2016-01-0621

ST-Lib: A Library for Specifying and Classifying Model Behaviors

Author, co-author (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)
Affiliation (Do NOT enter this information. It will be pulled from participant tab in MyTechZone)

Abstract

Test and verification procedures are a vital aspect of the development process for embedded control systems in the automotive domain. Formal requirements can be used in automated procedures to check whether simulation or experimental results adhere to design specifications and even to perform automatic test and formal verification of design models; however, developing formal requirements typically requires significant investment of time and effort for control software designers. We propose Signal Template Library (ST-Lib), a uniform modeling language to encapsulate a number of useful signal patterns in a formal requirement language with the goal of facilitating requirement formulation for automotive control applications. ST-Lib consists of basic modules known as signal templates. Informally, these specify a characteristic signal shape and provide numerical parameters to tune the shape. We propose two use-cases for ST-Lib: (1) allowing designers to classify design behaviors based on user-defined numerical parameters for signal templates, and (2) automatic identification of worst-case values for the signal template parameters for a given closed-loop model of an embedded control system. We show how ST-Lib can be used to improve user productivity by demonstrating its effectiveness on two case studies.

INTRODUCTION

Verification and validation (V&V) for automotive applications is a vital activity for ensuring that embedded control software performs as intended. The success of V&V activities is crucially dependent on the availability of appropriate requirements characterizing correctness and performance of the embedded control system. Often, such requirements define the intended behaviors of the sytem being designed. While requirements can take several forms, machine-checkable, formal requirements are ideal for V&V purposes, as they can be easily incorporated into automated testing procedures. Though control designers often use informal requirements, adoption of formal requirements in the automotive industry is limited due to the challenges involved in developing them. In order to improve the requirement-driven V&V processes, we present a library of signal templates (ST-Lib) intended for use in the automotive industry. Each template in the library is a building block in creating formal requirements, and templates in the library can be combined to generate complex formal requirements specifying correct behaviors.

A concrete usage scenario for a requirement-driven V&V process is the paradigm of model-based development (MBD) [9]. To manage the growing complexity of embedded control system development and the resulting increase in cost associated with test and verification, many organizations are now embracing MBD. MBD provides a way to create, document, and evaluate control software components and environment models in an integrated environment. A key feature of MBD is that it can be used to ensure that control design goals are met at every stage of the development process, as long as appropriate requirements are defined upfront.

Figure 1 illustrates an MBD process known as the Design V. Activities on the left hand side of the V are associated with models of the system, where each subsequent activity represents an enhancement of the model from the previous activity. Processes on the right hand side of the V correspond to experimental (testing) activities. System requirements are provided upfront, at the top left hand side of the V, and are used to check whether the models produce behaviors that are acceptable.

V&V activities can be performed on the system models to check whether the design satisfies requirements early in the design process (before hardware is developed) and have the benefit of allowing necessary improvements to be made early in the design process. Performing improvements early in the design process reduces development cost as compared to the cost of rework at later stages. To use V&V methodologies in an MBD design process, it is essential to have formal requirements. Formal re-

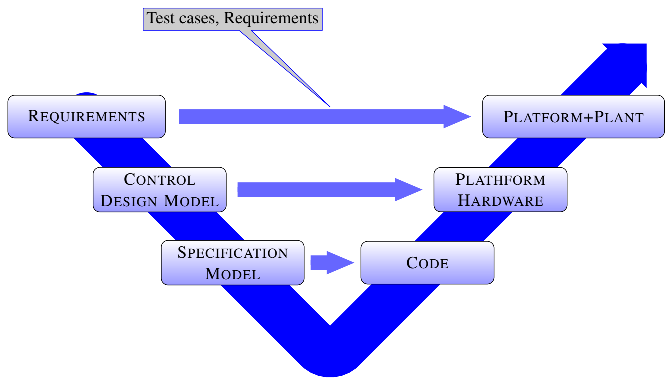

Figure 1: The model-based development design V.

quirements offer two key benefits:

- Provide a clear definition of correct behavior, which can be used by ad hoc model testing techniques to evaluate system performance, and
- Are essential inputs to automated model testing and verification tools.

Given a collection of formal requirements, there are several methods for performing V&V on embedded control system models. The simplest technique is to perform *simulations* of the models in an ad hoc fashion. Simulations are numerical estimates of the behaviors of the models based on a collection of operating parameters. Simulations can be used to check whether system models conform to requirements for specific operating parameters. Using an ad hoc approach, parameter values are selected manually based on designer insight. This approach has the benefit that the designer is often aware of system parameters that correspond to critical system behaviors. The downside is that it is time-consuming for control engineers to identify system parameters that correspond to significant behaviors.

To address the shortcomings of ad hoc testing, automated model testing tools can be employed. Tools such as S-TaLiRo [1, 5] and Breach [2, 3] automate the search for significant system behaviors based on a Simulink® model of the system and a formal requirement. These tools use the formal requirement to formulate a function that maps system behaviors to real values, where the values quantify the degree to which the requirement is satisfied (or is not satisfied), with positive values corresponding to satisfactory behaviors and negative values corresponding to unsatisfactory behaviors. This quantity indicates *how well* the system behavior satisfies a given performance requirement, and can be used as the cost function to a suitably framed optimization problem. Then, a global optimization tool is used to select system parameters to minimize the cost function. If the optimizer selects parameters that correspond to a negative cost valuation, then the parameters correspond to behaviors that do not satisfy the requirement, and the parameters are returned to the user for inspection so that the design can be reexamined. These tools have the benefit that they provide an automatic way to test system models against requirements and thus alleviate the burden on control engineers to some extent. The downside is that these tools are best-effort tools, that is, it is not guaranteed that the tools will identify behaviors that do not satisfy requirements (if they exist).

Machine-checkable formal requirements are not commonplace in the automotive industry. There are several reasons for this; first, traditional metrics used to evaluate control system performance, such as overshoot and settling time, are inherently difficult to formulate in the languages typically used to create formal requirements. This is because formal requirement languages are traditionally used to specify behaviors of finite state systems, rather than the continuous behaviors considered by control engineers. The second reason is that formal requirements take time to develop, and automotive design cycles are usually tight, meaning that the time available for activities that lie outside of the traditional automotive development processes such as requirements development is quite limited. The final issue is that of cross-domain knowledge: automotive engineers are not typically familiar with the languages used to create formal requirements. Such languages and associated logics are typically better known to computer scientists, while automotive engineers are typically trained as mechanical and electrical engineers.

In an ideal process, formal requirements are identified during the initial development phase and do not evolve during the development process. In practice this is not realistic; the challenge is that many automotive control systems exhibit complex behaviors that are difficult to anticipate at design time. Nevertheless, formal requirements provide value even for cases where behaviors are not anticipated; once the behaviors have been identified, they can be captured as new requirements, included in the set of system requirements, and used during the next development cycle to check whether the system behaves correctly.

In what follows, we give a brief background on temporal logic, the underlying formalism supporting the proposed signal template library.

Background

A temporal logic is a language in which formal specifications can be written for embedded control systems, such as automotive control systems. In this section, we provide a brief background on temporal logics, and we give an overview of Signal Temporal Logic (STL), which is the logic that we propose to use to create requirements for automotive applications.

In the late 70s, Amir Pnueli [10] introduced temporal logic to computer science to reason formally about the temporal behaviors of *reactive* systems. The use of temporal logic was originally to reason about input-output systems with Boolean, discrete-time signals, and heavily focused on verification, specification, and synthesis of concurrent systems. A number of temporal logics were introduced to reason about real-time signals, such as Metric Temporal Logic (MTL), and, more recently, Signal Temporal Logic (STL) [7]. MTL typically allowed reasoning over Boolean signals but over dense-time domains while STL [8] was proposed in the context of analog and mixed-signal circuits as a specification language for constraints on real-valued signals.

An STL formula is composed of a set of logical connectives, like $\wedge$ and $\vee$, temporal operators, and atomic subformulae. An atomic formula expresses constraints on signals and, without loss of generality, can be reduced to a form $f(\mathbf{x}) \bowtie 0$, where $\mathbf{x}$ represents the name of a signal (a function from $\mathbb{R}^{\geq 0}$ to $\mathbb{R}^n$), $\bowtie \in \{<, \leq, >, \geq, =\}$, and f is an arbitrary function from $\mathbb{R}^n$ to $\mathbb{R}$. A temporal formula is formed using temporal operators "always" (denoted as $\Box$), "eventually" (denoted as $\Diamond$) and "until" (denoted as $\mathbf{U}$). Each temporal operator is indexed by an interval I over $\mathbb{R}^{\geq 0} \cup \{\infty\}$; this can be an open interval (a, b), a closed interval $[a, b]$, open-closed $(a, b]$ or closed-open $[a, b)$.

For example, consider the following STL formula:

$$\Box_{[0,100]} \left(\texttt{boost\_pressure} < 250\right). \tag{1}$$

The requirement in STL formula (1) specifies that for all times t in $[0, 100]$, the physical quantity $\texttt{boost\_pressure}$ is less than 250 kPA. This STL requirement can be used to characterize the maximum value allowed for a given signal. For another STL requirement example, consider the following:

$$\Box_{[0,100]} \Big((\texttt{gear} = 1 \wedge \Diamond_{[0,\epsilon]} \ \texttt{gear} = 2) \implies \Box_{[\epsilon,\tau+\epsilon]} \left(\texttt{gear} = 2\right) \Big). \tag{2}$$

The requirement (2) specifies that if the $\texttt{gear}$ changes from 1 to 2 within a small time (ϵ), then it stays at 2 for at least τ seconds. Such a requirement can be used to specify the minimum dwell-time for a transmission gear.

In the following section, we explain how to use STL formulae to describe common control signal behaviors using ST-Lib.

TAXONOMY OF COMMON CONTROL BEHAVIORS

There is often a mismatch between the languages used to create formal requirements, such as STL, and the kinds of qualitative behaviors considered by control engineers, which exacerbates the challenges that control engineers face in creating formal requirements. Below, we explain why this is, and we describe behaviors that automotive control engineers expect that are difficult to capture in languages such as temporal logic.

System performance measures commonly used by control engineers typically arise out of test scenarios. For example, a step response requirement implies that the output signal of interest should adhere to some performance requirement (such as the overshoot or settling time) in response to the step input. Control engineers usually expect the step response performance to hold in response to any edge in the input signal, such as from a single pulse or a train of pulses. While using either a step or a pulse signal to excite a system is natural to a control engineer, from a formal requirements perspective, this corresponds to a multiplicity in acceptable behaviors, and hence corresponds to multiple individual formal requirements (or a single but a complex requirement). Envisaging all possible scenarios that may correspond to a step response for a control engineer and capturing them as a uniform requirement requires considerably more work than an informal statement about the desired step response.

Even when control engineers have a clear idea of the expected input/output behavior (or behavior they want to avoid), the behavior may be difficult to express using STL. For example, consider the case where the control engineer wants to avoid a *ringing* behavior. Ringing may be easy to describe informally and may be easy to identify manually when it is exhibited, but defining it formally may require precise articulation of different "features" that comprise a ringing behavior. If the ringing signal has a known average value, it may be easier to define, but in some cases such an average value is not known a priori, or simply does not exist; for example, consider the case when the ringing occurs on the slope of a ramp function.

Lastly, control engineers expect qualitative behaviors from the system that they themselves may have difficulty describing, even informally. For example consider the case where the control engineer expects a system to exhibit an oscillating (sinusoidal) behavior (or something similar to it) but instead the engineer observes a pulse train. It is not obvious how to differentiate the expected signal from the pulse train. One could specify, for example, how sharp the rising and falling parts of the signal are (e.g., the magnitude of the derivative), but it is not clear what derivative magnitude limit separates the sinusoidal behavior from the pulse train.

In the following, we describe behavior classes that the automotive control engineer typically wants to avoid; formal system requirements will typically dictate that these behaviors do *not* occur. The following is a partial list of common undesirable behaviors. We note that some of the behaviors described below can be challenging for control engineers to articulate using a formal requirement language such as STL.

- **Ringing:** Some practical systems are designed to ring (or oscillate) indefinitely. For example, the air charge in the combustion cylinder of an internal combustion engine will alternate between charged and discharged over every engine cycle. But ringing in most systems is considered undesirable. Ringing behavior defined as a signal oscillating around a given average value can be easily captured in a temporal logic like STL, but more subtle ringing that modulates, some qualitatively different signal like a ramp may be difficult to capture for a non-expert.

- **Spikes and Glitches:** Sharp changes in a signal value can lead to unacceptable system performance and are often considered to be undesirable. As with ringing, using a temporal logic like STL to capture the case where a spike modulates some signal can be challenging.

- **Excessive Overshoot or Undershoot:** Control systems are often expected to exhibit some overshoot (or undershoot), but these quantities are usually expected to respect some upper (or lower) limit. Since the related requirement involves the corresponding input signals that trigger the step response behavior, many relational behaviors are assumed to be acceptable.

- **Slow Response Time (settling, rising, or falling):** As in the case of overshoot and undershoot, settling, rising, and falling times are expected behaviors from a control system in response to a step input.
- **Undesirable Timed Relation Behaviors:** Expressing timed relationships between signals is elegant in a formalism like STL. For example, consider two signals a and b and suppose the designer wants to forbid b from switching from low to high within x seconds of a switching from low to high. This kind of behavior can be easily captured by STL. A requirement template to express such timed relations can be useful for control engineers.
- **Steady State or Tracking Error:** Steady state errors are usually undesirable, but they are often tolerated in moderation. As with the overshoot and response time requirements, steady state error requires at least two signals to specify. While this can be accomplished using STL, a requirement template with parameters that the control engineer is more familiar with can be quite useful.

In the sequel, we present a library of signal templates, ST-Lib, along with a framework that will allow the automotive control engineer to easily construct formal requirements related to each of the above behavior types.

ST-LIB

We present ST-Lib, a library of formal requirements for behavior classes that are often the focus of testing processes performed by automotive control engineers. We provide a description and use case for each requierment-class, the parameters used to define each requirement, and examples illustrating the difference between allowed and disallowed signals for each class. The requirements are provided in STL format and can be used in testing and verification environments such as the Breach tool.

Assumptions

The following assumptions are made regarding the signals used to define the STL formulae in the ST-Lib. We assume that the requirements are based on some signal $\mathbf{x}$, which is defined at a finite set of time instants $t_0, \ldots, t_N \in \mathbb{R}_{\geq 0}$. Here $t_0 = 0$, and $t_N = T$ is called the time horizon. Also, we assume that $\min_{i \in [0,N-1]} t_i - t_{i+1} > \epsilon$, which provides a lower bound for the time-step of the signals, ϵ. Signals may contain measurement noise (or approximation error), which has the potential to trigger false positive results, but also can be used to automatically detect undesirable levels of certain types of noise. We use $\mathbf{x}_{\mathit{diff}}$ to denote the discrete-time derivative of $\mathbf{x}$, where $\mathbf{x}_{\mathit{diff}}(t_i) = \frac{x(t_{i+1}) - x(t_1)}{t_{i+1} - t_i}$ for all $i \in \{1, \ldots, N-1\}$, and $\mathbf{x}_{\mathit{diff}}(t_N)$ is not defined.

In the automotive domain, the term *requirements* is often used to capture high-level stipulations of the system behavior, while in the formal verification community in computer sciences, the term *specification* is used for the same purpose [11]. In an automotive context, a specification model typically implies an executable model of the embedded control software. In the rest of this section, we use the words requirements and specification interchangeably, as in computer science.

Also, to ease the exposition, we provide requirements that describe behaviors that are *not* acceptable. To create a requirement that allows any signal except a selected unacceptable behavior, say behavior B_1, one would use the negation of the given formula, which is obtained by placing a *not* ($\neg$) operator in front of the indicated STL formula, as in $\neg B_1$. To create a requirement that allows any behavior except those indicated by multiple formulas, for example B_1 and B_2, one would use a logical *and* ($\wedge$) operator between the negated formulae, as in $(\neg B_1) \wedge (\neg B_2)$.

Ringing

Formally defining ringing behavior can be challenging for two reasons. First, it is often the case that the ringing behavior is defined around some mean value, which may not be defined at the time the formal requirement is defined. Second, there may be some ringing behaviors that are acceptable, while others are not, as in the case where ringing that occurs at the magnitude of signal noise may be allowable but ringing of larger magnitude may not.

The following STL formula addresses the case where the mean signal value is not known by defining predicates on the difference between the signal and its time-shifted version as follows:

$$\mathsf{rise} \overset{\text{def}}{=} \mathbf{x}(t+d) - \mathbf{x}(t) > a, \tag{3}$$

$$\mathsf{fall} \overset{\text{def}}{=} \mathbf{x}(t) - \mathbf{x}(t+d) > a. \tag{4}$$

The time-shift d, and the minimum amplitude a are both parameters controlled by the user. The following formula is satisfied by signals that have one peak followed by one valley at time 0:

$$\mathsf{ring\_once\_0} \overset{\text{def}}{=} \mathsf{rise} \wedge \left(\Diamond_{[0,2d]}\mathsf{fall}\right). \tag{5}$$

The following formula accepts signals that have at least one instance of ringing at some time between time 0 and the horizon T:

$$\mathsf{ring\_once} \overset{\text{def}}{=} \Diamond_{[0,T]}\mathsf{ring\_once\_0}. \tag{6}$$

The next formula represents signals that have at least two consecutive instances of ringing starting at some time in $[0, T]$:

$$\mathsf{ring\_twice} \overset{\text{def}}{=} \Diamond_{[0,T]}\left(\mathsf{ring\_once\_0} \wedge \Diamond_{[2d,4d]}\mathsf{ring\_once\_0}\right). \tag{7}$$

The user specifies d and a as parameters. The parameter a controls the minimum amplitude of the ringing, whereas d is related to the largest period (i.e., the smallest frequency) of the ringing behavior. We note that d should be selected to be no larger than approximately 25% of the smallest period that the designer intends to capture, to avoid effects of *undersampling* the signal with (3) and (4).

Figure 2 provides examples of signals that satisfy (unacceptable) and do not satisfy (acceptable) a given ringing specification, given parameters $d = 0.0625$ and $a = 0.25$. Note that the examples of ringing modulate an exponential signal; this is to illustrate that the formula is agnostic to a given reference value, as opposed to a naïve formulation that only considers the number of times a signal crosses some mean signal value.

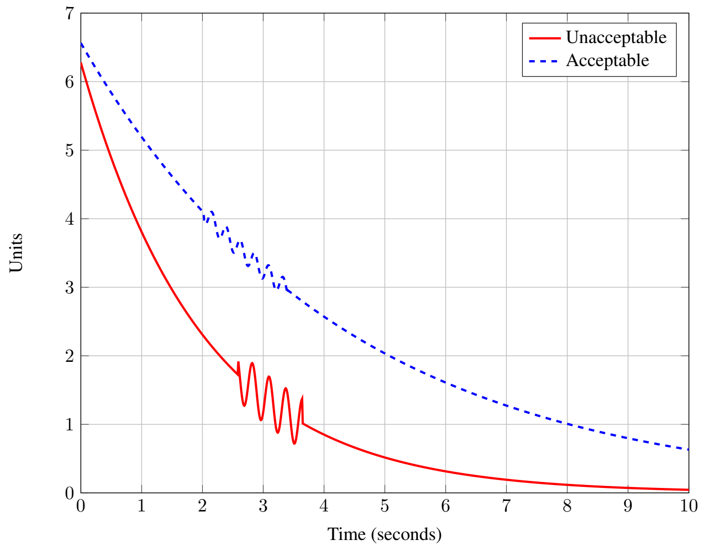

Figure 2: Example of acceptable and unacceptable signals, as defined by the STL specification for ringing behavior.

Spike

A spike can be informally defined as a large, brief jump in a signal value. The spike specification allows the designer to formally define unacceptable sharp transient system behaviors. Many signals will be expected to exhibit some brief jumps in value of some magnitude even though larger jumps in value will be unacceptable. The following STL formula is a template to define unacceptable spike behaviors.

$$\Diamond_{[0,T]}\left((\mathbf{x}_{diff} > m) \wedge \Diamond_{[0,w]}(\mathbf{x}_{diff} < -m)\right) \tag{8}$$

Here, m, w are positive real numbers, with $w > \epsilon$. The parameter w is related to the spike width, and $m \cdot w$ is proportional to the spike amplitude.

There are several specializations and generalizations of the above formula that could be made to restrict or loosen the kind of signals that are classified as spikes. For example, modest changes to the formula could specify that only one spike in a given w interval is acceptable but more than one is not.

Figures 3 and 4 illustrate acceptable and unacceptable behaviors as defined by the spike specification for a given m and w. For Figure 3, the parameter specifications are $m = 10.0$ and $w = 0.5$. For Figure 4, the parameter specifications are $m = 8.0$ and $w = 0.5$. Note that the distinguishing feature between the acceptable and unacceptable signals is the magnitude of the spikes.

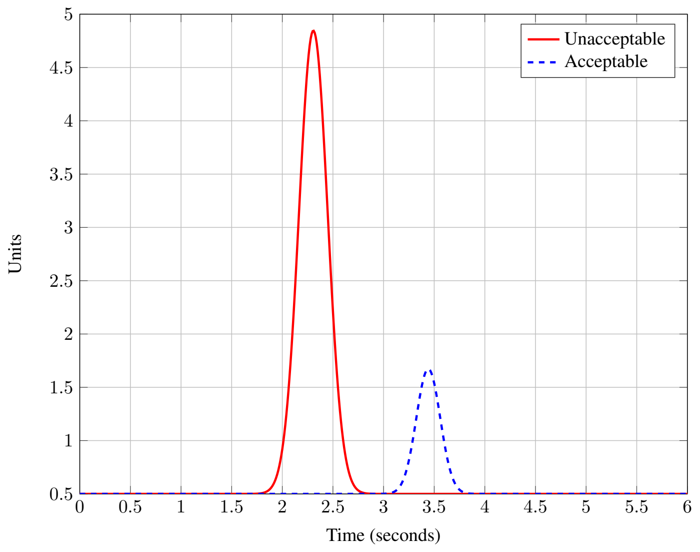

Figure 3: Example of acceptable and unacceptable signals, as defined by the spike STL specification.

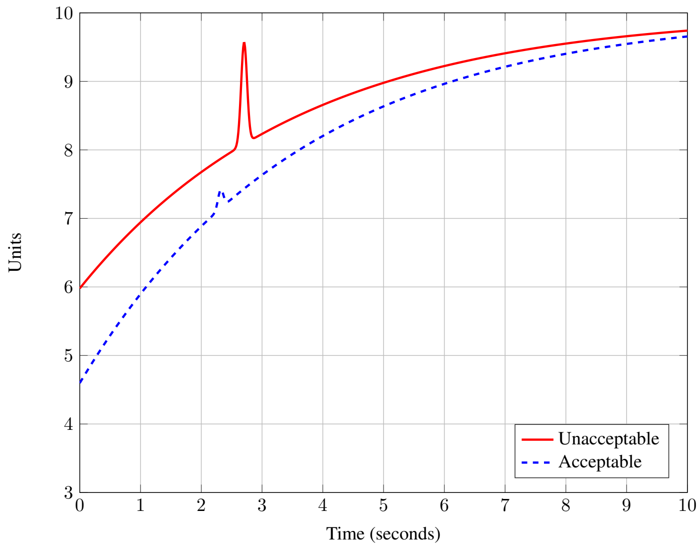

Figure 4: Advanced example of acceptable and unacceptable signals, as defined by the spike STL specification.

Overshoot

Overshoot and undershoot are quantities that define how much a signal goes beyond an expected steady state value in response to a step input and is often used as an indicator of control system performance. The overshoot specification provides parameters to allow the designer to differentiate acceptable from unacceptable signal excursions beyond steady state values in response to steps in the input.

As with the ringing specification, defining a formal specification capturing unacceptable levels of overshoot is challenging because the expected steady state value of the signal may be unknown when the specification is created. The following STL formula assumes that the overshoot is defined with respect to a known reference signal (reference tracking).

We first define a predicate characterizing a step of amplitude at least r units in a given signal at some time t.

$$\mathsf{step}(\mathbf{y}, r) \overset{\text{def}}{=} \mathbf{y}(t+\epsilon) - \mathbf{y}(t) > r \qquad (9)$$

In specifying overshoot, we assume that the system is trying to track such a step in a given reference signal. Given a reference signal $\mathbf{x}_{\mathrm{ref}}$, and a signal $\mathbf{x}$, the following STL formula characterizes signals $\mathbf{x}$ that overshoot the reference signal $\mathbf{x}_{\mathrm{ref}}$ by a quantity greater than the user-specified threshold c.

$$\Diamond_{[0,T]} \left(\mathsf{step}(\mathbf{x}_{\mathrm{ref}}, r) \wedge \Diamond \left(\mathbf{x} - \mathbf{x}_{\mathrm{ref}} > c\right)\right) \qquad (10)$$

Figure 5 provides an example of an acceptable and an unacceptable signal as defined by the overshoot specification, using parameters $r = 5.0$ and $c = 2.5$. The unacceptable signal has a significantly higher peak amplitude than the unacceptable signal.

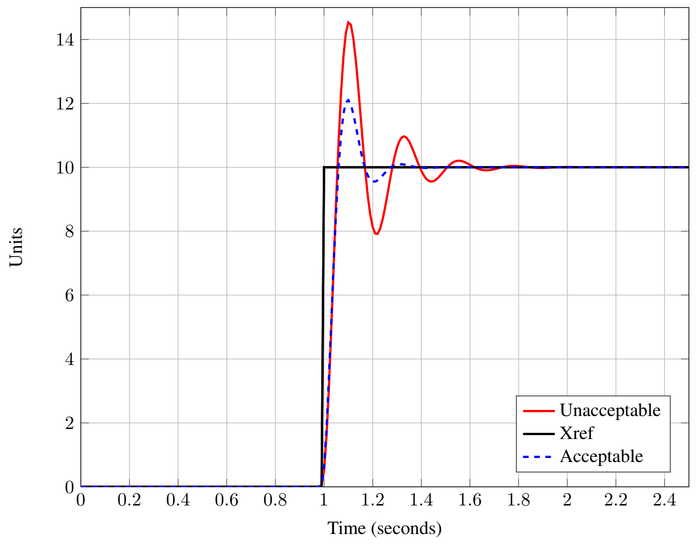

Figure 5: Example of acceptable and unacceptable signals, as defined by the overshoot STL specification.

Settling Time

Settling time is the time it takes a signal to reach and remain within a band around a steady state value in response to a step input. Settling time is a common measure of control system performance and is one common way to define worst case timing performance for a control system.

Settling time is usually measured in response to a step input in a given reference signal, and quantifies how soon the signal can settle to some region around the reference signal. The following STL formula characterizes signals $\mathbf{x}$ exhibiting settling time greater than s for a region of size β around the reference signal.

$$\Diamond_{[0,T]} \left(\mathsf{step}(\mathbf{x}_{\mathrm{ref}}, r) \wedge \Diamond_{[s,T]} \left(|\mathbf{x} - \mathbf{x}_{\mathrm{ref}}| > \beta \mathbf{x}_{\mathrm{ref}}\right)\right) \qquad (11)$$

Here, β is some small fraction (typically between 1 and 5%) of the steady state value of the reference signal $\mathbf{x}_{\mathrm{ref}}$. Figure 6 provides examples of acceptable versus unacceptable settling time as defined by (11), based on parameter values $r = 5.0$, $s = 5.0$, and $\beta = 0.1$. As expected, the signal exhibiting unacceptable settling time takes longer to approach the reference value than the acceptable signal.

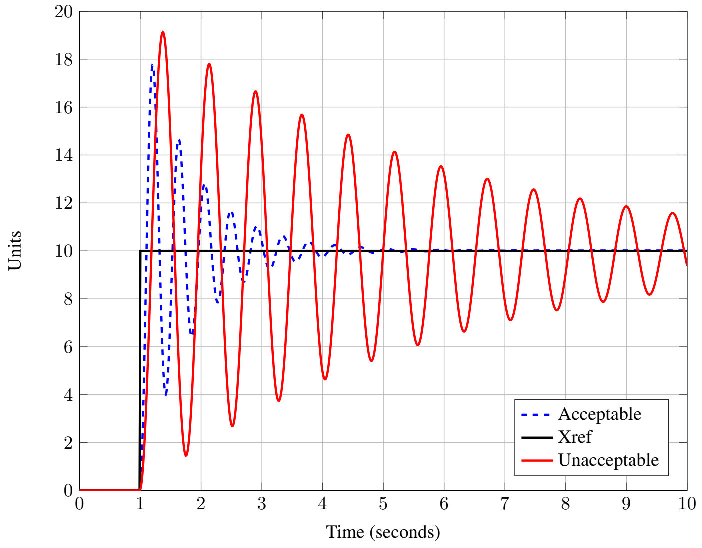

Figure 6: Example of acceptable and unacceptable signals, as defined by the settling time STL specification.

Rise Time

Rise and fall times are defined informally as the amount of time it takes for a signal to change from an initial value to another value some distance from an expected steady state value in response to a step input. As with settling time, rise time is a common measure of control system performance and is another way to define worst case timing performance for a control system.

The following STL formula assumes a reference signal, $\mathbf{x}_{\mathrm{ref}}$, along with parameters, T, ζ, and μ, that allow the designer to define acceptable rise time system performance. Here, we assume that the steady state value of the signal $\mathbf{x}_1$ is known (denoted $\mathbf{x}_1^*$), and rise time measures the amount of time required to reach a user-specifed fraction μ of $\mathbf{x}_1^*$ after a given step event of amplitude at least r units in the stimulating signal $\mathbf{x}_{\mathrm{ref}}$. The following STL formula characterizes signals that exceed the rise time requirement.

$$\Diamond_{[0,T]} \left(\mathsf{step}(\mathbf{x}_{\mathrm{ref}}, r) \wedge \Box_{[0,\zeta]} \left(\mathbf{x}_1 < \mu \cdot \mathbf{x}_1^*\right)\right) \qquad (12)$$

Figure 7 provides examples of acceptable versus unacceptable rise time performance, as defined by (12), using parameters $r = 5.0$, $\zeta = 1.0$, $\mu = 0.8$, and $\mathbf{x}_1^* = 10.0$.

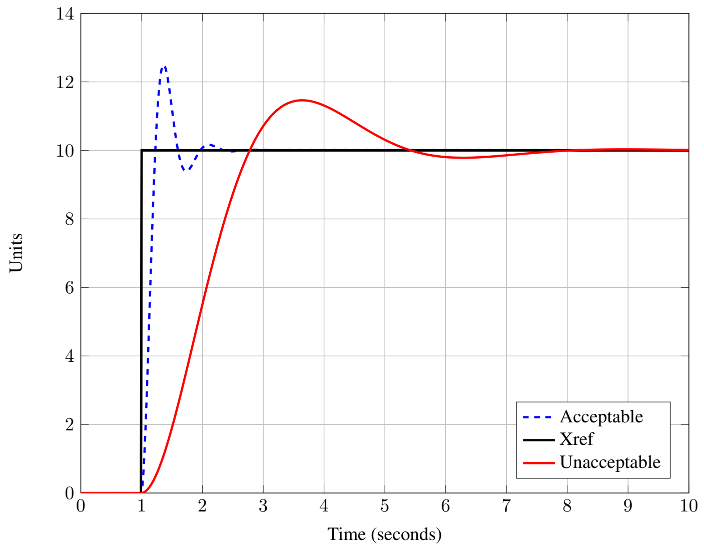

Figure 7: Example of acceptable and unacceptable signals, as defined by the rise time STL specification.

Timed Relationships

The timed relationship signal behavior class characterizes the temporal difference in behaviors between two or more signals. This class of signal behaviors is useful to characterize acceptable performance when one signal is expected to exhibit a designated behavior at some specified time after another signal exhibits some other specified behavior. The ability to specify timed relationships between signals is vital whenever signals are expected to exhibit coordinated behavior.

Timed behaviors are easily specified using temporal logic, as this is the type of behavior that these languages were originally designed to capture. The following STL formulae provide parameters that allow the designer to specify acceptable timed relational behaviors between two signals. The formulae are based on two Boolean signals $\mathbf{p}_1$ and $\mathbf{p}_2$. In the list below, for each item, a desired time relation is stated, and then the formula characterizing signals *not* satisfying the relation is given.

- Desired relation: $\mathbf{p}_2$ becomes high within τ seconds after $\mathbf{p}_1$ goes high. Signals not satisfying this relation are characterized by:

$$\Diamond_{[0,T]}\left(\mathbf{p}_1 \wedge \Box_{[0,\tau]} \neg \mathbf{p}_2\right). \tag{13}$$

- Desired relation: If $\mathbf{p}_1$ stays high for τ seconds, $\mathbf{p}_2$ becomes high immediately. Signals not satisfying this relation are characterized by:

$$\Diamond_{[0,T]}\left(\mathbf{p}_1 \mathbf{U}_{[\tau,\tau]} \neg \mathbf{p}_2\right). \tag{14}$$

Figure 8 illustrates examples of acceptable versus unacceptable timed behaviors as defined by (13), based on the parameter value $\tau = 3.0$. For the unacceptable example, $\mathbf{p}_2$ takes longer than 3.0 seconds to react to $\mathbf{p}_1$. This is as compared to the acceptable case, where $\mathbf{p}_2$ takes much less than 3.0 seconds to react to $\mathbf{p}_1$.

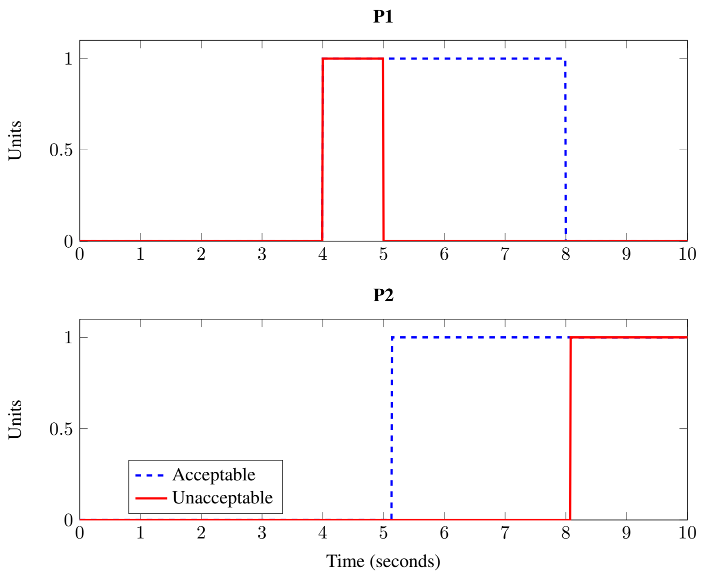

Figure 8: Example of acceptable and unacceptable signals, as defined by the timed relationship STL specification.

Steady State Error

Steady state error is informally defined as the difference between the final value of a signal and the desired final value. Steady state error is a common way to characterize the accuracy of a control system. The steady state error specification provides the designer with a formal way to define acceptable versus unacceptable accuracy after transient behaviors have dissipated.

For the steady state error specification, we assume that a reference is available to compare against a given output signal. We provide formulae for two different classes of reference signals: step inputs and pulse trains. For each case, we assume we have a reference signal $\mathbf{x}_{\text{ref}}$ and the steady state error of the signal $\mathbf{x}$ is measured with respect to $\mathbf{x}_{\text{ref}}$.

First, consider the case where the input reference signal is a step function. Here, we can assume that $\mathbf{x}$ is in steady-state at the time horizon T. The STL formula characterizing a steady state error of greater than a is

$$\Diamond_{[T,T]}|\mathbf{x} - \mathbf{x}_{\text{ref}}| > a. \tag{15}$$

Next, consider the case where the input reference signal is similar to a pulse train. Assume the reference signal is a series of steps of varying magnitudes. We assume that if both $\mathbf{x}_{\text{ref}}$ and $\mathbf{x}$ are not changing or are changing slowly for at least k steps, then $\mathbf{x}$ has reached steady state for the corresponding step. The corresponding STL formula is

$$\Diamond_{[0,T]} \left((|\mathbf{x}_{diff}| < \beta \;\wedge\; |\mathbf{x}_{\text{ref}\_diff}| < \beta) \, \mathbf{U}_{[k\epsilon,k\epsilon]} |\mathbf{x} - \mathbf{x}_{\text{ref}}| > a \right). \tag{16}$$

Here, $\mathbf{x}_{\text{ref}\_diff}$ is the discrete-time derivative of $\mathbf{x}_{\text{ref}}$, k is a user-defined parameter such that $k\epsilon$ is smaller than the duration of any two consecutive steps in the reference signal, and β is some small positive number. Both k and β can be chosen by the user. The constant a is the magnitude of the tolerated steady-state error.

Figure 9 provides examples of acceptable versus unacceptable steady state error behaviors as defined by (15), based on the parameter value $a = 0.5$. In the figure, the steady state value for $\mathbf{x}_{\text{ref}}$ is 10.0; signals satisfying the specification will therefore fall within 9.5 and 10.5 at $t = 10.0$ seconds. Notice that the steady state value for the unacceptable trace is slightly larger than 10.5, while the steady state value for the acceptable trace is between 9.5 and 10.5.

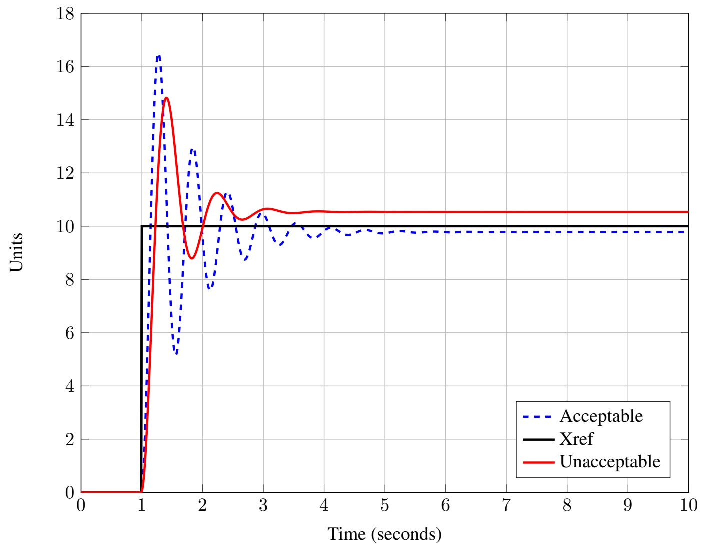

Figure 9: Example of acceptable and unacceptable signals, as defined by the steady state error STL specification.

CASE STUDIES

In this section, we describe an example model of an automotive powertrain control system and demonstrate how specification templates from ST-Lib can be used in conjunction with a design analysis tool to automatically test the model against specifications.

Example model: abstract prototype automotive fuel control system

Consider the automotive powertrain control (PTC) system presented in [6]. The model is a simplified representation of the fuel control system (FCS) in a gasoline engine; the purpose of the system is to accurately regulate the ratio of air-to-fuel in the engine. The model is implemented in Simulink® and contains representations of the air-fuel controller and the air-fuel dynamics in the engine. The engine is represented by a simplified mean-value model and contains the throttle, intake manifold air, and fuel dynamics. The controller has three modes of operation: STARTUP, NORMAL, POWER, and FAULT. In the NORMAL mode of operation, the controller uses a simple proportional plus integral (PI) control scheme.

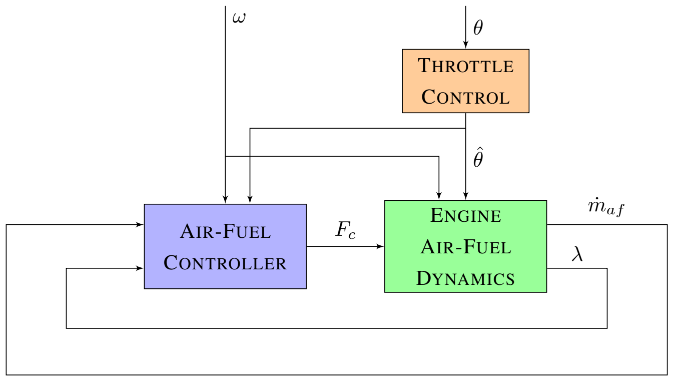

Figure 10: Block diagram of the fuel control system.

Inputs to the model are the throttle position command in degrees, θ, and the engine speed in RPM, ω. The output of the plant model to the controller is the air-fuel ratio, λ, and the intake manifold inlet mass airflow rate in grams per second, $\dot{m}_{af}$. The controller output to the plant is the fuel rate command in grams per second, F_c. The main output signal is the air-fuel ratio, λ. Figure 10 provides a block diagram of the fuel control system. For a detailed description of the model, see [6].

The FCS model contains multiplicative error terms that model calibration and other tolerances in the corresponding components. Multiplicative error terms are present in the following components: the air-fuel ratio sensor, the fuel injection actuator, and the mass air flow sensor. In the experiments that follow, the multiplicative error terms associated with the fuel injection actuator and the mass air flow sensor are set to 1.0 (0.0% error), and the multiplicative error parameter for the air-fuel ratio sensor error r_{AF} is assumed to be between 0.98 to 1.02 ($\pm 2.0\%$).

ST-Lib application: checking overshoot and settling time

One application of ST-Lib is simulation-based requirement falsification. This technique has been successfully applied to automotive applications [4] and provides valuable insights to help design engineers improve control designs; however, this technique requires users to specify requirements in a formal logic language. The ST-Lib provides an easy way for engineers to write correct and accurate formal requirements. In the following, we describe how to use the ST-Lib to create formal requirements for the FCS model and then describe how to use the resulting requirements in the Breach tool [2] to automatically check whether the design satisfies the property.

The Breach tool provides a framework to perform automatic testing of Simulink models, based on given requirements specified in STL format. To analyze the FCS model, we create input scripts for Breach, which contain the formal specifica-

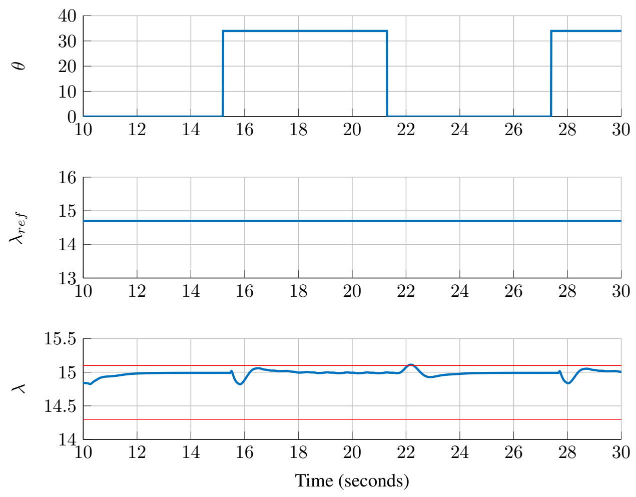

Figure 11: The counterexample trace found by Breach for the overshoot requirement.

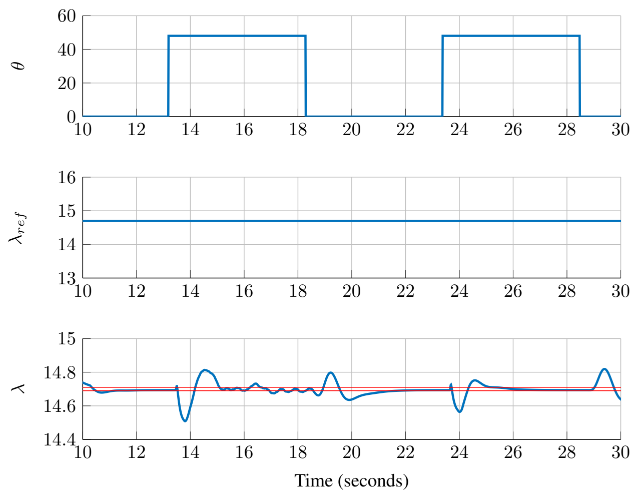

Figure 12: The counterexample trace found by Breach for the settling time requirement.

tions, along with other analysis parameters, such as input signal bounds. Breach then uses an automated procedure to select input sequences. The input sequences are used to produce simulations using a numerical integration package built into Simulink. The simulation traces are evaluated to determine whether they satisfy the given STL formula; if any traces do not satisfy the formula, the trace is reported back to the user.

To perform the analysis for the FCS model, we select templates and example parameters for two behaviors types: overshoot and settling time. For the overshoot case, we use modified versions of Equations (9) and (10) and select parameters $r = 5$, $T = 30$, $\epsilon = 0.02$, and $c = 0.04 * \lambda_{ref}$. The resulting STL formula is as follows.

$$\Diamond_{[10,30]} \left(\mathsf{step}(\theta, 5) \wedge \Diamond \left(\lambda - \lambda_{ref} > 0.04 * \lambda_{ref}\right)\right) \qquad (17)$$

Breach can falsify the requirement in about 70 seconds on a typical desktop PC. Figure 11 shows the resulting output trace from Breach, which demonstrates a case where the overshoot is greater than 4% of the reference λ_{ref}; the top red line in the fourth sub-figure shows the threshold $0.04 * \lambda_{ref}$ for the requirement. Note that, though the behavior of interest occurs just after the rising edge in the pedal input that occurs near $t = 23.5$ seconds, the sequence of rising and falling edges in the input that occur between 10.0 and 27.5 seconds contribute to the behavior; before $t = 10.0$ seconds, the controller is in the STARTUP mode and does not react to inputs. In a typical development context, the counterexample trace shown in Figure 11 could be used by engineers to improve the control design.

Next, we formulate the settling time specification using Eq. (11) and parameters $r = 5$, $s = 3$, $T = 30$, $\epsilon = 0.02$ and $\beta = 1\%$. The resulting STL formula used for the Breach analysis is as follows:

$$\Diamond_{[10,30]} \left(\mathsf{step}(\theta, 5) \wedge \Diamond_{[3,5]} \left(|\lambda - \lambda_{ref}| > 0.01 * \lambda_{ref}\right)\right). \qquad (18)$$

Breach can successfully falsify the requirement within approximately 10 seconds using a standard desktop PC. Figure 12 illustrates the resulting output trace from Breach, which demonstrates an instance where the settling time is greater than the specified 3.0 seconds; the red band shows the range that the signal should remain in after 3.0 seconds. The counterexample found demonstrates a situation where the system cannot settle before a new falling edge of the pedal signal occurs. As with Figure 11, the counterexample trace shown in Figure 12 could be used by engineers to improve the control design.

CONCLUSIONS

We introduced ST-Lib, a new framework for constructing formal requirements for automotive control applications. The new framework produces requirements in a temporal logic language called signal temporal logic (STL), which can be used by automated testing tools to evaluate design models. Requirements developed using ST-Lib can be integrated with a model-based development (MBD) process to decrease the cost of V&V activities for automotive applications. We demonstrated how ST-Lib can be used to automatically test design models using a benchmark powertrain control model. The examples we present use Breach, an advanced simulation-based testing tool, to automatically evaluate the benchmark model.

One challenge is that it may be difficult to select appropriate parameters for the specifications created from ST-Lib. Future work will investigate the use of automated techniques for inferring appropriate parameters for the template specifications. Also, in this paper, we consider time domain requirements on signals. Future work will also enhance ST-Lib to express frequency domain requirements. Lastly, we plan to extend ST-Lib to handle performance requirements that are difficult to articulate, such as requirements on the *smoothness* of a signal.

ACKNOLEDGEMENTS

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

REFERENCES

[1] Yashwanth S. R. Annapureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257, 2011.

[2] Alexandre Donzé. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Proc. of Computer Aided Verification*, pages 167–170, 2010.

[3] Alexandre Donzé. Breach toolbox. `http://www.eecs.berkeley.edu/~donze/breach_page.html`, 2015.

[4] Georgios E. Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of Automotive Control Applications using S-TaLiRo. In *Proc. of the American Control Conference*, 2012.

[5] Sriram Sankaranarayanan Georgios Fainekos. S-taliro toolbox. `https://sites.google.com/a/asu.edu/s-taliro/s-taliro`, 2015.

[6] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 253–262. ACM, 2014.

[7] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.

[8] Oded Maler and Dejan Nickovic. Monitoring Temporal Properties of Continuous Signals. In *Proc. of Formal Modeling and Analysis of Timed Systems/ Formal Techniques in Real-Time and Fault Tolerant Systems*, pages 152–166, 2004.

[9] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. CRC Press, 2009.

[10] A. Pnueli. The Temporal Logic of Programs. In *Proc. of Foundations of Computer Science*, pages 46–57, 1977.

[11] Jeannette M Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–22, 1990.