Requirement Synthesis for Industrial-scale Control Systems Design

Alexandre Donzé

Joint work with
Xiaoqing Jin, Jyotirmoy V. Deshmukh, Sanjit A. Seshia

University of California, Berkeley
Toyota Technical Center, Los Angeles
University of California, Riverside

June 12, 2013
Control Systems

Technically, cyberphysical systems integrating continuous dynamics, switching logics, etc.
The model-based design (MBD) V design process.
Model-Based Design

The actual design process.

- Alternation between specification and design
- A flavor of *chicken and egg* problem
Motivations for Requirement Synthesis

Specifications should be objects of similar nature as the design itself.

This enables

- co-developement of design and specifications
- automatization of verification and testing

However

- this is not (yet) the case: specifications are often high level, vague textual/oral/implicit requirements
- this was not the case: problems of reusability of older component (legacy code).

To construct/reconstruct formal and usable specifications, there is a need for requirement synthesis techniques.
Challenges

Closed-loop setting very complex

- nonlinear dynamics
- look-up tables
- large amounts of switching
- components with no models
- unclear semantics of modeling language

We can assume that all we have is

- the ability to simulate the system
- some vague idea of what the system should satisfy
- the ability to check if simulation traces satisfy properties
Requirement Synthesis

Our approach takes two major ingredients

- A specification language to formalize requirements
- A counter-example guided inductive synthesis approach to generate requirements in this language
1. Signal Temporal Logics
   - Definition and Examples
   - Template Requirements

2. Requirement Synthesis Framework
   - Algorithm
   - Parameter Synthesis for PSTL
   - STL Falsification

3. Experimental Results
1 Signal Temporal Logics
   • Definition and Examples
   • Template Requirements

2 Requirement Synthesis Framework
   • Algorithm
   • Parameter Synthesis for PSTL
   • STL Falsification

3 Experimental Results
Temporal logics specify patterns that timed behaviors of systems may or may not satisfy.

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” (G), “eventually” (F) and “until” ($U$)
From LTL to STL

Extension of LTL with **real-time** and **real-valued** constraints

MITL $G( r \Rightarrow F_{[0,.5s]} g )$
Boolean predicates, real-time

STL $G( x[t] > 0 \Rightarrow F_{[0,.5s]} y[t] > 0 )$
Predicates over real values, real-time
From LTL to STL

Extension of LTL with real-time and real-valued constraints

**Ex: request-grant property**

**LTL** \( G( r \implies F g) \)

Boolean predicates, discrete-time
From LTL to STL

Extension of LTL with **real-time** and **real-valued** constraints

**Ex: request-grant property**

**LTL** \( G( r \implies F g) \)
Boolean predicates, discrete-time

**MITL** \( G( r \implies F_{[0, .5s]} g) \)
Boolean predicates, real-time
From LTL to STL

Extension of LTL with **real-time** and **real-valued** constraints

### Ex: request-grant property

<table>
<thead>
<tr>
<th>LTL</th>
<th>( G( r \Rightarrow F g) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolean predicates, discrete-time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MITL</th>
<th>( G( r \Rightarrow F_{[0,0.5s]} g) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boolean predicates, real-time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STL</th>
<th>( G( x[t] &gt; 0 \Rightarrow F_{[0,0.5s]} y[t] &gt; 0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Predicates over real values, real-time</td>
</tr>
</tbody>
</table>
Formal Definitions

Definition (STL Syntax)

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

where \( \mu \) is a predicate of the form \( \mu : f(x[t]) > 0 \)
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 \( \mu : f(x[t]) > 0 \)

Definition (STL Semantics)

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

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

Additionally: \( F_{[a,b]} \varphi = \top \mathcal{U}_{[a,b]} \varphi \) and \( G_{[a,b]} \varphi = \neg ( F_{[a,b]} \neg \varphi) \).
STL Examples
The signal is never above 3.5

\[ \varphi := G \left( x[t] < 3.5 \right) \]
Between 2s and 6s the signal is between -2 and 2

\[ \varphi := G_{[2,6]} \ (|x[t]| < 2) \]
STL Examples

Always $|x| > 0.5 \Rightarrow$ after 1 s, $|x|$ settles under 0.5 for 1.5 s

$\varphi := G(x[t] > .5 \rightarrow F_{[0,.6]} \left( G_{[0,1.5]} x[t] < 0.5 \right))$
1 Signal Temporal Logics
   • Definition and Examples
   • Template Requirements

2 Requirement Synthesis Framework
   • Algorithm
   • Parameter Synthesis for PSTL
   • STL Falsification

3 Experimental Results
Informally, a PSTL formula is an STL formula where (some) numeric constants are left unspecified, represented by symbolic parameters.

**Definition (PSTL syntax)**

\[ \varphi : = \mu(x[t]) > \pi \ | \ \neg \varphi \ | \ \varphi \land \psi \ | \ \varphi \ U_{[\tau_1, \tau_2]} \psi \]

where

- \( \pi \) is a **scale** parameter
- \( \tau_1, \tau_2 \) are **time** parameters
Parametric STL - Illustration

"After 2s, the signal is never above 3"

ϕ := F [2, ∞) (x[t] < 3)
Parametric STL - Illustration

“After 2s, the signal is never above 3”

\[ \varphi := F_{[2, \infty]} (x[t] < 3) \]
“After $\tau$ s, the signal is never above $\pi$”

$\varphi := G_{[\tau, \infty]} \ (x[t] < \pi)$
Template Requirement Examples

Consider the following automatic transmission system:

- What is the maximum speed that the vehicle can reach?
- What is the minimum dwell time in a given gear?
- etc
the speed is always below $\pi_1$ and RPM below $\pi_2$

$$\varphi_{sp\_rpm}(\pi_1, \pi_2) := G( (\text{speed} < \pi_1) \land (\text{RPM} < \pi_2) ).$$
Template Requirements Examples

- **the speed is always below $\pi_1$ and RPM below $\pi_2$**

  $$\varphi_{sp\_rpm}(\pi_1, \pi_2) := G ( (\text{speed} < \pi_1) \land (\text{RPM} < \pi_2) ) .$$

- **the vehicle cannot reach 100 mph in $\tau$ seconds with RPM always below $\pi$**

  $$\varphi_{rpm100}(\tau, \pi) := \neg ( F_{[0, \tau]} (\text{speed} > 100) \land G (\text{RPM} < \pi) ) .$$
the speed is always below $\pi_1$ and RPM below $\pi_2$

$$\varphi_{sp\_rpm}(\pi_1, \pi_2) := G\left((\text{speed} < \pi_1) \land (\text{RPM} < \pi_2)\right).$$

the vehicle cannot reach 100 mph in $\tau$ seconds with RPM always below $\pi$

$$\varphi_{rpm100}(\tau, \pi) := \neg( F_{[0,\tau]} (\text{speed} > 100) \land G(\text{RPM} < \pi)).$$

whenever it shift to gear 2, it dwells in gear 2 for at least $\tau$ seconds

$$\varphi_{stay}(\tau) := G\left(\left(\neg \big(\text{gear} \neq 2 \land F_{[0,\varepsilon]} \text{gear} = 2\big) \implies G_{[\varepsilon,\tau]} \text{gear} = 2\right)\right).$$
1 Signal Temporal Logics
   • Definition and Examples
   • Template Requirements

2 Requirement Synthesis Framework
   • Algorithm
   • Parameter Synthesis for PSTL
   • STL Falsification

3 Experimental Results
Requirement Synthesis Algorithm

\[
\begin{align*}
&F[0, \tau_1](x_1 < \pi_1 \land G[0, \tau_2](x_2 > \pi_2)) \\
&\tau_1 \leftarrow 0.7, \pi_1 \leftarrow 2, \pi_2 \leftarrow 3 \\
&F[0, 1.1](x_1 < 3.7 \land G[0, 5](x_2 > 0.1))
\end{align*}
\]

\[\text{CounterexampleFound}\]

No Counterexample
Requirement Synthesis Algorithm

\[
F_{[0,\tau_1]}(x_1 < \pi_1 \land G_{[0,\tau_2]}(x_2 > \pi_2))
\]

Template Specification
Requirement Synthesis Algorithm

\[ F_{[0, \tau_1]}(x_1 < \pi_1 \land G_{[0, \tau_2]}(x_2 > \pi_2)) \]

Template Specification
Requirement Synthesis Algorithm

\[ F_{[0,\tau_1]}(x_1 < \pi_1 \land G_{[0,\tau_2]}(x_2 > \pi_2)) \]

FindParam

Simulation Traces

Candidate Requirement

\( \tau_1 \leftarrow .7, \pi_1 \leftarrow 2, \pi_2 \leftarrow 3 \)

Controller

Plant Model

Simulation Traces

init

Controller + e u

y
Requirement Synthesis Algorithm

\[
F_{[0, \tau_1]}(x_1 < \pi_1 \land G_{[0, \tau_2]}(x_2 > \pi_2))
\]

\[
\tau_1 \leftarrow 0.7, \ pi_1 \leftarrow 2, \ pi_2 \leftarrow 3
\]
Requirement Synthesis Algorithm

\[ F_{[0, \tau_1]}(x_1 < \pi_1 \land G_{[0, \tau_2]}(x_2 > \pi_2)) \]

\[ \tau_1 \leftarrow 0.7, \pi_1 \leftarrow 2, \pi_2 \leftarrow 3 \]
Requirement Synthesis Algorithm

\[
F_{[0,\tau_1]}(x_1 < \pi_1 \land G_{[0,\tau_2]}(x_2 > \pi_2))
\]

**Template Specification**

\[
\tau_1 \leftarrow 1.1, \pi_1 \leftarrow 3.7, \pi_2 \leftarrow 5
\]
**Requirement Synthesis Algorithm**

- **Controller**: \( e \rightarrow u \)
- **Plant Model**: \( y \)
- **Simulation Traces**
- **Counterexample Traces**
- **FindParam**
- **Candidate Requirement**
- **FalsifyAlgo**

**Template Specification**

\[
F_{[0, \tau_1]}(x_1 < \pi_1 \land G_{[0, \tau_2]}(x_2 > \pi_2))
\]

**Inferred Specification**

\[
F_{[0, 1.1]}(x_1 < 3.7 \land G_{[0, 5]}(x_2 > 0.1))
\]
1. Signal Temporal Logics
   - Definition and Examples
   - Template Requirements

2. Requirement Synthesis Framework
   - Algorithm
   - Parameter Synthesis for PSTL
   - STL Falsification

3. Experimental Results
Parameter synthesis for PSTL

Problem

Given a system $S$ with a PSTL formula with $n$ symbolic parameters $\varphi(p_1, \ldots, p_n)$, find a tight valuation function $v$ such that

$$ x, t \models \varphi(v(p_1), \ldots, v(p_n)),$$

Informally, a valuation $v$ is tight if there exists a valuation $v'$ in a $\delta$-close neighborhood of $v$, with $\delta$ “small”, such that

$$ x, t \not\models \varphi(v'(p_1), \ldots, v'(p_n)).$$
Example

\[ \varphi := G \left( x[t] > \pi \rightarrow F_{[0, \tau_1]} \ ( G_{[0, \tau_2]} x[t] < \pi) \right) \]
Example

\[ \varphi := G(x[t] > \pi \rightarrow F_{[0, \tau_1]} (G_{[0, \tau_2]} x[t] < \pi)) \]

- Valuation 1: \( \pi \leftarrow 1.5, \tau_1 \leftarrow 1 \text{ s}, \tau_2 \leftarrow 1.15 \text{ s} \)
Example

\[ \varphi := G \left( x[t] > \pi \rightarrow F_{[0, \tau_1]} \ \left( G_{[0, \tau_2]} \ x[t] < \pi \right) \right) \]

- Valuation 1: \( \pi \leftarrow 1.5, \tau_1 \leftarrow 1 \text{ s}, \tau_2 \leftarrow 1.15 \text{ s} \)
- Valuation 2 (tight): \( \pi \leftarrow .5, \tau_1 \leftarrow 0.65 \text{ s}, \tau_2 \leftarrow 2 \text{ s} \)
Parameter synthesis

Challenges

- Multiple solutions: which one to chose?
- Tightness implies to “optimize” the valuation $v(p_i)$ for each $p_i$

The problem can be greatly simplified if the formula is *monotonic* in each $p_i$. 

A PSTL formula $\phi(p_1, \cdots, p_n)$ is monotonically increasing wrt $p_i$ if

\[
\forall x,v,v', v(p_1) = v'(p_j), j \neq i \implies x|\phi = \phi(v'(p_1), \cdots, v'(p_i), \cdots)
\]

\[
v'(p_i) \geq v(p_i)
\]

It is monotonically decreasing if this holds when replacing $v'(p_i) \geq v(p_i)$ with $v'(p_i) \leq v(p_i)$. 

Alexandre Donzé

Requirement Synthesis for Control Systems ExCAPE Summer School 2013
Parameter synthesis

Challenges

- Multiple solutions: which one to chose?
- Tightness implies to “optimize” the valuation $v(p_i)$ for each $p_i$

The problem can be greatly simplified if the formula is *monotonic* in each $p_i$.

**Definition**

A PSTL formula $\varphi(p_1, \ldots, p_n)$ is monotonically increasing wrt $p_i$ if

$$\forall x, v, v', \left( \begin{array}{l} x \models \varphi(v(p_1), \ldots, v(p_i), \ldots) \\ v(p_j) = v'(p_j), j \neq i \\ v'(p_i) \geq v(p_i) \end{array} \right) \Rightarrow x \models \varphi(v'(p_1), \ldots, v'(p_i), \ldots)$$

It is monotonically decreasing if this holds when replacing $v'(p_i) \geq v(p_i)$ with $v'(p_i) \leq v(p_i)$. 
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics

![Diagram showing monotonic validity domain](image)
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics

![Diagram showing the validity domain $D(x, \varphi)$ with a Pareto front structure.](image)
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics

\begin{align*}
p_1 & \notin D(x, \varphi) \\
& \subseteq D(x, \varphi)
\end{align*}
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monoticity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics
Monotonic Validity Domains

- The validity domain $D$ of $\varphi$ and $x$ is the set of valuations $v$ s.t. $x \models \varphi(v)$
- A tight valuation is a valuation in $D$ close to its boundary $\partial D$
- In case of monotonicity, $\partial D$ has the structure of a Pareto front which can be estimated with generalized binary search heuristics
Deciding Monotonicity

Simple cases

- $f(x) > \pi \downarrow$ $\quad f(x) < \pi \uparrow$
- $G_{[0,\tau]} \varphi \downarrow$ $\quad F_{[0,\tau]} \varphi \uparrow$
- etc
Deciding Monotonicity

Simple cases

- $f(x) > \pi \downarrow f(x) < \pi \uparrow$
- $G_{[0,\tau]} \varphi \downarrow F_{[0,\tau]} \varphi \uparrow$
- etc

General case

- Deciding monotonicity can be encoded in an SMT query
- However, the problem is undecidable, due to undecidability of STL
- In practice, monotonicity can be decided easily (in our experience so far)
1 Signal Temporal Logics
   • Definition and Examples
   • Template Requirements

2 Requirement Synthesis Framework
   • Algorithm
   • Parameter Synthesis for PSTL
   • STL Falsification

3 Experimental Results
Solving the Falsification problem

Problem

*Given the system:*

\[ u(t) \xrightarrow{} \text{System } S \xrightarrow{} S(u(t)) \]

*Find an input signal* \( u \in \mathcal{U} \) *such that* \( S(u(t)), 0 \not\models \varphi \)
Solving the Falsification problem

**Problem**

*Given the system:*

\[ u(t) \rightarrow \text{System } S \rightarrow S(u(t)) \]

*Find an input signal* \( u \in \mathcal{U} \) *such that* \( S(u(t)), 0 \not\models \varphi \)

**In practice**

- We parameterize \( \mathcal{U} \) and reduce the problem to a parameter synthesis problem within some set \( \mathcal{P}_u \)
- The search of a solution is guided by a quantitative measure of satisfaction of \( \varphi \)
Parameterizing the Input Space

Input parameter set $\mathcal{P}_u$

Input signals $u(t) \in \mathcal{U}$

Note
The set of input signals generated by $\mathcal{P}_u$ is in general a subset of $\mathcal{U}$
I.e., we do not guarantee completeness.
Falsification with Quantitative Satisfaction

Given a formula $\varphi$, a signal $x$ and a time $t$, define a function $\rho$:

\[
\rho(\varphi, x, t) > 0 \Rightarrow x, t \models \varphi
\]
\[
\rho(\varphi, x, t) < 0 \Rightarrow x, t \not\models \varphi
\]

As $x$ is obtained by simulation using input parameters $p_u$, the falsification problem can be reduced to solving

\[
\rho^* = \min_{p_u \in P} \rho(\varphi, x, 0)
\]

If $\rho^* < 0$, we found a counterexample.
Falsification with Quantitative Satisfaction

Given a formula $\varphi$, a signal $x$ and a time $t$, define a function $\rho$:

$$
\rho(\varphi, x, t) > 0 \Rightarrow x, t \models \varphi \\
\rho(\varphi, x, t) < 0 \Rightarrow x, t \not\models \varphi
$$

As $x$ is obtained by simulation using input parameters $p_u$, the falsification problem can be reduced to solving

$$
\rho^* = \min_{p_u \in P_u} \rho(\varphi, x, 0)
$$

If $\rho^* < 0$, we found a counterexample.
Optimizing Satisfaction Function

Solving

$$\rho^* = \min_{p_u \in P_u} F(p_u) = \rho(\varphi, x, 0)$$

is difficult in general, as nothing can be assumed on $F$.

In practice, any global nonlinear, stochastic optimization algorithm can be used. Success will depend on how smooth is $F_u$, its local optima, etc.

Critical is the ability to compute $\rho$ efficiently.

Next we present the standard definition of $\rho$ and an efficient method to compute it.
Computing the Satisfaction Function

\( \rho(\varphi, \cdot, \cdot) \) transforms \( x \) into a *satisfaction* signal, that we note \( \varphi(x)[t] \).
Computing the Satisfaction Function

\( \rho(\varphi, \cdot, \cdot) \) transforms \( x \) into a satisfaction signal, that we note \( \varphi(x)[t] \).
Computing the Satisfaction Function

\( \rho(\varphi, \cdot, \cdot) \) transforms \( x \) into a satisfaction signal, that we note \( \varphi(x)[t] \).
Computing the Satisfaction Function

\[ \rho(\varphi, \cdot, \cdot) \] transforms \( x \) into a satisfaction signal, that we note \( \varphi(x)[t] \).

\[ \varphi(x)[t] \] can be computed inductively on the structure of \( \varphi \)

All we need is to define transducers for each atomic predicate and operators of STL.
STL Atomic Transducers

Predicate $x > 5$

$\mu(x) = x - 5$

$x[t] \rightarrow x[t] - 5$
STL Atomic Transducers

\[ x[t] \rightarrow \text{Predicate } x > 5 \]
\[ \mu(x) = x - 5 \]
\[ x[t] - 5 \]

\[ \varphi(x)[t] \rightarrow \text{Negation } \neg \varphi \]
\[ \psi(x) = -\varphi(x) \]
\[ -\varphi(x)[t] \]
STL Atomic Transducers

$x[t] \rightarrow \text{Predicate } x > 5$
$\mu(x) = x - 5 \rightarrow x[t] - 5$

$\varphi(x)[t] \rightarrow \text{Negation } \neg \varphi$
$\psi(x) = \neg \varphi(x) \rightarrow \neg \varphi(x)[t]$

$\varphi_1(x)[t] \rightarrow \text{Conjunction } \varphi_1 \land \varphi_2$
$\psi(x) = \min(\varphi_1(x), \varphi_2(x)) \rightarrow \min(\varphi_1(x)[t], \varphi_2(x)[t])$

$\varphi_2(x)[t] \rightarrow $
STL Atomic Transducers

Eventually $F_{[1,2]}\varphi$

$$\psi(x) = \max_{[t+.1,t+.2]} \varphi(x)$$

$$\max_{t'\in[t+.1,t+.2]} \varphi(x)[t']$$
STL Atomic Transducers

\[ \varphi(x)[t] \quad \text{Eventually} \quad F_{[1,2]} \varphi \quad \psi(x) = \max_{[t+.1,t+.2]} \varphi(x) \quad \max_{t' \in [t+.1,t+.2]} \varphi(x)[t'] \]

\[ \varphi(x)[t] \quad \text{Always} \quad G_{[1,2]} \varphi \quad \psi(x) = \min_{[t+.1,t+.2]} \varphi(x) \quad \min_{t' \in [t+.1,t+.2]} \varphi(x)[t'] \]
STL Atomic Transducers

**Eventually** \( F_{[1,2]} \varphi \)

\[
\psi(x) = \max_{[t+.1,t+.2]} \varphi(x)
\]

\[
\varphi(x)[t] \rightarrow \max_{t' \in [t+.1,t+.2]} \varphi(x)[t']
\]

**Always** \( G_{[1,2]} \varphi \)

\[
\psi(x) = \min_{[t+.1,t+.2]} \varphi(x)
\]

\[
\varphi(x)[t] \rightarrow \min_{t' \in [t+.1,t+.2]} \varphi(x)[t']
\]

**Note**

- The “Until” can be rewritten by a combination of untimed until, timed \( F \) and \( G \).
Computing the Robust Satisfaction Function
(Donze, Ferrere, Maler, Efficient Robust Monitoring of STL Formula, CAV’13)

- Atomic transducers can be computed in linear time in the size of the input signals
  - A key idea is to exploit efficient streaming algorithm (Lemire’s) computing the max and min over a moving window

- The function \( \varphi(x)[t] \) is computed inductively on the structure of \( \varphi \)
  - linear time complexity in size of \( x \) is preserved
  - exponential worst case complexity in the size of \( \varphi \)
Performance Results

\[ |\varphi| = 1, \text{Time}_\rho \approx 2.34 \times 10^{-6} n_y \]
\[ |\varphi| = 25, \text{Time}_\rho \approx 1.63 \times 10^{-5} n_y \]
\[ |\varphi| = 50, \text{Time}_\rho \approx 2.45 \times 10^{-5} n_y \]
1 Signal Temporal Logics
   - Definition and Examples
   - Template Requirements

2 Requirement Synthesis Framework
   - Algorithm
   - Parameter Synthesis for PSTL
   - STL Falsification

3 Experimental Results
Automatic Transmission System

Engine

- Ti
- Ne
- EngineRPM

ShiftLogic

- up_th
- down_th
- CALC_TH

ThresholdCalculation

- down_th
- run()
- up_th
- throttle

Transmission

- Ne
- Ti
- gear
- Nout
- Tout
- OutputTorque

Vehicle

- VehicleSpeed
- TransmissionRPM

1 throttle

2 RPM

3 gear

2 brake

1 speed
Formulas

- the speed is always below $\pi_1$ and RPM below $\pi_2$

$$\varphi_{sp\_rpm}(\pi_1,\pi_2) := G((\text{speed} < \pi_1) \land (\text{RPM} < \pi_2)).$$

- the vehicle cannot reach 100 mph in $\tau$ seconds with RPM always below $\pi$

$$\varphi_{rpm100}(\tau,\pi) := \neg(F_{[0,\tau]}(\text{speed} > 100) \land G(\text{RPM} < \pi)).$$

- whenever it shift to gear 2, it dwells in gear 2 for at least $\tau$ seconds

$$\varphi_{stay}(\tau) := G\left(\left(\neg\text{gear} \neq 2 \land F_{[0,\varepsilon]}(\text{gear} = 2) \Rightarrow G_{[\varepsilon,\tau]}(\text{gear} = 2)\right)\right).$$
## Results

<table>
<thead>
<tr>
<th>Template</th>
<th>Parameter values</th>
<th>Fals.</th>
<th>Synth.</th>
<th>#Sim.</th>
<th>Sat./x</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi_{sp_rpm}(\pi_1, \pi_2)$</td>
<td>(155 mph, 4858 rpm)</td>
<td>197.2 s</td>
<td>23.1 s</td>
<td>496</td>
<td>0.043 s</td>
</tr>
<tr>
<td>$\varphi_{rpm100}(\pi, \tau)$</td>
<td>(3278.3 rpm, 49.91 s)</td>
<td>267.7 s</td>
<td>10.51 s</td>
<td>709</td>
<td>0.026 s</td>
</tr>
<tr>
<td>$\varphi_{rpm100}(\tau, \pi)$</td>
<td>(4997 rpm, 12.20 s)</td>
<td>147.8 s</td>
<td>5.188 s</td>
<td>411</td>
<td>0.021 s</td>
</tr>
<tr>
<td>$\varphi_{stay}(\pi)$</td>
<td>1.79 s</td>
<td>430.9 s</td>
<td>2.157 s</td>
<td>1015</td>
<td>0.032 s</td>
</tr>
</tbody>
</table>

---

Alexandre Donzé  
Requirement Synthesis for Control Systems  
ExCAPE Summer School 2013
Results on Industrial-scale Model

4000+ Simulink blocks
Look-up tables
nonlinear dynamics

- Found max overshoot with 7000+ simulations in 13 hours
- Attempt to mine maximum observed settling time:
  - stops after 4 iterations
  - gives answer $t_{\text{settle}} = \text{simulation time horizon}$
The above trace found an actual (unexpected) bug in the model.

- The cause was identified as a wrong value in a look-up table.
Requirement Synthesis and Bug Finding

The requirement synthesis framework doubles as an effective bug finding technique:

- Template requirements are implicitly “good” requirement
- The falsification procedure aims at pushing the template boundaries
- The whole framework simultaneously look for a reasonable specification and a violation of it
Summary

- A general framework for requirement synthesis of complex cyber-physical systems

Outlook

- Falsification/optimization of satisfaction functions
- Online monitoring and specification mining
- More elaborate templates mining (beyond parameters)
- How to help designers writing and using temporal logics templates and formulas?

Thanks!
Smoothing Quantitative Satisfaction Functions

Depending on how $\rho$ is defined, the function to optimize can have different profiles.
Smoothing Quantitative Satisfaction Functions

Depending on how \( \rho \) is defined, the function to optimize can have different profiles

\[
\text{(not (ev[0, 5] (gear4w))) and (not ((ev (speed[t]>70)) and (alw[40, inf] (speed[t]<30))))}
\]
Smoothing Quantitative Satisfaction Functions

Depending on how $\rho$ is defined, the function to optimize can have different profiles.

$\neg (\text{ev}_{[0, 5]}(\text{gear4w})) \land \neg ((\text{ev}(\text{speed}[t] > 70)) \land (\text{alw}_{[40, \infty]}(\text{speed}[t] < 30))))$
Dense-Time and Exponential Complexity

A theoretical example with exponential complexity

\[ \varphi_1 = [0,1] (\diamond [0,2] \varphi_0 \land \diamond [2,4] \varphi_0) \]
Dense-Time and Exponential Complexity

A theoretical example with exponential complexity