Robust Satisfaction of Signal Temporal Logics and Applications

Alexandre Donzé

Verimag, Grenoble

September 26th, 2011

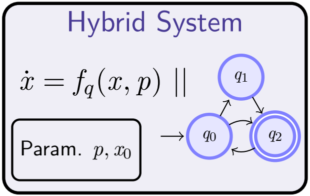
Overview

Design and analysis of hybrid systems

e.g., embedded systems, mixed-signal circuits, biological systems

Simulation-based approaches for verification and parameter synthesis

Lightweight verification, as opposed to full-fledged Model-Checking



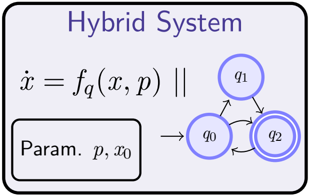
Overview

Design and analysis of hybrid systems

e.g., embedded systems, mixed-signal circuits, biological systems

Simulation-based approaches for verification and parameter synthesis

Lightweight verification, as opposed to full-fledged Model-Checking



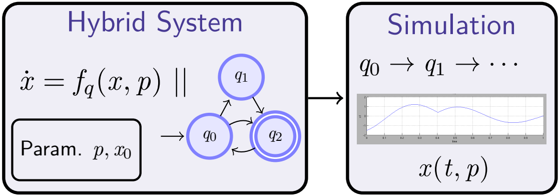
Overview

Design and analysis of hybrid systems

e.g., embedded systems, mixed-signal circuits, biological systems

Simulation-based approaches for verification and parameter synthesis

Lightweight verification, as opposed to full-fledged Model-Checking



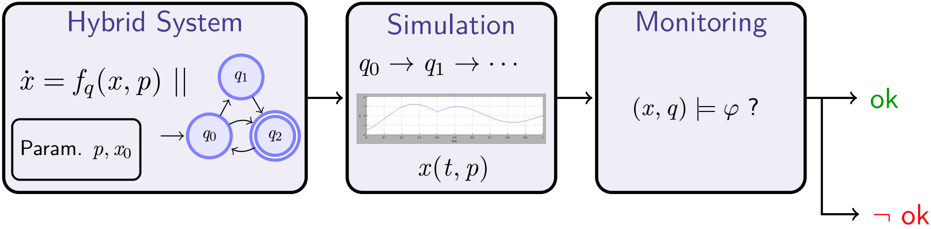
Overview

Design and analysis of hybrid systems

e.g., embedded systems, mixed-signal circuits, biological systems

Simulation-based approaches for verification and parameter synthesis

Lightweight verification, as opposed to full-fledged Model-Checking



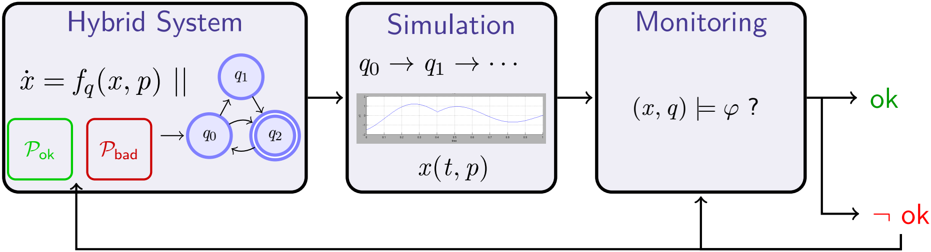
Overview

Design and analysis of hybrid systems

e.g., embedded systems, mixed-signal circuits, biological systems

Simulation-based approaches for verification and parameter synthesis

Lightweight verification, as opposed to full-fledged Model-Checking

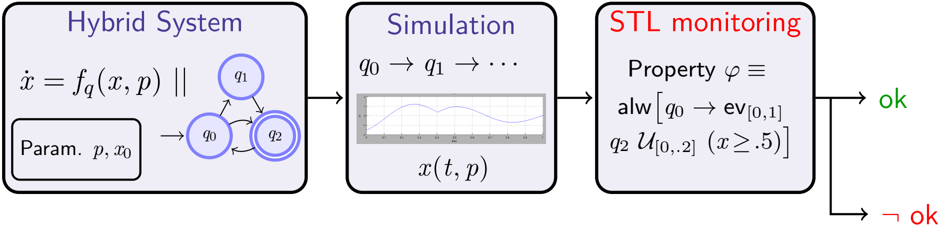


Overview

- Signal Temporal Logic (STL): temporal specifications for continuous and hybrid systems
- Quantitative (Robust) satisfaction of STL adapted to deal with uncertainty

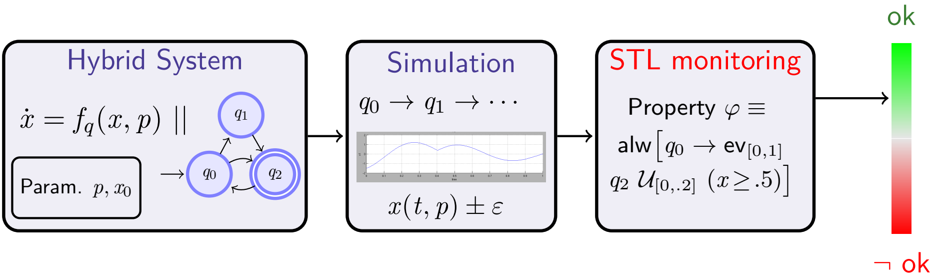
Overview

- Signal Temporal Logic (STL): temporal specifications for continuous and hybrid systems
- Quantitative (Robust) satisfaction of STL adapted to deal with uncertainty



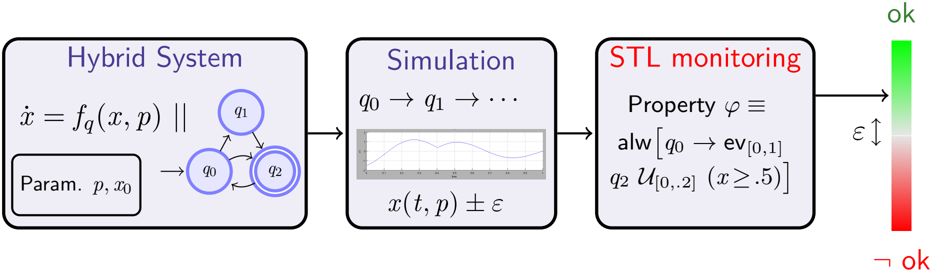
Overview

- Signal Temporal Logic (STL): temporal specifications for continuous and hybrid systems
- Quantitative (Robust) satisfaction of STL adapted to deal with uncertainty



Overview

- Signal Temporal Logic (STL): temporal specifications for continuous and hybrid systems
- Quantitative (Robust) satisfaction of STL adapted to deal with uncertainty



Outline

1. Temporal Logics for Continuous Time and Space
    - Signal Temporal Logic
    - Quantitative Satisfaction of STL
2. An Implementation: The Breach Toolbox
    - Simulation of Parametric Hybrid Systems
    - Specifying STL Formulas
3. Applications
    - Case Study: Voltage Controlled Oscillator
    - An Example from Systems Biology

Outline

1. Temporal Logics for Continuous Time and Space
   - Signal Temporal Logic
   - Quantitative Satisfaction of STL
2. An Implementation: The Breach Toolbox
   - Simulation of Parametric Hybrid Systems
   - Specifying STL Formulas
3. Applications
   - Case Study: Voltage Controlled Oscillator
   - An Example from Systems Biology

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.

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

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

Examples:

- $\varphi\ \varphi\ \varphi\ \varphi\ \cdots$ satisfies alw φ
- $\psi\ \psi\ \psi\ \varphi\ \psi\ \cdots$ satisfies ev φ
- $\varphi\ \varphi\ \varphi\ \varphi\ \psi\ \cdots$ satisfies $\varphi\ \mathcal{U}\ \psi$

From Discrete to Continuous

Temporal logics mostly developed for discrete systems

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

Some reasons:

- Discretization often leads to state-explosion problem
- 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

From Discrete to Continuous

Temporal logics mostly developed for discrete systems

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

Some reasons:

- Discretization often leads to state-explosion problem
- 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

From Discrete to Continuous

Temporal logics mostly developed for discrete systems

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

Some reasons:

- Discretization often leads to state-explosion problem
- 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 $\mu : \mu(x) > 0$

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: $\text{ev}_{[a,b]}\varphi = \top\, \mathcal{U}_{[a,b)}\ \varphi$ and $\text{alw}_{[a,b]}\varphi = \varphi\, \mathcal{U}_{[a,b)}\ \bot$.

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)

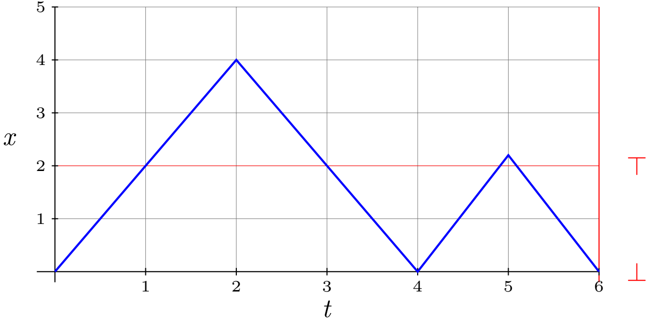
The validity of a formula φ with respect to a signal x at time t is

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

Additionally: $\text{ev}_{[a,b]}\varphi = \top\, \mathcal{U}_{[a,b)}\ \varphi$ and $\text{alw}_{[a,b]}\varphi = \varphi\, \mathcal{U}_{[a,b)}\ \perp$.

Examples

Consider a simple piecewise affine signal:

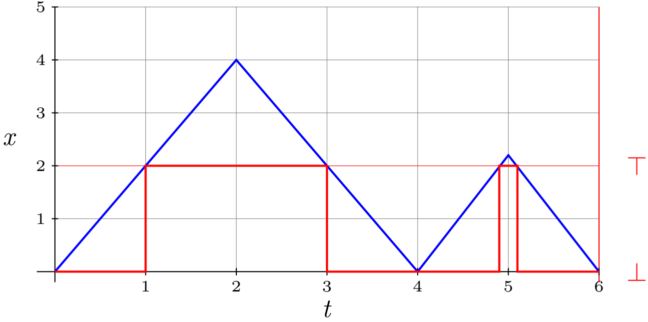


Truth value of :

- $\varphi = x > 2$

Examples

Consider a simple piecewise affine signal:

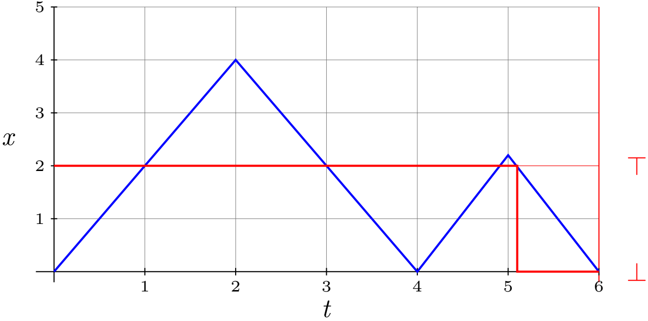


Truth value of :

- $\varphi = x > 2$

Examples

Consider a simple piecewise affine signal:

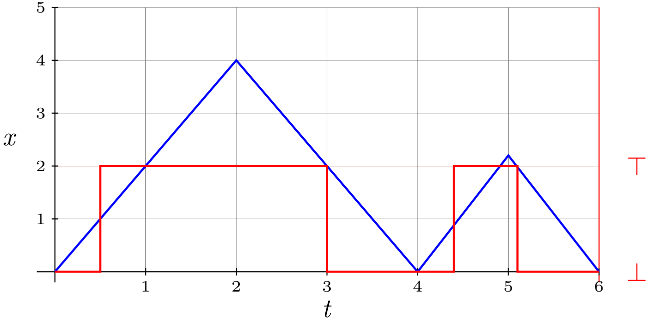


Truth value of :

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

Examples

Consider a simple piecewise affine signal:

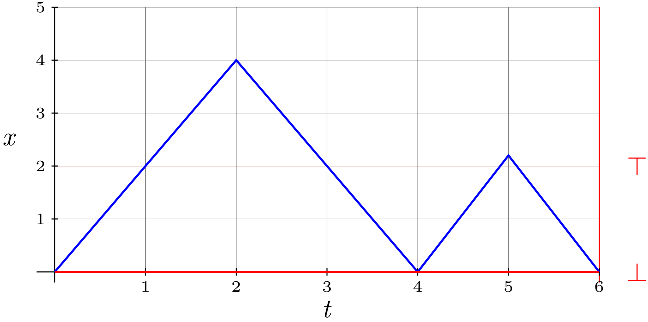


Truth value of :

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

Examples

Consider a simple piecewise affine signal:



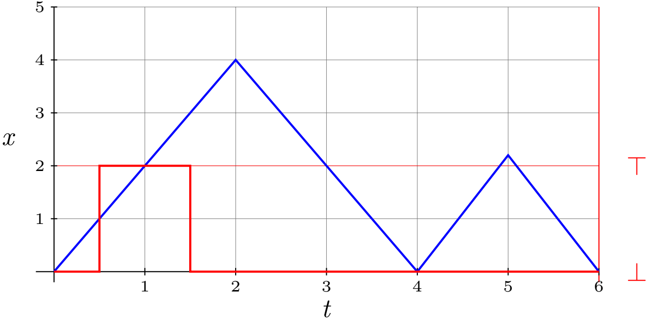
Truth value of :

- $\varphi = x > 2$

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

Examples

Consider a simple piecewise affine signal:



Truth value of :

- $\varphi = x > 2$

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

Outline

1. Temporal Logics for Continuous Time and Space
   - Signal Temporal Logic
   - Quantitative Satisfaction of STL
2. An Implementation: The Breach Toolbox
   - Simulation of Parametric Hybrid Systems
   - Specifying STL Formulas
3. Applications
   - Case Study: Voltage Controlled Oscillator
   - An Example from Systems Biology

From Semantics to Satisfaction Functions

STL semantics

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

A Boolean Satisfaction Function χ

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

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 $\chi : (x,t) \rightarrow \{-\infty, \infty\}$:

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

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

From 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 $\chi : (x,t) \to \{-\infty, \infty\}$:

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

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

From 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 $\chi : (x,t) \to \{-\infty, \infty\}$:

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

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

From 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 $\chi : (x,t) \to \{-\infty, \infty\}$:

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

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

From 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 $\chi : (x,t) \to \{-\infty, \infty\}$:

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

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

From 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 $\chi : (x,t) \to \{-\infty, \infty\}$:

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

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

Simple idea

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

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

From Boolean to Quantitative Satisfaction Function

For atomic predicates:

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

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

Simple idea

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

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

From Boolean to Quantitative Satisfaction Function

For atomic predicates:

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

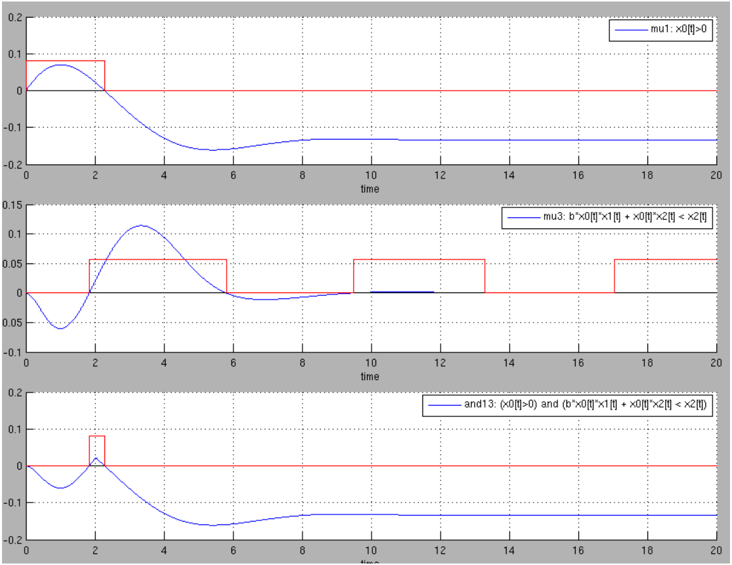
The sign removes the quantitative information in μ to get a boolean signal

Simple idea

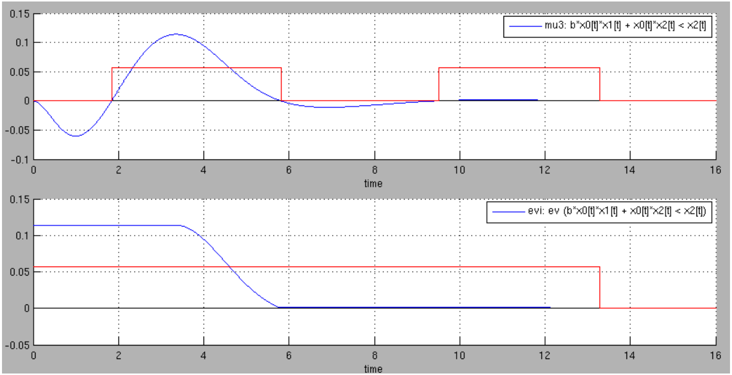
- Get rid of sign to get a quantitative satisfaction function ρ
- Keep the same inductive rules for the quantitative semantics:

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

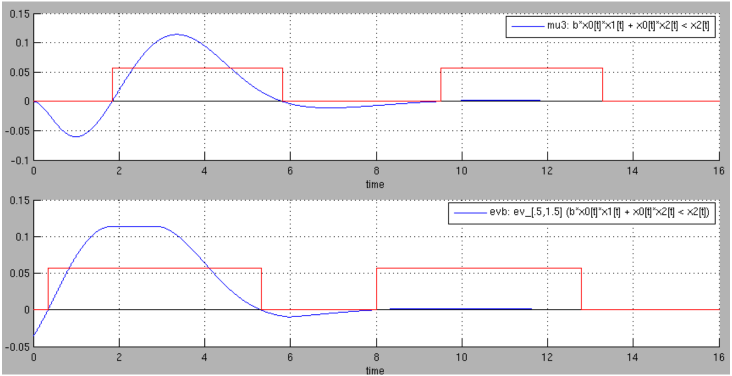
Robust Satisfaction, Examples



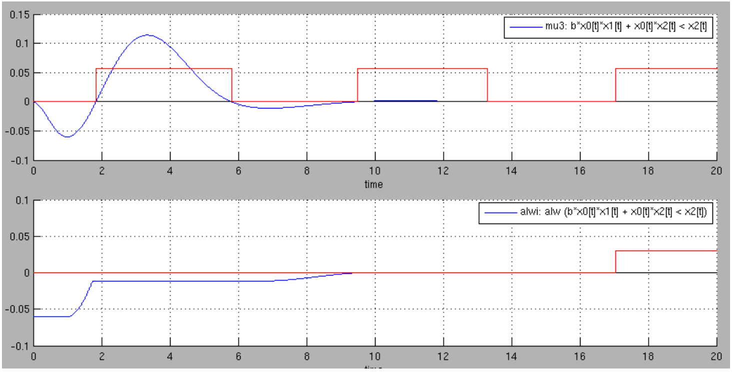
Robust Satisfaction, Examples



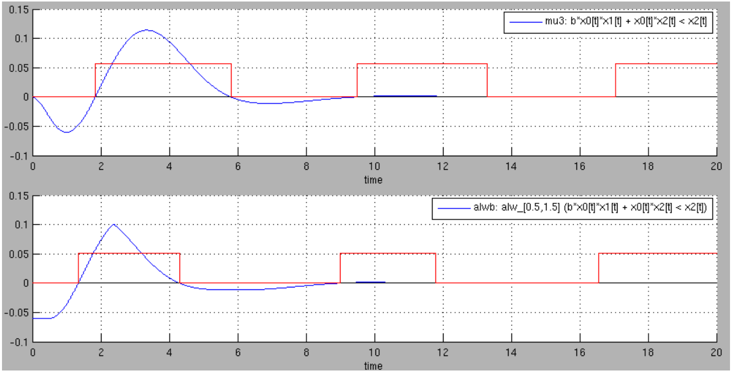
Robust Satisfaction, Examples



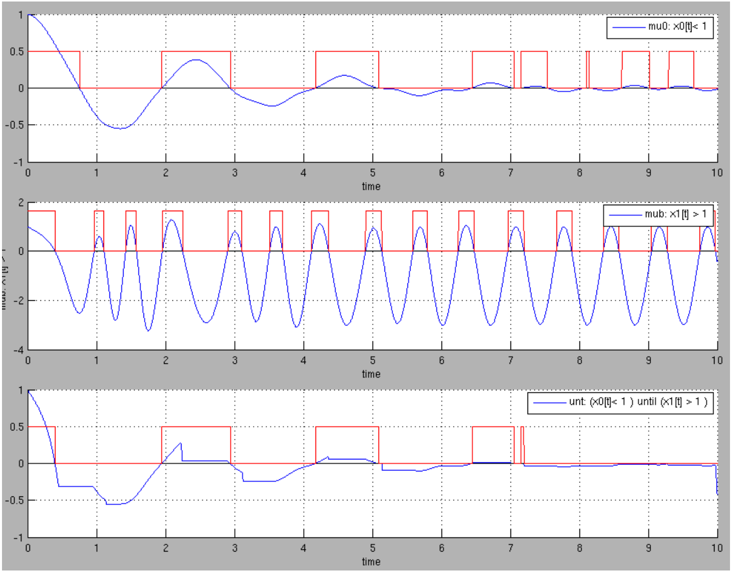
Robust Satisfaction, Examples



Robust Satisfaction, Examples

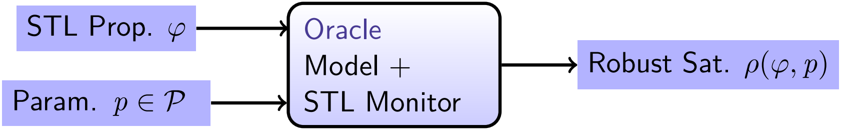


Robust Satisfaction, Examples



Robust Satisfaction, Applications

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



Parameter synthesis can be solved by solving

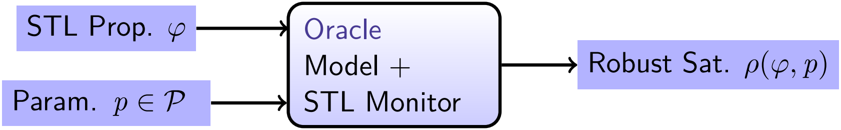
$$p^* = \max \{\rho(\varphi, p) \mid p \in \mathcal{P}\}$$

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 $d(\varphi, \mathcal{P}) = \{p \in \mathcal{P} \mid \rho(\varphi, p) > 0\}$

Robust Satisfaction, Applications

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



Parameter synthesis can be solved by solving

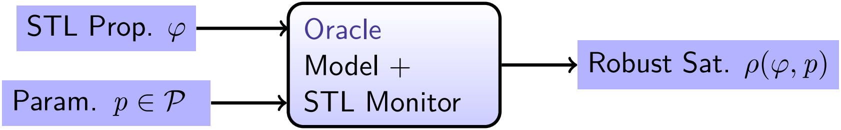
$$p^* = \max \{\rho(\varphi, p) \mid p \in \mathcal{P}\}$$

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 $d(\varphi, \mathcal{P}) = \{p \in \mathcal{P} \mid \rho(\varphi, p) > 0\}$

Robust Satisfaction, Applications

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



Parameter synthesis can be solved by solving

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

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 $d(\varphi, \mathcal{P}) = \{p \in \mathcal{P} \mid \rho(\varphi, p) > 0\}$

Outline

Outline

Hybrid Model

Breach deals with piecewise-continuous models of the form

$$\begin{cases} \dot{\mathbf{x}} & = & f(q, \mathbf{x}, \mathbf{p}),\ \ \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{y} & = & g(\mathbf{x}) \\ q^+ & = & e(q^-, \mathbf{y}),\ \ q(0) = q_0 \end{cases}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state variable

$q \in \mathbb{N}$ is the discrete state,

$\mathbf{p} \in \mathbb{R}^{n_p}$ is the parameter vector,

g is the guard function and

e is the event or transition function, where $q^+ \neq q^-$ only if $g(\mathbf{x}) = 0$

Simulation Algorithm

Discontinuity locking + Event detection by zero crossing detection

1. Let $f_k(\mathbf{x}, \mathbf{p}) = f(q(t_k), \mathbf{x}, \mathbf{p})$ (block switching between t_k and t_{k+1})
2. Solve ODE $\dot{\mathbf{x}} = f_k(\mathbf{x}, \mathbf{p})$ on $[t_k, t_k + h_k]$
3. **If** for all i, sign$(g_i(\mathbf{x})) = \texttt{Constant}$ on $(t_k, t_k + h_k]$ then let $t_{k+1} = t_k + h_k$
4. **Else** find the minimum time $\tau > t_k$ for which $g_i(\mathbf{x}(\tau)) = 0$ and let $t_{k+1} = \tau$
5. Return $\xi_{\mathbf{p}}(t_{k+1})$ and restart with $q(t_{k+1}^+) = e(q(t_k), \lambda(t_{k+1}^-)))$

Simulation and Sensitivity Analysis

Simulation based on a state-of-the-art ODE solver CVodes

- Variable-steps variable order implicit methods, efficient for stiff and non-stiff dynamics
- Builtin zero-crossing detection for guards.

Sensitivity functions $s_{ij}(t) = \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}_j}(t)$ are also computed by CVodes solver

Breach implementation adds

- the computation of sensitivity discontinuities at transitions
- an efficient Matlab-C interface:
  - The solver and the dynamics are in C
  - Matlab manipulates arrays of parameters and externally computed arrays of trajectories

$\Rightarrow$ Much more efficient than Matlab native ODE solvers

Simulation and Sensitivity Analysis

Simulation based on a state-of-the-art ODE solver CVodes

- Variable-steps variable order implicit methods, efficient for stiff and non-stiff dynamics
- Builtin zero-crossing detection for guards.

Sensitivity functions $s_{ij}(t) = \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}_j}(t)$ are also computed by CVodes solver

Breach implementation adds

- the computation of sensitivity discontinuities at transitions
- an efficient Matlab-C interface:
  - The solver and the dynamics are in C
  - Matlab manipulates arrays of parameters and externally computed arrays of trajectories

$\Rightarrow$ Much more efficient than Matlab native ODE solvers

Simulation and Sensitivity Analysis

Simulation based on a state-of-the-art ODE solver CVodes

- Variable-steps variable order implicit methods, efficient for stiff and non-stiff dynamics
- Builtin zero-crossing detection for guards.

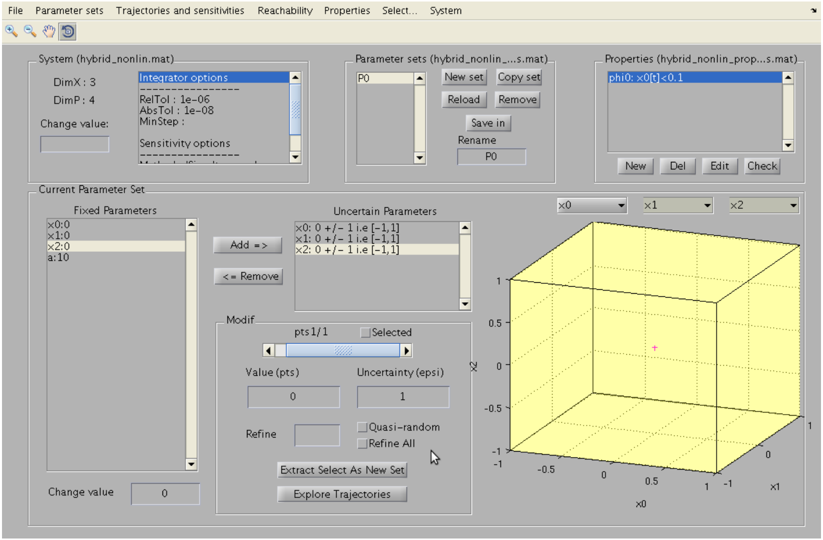
Sensitivity functions $s_{ij}(t) = \frac{\partial \mathbf{x}_i}{\partial \mathbf{p}_j}(t)$ are also computed by CVodes solver

Breach implementation adds

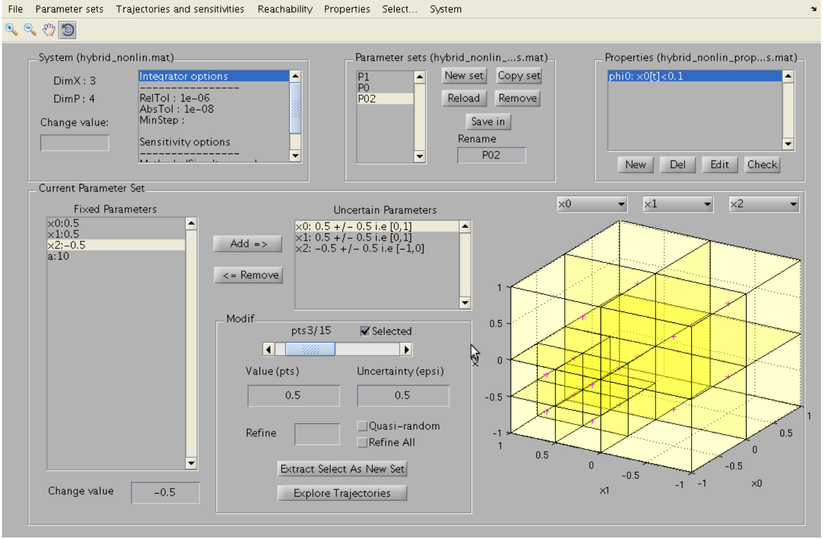
- the computation of sensitivity discontinuities at transitions
- an efficient Matlab-C interface:
  - The solver and the dynamics are in C
  - Matlab manipulates arrays of parameters and externally computed arrays of trajectories

$\Rightarrow$ Much more efficient than Matlab native ODE solvers

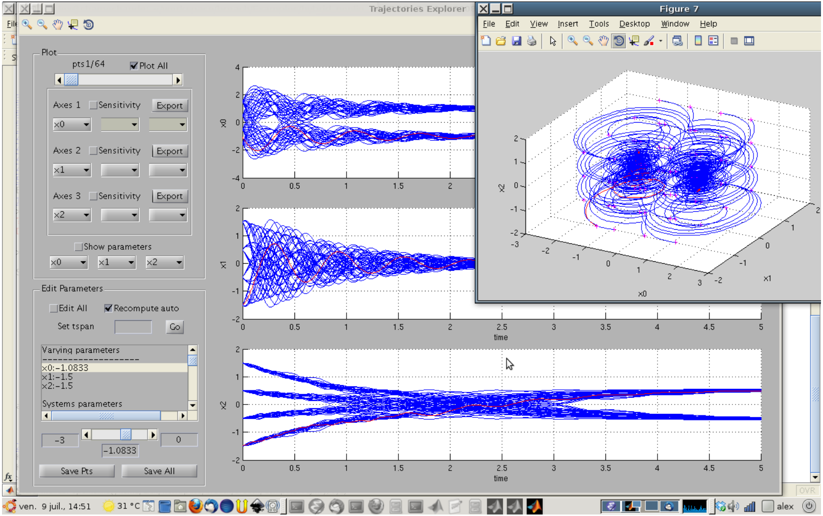
Breach GUIs for trajectories exploration



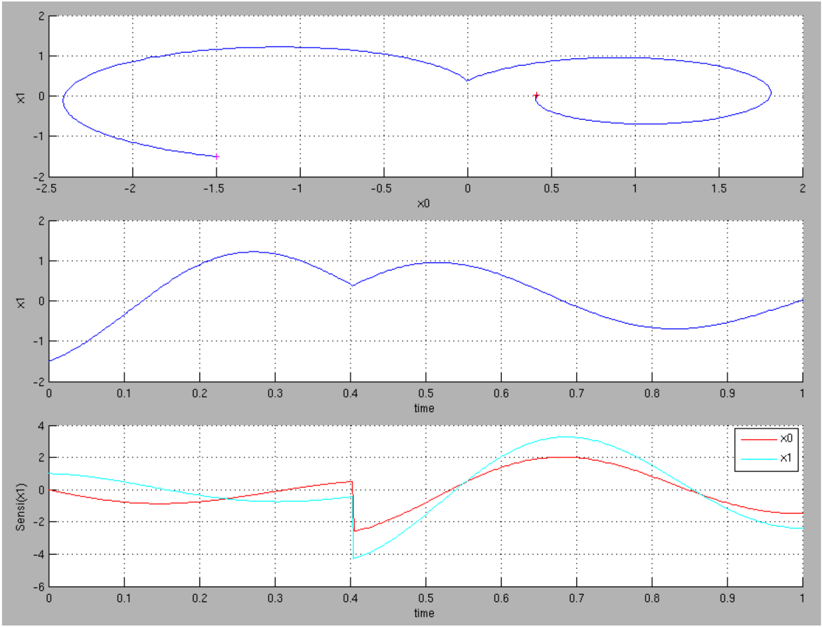
Breach GUIs for trajectories exploration



Breach GUIs for trajectories exploration



Breach GUIs for trajectories exploration



Outline

Temporal logic formulas: atomic predicates

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

$+$ usual syntactic sugars for disjunction, eventually and always.

Predicates: General constraints on the variables: $\mu \equiv \mu(\mathbf{x}, \mathbf{p}, t) \geq 0$

```
% distance to (p0,p1) is more than 2.
 (x0[t]-p0)^2 + (x1[t]-p1)^2) >= 4.

% the system reached steady state (very slow evolution)
abs(ddt{x0}[t])+abs(ddt{x1}[t])) <= 1e-3

% x0 is sensitive to parameter p3
abs(d{x0}{p3}[t]) >= 10*x0[t]/p3
```

Temporal logic formulas: formulas

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

$+$ usual syntactic sugars for disjunction, eventually and always.

```
% x0 will become more than -.9 whithin .5 s
ev_[0,.5] (x0[t]>-.9)

%  the system will eventually remain close to 0
ev (always (abs(x0)[t] < 1e-6))

% x0 remains low until x1 stabilizes before 10 seconds
(x0[t] < 0.1) until_[0, 10] always ((abs(ddt{x1}[t]) < 1e-6))
```

Computing the satisfaction functions

Breach computes the function $\rho(\varphi, x, \cdot)$ by induction on the structure of φ.

This reduces to three subproblems: given two functions $y, y' : \mathbb{T} \to \mathbb{R}$, and an interval $[a, b]$

1. (operator $\neg$) compute $\forall t,\ z[t] = -y[t]$;
2. (operator $\wedge$) compute $\forall t,\ z[t] = \min(y[t], y'[t])$
3. (operator $\mathcal{U}$) compute $\forall t,\ z[t] = \max_{\tau \in t+[a,b]} (\min(y'[\tau], \min_{s\in[t,\tau]} y[s]))$

1\. and 2. are reasonably trivials. 3., less (maybe for a $\min - \max$ guru).

In practice, Breach implementation behaves linearly in the size of formulas and the size of traces

Computing the satisfaction functions

Breach computes the function $\rho(\varphi, x, \cdot)$ by induction on the structure of φ.

This reduces to three subproblems: given two functions $y, y' : \mathbb{T} \to \mathbb{R}$, and an interval $[a, b]$

1. (operator $\neg$) compute $\forall t,\ z[t] = -y[t]$;
2. (operator $\wedge$) compute $\forall t,\ z[t] = \min(y[t], y'[t])$
3. (operator $\mathcal{U}$) compute $\forall t,\ z[t] = \max_{\tau \in t+[a,b]} (\min(y'[\tau], \min_{s\in[t,\tau]} y[s])))$

1. and 2. are reasonably trivials. 3., less (maybe for a $\min - \max$ guru).

In practice, Breach implementation behaves linearly in the size of formulas and the size of traces

Computing the satisfaction functions

Breach computes the function $\rho(\varphi, x, \cdot)$ by induction on the structure of φ.

This reduces to three subproblems: given two functions $y, y' : \mathbb{T} \to \mathbb{R}$, and an interval $[a, b]$

1. (operator $\neg$) compute $\forall t,\ z[t] = -y[t]$;
2. (operator $\wedge$) compute $\forall t,\ z[t] = \min(y[t], y'[t])$
3. (operator $\mathcal{U}$) compute $\forall t,\ z[t] = \max_{\tau \in t+[a,b]} (\min(y'[\tau], \min_{s \in [t,\tau]} y[s])))$

1. and 2. are reasonably trivials. 3., less (maybe for a $\min - \max$ guru).

In practice, Breach implementation behaves linearly in the size of formulas and the size of traces

Computing the satisfaction functions

Breach computes the function $\rho(\varphi, x, \cdot)$ by induction on the structure of φ.

This reduces to three subproblems: given two functions $y, y' : \mathbb{T} \to \mathbb{R}$, and an interval $[a, b]$

1. (operator $\neg$) compute $\forall t,\ z[t] = -y[t]$;
2. (operator $\wedge$) compute $\forall t,\ z[t] = \min(y[t], y'[t])$
3. (operator $\mathcal{U}$) compute $\forall t,\ z[t] = \max_{\tau \in t+[a,b]}(\min(y'[\tau], \min_{s\in[t,\tau]} y[s])))$

1. and 2. are reasonably trivials. 3., less (maybe for a $\min - \max$ guru).

In practice, Breach implementation behaves linearly in the size of formulas and the size of traces

Computational Cost, Some Experiments

(a) Same signal, formula $\varphi = (x > 0) \underbrace{\mathcal{U}_{[0,1)}\ (x > 0)\ \mathcal{U}_{[0,1)}\ (x > 0) \ldots}_{i \text{ times}}$

(b) Same formula: $\varphi = \text{alw}(x > 1.5 \Rightarrow \text{ev}(\text{alw}(x < .1)))$, different input sizes

(a)

i	time(s)
1	0.34747
2	0.46335
3	0.60599
4	0.76067
5	0.89201
6	1.03761

(b)

input size	time(s)
31416	0.18402
345566	0.40761
659716	0.75508
973866	1.09268
1288016	1.4587

Computational Cost, Some Experiments

(a) Same signal, formula $\varphi = (x > 0)\ \underbrace{\mathcal{U}_{[0,1)}\ (x > 0)\ \mathcal{U}_{[0,1)}\ (x > 0) \ldots}_{i \text{ times}}$

(b) Same formula: $\varphi = \mathsf{alw}(x > 1.5 \Rightarrow \mathsf{ev}(\mathsf{alw}(x < .1)))$, different input sizes

(a)

i	time(s)
1	0.34747
2	0.46335
3	0.60599
4	0.76067
5	0.89201
6	1.03761

(b)

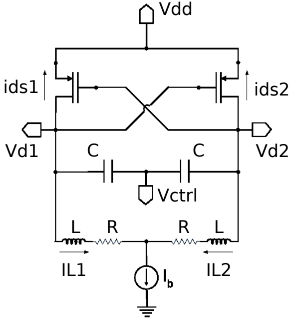
input size	time(s)
31416	0.18402
345566	0.40761
659716	0.75508
973866	1.09268
1288016	1.4587

Outline

Outline

Case Study: Voltage Controlled Oscillators

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



Specifying Oscillations, Predicates

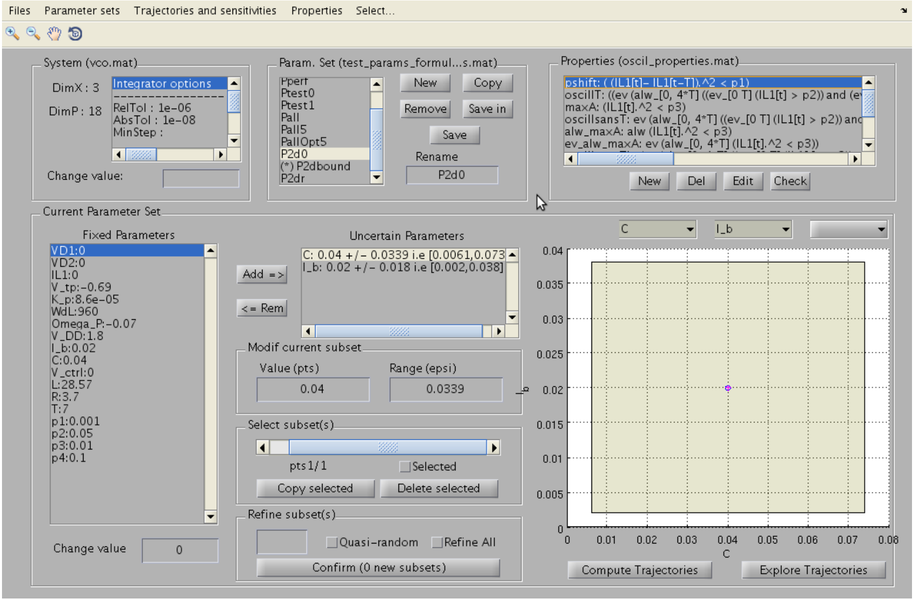
We look for oscillations of period T and given minimum and maximum amplitudes around 0

```
%  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: (abs(IL1[t] - IL1[t-T]) < epsi)
```

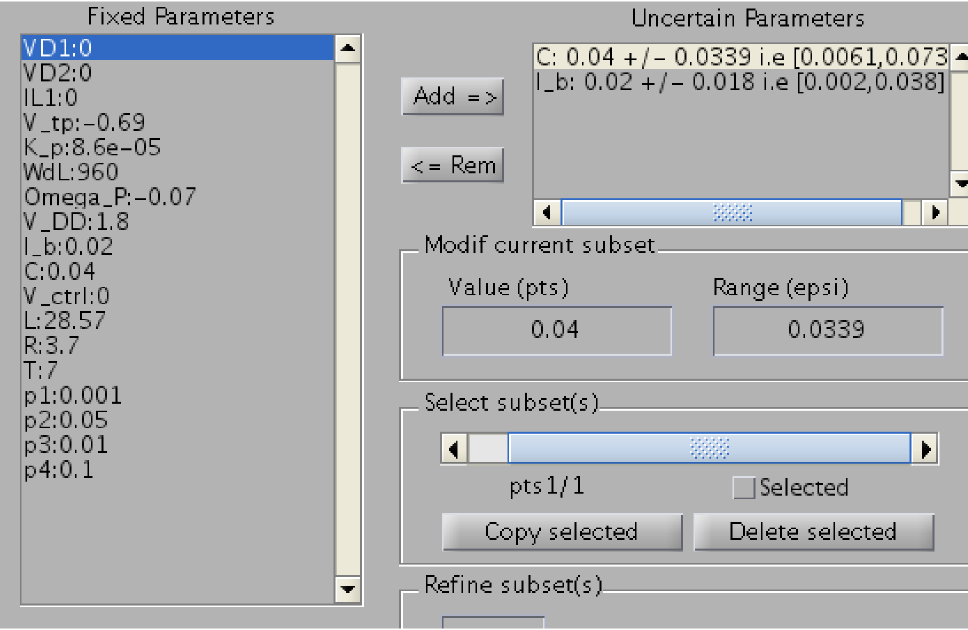
Specifying Oscillations, Formulas

```
% 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 (IL1[t]^2 < Amax)

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

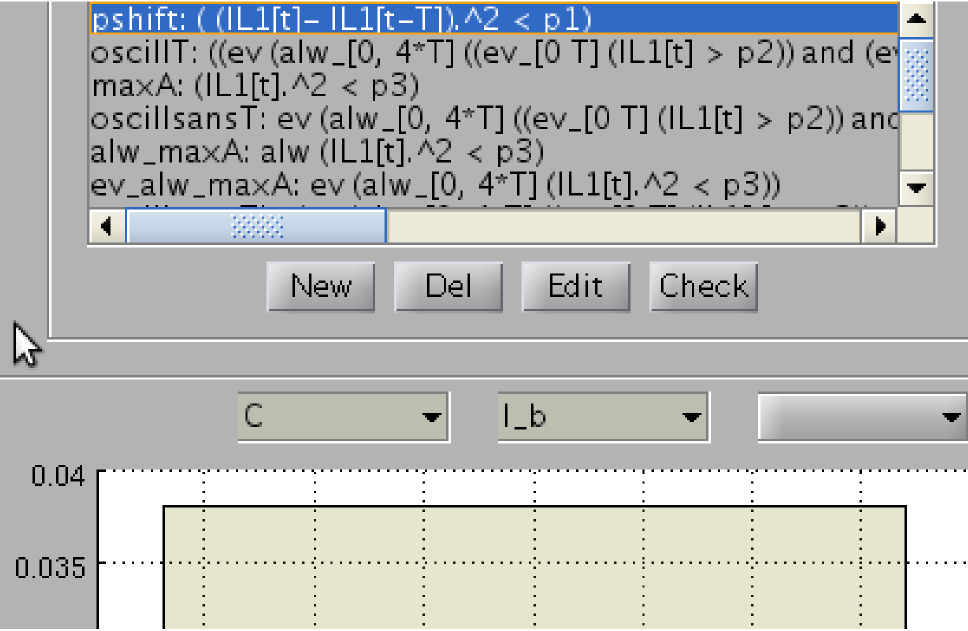
Breach Interface



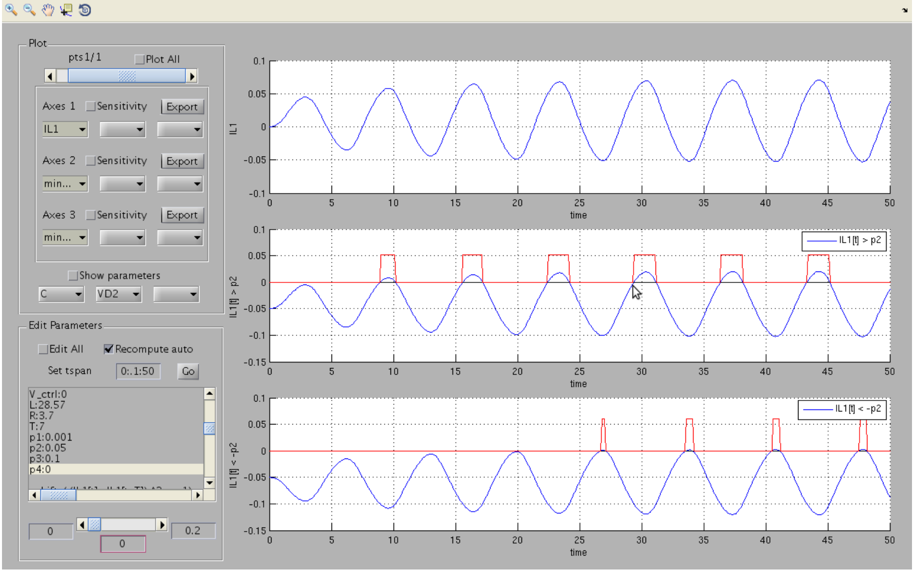
Breach Interface



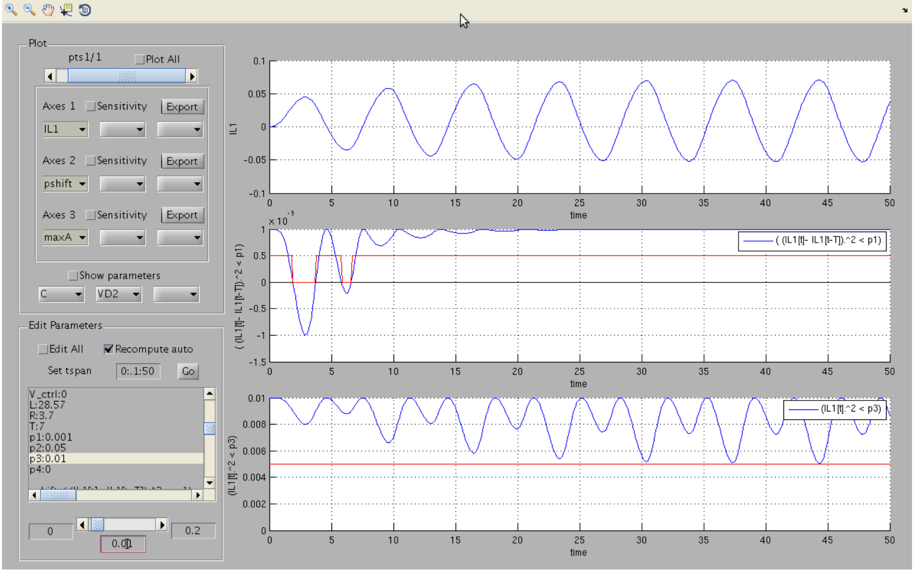
Breach Interface



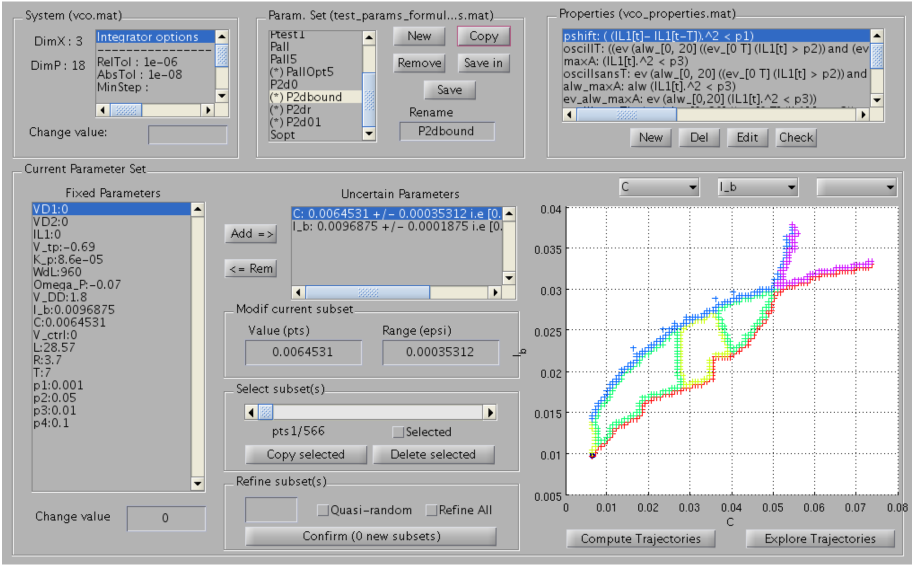
Result on a Single Trace



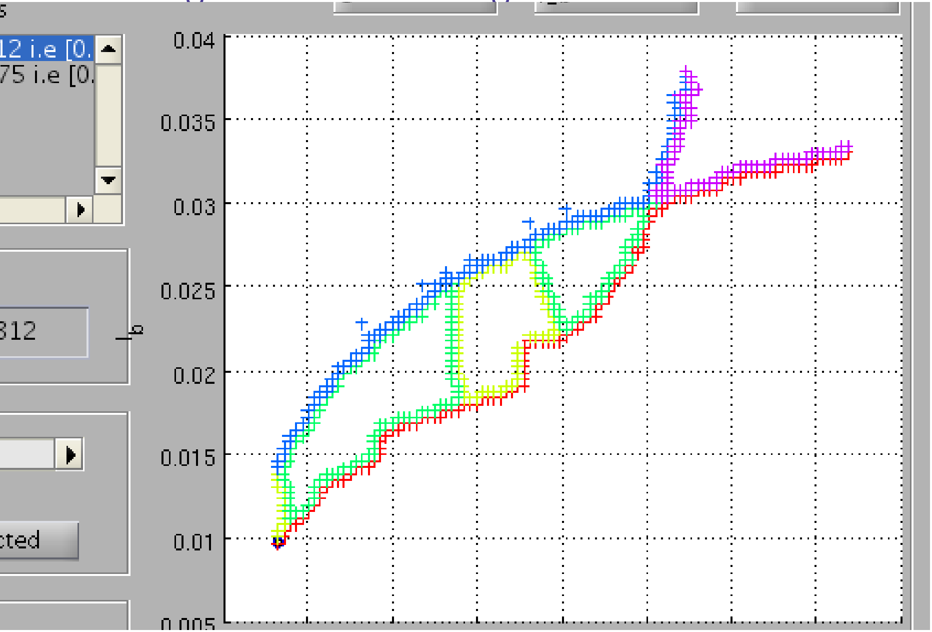
Result on a Single Trace



Partitioning the Parameter Region

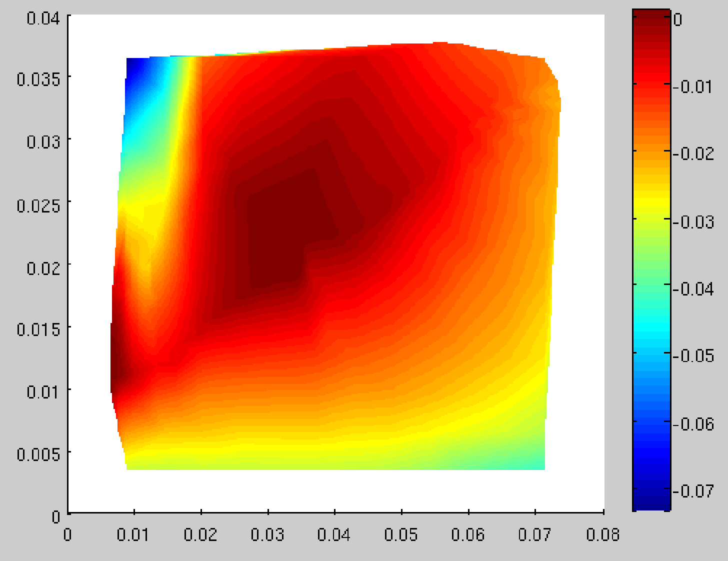


Partitioning the Parameter Region



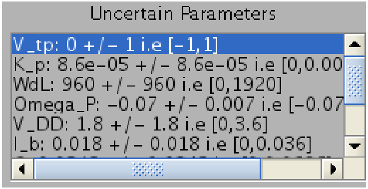
Satisfaction Function

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

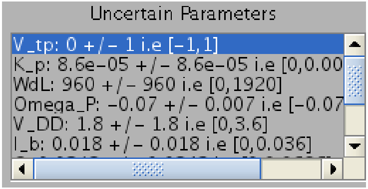


Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

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

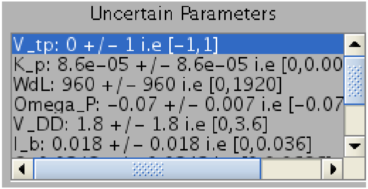


Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

It turned out those were perfectly valid oscillations

Finding Oscillations

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

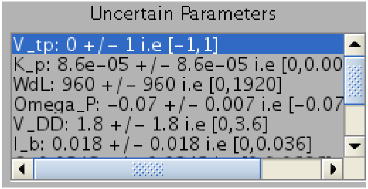


Using an implementation of the Nelder Mead optimization algorithm, Breach was able to find two parameter valuations satisfying the property in 98 s of computation time.

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.

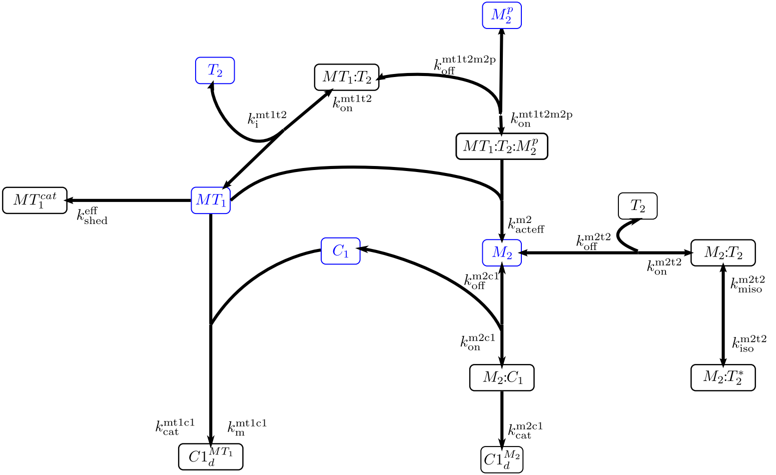
It turned out those were perfectly valid oscillations ... of period $T/4$ and $T/2$

Outline

Outline

An Enzymatic Network Involved in Angiogenesis

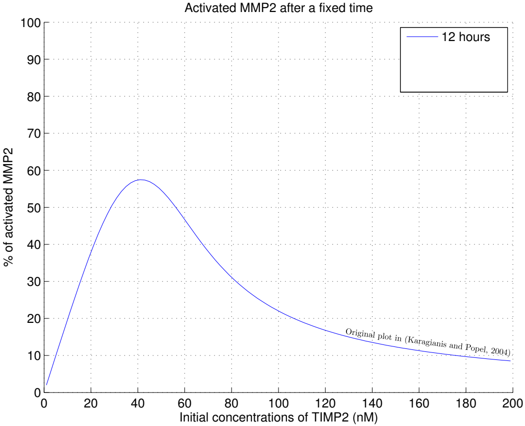
Collagen (C_1) degradation by matrix metalloproteinase (M_2^P) and membrane type 1 metalloproteinase (MT_1).



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. It turned out that steady state was not reached for $T_2(0) > 20$ nM.

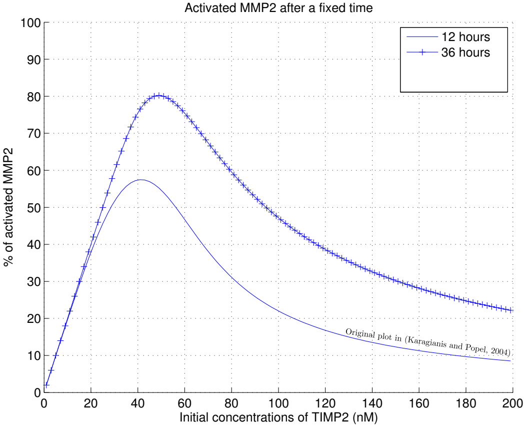
Using $\varphi \equiv \text{ev alw } (|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ we could guarantee the correct plot.



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.

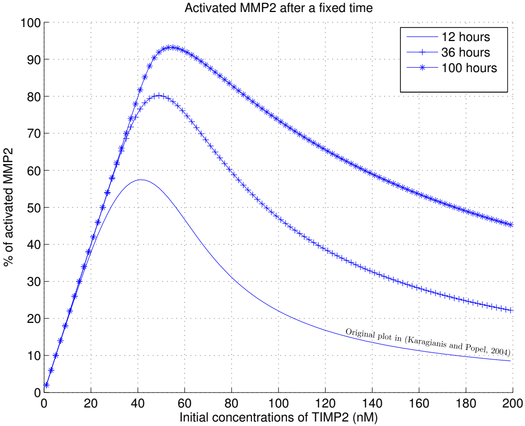
Using $\varphi \equiv$ ev alw $(|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ we could guarantee the correct plot.



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.

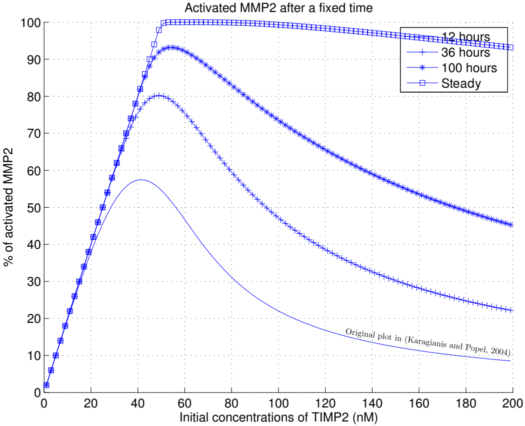
Using $\varphi \equiv$ ev alw $(|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ we could guarantee the correct plot.



Rigorous Steady State Analysis

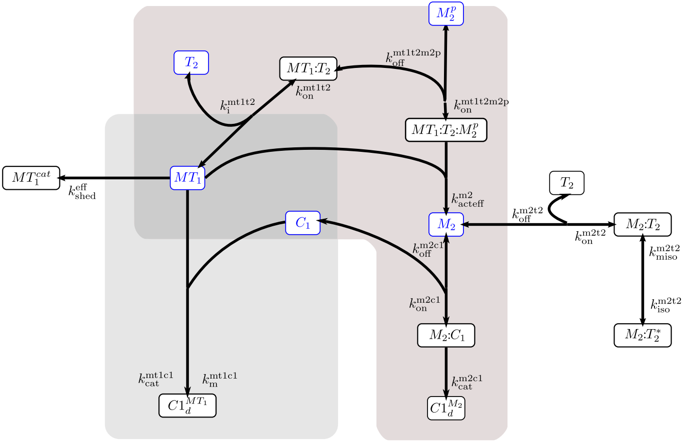
In [KP04], activation of M_2^P after 12h "Nearly steady state" for $T_2(0)$ between 0 and 200 nM. It turned out that steady state was not reached for $T_2(0) > 20$ nM.

Using $\varphi \equiv$ ev alw $(|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ we could guarantee the correct plot.



Open Model

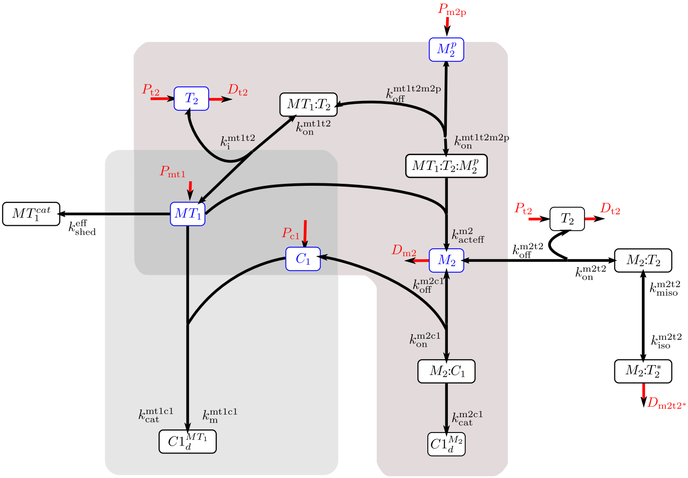
We extended the model by introducing production and degradation terms



More complex behaviors becomes possible, such as oscillatory regimes

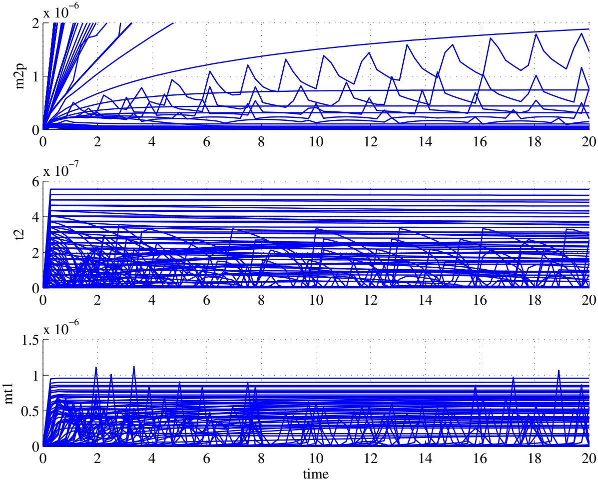
Open Model

We extended the model by introducing production and degradation terms

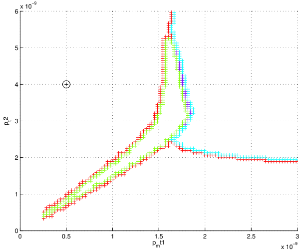


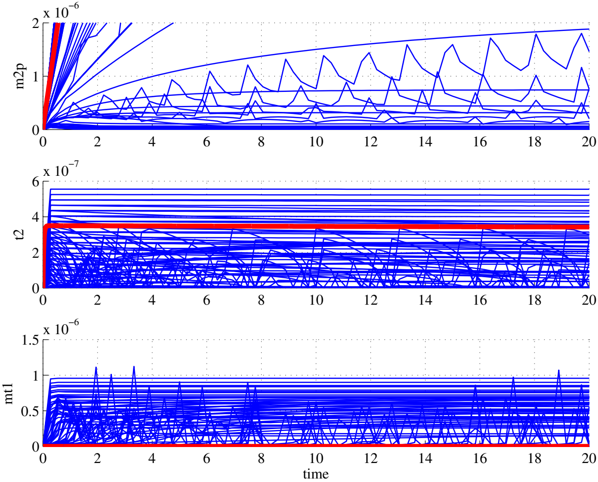
More complex behaviors becomes possible, such as oscillatory regimes

Oscillations Map

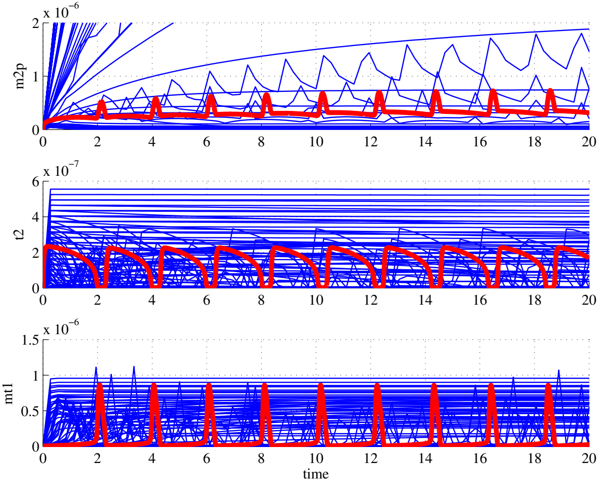


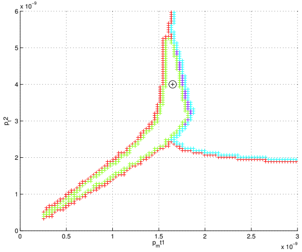
Oscillations Map



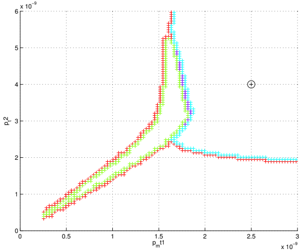


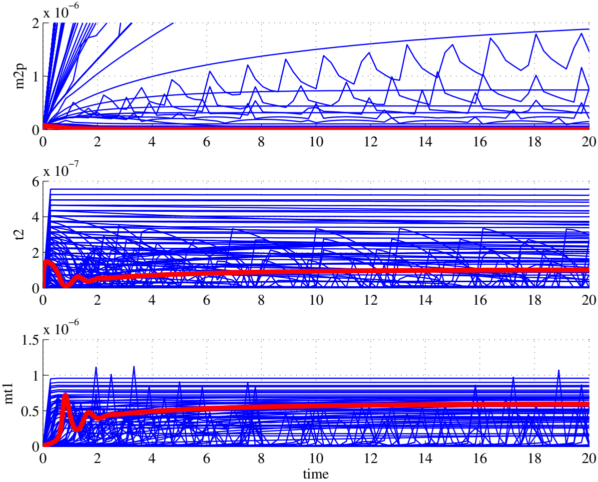
Oscillations Map



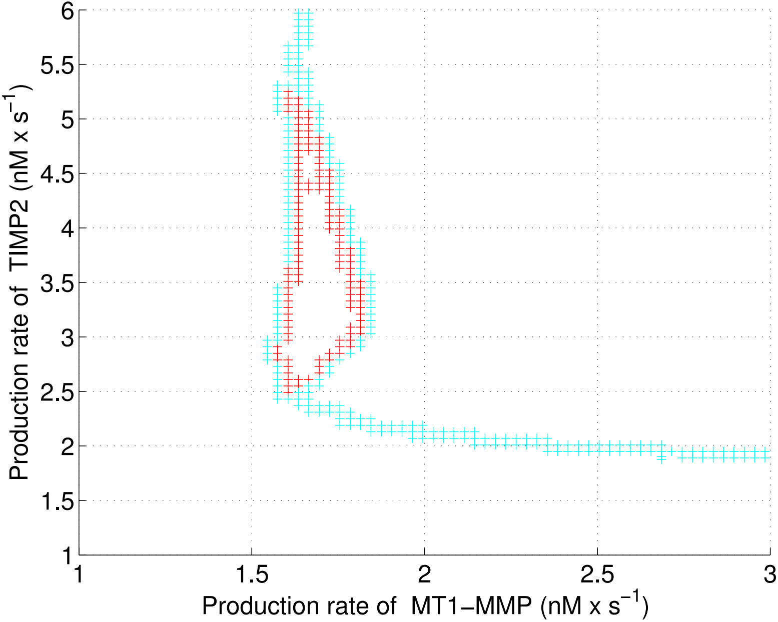


Oscillations Map

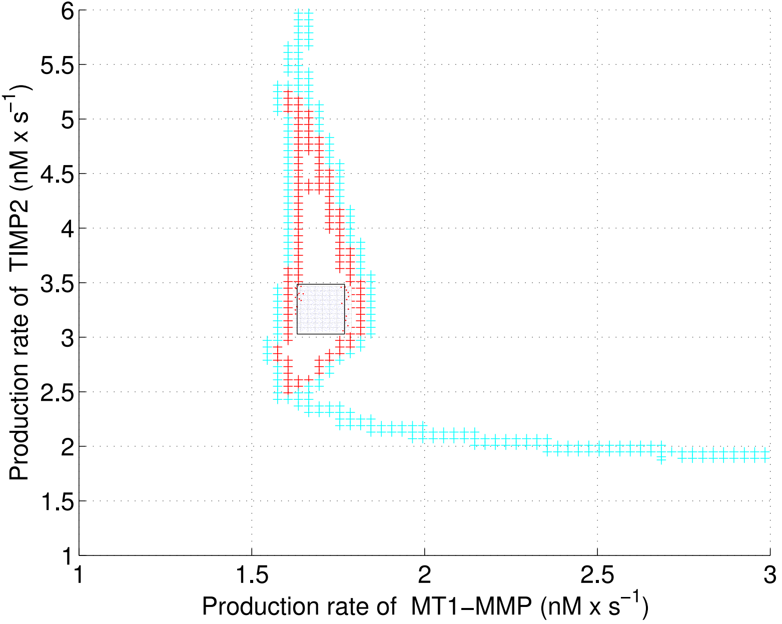




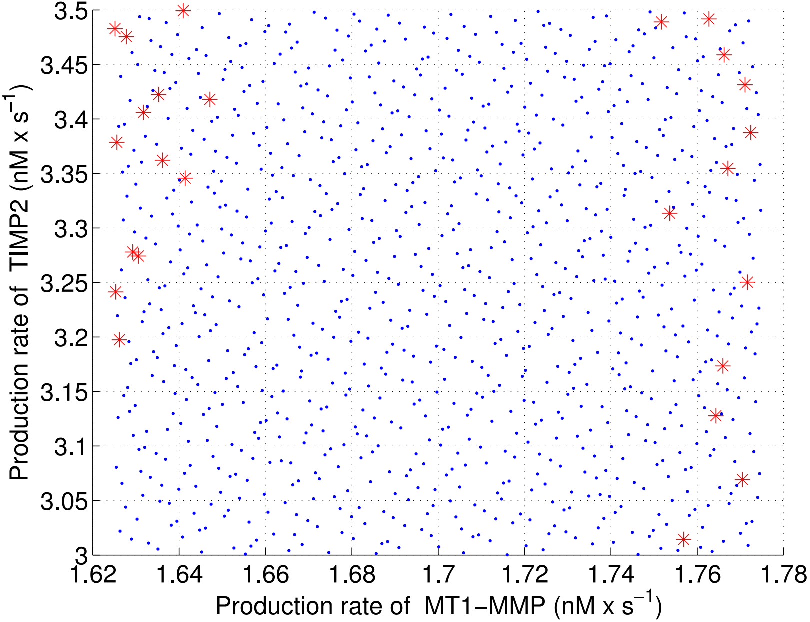
Oscillation, Robustness



Oscillation, Robustness



Oscillation, Robustness



Conclusion

Summary

- Specification language for hybrid systems behaviors, with a robust semantics
- An implementation with advanced simulation and parameter exploration features
- Case studies of parameter synthesis problems

Perspectives

- Going further with global robustness and sensitivity analysis for specifications
- Different optimization strategies for parameter synthesis/optimal control
- From robust satisfaction to *formal* specifications