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

Hybrid System

\[ \dot{x} = f_q(x, p) \mid \]

Param. \( p, x_0 \)

Robust Satisfaction of STL

Introduction

DREAM Seminar
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

Hybrid System

\[ \dot{x} = f_q(x, p) \]

Param. \( p, x_0 \)

Diagram showing a hybrid system with states \( q_0, q_1, q_2 \) and transitions between them.
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

\[
\dot{x} = f_q(x, p)
\]

Hybrid System

Simulation

\[
x(t, p)
\]
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

Hybrid System

\[
\dot{x} = f_q(x, p) \quad \text{Param. } p, x_0 \quad q_0 \rightarrow q_1 \rightarrow q_2
\]

Simulation

\[
q_0 \rightarrow q_1 \rightarrow \cdots \quad x(t, p)
\]

Monitoring

\[
(x, q) \models \varphi \quad \text{ok} \quad \neg \text{ok}
\]
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

\[
\dot{x} = f_q(x, p) \quad \parallel \quad P_{\text{ok}} \rightarrow q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow \neg \text{ok}
\]

Robust Satisfaction of STL
Overview

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

Hybrid System

\[
\dot{x} = f_q(x, p) \quad \text{Param. } p, x_0
\]

Simulation

\[
q_0 \rightarrow q_1 \rightarrow \cdots
\]

Property \( \varphi \equiv \text{alw} \left[ q_0 \rightarrow \text{ev}_{[0,1]} \quad q_2 \cup_{[0,2]} (x \geq 0.5) \right] \)
Overview

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

Hybrid System

\[
\dot{x} = f_q(x, p) || q_1 \rightarrow q_0 \\
\text{Param. } p, x_0 \\
\rightarrow q_0 \rightarrow q_2
\]

Simulation

\[
q_0 \rightarrow q_1 \rightarrow \cdots \\\nx(t, p)
\]

STL monitoring

Property \( \varphi \equiv \text{alw} [q_0 \rightarrow \text{ev}_{[0,1]} q_2 \cup_{[0,2]} (x \geq 0.5)] \)

\[\text{ok} \rightarrow \neg \text{ok}\]
Overview

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

Hybrid System

$$\dot{x} = f_q(x, p) \parallel q_0 \rightarrow q_1 \rightarrow \cdots$$

Param. $p, x_0$

Simulation

$$x(t, p) \pm \varepsilon$$

STL monitoring

Property $\varphi \equiv \text{alw}[q_0 \rightarrow \text{ev}_{[0,1]} q_2 \text{ } \cup_{[0,2]} (x \geq 0.5)]$

Robust Satisfaction of STL

Introduction

DREAM Seminar
Overview

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

Hybrid System

\[ \dot{x} = f_q(x, p) \quad |\quad q_0 \rightarrow q_1 \rightarrow q_2 \]

Param. \( p, x_0 \)

Simulation

\[ q_0 \rightarrow q_1 \rightarrow \cdots \]

\[ x(t, p) \pm \varepsilon \]

STL monitoring

Property \( \varphi \equiv \)

\[ \text{alw} \left[ q_0 \rightarrow \text{ev}_{[0,1]} \right] \]

\[ q_2 \cup_{[0,2]} \left( x \geq 0.5 \right) \]

\( \varepsilon \downarrow \)

\( \neg \text{ok} \)

\( \text{ok} \)
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 ($\neg$, $\land$, $\lor$) and temporal operators: “next”, “always” (alw), “eventually” (ev) and “until” ($U$)

Examples:

- $\varphi \varphi \varphi \varphi \cdots$ satisfies alw $\varphi$
- $\psi \psi \psi \varphi \psi \cdots$ satisfies ev $\varphi$
- $\varphi \varphi \varphi \varphi \psi \cdots$ satisfies $\varphi U \psi$
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)

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

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

Definition (STL Semantics)

The validity of a formula \( \varphi \) with respect to a signal \( x \) at time \( t \) is

\[
\begin{align*}
(x, t) \models \mu & \quad \Leftrightarrow \quad \mu(x[t]) > 0 \\
(x, t) \models \varphi \land \psi & \quad \Leftrightarrow \quad (x, t) \models \varphi \land (x, t) \models \psi \\
(x, t) \models \neg \varphi & \quad \Leftrightarrow \quad \neg((x, t) \models \varphi) \\
(x, t) \models \varphi \mathcal{U}_{[a,b]} \psi & \quad \Leftrightarrow \quad \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \psi \land \\
& \quad \forall t'' \in [t, t'], (x, t'') \models \varphi \}
\end{align*}
\]

Additionally:

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

**Definition (STL Syntax)**

\[ \varphi ::= \mu \mid \neg \varphi \mid \varphi \land \psi \mid \varphi \land [a,b] \psi \]

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

**Definition (STL Semantics)**

The validity of a formula \( \varphi \) with respect to a signal \( x \) at time \( t \) is

\[
\begin{align*}
(x, t) \models \mu & \iff \mu(x[t]) > 0 \\
(x, t) \models \varphi \land \psi & \iff (x, t) \models \varphi \land (x, t) \models \psi \\
(x, t) \models \neg \varphi & \iff \neg((x, t) \models \varphi) \\
(x, t) \models \varphi \land [a,b] \psi & \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \psi \land \\
& \quad \forall t'' \in [t, t'], (x, t'') \models \varphi
\end{align*}
\]

Additionally:

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

\[
\begin{align*}
\varphi &= x > 2 \\
\varphi &= \text{ev}_{[0,\infty]}(x > 2)
\end{align*}
\]
Examples

Consider a simple piecewise affine signal:

Truth value of :

- $\varphi = x > 2$
- $\varphi = \text{ev}_{[0,0.5]}(x > 2)$
Examples
Consider a simple piecewise affine signal:

![Graph of a piecewise affine signal with labels and axes]

Truth value of :

- $\varphi = x > 2$

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

Consider a simple piecewise affine signal:

Truth value of:

- $\varphi = x > 2$

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

<table>
<thead>
<tr>
<th>Condition</th>
<th>Equivalent Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x, t) \models \mu$</td>
<td>$\iff \mu(x[t]) &gt; 0$</td>
</tr>
<tr>
<td>$(x, t) \models \neg \varphi$</td>
<td>$\iff (x, t) \not\models \varphi$</td>
</tr>
<tr>
<td>$(x, t) \models \varphi_1 \land \varphi_2$</td>
<td>$\iff (x, t) \models \varphi_1$ and $(x, t) \models \varphi_2$</td>
</tr>
<tr>
<td>$(x, t) \models \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$</td>
<td>$\iff \exists t' \in [t + a, t + b]$ s.t. $(x, t') \models \varphi_2$ and $\forall t'' \in [t, t'], (x, t'') \models \varphi_1$</td>
</tr>
</tbody>
</table>

### A Boolean Satisfaction Function $\chi$

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

- $\chi(\mu, x, t) = \text{sign}(\mu(x[t])) \times \infty$
- $\chi(\neg \varphi, x, t) = -\chi(\varphi, x, t)$
- $\chi(\varphi_1 \land \varphi_2, x, t) = \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t))$
- $\chi(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2, x, t) = \max_{t' \in [t + a, t + b]}(\min(\chi(\varphi_2, x, t'), \min_{s \in [t, t']} \chi(\varphi_1, x, s)))$

We can verify that $(x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty$.
From Semantics to Satisfaction Functions

**STL semantics**

\[(x, t) \models \mu \iff \mu(x[t]) > 0\]

\[(x, t) \models \neg \varphi \iff (x, t) \not\models \varphi\]

\[(x, t) \models \varphi_1 \land \varphi_2 \iff (x, t) \models \varphi_1 \text{ and } (x, t) \models \varphi_2\]

\[(x, t) \models \varphi_1 U_{[a, b]} \varphi_2 \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \text{ and } \forall t'' \in [t, t'], (x, t'') \models \varphi_1\]

**A Boolean Satisfaction Function \( \chi \)**

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

\[
\begin{align*}
\chi(\mu, x, t) &= \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg \varphi, x, t) &= -\chi(\varphi, x, t) \\
\chi(\varphi_1 \land \varphi_2, x, t) &= \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t)) \\
\chi(\varphi_1 U_{[a, b]} \varphi_2, x, t) &= \max_{\tau \in t+[a, b]} \min_s (\min(\chi(\varphi_2, x, \tau), \chi(\varphi_1, x, s))
\end{align*}
\]

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
From Semantics to Satisfaction Functions

STL semantics

\[(x, t) \models \mu \iff \mu(x[t]) > 0\]
\[(x, t) \models \neg \varphi \iff (x, t) \not\models \varphi\]
\[(x, t) \models \varphi_1 \land \varphi_2 \iff (x, t) \models \varphi_1 \text{ and } (x, t) \models \varphi_2\]
\[(x, t) \models \varphi_1 \mathcal{U}_{[a, b]} \varphi_2 \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \text{ and } \forall t'' \in [t, t'], (x, t'') \models \varphi_1\]

A Boolean Satisfaction Function \(\chi\)

Map \{false, true\} to \{−∞, ∞\} and define the function \(\chi : (x, t) \rightarrow \{−∞, ∞\}\):

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

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
From Semantics to Satisfaction Functions

**STL semantics**

\[(x, t) \models \mu \iff \mu(x[t]) > 0\]

\[(x, t) \models \neg \varphi \iff (x, t) \not\models \varphi\]

\[(x, t) \models \varphi_1 \land \varphi_2 \iff (x, t) \models \varphi_1 \text{ and } (x, t) \models \varphi_2\]

\[(x, t) \models \varphi_1 \mathcal{U}_{[a, b]} \varphi_2 \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \text{ and } \forall t'' \in [t, t'], (x, t'') \models \varphi_1\]

**A Boolean Satisfaction Function \( \chi \)**

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

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

\[
\chi(\neg \varphi, x, t) = -\chi(\varphi, x, t)
\]

\[
\chi(\varphi_1 \land \varphi_2, x, t) = \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t))
\]

\[
\chi(\varphi_1 \mathcal{U}_{[a, b]} \varphi_2, x, t) = \max_{\tau \in t + [a, b]} \left( \min_{s \in [t, \tau]} \chi(\varphi_2, x, s) \right)
\]

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
From Semantics to Satisfaction Functions

**STL semantics**

\((x, t) \models \mu \iff \mu(x[t]) > 0\)

\((x, t) \models \neg \varphi \iff (x, t) \not\models \varphi\)

\((x, t) \models \varphi_1 \land \varphi_2 \iff (x, t) \models \varphi_1 \text{ and } (x, t) \models \varphi_2\)

\((x, t) \models \varphi_1 U_{[a, b]} \varphi_2 \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \text{ and } \forall t'' \in [t, t'], (x, t'') \models \varphi_1\)

**A Boolean Satisfaction Function \(\chi\)**

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

\[
\begin{align*}
\chi(\mu, x, t) &= \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg \varphi, x, t) &= -\chi(\varphi, x, t) \\
\chi(\varphi_1 \land \varphi_2, x, t) &= \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t)) \\
\chi(\varphi_1 U_{[a, b]} \varphi_2, x, t) &= \max_{\tau \in t+[a, b]} \left(\min(\chi(\varphi_2, x, \tau), \min_{s \in [t, \tau]} \chi(\varphi_1, x, s))\right)
\end{align*}
\]

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
From Semantics to Satisfaction Functions

**STL semantics**

\[(x, t) \models \mu \iff \mu(x[t]) > 0\]
\[(x, t) \models \neg \varphi \iff (x, t) \not\models \varphi\]
\[(x, t) \models \varphi_1 \land \varphi_2 \iff (x, t) \models \varphi_1 \text{ and } (x, t) \models \varphi_2\]
\[(x, t) \models \varphi_1 U_{[a, b]} \varphi_2 \iff \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \text{ and } \forall t'' \in [t, t'], (x, t'') \models \varphi_1\]

**A Boolean Satisfaction Function \( \chi \)**

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

\[
\begin{align*}
\chi(\mu, x, t) &= \text{sign}(\mu(x[t])) \times \infty \\
\chi(\neg \varphi, x, t) &= -\chi(\varphi, x, t) \\
\chi(\varphi_1 \land \varphi_2, x, t) &= \min(\chi(\varphi_1, x, t), \chi(\varphi_2, x, t)) \\
\chi(\varphi_1 U_{[a, b]} \varphi_2, x, t) &= \max_{\tau \in t + [a, b]} \left( \min(\chi(\varphi_2, x, \tau), \min_{s \in [t, \tau]} \chi(\varphi_1, x, s)) \right)
\end{align*}
\]

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
From Semantics to Satisfaction Functions

STL semantics

\[(x, t) \models \mu \quad \iff \quad \mu(x[t]) > 0\]
\[(x, t) \models \neg \varphi \quad \iff \quad (x, t) \not\models \varphi\]
\[(x, t) \models \varphi_1 \land \varphi_2 \quad \iff \quad (x, t) \models \varphi_1 \land (x, t) \models \varphi_2\]
\[(x, t) \models \varphi_1 \mathcal{U}_{[a, b]} \varphi_2 \quad \iff \quad \exists t' \in [t + a, t + b] \text{ s.t. } (x, t') \models \varphi_2 \land \forall t'' \in [t, t'], (x, t'') \models \varphi_1\]

A Boolean Satisfaction Function \(\chi\)

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

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

We can verify that \((x, t) \models \varphi \iff \chi(\varphi, x, t) = +\infty\)
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 \( \mu \) to get a boolean signal

Simple idea

- Get rid of sign to get a quantitative satisfaction function \( \rho \)
- Keep the same inductive rules for the quantitative semantics:

\[
\begin{align*}
\rho(\mu, x, t) &= \mu(x[t]) \\
\rho(\neg \varphi, x, t) &= -\rho(\varphi, x, t) \\
\rho(\varphi_1 \land \varphi_2, x, t) &= \min(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)) \\
\rho(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2, x, t) &= \max_{\tau \in t+[a,b]} \left( \min(\rho(\varphi_2, x, \tau), \min_{s \in [t,\tau]} \rho(\varphi_1, x, s)) \right)
\end{align*}
\]
From Boolean to Quantitative Satisfaction Function

For atomic predicates:

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

The sign removes the quantitative information in \( \mu \) to get a boolean signal

Simple idea

- Get rid of sign to get a quantitative satisfaction function \( \rho \)
- Keep the same inductive rules for the quantitative semantics:

\[
\begin{align*}
\rho(\mu, x, t) &= \mu(x[t]) \\
\rho(\neg \varphi, x, t) &= -\rho(\varphi, x, t) \\
\rho(\varphi_1 \land \varphi_2, x, t) &= \min(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)) \\
\rho(\varphi_1 U_{[a,b]} \varphi_2, x, t) &= \max_{\tau \in [t+ [a,b]]} \left( \min(\rho(\varphi_2, x, \tau), \min_{s \in [t, \tau]} \rho(\varphi_1, x, s)) \right)
\end{align*}
\]
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 \( \mu \) to get a boolean signal.

Simple idea

- Get rid of sign to get a quantitative satisfaction function \( \rho \)
- Keep the same inductive rules for the quantitative semantics:

\[
\begin{align*}
\rho(\mu, x, t) & = \mu(x[t]) \\
\rho(\neg \varphi, x, t) & = -\rho(\varphi, x, t) \\
\rho(\varphi_1 \land \varphi_2, x, t) & = \min(\rho(\varphi_1, x, t), \rho(\varphi_2, x, t)) \\
\rho(\varphi_1 U_{[a,b]} \varphi_2, x, t) & = \max_{\tau \in t+[a,b]} \left( \min(\rho(\varphi_2, x, \tau), \min_{s \in [t,\tau]} \rho(\varphi_1, x, s)) \right)
\end{align*}
\]
Robust Satisfaction, Examples

Robust Satisfaction of STL

Temporal Logics for Continuous Time and Space

DREAM Seminar 12 / 39
Robust Satisfaction, Examples

\[ \mu \exists t_0 (\exists t_1 \exists t_2 [ x(t_1) \times \exists t_1 \times x(t_2) \times \exists t_2 ]) \]

\[ \nu \exists t_0 \exists t_1 (x(t_1) \times \exists t_1 \times x(t_2) \times \exists t_2) \]
Robust Satisfaction, Examples

\[ \text{nu3: } \psi \equiv [0.1 \land 1][t] \land x(t)[t] > 0 \land x(t)[t+1] < x(t)[t] \]

\[ \text{evo: } \psi([5, 1.5]) \equiv [0.1 \land 1][t] \land x(t)[t] > 0 \land x(t)[t+1] < x(t)[t] \]
Robust Satisfaction, Examples

- \( \nu \exists \bar{y} [0[t] \leq 1[t] + \exists 0[t] \leq 2[t] \leq 2[t] \)

- \( \exists \bar{y} \forall [b[t] \leq 0[t] \leq 1[t] + \exists 0[t] \leq 2[t] \leq 2[t] \]

Robust Satisfaction of STL

Temporal Logics for Continuous Time and Space

DREAM Seminar 12 / 39
Robust Satisfaction, Examples
Robust Satisfaction, Applications

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

- **STL Prop.** $\varphi$
- **Param.** $p \in \mathcal{P}$

Oracle
- **Model + STL Monitor**

Robust Sat. $\rho(\varphi, p)$

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 $\varphi$, 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

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

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 $\varphi$, given by

$$d(\varphi, P) = \{ p \in P \mid \rho(\varphi, p) > 0 \}$$
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 \textit{validity domain} of $\varphi$, given by

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

Breach deals with piecewise-continuous models of the form

\[
\begin{align*}
\dot{x} &= f(q, x, p), \quad x(0) = x_0 \\
y &= g(x) \\
q^+ &= e(q^-, y), \quad q(0) = q_0
\end{align*}
\]

where \( x \in \mathbb{R}^n \) is the state variable

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

\( p \in \mathbb{R}^{n_p} \) is the parameter vector,

\( g \) is the guard function and

\( e \) is the event or transition function, where \( q^+ \neq q^- \) only if \( g(x) = 0 \)
Simulation Algorithm

Discontinuity locking + Event detection by zero crossing detection

1. Let $f_k(x, p) = f(q(t_k), x, p)$ (block switching between $t_k$ and $t_{k+1}$)
2. Solve ODE $\dot{x} = f_k(x, p)$ on $[t_k, t_k + h_k]$
3. If for all $i$, $\text{sign}(g_i(x)) = \text{Constant}$ 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 $g_i(x(\tau)) = 0$ and let $t_{k+1} = \tau$
5. Return $\xi_p(t_{k+1})$ 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
- Built-in zero-crossing detection for guards.

Sensitivity functions $s_{ij}(t) = \frac{\partial x_i}{\partial p_j}(t)$ 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
  ⇒ 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
- Built-in zero-crossing detection for guards.

Sensitivity functions $s_{ij}(t) = \frac{\partial x_i}{\partial p_j}(t)$ 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
  ⇒ 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
- Built-in zero-crossing detection for guards.

Sensitivity functions $s_{ij}(t) = \frac{\partial x_i}{\partial p_j}(t)$ 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

Robust Satisfaction of STL
An Implementation: The Breach Toolbox
DREAM Seminar 18 / 39
Breach GUIs for trajectories exploration

Robust Satisfaction of STL
An Implementation: The Breach Toolbox

DREAM Seminar 18 / 39
Breach GUIs for trajectories exploration
Breach GUIs for trajectories exploration

Robust Satisfaction of STL
An Implementation: The Breach Toolbox

DREAM Seminar 18 / 39
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 logic formulas: atomic predicates

STL Syntax: \( \varphi := \mu \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi \ U_{[a,b]} \varphi \)

+ usual syntactic sugars for disjunction, eventually and always.

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

\begin{verbatim}
% 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
\end{verbatim}
Temporal logic formulas: formulas

STL Syntax: \[ \varphi := \mu \mid \neg \varphi \mid \varphi \land \varphi \mid \varphi U_{[a,b]} \varphi \]

+ 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 $\varphi$.

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

1. (operator $\neg$) compute $\forall t, z[t] = -y[t]$;
2. (operator $\land$) compute $\forall t, z[t] = \min(y[t], y'[t])$
3. (operator $\cup$) 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 $\varphi$.

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

1. (operator $\neg$) compute $\forall t, z[t] = -y[t]$;
2. (operator $\land$) 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 $\varphi$.

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

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

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 $\varphi$.

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 $\land$) compute $\forall t, z[t] = \min(y[t], y'[t])$
3. (operator $\cup$) 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 $\varphi = (x > 0) \cup_{[0,1)} (x > 0) \cup_{[0,1)} (x > 0) \cdots$ $i$ times

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

<table>
<thead>
<tr>
<th>$i$</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34747</td>
</tr>
<tr>
<td>2</td>
<td>0.46335</td>
</tr>
<tr>
<td>3</td>
<td>0.60599</td>
</tr>
<tr>
<td>4</td>
<td>0.76067</td>
</tr>
<tr>
<td>5</td>
<td>0.89201</td>
</tr>
<tr>
<td>6</td>
<td>1.03761</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input size</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31416</td>
<td>0.18402</td>
</tr>
<tr>
<td>345566</td>
<td>0.40761</td>
</tr>
<tr>
<td>659716</td>
<td>0.75508</td>
</tr>
<tr>
<td>973866</td>
<td>1.09268</td>
</tr>
<tr>
<td>1288016</td>
<td>1.4587</td>
</tr>
</tbody>
</table>
Computational Cost, Some Experiments

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

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

(a)  

<table>
<thead>
<tr>
<th>$i$</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.34747</td>
</tr>
<tr>
<td>2</td>
<td>0.46335</td>
</tr>
<tr>
<td>3</td>
<td>0.60599</td>
</tr>
<tr>
<td>4</td>
<td>0.76067</td>
</tr>
<tr>
<td>5</td>
<td>0.89201</td>
</tr>
<tr>
<td>6</td>
<td>1.03761</td>
</tr>
</tbody>
</table>

(b)  

<table>
<thead>
<tr>
<th>input size</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31416</td>
<td>0.18402</td>
</tr>
<tr>
<td>345566</td>
<td>0.40761</td>
</tr>
<tr>
<td>659716</td>
<td>0.75508</td>
</tr>
<tr>
<td>973866</td>
<td>1.09268</td>
</tr>
<tr>
<td>1288016</td>
<td>1.4587</td>
</tr>
</tbody>
</table>
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
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.)
We look for oscillations of period $T$ and given minimum and maximum amplitudes around 0

Above and below a minimum amplitude
mu0: $IL1[t] > A_{min}$
mu1: $IL1[t] < -A_{min}$

Bounded by a maximum amplitude
mu2: $\text{abs}(IL1[t]) < A_{max}$

(almost) Strict periodicity
mu3: $(\text{abs}(IL1[t] - IL1[t-T]) < \epsilon_i)$
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

Fixed Parameters

- VD1: 0
- VD2: 0
- IL1: 0
- V_tp: -0.69
- K_p: 8.6e-05
- WdL: 960
- Omega_P: -0.07
- V_DD: 1.8
- I_b: 0.02
- C: 0.04
- V_ctrl: 0
- L: 28.57
- R: 3.7
- T: 7
- p1: 0.001
- p2: 0.05
- p3: 0.01
- p4: 0.1

Uncertain Parameters

- C: 0.04 +/- 0.0339 i.e [0.0061, 0.073]
- I_b: 0.02 +/- 0.018 i.e [0.002, 0.038]

Add =>

<= Rem

Modify current subset

Value (pts) | Range (eps)
---|---
0.04 | 0.0339

Select subset(s)

Copy selected | Delete selected

Refine subset(s)
Breach Interface

\[
p_{\text{shift}}: ((|I_{L1}[t]| - |I_{L1}[t-T]|)^2 < p1)
\]
\[
\text{oscillT}: (ev(alw_0, 4*T)((ev_0[T](|I_{L1}[t]| > p2)) \text{ and } (ev_{\text{maxA}}(|I_{L1}[t]|^2 < p3))
\]
\[
\text{oscillsansT}: ev(alw_0, 4*T)((ev_0[T](|I_{L1}[t]| > p2)) \text{ and } (alw_{\text{maxA}}(alw(|I_{L1}[t]|^2 < p3))
\]
\[
ev_{\text{alw_maxA}}: ev(alw_0, 4*T)((|I_{L1}[t]|^2 < p3))
\]
Result on a Single Trace

Robust Satisfaction of STL Applications

DREAM Seminar 29 / 39
Result on a Single Trace
Partitioning the Parameter Region

Robust Satisfaction of STL Applications
Partitioning the Parameter Region

Robust Satisfaction of STL Applications

DREAM Seminar 30 / 39
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

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
An Enzymatic Network Involved in Angiogenesis

Collagen ($C_1$) degradation by matrix metalloproteinase ($M^P_2$) 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 $\varphi \equiv \text{ev alw } (|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ 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 $\varphi \equiv \text{ev alw } (|\dot{M}_2(t)| < \epsilon \times M_2^P(0))$ 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 $\varphi \equiv \text{ev alw } \left( |\dot{M}_2(t)| < \epsilon \times M_2^P(0) \right)$ we could guarantee the correct plot.

![Graph showing activated MMP2 after a fixed time vs. initial concentrations of TIMP2 (nM)].
Open Model

We extended the model by introducing production and degradation terms. More complex behaviors become possible, such as oscillatory regimes.
Open Model

We extended the model by introducing production and degradation terms

More complex behaviors become possible, such as oscillatory regimes

Robust Satisfaction of STL

Applications

DREAM Seminar 36 / 39
Oscillations Map
Oscillations Map

Robust Satisfaction of STL Applications

DREAM Seminar 37 / 39
Oscillations Map

Robust Satisfaction of STL Applications
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