Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems

Citation for the original published paper (version of record):
Lidén Eddeland, J., Donzé, A., Miremadi, S. et al (2020). Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems. EPiC Series in Computing, 74: 267-274.
http://dx.doi.org/10.29007/r74f

(article starts on next page)

EPiC Series in Computing
Volume 74, 2020, Pages 267–274
ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)

Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems*

Johan Lidén Eddeland[1][2], Alexandre Donzé[3], Sajed Miremadi[1], and Knut Åkesson[2]

[1] Volvo Car Corporation, Gothenburg, Sweden
{johan.eddeland, sajed.miremadi}@volvocars.com
[2] Chalmers University of Technology, Gothenburg, Sweden
{johan.eddeland, knut}@chalmers.se
[3] Decyphir SAS, Moirans, France
alex@decyphir.com

Abstract

In this benchmark proposal, we present a set of large specifications stated in Signal Temporal Logic (STL) intended for use in falsification of Cyber-Physical Systems. The main purpose of the benchmark is for tools that monitor STL specifications to be able to test their performance on complex specifications that have structure similar to industrial specifications. The benchmark itself is a Git repository which will therefore be updated over time, and new specifications can be added. At the time of submission, the repository contains a total of seven Simulink requirement models, resulting in 17 generated STL specifications.

1 Introduction

We present a set of large specifications stated in Signal Temporal Logic (STL [5]) intended for use in falsification of Cyber-Physical Systems. In the research area of falsification, earlier benchmarks typically consist of potentially complex systems, but where the specifications are small and easy to get an overview of [3, 4, 2]. To contrast this, the specifications we present in this benchmark are complex and difficult to get an overview of without examining them extensively. The specifications are inspired by specifications used in Model-in-the-loop (MIL) and Software-in-the-Loop (SIL) testing at Volvo Car Corporation.

*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.

Due to the proprietary nature of the specifications at Volvo Car Corporation, the specifications we share are modified and anonymized by remodeling them for use with the Automatic Transmission model (originally from Mathworks [6]) that has been used in previous benchmarks [3]. Even though we cannot share exactly how the proposed specifications differ from those at Volvo Car Corporation, we believe they are still useful to the falsification community because of the following points.

1. The specifications exemplify how one can model specifications in a graphical way (using Simulink);
2. The STL specifications are generally much larger than specifications proposed in previous benchmarks; and
3. The overall structure of the specifications are similar to specifications found at Volvo Car Corporation.

The specifications in Simulink are automatically translated to STL specifications by use of the tool specTransformer[1]. For further details on how this translation is performed, we refer to our previous work [1]. Because of the size of the generated specifications, one potential use for them is to test how robust a monitoring tool is to large formulas. It should be noted that the generated STL formulas use past temporal operators, *e.g. once* and *historically*. For further details on STL, including future and past operators, we refer to [5].

The benchmark is available as a public Git repository[2]. It is a living repository which can and will receive updates to the current requirement models and parameters, and new requirements can be added over time as well.

2 Requirement Models

In this section we first briefly introduce Signal Temporal Logic, then we describe the overall structure of the requirement models included in the benchmark. The reason for using requirement models modeled in Simulink at Volvo Car Corporation is both because the engineers are typically well-versed in the tool for modelling purposes, but also because this automatically allows evaluation of the requirements every time the entire system is simulated.

2.1 Signal Temporal Logic

The grammar of STL formulas is defined as

$$\varphi ::= \pi^{\mu} \mid \neg\varphi \mid \varphi \wedge \psi \mid \Box_{[a,b]}\, \psi \mid \varphi\, \mathcal{U}_{[a,b]}\psi \mid \boxdot_{[a,b]}\psi.$$

Here, π^{μ} is an atomic predicate, and φ and ψ are STL formulas. The truth value of π^{μ} is determined by the sign of a real-valued function μ. $\wedge$ denotes logical *and*, $\Box_{[a,b]}$ is the timed *globally* (or *always*) operator, $\mathcal{U}_{[a,b]}$ is the timed *until* operator, and $\boxdot_{[a,b]}$ is the timed *historically* (or *past always*) operator. We define logical *or* $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$, the timed *eventually* operator $\Diamond_{[a,b]}\varphi$ as $\neg(\Box_{[a,b]}\neg\varphi)$, and the timed *once* operator $\Diamonddot_{[a,b]}\varphi$ as $\neg(\boxdot_{[a,b]}\neg\varphi)$. Similarly to earlier works [7], we define the validity of a formula φ with respect to the signal x at time t_k as

[1] https://github.com/JohanEddeland/specTransformer

[2] https://github.com/decyphir/ARCH20_ATwSS

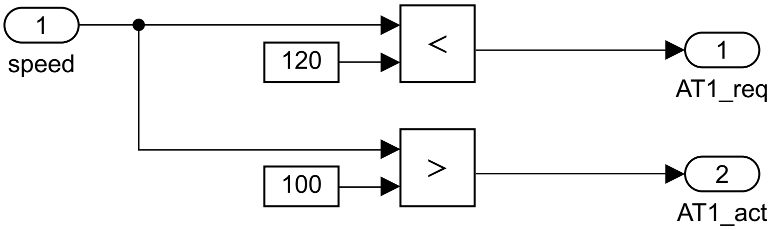

Figure 1: An example of the requirement structure used in the benchmark. Each requirement model contains one safety specification, here `AT1_req`, and one or more activation specifications, here `AT1_act`.

$$
\begin{aligned}
(x,t_k) \models \pi^{\mu} &\iff \mu(x_k) > 0\\
(x,t_k) \models \neg\varphi &\iff \neg((x,t_k) \models \varphi)\\
(x,t_k) \models \varphi \wedge \psi &\iff (x,t_k) \models \varphi \wedge (x,t_k) \models \psi\\
(x,t_k) \models \varphi \vee \psi &\iff (x,t_k) \models \varphi \vee (x,t_k) \models \psi\\
(x,t_k) \models \Box_{[a,b]}\varphi &\iff \forall t_{k'} \in [t_k + a, t_k + b], (x,t_{k'}) \models \varphi\\
(x,t_k) \models \Diamond_{[a,b]}\varphi &\iff \exists t_{k'} \in [t_k + a, t_k + b], (x,t_{k'}) \models \varphi\\
(x,t_k) \models \varphi\, \mathcal{U}_{[a,b]}\psi &\iff \exists t_{k'} \in [t_k + a, t_k + b] \;\; (x,t_{k'}) \models \psi\\
&\qquad \wedge\, \forall t_{k''} \in [t_k, t_{k'}], (x,t_{k''}) \models \varphi\\
(x,t_k) \models \boxdot_{[a,b]}\varphi &\iff \forall t_{k'} \in [t_k - b, t_k - a], (x,t_{k'}) \models \varphi\\
(x,t_k) \models \dot{\Diamond}_{[a,b]}\varphi &\iff \exists t_{k'} \in [t_k - b, t_k - a], (x,t_{k'}) \models \varphi
\end{aligned}
$$

2.2 Simple requirement model example

Each Simulink requirement model contains at least two outputs, where each of these outputs is Boolean (either true or false) and corresponds to an STL formula. To be more specific, each requirement model contains exactly one *safety* requirement and one or more *activation* requirements – this is inspired by how testing of requirements is performed at Volvo Car Corporation. The safety requirement must hold at all times, meaning that the corresponding STL formula is of the form $\varphi_{req} := \Box\varphi_{req}^{inner}$. An activation requirement is supposed to give complementary information for a safety requirement by telling the user whether a specific requirement has been tested or not (interpreted by the modeller of the requirement). This means that if a given activation requirement is ever true during a simulation of the system, the requirement is considered to be activated – hence, for falsification purposes it is desired to find input parameters that do not satisfy the following formula, $\varphi_{act} := \neg\Diamond\varphi_{act}^{inner}$ (or equivalently formulated $\varphi_{act} := \Box(\neg\varphi_{act}^{inner})$).

To clarify the basic structure of a Simulink requirement model, we show a small example. The example is a modified version of ϕ_1^{AT} which has been used in previous benchmarks [3], with the addition of an activation specification. The Simulink implementation is shown in Figure 1.

The interpretation of the safety specification is that the engine speed never reaches 120, while the activation specification states that the requirement has been activated if the speed ever goes above 100. The purpose of adding this activation specification is for example to create regression testing suites – even though we may not be able to falsify each and every safety specification, a good test suite should at least fulfill each activation specification. In practice, this becomes a three-valued logic with the following cases (for a given test case).

1. If the safety specification is falsified, there is a fault in the system which should be rectified;

2. If the safety specification is satisfied and the activation specification is fulfilled, there is no identified fault in the system, but the given test case is considered to have "activated" the requirement and is therefore useful to include in a test suite; and

3. If the safety specification is satisfied and the activation specification is not fulfilled, the given test case is of no particular interest when evaluating the given requirement.

For the example in Figure 1, assuming a simulation in the time range $[0, T]$ and with v denoting the `speed` signal, the corresponding STL specifications are $\varphi_{req}^{AT1} := \square_{[0,T]}(v < 120)$ and $\varphi_{act}^{AT1} := \square_{[0,T]}(\neg(v > 100))$. This means that a simulation corresponds to case 1 above if the speed is ever greater than 120, the simulation corresponds to case 2 if the speed is ever greater than 100 but never above 120, and the simulation corresponds to case 3 if the speed is never greater than 100. Note that for this specific example, the activation specification is always fulfilled whenever the safety specification is satisfied, however this is not always the case. When the safety specification is falsified, it does not matter whether the activation specification is fulfilled or not, since there exists a fault in the system which should be dealt with.

2.3 Requirement models in the benchmark

The requirement models presented in the benchmark contain more complex structures and many more blocks than the example in Figure 1. The name of each requirement model is a three letter abbreviation, *e.g.* *ADA* or *BTL*, which has no other meaning than to differentiate the different models from each other. To be able to simulate the requirement models, there are many parameter values that need to be set (which are manifested in the models as values in "Constant" blocks). An example of this can be seen in Figure 2, which shows an excerpt of the requirement model *BTL* from the benchmark. To be able to simulate the *BTL* requirement, there needs to be values set to the variables `BTL_preconditionSpeedLim`, `BTL_preconditionGear`, `BTL_preconditionSub1Value`, and `BTL_preconditionSub1Value2`.

It is important to note that the requirement models are not designed to make sense for the particular model – there has been no attempt to interpret the benchmark requirements in natural language. The point of the proposed benchmark is *not* to have a set of specifications that the Automatic Transmission model *should* reasonably fulfill, but rather to have a set of complicated requirements that can also have instances, *i.e.* parameter valuations, that make the requirements difficult but possible to falsify. A further discussion of these instances can be found in Section 3.

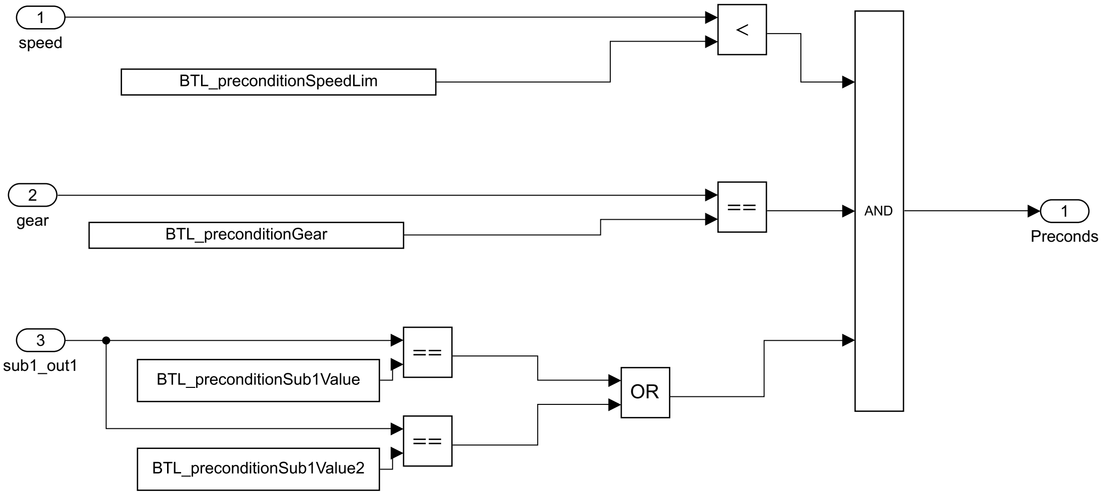

Figure 2: An excerpt of the requirement model *BTL* in the benchmark. Specifically, what is shown is the subsystem called *Preconditions*, which is a conjunction of conditions that need to be fulfilled to both falsify the safety specification `BTL_req` and satisfy the activation specifications `BTL_act1`, `BTL_act2`.

3 Instance Tuning Method and Organization

As mentioned in Section 2.3, certain parameters need to have values set to be able to simulate the different requirements in the benchmark. As there typically exists a large number of parameters for each requirement model, there are many possible parameter combinations, each of which will potentially yield different falsification rates.

For testing at Volvo Car Corporation, the satisfaction of *all* requirements is monitored for each executed test case. If we wanted to consider the problem of falsification in a similar fashion, it would correspond to including a conjunction of all the specifications derived from the requirement models. However, we also note that there is no interaction between the different requirement models in the benchmark, meaning that each specific parameter is part of exactly one requirement model. As such, we could also consider the specifications corresponding to a single requirement model to be its own falsification problem, meaning that the process of falsifying all requirement models would be stated as many different falsification problems in total.

3.1 Parameter instances

In the benchmark repository, we provide a *base* instance of parameter values that aims to yield falsification rates above 0% and below 100% for each requirement, in 1100 simulations. In this case, the falsification rates are calculated for a specific input parametrization, in order to represent the input to the system with a discrete set of parameter values. More specifically, the input throttle has 7 control points distributed evenly over the simulation time, while the input brake has 3 control points distributed evenly over the simulation time. Each control point for the throttle has a value in the range $[0, 100]$, while each control point for the brake has a value in the range $[0, 325]$. The simulation time is 30 seconds, and the input values are interpolated between control points using MATLAB's `pchip` setting.

To calculate the falsification rate for a set of parameters, we sample parameters in two different ways; by corner samples, and by pseudo-random samples. Corner samples are inputs for which the control points only take extreme values. We also provide *hard* instances for several of the specifications; for a hard instance, one or several requirement parameters have been changed to make it more difficult to falsify the specification. Specifically, the changed parameters yield zero or close-to-zero falsification rates, which means that the parameter set can be used for a somewhat interesting falsification problem. Note that to get these falsification rates, we ran 100 corner samples and 1000 pseudo-random samples using one specific seed. To get more reliable falsification rates, one would need to repeat this procedure for many different seeds, which is something we aim to do in the future.

3.2 Instance tuning method

To start with[3], the parameters of all requirement models had values that were in respective signals ranges. To clarify, consider a parameter `speed_parameter` used to compare against the signal `speed`, yielding a predicate `speed > speed_parameter`. By running 100 simulations using the stated input parametrization, we observed the minimum and maximum values of the `speed` signal, and started with `speed_parameter` having a value in this range. For temporal parameters (such as a time horizon for a temporal operator), the minimum value was the simulation step time, and the maximum value was chosen to make sure that the specification could be monitored within the simulation time of the system.

This initialization process was repeated for all parameters, after which we could calculate initial falsification rates for each requirement. Following this, we repeatedly simulated and observed signal values in order to find out which parameters needed to be changed to get a falsification rate above 0% and below 100% (ideally aiming for a rate between 1% and 10%).

4 Preliminary Results

In Table 1, we show statistics of the generated STL specifications, which indicate the relative difference in complexity between the different specifications. The table shows the numbers of operators, the nested temporal depth[4], the average monitoring time when monitored by Breach over 1000 simulations, and the falsification rates for the *base* and *hard* scenarios. The monitoring time is measured on a computer with Intel Xeon E5 2.60GHz and 64 GB RAM. For this model, each simulation takes 0.26 seconds, meaning that a majority of the total computational time is spent monitoring STL formulas. Note that for some specifications, there exist several sets of parameters for the *hard* scenario; in this case, only the falsification rate of one of those sets are presented in the table. For other specifications, there currently exists no parameter set for the *hard* scenario, something we aim to add in future versions of the benchmark. For the specifications showing 0% falsification rate, it is as of yet unknown whether the specifications are falsifiable using the given scenario and input parametrization.

[3] Note that we do not detail the step of going from requirement models at Volvo Car Corporation to the requirement models presented in the benchmark, since that could contain sensitive company information.

[4] The nested temporal depth is defined as the maximum number of nested temporal operators in a formula. For example, $\square_{[0,T]}(v < 120)$ has a depth of 1, so has $\square_{[0,T]}(v < 120) \wedge \square_{[0,T]}(rpm < 4500)$ but $\square_{[0,T]}(\Diamond v > 20)$ has a depth of 2, etc.

Table 1: Statistics of the generated STL specifications in the benchmark. For each specification we show the number of operators, the nested temporal depth, the average monitoring time (when monitored by Breach over 1000 simulations), and the falsification rates (FR) for both the *base* and *hard* parameter sets described in Section 3.1. These falsification rates are calculated over 1100 simulations (100 corner samples and 1000 pseudo-random samples). We also include, in parentheses, information about whether the specification was falsified by a corner sample (c), a pseudo-random sample (r), or both (cr).

Specification	#operators	Temporal depth	Time (s)	Base FR	Hard FR
AOT_req	11	2	0.028	20.1% (cr)	0.18% (c)
AOT_act	25	2	0.049	6.2% (cr)	- (-)
AFE_req	20	2	0.049	9.3% (r)	0.36% (r)
AFE_act	7	1	0.024	8.6% (r)	0% (-)
ASL_req	112	2	0.19	4.1% (cr)	- (-)
ASL_act	23	2	0.046	4.1% (r)	0.18% (r)
RFC_req	233	4	0.40	5.8% (r)	0.18% (r)
RFC_act	10	4	0.027	22.9% (cr)	0% (-)
ADA_req	316	2	0.62	10.1% (cr)	0.09% (r)
ADA_act	7	2	0.02	10.5% (r)	0.09% (r)
ADI_req	896	4	2.00	7.9% (cr)	- (-)
ADI_act1	287	5	0.61	11.5% (r)	- (-)
ADI_act2	67	3	0.15	2.5% (cr)	- (-)
ADI_act3	17	2	0.038	34.2% (cr)	- (-)
BTL_req	1562	2	2.60	5.1% (cr)	0% (-)
BTL_act1	191	2	0.33	4.5% (r)	0% (-)
BTL_act2	191	2	0.33	4.5% (cr)	0% (-)

5 Conclusions

We presented a set of requirements expressed in STL for a standard automotive model, for the purpose of benchmarking STL falsification and monitoring algorithms. This set complements and differs from previous benchmarks in several ways. The requirements are directly inspired by industrial specifications for large systems from Volvo Car Corporation, and their STL formulations are typically larger and more complex. Efforts have been put in tuning the requirement parameters so that the falsification problems are neither trivial nor seemingly impossible; the tuning is still going on at the time of submission and future instances will be tracked with a rigorous version control system for easier references and comparisons. The requirements were treated in a mostly independent manner, however as mentioned in the text, in practice they are checked together and do not all have the same function. So-called *activation* requirements act as coverage criterion, in other words, they "have" to be falsified for the testing to be deemed valid, whereas other requirements are actual safety requirements, i.e., falsification is indicative of a design flaw. Another ongoing and future work is to devise efficient algorithms to treat the set of specifications as a whole, taking into account this difference in function between safety requirements and activation requirements.

References

[1] J. L. Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi, and K. Åkesson. Enhancing temporal logic falsification with specification transformation and valued booleans. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. Accepted for publication.

[2] Gidon Ernst, Paolo Arcaini, Alexandre Donze, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2019 category report: Falsification. *EPiC Series in Computing*, 61:129–140, 2019.

[3] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 25–30. EasyChair, 2015.

[4] 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.

[5] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, volume 4800 of *Lecture Notes in Computer Science*, pages 475–505. Springer, 2008.

[6] Mathworks. Modeling an automatic transmission and controller. https://mathworks.com/videos/modeling-an-automatic-transmission-and-controller-68823.html.

[7] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248. ACM, 2015.