Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans

Citation for the original published paper (version of record):
Lidén Eddeland, J., Claessen, K., Smallbone, N. et al (2020). Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(12): 5247-5260.
http://dx.doi.org/10.1109/TCAD.2020.2966480

Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans

Johan Lidén Eddeland, Koen Claessen, Nicholas Smallbone, Zahra Ramezani, Sajed Miremadi, and Knut Åkesson

***Abstract*—Cyber-Physical Systems (CPSs) are systems with both physical and software components, for example cars and industrial robots. Since these systems exhibit both discrete and continuous dynamics, they are complex and it is thus difficult to verify that they behave as expected. Falsification of temporal logic properties is an approach to find counterexamples to CPSs by means of simulation. In this paper, we propose two additions to enhance the capability of falsification and make it more viable in a large-scale industrial setting. The first addition is a framework for transforming specifications from a signal-based model into Signal Temporal Logic. The second addition is the use of Valued Booleans and an additive robust semantics in the falsification process. We evaluate the performance of the additive robust semantics on a set of benchmark models, and we can see that which semantics are preferable depend both on the model and on the specification.**

***Index Terms*—Simulation, test generation, testing, embedded systems**

I. INTRODUCTION

ASSURING the quality of Cyber-Physical Systems (CPSs) is an important task that is growing more and more complex. Industrial-size systems with both discrete and continuous dynamics, *i.e.* hybrid systems, require durable methods for design automation [1], as well as validation methods that are beyond the current capabilities of *e.g.* model-checking [2]. Since the general problem of finding the set of reachable states for this kind of systems is undecidable [3], we instead resort to testing the systems. For testing and/or monitoring of CPSs, there are many possible approaches (see [4] for a survey) – in this work, we consider *falsification* of temporal logic specifications. Falsification can be done for CPSs both with the actual hardware, or as in the case of this paper, where the hardware is being simulated.

Falsification of temporal logic specifications for CPSs is a method which attempts to find counterexamples to properties of systems by optimization over *robustness* of the specification. Here, robustness is a measure of distance to violation of the specification. The falsification framework has been shown to be useful for several different applications [5], [6], and it can still be modified in many different ways. For example, one can consider different optimization algorithms to search for the counterexample (*e.g.* ant colony optimization [7] or functional gradient descent [8]).

Falsification requires use of a formal specification, typically written in *Metric Interval Temporal Logic* (MITL) [9] or *Signal Temporal Logic* (STL) [10] (or some variant thereof). However, since these formal logics are not currently well established in industry, it can be difficult to apply falsification when there is no appropriate specification available to test against.

In an attempt to tackle this problem, we present a framework for transforming requirements modelled in a causal, signal-based language (*e.g.* Simulink [11]) into specifications in STL. This allows expert test engineers to model executable requirements using a tool they are familiar with, while also making falsification possible for the models under development.

As an additional measure to enhance the falsification process for industrial-size problems, we apply an alternative robust semantics to be used in the falsification problem. Specifically, we use the additive semantics presented for the logical framework Valued Booleans [12]. We evaluate the performance of additive semantics for several specifications and see in which cases they are preferable to the "standard" semantics of STL robustness.

A. Related work

The main focus of this paper is to adapt the framework of falsification to work better in certain industrial applications. The tools Breach [13] and S-TaLiRo [6] are used to perform falsification with STL and MTL, respectively. Both of these tools are based on the idea of a robustness measure for temporal logic specifications [14]. Apart from falsification, recent research has also focused on *mining* of temporal properties for CPSs [15], which can make it easier to understand what proper specifications could be, given simulations of a system.

When it comes to improvements of the falsification process itself, previous work has defined a modified version of STL [16], and there has been discussion showing the need for similar modifications in industrial applications [17]. The main point has been to improve the robustness information from temporal operators by averaging the robustness inside the timed intervals in question.

Several works [18] [19] have designed methods for faster falsification of a specific sub-class of specifications, namely *request-response specifications*. Recently, an extension to falsification has been proposed where meta-parameters of falsifi-

J. Lidén Eddeland is with Volvo Car Corporation and the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden. E-mail: johan.eddeland@volvocars.com and johan.eddeland@chalmers.se

K. Claessen and N. Smallbone are with the Department of Computer Science and Engineering, Chalmers University and Technology, Gothenburg, Sweden. E-mail: {koen, nicsma}@chalmers.se

S. Miremadi is with Volvo Car Group, Gothenburg, Sweden. E-mail: sajed.miremadi@volvocars.com

K. Åkesson and Z. Ramezani are with the Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden. E-mail: {knut, rzahra}@chalmers.se

cation, *e.g.* the number of control points, are variable and put into an outer optimization problem [20].

Valued Booleans [12] is a recently-proposed logic that captures both the truth value of properties, as well as how severely the properties are falsified. In this paper, we use a version of Valued Booleans to enhance the capabilities of falsification.

B. Contributions

The main contributions of this work are:

i) transformation of causal signal-based requirements into STL specifications;
ii) application of Valued Boolean additive semantics to the falsification process;
iii) evaluation of additive semantics for falsification of benchmark requirements.

The rest of the paper is organized as follows: in Section II, STL and the falsification problem are defined. The latter is used to evaluate different robust semantics later on. In Section III, we define a framework for translating causal signal-based specifications into STL. Section IV details the logic of Valued Booleans, with two kinds of robust semantics. Section V compares the two robust semantics in falsification for a set of benchmark models, and in Section VI our conclusions are presented.

II. Signal Temporal Logic and Falsification

The specification language STL is widely used for falsification of CPSs. We omit the definition of the robust semantics of STL, as it is almost identical to the *max* semantics of VBools, which we define in Section IV-A. For details on STL, we refer the reader to other works [21].

A. Discrete-time signals

Throughout this paper, we discuss specifications defined for signals and signal values. The semantics of VBools is in terms of discrete-time signals, and for the sake of consistency we also define STL this way, even though it is usually defined in terms of continuous-time signals [22]. The main point of doing this is to make it clear how temporal operators can be defined in terms of conjunction, but generalizing to continuous time is possible [23] [14].

Definition 1: A *discrete-time signal* is a function $x[k]$ from a finite subset of $I \subset \mathbb{Z}$ to $\mathbb{R}$, where $k \in I$. The set I labels the time instants of the signals, and the signal takes on continuous values at each of those time instants.

B. Signal Temporal Logic

The grammar of STL formulas is defined as

$$\begin{aligned} \mu &::= x < r \mid x \leq r \mid x \geq r \mid x > r \mid x = r \\ \varphi &::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \Box_{[a,b]}\psi \mid \varphi\, \mathcal{U}_{[a,b]}\psi, \end{aligned}$$

where μ is a predicate, and φ and ψ are STL formulas. We define $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$, and $\Diamond_{[a,b]}\varphi$ as $\neg(\Box_{[a,b]}\neg\varphi)$. Similarly to [24], we define the validity of a formula φ with respect to the discrete-time signal x at time instant k as

$$\begin{aligned}
(x,k) \models \mu \quad &\Leftrightarrow \quad \mu(x[k]) \\
(x,k) \models \neg\mu \quad &\Leftrightarrow \quad \neg((x,k) \models \mu) \\
(x,k) \models \varphi \wedge \psi \quad &\Leftrightarrow \quad (x,k) \models \varphi \wedge (x,k) \models \psi \\
(x,k) \models \varphi \vee \psi \quad &\Leftrightarrow \quad (x,k) \models \varphi \vee (x,k) \models \psi \\
(x,k) \models \Box_{[a,b]}\varphi \quad &\Leftrightarrow \quad \forall k' \in [k+a, k+b], (x,k') \models \varphi \\
(x,k) \models \Diamond_{[a,b]}\varphi \quad &\Leftrightarrow \quad \exists k' \in [k+a, k+b], (x,k') \models \varphi \\
(x,k) \models \varphi\, \mathcal{U}_{[a,b]}\psi \quad &\Leftrightarrow \quad \exists k' \in [k+a, k+b] \ \ (x,k') \models \psi \\
&\qquad \wedge \forall k'' \in [k,k'), (x,k'') \models \varphi
\end{aligned}$$

We will provide an example of STL specification for clarity. The first example is a benchmark specification from [25], informally stated as *"During all simulation times, the engine speed ω and the vehicle speed v never reach $\bar{\omega}$ and $\bar{v}$, respectively."* The corresponding STL formula is

$$\phi_2^{AT} = \Box((\omega < \bar{\omega}) \wedge (v < \bar{v})).$$

ϕ_2^{AT} contains two operators: $\Box$ and $\wedge$. The *modal depth* of a formula is the deepest nesting of temporal operators (*i.e.* $\Box, \Diamond, \mathcal{U}$) in it. For ϕ_2^{AT}, the modal depth is 1.

C. Falsification

Temporal logic falsification is an approach to finding counterexamples to models of CPSs, given a specification in temporal logic. The problem of generating a test case for the CPS is treated as an optimization problem, where one attempts to minimize the robustness of the STL specification, given an input parametrization of the system. Figure 1 illustrates the main falsification procedure used in this paper (with the use of the tool Breach), which we have adapted to use VBools instead of STL robust semantics.

The *Generator* takes the input parametrization to generate an input to the system under test. The *Simulator* generates a simulation trace, which is used together with the specification φ to evaluate VBool robustness for the simulation. The VBool robustness is evaluated to see whether the specification is falsified or not. If it is not falsified, new parameters are sampled and the process is repeated. The *Parameter Optimizer* is a global optimizer which attempts to find new input parameters that are closer to falsifying the specification, *i.e.*, parameters that lead to a lower VBool robustness.

In this work, we investigate two modifications to the falsification procedure. In Section III, we introduce a transformation of signal-based requirements into STL (with a *specification transformer*) as a means of allowing falsification to be performed by testers who are not used to temporal logic specifications. In Section IV, the logic of VBools is introduced which allows the tester to control the objective function used in the falsification optimization problem.

III. Signal-Based Specifications

As has been noted before [26], writing specifications in temporal logic is not trivial. Approaches that have been used

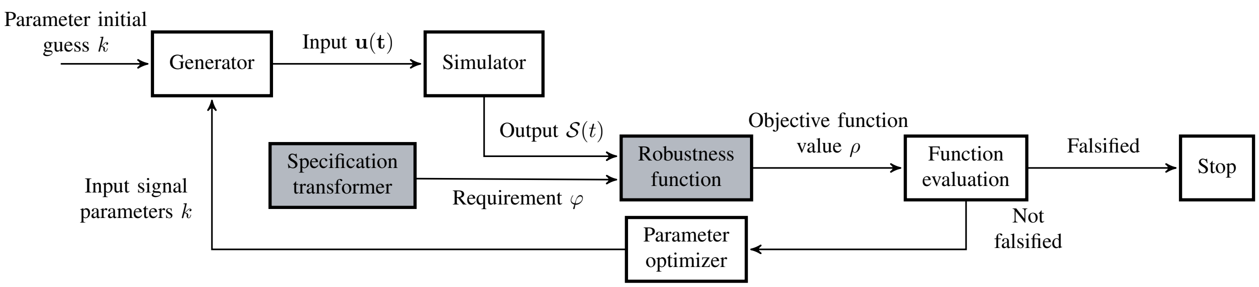


Fig. 1: A flowchart describing a slightly modified version of the optimization-based falsification procedure of Breach. In this paper we deal with defining a specification transformer, as well as considering alternative robustness functions (the two shaded nodes).

to solve this problem are creating tools that make it easier to write specifications [27], automatically detecting faulty specifications [28], and defining template specifications to make it easier for testers to formulate their requirements formally [29]. In this paper, we instead allow test engineers to write specifications in a formalism they already know, namely a causal signal-based framework (in our case using Simulink [11]). The main idea behind a signal-based safety specification is to directly take signals from the simulated system, then using different operators (blocks) to give an output signal that *at each simulated time instant* is either 1 (specification is fulfilled) or 0 (specification is not fulfilled). The advantage of this is that a test can easily be automatically executed and evaluated at the same time as the system itself is simulated.

By using a signal-based specification, we exploit the fact that the test engineers are experts at expressing specifications in, for example, Simulink. The drawback is that a signal-based specification does not compute robustness values, and so can not be directly used for falsification. To solve this problem, we automatically translate signal-based specifications into STL formulas to be used by Breach.

A. STL specifications in a signal-based framework

As an example, we wish to show an implementation of a version of ϕ_1^{AT} from [25], which is defined as

$$\phi_1^{AT} = \Box(\omega < \bar{\omega}). \tag{1}$$

It should be noted that since a specification implemented in Simulink must be causal, temporal operators that look forward in time cannot be explicitly modeled. However, a specification model with a similar meaning to ϕ_1^{AT} (with $\bar{\omega} = 4500$) is presented in Figure 2.

Assume that the specification is evaluated on a simulation trace with a finite set of sampled data points K. The interpretation of the signal $\texttt{req}$ at sample $k \in K$ is then

$$\texttt{req}[k] = \begin{cases} \top & \text{if } \omega[k'] < 4500, \forall k' \in K \cap [0, k] \\ \bot & \text{otherwise,} \end{cases} \tag{2}$$

while the Boolean evaluation of the STL formula ϕ_1^{AT} at time k can be informally expressed as

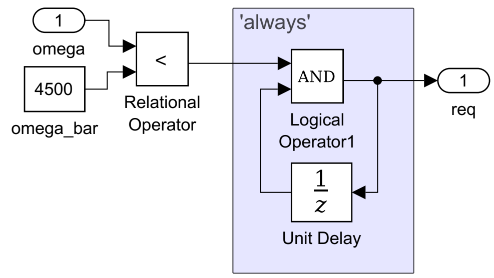


Fig. 2: A simple example of a specification expressed in Simulink. The natural language interpretation is *"During all simulation times $t \in [0, T]$, the engine speed ω never reaches $\bar{\omega}$"*. For the implementation to be correct, the initial condition of the *Unit Delay* block must be non-zero.

$$\phi_1^{AT}[k] = \begin{cases} \top & \text{if } \omega[k'] < 4500, \forall k' \in K \cap [k, k+T] \\ \bot & \text{otherwise.} \end{cases} \tag{3}$$

As can be easily seen, $\texttt{req}(k)$ is not equal to the Boolean evaluation of $\phi_1^{AT}(k)$ for all k, but $\texttt{req}(T) = \phi_1^{AT}(0)$. This is the only thing that is needed to achieve equivalence between the Boolean interpretation of a causal signal-based requirement and its STL equivalent, since the STL formula will be evaluated for time 0, and the signal-based specification will be evaluated at the final simulation time.

B. Signal-based specifications expressed in STL

The goal is to be able to take any signal-based specification, and then transform it into an STL formula so that it can be used for falsification. Ideally, each signal in the signal-based model would be assigned an STL formula, but since the semantics of a signal-based framework are not typically equivalent to the semantics of STL, they have different levels of expressivity.

In this section, a *Signal* is a variable that has defined values over time, and it can be a scalar or a vector. A Signal corresponds to a signal in a causal model. A *Formula* is a special case of a Signal, namely a Signal that always has a Boolean value (*i.e.* it is either true or false).

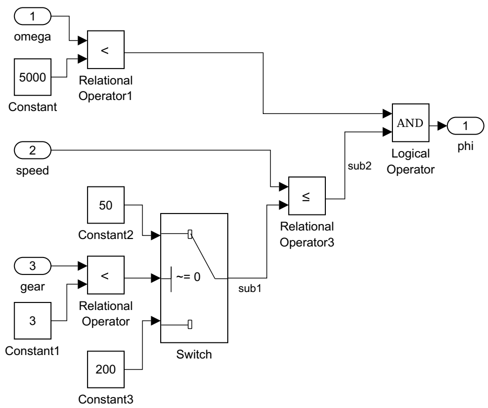


Fig. 3: An example of a requirement with a conditional statement, implemented with the use of a Simulink Switch block.

To model signals whose behaviour varies depending on the value of a Boolean expression, we define the types *FormulaTable* and *SignalTable* as

$$FormulaTable = \mathcal{P}(Formula \times Formula) \qquad (4)$$
$$SignalTable = \mathcal{P}(Formula \times Signal), \qquad (5)$$

where $\mathcal{P}$ denotes the powerset operation. A FormulaTable or SignalTable consists of a set of entries, where each entry is a pair of a precondition, expressed as an STL formula, and a consequent, which is the value taken by the formula or signal when the precondition is true. The disjunction of all preconditions for any FormulaTable or SignalTable must be $\top$.[1]

Figure 3 shows a Simulink encoding of the natural language requirement *"The engine speed ω should always be below 5000 RPM. Additionally, if we are in third gear or lower, the speed v should be below 50 km/h; otherwise, the speed should be below 200 km/h."* The Switch block assigns a value to its output signal according to the rule:

if $gear \leq 3$ **then**
$sub1 = 50$
else
$sub1 = 200$
end if

The signal $sub1$ is translated into a SignalTable, shown in Table I. The signals $sub2$ and phi are translated into FormulaTables, seen in Tables II and III respectively. Since there are two conditions, the SignalTables and FormulaTables have two entries. The SignalTable for $sub1$ has two entries because it is the output of a Switch block; the FormulaTables for $sub2$ and phi have two entries because the FormulaTable for the output of a block has an entry for each possible combination of preconditions from the block's inputs.

As can be seen, when transforming an block from a signal-based model to a FormulaTable or SignalTable, the operator of the block is applied to each consequent of the table. An algorithm $transformBlock$ describing how this is done for a binary operator[2] is shown in Figure 4. The effect of the algorithm is to construct the table:

$$\{(prereq1 \wedge prereq2, operator(conseq1, conseq2)) \\ \mid (prereq1, conseq1) \in in1, (prereq2, conseq2) \in in2\}.$$

The number of entries α in the table that is produced from a block with K inputs $u_1, u_2, \ldots, u_K$ will be $\Pi_{k=1}^{K} \alpha_{u_k}$, where α_{u_k} is the number of entries in the table of input u_k.

An important difference between signal-based specifications and STL specifications is due to conditional blocks. The archetypical conditional block is the *Switch* block, which takes three inputs and lets the output be either the first or the third input, depending on a user-defined condition on the second input. The output table of a Switch block has $\alpha = \alpha_2(\alpha_1 + \alpha_3)$ entries; an algorithm for determining the output table is shown in Figure 5.

To translate a FormulaTable into an STL formula, one can consider the "STL semantics" for a Simulink switch (with

TABLE I: The SignalTable for the signal $sub1$ in the Example in Figure 3.

Precondition	Consequent
$gear < 3$	50
$\neg(gear < 3)$	200

TABLE II: The FormulaTable for the signal $sub2$ in the Example in Figure 3.

Precondition	Consequent
$gear < 3$	$v < 50$
$\neg(gear < 3)$	$v < 200$

TABLE III: The FormulaTable for the output phi in the Example in Figure 3.

Precondition	Consequent
$gear < 3$	$(\omega < 5000) \wedge (v < 50)$
$\neg(gear < 3)$	$(\omega < 5000) \wedge (v < 200)$

function TRANSFORMBLOCK($in1, in2, operator$)
  $outputTable \leftarrow newTable(\langle precond, conseq \rangle)$
  for each $\langle prereq1, conseq1 \rangle \in in1$ **do**
    for each $\langle prereq2, conseq2 \rangle \in in2$ **do**
      $newPrereq \leftarrow (prereq1 \wedge prereq2)$
      $newConseq \leftarrow operator(conseq1, conseq2)$
      $newEntry \leftarrow \langle newPrereq, newConseq \rangle$
      $append(outputTable, newEntry)$
    end for
  end for
end function

Fig. 4: An algorithm $transformBlock$ for transforming the STL formula for a binary operator (block).

[1] In particular, if a FormulaTable or SignalTable only has one entry, the precondition in that entry must be $\top$.

[2] A unary operator is a simplification of the algorithm presented. An n-ary operator, for example $\wedge$, is implemented pairwise (meaning that $a \wedge b \wedge c$ is transformed to $(a \wedge b) \wedge c$, which is possible due to associativity of both max and additive semantics).

```
function TRANSFORMSWITCH(in1, in2, in3)
    outputTable ← newTable(⟨precond, conseq⟩)
    for each ⟨prereq2, conseq2⟩ ∈ in2 do
        for each ⟨prereq1, conseq1⟩ ∈ in1 do
            newPrereq ←
              ((prereq2 ∧ conseq2) ∧ prereq1)
            newConseq ← conseq1
            newEntry ← ⟨newPrereq, newConseq⟩
            append(outputTable, newEntry)
        end for
        for each ⟨prereq3, conseq3⟩ ∈ in3 do
            newPrereq ←
              ((prereq2 ∧ ¬(conseq2)) ∧ prereq3)
            newConseq ← conseq3
            newEntry ← ⟨newPrereq, newConseq⟩
            append(outputTable, newEntry)
        end for
    end for
end function
```

Fig. 5: An algorithm $transformSwitch$ for transforming the STL formula for a Switch block.

inputs x_1, x_2, x_3) as either

$$(x_2 \wedge x_1) \vee (\neg(x_2) \wedge x_3) \tag{6}$$

or

$$(x_2 \implies x_1) \wedge (\neg(x_2) \implies x_3). \tag{7}$$

Note that these two expressions are logically equivalent, but they do not necessarily yield the same robustness value.

C. Recursive loops in specifications

To transform a signal-based specification into STL, we perform a backwards depth-first search from the output of the specification, assigning a FormulaTable or SignalTable to each signal in the specification. For simple specifications, the search algorithm discussed will terminate and assign an STL formula to the signal leading to the outport of the specification. However, any kind of temporal behaviour in a specification is typically implemented as a recursive loop, which leads to the basic search algorithm not terminating – something that needs to be taken care of when transforming the STL formula.

1) Handling recursive loops, approach 1: If the length of the simulation is known and finite, we can transform a recursive loop into a formula that explicitly computes its value in terms of the values at all earlier time steps. For the example presented in Figure 2, this corresponds to the final output

$$req(k) = \bigwedge_{k'=0}^{k} (\omega(k') < \bar{\omega}). \tag{8}$$

However, this results in large and potentially unreadable STL formulas as soon as there is some recursion involved, even for simple specifications. For example, given a simulation time in $[0, 10]$ and a fixed simulation step time of 0.01, requirement (8) results in an STL specification with 1001 $\wedge$-connectives, and more than 31000 characters when written

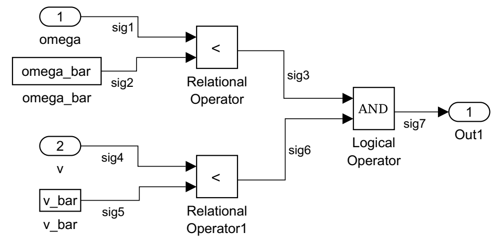


Fig. 6: An implementation of the STL specification $(\omega < \bar{\omega}) \wedge (v < \bar{v})$, which is interpreted as *"The engine speed ω and the vehicle speed v never reach $\bar{\omega}$ and $\bar{v}$, respectively"*.

in Breach syntax. Even though the robustness values for the formula will still be the same as for the STL formula $\varphi = \Box_{[0,10]}(\omega(t) < \bar{\omega})$, we typically want something that is as readable as possible.

2) Handling recursive loops, approach 2: If it is a goal to keep the automatically transformed STL formulas as short as possible, we use *templates* of combinations of different temporal operators that are implemented as their own subsystems in the model. This is in a way very similar to ST-Lib [29], but instead of defining templates that can be used to build specifications from the ground up directly in STL, we define templates in Simulink that are associated to predefined STL formulas.

For the example in Figure 2, one such template could be the $\Box$ operator, which in practice would be a subsystem replacing the blocks in the shaded area.

3) Handling recursive loops, approach 3: A final possibility is to treat a recursive loop as a black box rather than translating it to STL. To do this, we treat the output of the delay block[3] as a signal in the specification, *i.e.* consider anything before the delay block to be part of the model. The value of the signal is computed by the model, and the STL specification simply refers to the signal. This approach is useful when we want to avoid the inefficient encoding of approach 1 and the recursive loop does not correspond to a predefined template. It is also needed when a block applies a general function to its input, in which case the function output cannot be explicitly defined as a formula, but by treating the function as part of the system we are still able to translate the specification to STL.

An extended example of this can be shown by considering the signal-based specification in Figure 6. The specification itself is part of ϕ_2^{AT} [25]. Some different ways to interpret this specification, based on which of the given signals are considered as part of the model (*logged signals*), are shown in Table IV.

The advantage of this approach is that we can be certain to translate *any* signal-based specification to STL, while the disadvantage is that the generated STL specification might be less suited to falsification than had we translated the recursive loops to STL. For example, the specification $(\omega < \bar{\omega}) \wedge (v < \bar{v})$ (for the Automatic Transmission benchmark) has many possible robustness values since the signals ω and v have

[3]Note that a delay block must be present in the loop, otherwise it would be an algebraic loop.

TABLE IV: Some Possible Interpretations of Specification in Figure 6

Logged signals	STL Formula
-	$(\omega < \bar{\omega}) \wedge (v < \bar{v})$
sig1	$(sig1 < \bar{\omega}) \wedge (v < \bar{v})$
sig2	$(\omega < sig2) \wedge (v < \bar{v})$
sig1, sig4	$(sig1 < om\bar{e}ga) \wedge (sig4 < \bar{v})$
sig3	$\neg(sig3 = 0) \wedge (v < \bar{v})$
sig6	$(\omega < \bar{\omega}) \wedge \neg(sig6 = 0)$
sig3, sig6	$\neg(sig3 = 0) \wedge \neg(sig6 = 0)$
sig7	$\neg(sig7 = 0)$

many different potential values. However, the specification $\neg(sig7 = 0)$ (which has exactly the same Boolean truth value) only has two possible robustness values. This makes falsification harder, since the optimization solver will not see how close the specification came to failing.

D. When semantics do not match

For the specification transformation framework presented in this paper, there is a difference between logical formulas and signals. However, in a signal-based setting there is not, so it is possible for a block to get the wrong type of input. For example, consider the expected inputs and outputs of the following blocks:

$$\begin{aligned}
&\wedge : FormulaTable \times FormulaTable \rightarrow FormulaTable \\
&< : SignalTable \times SignalTable \rightarrow FormulaTable \\
&+ : SignalTable \times SignalTable \rightarrow SignalTable
\end{aligned}$$

There are two cases for unexpected input types: either a SignalTable is provided when a FormulaTable should be, or a FormulaTable is provided when a SignalTable should be.

1) SignalTable provided instead of FormulaTable: This can occur if, for example, we apply the $\wedge$ operator to two real-valued signals x and y. Simulink (and MATLAB) semantics interpret the Boolean evaluation of these signals as being false if they are equal to zero, and true otherwise. This means that we can transform a SignalTable to a FormulaTable by comparing equality of the SignalTable's consequent to zero, and then applying the $\neg$ operator. This is accomplished by the $S2F$ function:

$$\begin{aligned}
&S2F : SignalTable \rightarrow FormulaTable \\
&S2F(\langle precond, conseq\rangle) = \langle precond, \neg(conseq = 0)\rangle
\end{aligned}$$

2) FormulaTable provided instead of SignalTable: This can occur if, for example, we try to add (using the + operator) two predicates, such as $x > 0$ and $y < 10$. The meaning of this is clear when interpreted as signals according to the Simulink semantics: the output of the + operator will have value 0 (when both predicates are false), 1 (when exactly one of the predicates are true), or 2 (when both predicates are true). However, in STL we cannot define a formula by adding logical formulas together.

In this case, if the sum is later used as a formula by comparing it to zero (*i.e.* the signal expression to be evaluated is $\big((x > 0) + (y < 10)\big) = 0$), then an equivalent STL formula would be $\neg((x > 0) \vee (y < 10))$. However, it is not clear how to generalize this observation, so instead we consider anything before the block in question (here, the + operator) to be a black box, and the output of the block is treated as a signal, using the same method described in Section III-C3.

IV. Valued Booleans

Valued Booleans (VBools) [12] is a logical framework in which the tester can customize how robustness is computed by choosing between several possible semantics for each connective. The semantics that are currently available are a *max* semantics (which is essentially the same as STL) and an *additive* semantics.

A VBool is formally defined as a pair of a Boolean value and a robustness value. The robustness is a non-negative number, which may be infinite:

$$\mathbb{V} = \mathbb{B} \times \mathbb{R}_{\geq 0}$$

Note the difference between VBools and STL. In STL, there is no explicit Boolean value, but the robustness may be negative, and negative robustness represents falsehood. For VBools, the Boolean value is explicit and robustness may not be negative.

The VBool comparison operator $\leq_v$ is defined as:

$$\leq_v : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{V}$$

$$x \leq_v y = \begin{cases} (\top, y - x) & \text{if} \quad x \leq y \\ (\bot, x - y) & \text{otherwise.} \end{cases}$$

$\top$ and $\bot$ denote true and false, respectively. The other comparison operators are defined in terms of $\leq_v$, except for $=_v$ which is defined as

$$x =_v y = \begin{cases} (\top, K) & \text{if} \quad x = y \\ (\bot, K) & \text{otherwise,} \end{cases}$$

where K is an arbitrary constant. Truth values and negation are defined as

$$\begin{aligned}
\top_\mathrm{v} &= (\top, \infty) \\
\bot_\mathrm{v} &= (\bot, \infty) \\
\neg_v(b, x) &= (\neg b, x).
\end{aligned}$$

The rest of the operators are defined in two different ways. One is called *max* semantics and the other *additive* semantics.

A. Max semantics

The *max and* operator is defined as

$$\begin{aligned}
(\top, x) \wedge_{max} (\top, y) &= (\top, min(x, y)) \\
(\bot, x) \wedge_{max} (\top, y) &= (\bot, x) \\
(\top, x) \wedge_{max} (\bot, y) &= (\bot, y) \\
(\bot, x) \wedge_{max} (\bot, y) &= (\bot, max(x, y)).
\end{aligned}$$

The first clause models the idea that in order to falsify $x \wedge y$, it is enough to falsify whichever of x and y has the lowest robustness. If we are in the second clause, then $x \wedge y$ is false, and in order to make it true, we must make x true; the third clause is similar. The final clause is dual to the first clause: in order to make $x \wedge y$ true we must make both x and y true, and the robustness is determined by whichever of x and y seems to be hardest to make true, i.e., has the highest robustness as a false VBool.

The *max or* operator is defined in terms of the *max and* operator: $(b_x, x) \vee_{max} (b_y, y) = \neg_v(\neg_v(b_x, x) \wedge_{max} \neg_v(b_y, y))$.

The timed *max always* operator (over the interval $[a, b]$) is also defined in terms of the *max and* operator as

$$\Box_{max,[a,b]}\varphi = \bigwedge_{k=a}^{b}{}_{max}\, \varphi[k],$$

where φ is a finite sequence of VBools defined for all the discrete time instants in $[a, b]$.

The timed *max eventually*-operator is defined as $\Diamond_{max,[a,b]}\varphi = \neg(\Box_{max,[a,b]}(\neg_v\varphi))$. Finally, for completeness we also define the *max until*-operator as

$$\begin{aligned}&\varphi\, \mathcal{U}_{max,[a,b]}\ \psi \\ &= \bigvee_{k=a}^{b}{}_{max} \left(\psi \wedge_{max} \left(\bigwedge_{k'=a}^{b-1}{}_{max}\, \varphi[k']\right)\right).\end{aligned}$$

It can be seen that the max semantics for VBool are almost equivalent to the robust semantics of STL, with the only difference being that VBools distinguish between "true with robustness 0" and "false with robustness 0", while STL does not.

B. Additive semantics

The *additive and*-operator is defined as

$$\begin{aligned}(\top, x) \wedge_+ (\top, y) &= \left(\top, \frac{1}{\frac{1}{x} + \frac{1}{y}}\right)\\ (\bot, x) \wedge_+ (\top, y) &= (\bot, x)\\ (\top, x) \wedge_+ (\bot, y) &= (\bot, y)\\ (\bot, x) \wedge_+ (\bot, y) &= (\bot, x + y).\end{aligned}$$

As with the max semantics, the additive semantics for $\wedge$ is based on the observation that in order to falsify $x \wedge y$, it is enough to falsify either x or y. The first clause is inspired by the formula for parallel resistance; the formula $1/(1/x + 1/y)$ gives a robustness which is less than the maximum of x and y. It roughly models the idea that although we need only falsify one of x and y, we do not know which one of them can be falsified. The second and third clauses are the same as in the max semantics. By using addition in the fourth clause rather than max, we model the idea that in order to make $x \wedge y$ true, we need to make both x and y true, not just whichever of them has the highest robustness.

The *additive or*-operator is defined as $(b_x, x) \vee_+ (b_y, y) = \neg_v(\neg_v(b_x, x) \wedge_+ \neg_v(b_y, y))$, and the timed *additive always*-operator (over the time interval $[a, b]$) is defined (similar to the max case) as

$$\Box_{+,[a,b]}\varphi = \bigwedge_{k=a}^{b}{}_{+} (\varphi[k]\#'\delta t),$$

where φ is a finite sequence of VBools defined for the time instants in $[a, b]$, δt is the simulation step time for the time point in question, and $\#'$ is defined as

$$\begin{aligned}(\bot, x)\#'k &= (\bot, x \cdot k)\\ (\top, x)\#'k &= (\top, x/k).\end{aligned}$$

The use of $\#'$ makes the robustness independent of the simulation time step, and means that the robustness of $\Box_{+,[a,b]}\varphi$, if φ is false over the interval $[a, b]$, is equal to the *integral* of the robustness of φ over $[a, b]$.

The timed *additive eventually*-operator is defined as $\Diamond_{+,[a,b]}\varphi = \neg(\Box_{+,[a,b]}(\neg_v\varphi))$.

The *additive until*-operator is defined as

$$\begin{aligned}&\varphi\, \mathcal{U}_{+,[a,b]}\ \psi \\ &= \bigvee_{k=a}^{b}{}_{+} \left((\psi[k]\#'\delta t) \wedge_+ \left(\bigwedge_{k'=a}^{b-1}{}_{+} (\varphi[k']\#'\delta t)\right)\right).\end{aligned}$$

Implication is defined slightly differently than in classical logic:

$$\phi \rightarrow_+ \psi = \neg(\phi\#k) \vee \psi.$$

Here k is an arbitrary constant, and $\#$ scales the robustness of its argument:

$$\begin{aligned}(\bot, x)\#k &= (\bot, x \cdot k)\\ (\top, x)\#k &= (\top, x \cdot k).\end{aligned}$$

By scaling the left-hand side of the implication, we encourage the parameter optimizer to make the left-hand side true before trying to falsify the right-hand side.

C. Properties for reasoning about Valued Booleans

Most Valued Boolean connectives have two possible semantics, and the tester must choose one of the semantics for each connective in the specification. The max semantics corresponds closely to the existing robust semantics of STL, but the additive semantics is entirely different. This section compares the two semantics of Valued Booleans and describes the different properties they have which explain why a tester might choose to use one or the other.

The ultimate goal of a robust semantics is to guide the falsification in the right direction. Therefore, when a change in the input to the system brings a formula closer to being falsified, the robustness of the formula should go down. This is the property we ideally want from a robust semantics. It is not always achievable in reality (because we can never be sure if we are really moving closer to a counterexample or not), but the more often it holds, the better. We are particularly interested in two special cases of this property, *monotonicity* and *sensitivity*:

- A formula[4] is *monotonic* if, when the robustness of some atomic subformula decreases (leaving the others unchanged), the robustness of the formula does *not* increase. A nonmonotonic formula is disastrous for falsification as the parameter optimizer will, moving from a test case to a strictly better one, observe the better test case as being worse instead. All VBool formulas are monotonic.
- A formula is *sensitive* if changing the robustness of some atomic subformula (leaving the others unchanged) causes a change in the robustness of the formula. This captures the idea that if the output of the system changes then the robustness of the formula should usually change.
  Sensitivity is vital for falsification because measuring changes in robustness is how the parameter optimizer explores the input space. (For falsification it is only important that *true* formulas be sensitive, however.) If moving from a test case to a strictly better test case does not affect robustness, then the parameter optimizer will not know when it has found a better test case. The traditional semantics of Boolean logic is completely insensitive, which is why a robust semantics is needed for falsification.
  Unfortunately, the max semantics is *not* sensitive: only one parameter of $p \wedge q$ is taken into account for any given test case. For example, if $p \wedge q$ is true, then the semantics is only sensitive to changes in whichever of p and q has the *lowest* robustness.
  The additive semantics is sensitive for many true formulas. For example, $p \wedge q$ is fully sensitive when p and q are both true. This means that, when falsifying the conjunction of several formulas, the parameter optimizer is able to observe changes in the robustness of any of the subformulas. However, the formula $p \vee q$ is not fully sensitive when exactly one of its arguments is true.

Figure 7 illustrates why sensitivity is important to falsification. Suppose that the formula to be falsified is $\Box\phi$, and that this formula happened to be true in the current test case. Figure 7(a) illustrates how the robustness of ϕ varies with time in this hypothetical test case. Recall that $\Box\phi$ is computed by sampling ϕ at each time step and taking the conjunction of each sample, up to a constant factor depending on δt. In this case, the robustness dips from 3 to 1 at about $t = 48s$, and in both semantics, the robustness of $\Box\phi$ will be lower compared to if the robustness had been a constant 3.

Now suppose that the optimizer modifies the test case and observes the output seen in Figure 7(b). It seems that this test case is closer to failing than Figure 7(a), because there is an extra dip in robustness. Therefore, we would like the optimiser to prefer (b) to (a), and for this to happen the robustness of $\Box\phi$ must be lower under (b) than (a). Under the additive semantics, this is indeed the case, because of sensitivity. Under the max semantics, however, Figures 7(a) and (b) give the same robustness for $\Box\phi$, as the minimum robustness is the same in both cases. Thus the optimiser is not able to see that moving from Figure 7(a) to 7(b) is a good idea. Because the max semantics is not sensitive, the parameter optimizer is only able to notice changes in the minimum value of ϕ.

It is not always the case that additive semantics is better than max semantics. Suppose instead that the optimizer observes the result in Figure 7(c). This test case appears much closer to failing than Figure 7(a): the minimum is very close to 0. However, the additive semantics will assign Figure 7(c) a *higher* robustness than Figure 7(a), because the initial segment of the test case has a higher robustness and continues for a long time, which cancels out the lower minimum. The max semantics considers 7(c) to have lower robustness than 7(a), as we might hope.

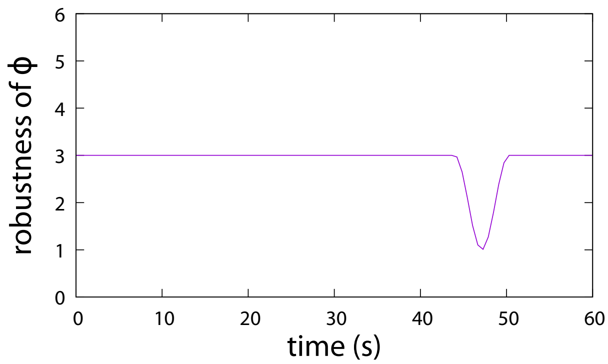
This problem only occurs because the robustness of the initial segment of the test case is quite large. Figure 7(d) shows a less extreme variant. Both the additive and the max semantics judge this test case as having lower robustness than Figure 7(a). This is because, if we take two true VBools $(\top, x)$ and $(\top, y)$, their conjunction under the additive semantics is $(\top, z)$ where $z = 1/(1/x + 1/y)$. Now we can observe that if $x \ll y$, then $\frac{1}{x} \gg \frac{1}{y}$, so $z = 1/(\frac{1}{x} + \frac{1}{y}) \approx x$. That is, when taking the conjunction of a set of formulas, formulas that have a low robustness have a disproportionate effect on the result. In particular, in the formula $\Box\varphi$, a small decrease in the minimum value cancels out quite a large increase in the maximum value.

Figure 8 illustrates the robustness of $p \wedge_+ q$ and $p \wedge_{max} q$. The x-axis gives the robustness of p and the y-axis gives the robustness of q; negative values here stand for false VBools. The graph illustrates the robustness of $p \wedge q$ using isolines, which connect points that have equal robustness. Where an isoline is vertical or horizontal, the connective is insensitive: only changes in a particular argument have an effect on robustness. We see in the upper-right quadrant that when p and q have very different robustnesses, $p \wedge_+ q$ assigns much more importance to the lower robustness (it starts to approximate the max semantics), but that it always remains sensitive. This weighting is a deliberate feature of the additive semantics: a subformula with low robustness is likely to be a better target for optimization than a subformula with high robustness, as it it more likely to be easily falsifiable.

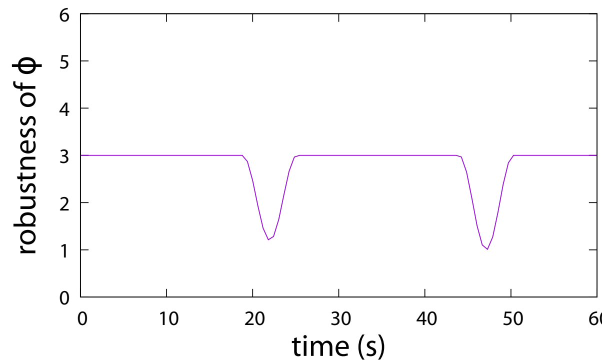
Apart from monotonicity and sensitivity, there are several more commonplace properties that we would like our semantics to have. The most essential is *soundness*: a Valued Boolean formula (e.g. $p \wedge_+ (\neg_V q \vee +r)$) and the corresponding Boolean formula (in this case, $p \wedge (\neg q \vee r)$ should always evaluate to the same Boolean result; the only difference is that the Valued Boolean also computes a robustness. All of the connectives we have defined are easily seen to be sound, since the Boolean part of each definition uses the corresponding Boolean connective. Therefore, the choice of semantics only affects the optimization process, not the truth or falsehood of the property.

We would also like the usual laws of Boolean logic to hold: connectives should be associative, commutative, idempotent, have an identity element, have a zero element, and obey the usual distributivity and negation laws. As mentioned above,

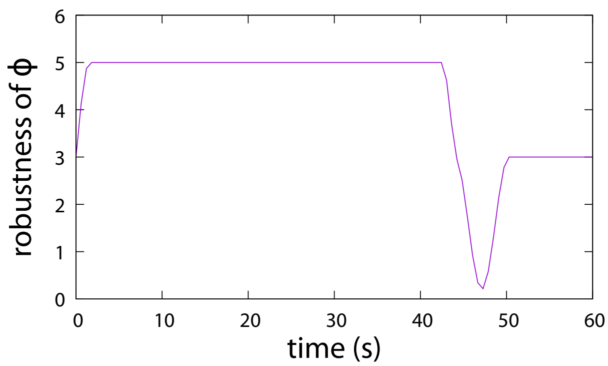
[4]For simplicity we assume that formulas are in negation normal form, i.e., negation only occurs as part of an atomic formula.



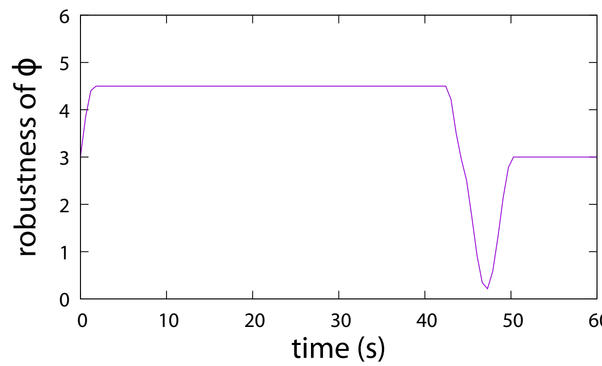
(a) An example of a signal defining a property ϕ. Robustness is lowest at about 48 seconds.



(b) In this trace, the property comes close to being falsified twice, at 20 seconds and 48 seconds. The "max" semantics assigns this trace the same robustness as for figure (a), even though it is strictly worse. The "+" semantics assigns it a lower robustness.



(c) This trace is similar to figure (a), but the robustness in the initial part of the trace is even higher. The "+" semantics assigns this trace a higher robustness than figure (a), even though it appears much closer to being falsified.



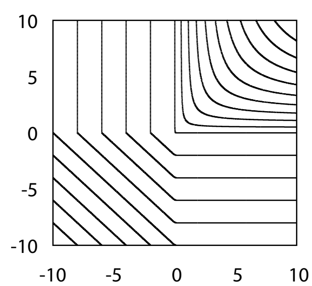
(d) In this trace, the property comes very close to being falsified at 48 seconds, but is more robustly true the rest of the time. The "max" semantics assigns this trace a lower robustness than figure (a). The "+" semantics also assigns it a slightly lower robustness.

Fig. 7: Four graphs showing the value of a hypothetical property ϕ over time. The different definitions of robustness assign a different robustness to $\Box\phi$.

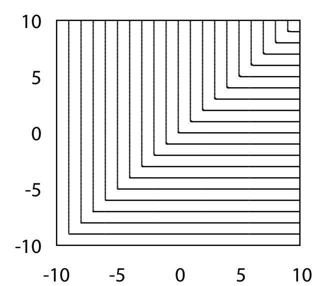
these laws all hold if one ignores the computed robustness, but we would like robustness to respect these laws too. These properties are important because we do not want the robustness of a formula to depend on, for example, how conjunctions are bracketed or what order they are written in, and we do not want the tester to have to think about what arrangement of brackets is most suitable.

The max semantics obeys all laws of Boolean logic, except for the laws $p \vee_{max} \neg_v p = \top$ and $p \wedge_{max} \neg_v p = \bot$. The infinitely-robust truth values $\top_v$ and $\bot_v$ act as identity and zero elements in the usual way.

The additive connectives satisfy fewer laws than the max semantics. They are associative, commutative, have $\top_v$ and $\bot_v$ as identity and zero elements, and respect de Morgan's laws. They do not satisfy idempotence or distributivity. Idempotence fails because, for example, $p \wedge_+ p$ is a Valued Boolean whose robustness is either twice that of p (if p is false) or half that of p (if p is true). Distributivity fails for a similar reason, because expanding $p \wedge_+ (q \vee_+ r)$ duplicates p, increasing its influence on the robustness computation. We are not aware of a semantics that combines associativity, commutativity, idempotence and sensitivity; we conjecture that these four properties are incompatible.[5]



(a) The robustness of $x \wedge_+ y$.



(b) The robustness of $x \wedge_{max} y$.

Fig. 8: Isobar plots of the robustness of the two semantics of $\wedge$. Here, negative robustnesses represent false VBools.

To summarise, both max and additive semantics satisfy many Boolean properties, but max satisfies more; in return for giving up some properties, the additive semantics gains sensitivity, which is useful for falsification. In an additive

[5]One could for example recover idempotence by multiplying or dividing by 2 in the definition of $\wedge_+$, but this would destroy associativity.

conjunction, the parameter optimizer is able to see when any of the conjuncts' robustness decreases, which is not the case for the max semantics. A final observation is that the additive semantics for conjunction assigns greater weight to less robust conjuncts, which means that when a conjunct is close to being falsified it can be reduced even if this causes the robustness of other conjuncts to increase markedly.

V. Results and Discussion

To show the performance of using additive semantics for STL during falsification (compared to max semantics), we perform falsification with additive semantics for four examples. The results are presented in a set of tables, and the layout of each table is the same. The results are shown in Sections V-A - V-D.

The rows of the tables show which specification is attempted to be falsified, which parameters or specific settings are used, and also which semantics are used. We use the max and additive semantics defined earlier in the paper, but we also include a third *constant* semantics. The robustness value for a constant semantics is equal to 100 if the specification is true, and -100 if the specification is not true. This constant semantics is used as a baseline to verify whether max and additive semantics yield better results than purely random testing[6].

For each parameter setting, the "Succ" column shows how many times the specification was actually falsified, and the "Iter" column shows the average number of iterations used by the optimization solver in each falsification attempt (the maximum is set to 1000). The "Iter/Succ" column shows the average number of iterations for the *falsification attempts that were successful*. The optimization solver used in these examples is a Simulated Annealing solver [30].

A. Automatic Transmission Benchmark

The model takes as input the throttle and brake of a vehicle, and simulates the automatic transmission system (for details, see [25]). The model has been used in several other works [15], [16], and in this work we perform falsification with the Breach toolbox. The outputs of the system are the vehicle speed (v), the engine speed (ω), and the gear.

The model is simulated with a fixed-step setting (automatic step size), using the MATLAB solver ode5 (Dormand-Prince).

1) Falsification parameters: The throttle is generated using 7 control points distributed evenly in time, interpolated using the MATLAB interpolation setting `pchip`. Each control point has a value in the range $[0, 100]$. The brake input is interpolated similarly but only using 3 control points, each in the range $[0, 500]$.

The specifications to falsify are shown in Table V. Specifications φ_1–φ_6 are taken from [16]. Note, however, that we do not modify the specifications to improve the falsification capability of our additive semantics.

[6]We have also implemented a *random* semantics, where the robustness at each sample is a uniform random number, but with correct sign (for STL robustness). Falsification for random semantics performs worse than max, additive and constant semantics for all the examples in this paper.

TABLE V: Specifications to Falsify for the Automatic Transmission Benchmark.

Specification	Formula
φ_1	$\Diamond_{[0,T]}(\omega \geq 2000)$
φ_2	$\Box\Diamond_{[0,T]}(\omega \leq 3500 \vee \omega \geq 4500)$
φ_3	$\Box_{[0,T]}(\neg(gear == 4))$
φ_4	$\Diamond(\Box_{[0,T]}(gear == 3))$
φ_5	$\bigwedge_{i=1,\ldots,4} \Box((\neg(gear == i) \wedge \Diamond_{[0,\epsilon]}(gear == i) \implies (\Box_{[\epsilon,T+\epsilon]}(gear == i)))$
φ_6	$\Box_{[0,T]}(v \leq 85) \vee \Diamond(\omega \geq 4500)$
φ_7	$(\Box_{[0,1]} gear == 1) \wedge (\Box_{[2,4]} gear == 2) \wedge(\Box_{[5,7]} gear == 3) \wedge (\Box_{[8,10]} gear == 3) \wedge(\Box_{[12,15]} gear == 2)$
φ_8	$\Box_{[0,20]}((gear == 4 \wedge throttle > 45 \wedge throttle < 50) \implies \omega < \bar{\omega})$

The comparison between max semantics and additive semantics for each specification is shown in Table VI.

B. Abstract Fuel Control Benchmark

The model is an Abstract Fuel Control system implemented in Simulink, and it has been proposed as a benchmark for temporal logic falsification [31]. The inputs to the model are the input throttle θ (in degrees) and the engine speed ω. The outputs of interest are the Air/Fuel ratio λ and the controller mode (either closed-loop or open-loop). The reference value λ_{ref} is equal to 14.7 for the specifications we are considering.

The model is simulated with a variable-step setting using the MATLAB solver ode15s (stiff/NDF).

1) Falsification parameters: The engine speed is constant and allowed to be in the range $[900, 1100]$. The throttle angle is generated as a pulse signal with a base value of 8.9, a delay of 3, a period in the range $[10, 30]$ and amplitude in the range $[0.161]$. Thus, the throttle angle always has a value in the range $[8.9, 69.9]$, always switching back and forth between two values at different times of each simulation. We always simulate the system for 40 seconds.

The specifications to falsify are shown in Table VII. The specifications are variations of Req. (26) and (27) in [31], using $\eta = 1$.

The results for the Abstract Fuel Control benchmark are shown in Table VIII.

C. Third Order $\Delta - \Sigma$ Modulator

The third order $\Delta - \Sigma$ modulator is used as a technique for analog to digital conversion. The model is described in detail in [32] and has previously been used for falsification benchmark purposes [30], [20]. The model has one input U, three states x_1, x_2, x_3, and three initial conditions $x_1^{init}, x_2^{init}, x_3^{init}$.

The model is simulated with a fixed-step setting (automatic step size), using the MATLAB solver discrete (no continuous states).

1) Falsification parameters and specification: The input U is constant during the whole simulation, and the allowed values are in different sets for different scenarios (see Table X for

TABLE VI: Results for the automatic transmission benchmark.

Specification	Semantics	Parameters								
		$T = 20$			$T = 30$			$T = 40$		
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ
φ_1	Max	20	103.1	103.1	19	209.1	167.5	14	500.6	286.6
	Additive	20	80.3	80.3	20	133.2	133.2	20	215.1	215.1
	Constant	14	734.0	619.9	3	930.5	536.3	0	1000.0	-
		$T = 10$								
		Succ	Iter	Iter/Succ						
φ_2	Max	16	247.9	59.9						
	Additive	20	172.1	172.1						
	Constant	20	277.3	277.3						
		$T = 4$			$T = 4.5$			$T = 5$		
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ
φ_3	Max	0	1000.0	-	9	796.4	547.6	17	467.8	373.9
	Additive	0	1000.0	-	10	736.0	472.0	17	532.9	450.4
	Constant	0	1000.0	-	11	641.9	348.8	16	472.9	341.1
		$T = 1$			$T = 2$					
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ			
φ_4	Max	5	852.7	410.8	20	182.0	182.0			
	Additive	6	795.2	317.3	20	90.6	90.6			
	Constant	1	998.4	967.0	20	160.0	160.0			
		$T = 0.8$			$T = 1$			$T = 2$		
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ
φ_5	Max	0	1000.0	-	12	754.4	590.7	20	60.1	60.1
	Additive	0	1000.0	-	4	864.5	322.5	20	90.7	90.7
	Constant	0	1000.0	-	13	704.6	545.5	20	64.7	64.7
		$T = 10$			$T = 12$					
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ			
φ_6	Max	9	731.4	403.0	20	153.5	153.5			
	Additive	12	665.9	443.1	20	182.9	182.9			
	Constant	0	1000.0	-	4	899.1	495.5			
		Succ	Iter	Iter/Succ						
φ_7	Max	4	905.4	527.0						
	Additive	15	493.3	324.4						
	Constant	4	836.7	183.5						
		$\hat{\omega} = 3000$			$\hat{\omega} = 3500$					
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ			
φ_8	Max	20	16.9	16.9	0	1000.0	-			
	Additive	20	20.4	20.4	19	296.1	259.1			
	Constant	20	10.1	10.1	0	1000.0	-			

TABLE VII: Specifications to Falsify for the Abstract Fuel Control benchmark

Specification	Formula
φ_1^{AFC}	$\Box_{[11,40]}(\lvert\frac{\lambda(t)-\lambda_{ref}}{\lambda_{ref}}\rvert < tol)$
φ_2^{AFC}	$\Box_{[11,35]}\big((\theta(t) < \theta(t+0.01) \vee \theta(t) > \theta(t+0.01)) \implies \Box_{[1,5]}(\lvert\frac{\lambda-\lambda_{ref}}{\lambda_{ref}}\rvert < tol)\big)$

detailed scenarios). The initial conditions are all in the range $[-0.1, 0.1]$. The specifications to falsify are shown in Table IX (note that $\varphi_1^{\Delta-\Sigma}$ is equivalent to $((\varphi_2^{\Delta-\Sigma} \wedge \varphi_3^{\Delta-\Sigma}) \wedge \varphi_4^{\Delta-\Sigma})$).

The results for the modulator benchmark are shown in Table X.

D. Static Switched System

The static switched system has no dynamics and is included to show that both max and additive semantics can worsen the performance of falsification, compared to falsifying with Boolean semantics. The model is inspired by [33], and it has two inputs $(u_1, u_2) \in [0, 1]^2$ which are kept constant. The output $y(t)$ is assigned according to

$$y = \begin{cases} -2(u_1 + u_2) - 5 & \text{if } \ u_i \geq thresh, \ \forall i \\ 2((u_1 + 1)^2 + (u_2 + 1)^2) & \text{otherwise.} \end{cases}$$

The specification to falsify is $\varphi^{SS} = \Box(y \geq 0)$. In other words, the falsification problem consists of finding a scenario where both inputs have a value above $thresh$. This is difficult since the gradient of the robustness (for max and additive) with respect to the input parameters will point away from the area where the specification is falsified. The results for the static switched system are shown in Table XI.

E. Transforming Volvo requirements to STL

We have successfully implemented the framework presented in this paper for transforming causal signal-based specifications into STL. We have transformed the requirements for two

TABLE VIII: Results for the Abstract Fuel Control benchmark.

Specification	Semantics	$tol = 0.16$ Succ	$tol = 0.16$ Iter	$tol = 0.16$ Iter/Succ	$tol = 0.17$ Succ	$tol = 0.17$ Iter	$tol = 0.17$ Iter/Succ	$tol = 0.18$ Succ	$tol = 0.18$ Iter	$tol = 0.18$ Iter/Succ
φ_1^{AFC}	Max	20	313.2	313.2	9	758.6	463.7	3	934.1	560.7
	Additive	19	589.2	567.6	7	837.8	536.6	2	962.0	620.0
	Constant	14	564.4	377.7	2	967.8	678.0	1	971.1	423.0
φ_2^{AFC}	Max	20	319.4	319.4	7	786.6	390.4	2	954.8	548.0
	Additive	19	545.5	521.6	2	942.8	428.0	2	962.5	625.5
	Constant	12	680.8	468.0	3	937.8	585.0	3	929.1	527.7

TABLE IX: Specifications to Falsify for the Third Order $\Delta-\Sigma$ Modulator

Specification	Formula
$\varphi_1^{\Delta-\Sigma}$	$\Box\left(\bigwedge_{i=1}^{3}(-1 \leq x_i \wedge x_i \leq 1)\right).$
$\varphi_2^{\Delta-\Sigma}$	$\Box(-1 \leq x_1 \wedge x_1 \leq 1)$
$\varphi_3^{\Delta-\Sigma}$	$\Box(-1 \leq x_2 \wedge x_2 \leq 1)$
$\varphi_4^{\Delta-\Sigma}$	$\Box(-1 \leq x_3 \wedge x_3 \leq 1)$

industrial models at Volvo Car Corporation, which model the electric machine of an electric vehicle, as well as the battery for an electric vehicle. The models contain 19846 and 18294 blocks, respectively[7].

In total, there are 58 transformed requirements for the first model and 36 transformed requirements for the second model. The transformation of requirements into STL specifications have enabled the use of temporal logic falsification for both models. Falsification is now being run continuously for both models, in order to catch software defects during development. Statistics for the transformed STL formulas are shown in Table XII.

F. Discussion

For the specifications shown in this paper, we can see that no specific semantics perform better than the other two for all models. For several specifications, the constant semantics performs just as well as one or other of the robust semantics.

It is clear from the tables that sometimes max semantics are preferable, and sometimes additive semantics are preferable. For example, for specifications $\varphi_1, \varphi_2, \varphi_7, \varphi_8$ additive semantics clearly perform better, while for specifications $\varphi_5, \varphi_6, \varphi_1^{AFC}, \varphi_2^{AFC}$ max semantics clearly perform better. The static switched system was also introduced to show that the constant semantics (*i.e.* random testing) can be better than max or additive semantics. It is clear that φ^{SS} is easier to falsify for the constant semantics than for the other semantics.

Whether a specific semantics outperforms the others depends not only on the specification, but also on the system that is being falsified.

1) Preferable semantics for different specifications: The intuitive explanation for why additive semantics can be better in some cases is that it takes into account all the different subformulas of $\wedge$ and $\vee$ formulas (and by extension also the temporal operators). In a conjunction, if only the highest robustness value decreases in between simulations, it is not certain that the max semantics will capture the change, but it will affect the total additive robustness. On the other hand, if one robustness increases while the other robustness decreases, the additive semantics robustness may not be affected, while the max semantics robustness will.

An example of when it is preferable to notice changes in all clauses of a conjunction is φ_7. Here, each clause is not difficult to falsify individually, but to be able to falsify them all at once it helps a great deal to include more detailed robustness information about each clause. As such, having conjunctions with many clauses in a specification can indicate that additive semantics would be preferable for that (sub)-specification.

2) Preferable semantics for different systems: For some system behaviour, it can be non-beneficial to consider changes in all parts of a conjunction. An example of this is the third order $\Delta-\Sigma$ modulator. The results in Table X indicate that $\varphi_4^{\Delta-\Sigma}$ is by far the easiest sub-specification of $\varphi_1^{\Delta-\Sigma}$ to falsify. Including more detailed robustness information about the other sub-specifications ($\varphi_2^{\Delta-\Sigma}$ and $\varphi_3^{\Delta-\Sigma}$) makes the robustness information from $\varphi_4^{\Delta-\Sigma}$ diluted in a sense, meaning that changing from max to additive semantics will not increase falsification capability.

VI. Conclusions

We have presented two additions to potentially increase the capability of falsification of temporal logic specification for Cyber-Physical Systems.

The first addition is a specification transformation framework, which takes requirements modeled in a causal signal-based frameworks and transforms them into Signal Temporal Logic (STL) formulas. The framework has been implemented for the specifications in two industrial-sized models at Volvo Car Corporation, and it has enabled the use of falsification for both of the models. The specification transformation outputs a specification where we also have information about which preconditions should be fulfilled for different parts of the specification to be evaluated for given signal values and a given time.

The second addition is the introduction of additive semantics in the falsification process. Considering the established robust semantics of STL formulas as the max semantics, the difference for additive semantics is that the robustness of each clause in a conjunction can affect the total robustness

[7]The block counts include blocks in referenced models.

TABLE X: Results for the Third Order $\Delta - \Sigma$ modulator.

Specification	Semantics	Parameters								
		$U \in [-0.35, 0.35]$			$U \in [-0.40, 0.40]$			$U \in [-0.45, 0.45]$		
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ
	Max	12	799.7	666.2	19	281.6	243.7	20	144.3	144.3
$\varphi_1^{\Delta-\Sigma}$	Additive	20	296.8	296.8	20	335.1	335.1	12	730.8	551.3
	Constant	20	205.5	205.5	17	513.3	427.4	4	875.0	374.8
		$U \in [-0.35, -0.35]$								
		Succ	Iter	Iter/Succ						
	Max	0	1000.0	-						
$\varphi_2^{\Delta-\Sigma}$	Additive	0	1000.0	-						
	Constant	0	1000.0	-						
		$U \in [-0.35, -0.35]$								
		Succ	Iter	Iter/Succ						
	Max	0	1000.0	-						
$\varphi_3^{\Delta-\Sigma}$	Additive	0	1000.0	-						
	Constant	0	1000.0	-						
		$U \in [-0.35, -0.35]$								
		Succ	Iter	Iter/Succ						
	Max	13	627.0	426.2						
$\varphi_4^{\Delta-\Sigma}$	Additive	12	739.5	565.9						
	Constant	5	872.6	490.4						

TABLE XI: Results for the Static Switched System

Specification	Semantics	Parameters								
		$thresh = 0.7$			$thresh = 0.8$			$thresh = 0.9$		
		Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ	Succ	Iter	Iter/Succ
	Max	15	566.3	421.7	10	741.1	482.2	3	937.7	584.7
φ^{SS}	Additive	16	518.4	397.9	7	861.3	603.7	3	943.3	622.0
	Constant	20	118.3	118.3	20	136.1	136.1	20	373.4	373.4

TABLE XII: Statistics of STL Formulas for the two Volvo models

Model	Number of operators			Depth			Modal depth		
	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
Electric machine	1	61.1	336	1	7.65	15	1	2.02	4
Battery	2	33.5	171	1	7.95	16	1	2.08	4

of the conjunction, even if only one of the clause's robustness changes. Disjunction and temporal operators are defined in terms of conjunction for the additive semantics.

To indicate the usability of additive semantics for falsification, we have compared them to max semantics as well as *constant* semantics (essentially only Boolean information and no robustness) for several different models and specifications. The models we show results for are both well-known benchmark models, as well as a simple non-dynamic model to prove that all three choices of semantics can be the most viable. Previous work on falsification has overlooked the need to compare against the constant semantics as a baseline; our evaluation made it clear that for some specifications, falsification had no benefit over random testing. We encourage other researchers to include a baseline comparison in their future work.

Which of the three semantics performs best depends both on the specification and the model. In a black-box setting, it is thus very difficult to decide which semantics to use for which operator in the specification to get the best results for falsification.

A. Future work

We have so far defined two semantics for Valued Booleans. There are most likely many more, each with their own trade-offs; we plan to explore these. Also, since the best choice of semantics can be different for each connective in a given specification, we would like to both

- formulate principles that can guide a tester in choosing a suitable semantics for each operator in a given specification, and
- analyze *both* the model and the specification to reason about which semantics would be best for falsification (*i.e.* grey-box or white-box testing).

Finally, it would be interesting to look at falsification which includes the extra information that we get from the specification transformation presented in this paper – namely, the information about all the preconditions that need to be fulfilled for different parts of the specification to be evaluated.

Acknowledgment

The authors would like to thank Alexandre Donzé for his helpful comments on a draft of this paper. This work has been performed with support from the Swedish Governmental

Agency for Innovation Systems (VINNOVA) project TESTRON 2015-04893 and from the Swedish Research Council (VR) project SyTeC 2016-06204. This support is gratefully acknowledged.

REFERENCES

[1] S. A. Seshia, S. Hu, W. Li, and Q. Zhu, "Design automation of cyber-physical systems: Challenges, advances, and opportunities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 9, pp. 1421–1434, 2017.

[2] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: algorithmic verification and debugging," *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.

[3] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*. ACM, 1995, pp. 373–382.

[4] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.

[5] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using s-taliro," in *American Control Conference (ACC), 2012*. IEEE, 2012, pp. 3567–3572.

[6] Y. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems." in *TACAS*, vol. 6605. Springer, 2011, pp. 254–257.

[7] Y. S. R. Annapureddy and G. E. Fainekos, "Ant colonies for temporal logic falsification of hybrid systems," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2010, pp. 91–96.

[8] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius, "Functional gradient descent method for metric temporal logic specifications," in *American Control Conference (ACC), 2014*. IEEE, 2014, pp. 2312–2317.

[9] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[10] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, pp. 152–166.

[11] The MathWorks, Inc., Natick, Massachusetts, "Simulink R2013b," 2017.

[12] K. Claessen, N. Smallbone, J. Eddeland, Z. Ramezani, and K. Åkesson, "Using valued booleans to find simpler counterexamples in random testing of cyber-physical systems," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 408–415, 2018.

[13] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems." in *CAV*, vol. 10. Springer, 2010, pp. 167–170.

[14] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[15] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining requirements from closed-loop control models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.

[16] T. Akazaki and I. Hasuo, "Time robustness in MTL and expressivity in hybrid system falsification," in *International Conference on Computer Aided Verification*. Springer, 2015, pp. 356–374.

[17] J. Eddeland, S. Miremadi, M. Fabian, and K. Åkesson, "Objective functions for falsification of signal temporal logic properties in cyber-physical systems," in *International Conference on Automation Science and Engineering*, 2017, pp. 1326–1331.

[18] A. Dokhanchi, S. Yaghoubi, B. Hoxha, and G. Fainekos, "Vacuity aware falsification for mtl request-response specifications," in *Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE17)*, 2017.

[19] T. Akazaki, "Falsification of conditional safety properties for cyber-physical systems with gaussian process regression," in *International Conference on Runtime Verification*. Springer, 2016, pp. 439–446.

[20] A. Aerts, B. Tong Minh, M. Reza Mousavi, and M. A. Reniers, "Temporal logic falsification of cyber-physical systems: An input-signal space optimization approach," in *14th Workshop on Advances in Model Based Testing (A-MOST)*, 2018.

[21] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems: 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*, K. Chatterjee and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–106.

[22] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals." in *FORMATS*, vol. 6246. Springer, 2010, pp. 92–106.

[23] G. E. Fainekos and G. J. Pappas, "Robust sampling for mitl specifications," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2007, pp. 147–162.

[24] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th international conference on hybrid systems: Computation and control*. ACM, 2015, pp. 239–248.

[25] B. Hoxha, H. Abbas, and G. Fainekos, "Benchmarks for temporal logic requirements for automotive systems," *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.

[26] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Metric interval temporal logic specification elicitation and debugging," in *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*. IEEE, 2015, pp. 70–79.

[27] B. Hoxha, N. Mavridis, and G. Fainekos, "Vispec: A graphical tool for elicitation of mtl requirements," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 3486–3492.

[28] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Formal requirement debugging for testing and verification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, p. 34, 2018.

[29] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia, "St-lib: A library for specifying and classifying model behaviors," SAE Technical Paper, Tech. Rep., 2016.

[30] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 95, 2013.

[31] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain control verification benchmark," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*. ACM, 2014, pp. 253–262.

[32] T. Dang, A. Donzé, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *International Conference on Formal Methods in Computer-Aided Design*. Springer, 2004, pp. 21–36.

[33] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos, "Requirements driven falsification with coverage metrics," in *Proceedings of the 12th International Conference on Embedded Software*. IEEE Press, 2015, pp. 31–40.