ that can be used to reason about the stability of an equilibrium point.

We define the derivative of V along the trajectory of the system as

$$\dot{V}(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} \dot{\mathbf{x}} = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x})$$

Assuming $\mathbf{x}_e = 0$ and $V(0) = 0$

Condition on V	Condition on $\dot{V}$	Stability
$V(\mathbf{x}) > 0,\ \mathbf{x} \neq 0$	$\dot{V}(\mathbf{x}) \leq 0$ for all $\mathbf{x}$	$\mathbf{x}_e$ is stable
$V(\mathbf{x}) > 0,\ \mathbf{x} \neq 0$	$\dot{V}(\mathbf{x}) < 0$, $\mathbf{x} \neq 0$	$\mathbf{x}_e$ is asymptotically stable

Finding a Lyapunov function is a difficult problem in general - systematic method exist mostly for linear systems.

Idea: characterizing behaviors of dynamical systems in a formalized way

Idea: characterizing behaviors of dynamical systems in a formalized way

In the following, I will be using Breach, a matlab toolbox available at

`www.eecs.berkeley.edu/~donze/breach_page.html`

It allows to (among other things)

- Simulate ODEs and Simulink systems
- Define Signal Temporal Logic properties
- Verify them on simulations results

Outline

Outline

Temporal logics in a nutshell

Temporal logics allow to specify patterns that timed behaviors of systems may or may not satisfy. They come in many flavors.

One of the most common is Linear Temporal Logic (LTL), dealing with discrete sequences of states.

Based on logic operators ($\neg$, $\wedge$, $\vee$) and temporal operators: "next", "always" ($\Box$), "eventually" ($\Diamond$) and "until" ($\mathcal{U}$)

Examples:

- $\varphi\ \varphi\ \varphi\ \varphi\ \cdots$ satisfies $\Box\,\varphi$
- $\psi\ \psi\ \psi\ \varphi\ \psi\ \cdots$ satisfies $\Diamond\,\varphi$
- $\varphi\ \varphi\ \varphi\ \varphi\ \psi\ \cdots$ satisfies $\varphi\ \mathcal{U}\ \psi$

From Discrete to Continuous

Temporal logics developed for discrete systems

Why not discretize time and space and reuse existing logics and tools ?

From Discrete to Continuous

Temporal logics developed for discrete systems

Why not discretize time and space and reuse existing logics and tools ?

Some reasons:

- A priori arbitrary discretization often leads either to state-explosion or too coarse approximation
- Specifications should not depend on the discretization used (e.g., “next” depends on time step)

From Discrete to Continuous

Temporal logics developed for discrete systems

Why not discretize time and space and reuse existing logics and tools ?

Some reasons:

- A priori arbitrary discretization often leads either to state-explosion or too coarse approximation
- Specifications should not depend on the discretization used (e.g., “next” depends on time step)

Thus we need:

- Temporal specifications involving dense-time intervals
- Constraints applying on variable in the continuous domain

Formal Definitions

Definition (STL Syntax)

$$\varphi := \mu \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \ \mathcal{U}_{[a,b]} \ \psi$$

where μ is a predicate of the form $\mu : \mu(x) > 0$

Formal Definitions

Definition (STL Syntax)

$$\varphi := \mu \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \ \mathcal{U}_{[a,b]} \ \psi$$

where μ is a predicate of the form $\mu : \mu(x) > 0$

Definition (STL Semantics)

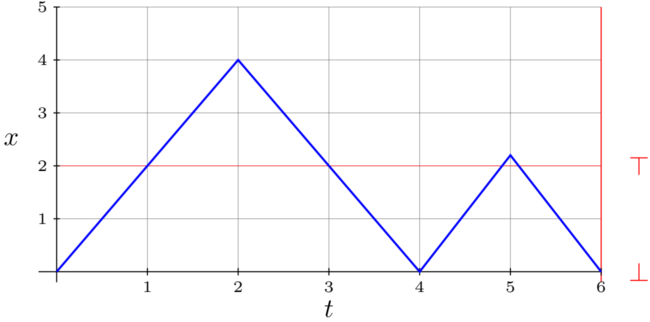
The validity of a formula φ with respect to a signal x at time t is

$$\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \varphi \wedge \psi & \Leftrightarrow & (x,t) \models \varphi \wedge (x,t) \models \psi \\
(x,t) \models \neg\varphi & \Leftrightarrow & \neg((x,t) \models \varphi) \\
(x,t) \models \varphi \, \mathcal{U}_{[a,b]} \ \psi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \psi \ \wedge \\
 & & \qquad \forall t'' \in [t,t'],\ (x,t'') \models \varphi \}
\end{array}$$

Additionally: $\Diamond_{[a,b]}\varphi = \top \, \mathcal{U}_{[a,b]} \ \varphi$ and $\Box_{[a,b]}\varphi = \varphi \, \mathcal{U}_{[a,b]} \ \bot$.

Examples

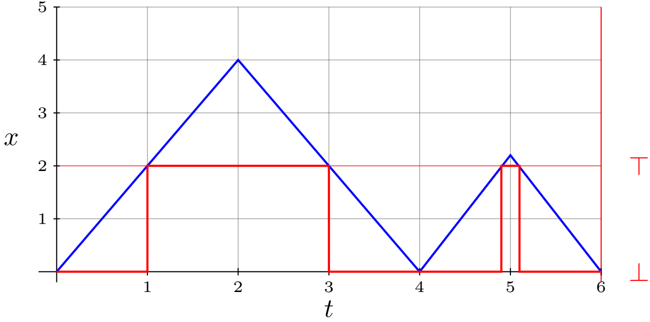
Consider a simple piecewise-affine signal:



Truth value of :

Examples

Consider a simple piecewise-affine signal:

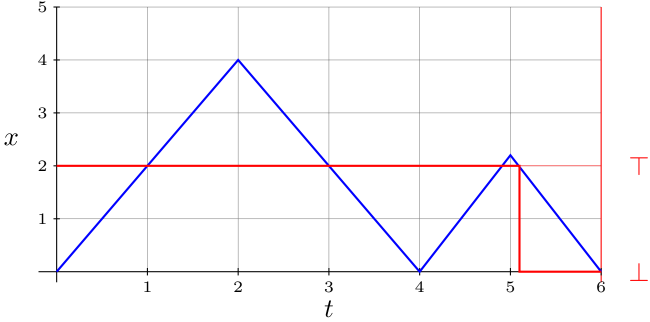


Truth value of :

- $\varphi = x > 2$

Examples

Consider a simple piecewise-affine signal:

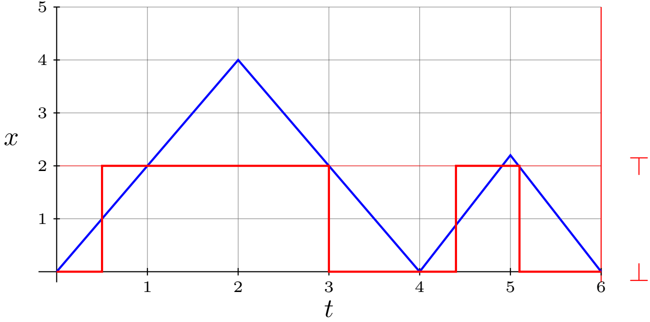


Truth value of :

- $\varphi = \Diamond_{[0,\infty]}(x > 2)$

Examples

Consider a simple piecewise-affine signal:

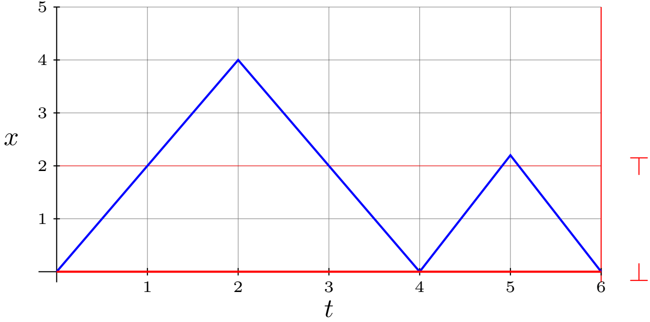


Truth value of :

- $\varphi = \Diamond_{[0,.5]}(x > 2)$

Examples

Consider a simple piecewise-affine signal:

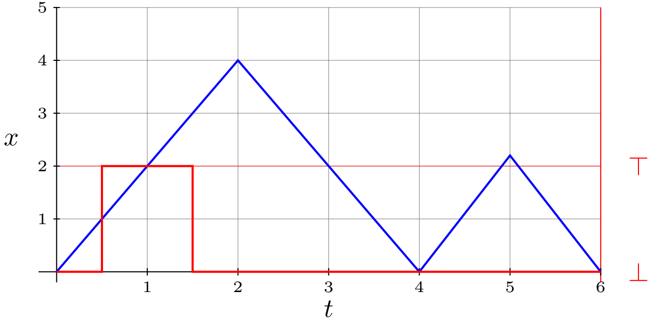


Truth value of :

- $\varphi = \Box_{[0,\infty]}(x > 2)$

Examples

Consider a simple piecewise-affine signal:



Truth value of :

- $\varphi = \Box_{[0.5,1.5]}(x > 2)$

Outline

1. Definitions (?)

2. Steady-state Analysis

3. Temporal Logics for Continuous Systems
   - Signal Temporal Logic
   - Quantitative Semantics of STL

4. Applications
   - Voltage Controlled Oscillator
   - Systems Biology

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{ and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}
$$

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \,\mathcal{U}_{[a,b]}\varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}
$$

A Boolean Satisfaction Function χ

Map $\{\text{false}, \text{true}\}$ to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \rightarrow \{-\infty, \infty\}$:

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}
$$

A Boolean Satisfaction Function χ

Map $\{\text{false}, \text{true}\}$ to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \to \{-\infty, \infty\}$:

$$\chi(\mu, x, t) = \text{sign}(\mu(x[t])) \times \infty$$

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}
$$

A Boolean Satisfaction Function χ

Map $\{\text{false}, \text{true}\}$ to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \rightarrow \{-\infty, \infty\}$:

$$
\begin{array}{lcl}
\chi(\mu, x, t) & = & \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg\varphi, x, t) & = & -\chi(\varphi, x, t)
\end{array}
$$

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \, \mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}
$$

A Boolean Satisfaction Function χ

Map {false, true} to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \rightarrow \{-\infty, \infty\}$:

$$
\begin{array}{lcl}
\chi(\mu, x, t) & = & \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg\varphi, x, t) & = & -\chi(\varphi, x, t) \\
\chi(\varphi_1 \wedge \varphi_2, x, t) & = & \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t))
\end{array}
$$

From Semantics to Satisfaction Functions

STL semantics

$$\begin{array}{lcl}
(x,t) \models \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \models \neg\varphi & \Leftrightarrow & (x,t) \not\models \varphi \\
(x,t) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \models \varphi_1 \text{ and } (x,t) \models \varphi_2 \\
(x,t) \models \varphi_1 \;\mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \models \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \models \varphi_1
\end{array}$$

A Boolean Satisfaction Function χ

Map {false, true} to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \to \{-\infty, \infty\}$:

$$\begin{array}{lcl}
\chi(\mu, x, t) & = & \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg\varphi, x, t) & = & -\,\chi(\varphi, x, t) \\
\chi(\varphi_1 \wedge \varphi_2, x, t) & = & \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t)) \\
\chi(\varphi_1 \;\mathcal{U}_{[a,b]} \varphi_2, x, t) & = & \max\limits_{\tau \in t+[a,b]} (\min(\chi(\varphi_2, x, \tau), \min\limits_{s \in [t,\tau]} \chi(\varphi_1, x, s))
\end{array}$$

From Semantics to Satisfaction Functions

STL semantics

$$
\begin{array}{lcl}
(x,t) \vDash \mu & \Leftrightarrow & \mu(x[t]) > 0 \\
(x,t) \vDash \neg\varphi & \Leftrightarrow & (x,t) \nvDash \varphi \\
(x,t) \vDash \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (x,t) \vDash \varphi_1 \text{ and } (x,t) \vDash \varphi_2 \\
(x,t) \vDash \varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (x,t') \vDash \varphi_2 \\
 & & \text{and } \forall t'' \in [t,t'], (x,t'') \vDash \varphi_1
\end{array}
$$

A Boolean Satisfaction Function χ

Map $\{\text{false}, \text{true}\}$ to $\{-\infty, \infty\}$ and define the function $\chi : (x,t) \to \{-\infty, \infty\}$:

$$
\begin{array}{lcl}
\chi(\mu, x, t) & = & \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg\varphi, x, t) & = & -\chi(\varphi, x, t) \\
\chi(\varphi_1 \wedge \varphi_2, x, t) & = & \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t)) \\
\chi(\varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2, x, t) & = & \max\limits_{\tau \in t+[a,b]} (\min(\chi(\varphi_2, x, \tau), \min\limits_{s \in [t,\tau]} \chi(\varphi_1, x, s))
\end{array}
$$

We can verify that $(x,t) \models \varphi \Leftrightarrow \chi(\varphi, x, t) = +\infty$

From Boolean to Quantitative Satisfaction Function

For atomic predicates:

$$\chi(\mu, x, t) = \mathsf{sign}(\mu(x[t])) \times \infty$$

The sign removes the quantitative information in μ to get a boolean signal

From Boolean to Quantitative Satisfaction Function

For atomic predicates:

$$\chi(\mu, x, t) = \mathsf{sign}(\mu(x[t])) \times \infty$$

The sign removes the quantitative information in μ to get a boolean signal

Simple idea

- Get rid of sign and ∞ to get a quantitative satisfaction function ρ

From Boolean to Quantitative Satisfaction Function

For atomic predicates:

$$\chi(\mu, x, t) = \text{sign}(\mu(x[t])) \times \infty$$

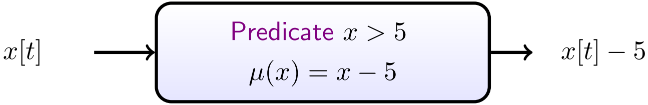
The sign removes the quantitative information in μ to get a boolean signal

Simple idea

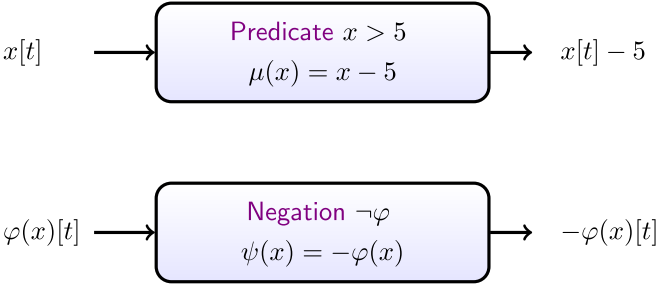
- Get rid of sign and ∞ to get a quantitative satisfaction function ρ
- Keep the same inductive rules for the quantitative semantics:

$$\begin{aligned}
\rho(\mu, x, t) &= \mu(x[t]) \\
\rho(\neg\varphi, x, t) &= -\rho(\varphi, x, t) \\
\rho(\varphi_1 \wedge \varphi_2, x, t) &= \min(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)) \\
\rho(\varphi_1 \ \mathcal{U}_{[a,b]} \varphi_2, x, t) &= \max_{\tau \in t+[a,b]} (\min(\rho(\varphi_2, x, \tau), \min_{s \in [t,\tau]} \rho(\varphi_1, x, s))
\end{aligned}$$

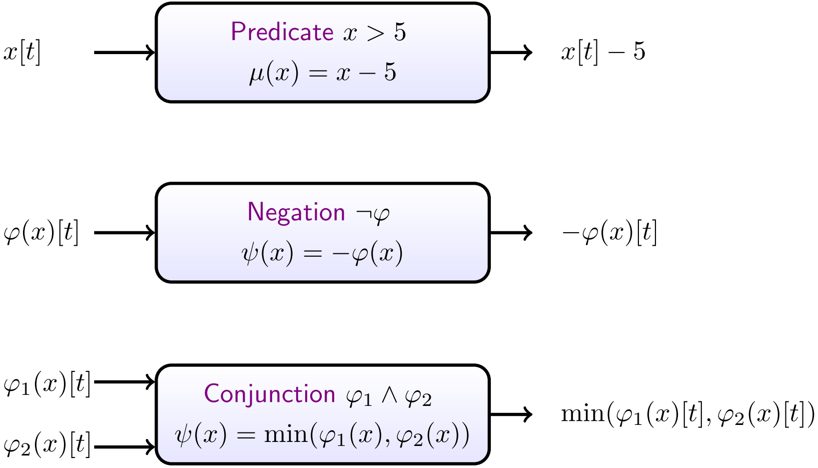
STL operators as systems



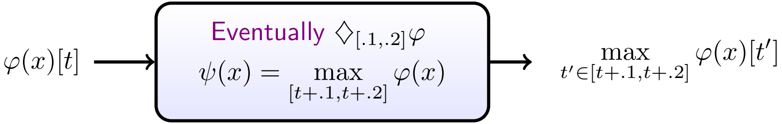
STL operators as systems



STL operators as systems



STL operators as systems

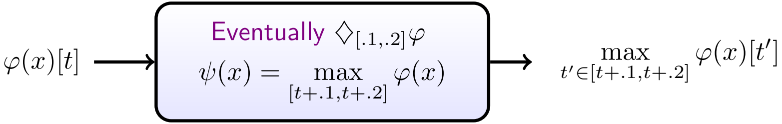


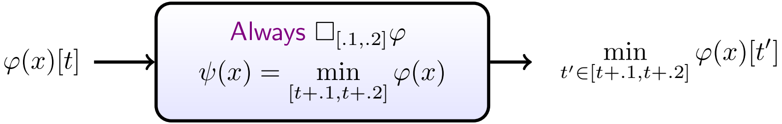
STL operators as systems

$$\varphi(x)[t] \longrightarrow \boxed{\begin{array}{c}\text{Eventually } \Diamond_{[.1,.2]}\varphi \\ \psi(x) = \max\limits_{[t+.1,t+.2]} \varphi(x)\end{array}} \longrightarrow \max_{t' \in [t+.1,t+.2]} \varphi(x)[t']$$

$$\varphi(x)[t] \longrightarrow \boxed{\begin{array}{c}\text{Always } \Box_{[.1,.2]}\varphi \\ \psi(x) = \min\limits_{[t+.1,t+.2]} \varphi(x)\end{array}} \longrightarrow \min_{t' \in [t+.1,t+.2]} \varphi(x)[t']$$

STL operators as systems

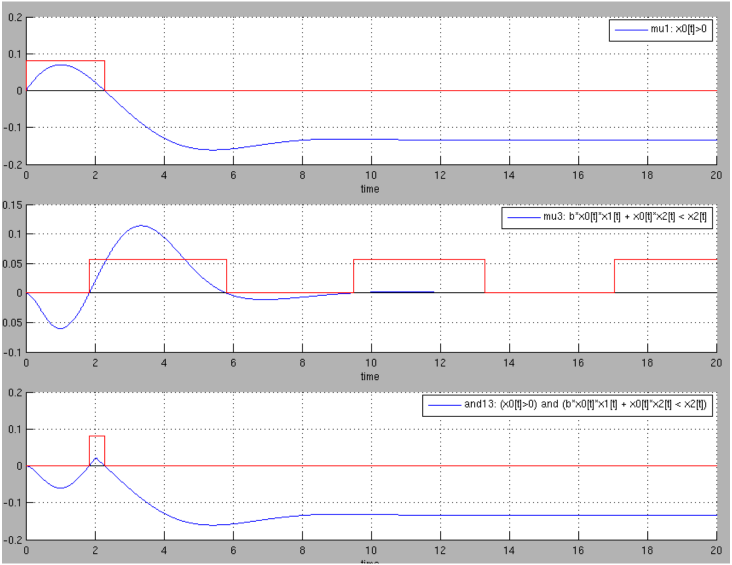




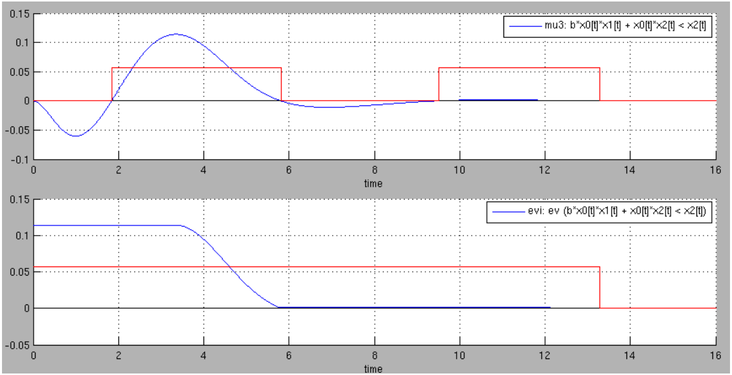
Note

- For the until operator $\mathcal{U}$, we get some min-max combination of φ_1 and φ_2
- $\Diamond$ and $\Box$ are actually deduced from $\mathcal{U}$

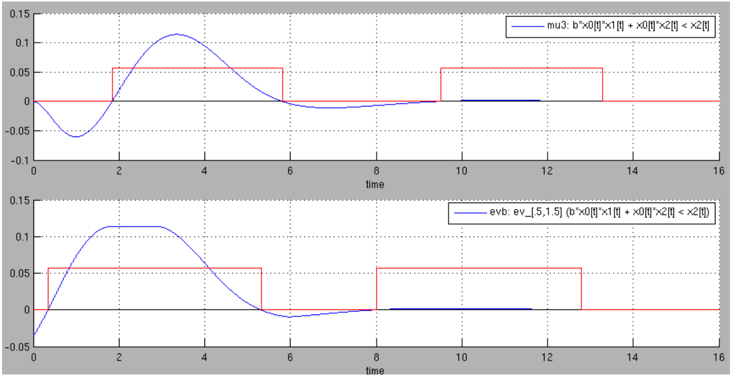
Robust Satisfaction, Examples



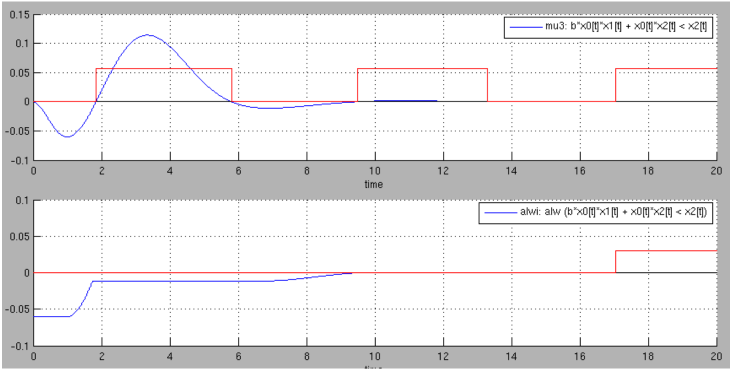
Robust Satisfaction, Examples



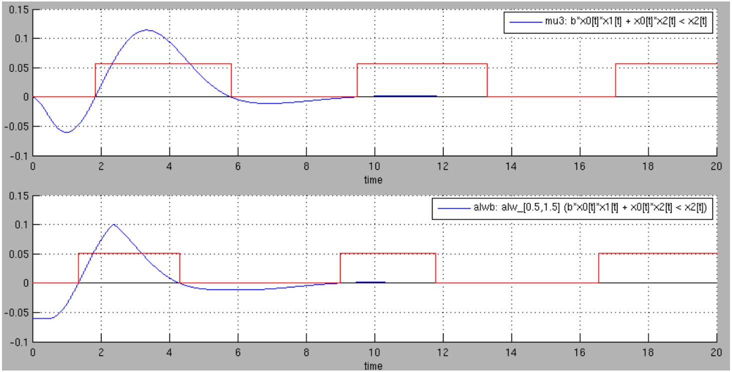
Robust Satisfaction, Examples



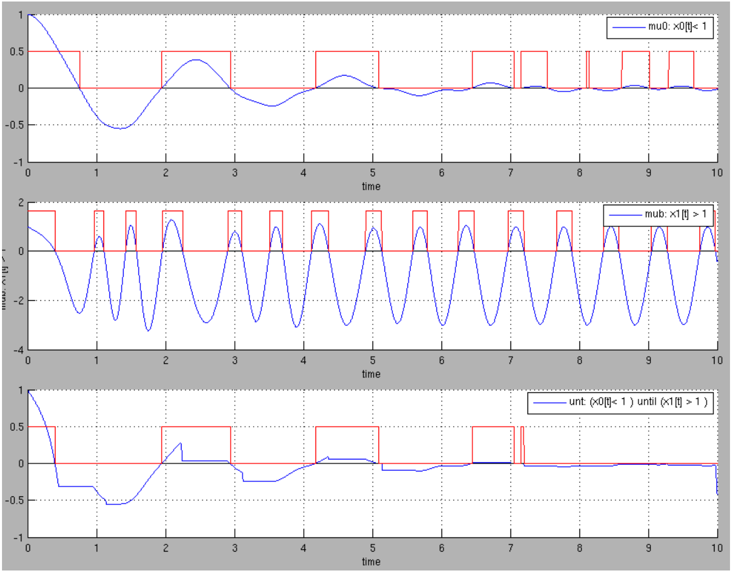
Robust Satisfaction, Examples



Robust Satisfaction, Examples

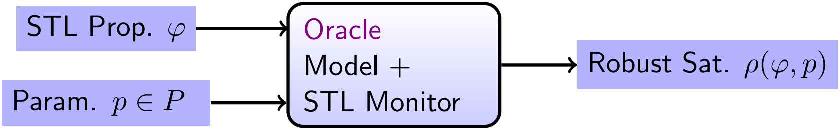


Robust Satisfaction, Examples



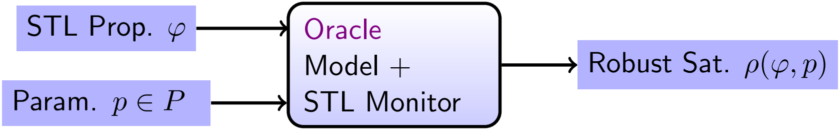
Robust Satisfaction, Applications

Assume that x depends on p, we get the following oracle:



Robust Satisfaction, Applications

Assume that x depends on p, we get the following oracle:



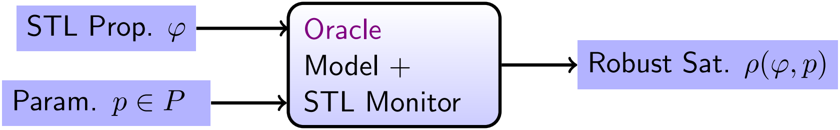
Parameter synthesis can be solved by solving

$$p^* = \max \{\rho(\varphi, p) \mid p \in P\}$$

If $\rho(\varphi, p^*) > 0$ then parameter p^* is such that $(x, p^*) \models \varphi$. Moreover, it maximizes the robustness of satisfaction.

Robust Satisfaction, Applications

Assume that x depends on p, we get the following oracle:



Parameter synthesis can be solved by solving

$$p^* = \max \{\rho(\varphi, p) \mid p \in P\}$$

If $\rho(\varphi, p^*) > 0$ then parameter p^* is such that $(x, p^*) \models \varphi$. Moreover, it maximizes the robustness of satisfaction.

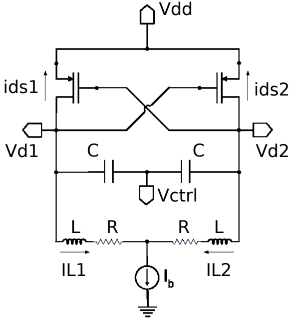
More generally, one can characterize the *validity domain* of φ, given by $d(\varphi, P) = \{p \in P \mid \rho(\varphi, p) > 0\}$

Outline

Outline

A Voltage Controlled Oscillator

- Characterizing oscillations in a Voltage Controlled Oscillator (using unconventional method involving STL)
- Non linear circuit with 3 state variables (IL1, VD1, VD2) and 10 parameters (C, Vctrl, L, R, etc)



Specifying Oscillations, Predicates

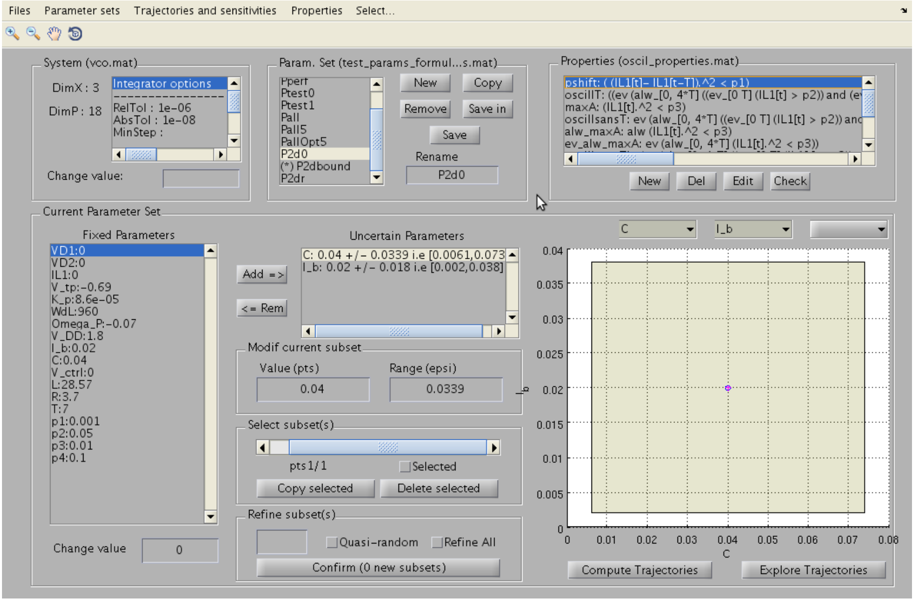
We look for oscillations of period T and given minimum and maximum amplitudes around 0

```
%  Above and below a minimum amplitude
mu0: IL1[t] > Amin
mu1: IL1[t] < -Amin

%  Bounded by a maximum amplitude
mu2: abs(IL1[t]) < Amax

% (almost) Strict periodicity
mu3: ((IL1[t] - IL1[t-T])^2 < epsi)
```

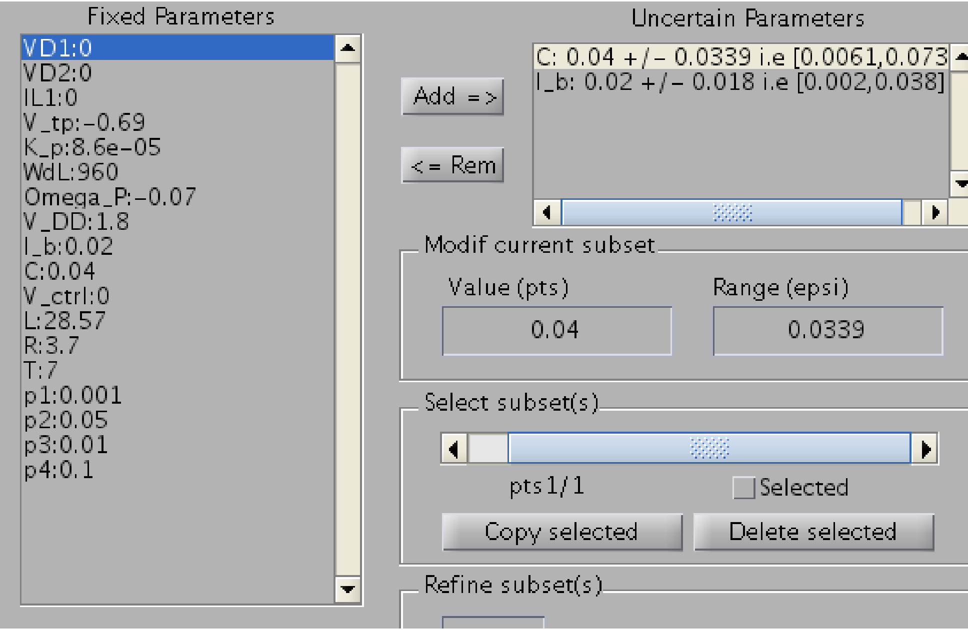
Specifying Oscillations, Formulas

```
% Alternating above and below a minimum amplitude
phi0: (ev_[0,T] (IL1[t]>Amin)) and (ev_[0,T] (IL1[t]<-Amin))

% and holding for 4 periods
phi1: alw_[0,4*T] (phi0)

% Holding strict periodicity
phi2: alw_[0,4*T] ( (IL1[t] - IL1[t-T])^2 ) < epsi)

% Bounding amplitude globally
phi3: alw_[0,4*T] (IL1[t]^2 < Amax)

% Final formula, the ev operator gets rids of transient
phi: ev (phi1 and phi2 and phi3)
```

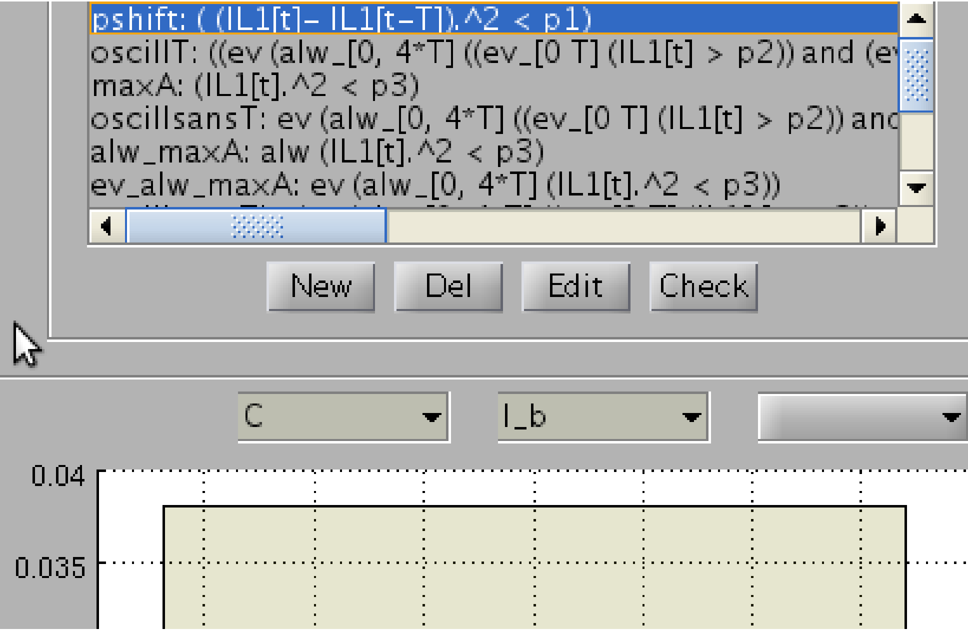
Breach Interface



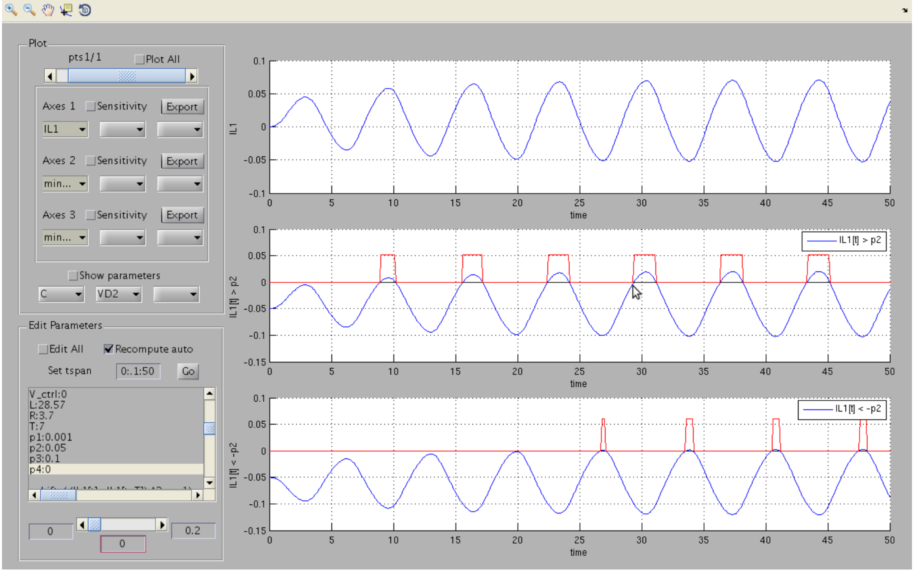
Breach Interface



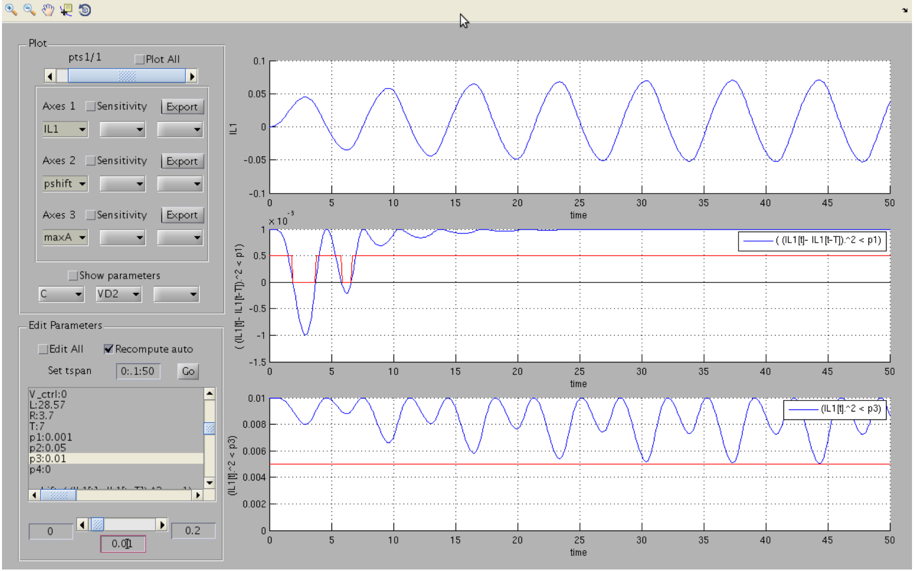
Breach Interface



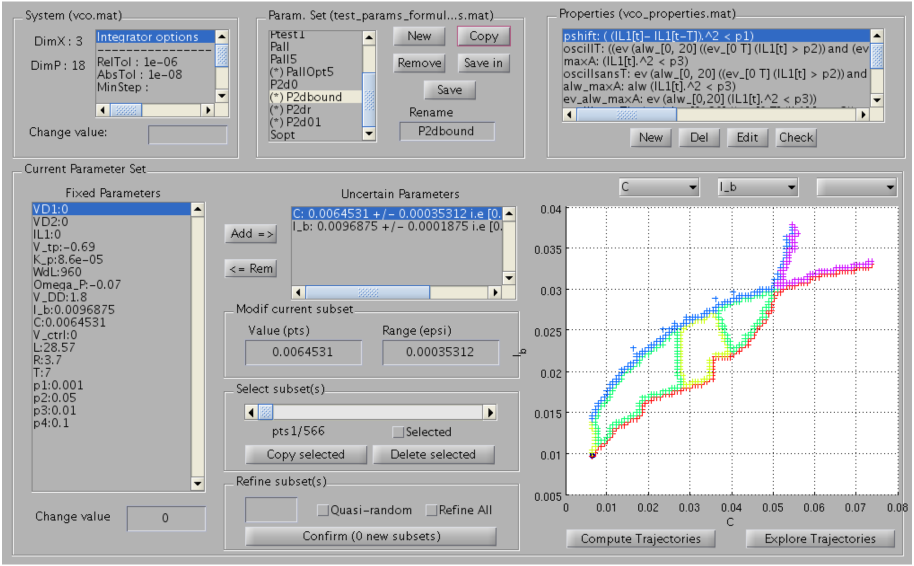
Result on a Single Trace



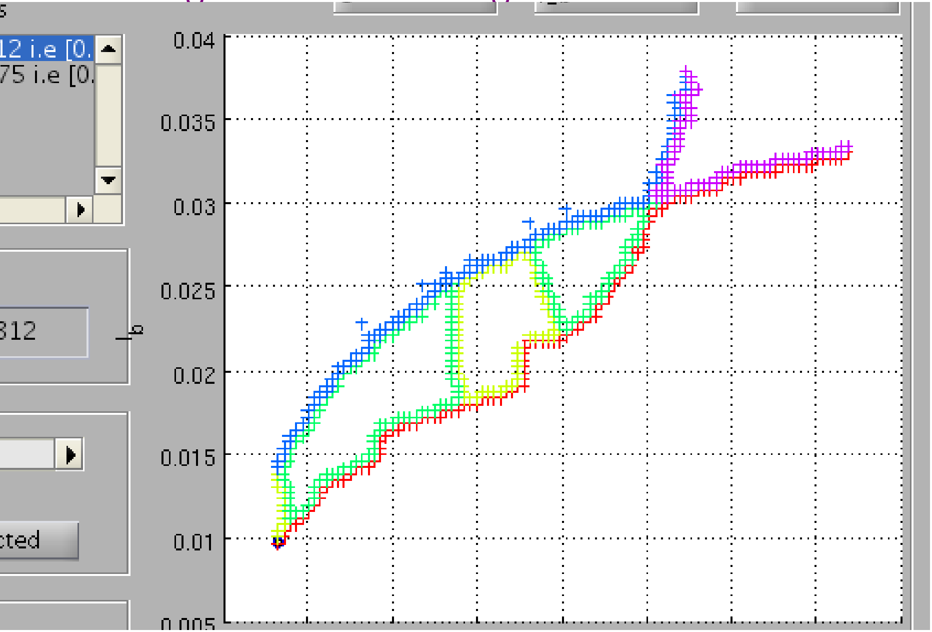
Result on a Single Trace



Partitioning the Parameter Region

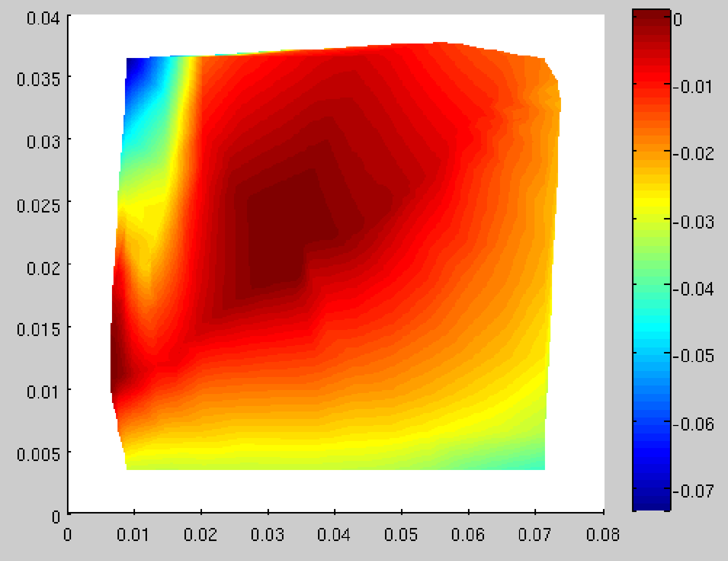


Partitioning the Parameter Region



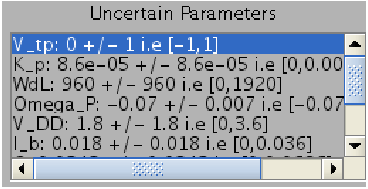
Satisfaction Function

i.e., the resulting cost function



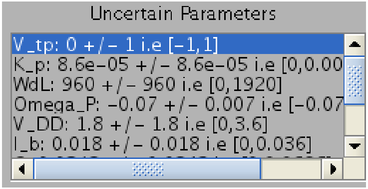
Finding Oscillations

- We defined 10 uncertain parameters with given ranges
- and picked 5 starting points randomly distributed in this domain



Finding Oscillations

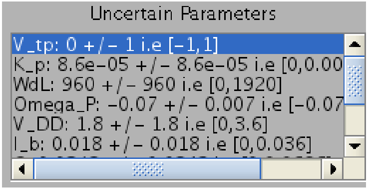
- We defined 10 uncertain parameters with given ranges
- and picked 5 starting points randomly distributed in this domain



Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

Finding Oscillations

- We defined 10 uncertain parameters with given ranges
- and picked 5 starting points randomly distributed in this domain

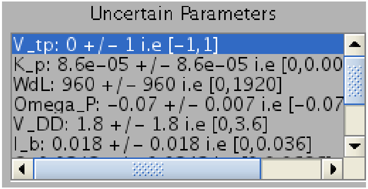


Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

It turned out those were perfectly valid oscillations

Finding Oscillations

- We defined 10 uncertain parameters with given ranges
- and picked 5 starting points randomly distributed in this domain



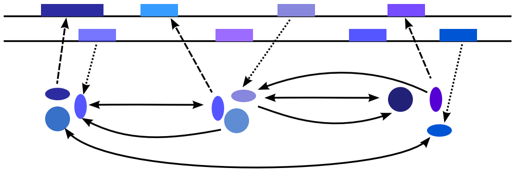
Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

It turned out those were perfectly valid oscillations ... of period $T/4$ and $T/2$

Outline

Biological networks

- Understanding a biological process through interactions between its elements
- Biological networks represents metabolism, gene regulation, signal transduction, protein interactions, etc

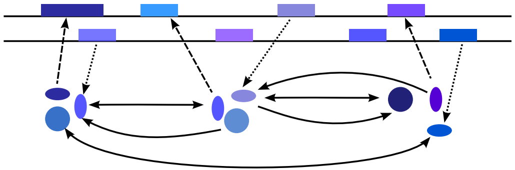


genes network

proteins network

Biological networks

- Understanding a biological process through interactions between its elements
- Biological networks represents metabolism, gene regulation, signal transduction, protein interactions, etc



genes network

proteins network

Application of formal methods

- Formalizing biological hypotheses and test them *in silico*
- Infer new properties and observe them *in vivo*

Models for biological networks

Interaction Graphs

Petri Nets

Flux based models

Thomas networks

Differential equations,
Hybrid systems

qualitative

↓

quantitative

Models for biological networks

Interaction Graphs

Petri Nets

Flux based models

Thomas networks

Differential equations,
Hybrid systems

qualitative → quantitative

A number of formal methods exist for qualitative models but only a few apply for quantitative models

Models for biological networks

Interaction Graphs

Petri Nets

Flux based models

Thomas networks

Differential equations,
Hybrid systems

qualitative → quantitative

A number of formal methods exist for qualitative models but only a few apply for quantitative models

STL can be used in that context

Chemical Reaction Networks

A chemical reaction network (CRN) is pair $(\mathcal{S}, \mathcal{R})$ where

- $\mathcal{S}$ is a set of species
- $\mathcal{R}$ is a set of reactions of the form:

$$\lambda_1 P_1 + \cdots + \lambda_n P_n \to \mu_1 R_1 + \cdots + \mu_m R_m$$

where $P_i \in \mathcal{S}$ are the *products* of the reaction and $R_i \in \mathcal{S}$ are the *reactant*.

Chemical Reaction Networks

A chemical reaction network (CRN) is pair $(\mathcal{S}, \mathcal{R})$ where

- $\mathcal{S}$ is a set of species
- $\mathcal{R}$ is a set of reactions of the form:

$$\lambda_1 P_1 + \cdots + \lambda_n P_n \to \mu_1 R_1 + \cdots + \mu_m R_m$$

where $P_i \in \mathcal{S}$ are the *products* of the reaction and $R_i \in \mathcal{S}$ are the *reactant*.

E.g. a simple CRN is given by $\mathcal{S} = \{H_2, O_2, H_2O, C, CO_2\}$ and the two reactions

$$\begin{aligned} C + O_2 &\to CO_2 \\ 2H_2 + O_2 &\to 2H_20 \end{aligned}$$

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

The Law of Mass Action states that the rate of a reaction is proportional to the product of the concentrations of the reactants.

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

The Law of Mass Action states that the rate of a reaction is proportional to the product of the concentrations of the reactants.

In the previous example,

$$\begin{aligned} 2H_2 + O_2 &\xrightarrow{k_1} 2H_20 \\ C + O_2 &\xrightarrow{k_2} CO_2 \end{aligned}$$

we get

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

The Law of Mass Action states that the rate of a reaction is proportional to the product of the concentrations of the reactants.

In the previous example,

$$\begin{aligned} 2H_2 + O_2 &\xrightarrow{k_1} 2H_20 \\ C + O_2 &\xrightarrow{k_2} CO_2 \end{aligned}$$

we get

$$\frac{d[CO_2]}{dt} = k_2[C][O_2]$$

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

The Law of Mass Action states that the rate of a reaction is proportional to the product of the concentrations of the reactants.

In the previous example,

$$\begin{array}{rcl} 2H_2 + O_2 & \xrightarrow{k_1} & 2H_2 0 \\ C + O_2 & \xrightarrow{k_2} & CO_2 \end{array}$$

we get

$$\begin{array}{rcl} \frac{d[CO_2]}{dt} & = & k_2[C][O_2] \\ \frac{d[O_2]}{dt} & = & -k_2[C][O_2] - k_1[H_2]^2[O_2] \end{array}$$

Mass-action kinetics

Goal Given initial concentrations, predict the evolution of concentrations

The Law of Mass Action states that the rate of a reaction is proportional to the product of the concentrations of the reactants.

In the previous example,

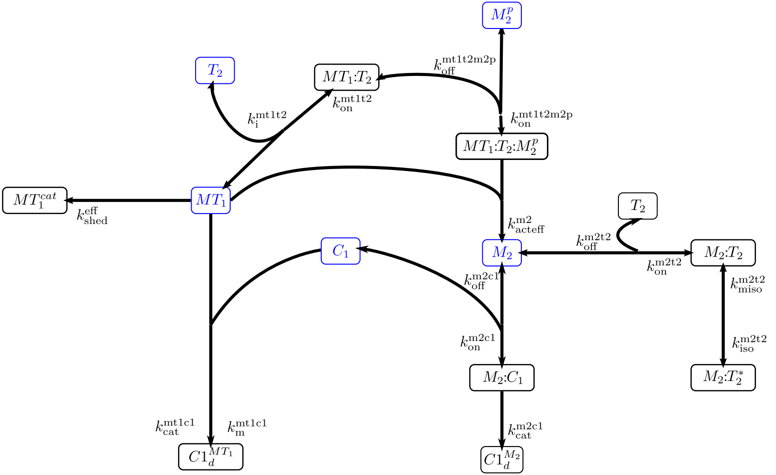
$$\begin{aligned} 2H_2 + O_2 &\xrightarrow{k_1} 2H_20 \\ C + O_2 &\xrightarrow{k_2} CO_2 \end{aligned}$$

we get

$$\begin{aligned} \frac{d[CO_2]}{dt} &= k_2[C][O_2] \\ \frac{d[O_2]}{dt} &= -k_2[C][O_2] - k_1[H_2]^2[O_2] \\ & etc \end{aligned}$$

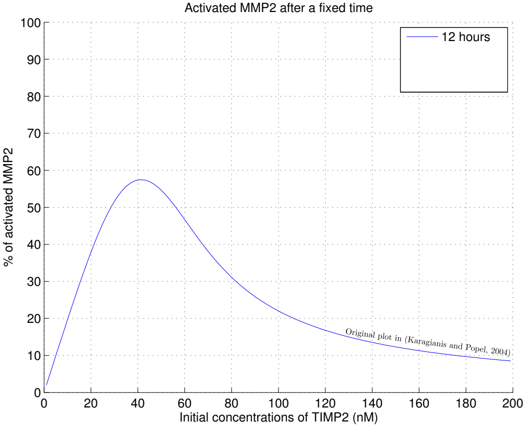
An Enzymatic Network Involved in Angiogenesis

Collagen (C_1) degradation by matrix metalloproteinase (M_2^P) and membrane type 1 metalloproteinase (MT_1).



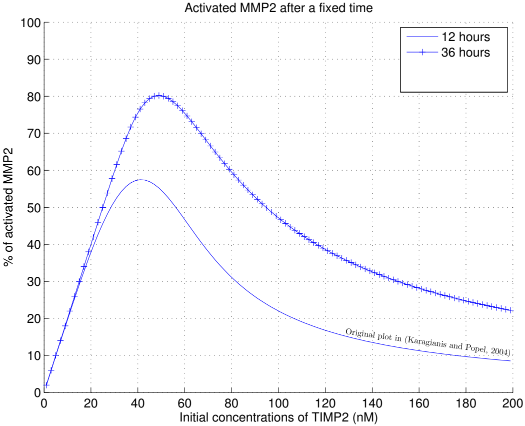
Rigorous Steady State Analysis

In [KP04], activation of M_2^P after 12h "Nearly steady state" for $T_2(0)$ between 0 and 200 nM.



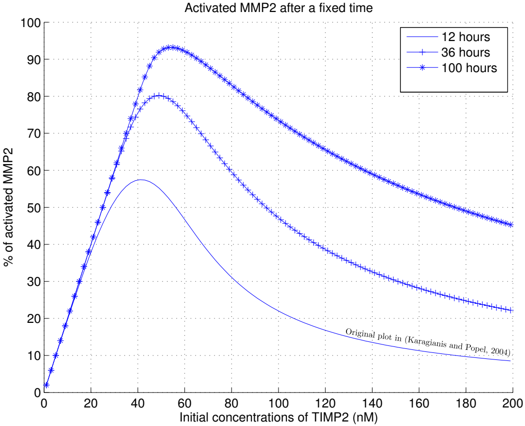
Rigorous Steady State Analysis

In [KP04], activation of M_2^P after 12h "Nearly steady state" for $T_2(0)$ between 0 and 200 nM. It turned out that steady state was not reached for $T_2(0) > 20$ nM.



Rigorous Steady State Analysis

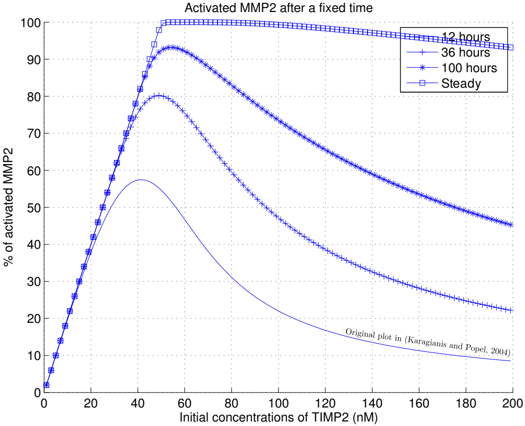
In [KP04], activation of M_2^P after 12h "Nearly steady state" for $T_2(0)$ between 0 and 200 nM. It turned out that steady state was not reached for $T_2(0) > 20$ nM.



Rigorous Steady State Analysis

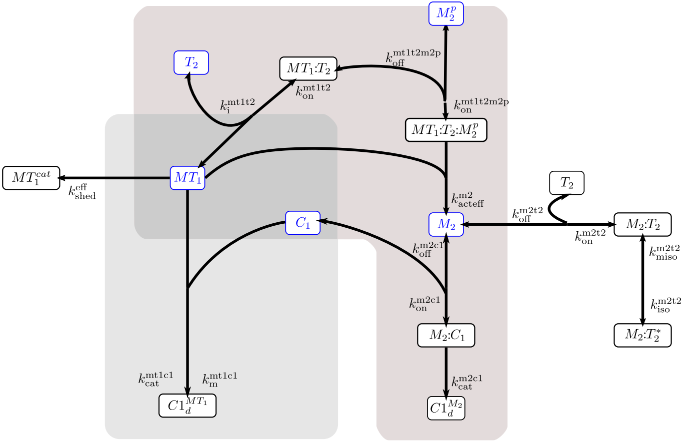
In [KP04], activation of M_2^P after 12h "Nearly steady state" for $T_2(0)$ between 0 and 200 nM. It turned out that steady state was not reached for $T_2(0) > 20$ nM.

Using $\varphi \Leftrightarrow \Diamond\,\Box\,(|\dot{M}_2(t)| < \varepsilon \times M_2^P(0))$ we could guarantee the correct plot.



Open Model

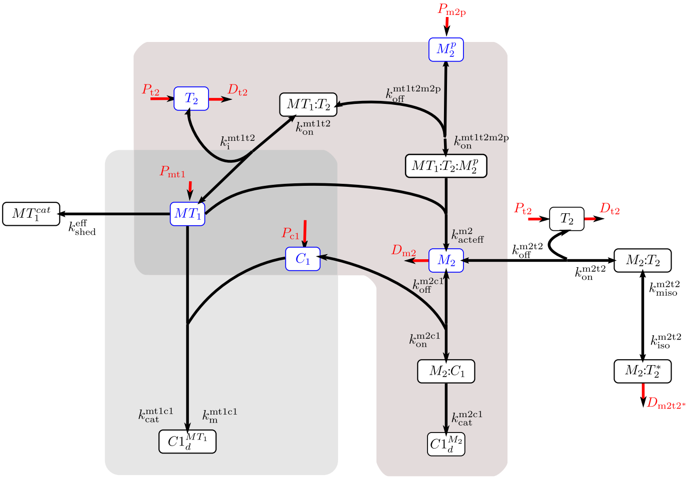
We extended the model by introducing production and degradation terms



More complex behaviors becomes possible, such as oscillatory regimes

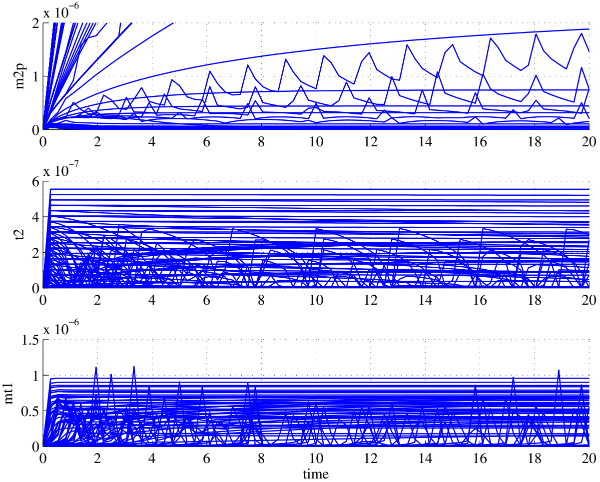
Open Model

We extended the model by introducing production and degradation terms

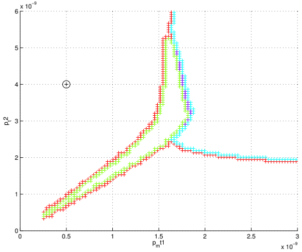


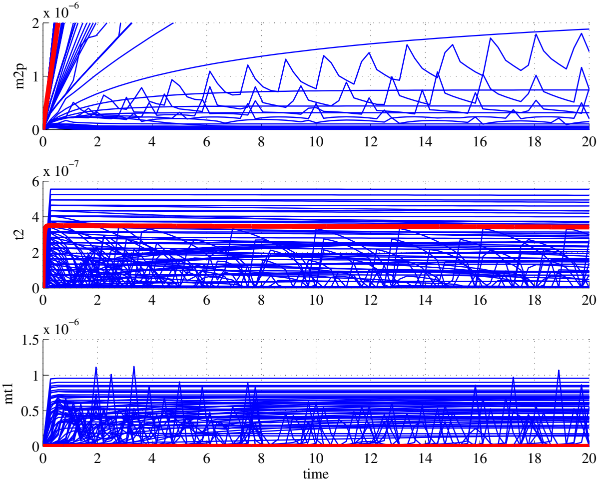
More complex behaviors becomes possible, such as oscillatory regimes

Oscillations Map

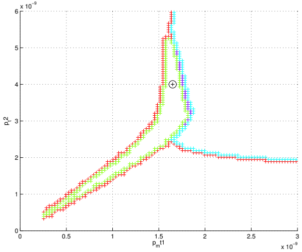


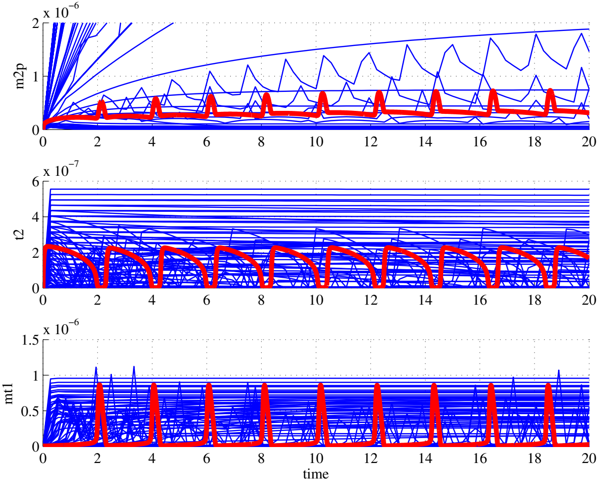
Oscillations Map





Oscillations Map





Oscillations Map