FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Real-Time Monitoring for Correctness and Robustness

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Stefan Jakšić, MSc
Matrikelnummer 01429719

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof Radu Grosu, Dr.rer.nat
Zweitbetreuung: Dr. Dejan Ničković

Diese Dissertation haben begutachtet:

Prof. Saddek Bensalem	Prof. Jyotirmoy Deshmukh

Wien, 20. September 2018

Stefan Jakšić

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Real-Time Monitoring for Correctness and Robustness

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Stefan Jakšić, MSc
Registration Number 01429719

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof Radu Grosu, Dr.rer.nat
Second advisor: Dr. Dejan Ničković

The dissertation has been reviewed by:

Prof. Saddek Bensalem

Prof. Jyotirmoy Deshmukh

Vienna, 20th September, 2018

Stefan Jakšić

Erklärung zur Verfassung der Arbeit

Stefan Jakšić, MSc
Barichgasse 35/7, 1030 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. September 2018

Stefan Jakšić

Acknowledgements

This thesis represents the accomplishements of the research conducted in order to obtain PhD degree at TU Wien. I am very grateful that I had the chance to work with TU Wien and Austrian Institute of Technology (AIT) for three and a half years.

I would like to thank my supervisor professor Radu Grosu and co-supervisor Dr. Dejan Ničković for their mentoring and support. During my studies, I frequently collaborated with professor Ezio Bartocci, Dr. Konstantin Selyunin and Dr. Thang Nguyen. I would like to thank them all for the fruitful collaboration we had.

Two most important figures in my life encouraged me along the way. I want to thank my mother Borislava and my wife Ivana for inspiring me and giving me the strength to complete this challenging journey. Thank you for always believing in me. My entire family was always there for me, and I cannot thank them enough. However, I must specifically acknowledge my cousin Nikola Jeličić for being a true friend in need.

I would also like to thank the professors from the Faculty of Electrical Engineering, University of Belgrade, who supported my idea to pursue a PhD: professor Boško Nikolić, professor Zaharije Radivojević and professor Slavko Gajin. At AIT I enjoyed a friendly working environment for which I want to thank my colleagues: Niveditha Manjunath, Cristinel Mateis, Thorsten Tarrach, Andreas Fellner, Willibald Krenn, Stefan Zelenbaba, Mario Klima and Dieter Holzer.

Last, but definitely not the least, are all the friends that shared so many important moments with me: Đorđe Žegarac, Vuk Miljković, Filip Dojčinović, Veljko Stamenković, Adam Miladinović, Marko Janković, Vuk Batanović, Đorđe Grozdić, Mihailo and Nenad Ivanović, Andrija Bogojević, Jelena Vukmirović, Davor Ostojić, Srđan Rajković, Nikola Zečević and Rohit Dureja.

The research was funded by a national Austrian grant from FFG (Österreichische Forschungsförderungsgesellschaft) under the program IKT der Zukunft, for which I am grateful.

Kurzfassung

Ziel dieser Dissertation ist es, ein Framework für die Echtzeitüberwachung von Designs industrieller Größenordnungen zu entwickeln, um die Korrektheit und Robustheit zur Laufzeit zu überwachen. Wir glauben, dass die Laufzeitverifikation, die auf temporaler Logik basiert, eine rigorose, systematische und effiziente Lösung für die Herausforderungen der Echtzeitüberwachung bietet.

Um unsere Monitore in moderne Hardware-in-the-Loop (HiL) - Prüfstände zu integrieren, präsentieren wir einen Algorithmus zur Synthese qualitativer Monitore in Hardware aus Sicherheitseigenschaften, welche in Signal Temporal Logic (STL) ausgedrückt werden. Indem wir den Monitor aus grundlegenden temporalen Testern zusammensetzen, können wir die Berechnung des Korrektheitsurteils parallelisieren. Somit minimieren wir die Rechenverzögerung und erhalten reaktionsfähige Monitore, die Echtzeit-Korrektheitsbeurteilungen liefern.

Binäre Korrektheitsurteile sind möglicherweise nicht ausreichend für Eigenschaften, die Verhaltensweisen mit reellen Werten bewerten. Die Korrektheitsrelation kann durch ein metrikbasiertes quantitatives Urteil ersetzt werden, welches ein feineres Maß dafür liefert, wie robust ein Verhalten einer Spezifikation entspricht. In der Regel wird die Entwicklung solcher Urteile von einer bestimmten Art von Anforderungen und Verhaltensweisen bestimmt. Dies führte zu einer Fülle von Urteilen, jedes mit seiner eigenen Definition und seinem eigenen Algorithmus. Unser Ziel ist es, einen einheitlichen Ansatz zur präzisen Robustheitsüberwachung zu definieren, indem wir einen regulären Formalismus für reguläre Spezifikationen über große, geordnete Alphabete verwenden: symbolische Automaten (SA). Durch die Verwendung von algebraisch definierten Gewichten in SAs erhalten wir einen generischen Algorithmus zur Überwachung sowohl der Korrektheit als auch der Robustheit. Es liefert Urteile, die an die gewünschte Klasse von Verhaltensweisen angepasst werden können, indem einfach die entsprechenden algebraischen Operationen ausgewählt werden. Da der Algorithmus auf dynamischer Programmierung basiert, berechnet er Urteile in Echtzeit und ermöglicht eine direkte Implementierung in Hardware.

Ein typisches CPS nutzt digitale Verarbeitungseinheiten, um Prozesse in einer physischen Umgebung zu kontrollieren. Während der Ausführung zeigt ein CPS ein komplexes Verhalten und generiert Ablaufprotokolle. Um die Konformität von CPS-Ablaufprotokollen mit der Spezifikation zu bewerten, müssen wir die Diskrepanzen im Wert- und Zeitbereich berücksichtigen. Um Einflüsse aus beiden Domänen angemessen darzustellen, definieren wir einen Robustheitsgrad basierend auf Weighted Edit Distance (WED). Wir demonstrieren einen Echtzeitalgorithmus für

die Berechnung des Robustheitsgrads von Ablaufprotokollen eines seriellen Protokolls SENT, das aus der Automobilindustrie stammt.

Wir führen eine große Fallstudie durch, um die Vorteile der Verwendung von Laufzeitmonitoren in einem industriellen Umfeld zu bewerten und die Wiederverwendung von Monitoren auf verschiedenen Abstraktionsebenen und Produktdesignphasen zu fördern. Um die Monitore auf eine Klasse von asynchronen, seriellen Protokollen anzuwenden, die häufig in der Industrie verwendet werden, rüsten wir sie mit einem Verfahren zur Regenerierung nach Fehlern und Protokollierung aus. Die Fähigkeit zur Selbstregenerierung ermöglicht die Überwachung langer Datenströme auch nach mehreren Anforderungsverletzungen.

Abstract

The objective of this thesis is to develop a framework for real-time monitoring of Cyber-Physical System (CPS) designs for correctness and robustness and to offer verification methods applicable to real-scale industrial designs. We believe that runtime verification based on temporal logic provides a rigorous, systematic and efficient solution to the challenge of real-time monitoring.

In order to integrate our monitors with modern Hardware-in-the-Loop (HiL) testbenches, we provide an algorithm for synthesizing qualitative monitors in hardware from safety properties expressed in Signal Temporal Logic (STL). By composing the monitor from basic temporal testers, we are able to parallelize the computation of the satisfaction verdict. Thus, we minimize computational delay and obtain responsive monitors which provide real-time correctness verdicts.

Binary satisfaction verdicts may not be sufficient for properties evaluated against real-valued behaviors. The satisfaction relation can be replaced by a metric-based quantitative verdict which gives a finer measure of how *robustly* a behavior satisfies a specification. Typically, the development of such verdicts is driven by a specific type of requirements and behaviors. This resulted in a plethora of verdicts, each with its own definition and algorithm. We aim to define a uniform approach to precise robustness monitoring, by leveraging a common formalism for regular specifications over large ordered alphabets: symbolic automata (SA). By introducing algebraically-defined weights into SA, we obtain a generic algorithm for monitoring both correctness and robustness. It provides verdicts which can be adapted to the desired class of behaviors simply by selecting the appropriate set of algebraic operations. Since the algorithm is based on dynamic programming it computes verdicts in real-time and admits direct hardware implementation.

A typical CPS exploits digital processing units to control processes in physical environment. During the execution, a CPS exhibits complex behavior and generates execution traces. For evaluating conformance of CPS traces with the specification, we must consider trace discrepancies in value and time domain. In order to appropriately represent influences from both domains we define a robustness degree based on Weighted Edit Distance (WED). We demonstrate a real-time algorithm for computing WED-based robustness degree of traces of a serial protocol SENT, taken from the automotive domain.

We conduct a large case study to assess benefits of using runtime monitors in an industrial setting and promote monitor reuse on different abstraction levels and product design phases. In order to apply the monitors to a class of asynchronous serial protocols, frequently used in the industry, we equip them with a procedure for error recovery and logging. The ability to self-recover allows to continue monitoring long data streams even after multiple requirement violations occur.

Contents

List of Figures

List of Tables

List of Algorithms

Thesis Publications

The thesis is based on the following research papers, published in proceedings of international conferences and journals.

- Stefan Jakšić, Ezio Bartocci, Radu Grosu, Reinhard Kloibhofer, Thang Nguyen, Dejan Ničković: From signal temporal logic to FPGA monitors. *In 13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA,* September 21-23, 2015, pp.218–227, 2015.

- Stefan Jakšić, Ezio Bartocci, Radu Grosu, Dejan Ničković: Quantitative Monitoring of STL with Edit Distance. *In Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings,* pp.201–218, 2016.

- Thang Nguyen, Ezio Bartocci, Dejan Ničković, Radu Grosu, Stefan Jakšić, Konstantin Selyunin: The HARMONIA Project: Hardware Monitoring for Automotive Systems-of-Systems. *In Proceedings of the* 7^{th} *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications (ISoLA 2016) Corfu, Greece* pp.371–379, October 10-14, 2016.

- Konstantin Selyunin, Stefan Jakšić, Thang Nguyen, Christian Reidl, Udo Hafner, Ezio Bartocci, Dejan Ničković, Radu Grosu: Runtime Monitoring with Recovery of the SENT Communication Protocol. *In Proceedings of CAV 2017: the 29th International Conference on Computer Aided Verification, Heidelberg, Germany* pp.336–355, July 24-28, 2017.

- Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, Dejan Ničković: Quantitative Monitoring of STL with Edit Distance. *In International Journal on Formal Methods in System Design, 2018.*

- Stefan Jakšić, Ezio Bartocci, Radu Grosu, Dejan Ničković: An Algebraic Framework for Runtime Verification. *International Conference on Embedded Software* EMSOFT 2018 (to appear).

CHAPTER 1

Introduction

Modern society is ever more dependent on computer systems in all aspects of life. Computer systems admit wide range of applications such as transportation, medical, banking and many more. Computational and networked systems that interact with the physical environment, called Cyber-Physical System (CPS), typically deploy digital controllers to react to stimuli originating from physical environment. CPS in which a failure can result in catastrophic consequences, such as loss of human lives, are commonly designated as safety-critical. The design constraints for such systems often depend on their level of criticality. Poorly implemented vehicle cruise control software will always have more impact than a malfunctioning microwave. In order to comply with safety standards the manufacturers have to ensure very low failure rates [ISO]. Verification is the process of ensuring CPS reliability and compliance.

Devices which implement interaction between analog components and digital components, such as CPS, impose a significant verification challenge for simulation-based methods which are still frequently used in the industrial practice. Rigorous verification of mixed-signal CPS requires simulating the complex dynamics of the system, typically defined by a set of differential equations. Thus, the verification of such devices is time consuming. In case of complex CPS verification usually takes more time than the design process. Reducing verification time allows a company to release a product before the competition and to gain a crucial advantage on the market.

Verification is an inevitable process which also determines the feasibility of a system. Over the past years, several catastrophes highlighted the importance of verification. A hidden bug inside a CPS design is very costly: the cost of 2011 Intel Sandy Bridge processor recall is estimated at stunning 1 billion US dollars [Shi11]. NASA space mission crashes [Dow97], Ariane 5 failure [Dow97], a blackout which leaves millions without electricity [ADF+05] and many similar cases [LT93, Pri95, KK13] are a clear indicator about the importance of verification.

In spite of all the complexity of the verification problem, our cars still deploy electronic systems when necessary, planes are flying and pacemakers allow many people to continue with their lives. These are witnesses that the industry is able to respond to many verification challenges, with

more or less effort. Our research is aiming to improve CPS verification by increasing rigor, speed and automation and proliferate theoretical novelties into industrial practice.

In order to explain the context of our research, in the following sections we provide a scientific and an industrial point-of-view on verification. Exhaustive verification methods originate from scientific community and aim to provide firm correctness guarantees for all possible executions of the system. In an industrial setting, verification teams favor a simulation-based approach which can achieve usable results in domains such as digital semiconductor design and CPS testing.

Finally we present a lightweight formal verification method, Runtime Verification (RV), which is focused on observing and evaluating individual behaviors. In semiconductor industry this approach is also known as Assertion-Based Verification (ABV) and is used to complement the CDV methods. RV is an integral part of Falsification Testing as well.

1.1 Exhaustive Verification Methods

To present the exhaustive verification methods, we focus on hybrid systems verification and model checking. These methods offer full correctness guarantees, but the price is paid with limited usability due to scalability issues.

1.1.1 Model Checking

Model checking [CE81, QS82, BKL08] is an exhaustive verification technique aiming to explore the model of a system in order to check whether *all* the possible executions of a system, defined by a Kripke structure, conform to the specification. Model checking aims to guarantee correctness, or find counterexamples which are typically described by temporal patterns. Conversely to CDV which adopts a black-box approach, model checking directly leverages the knowledge of the structure of the model. Model checking is applicable when we have access to the model of a SUT, when the size of the model can be handled, and when the results of model checking are guaranteed to hold for real system as well.

Model checking has evolved from explicit state model checking to symbolic model checking [BCM$^+$92] and bounded model checking [BCC$^+$03]. The latter applies a different strategy: instead of exhaustive model exploration, it tries to incrementally explore finite length paths of a model in order to find the counterexample. The problem of checking a requirement over finite paths is encoded as boolean satisfiability problem (SAT) problem. Although efficient in practice, bounded model checking cannot guarantee the absence of bugs. Introduction of Counterexample-guided Abstraction Refinement (CEGAR) [CGJ$^+$00] approach led to further reducing the size of the space by allowing to abstract away the irrelevant parts of the model. Model checking did contribute to proliferation of formal methods in the industrial practice [WBKW07, Lev04].

Due to scalability issues, model checking can be applied to formally prove absence of bugs only in critical parts of complex devices (e.g. floating point arithmetic in a Central Processing Unit (CPU)). Recent advances show that with sufficient effort it is possible to verify a complex processor pipeline control [RCD$^+$16]. An other important concern for verification teams is the

return of investment. Formal verification techniques based on model checking have significant ramp-up time and limited re-usability. Only certain cases for limited number of features have shown to scale successfully so far.

1.1.2 Verification of Hybrid Systems

Systems which operate in different modes with continuous dynamics, defined by differential equations, can be modeled with hybrid automata [ACHH93, MMP91]. A hybrid automaton can either make a discrete transition between the locations, or stay in the location and evolve according to a particular differential equation specified for that state. The evolution of a location is memorized in a variable, which will be tested as a condition whether to take a discrete transition or not.

Full safety verification of hybrid systems is an undecidable problem [HKPV95, ACH+95]. Therefore, the mainstream approach to verification of hybrid systems is reachability analysis. The solution paradigm moved from computing the exact reachable set to computing over-approximations of the reachable set of states. It is possible to trade complexity of dynamics for the size of the automata to achieve better scalability. With breakthrough in models for representing approximations of reachable states, namely Zonotopes and support functions, modern tools for hybrid systems verification can now handle hundreds of continuous state variables. Many of these concepts are implemented in a tool SpaceEx [FGD+11]. However, the formal methods for verification of hybrid systems [DDM04, FKR06, LWJ+10, SH08, ARK+13] still do not scale well enough due to the state-space explosion. The compositional-based method described in [ABB16] shows good scalability results for linear hybrid systems, but suffers from false positives.

1.2 Systematic Testing

The industrial verification methods extensively rely on testing which include systematic generation of input stimuli according to a specific metric. In the world of digital design, Coverage Driven Verification (CDV) achieved significant success and has been established as a de-facto standard. Falsification Testing (FT), which proved very useful for systems with complex dynamics, uses *robustness degree* as metric to select optimal test parameters.

1.2.1 Coverage Driven Verification

Digital hardware industry has a long tradition in developing verification methodologies based on own experience and trials. These methodologies fall into Coverage Driven Verification (CDV) approach. The main motivation which brought about the development of CDV, was the very high cost of failure of semiconductor products [Shi11].

Due to processing power increase over the last decades, simulation has become computationally cheaper. Major semiconductor solution vendors put CDV in focus and offer a palette of Verification IPs, adherent to the principles of CDV. VIPs are protocol-specific verification blocks which can be directly integrated into an existing verification environment. Verification teams usually

prefer deploying an established VIP from a major vendor, rather than investing time and effort to develop it in-house.

Digital design requires no additional effort for modeling the device: register transfer level (RTL) implementation is exactly an instance of simulation-ready model.CDV starts with defining a set of coverage goals to be achieved by the verification environment and the System Under Test (SUT). Coverage goals bound the part of system's execution space which is relevant and necessary to check. Verification engineers create a verification environment for an SUT according to the rules from a certain methodology such as Universal Verification Methodology [Gla09, IJ13, IEE17]. CDV environments exploit randomization of stimulus in order to achieve coverage goals. The engineers specify the constraints for generating meaningful stimulus for the scenarios of interest. Thus, CDV implements constraint-driven random generation of stimuli. During this process the environment will drive the stimuli to the inputs of System Under Test (SUT), perform self-checking of the monitored features and track the achieved coverage progress. The tests are repeatedly executed until a saturation point is reached when achieved coverage does not increase. At this point verification teams restore to old-fashioned directed tests to exercise the unreachable corner cases. To summarize, CDV can be regarded as *constrained-random exhaustive testing with coverage metrics*.

The rigor and correctness of this procedure significantly depends on the human factor. If coverage goals are not thoroughly planned, there may be parts of the design which are not tested. Feature coverage gets collected when conditions in the manually-programmed, thus error-prone, verification environment are matched. The weakest links in CDV are manually coded monitor units which are supposed to implement protocol checks. A protocol checker which does not trigger in a proper moment, yields unreliable feature coverage results. An incorrectly implemented correctness conditions in a protocol checker lead to trivial condition satisfaction or false alarms.

Over the years, CDV best practice was unified into verification methodologies such as *e* Reuse Methodology (eRM) [IJ13], originally created for *e* programming language [HMN01], Open Verification Methodology [Gla09] and finally UVM [IEE17] which evolved into an IEEE standard. UVM methodology, widely used in digital design verification, is developed and updated by a standardization organization known as Accellera Systems Initiative [1].

Apart from digital hardware, Coverage Driven Verification is a prominent approach in the context of CPS [KDJ+16]. Simulink Design Verifier™ is a tool developed by MathWorks™ that performs formal analysis to generate test-cases for *model coverage objectives*. It also allows detection of dead logic, integer overflows, array out-of-bound errors, division by zero and violation of design requirements. Complementary tools that are also based on coverage metrics and used for testing and validation of Simulink™ models are developed by Reactis™. Another prominent tool for testing CPS is QTronic's TestWeaver™ which allows for tolerance analysis, failure analysis, safety and robustness analysis of control functions.

[1] http://accellera.org/

1.2.2 Falsification Testing

Control systems for CPS are typically developed using model-based development paradigm. Models can become very complicated since they are built from many components which implement continuous control and finite state machines. Such models might be a direct and understandable way to represent the system, however they are notoriously hard to translate to a form which admits testing. They involve continuous behaviors and non-linear dynamics which renders full-scale verification untractable. It is very hard to exercise the necessary corner cases and guarantee correctness due to the infinite space of behaviors. Hence, CDV method is not directly applicable.

A proven approach to verification of such systems with non-linear dynamics is the *falsification testing*. Instead of exhaustively verifying all the possible scenarios, falsification testing usually deploys an iterative process where in each iteration the inputs to the system are optimized to steer the system to falsify a specification [DJKM15]. In order to find the optimal inputs, falsification method leverages *robustness degree*, which is a sensible measure of how far is the trace from violating the requirement. In other words, falsification testing performs worst-case analysis by selecting the outputs which minimize the robustness degree of a trace produced by a CPS model. The iterative procedure stops in either of the following cases: (1) when a violating trace is found or (2) a fixed point is reached and a trace which is nearest to violating the requirement is found. The design team obtains a useful feedback about the quality of the model: in the former case the model must be corrected, in the latter the model could be optimized to achieve greater robustness w.r.t. the requirement.

The falsification method was initially developed to enable proving the absence of unsafe states, for hybrid systems where reachability analysis did not provide satisfactory results [EKK05, PKV07]. Techniques such as Rapidly exploring Random Trees were leveraged to obtain test inputs to search for violating trajectory [EKK05], as well as model checking and motion planning [PKV09].

Two most prominent tools for falsification of real-time temporal logic specifications are S-TaLiRo [ALFS11a] and Breach [Don10]. S-TaLiRo is implemented as software toolbox for Matlab, has proved itself useful for falsification testing of automotive control applications [FSUY12], Unmanned Systems Autonomy Services (UxAS) [THD+18] and an artificial pancreas control system [SKC+17]. Breach [Don10], which is originally created to perform verification and parameter synthesis [DJDS15] of hybrid systems, can also perform falsification testing [JDDS15, DDS17]. An efficient algorithm for calculating robustness degree for STL was integrated into Breach [DFM13], which allows performing robustness-guided falsification. Contrary to S-TaLiRo, Breach allows more flexibility w.r.t. the property which we want to falsify, since it supports parametric STL to specify behavior patterns.

1.3 Runtime Verification

Runtime Verification (RV) [FHR13] is a verification method where a single execution trace is verified against a formalized requirement. Instead of statically checking all the possible traces which can be generated from an abstract model, runtime monitors observe the execution trace of a *real* device and report violations. RV is a lightweight formal method which, in contrast to

model checking, provides a scalable verification solution [Mal16] for both software and hardware systems. In general, the access to internal structure of the SUT is not needed, thus we can monitor traces regardless of the source. RV can also be seen as an additional layer to simulation-based testing, where a simulation trace is checked w.r.t. a requirement expressed in a formal specification language [Mal16].

Necessary ingredients for RV are an SUT to be verified and a set of formal requirements. The requirements are expressed in a formal language (such as temporal logic, regular expressions or finite state machines.) and translated to a decision procedure - a monitor. The monitor consumes events produced by the system and produces satisfaction verdicts. System can be instrumented to provide events for the monitors which is a less preferred flavor of RV due to runtime overhead introduced. Monitoring is performed either online, during the actual execution of the system, or offline on the recorded execution trace. System can leverage online verdicts by immediately detecting unsafe conditions and restore to a safe state [LW06, FMFR11, BKKW15]. This method is known as *runtime enforcement*.

As stated in Section 1.2.2, the notion of *robustness degree* allows us to express not only binary satisfaction verdicts but rather to say "how robustly" is the specification satisfied. We can leverage robustness degree and perform *robustness analysis* of violating traces to analyze the severity of the violation. A number of runtime verification methods define quantitative semantics for specification languages in order to provide more verbose verdicts [FP09, DM10, DFM13, ALFS11b, Don10], which can be directly used to perform robustness analysis [DFM13].

RV has been proven successful in a variety of application domains. RV of software with code instrumentation using AspectJ [KHH+01] is implemented in Java-MOP [CR05], a Monitoring Oriented Programming Environment for Java. Similar approach, which was applied in order to verify the NASA Ames planetary rover K9, is implemented in Java PathExplorer [HR04]. Recently an open source runtime library for monitoring formal safety properties on Android was released [DFM+15]. The Analog Monitoring Tool (AMT) [NM07] implements both an offline algorithm and an incremental algorithm for evaluating satisfaction of requirements expressed in STL. AMT was used to verify DDR2 memory physical layer requirements [JKN10] as well as correctness of an automotive sensor interface [NN14a]. Monitoring an Autonomous Research Vehicle without code instrumentation [KCDK15], was able to detect safety violations in real-time. RV was also successfully applied to component-based systems in [FJN+11, BBCT14].

1.3.1 Assertion-Based Verification

In an industrial context of hardware verification, RV methods are usually renamed as Assertion-Based Verification (ABV). In this setting, ABV [Yeu04] is frequently used to simplify, complement, or increase confidence in CDV-based verification environments. ABV relies on assertions, which are invariant rules specified in an assertion language and checked throughout the execution. It is a very efficient approach - writing an assertion template in a single line of code can replace an entire verification environment block.

The most notable assertion languages are Property Specification Language (PSL) [PSL10, EF06], which is frequently used in VHDL environment, and System Verilog Assertions (SVA) [SVs13,

VKP15] which gained popularity with the adoption of System Verilog. Both of these languages are based on Linear Temporal Logic (LTL), where each time instance has a unique successor [Pnu77]. In order to achieve expressiveness of ω-regular languages PSL supports regular expressions and the suffix implication operator [EF18]. The latter allows to define a desired temporal pattern to be satisfied after the trigger condition, described by regular expression, is successfully matched. In order to improve succinctness of the language PSL introduces syntactic sugar for event counting operators. It also supports features such as first match, hardware clocks and resets, which are introduced to tackle typical requirements in hardware verification.

Using ABV for hardware verification offers practical merits. Assertions can be specified either in the design or in the verification environment and can relate to specific signals deep in the design logic. Once an assertion is violated, user is able to trace the failure directly to its root in the design. The assertions can also be synthesized into logic which is included in the silicon, therefore allowing to find bugs even in the post-silicon phase. At the moment all major vendors support ABV in their verification solutions: Mentor^{TM} provides PSL and SVA support in their latest Questa simulator. Both Cadence^{TM} and Synopsys^{TM} offer sets of predefined assertion libraries (Assertion IPs) for verifying specific protocols.

To summarize, ABV, known as RV in the scientific community, offers several benefits compared to other verification approaches:

1. it is formal, yet scalable method,
2. it blends naturally into simulation-based verification landscape,
3. it can be reused on different abstraction levels of an SUT,
4. it can verify the system both during design time and during the execution,
5. it can be used to perform robustness analysis.

1.4 Motivation

The motivation for our work comes from several practical and theoretical challenges.

Semiconductor designs in automotive are typical example of safety-criticalCPS which are notoriously hard to verify. To overcome performance issues with current simulation-based methods, the industry shifts towards hardware-accelerated testbenches. We aim to improve rigor and automation of verification methods which rely on hardware acceleration.

Development of autonomous vehicles, which operate in unpredictable environment and use artificial intelligence, raised new verification challenges. We believe shifting paradigm to monitoring, rather than applying exhaustive methods, is the appropriate response to these challenges. We have seen that assessing *robustness* of CPS requires measuring how far are the observed behaviors from the specification requirements. As a consequence, we need to equip our specification languages with quantitative semantics.

Having an understandable and precise robustness measure opens the door to new applications such as *falsification testing* and *specification mining*. Improved quantitative semantics would

enable these applications, which depend on robustness degree, to converge faster and produce more reliable results.

The challenges described require real-time correctness and robustness information and demand an online algorithm for computing correctness and robustness of a trace w.r.t. the specification. Monitoring in real-time implies that the verdict delay should be minimal, predictable and fixed.

1.4.1 Emerging Verification Challenges

Latest advances in development of autonomous vehicles, which primarily include the application of machine learning, impose new verification challenges. The verification challenge for autonomous vehicles comes from the fact that we have a new level of complexity caused by the environment. The number of scenarios to check in order to ensure safety is practically infinite. Even if we abstract away the human factor, security challenges and cultural settings of an environment, it is still impossible to predict all the scenarios occurring in the physical surroundings of an autonomous car at design time.

Neural network based systems are not built in top-down fashion by deriving implementation from high-level requirements, but rather by tweaking the parameters of neuron transfer functions by means of extensive *training*. A training set is nothing more than a very large set of labeled examples. Once network is trained, it is hard to distinguish specific features withing a large neural network. Thus, the modular verification approach is not applicable in this case.

Significant issues are exposed in the evaluation of classifiers based on machine learning. It has been shown that a small distortion on the picture, which can be generated systematically [GPAM+14], can cause a misclassification, thus rendering such classifiers prone to adversarial attacks. Recent research efforts [HKWW17, KBD+17, DDS17, TKID18] provide novel methods to verify controllers based on deep-neural networks. However, those solutions suffer from scalability issues and lack generality.

Data-driven controllers are self-adaptable during lifetime [KW17], thus able to learn new behavior patterns during operation. They may exhibit behavior which cannot be anticipated prior to their deployment in real conditions. This undermines the foundations of verification, where quality of testing is measured by the coverage of the relevant test space. If all the scenarios cannot be predicted upfront in the design process, verification at design time cannot provide firm guarantees. On the other hand, safety requirements are stable by their definition. For this reason, we expect that the focus will naturally shift from conservative, design-time approach (verification) to a more liberal, runtime approach (monitoring). Monitoring is a prerequisite for *safety enforcement*, a method where monitors not only recognize an illegal state but also influence the system to restore to safety.

1.4.2 Hardware-Accelerated Verification

RTL simulators are an understandable and well-established tool in hardware verification. However, simulation of large and mixed signal designs represents a major bottleneck in pre-silicon

verification. Development teams leverage hardware platforms to achieve better performance. Two most prominent methods are the emulation-based approach and Hardware-in-the-Loop.

Emulation. Cutting edge chip designs today may exceed more than 100 million gates. Due to limits in processor architecture and memory organization, designs of this size degrade the simulation speed due to frequent cache misses and swapping memory pages. This problem can be handled to some extent by deploying more powerful servers to parallelize simulations. However, it is often required to verify the embedded software (e.g. firmware) which runs on top of hardware. This new level of complexity renders simulation infeasible for scenarios of realistic duration. It can take weeks to simulate a single second of software executing on top of design simulation. This effectively means that classical simulation can be used only for short fragments of realistic scenarios, thus unable to expose potential critical bugs.

The solution to this problem naturally lies in the use of hardware acceleration. Instead of simulation, the design can be ported to hardware platform where it would execute almost at the same speed as the real silicon, thus reducing simulation overhead. First such technique was FPGA prototyping [CSM91]. Since FPGAs are constructed with a specific architecture in mind which aims for generality, the physical size of the design mapped to an FPGA increases significantly. A single FPGA usually does not suffice, which further increases the effort invested in design mapping. Finally, such prototypes have very limited debugging capabilities.

In order to address problems with limited design observability, firmware verification and significant mapping effort to an FPGA prototype, vendors promote *hardware emulation* approach. They provide a hardware platform with a proprietary architecture, created to enable hardware acceleration. Similar to FPGA prototyping, the emulator architecture is used to map the design and avoid costly RTL simulation. The advantages of this approach are several. First of all, one can emulate larger designs than by FPGA prototyping. The emulator also provides design accuracy which enables engineers to easily track the bugs down the design hierarchy. Most important, they provide end-to-end solution for verifying embedded software and hardware together.

It is also possible to interface the emulator to modern test benches in System Verilog UVM. In this setting, it is necessary to port the timed functionalities from the test bench to the HDL code which will be run in the emulator [vdSY15]. The remaining part of the test bench is ignorant of the physical clock, and communicates with timed portion of the test bench and emulated design on transaction level. When optimized properly, a several orders of magnitude speed-up can be achieved [vdSY15].

Hardware-in-the-Loop Modern SoC often include both analog components which interact with the environment and digital blocks which typically implement control. Due to the mixed nature they impose significant computational requirements to simulation. An approach to overcoming this problem is to replace the complex parts, usually the analog, with a mathematical approximation implemented on an FPGA. This way, high-level testing can be performed early on, before the analog design is ready.

Test benches are also implemented in hardware, to allow greater simulation speed and longer scenarios. In this setting, physical stimulus is replaced with a modeled behavior. The SUT is placed in central position, whereas interaction with physical environment and computationally demanding design are approximated. This technique is called Hardware-in-the-Loop (HiL) [RSM+99]. HiL testing is performed in laboratory conditions where engineers often manually interpret test results shown in signal waveforms, which is an error-prone process.

1.4.3 An Abundance of Quantitative Metrics

In a CPS context, quantitative verdicts are used to measure the satisfaction robustness of a CPS run with respect to a specification [AHF+14, DZS+15, ALFS11b]. Most measures were associated to STL specifications [FP09, DM10, DFM13, DZS+15, AHF+14, ALFS11b, Don10, RDS+15, DDD+15] and to its variants or extensions [BVSF13, AT15]. The general approach was to first develop the measure, based on the STL syntax, and then use the measure for monitoring. This resulted in a plethora of measures, assessing space [FP09, DM10], time [DM10], and averaging [AT15] robustness, for STL, and a space-robustness measure for Signal Regular Expressions (SRE) in [BFMU17]. A palette of distance metrics represents a value by itself, since there are many different problems that can benefit from a variety of metrics. We can identify proper continuous-time and discrete-time properties and their robustness, perform worst-case analysis for safety critical scenarios or average case analysis.

Such abundance of measures obscures the main problem of quantifying the distance from a specification. Every new distance specifies a dedicated algorithm. Syntactic-based methods aim to calculate robustness inductively, on the structure of the formula. For this reason, they provide inaccurate robustness degree for specific types of formulas. The problem of defining a distance among two behaviors is not directly related to the interpretation of robustness on the level of temporal operators. Natural approach to measuring robustness would work directly on the trace and the set.

We believe in separation of concerns: distance metrics and specification representation should be regarded as two separate ingredients of robustness analysis. Instead of having an abundance of metrics with separate algorithms, each of them with specific requirements, we aim towards a generic and unified approach to robustness.

1.4.4 Beyond Runtime Verification

As previously stated, the binary true/false verdicts may not be sufficient for real-valued behaviors. The classical satisfaction relation can be replaced by a more quantitative *robustness degree* [FP09, DM10, DFM13] of a behavior with respect to a temporal specification. The robustness degree provides a finer measure of how far is the behavior from satisfying or violating the specification. It is calculated as the distance of a single trace from a set of satisfying or violating traces, according to a specification. Quantitative monitoring has opened the door to interesting applications, primarily *falsification testing* and *specification mining*.

Model based design techniques play a crucial role in development of industrial analog designs. Such devices are essentially hybrid dynamical systems which exhibit infinite number of possible

behaviors, thus hard to cover by simulation. Falsification-based testing aims to guarantee correctness by performing robustness-guided reachability analysis on the set of behaviors. This method uses system dynamics to calculate the inputs which minimize the trace robustness degree w.r.t. the specification. The output of the falsification procedure is the information whether a model can reach the set of illegal traces. Two most prominent tools for falsification testing are Breach [Don10] and S-TaLiRo [ALFS11a].

For many systems, it is useful to know not only if the a single requirement can be falsified, but also to define a set of properties which are satisfied by the system. Such problem of *specification mining*, can be reduced to optimization problem [HDF18, YHF12] by leveraging quantitative verdicts. Input for this exploration technique is a model of a system, a parameterized specification and an initial simulation point. Simulation of the system produces a trace which is then evaluated for robustness w.r.t. the parameterized specification. The robustness degree influences the generation of initial conditions, inputs, and specification parameters for next iteration of the mining algorithm. Iterative parameter refinement ends when the procedure reaches a fixed point. Confidence in results depends on the precision of robustness degree which is used to guide parameter refinement. We aim to improve the current state-of-the-art, which relies on robustness *estimate* for Metric Temporal Logic [FP09] specifications.

The common foundation of these applications are the *quantitative verdicts*. Spatial, temporal and spatio-temporal quantitative semantics describe similarity of two signals. Usability of these verdicts depends on how well they reflect discrepancies among the actual signals. For a signal which deviates from a specification both in space and time, a meaningful verdict is the one which correctly balances the influence of both domains.

We believe that the applications would benefit from using quantitative verdicts based on a multi-purpose, spatio-temporal robustness degree, computable in an online fashion either in software or hardware. The falsification-based testing would capture the reachable set more precisely and the parameter refinement performed by a specification mining algorithm would converge faster.

1.4.5 Fostering RV Methods in the Industry

In the last decades, formal methods have been adopted to certain extent in the industrial practice. Theorem proving, model checking and runtime verification are the most prominent formal methods used to tackle challenges in industrial practice. As previously mentioned, full-blown static verification methods suffer from scalability issues.

We believe that fast hardware monitors, reusable in different levels of product development, with formal, non-ambiguous semantics can improve verification time and rigor, thus significantly shorten time-to market. For the verification challenges of future, such as autonomous vehicles which exhibit safety critical scenarios, online runtime verification seems like a promising approach. The purpose of this research is not only to bring new methods, but also to prove runtime monitor applicability in a real-world setting, with the hope to promote lightweight formal methods in the industrial community.

1.5 Contributions

Our main line of research follows the goal of creating theoretically well-founded framework for assessing correctness and robustness of CPS. To reach this goal, our research introduced the following scientific and practical contributions.

1.5.1 Hardware Correctness Monitors

We provide an efficient procedure for synthesizing correctness monitors for safety properties expressed in Signal Temporal Logic [JBG+15]. The monitors naturally integrate with hardware-accelerated verification approach. Such monitors add no computation overhead to the SUT, have predictable and minimal verdict delay and admit direct implementation in hardware. Thus, our monitors can monitor systems at runtime, orders of magnitude faster than the simulation. Hardware monitors, synthetized from formal specifications, increase confidence in HiL methods which frequently include manually crafted components. The monitor resource consumption is independent of the length of the execution trace, which is important because hardware-accelerated methods are used to exercise long scenarios.

The choice of STL allows monitoring pure digital, analog or mixed-signal requirements. Our procedure supports expressive specifications in bounded and unbounded, past and future Signal Temporal Logic. The monitors have a generic hardware architecture which is implemented on a "bare metal" FPGA and does not require a full-blown CPU to execute embedded software. Consequently, speed of the monitor is only limited by physical characteristics of a hardware platform. Our monitors minimize the verdict delay by adopting a compositional approach which splits the computation of satisfaction verdict into smaller instances evaluated in parallel. Fixed verdict latency allows monitors to produce correctness verdicts in real-time. We evaluate resource consumption of our monitors w.r.t. memory and arithmetic units available on FPGA platform.

1.5.2 Monitors with Recovery

In addition to correctness monitoring, our work is also focused on improving usability of such hardware monitors in an industrial setting. For exercising long scenarios in post-silicon verification or HiL techniques, the monitors need to be able to continue monitoring long streams of data, after one or more errors occur. We provide a generic procedure for an error recovery which is applicable to a class of asynchronous serial protocols [Int16a, Axe07, (R208].

To enable diagnostic features of our monitors, we also specify and implement an interface for logging and recognizing types of errors [SJN+17].

1.5.3 Unification of RV Approaches for CPS

The proliferation of CPS trace robustness measures reflects the various needs of different applications, and the limitations of STL-based measure definitions. We propose Algebraic Runtime Verification ARV [JBGN18], as a general, semantic approach to measuring a run, with respect to a specification automaton. Algebraic structures, such as *monoids* and *semirings*, guarantee

certain axioms to be satisfied over a set and associated generic binary operations: $\oplus$-*addition* and $\otimes$-*multiplication*. In the context of CPS traces and specifications, we designate these two operations as *choice* and *concatenation*, respectfully.

A CPS trace is defined as a sequence of valuations of variables over a specific time period. We identify that the structure of a CPS trace conforms to axoims of a monoid, with respect to concatenation of new valuations to the existing trace, as well as the number of variables. Following the same idea, we identify that our specifications admit a semiring structure with respect to choice of the "closest" trace and concatenation of new valuations to the trace. We exploit this algebraic nature of traces and specifications to define a measure in an incremental and generic way. By allowing to associate arbitrary real-valued functions to choice and concatenation, we are able to adapt our measures to a desired application.

ARV simplifies and unifies qualitative and quantitative approaches to the RV of CPS. It exposes the core of the RV problem, and its underlying structure, allowing us thus to develop an abstract monitoring algorithm, that we instantiate with different specification languages, as well as qualitative and quantitative semantics. The use of automata, admits semantic and precise monitoring procedures, that are invariant to different syntactic representations of the same specification. In addition, it opens the door for flexible code generation which can directly translate the abstract monitoring procedure into real-time monitors, implemented in either software, embedded software, or hardware.

Input to ARV is a regular specification formalism that admits an effective translation into symbolic automata. We then decorate the symbolic acceptor with weights and associate a semiring to the resulting Weighted Symbolic Automata (WSA). The weights in the WSA measure the distance at a given point in time between the current trace observation and the constraints induced by the specification. ARV defines the monitoring procedure over the WSA as a dynamic programming algorithm that computes the shortest path induced by an input trace. By instantiating the semiring, it provides various qualitative and quantitative semantics to the monitoring procedure without changing the underlying algorithms. We study the basic properties of our generic ARV framework and evaluate it with two case studies from the automotive domain.

We give proofs for precision of ARV-based robustness degree and compare it to syntactic-based robustness. The automata-based approach is capable of capturing contradictions and tautologies and it guarantees that two semantically-equivalent but syntactically-different specifications have the same robustness degree with respect to all behaviors.

1.5.4 Novel Quantitative Semantics for STL

We define a distance metric for robustness analysis, which achieves appropriate balance between spatial and temporal discrepancies, without being tailored for a specific problem. Our approach [JBGN16, JBG$^+$18] introduces a novel quantitative semantics for STL, based on Weighted Edit Distance (WED), which measures the behavior mismatches of discrete-time digitized behaviors in both *space* and *time*. We believe WED is the appropriate metric for measuring similarity between CPS behaviors due to the following desirable characteristics:

1. It is cumulative, hence it can differentiate cases of single and multiple deviations from a reference behavior;
2. It combines spatial and temporal aspects, which are both valuable when reasoning about CPS behaviors; and
3. It is defined in discrete time, which is a necessary condition for the applications that we consider.

In the spirit of ARV, we develop an efficient online automata-based algorithm for computing the robustness degree between a behavior and an STL formula in real-time. Since we highly value the integration of our monitors with hardware accelerated verification, the algorithm for calculating WED-based robustness degree admits scalable, low-overhead hardware implementation. In the former case, the implemented procedure can be connected to the simulation engine of the CPS design and used to monitor its correctness and quality. In the latter case, the resulting implementation can be deployed on a Field Programmable Gate Array (FPGA) and used to monitor real systems or design emulations. We implement the algorithms in Verilog and deploy them on FPGAs, thus providing an effective bridge from design-time (e.g. Simulink) to deployment-time (e.g. autonomous vehicles) quantitative monitoring. Using WED to reason about the similarity between behaviors and specifications is novel in the context of STL robustness.

1.5.5 Monitoring for Automotive Applications

We perform thorough evaluation of our monitors on benchmarks from the automotive domain [JBGN16]. In addition, we conduct a full scale automotive case study [SJN+17], for Single Edge Nibble Transmission SENT protocol, which is used on the magnetic sensor interface, typically found in electronic steering applications in automotive. The electrical and timing requirements are formalized in Timed Regular Expressions and Signal Temporal Logic [SJN+17]. We give valuable conclusions on monitor applicability in an industrial setting and compare two different specification formalisms, namely TRE and STL. We demonstrate the synergy between formal methods and industrial practice in a real-world setting.

1.6 Structure of the Thesis

Contributions of the thesis are both theoretical and practical. The classification of the contributions and the structure of the thesis are presented in 1.1.

The thesis is structured as follows:

- Chapter 2 gives an overview of state-of-the-art in correctness and robustness monitoring based on formal specifications expressed in temporal logic.
- Chapter 3 provides theoretical definitions of signals which are input for our monitors as well as specification languages we will use to specify our requirements.
- Chapter 4 defines automata-based specification representations – Weighted Symbolic Automata and Temporal Testers.

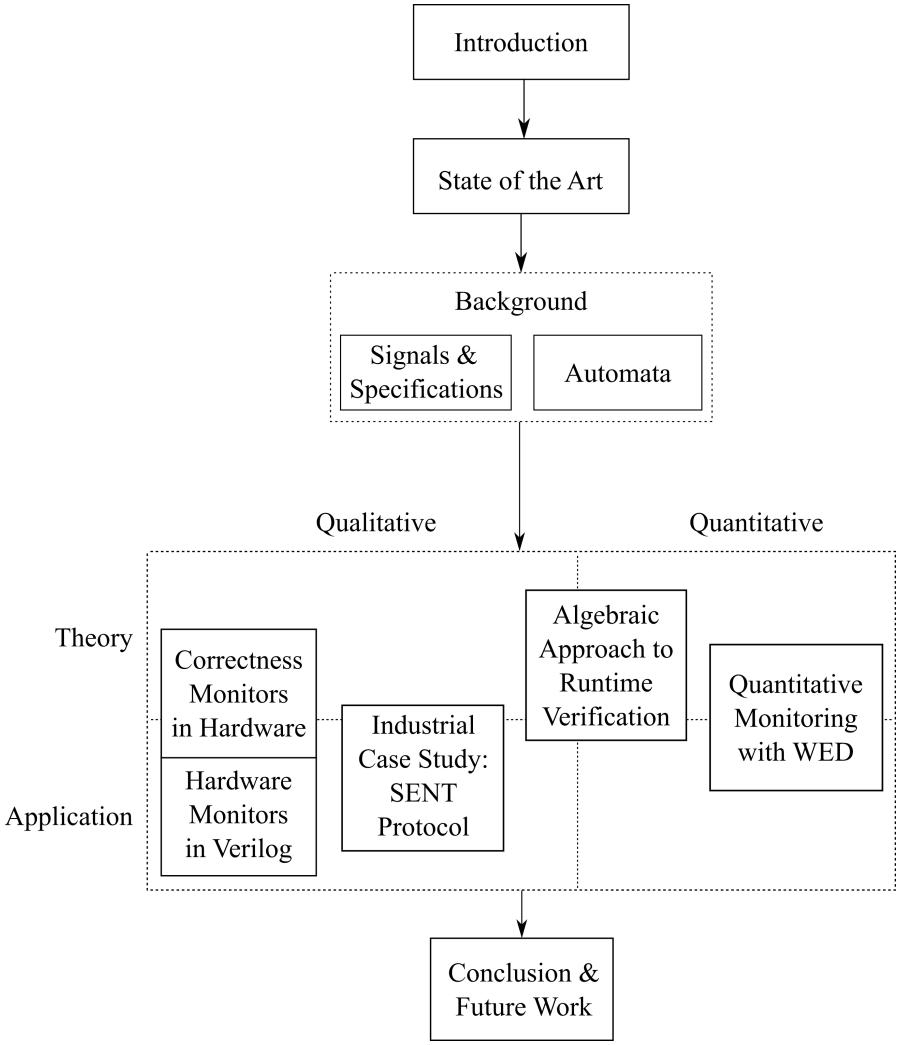


Figure 1.1: Structure of the thesis: theory and application, qualitatitive and quantitative monitoring

- Chapter 5 is focusing on presenting STL-based correctness monitors implementable in hardware and their synthesis from STL requirements .
- Chapter 6 discusses quantitative runtime verification and exposes a unified, algebraic approach to the problem of correctness and robustness monitoring.
- Chapter 7 provides an algorithm to calculate *robustness degree* based on Weighted Edit Distance, a spatio-temporal distance metric.
- In Chapter 8 we present a thorough case study of application of hardware runtime monitors on a communication protocol widely used in the automotive industry.
- Chapter 9 gives a summary of the research and an outlook on the future work.

- In Appendix A we provide proofs for the theorems stated in this thesis.
- We conclude with Appendix B which provide source code of hardware monitor templates for basic past-time STL operators.

CHAPTER 2

State of the Art

2.1 Related Work on Correctness Monitors

In the last decade assertion based hardware monitoring has received an increasing attention. Generating hardware monitors from Property Specification Language (PSL) has been proposed by several research groups and it has been implemented first in the tools FoCs [DGG+05] developed by IBM and MBAC [BZ05, BZ06, BZ08] developed by Zilic and Boulé. On the same line of research is the work of Borrione et. al. in [BLMA+05] and Backasch et. al. in [BHW+13]. In order to allow rich specifications we support past and future, timed and untimed temporal logic. In contrast to our work, all of them focus on untimed digital specifications. FoCs generates monitors for SystemC [sys] simulations. MBAC adopts an automata-oriented approach which conceptually differs from our transducer-based compositional construction.

Synthesizing hardware from formal specifications was successfully applied to obtain an arbiter for ARM's Advanced Microcontroller Bus Architecture (AMBA) Advanced High-performance Bus (AHB) bus for specifications given in PSL [BGJ+07]. The formal semantics of PSL does not include an explicit notion of time. On the other hand, formalisms such as STL and TRE allow precise definition of desired time intervals for the requirements.

Finkbeiner et al. in [FK09] present a technique to synthesize monitor circuits from LTL formulas with bounded and unbounded future operators. They allow only past-time specifications and evaluate their approach only with formulas with the lower time bound equal to zero.

Claessen et al. [CES13] propose some efficient techniques to synthesize a LTL safety and liveness property checkers as circuits with sequential elements rather than using the classical automata-based approach with transitions relations. The authors focus more on model checking of hardware system design rather than runtime monitoring of analog and mixed-signal (AMS) systems. Finally, they do not report any experiments on real hardware.

Another very popular extension of correctness monitors is the *system health* monitoring. In this context, a monitor is not only reporting violations, but rather recognizing trends in a behavior

of a system and estimating the system *health* in every time step. Such monitors, which rely on Bayesian network to estimate system health, are implemented in [MRS17]. Instead of giving *estimations* our correctness and robustness monitors rely on deterministic data and provide non-probabilistic, definitive verdicts. We achieve generic and scalable monitor architecture with minimum overhead, thus leveraging hardware speed.

Reinbacher et. al. propose in [RFB14, RRS14] hardware monitors from different fragments of Metric Temporal Logic. In [RFB14], the authors tackle only the past fragment of MTL using a transducer-based approach similar to ours. In contrast to relative clocks (counters) that we adopt to record events, the authors use an approach in which absolute time stamps are memorized. Hence, the resources needed for implementing their monitors depend on the duration of the emulation runtime. The authors develop a sophisticated architecture that targets reconfigurability of monitors. However, it also introduces an overhead that requires their monitors to operate at a considerable higher frequency than the SUT. This imposes additional constraints on the maximum speed of the SUT that such monitors can observe. In our approach, the monitor and the SUT run at the same clock frequency. Finally, the authors evaluate their approach using pre-collected data.

In [RRS14] the authors address the future fragment of MTL. They take a radically different approach to ours motivated by the problem of embedded system health estimation. They adopt a three-valued interpretation of the logic and produce a "maybe" output delaying a definite verdict until the formula can be really evaluated. This approach is suitable for estimating system health using a Bayesian network on top of the observers. On the other hand, we are motivated by the verification of AMS chip design along its physical interface (time, voltage, signal dependencies in time) and we choose to follow a different approach. We treat formulas with bounded future temporal operators, rewriting them using past temporal operators which gives us more uniform and simple approach in terms of formula evaluation and system architecture. Similarly to their previous work the authors evaluate their framework only on recorded data.

UPPAAL [BLL+95, LPY97] is a well established tool for the verification of real-time systems which can be modeled with timed automata. This tool provides a description language for modelling, a simulator, and a model checker. In contrast, our goal is to create a standalone monitor in order to verify a discrete time system during runtime. Our monitors are ignorant of the model of the system. In addition, since we are using a formally proven translation from formal specifications in TRE and STL to automata, our monitors are correct by construction.

Orthogonal to monitoring, an SMT-based approach to design and analysis of CPS was described in [CSRB13]. In that work, the authors show how to reduce several important verification and synthesis problems of CPS to exists-forall quantified propositional combinations of constraints which is then handled by a solver.

2.1.1 Relevant Case Studies

We are aware of several case studies on monitoring temporal logic specifications - the automotive bus standard [NN14b], the DDR2 memory interface [JKN10], typical automotive functional requirements [FSUY12]. All of these works focus on offline monitoring and continuous-time semantics, which covers STL and does not consider specifications based on regular expressions,

and omit monitor recovery aspects after capturing a violation. Although the authors in [SNBG16] runtime-verify a subset of requirements of the PSI5, the protocol uses different encoding scheme than the SENT; their emphasis is on how to apply runtime verification, and they by and large avoid technical details. In contrast, we compare two formalisms and implementations, to increase integration readiness level for the monitor itself and eliminate the "single source of truth" aspect from the monitoring system.

In [FMNU15] the authors also use TREs with events to evaluate the performance of a controller and sensor implementation. Orthogonally to our work, they define measurement specifications over timed patterns.

2.2 Related Work on Robustness Monitors

In this section we present the relevant research on *robustness degree* monitoring. We first discuss different distance metrics and robustness degree definitions, and then report on approaches based on *edit distance*.

2.2.1 Quantitative Semantics for Temporal Logic

In the recent past, property-based runtime monitoring of CPS centered around Signal Temporal Logic (STL) [MN13] and its variants have received considerable attention [FP09, DM10, DFM13, AT15, NN14a, BDSV14, BBS14a]. STL is a formal specification language for describing properties of continuous and hybrid behaviors. In its original form, STL allows to distinguish correct from incorrect behaviors.

The quantitative semantics for temporal logics were first proposed in [RBFS08, FP09], with the focus on the *spatial* similarity of behaviors, given by their point-wise comparison. The spatial quantitative semantics is sensitive to phase shifts and temporal inaccuracies in behaviors – a small temporal shift in the behavior may result in a large robustness degree change. This problem was addressed in [DM10], in which Signal Temporal Logic with spatial quantitative semantics is extended with time robustness. In [AT15], the authors propose another approach of combining space and time robustness, by extending STL with *averaged* temporal operators. Another approach to determining robustness of hybrid systems using self-validated arithmetics is shown in [FSIG09]. Monitoring of different quantitative semantics is implemented in tools such as S-TaLiRo [ALFS11b] and Breach [Don10].

The problem of online monitoring robustness was studied more recently in [DHF14, DDG$^+$17]. The authors of [DHF14] propose an online monitoring approach that uses a predictor, which requires for the future fragment of the logic the access to a model of the system. This is in contrast to our black-box view of monitoring. In [DDG$^+$17], the authors propose an interval-based approach of online evaluation that allows estimating the minimum and the maximum robustness with respect to both the observed prefix and unobserved suffix of the trace. Our work does not provide such estimation about the future. Instead, our robustness value at every point in time gives the distance of the observed prefix from the satisfaction/violation of the specification.

The authors of [DSS+05a] propose an online monitoring procedure where semantics describe the relation between input and output streams. In [BBLN17] the authors used an algebraic approach to define the robustness for a spatio-temporal extension of STL considering only semantics based on the MinMax semiring. Their approach works at the syntax level of the specification, resulting to be less precise than the one proposed here.

The recent results on using Skorokhod metric [Sko56] to compute the distance between piecewise-linear or piecewise-constant continuous behaviors [DMP15] partially inspired our work. Skorokhod metric quantifies both space and time mismatches between continuous behaviors by allowing application of time distortions in behaviors in order to minimize their point-wise distance. The distortion of the timeline is achieved by applying a retiming function - a continuous bijective strictly increasing function from time domain to time domain. Given a behavior $x(t)$, the resulting retimed behavior $r(x(t))$ preserves the values and their order but not the duration between two values. This information-preserving distance relies on continuous time and is not applicable to the discrete time domain – stretching and compressing the discrete time axis results inevitably in an information loss. Finally, the computation of the Skorokhod distance was extended to the flow-pipes in [MP16] and to the epsilon-tubes in [Dav09], where the authors consider computing the distance between hybrid (continuous and discrete-time) signals. We are not aware of any work addressing the problem of computing the Skorokhod distance between a behavior and a temporal specification.

Our work is also related with the notions of (ϵ, τ)-closeness in [AMF14] and (ϵ, τ)-similarity (requires the retiming to be order-preserving) introduced in [Que13] to compare two mixed-analog signals and in conformance testing [AMF14]. The parameters τ and ϵ are used to specify how much it is allowed to wiggle in both time and space in order to transform one trace into another. The main difference with this work is that our distance provides a cumulative measure, while the other notions try to find the max possible discrepancy.

A different kind of robustness called specification-centered robustness is defined in [BCG+11]. The authors aim to define a notion of robustness based on input and output sequences of reactive systems, without taking the hardware failures inside an SUT into consideration. Such robustness degree can be used to describe degradation of quality. Their specifications consist of a set of assumptions, related to the environment and the set of guarantees, related to the system.

2.2.2 Edit Distance

The Levenshtein (edit) distance [Lev66] has been extensively used in information theory, computer science and bioinformatics for many applications, including approximate string matching, spell checking and fuzzy string searching. Levenshtein automata [SM] were introduced to reason about the edit distance from a reference string. A Levenshtein automaton of degree n for a string w recognizes the set of all words whose edit distance from w is at most n. A dynamic programming procedure for computing the edit distance between a string and a regular language has been proposed in [Wag74]. The problem of computing the smallest edit distance between any pair of distinct strings in a regular language has been studied in [Kon07]. In contrast to our work, these

classical approaches to edit distance consider only operations with simple weights on unordered alphabets and are not applied to dynamic reactive behaviors.

The edit distance for weighted automata was studied in [Moh03], where the authors propose a procedure for computing the edit distance between weighted transducers. A space efficient algorithm for computing the edit distance between a string and a weighted automaton over a tropical semiring was developed in [AM09]. The resulting approach is generic and allows for instance to assign an arbitrary cost to each substitution pair. However, all substitution pairs must be enumerated by separate transitions. In contrast, we consider signals with naturally ordered alphabets as input strings and hence can efficiently handle substitution over large alphabets by treating allowed input values with symbolic constraints. In addition, we use the edit distance to define the semantics of a temporal specification formalism.

The weighted Hamming and edit distances between behaviors are also proposed in [SDC13], where the authors use it to develop procedures for reasoning about the Lipshitz-robustness of Mealy machines and string transducers. The notion of robustness is different from ours, and in contrast to our work it is not computed against a specification.

2.3 Related Work on Automata-Based RV

The theoretical and practical concerns regarding symbolic automata and transducers are studied in [DV17, DV15, VHL+12]. In our work, we use the theory developed for symbolic automata to develop our Algebraic Runtime Verification (ARV) framework.

An algebraic framework for the basic properties of the weighted automata is studied in [Moh03]. The shortest-distance problem in weighted automata is investigated in [Moh02]. It generalizes the classical Bellman-Ford algorithm [Bel58] to non-idempotent semirings. We are interested in a Hausdorff measure and hence restrict our attention to additively idempotent semirings. In constrast to fixed edge weights, we use Weighted Symbolic Automata with dynamic weights which depend on current trace valuation. Our ARV algorithm relies on *idempotent semiring* which allows us to prune the irrelevant candidates for shortest path. Thus, we are able to define recurrence relation and incrementally calculate shortest path for a given trace.

Qualitative monitoring for temporal specifications has been a well-studied problem in the runtime verification community. In [BLS11], the authors propose an automata-based approach to monitoring Linear Temporal Logic and its real-time extension TLTL with the three-valued semantics. Another automata-based approach to online monitoring of LTL for Java software instrumented with AspectJ is presented in [SB06]. The authors in [BGHS04] present a rule-based software monitoring framework for LTL and metric LTL. In the context of hardware, monitors synthesis from temporal specifications such as Property Specification Language, has been extensively studied in the last years, resulting in tools FoCs [DGG+05] and MBAC [BZ05]. Offline monitoring for STL with qualitative semantics was developed in [MN04, MN13]. In contrast to this work, which aim at developing optimal monitoring procedures for specific settings, our primary focus is to develop a framework that unifies RV of qualitative and quantitative specification.

An insteresting approach to automatic generation of observers from high-level specifications, which are translated to timed automata is described in [BBKT05]. In this work, the authors instrument the execution platform, to obtain traces to monitor.

A concept similar to the one presented in ARV, explores different interpretation of temporal logic operators [RBNG16]. In contrast to our work, these interpretations are applied directly on the syntax and semantics of the logic, with the aim to demonstrate a relation between temporal logic operators and convolution. We also mention the work on quantitative languages [CDH08] that is studied over infinite words and is complementary to our work.

CHAPTER 3

Signals and Specifications

We start laying theoretical foundations for our work by defining the inputs of our monitors: signals produced by the monitored system. Since we aim for hardware implementation of our monitors, we use digital signals. We explain the relation of a digital signal and an analog signal originating from the physical environment. We discuss the process of converting an analog signal to a digital signal and estimate the inaccuracy introduced in this process. Finally, we provide syntax and formal semantics of specification languages: Signal Temporal Logic and Timed Regular Expressions.

The expressiveness of our specification languages is illustrated with several functional requirements for Advanced High-performance Bus (AHB) Lite protocol, which is a de-facto standard for high performance bus in systems based on ARM CPUs. The full AMBA AHB protocol with multiple master devices has been used as a benchmark for several reactive synthesis tools [BJK14].

3.1 Signals

Let X be a finite set of variables defined over some domain $\mathbb{D}$. Then, a *signal* w is a function $w : \mathbb{T} \times X \to \mathbb{D}$, where $\mathbb{T}$ is the time domain[1] and $\mathbb{D}$ is the value domain. We distinguish between *analog*, *discrete* and *digital* signals. Analog signals have continuous value and time domains. The time domain of discrete signals is the set of integers, while digital signals have in addition their value domain restricted to a finite set.

3.1.1 Sampling and Quantization

Digital signals can be obtained by *sampling* and *quantization* of analog signals. The conversion from analog to digital is at the core of the signal processing field and is performed by an *analog-to-digital converter* (ADC). Sampling is the process of reducing the continuous time of analog

[1] We use $s(t)$ to denote the valuation vector of the variables in X at time t.

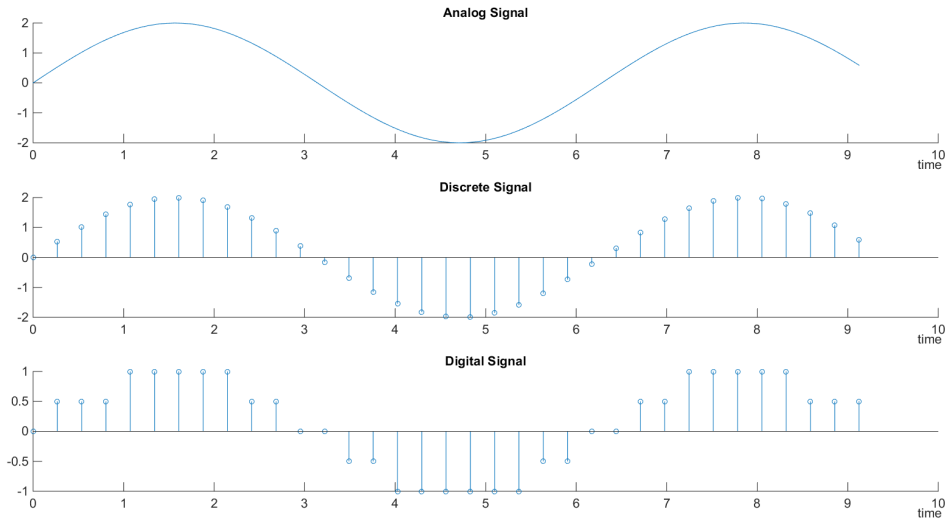


Figure 3.1: Analog, discrete and digital signal

signals to discrete time which results in obtaining a time-discrete signal. An ideal sampling function would periodically measure the value of the analog signal every T time units, where T denotes the *sampling interval*. Similarly, we define by f the *sampling frequency*, as the average number of measurements obtained by sampling in one second, where $f = 1/T$. Given an analog signal $w_a : \mathbb{R}_{\geq 0} \times X \to \mathbb{R}^n$ and a sampling interval T, applying the ideal sampling function to w_a results in a discrete signal $w_{disc} : \mathbb{N} \times X \to \mathbb{R}$ such that $w_{disc}(i, x) = w_a(iT, x)$ for all $i \geq 0$ and $x \in X$.

While sampling real-valued signals, it is impossible to maintain the arbitrary precision of its values which consequently must be restricted to a finite set. Quantization consists of converting real values to their discrete numerical approximations, which allows to map a discretized analog signal to digital domain. We consider the basic uniform quantization function with a *quantization step* Q which is defined as follows:

$$Q(r) = \mathsf{Q} \cdot \lfloor |r|/\mathsf{Q} + 0.5 \rfloor,$$

where $r \in \mathbb{R}$. We note that the quantization can be decomposed into two stages, *classification* and *reconstruction*. The classification function c maps the real input value into an integer index k, and the reconstruction function y converts k into the actual discrete approximation of the input. Hence, we have that $Q(r) = y(c(r))$ where

$$\begin{aligned} c(r) &= \lfloor |r|/\mathsf{Q} + 0.5 \rfloor \\ y(k) &= \mathsf{Q} \cdot k \end{aligned}$$

The decomposition of the quantization into two independent stages has a practical advantage – without loss of generality, we can from now on directly work with digital signals obtained after the classification stage with their value domain being a finite subset of $\mathbb{Z}$. In practical setting, such finite set is remaped to the subset of subset of $\mathbb{N}$ by an analog-to-digital converter. We also restrict ourselves to signals that have *finite-length* and hence are of the form $w_{dig} : [0, d) \times X \to [v_{min}, v_{max}]$, where $[0, d)$ and $[v_{min}, v_{max}]$ are intervals in $\mathbb{N}$, and X is now the set of variables defined over the domain $[v_{min}, v_{max}]$. We extend the signal notation $w(i, X)$ to denote the vector $\mathbb{D}^{|X|}$ of all variable values in X at time i. From now on, we refer to digital signals of finite length simply as signals and denote them by w.

Definition 1 (Discrete time signal)**.** *We define a* signal w *as a total function* $w : [0, d] \to \mathbb{B}^m \times \mathbb{D}^n$*, where* $[0, d] \subseteq \mathbb{N}$ *denotes the discrete time domain and* $\mathbb{D}$ *is a finite valued domain. We denote by* $|w| = \delta$ *the length of the signal* w*.*

In our work we use discrete signals and we interpret Signal Temporal Logic over discrete time. We justify this choice by our determination to build monitors in hardware, which belong and operate in discrete time. It is important to note that the monitoring in discrete time, can be successfully applied to continuous signals, provided that certain conditions are satisfied, as shown in [FP09].

3.2 Specification Languages

In our work we use two specification languages to formalize the requirements which we want to check with our runtime monitors. We use Signal Temporal Logic, interpreted over discrete time with past and future, timed and untimed, temporal operators. In addition, we use Timed Regular Expressions, also interpreted over discrete time.

3.2.1 Signal Temporal Logic

STL is a specification language which allows expressing real time requirements of mixed-signal behaviors. It is originally defined in dense time [MN04, MN13]. STL extends Metric Temporal Logic (MTL) [Koy90] with predicates over real-valued variables. The monitoring algorithms for STL were first developed in [MN04, MN13] and implemented in a tool [NM07], giving thus STL a practical purpose in the analysis of realistic CPS.

In this thesis, we use Signal Temporal Logic (STL) with both *past* and *future* operators in the context of runtime verification problem, in which the monitors are implemented on FPGAs. In order to accommodate the specification language to this particular domain of application, we interpret STL over discrete time and finite valued domains.

Let $P = \{p_1, \ldots, p_m\}$ be the set of boolean variables and $X = \{x_1, \ldots, x_n\}$ the set of variables defined over non-boolean domain $\mathbb{D}$. We denote by $\pi_e(w)$ the projection of w to $e \in P \cup X$. Given a signal w and its projection $\pi_p(w)$ to some $p \in P$, we say that $\pi_p(w)$ has (Δ, l)-variability if within every interval $[i, i + \Delta - 1]$, the value of p changes at most l times.

The syntax of a STL formula φ over $P \cup X$ is defined by the grammar

$$\varphi \quad := \quad p \mid x \sim u \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I \varphi_2 \mid \varphi_1 \,\mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a, b]$ or $[a, \infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the form $[a, a]$, we will use the notation $\{a\}$ instead.

The semantics of an STL formula with respect to a signal w is described via the satisfiability relation $(w, i) \models \varphi$, indicating that the signal w satisfies φ at the time index i, according to the following definition where $\mathbb{T} = [0, |w|]$.

$$\begin{array}{lcl}
(w,i) \models p & \leftrightarrow & \pi_p(w)[i] = \top \\
(w,i) \models x \sim u & \leftrightarrow & \pi_x(w)[i] \sim u \\
(w,i) \models \neg\varphi & \leftrightarrow & (w,i) \not\models \varphi \\
(w,i) \models \varphi_1 \vee \varphi_2 & \leftrightarrow & (w,i) \models \varphi_1 \text{ or } (w,i) \models \varphi_2 \\
(w,i) \models \varphi_1 \,\mathcal{U}_I \varphi_2 & \leftrightarrow & \exists j \in (i + I) \cap \mathbb{T} : (w,j) \models \varphi_2 \text{ and} \\
 & & \forall i < k < j, (w,k) \models \varphi_1 \\
(w,i) \models \varphi_1 \,\mathcal{S}_I \varphi_2 & \leftrightarrow & \exists j \in (i - I) \cap \mathbb{T} : (w,j) \models \varphi_2 \text{ and} \\
 & & \forall j < k < i, (w,k) \models \varphi_1
\end{array}$$

In case of truncated paths, future temporal operators may use either weak or strong semantics as defined in [EFH+03].

3.2.2 Derived STL Operators

From the basic definition of STL, we can derive the following standard operators.

$$\begin{array}{lcl}
\top & = & p \vee \neg p \\
\bot & = & \neg\top \\
\varphi_1 \wedge \varphi_2 & = & \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\Diamond_I \varphi & = & \top \,\mathcal{U}_I \varphi \\
\Box_I \varphi & = & \neg\Diamond_I \neg\varphi \\
\Diamond\!\!\!-_I \varphi & = & \top \,\mathcal{S}_I \varphi \\
\Box\!\!\!-_I \varphi & = & \neg\Diamond\!\!\!-_I \neg\varphi
\end{array}$$

We give strict semantics to the $\mathcal{U}_I$ and $\mathcal{S}_I$ operators from which we can derive classical future time and past time LTL operators[2], using the following rules.

$$\begin{array}{lcl}
\varphi_1 \,\mathcal{U}\, \varphi_2 & = & \varphi_2 \vee (\varphi_1 \wedge \varphi_1 \,\mathcal{U}_{[1,\infty)} \varphi_2) \\
\varphi_1 \,\mathcal{S}\, \varphi_2 & = & \varphi_2 \vee (\varphi_1 \wedge \varphi_1 \,\mathcal{S}_{[1,\infty)} \varphi_2) \\
\bigcirc \varphi & = & \bot \,\mathcal{U}_{\{1\}} \varphi \\
\ominus \varphi & = & \bot \,\mathcal{S}_{\{1\}} \varphi
\end{array}$$

[2]Note that in discrete time, STL and LTL with boolean predicates over $\mathbb{D}$ have the same expressive power, however treating temporal operator with bounds directly leads in general to more efficient monitoring algorithms.

Since we interpret STL in discrete time, we are able to define operators *previous* $\ominus$ and *next* $\bigcirc$. Finally, we often use the following syntactic sugar to describe rising and falling edge of boolean signal p, respectively.

$$\begin{aligned} \uparrow p &= \ominus(\neg p) \wedge p \\ \downarrow p &= \ominus p \wedge (\neg p) \end{aligned}$$

Given an STL formula φ defined over P, we denote by P_φ the set of all subformulas of φ that also includes P and that assigns to every sub-formula ψ of φ a boolean variable[3]. Given a signal w over $P \cup X$ and a formula φ, we define the *satisfaction signal* w_φ over $P_\varphi \cup X$ such that for all $\psi \in P_\varphi$ and $i \in [0, d]$, $\pi_{w_\varphi}(\psi)(i) = \top$ iff $(w, i) \models \psi$.

We also define two useful subsets of STL:

- *past* STL which forbids the usage of the $\mathcal{U}_I$ operator; and
- *bounded future* STL which restricts the usage of the $\mathcal{U}_I$ operator to the case where $I = [a, b]$ for some $0 \leq a \leq b < \infty$.

3.2.3 Rewriting Rules for STL

We will monitor arbitrary STL formulas with temporal testers, described in Section 4.4. For the past-time STL, we first observe that directly treating $\mathcal{S}_I$ and $\diamondminus_I$ may not be straightforward. Thus, we use the following equivalences to transform the formulas into a convenient form.

$$\begin{aligned} \varphi_1 \,\mathcal{S}_{[0,b]} \varphi_2 &= (\varphi_1 \,\mathcal{S} \varphi_2) \wedge \diamondminus_{[0,b]} \varphi_2 \\ \varphi_1 \,\mathcal{S}_{[a,b]} \varphi_2 &= \boxminus_{[0,a-1]} (\varphi_1 \wedge \ominus(\varphi_1 \,\mathcal{S} \varphi_2)) \wedge \diamondminus_{[a,b]} \varphi_2 \\ \boxminus_{[a,b]} \varphi &= \neg \diamondminus_{[a,b]} (\neg \varphi) \\ \diamondminus_{[a,b]} \varphi &= \diamondminus_{\{a\}} \diamondminus_{[0,b-a]} \varphi \end{aligned}$$

A direct way to implement time delay operator $\diamondminus_{\{a\}}$ is simply by instatiating a nested y operators:

$$\diamondminus_{\{a\}} \varphi = \underbrace{\ominus \ominus \ldots \ominus}_{a} \varphi$$

Such strategy is necessary in the most general case. However, for signals that can change only a limited number of times during a certain period of time, we defined a more efficient implementation in section 5.1.1.

The operators seen on right hand side in the equations above, are considered to be the basic past-time STL temporal operators. The benefit of rewriting formulas is clear: we can *efficiently* evaluate any past-time STL operator, if we know how to efficiently evaluate the basic ones.

[3] We abuse the notation and use the same symbol to denote the formula and its associated boolean variable.

Symmetric to past time, the following set of rules hold for the future time STL operators:

$$\begin{aligned}
\varphi_1 \,\mathcal{U}_{[0,b]} \varphi_2 &= (\varphi_1 \,\mathcal{U} \varphi_2) \wedge \Diamond_{[0,b]} \varphi_2 \\
\varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2 &= \Box_{[0,a-1]} (\varphi_1 \wedge \bigcirc (\varphi_1 \,\mathcal{U} \varphi_2)) \wedge \Diamond_{[a,b]} \varphi_2 \\
\Box_{[a,b]} \varphi &= \neg \Diamond_{[a,b]} (\neg \varphi) \\
\Diamond_{[a,b]} \varphi &= \Diamond_{\{a\}} \Diamond_{[0,b-a]} \varphi \\
\Diamond_{\{a\}} \varphi &= \underbrace{\bigcirc \bigcirc \ldots \bigcirc}_{a} \varphi
\end{aligned}$$

Example 1. *We now illustrate the expressiveness of STL with a bounded stabilization requirement. The mixed-signal bounded stabilization property is depicted in Figure 3.2, defines the following requirements:*

- *The absolute value of the continuous signal x is always smaller than 5;*
- *When the Boolean $trigger$ signal rises, within 600 time units the absolute value of x has to drop below 1 and remain below this threshold for at least 300 time units.*

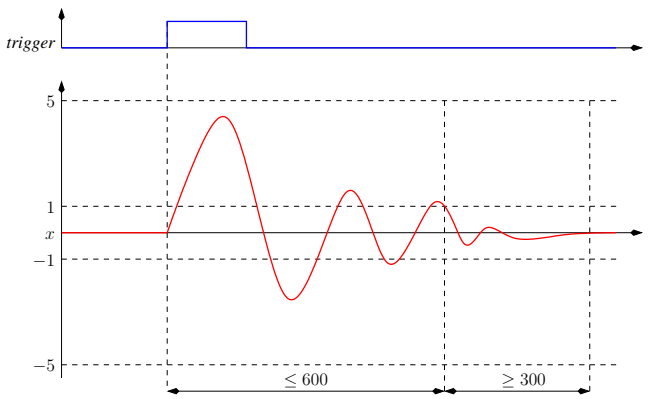


Figure 3.2: An example of bounded stabilization property [Nič08].

We now formalize in STL *the bounded stabilization property requirements with the formula:*

$$\Box(|x| < 5 \wedge (trigger \rightarrow \Diamond_{[0,600]} \Box_{[0,300]} |x| < 1))$$

3.2.4 An AHB Read Transfer in STL

In this section, we illustrate the potential use of STL in practice with an industrial bus communication standard. We take an example from AMBA 3 AHB-Lite protocol which specifies high-speed bus communication for systems based on ARM processors. The Lite version of the AHB protocol is used in systems with a single master. A master device can initiate read or write operations on the bus which are targeting a specific slave device. If the slave is not ready to perform the operation, it introduces a wait state on the bus. During a wait cycle, a master must not initiate a

new transfer. Every read or write transfer consists of two phases: an address phase and a data phase. The address phase is interleaved with the data phase of the previous operation in order to achieve high-speed data rate. The reader can refer to [Ltd06] for the protocol details.

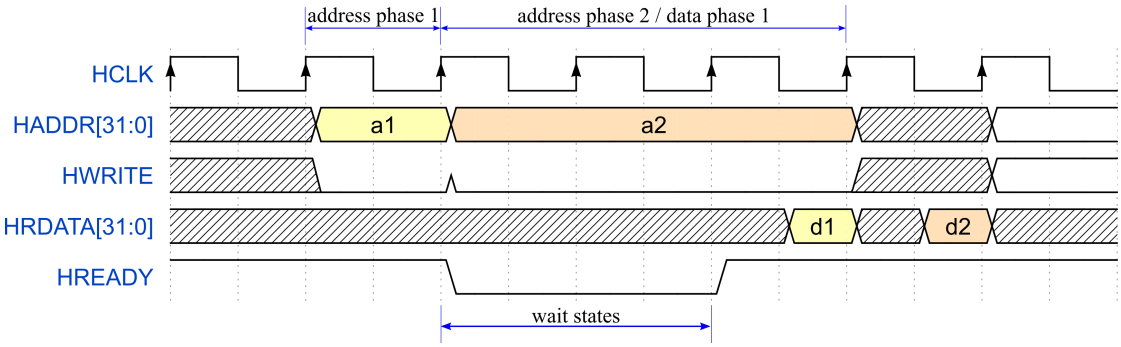


Figure 3.3: AHB Read transfer with two wait states inserted by the slave.

In Fig 3.3 we see two AHB read transfers initiated by the master. The master starts by driving the address for the first read transfer and sets HWRITE to 0. The data phase of the first transfer should take place during the next clock cycle and the slave should provide the data. However, the slave is not ready, and inserts two wait states by setting HREADY signal to 0. Once the slave is ready to provide the data to the master it drives the HRDATA bus and drives HREADY to 1. All control signals are sampled by the master and the slave on positive edge of the clock signal.

Another type of transfer specified in AHB Lite protocol is the burst transfer. It is used by the master to read or write an entire block of data. When a block consists of four memory words with addresses incremented in the size of memory word it is typically transferred as INCR4 type of burst. In Fig 3.5 we see an example of an INCR4 burst: the first transfer is designated as a non-sequential, which means it is the first word of the memory block. The other transfers are sequential, because the addresses are the increments of the first address.

Correctness Requirements

We now formalize the correctness requirements for the scenario in Fig 3.3 using STL. First, the control signal HWRITE must not change before the current value is sampled by a device (master or slave). If there was a transition of HWRITE signal, the value must remain the same *until* it gets sampled, indicated by a positive edge of AHB clock signal HCLK. We abbreviate this description with “HWRITE stable”.

The second requirement is directly driven by the AHB Lite specification [Ltd06]. Changing the value of AHB HWRITE signal while the slave has already initiated a wait state can cause the slave to skip a transfer. According to the protocol, HWRITE signal must not change as long as the slave is demanding the master to wait. Once the slave is ready it will drive HREADY signal and it will sample the HWRITE value. Only then HWRITE is allowed to change. This requirement can be summarized: “HWRITE constant during the wait state”.

Requirement	STL specification
HWRITE stable	$\square(\uparrow\text{HWRITE} \Rightarrow (\text{HWRITE}=1)\ \mathcal{U}\ (\uparrow\text{HCLK}))$ $\square(\downarrow\text{HWRITE} \Rightarrow (\text{HWRITE}=0)\ \mathcal{U}\ (\downarrow\text{HCLK}))$
HWRITE constant during wait state	$\square(\text{HWRITE}=1 \wedge \text{HREADY}=0 \Rightarrow \text{HWRITE}=1\ \mathcal{U}\ (\text{HREADY}=1 \wedge \uparrow\text{HCLK}))$ $\square(\text{HWRITE}=0 \wedge \text{HREADY}=0 \Rightarrow \text{HWRITE}=0\ \mathcal{U}\ (\text{HREADY}=1 \wedge \uparrow\text{HCLK}))$
16 wait states max	$\square(\text{HREADY}=0 \wedge \uparrow\text{HCLK} \Rightarrow \Diamond_{[0,15\cdot Tclk]}(\text{HREADY}=1 \wedge \uparrow\text{HCLK}))$

Table 3.1: AHB read transfer correctness requirements specified in STL.

The AHB Lite protocol standard sets a strong recommendation for the time a slave can spend waiting. In order to prevent a slave from occupying the bus for a very long time, thus degrading the performance, the slave must not initiate more than 16 wait states. In other words, when a slave notifies a master it is not ready to perform the transfer the master must not wait longer than 16 AHB clock cycles to initiate next request. We abbreviate the specification: "16 wait states max".

We use unbounded *always* operator ($\square$) to enforce checking the specification over the entire trace. The specifications contain implication operator ($\Rightarrow$) which allows to define a precondition which, if matched, triggers the check on the right hand side of the implication. The precondition is specified on the left hand side of the $\Rightarrow$ operator.

Comment on expressiveness: In case of requirements that a signal must remain stable, such as "HWRITE stable" and "HWRITE constant during wait state", we cannot formalize it in a single STL formula. Instead, we have to divide it into two more specific requirements where we explicitly state the reference signal value. The reason lies in the fact that STL cannot store a value of a signal inside a variable, which would be used for comparison with the future values. Support for local variables is provided by the industrial standards PSL and SVA [EF08, AFJ13].

3.2.5 Timed Regular Expressions

Timed Regular Expressions (TRE) [ACM97, ACM02a] allow to pattern-match a specification over a signal. As the authors in [FMNU15] mentioned, the fundamental difference between STL and TREs comes from the fact that satisfaction of an STL formula is computed for a time point, while a match of a TRE results in a time interval. In this work we adapt the definition of TREs from [FMNU15] with an assumption of interpreting TREs over discrete time. We reuse definitions of a signal and its projection from the Section 3.2.1. To adhere to the definition from [FMNU15], we make the following assumption: for every Boolean variable $p_j \in P^n$ we admit a definition of a complementary variable $p_j^{\neg}$ with an opposite value of p_j (to which we refer as $\neg p_j$). Every analog variable $x_j \in X^m$ is allowed to be used in TREs only in the form of $x_j \sim c$, where $\sim \in \{<, \leq\}$ and $c \in \mathbb{D}$. With every $x_j \sim c$ we associate the boolean satisfaction variable $p_{x_j \sim c}$; we then analogously define $p^{\neg}_{x_j \sim c}$ and refer to it as $\neg(x_j \sim c)$.

A timed regular expression ψ is defined according to the following syntax [FMNU15]:

$$\psi := \epsilon \mid q \mid rise(p) \mid fall(p) \mid \psi_1 \cdot \psi_2 \mid \psi_1 \cup \psi_2 \mid \psi_1 \cap \psi_2 \mid \psi^* \mid \langle \psi \rangle_I$$

where q is of the form p, $\neg p$, $x \sim c$ or $\neg(x \sim c)$; I is a time interval $[a, b] \subseteq \mathbb{N}$.

For improved readability, we will refer to discrete time instance $i \cdot T$, where T is discrete time step, simply as i. The semantics of timed regular expression φ with respect to discrete signal w and time instances $i \leq i'$ is given in terms of matching relation $(w, i, i') \vDash \varphi$:

$$\begin{array}{lcl}
(w,i,i') \vDash \epsilon & \equiv & i = i' \\
(w,i,i') \vDash q & \equiv & i \leq i' \text{ and } \forall i'' \text{ s.t. } i \leq i'' < i', \pi_q(w)[i''] = 1 \\
(w,i,i') \vDash rise(p) & \equiv & i = i' \text{ and } (w,i,i') \vDash p \text{ and } (w,i-1,i') \vDash \neg p \\
(w,i,i') \vDash fall(p) & \equiv & i = i' \text{ and } (w,i,i') \vDash \neg p \text{ and } (w,i-1,i') \vDash p \\
(w,i,i') \vDash \varphi_1 \cdot \varphi_2 & \equiv & \exists i'' \text{ s.t. } i \leq i'' < i', (w,i,i'') \models \varphi_1 \text{ and } (w,i'',i') \models \varphi_2 \\
(w,i,i') \vDash \varphi_1 \cup \varphi_2 & \equiv & (w,i,i') \models \varphi_1 \text{ or } (w,i,i') \models \varphi_2 \\
(w,i,i') \vDash \varphi_1 \cap \varphi_2 & \equiv & (w,i,i') \models \varphi_1 \text{ and } (w,i,i') \models \varphi_2 \\
(w,i,i') \vDash \varphi^* & \equiv & (w,i,i') \models \epsilon \text{ or } (w,i,i') \models \varphi \cdot \varphi^* \\
(w,i,i') \vDash \langle \varphi \rangle_I & \equiv & i' - i \in I \text{ and } (w,i,i') \models \varphi
\end{array}$$

As before, we use the notation $\{a\}$ for intervals of the form $[a, a]$. We use a superscript with a TRE to denote a number of concatenations of this TRE (e.g. if ψ is a TRE, then ψ^3 stands for $\psi \cdot \psi \cdot \psi$). Finally, we use ψ^+ as syntactic sugar for $\psi \cdot \psi^*$.

Example 2. *We illustrate usage of TRE as a specification language with an Anti-lock Break System from [FMNU15] as an example.*

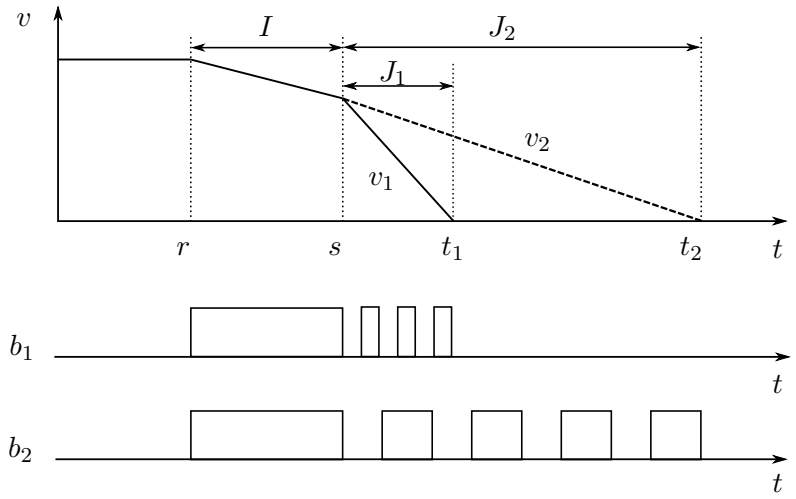


Figure 3.4: Braking patterns for different stopping distances with ABS.

In the Figure 3.4 we can observe two signals b_1 and b_2 for braking control and v_1 and v_2 for velocity. These signals represent behavior of two different braking controllers. At time unit r we can see that the driver is braking until ABS takes control at time s. After that moment controller with ABS is toggling the brake input in order to prevent the vehicle from sliding and losing traction with the road.

We describe the ABS anti-lock pattern with the following TRE:

$$\psi = rise(b) \cdot \langle b \rangle_I \cdot \langle \neg b \cdot b \rangle_J^+ \cdot fall(b).$$

The expression specifies that first the rising edge of braking signal b must occur meaning that the driver should start braking. The driver is allowed to brake up to I time units, when ABS takes control and produces pulses in total duration of up to J time units. The transition from the last breaking pulse to zero breaking is described by $fall(b)$. The specification which consists of sequential events is naturally formalized using the concatenation $(\cdot)$ operator.

From TRE to Automata

Translation from TRE to timed automata is defined in [ACM02a]. In this thesis, we use a straightforward adaptation of the TRE translation to timed automata from [ACM02a] to the discrete time setting.

3.2.6 An AHB Read Transfer in TRE

The requirements from section 3.2.4 can also be specified in Timed Regular Expressions. We extensively use TRE concatenation operator $(\cdot)$ to build longer expression sequences. TRE operator $\cup$ is analogous to logical $\wedge$, used in STL. Expressions prefixed with *sync_* represent values of respective signals, sampled at rising edge of the clock.

In TRE, we defined analogous constructs to $\uparrow x$ and $\downarrow x$ with `rise`(x) and `fall`(x). We also define $p^n = p \cdot p \cdot ... \cdot p$ as the abbreviation for n concatenations of TRE p.

Requirement	TRE specification
HWRITE stable	(HWRITE=1)$^+\cdot$ `rise`(HCLK) (HWRITE=0)$^+\cdot$ `rise`(HCLK)
HWRITE constant during wait state	sync_ready := `rise`(HCLK) $\cap$ (HREADY=1) (HWRITE=1 $\cap$ HREADY=0) $\cdot$ (HWRITE=1)$^+\cdot$ sync_ready (HWRITE=0 $\cap$ HREADY=0) $\cdot$ (HWRITE=0)$^+\cdot$ sync_ready
16 wait states max	sync_wait := `rise`(HCLK) $\cap$ (HREADY=0) $\langle$ sync_wait$^+\cdot$ sync_ready $\rangle_{[0,16\cdot T_{CLK}]}$

Table 3.2: AHB read transfer correctness requirements specified in TRE.

3.2.7 Usability of STL and TRE for different types of requirements

In this section we provide intuition on suitability of STL and TRE for different specifications.

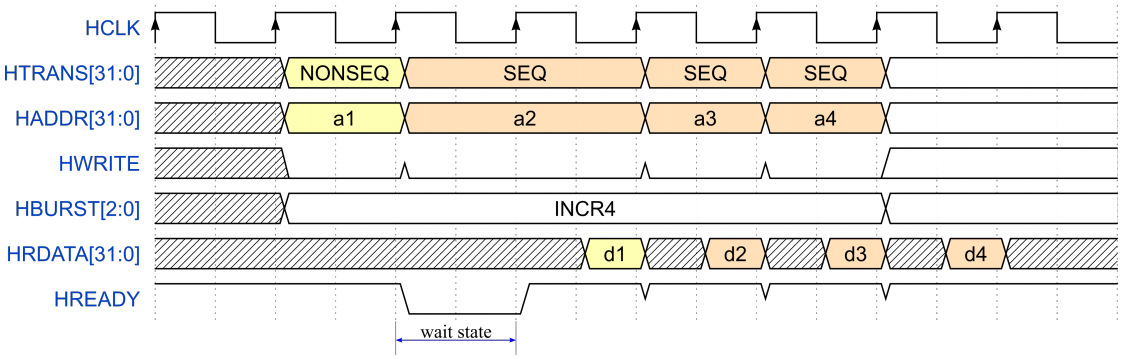


Figure 3.5: AHB Burst of four read transfers with incrementing addresses.

The AHB INCR4 burst shown in Figure 3.5, is used to transfer memory blocks which consist of four memory words. Therefore it is important to ensure that a master initiates exactly four transfers. For the simplicity of the example, we will disregard a possible scenario where a master can initiate more than four transfers. In other words, with the following specification we ensure there are no transfer of a burst is missing.

Requirement in natural language	TRE specification
INCR4 burst takes 4 transfers	sync_nonseq := (HTRANS=NONSEQ) ∩ (HREADY=1) sync_seq := (HTRANS=SEQ) ∩ (HREADY=1) incr4 := sync_nonseq · $[(\text{HREADY}=0)^* \cdot (\text{sync\_seq})]^3$
INCR4 burst takes between 5 and 64 cycles	$\langle$ incr4 $\rangle_{[5,64 \cdot T_{CLK}]}$

Table 3.3: AHB INCR4 burst transfer specified in TRE.

This requirement encapsulates a sequence of events. However, each of the events may have a variable duration due to wait states possibly introduced by a slave. For this reason it is very cumbersome to specify such requirement in STL, because the language does not allow specifying "0 or more" operator on the wait states in an elegant way. In theory, such requirement could be formalized by enumerating all the different wait state scenarios and grouping them with a disjunction. In Table 3.3 we see a succinct way to represent the requirement as a concatenation of smaller instances of TRE which are related to single transfers in a burst. Building a specification in a compositional manner results in more readable formulas.

The second INCR4 burst requirement is ensuring the total duration of the burst does not exceed time constraint. Since each of the transfer can introduce wait states and can take up to 16 clock

cycles to complete, total duration of INCR4 burst must not exceed 64 clock cycles. For this requirement it is necessary to specify a constraint on total duration of the sequence of events. However, STL operators allow defining timing constraints only over two consecutive events. In Table 3.3, we overcome this problem by leveraging the *time restriction operator* from TRE which specifies a time constraint over an entire sequence of events which constitute the INCR4 burst.

CHAPTER 4

Automata

After defining signals and specification languages for the requirements we want to monitor, we proceed with defining a formal representation of our specification. In this section we introduce two classes of automata that we use to represent specifications and generate monitors - weighted symbolic automata and temporal testers.

In first approach we define a language acceptor which will be used to accept only the traces which satisfy the specification. Due to hardware setting of our monitors, we typically reduce the alphabet of our automata to a set of integers or finite-precision reals. In order to efficiently represent such alphabets, we use symbolical constraints. The definition of *metric space* is used to define weight functions and obtain Weighted Symbolic Automata. We can further generalize the approach with weight functions defined using *algebraic structures*. In Chapters 7 and 6 we provide algorithms for quantitative monitoring based on Weighted Symbolic Automata and its generalization.

We also explain the theoretical background of *temporal testers*, which we use as foundation for our qualitative FPGA monitors. Pnueli and Zaks introduced temporal testers in [PZ08] and showed how to build monitors in a modular way following syntactical dependencies within a temporal logic formula. In cases when variability of input signal is known, we treat time symbolically and equip our temporal testers with discrete clock variables. Instead of accepting or rejecting an input sequence like an automata, a temporal tester outputs a satisfaction value for a given formula at an arbitrary time instance $t \geq 0$.

4.1 Predicates

We start by defining the predicates which will be used to build constraints for our Weighted Symbolic Automata and Temporal Testers.

Depending on the application, we can set the domain $\mathbb{D}$ to the set of reals or to some bounded integer domain. Let X be the set of variables defined over $\mathbb{D}$. We now define predicates over variables in X.

Definition 2 (Predicate)**.** *We define the syntax of a* predicate ψ *over* X *with the following grammar:*

$$\psi := \bot \mid \top \mid x \preceq k \mid \neg\psi \mid \psi_1 \vee \psi_2 \mid \psi_1 \wedge \psi_2,$$

where $\preceq \in \{<, \leq\}$, $x \in X$ *and* $k \in \mathbb{D}$.

We denote by $\Psi(X)$ the set of all predicates over X. We say that a valuation v *models* the predicate ψ, denoted by $v \models \psi$, iff ψ evaluates to true under v.

We note that $x \preceq k$ plays the role of basic propositions. In our framework, the predicates come from specifications, hence we allow predicates in arbitrary form. In order to define the subsequent RV algorithms, we need to transform arbitrary predicates into *(minimal) Disjunctive Normal Form (DNF)*.

Definition 3 (Predicate in Disjunctive Normal Form (DNF))**.** *A predicate in disjunctive normal form is generated by the following grammar:*

$$\psi := \psi^c \mid \psi^c \vee \psi \quad ; \quad \psi^c := \psi^l \mid \psi^l \wedge \psi^c \quad ; \quad \psi^l := \top \mid \bot \mid (x \preceq k) \mid \neg(x \preceq k)$$

where $\preceq \in \{<, \leq\}$, $x \in X$ *and* $k \in \mathbb{D}$. *The predicate has the following structure:*

$$\psi = \bigvee_{i=1}^{h} \psi_i^c, \quad \psi_i^c = \bigwedge_{h=1}^{m(i)} \psi_{i,h}^l$$

where h *is the number of clauses, each clause* i *is a conjunction of* $m(i)$ *literals and each literal can be either a basic proposition, its negation, true or false.*

Definition 4 (Predicate in $\wedge$-minimal DNF)**.** *A predicate* ψ *is expressed in a* $\wedge$*-minimal DNF if it satisfies the following properties:*

$$\bigwedge_{i=1}^{h} \bigwedge_{s=1,r=1,s\neq r}^{m(i)} (\psi_{i,s}^l \rightarrow \psi_{i,r}^l) = \bot$$

Example 3. *The predicate* $\psi_1 \equiv (x \leq 3 \wedge x \leq 5 \wedge y \leq 5) \vee (z > 0)$ *is in DNF, while the predicate* $\psi_2 \equiv (x \leq 3 \wedge y \leq 5) \vee (z > 0)$ *is in* $\wedge$*-minimal DNF.*

4.2 Weighted Symbolic Automata

In this section, we define a variant of *symbolic automata* [VBdM10] and also present its *weighted* extension. The notion of weighted automata and its well-established theory is provided

in [DKV09]. The symbolic weighted automata accepting input string over not necessarily finite set have been investigated in [HV16].

The variable valuation $v(x)$ is a function $v : X \to \mathbb{D}$, which we naturally extend to the valuation $v(X)$ of the set X. Given the valuation $v(X)$ and a predicate ψ over X, we write $v(X) \models \psi$ when $v(X)$ satisfies ψ.

Definition 5 (Symbolic Automata)**.** *A symbolic automaton (SA) is a tuple $\mathcal{A} = (X, Q, I, F, \Delta)$, where:*

- *X is a finite set of variables defined over a domain $\mathbb{D}$,*
- *Q is a finite set of* locations*,*
- *$I \subseteq Q$ is the set of* initial *states,*
- *$F \subseteq Q$ is the set of* final *states,*
- *$\Delta = \Delta_X \cup \Delta_\epsilon$ is the* transition relation *where $\Delta_X \subseteq Q \times \Psi(X) \times Q$ and $\Delta_\epsilon \subseteq Q \times \{\epsilon\} \times Q$.*

We note that the above definition of symbolic automata allows silent transitions Δ_ϵ, transitions which can be taken without consuming an input symbol. In case when $\Delta_\epsilon = \varnothing$ we speak of a subclass of *symbolic automata without silent transitions*.

Given a $q \in Q$, we define a set of states reachable from q by following transitions if Δ with $reachable(q)$. In addition, let $\mathcal{E}(q)$ denote the set of states reachable from q by following only ϵ-transitions in Δ_ϵ . Formally, we say that $q_k \in \mathcal{E}(q)$ iff there exists a sequence of states $q_1, \ldots, q_k$ such that $q = q_1$, $(q_i, \epsilon, q_{i+1}) \in \Delta_\epsilon$ for all $0 \leq i < k$. Let $s : [0, l) \times X \to \mathbb{D}$ be a signal.

Definition 6 (Trace of a Symbolic Automaton)**.** *We say that s is a* trace *of a symbolic automaton $\mathcal{A}$ if there exists a sequence of states $q_0, \ldots, q_l$ in Q such that:*

- *$q_0 \in I$ or $q_0 \in \mathcal{E}(q_{init})$ for some $q_{init} \in I$, and*
- *for all $0 \leq i < l$, there exists either $(q_i, \psi, q_{i+1}) \in \Delta_X$ for some $\psi \in \Psi(X)$ such that $s(i, X) \models \psi$ or $q_{i+1} \in \mathcal{E}(q_i)$, and*
- *$q_l \in F$.*

We denote by $L(\mathcal{A})$ the set of all traces of $\mathcal{A}$.

Definition 7 (Path in a Symbolic Automaton)**.** *A* path *π in a symbolic automaton $\mathcal{A}$ is a sequence $\pi = q_0 \cdot \delta_0 \cdot q_1 \cdots \delta_{n-1} \cdot q_n$ such that $q_0 \in I$ and for all $0 \leq i < n$, δ_i is either of the form (q_i, ψ, q_{i+1}) or (q_i, ϵ, q_{i+1}). We say that π is* accepting *if $q_n \in F$.*

Given a trace $s : [0, l) \times X \to \mathbb{D}$ and a path $\pi = q_0 \cdot \delta_0 \cdot q_1 \cdot \delta_1 \cdots \delta_{n-1} \cdot q_n$, we say that s induces π in $\mathcal{A}$ if π is an accepting path in $\mathcal{A}$ and its projection to observable alphabet letters gives s. We denote by $\Pi(\mathcal{A}, s) = \{\pi \mid s \text{ induces } \pi \text{ in } \mathcal{A}\}$ the set of all paths in $\mathcal{A}$ induced by s.

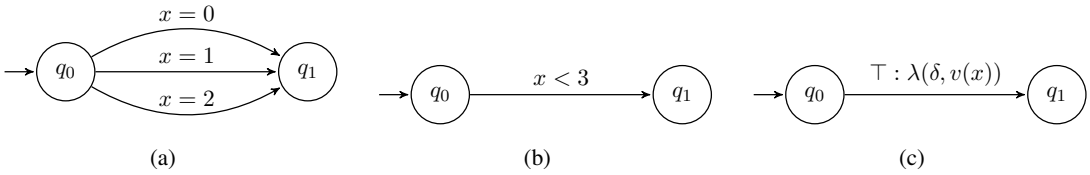


Figure 4.1: From automata transitions to WSA transitions.

We now introduce *weighted* symbolic automata, by adding weight function to the transitions of the symbolic automaton. The intuition is given in Figure 4.1 where one can notice the weight function $\lambda(\delta, v(x))$. In this example we set the domain $\mathbb{D} = \mathbb{N}_0$.

Definition 8 (Weighted Symbolic Automata)**.** *A weighted symbolic automaton (WSA) $\mathcal{W}$ is the pair $(\mathcal{A}, \lambda)$, where $\lambda : \Delta \times \mathbb{D}^{|X|} \to \mathbb{D}$ is the* weight *function.*

Example 4. *We now give an intuition of an acceptor and a Weighted Symbolic Automaton with two variables x and y. We select as domain a subset of positive integers $\mathbb{D} = \{0, 1, 2, 3, 4, 5\}$. On the left hand side of Figure 4.2 we can see an acceptor for the formula $\square(x < 3 \ \mathcal{S} \ y = 5)$. The acceptor has an implicit error sink state, which is omitted in Figure 4.2 (a). In a general setting, we assign an arbitrary weight function $\lambda(\delta_i, v(x), v(y))$ which depends on input variable valuations.*

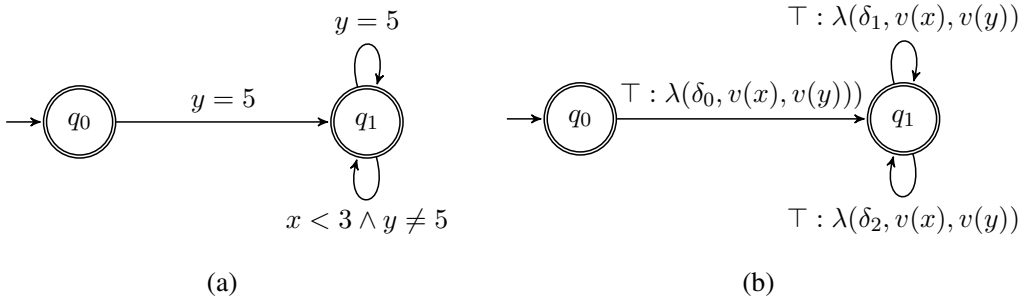


Figure 4.2: An acceptor and a weighted symbolic automaton for formula $\square(x < 3 \ \mathcal{S} \ y = 5)$.

In a WSA,the weight function $\lambda(\delta_i, v(x), v(y))$ allows transition to be taken with any input symbol, with a specific cost. Let's take an input sequence $(x, y) = \{(0, 5), (2, 5), (2, 4)\}$. This sequence is non-accepting in the automaton because of the last pair of input symbols $(2, 4)$. On the other hand, this sequence is accepting in WSA from Figure 4.2. For the simplicity of this example, we can say that the input sequence is accepted in WSA with the cost of $C = \lambda(\delta_0, 0, 5)) + \lambda(\delta_1, 2, 5)) + \lambda(\delta_2, 2, 4))$.

4.3 Algebraic Structures

In Example 4 we have seen a Weighted Symbolic Automata which accepts any input sequence with a specific cost. Each transition taken on a specific path incurs cost specified by the input symbols and the weight function λ. The cost incurred by an entire path is aggregation of the costs incurred by transitions taken along that path. In this section we define algebraic structures in order to lay foundations of a general algorithm for calculating path cost in WSA in Chapter 6.

Definition 9 (Semiring)**.** *A semiring* $\mathcal{S}$ *is a tuple* $\mathcal{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$*, where* S *is a set equipped with two binary operations* addition *(*$\oplus$*) and* multiplication *(*$\otimes$*) and two identity elements* $e_\oplus$ *and* $e_\otimes$ *such that:*

- $(S, \oplus, e_\oplus)$ *is a commutative monoid with an identity element* $e_\oplus$*;*
- $(S, \otimes, e_\otimes)$ *is a monoid with an identity element* $e_\otimes$*;*
- $\otimes$ *distributes over* $\oplus$*; and*
- $e_\oplus$ *is an annihilator element for* $\otimes$*.*

We say that a semiring is *commutative* if the $\otimes$-multiplication operation is commutative. A semiring is said to be *additively (multiplicatively) idempotent* if for all $s \in S$, we have that $s \oplus s = s$ ($s \otimes s = s$). We say that a semiring is *idempotent* if it is both additively and multiplicatively idempotent. A semiring is *bounded* if $e_\otimes$ is an annihilator element for $\oplus$. We note that a bounded semiring is also additively idempotent [Moh02].

Example 5. *We depict in Table 4.1 several examples of semiring structures. We note that both the Boolean, MinMax and the powerset semiring are commutative and idempotent. The tropical semiring is commutative and additively idempotent. All four semirings are bounded. Probability semiring is neither multiplicatively not additively idempotent. Note that we use a non-standard definition of the Boolean and MinMax semirings, in which* $\oplus$ *corresponds to* $\wedge$ *and* $\min$*, while* $\otimes$ *corresponds to* $\vee$ *and* $\max$*.*

Semiring	S	$\oplus$	$\otimes$	$e_\oplus$	$e_\otimes$
Boolean	$\{0, 1\}$	$\wedge$	$\vee$	1	0
MinMax	$\mathbb{R}_+ \cup \{\infty\}$	$\min$	$\max$	∞	0
Tropical	$\mathbb{R}_+ \cup \{\infty\}$	$\min$	$+$	∞	0
Probability	$\mathbb{R}_+ \cup \{\infty\}$	$+$	$\times$	0	1
Powerset	$\mathcal{P}(T)$	$\cap$	$\cup$	T	$\emptyset$

Table 4.1: Examples of semirings.

Additively idempotent semirings defined over sets of Booleans, naturals and reals admit a natural order between the elements in the set. In this thesis, we restrict our study to additively idempotent semirings.

Definition 10 (Natural order on S)**.** *Let $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ be an additively idempotent semiring. We define the* natural order *on S as the relation $\sqsubseteq$: $(a \sqsubseteq b) \leftrightarrow (a \oplus b = a)$. If $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ is also multiplicatively idempotent, we additionaly require: $(a \sqsubseteq b) \leftrightarrow (a \otimes b = b)$.*

Lemma 1 ([Moh02])**.** *Let $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ be an additively idempotent semiring. The natural order $\sqsubseteq$ on S defines a partial order.*

We now define the monotonicity of semirings, an important property that will allow us factoring and thus simplifying our RV operations.

Definition 11 (Negative and monotonic semirings)**.** *Let $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring. We say that S is* negative *if $e_\otimes \sqsubseteq e_\oplus$. We say that S is* monotonic *if for all $a, b, c \in S$:*

1. $a \sqsubseteq b \rightarrow (a \oplus c) \sqsubseteq (b \oplus c)$
2. $a \sqsubseteq b \rightarrow (a \otimes c) \sqsubseteq (b \otimes c)$
3. $a \sqsubseteq b \rightarrow (c \otimes a) \sqsubseteq (c \otimes b)$

Lemma 2 ([Moh02])**.** *Let $\mathbb{S}$ be an additively idempotent semiring $(S, \oplus, \otimes, e_\oplus, e_\otimes)$ equipped with the natural order $\sqsubseteq$ on S. Then S is both negative and monotonic.*

Lemma 3. *Let $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ be an additively idempotent, negative and monotonic semiring. Then, for all $a \in S$, $e_\otimes \sqsubseteq a \sqsubseteq e_\oplus$.*

The proof of Lemma 3 is provided in Appendix A.1.

4.3.1 Metric Space and Distance

A metric space is a set $\mathcal{M}$ which has a defined distance among its elements. The distance $d(m_1, m_2)$ between two elements $m_1, m_2 \in \mathcal{M}$ is a positive real value in $\mathbb{R}_+$.

Definition 12 (Metric space and distance)**.** *Given a set S, let $d : \mathcal{M} \times \mathcal{M} \rightarrow \mathbb{R}_+$ be a* distance*. Then $\mathcal{M}$ is a* metric space *with the* distance measure *d, if:*

1. $d(m_1, m_2) \geq 0$ *for all m_1, m_2 in $\mathcal{M}$;*
2. $d(m_1, m_2) = 0$ *if and only if $m_1 = m_2$;*
3. $d(m_1, m_2) = d(m_2, m_1)$ *for all m_1, m_2 in $\mathcal{M}$; and*
4. $d(m_1, m_2) \leq d(m_1, m) + d(m, m_2)$ *for all m, m_1, m_2 in $\mathcal{M}$.*

Example 6. *A very simple example is any set for which we are able to strictly test the equivalence of two elements x and y. In this case we can define a discrete distance measure $d(x, y) = 0$ iff $x = y$ and $d(x, y) = 1$ iff $x \neq y$.*

4.4 Temporal Testers

In this section we explain theoretical foundations of *temporal testers* [PZ08, KPR98], which we use to represent the requirement we want to verify. The motivation to use temporal testers comes from the fact that automata do not allow building top-level formula automaton in a compositional fashion.

Let's assume that we have acceptors for sub-formulas φ_1 and φ_2. Then, it is not easy to obtain an acceptor for $\varphi_1 \mathcal{U} \varphi_2$, because acceptors for φ_1 and φ_2 provide no information about satisfaction of the formula at an arbitrary time instance t. They only take into account the entire trace, starting from $t = 0$. To support compositional approach, it is necessary that building blocks for the top-level specification model provide fine grain temporal information on the satisfaction of the formula, as specifed by semantics of temporal logic operators.

In constrast to an acceptor, a temporal tester outputs a value which represents the satisfaction of the formula φ w.r.t. an arbitrary time instance $t > 0$ and all the *suffixes* of the trace at that time instance. This property allows to build a tester for a top-level formula by composing testers for the subformulas. Furthermore, if we have a subformula which appears on several places in top-level formula we can build the temporal tester for the subformula only once and reuse it for each of the occurrences. In Example 7 we compare a basic acceptor and a temporal tester for *next* operator $\bigcirc p$.

Example 7. *On left hand side of Figure 4.3 we can see an acceptor for $\bigcirc p$ with an accepting state q_2. Due to semantic of $\bigcirc$ operator, the acceptor skips current symbol and observes the following symbol in order to accept or reject an input sequence. If the following symbol is not p the acceptor goes to sink state q_3 and no suffix can ever make this sequence an accepting one. If the following input symbol is p automaton accepts the sequence regardless of the suffix, since such trace satisfies the temporal operator semantics at $t = 0$.*

Temporal tester, seen in Figure 4.3 (b), follows a different principle. In every time instance it outputs positive satifaction verdict through variable y if the trace satisfies the next operator in that time instance and all its' suffixes. The "next" operator satisfaction depends on future input symbols. Since the tester cannot know the future input symbol, it outputs y as a prediction of the formula satisfaction verdict, which can be either positive or negative. Once the future becomes known, only the path which made a correct prediction is taken into account.

4.4.1 Formal Definition

Temporal testers are represented with a set of states and transitions and input and output variables. Output variables do not represent input symbols, their values is assigned by the tester itself.

Definition 13 (Output condition)**.** *We define a* condition over an output variable y *with the following grammar:*

$$\psi := y \mid \neg y.$$

The set of all conditions of an output variable is designated with $\Gamma(Y)$.

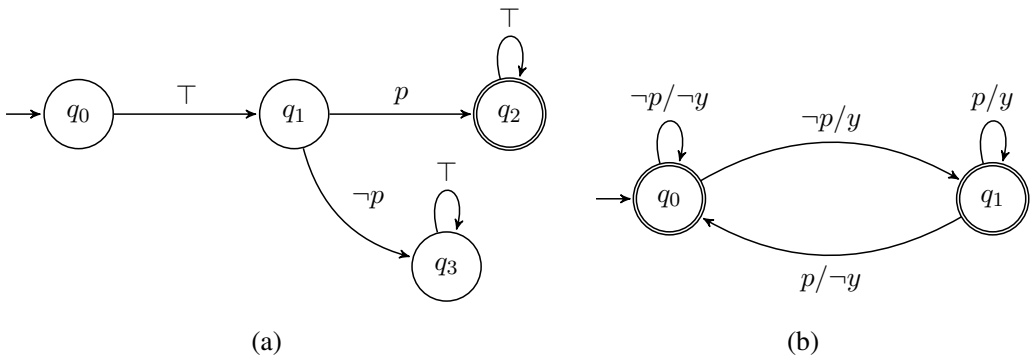


Figure 4.3: An acceptor and a temporal tester for $\bigcirc p$.

Let $\mathcal{C} = \{c_1, \ldots, c_m\}$ be a set of *clock variables*, each ranging over $\mathbb{N}_0$. A *configuration* of a temporal tester is a tuple (q, v) with q being the *discrete location* of the tester and v the valuation of variables. For a clock valuation $v_{\mathcal{C}} = (v_{c_1}, \ldots, v_{c_m})$, $v_{\mathcal{C}} + k$ is the valuation $(v'_{c_1}, \ldots, v'_{c_m})$ such that $v'_{c_i} = v_{c_i} + k$ for all $i \in [1, m]$. Given $C \subseteq \mathcal{C}$, $reset[C]$ is an assignment of values to each clock variable c_i such that $v'_{c_i} = 0$ if $c_i \in C$, and $v'_{c_i} = v_{c_i}$ otherwise. A *clock constraint* is a predicate ψ over $\mathcal{C}$, according to Definition 2.

Example 8. *On right hand side of the Figure 4.4 we can see an example of a transition with a clock constraint. Transition to location q_1 depends not only on the input variable p, but also on the valuation of clock variable c. Resetting clock c before entering location q_0 enforces tester to stay in location q_0 for at least 5 time units.*

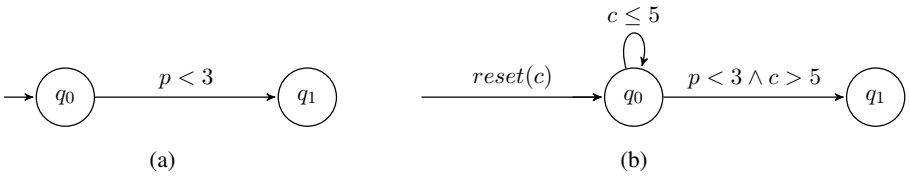


Figure 4.4: Adding clock constraints to transitions.

Definition 14 (Temporal Testers)**.** *A* temporal tester *is a tuple $T = (X, Y, \mathcal{C}, Q, I, F, \Delta)$, where:*

1. *X is a finite set of* input variables *defined over some domain $\mathbb{D}$;*
2. *Y is a finite set of* output variables *defined over $\mathbb{B}$;*
3. *$\mathcal{C}$ is a finite set of* clock variables*;*
4. *Q is a finite set of* locations*;*
5. *$I \subseteq Q$ is the set of* initial *locations;*

6. $F \subseteq Q$ *is the set of* final *locations;*

7. $\Delta \subseteq Q \times \Psi(X) \times \Gamma(Y) \times \Psi(\mathcal{C}) \times 2^{\mathcal{C}} \times Q$*, such that a transition* $\delta \in \Delta$ *is of the form* $(q, \psi_X, \gamma_Y, \psi_{\mathcal{C}}, C, q')$*, where:*
   a) $q, q' \in Q$ *are the* source *and* target *locations;*
   b) $\psi_X \in \Psi(X)$ *and* $\psi_{\mathcal{C}} \in \Psi(\mathcal{C})$ *are the* guards *over inputs* X *and clocks* $\mathcal{C}$*;*
   c) $\gamma_Y \in \Gamma(Y)$ *is the output condition over* Y*; and*
   d) $C \subseteq \mathcal{C}$ *is the set of variables to be* reset *by the transition.*

Let (q, v) and (q', v') be two configurations of a temporal tester, and $v(X)$, $v(Y)$ and $v(\mathcal{C})$ valuations over X, Y and $\mathcal{C}$, respectfully. We say that $(q, v) \cdot \delta \cdot (q', v')$ is a *step* for some $\delta = (q, \psi_X, \gamma_Y, \psi_{\mathcal{C}}, C, q') \in \Delta$, if:

1. $v(X) \models \psi_X$, $v(Y) \models \gamma_Y$ and $v(\mathcal{C}) \models \psi_{\mathcal{C}}$;
2. $v'(c) = 0$, for all $c \in C$; and
3. $v'(c) = v(c) + 1$, for all $c \in \mathcal{C} \backslash C$.

Let $s : [0, l) \times X \to \mathbb{D}$ be an input signal of length l defined over X. We say that s induces a *path* $\pi = (q_0, v_0) \cdot \delta_0 \cdot (q_1, v_1) \cdots \delta_{l-1} \cdot (q_l, v_l)$ and generates a Boolean signal $u : [0, l) \times Y \to \mathbb{B}$ in a temporal tester T if:

1. $q_0 \in I$ and for all $c \in \mathcal{C}$, $v_0(c) = 0$; and
2. for all $0 \leq i < l$, there exists a step $(q_i, v_i) \cdot \delta_i \cdot (q_{i+1}, v_{i+1})$ such that $v_i(X) = s(i, X)$ and $v_i(Y) = u(i, Y)$.

We say that a path π is *accepting* if $q_{l+1} \in F$. We say that s is an *input trace* in T if there exists an accepting path π in T induced by s. Given $s : [0, l) \times X \to \mathbb{D}$ and $u : [0, l) \times Y \to \mathbb{B}$, let $\tau : [0, l) \times (X \cup Y) \to (\mathbb{D} \cup \mathbb{B})$ be a signal such that for all $x \in X$, $\tau(i, x) = s(i, x)$ and for all $y \in Y$, $\tau(i, y) = u(i, y)$. We say that τ is a *trace* in T if s is an input trace in T that generates u.

We say that a temporal tester is *deterministic* if for all pairs of transitions originating from the same source location $(q, \psi^1_X, \gamma^1_Y, \psi^1_{\mathcal{C}}, C^1, q')$ and $(q, \psi^2_X, \gamma^2_Y, \psi^2_{\mathcal{C}}, C^2, q'')$ in Δ, we have that $\psi^1_X \wedge \gamma^1_Y \wedge \psi^1_{\mathcal{C}} \wedge \psi^2_X \wedge \gamma^2_Y \wedge \psi^2_{\mathcal{C}}$ is unsatisfiable. Similarly, we say that a temporal tester is *input-deterministic* if $\psi^1_X \wedge \psi^1_{\mathcal{C}} \wedge \psi^2_X \wedge \psi^2_{\mathcal{C}}$ is unsatisfiable.

Definition 15 (Parallel composition)**.** *Let* $T_1 = (X_1, Y_1, \mathcal{C}_1, Q_1, I_1, F_1, \Delta_1)$ *and* $T_2 = (X_2, Y_2, \mathcal{C}_2, Q_2, I_2, F_2, \Delta_2)$ *be two temporal testers such that* $Y_1 \cap Y_2 = \emptyset$ *and* $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$*. We define their parallel composition* $T = T_1 \,||\, T_2$ *as the temporal tester* $(X, Y, \mathcal{C}, Q, I, F, \Delta)$*, where:*

1. $Y = Y_1 \cup Y_2$*;*
2. $X = (X_1 \cup X_2) \backslash Y$*;*
3. $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$*;*

4. $Q = Q_1 \times Q_2$;

5. $I = I_1 \times I_2$;

6. $F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2\}$; *and*

7. Δ *is the set of transitions of the form* $((q_1, q_2), \psi_X, \gamma_Y, \psi_{\mathcal{C}}, C, (q_1', q_2'))$ *such that* $(q_1, \psi^1_{X_1}, \gamma^1_{Y_1}, \psi^1_{\mathcal{C}_1}, q_1') \in \Delta_1$, $(q_2, \psi^2_{X_2}, \gamma^2_{Y_2}, \psi^2_{\mathcal{C}_2}, q_2') \in \Delta_2$, *where:*
   a) $\psi_X = \psi^1_{X_1 \setminus Y} \wedge \psi^2_{X_2 \setminus Y}$;
   b) $\gamma_Y = \gamma^1_{Y_1} \wedge \gamma^2_{Y_2} \wedge \psi^1_{X_1 \cap Y} \wedge \psi^2_{X_2 \cap Y}$; *and*
   c) $\psi_{\mathcal{C}} = \psi^1_{\mathcal{C}_1} \wedge \psi^2_{\mathcal{C}_2}$ *and* $C = C_1 \cup C_2$.

The *basic temporal tester* for a formula φ is a transition system T_φ with variables $P_\varphi \subseteq X_\varphi$ and an output variable y_i, that satisfies the following condition: for every input trace s in T_φ, $y(i) = \top$ iff $(w, i) \models \varphi$. The *full temporal tester* (or simply temporal tester) $T_{\{\varphi\}}$ for an arbitrary temporal formula φ is the parallel composition $||_{\psi \in cl(\varphi)} T_\psi$ of the basic temporal testers for its sub-formulas.

As previously mentioned, we can treat time symbolically in case when we have guarantees that an input signal cannot change during at least N time units. This property of a signal is known as *signal variability*. We use temporal testers with discrete clocks to evaluate STL operator satisfaction over a signal of known bounded variability, otherwise we set $\mathcal{C} = \emptyset$.

4.4.2 Basic Temporal Testers for Past Time STL

Our approach for building monitors using temporal testers leverages rewriting rules for STL operators to transform formulas into a convenient form for hardware implementation. Here we present temporal testers for the operators we can efficiently map to hardware.

In order to handle arbitrary STL formulas, we will implement them as deterministic temporal testers. For the past-time STL, we first observe that directly treating $\mathcal{S}_I$ and $\diamondsuit_I$ may not be straightforward - we use instead the equivalences from section 3.2.3 to first simplify the formulas.

$p \mathcal{S} q$ tester

In Figure 4.5 we see a basic temporal tester for unbounded since operator. According to STL semantics, it is necessary to see q followed by a continuous sequence of p. In case there is no q input observed the operator is not satisfied, and the temporal tester loops in state q_0 and outputs $\neg y$. An input q observed is considered as a beginning of a satisfying sequence. Reaching location q_1 means that input q has been observed. Then, a sequence of p or q will suffice to satisfy temporal operator semantics. If we observe both $\neg p$ and $\neg q$ in location q_1 the semantic relation of unbounded since operator is not satisfied, the tester outputs $\neg y$ and transitions back to q_0.

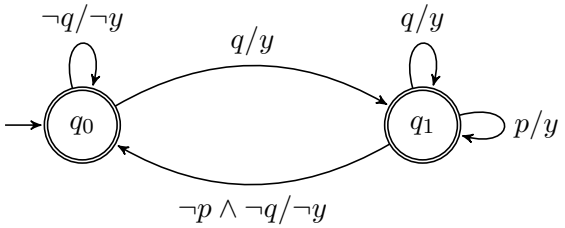


Figure 4.5: Temporal tester for $p \mathcal{S} q$.

$\ominus p$ tester

In similar fashion we define temporal tester for temporal operator $\ominus p$ in Figure 4.6. Current output value y depends on the value of p from the previous time instance. Initial output of the tester is $\neg y$. Initial location q_0 can be reached only if $\neg p$ input was consumed. Therefore, when tester is in q_0 the $\neg p$ symbol was previously received and the tester needs to output $\neg y$ at current time instance. Each of the exiting transitions from q_0 output $\neg y$. Transitions from and to state q_1 have an analogous functionality for the case when p was previously observed.

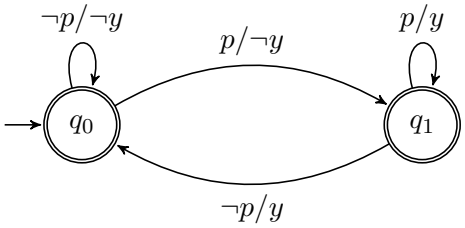


Figure 4.6: Temporal tester for $\ominus p$.

$\diamondsuit_{[0,a]} p$ tester

An important operator, which is used to evaluate the operators *bounded since* ($\mathcal{S}_{\{a,b\}}$) and *bounded eventually in the past* operator ($\diamondsuit_{\{a,b\}}$) is the operator $\diamondsuit_{\{0,a\}}$, shown in Figure 5.3. Before jumping into the general structure of the tester we focus reader's attention to Figure 4.7, where we present an input/output timing diagram produced by a tester for formula $\diamondsuit_{\{0,2\}} p$. In this diagram one can observe that whenever the tester consumes an input symbol p the output of the tester is active in the current time instance and two following instances. We first provide an approach which is enumerating time instances, which can be observed in Figure 4.8. Then, we generalize such model in Figure 4.9 where we can observe a tester which uses a clock variable as a counter. For $a = 2$ it will output three consecutive y values, as required by input/output relation shown in Figure 4.7.

In Figure 4.9 one can notice that location q_2 is used to detect the end of a sequence of consecutive

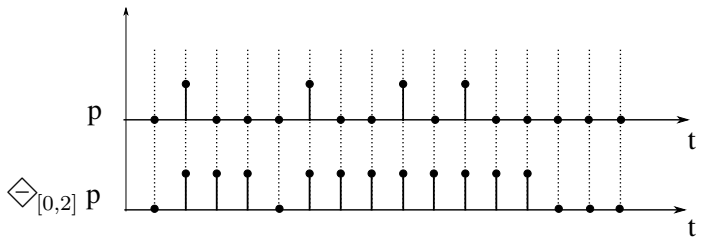


Figure 4.7: Timing diagram produced by a temporal tester for $\Diamond_{[0,2]}p$.

p symbols. When the tester transitions to q_2 it also resets clock c in order to count from 0 to a and output $a + 1$ times y. The clock variable is self-incrementing for every input trace valuation which gets consumed, which also corresponds to a single time instance which passed.

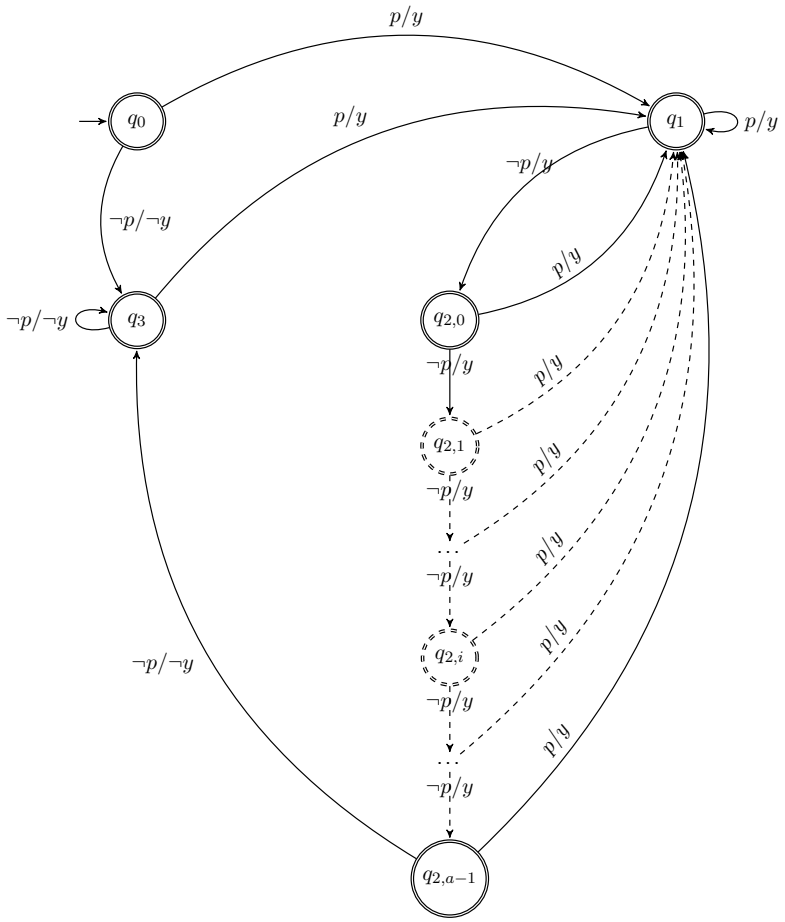


Figure 4.8: Temporal tester for $\Diamond_{[0,a]}p$ with enumerated time.

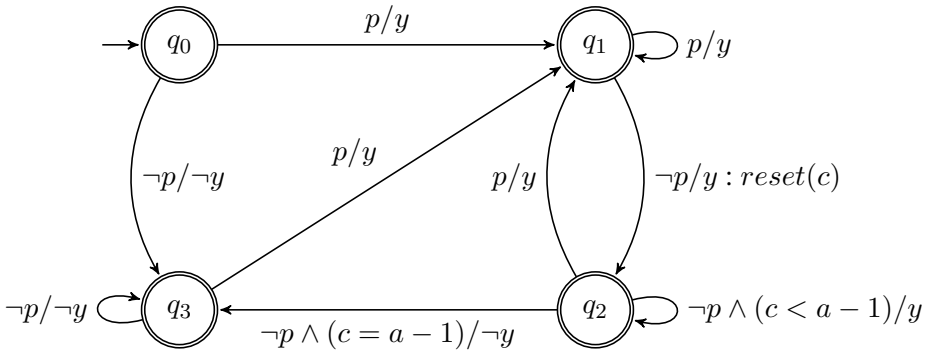


Figure 4.9: Temporal tester for $\diamondsuit_{[0,a]}p$ with symbolic representation of time.

4.4.3 Basic Temporal Testers for Future Time STL

We present temporal testers for future time STL operators: bounded eventually operator ($\diamondsuit_{[0,a]}p$) and unbounded until operator ($p\,\mathcal{U}q$). Temporal tester for $\bigcirc p$ operator was given in Figure 4.3.

$\diamondsuit_{[0,a]}p$ tester

Before presenting a general tester for an arbitrary $a \in \mathbb{N}$ we provide an intuition of our approach by presenting a simpler example of a tester for $\diamondsuit_{[0,2]}p$ in Figure 4.10. From the initial location q_0 there are two choices to make a positive prediction y, in case that we receive p, or in case we see $\neg p$ when we expect to see it in the future and still satisfy the prediction y. The former case is covered with transition $(q_0, p/y, q_1)$. In the latter case, tester must observe p in one of the following two time instances, modeled with q_3 and q_4.

If the tester consumes $\neg p$ in location q_0 it can also make a negative prediction $\neg y$ and transition to q_2. This location is used to describe situations when the tester observes $\neg p$ and must observe $\neg p$ in the following two time instances for the prediction to come true. For this reason there is a self-loop from location q_2. If the tester makes a positive prediction in location q_2, it still has to observe two consecutive $\neg p$ symbols, which is the purpose of q_5 and q_6. It can only reach q_6 with a positive prediction y in location q_2, which means that p needs to be observed in q_6 to fulfill the prediction.

We can generalize the approach following the patterns in $\diamondsuit_{[0,2]}p$ tester structure. Locations q_3 and q_4 and associated transitions cover the case when we the tester requires the prediction to get satisfied in the next a time instances (in this case $a = 2$). With increase of a we need an additional location for each additional time instance where the tester needs to be able to satisfy the positive prediction y. In Figure 4.11 such locations are $q_{3,i}$, where $0 < i < a$.

The locations q_5 and q_6 in Figure 4.10 are used to cover the case when a $\neg y$ prediction was made. It is then required that in the following $a = 2$ time instances, $\neg p$ must be observed for the prediction to come true. To generalize the solution for negative predictions from q_2 we

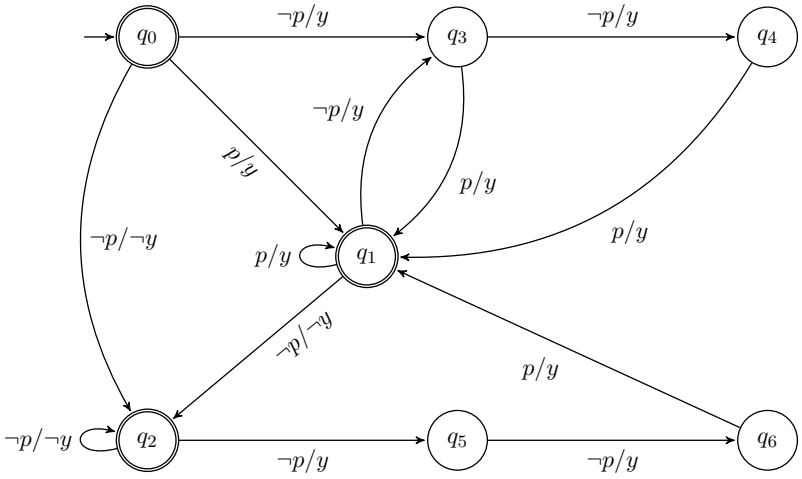


Figure 4.10: Temporal tester for $\Diamond_{[0,2]}p$.

instantiate $a-1$ additional locations, depicted as dashed locations and transitions in the bottom of Figure 4.11.

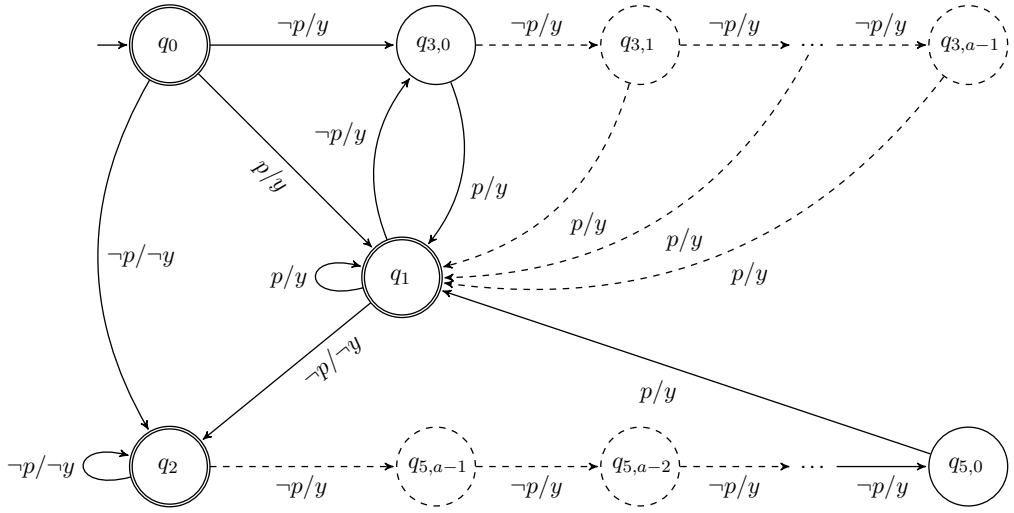


Figure 4.11: Generalized temporal tester for $\Diamond_{[0,a]}p$, for $a \geq 1$, with enumerated time.

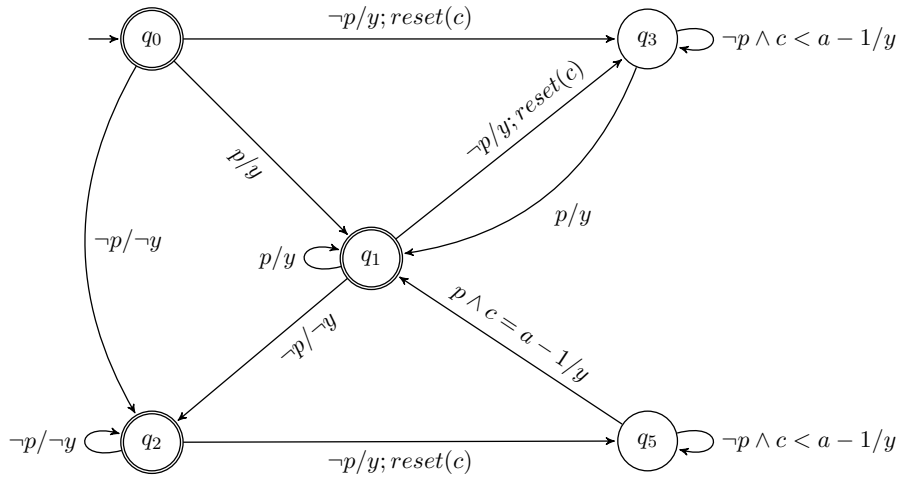


Figure 4.12: Generalized temporal tester for $\Diamond_{[0,a]}p$, for $a \geq 1$; implementation with discrete clocks.

$p\,\mathcal{U}q$ tester

In Figure 4.13 we present the basic temporal tester for unbounded future time operator $p\,\mathcal{U}q$. Tester starts from the initial location q_0 where it can make two optimistic predictions: (1) in case p is observed and q is not observed but expected in the future *or* (2) q is observed in current time instance. Pesimistic prediction $\neg y$ is given in cases when $\neg q$ and we expect that q will never be observed, regardless of p.

In case tester is in location q_0 and sees only p, it transitions to location q_1 where it remains waiting for the q to occur. Since the prediction was y, if we don't actually get to observe q before sequence ends, this input sequence is not acceptable because the prediction was not satisfied. Thus, q_1 is not an accepting location. Other locations are accepting because if a sequence ends any of those state the given prediction is satisfied.

4.4.4 From Temporal Testers to Acceptors

In our approach we use temporal testers to represent our specifications, which we may need to convert to an acceptor automaton. In this section we show an elegant way to convert a specification expressed in temporal tester to an acceptor.

We achieve the conversion by composing the tester, which we want to convert to an acceptor, with a very simple tester shown in Figure 4.14. As previously noted, the difference between an acceptor and a temporal tester is that a tester outputs in every time instance the satisfaction of the trace and all its suffixes, while an acceptor accepts only sequences which are satisfied at $t = 0$. By composing a tester with the one from Figure 4.14, we achieve that the resulting tester keeps only the paths which output y at $t = 0$. Finally, we can simply disregard output variable y in the

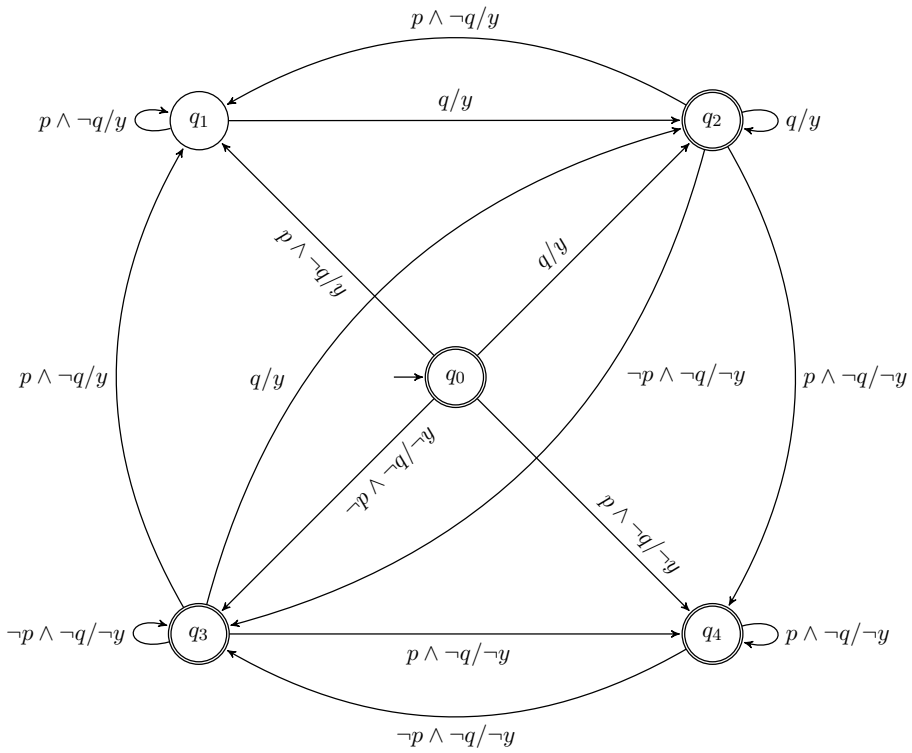


Figure 4.13: Temporal tester for $p \,\mathcal{U}\, q$.

composed tester and obtain the acceptor automaton.

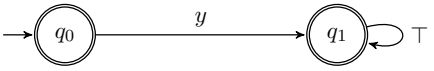


Figure 4.14: Temporal tester used for tester-to-acceptor conversion.

CHAPTER 5

Correctness Monitors in Hardware

The industrial V&V process of chip designs is very involving and time consuming. In the first phase, so-called pre-silicon phase, a simulation is extensively used to search for functional bugs in the design. A natural alternative to computationally demanding simulation is to use dedicated hardware architectures (accelerators) to achieve speedup. To fully leverage hardware acceleration platform, it is necessary to deploy verification blocks implemented in hardware.

During the post-silicon verification phase, longer and computationally expensive scenarios are executed on a design emulation or a real integrated circuit (IC) in the Hardware-in-the-Loop setting [NW14]. During the Hardware-in-the-Loop testing, verification engineers need to execute tests and observe the correctness of the output waveforms. This is a tedious and time-consuming process, lacking automation. Hence, an approach which would improve time-to-market and automate verification is of extreme interest.

As a solution to these problems we propose combining *runtime verification* with *hardware-accelerated* verification methods. Previous chapter provides theoretical foundations for temporal testers, which are used as building blocks for our correctness monitors. In this chapter we explain how to synthesize correctness monitors from high-level formal specifications and implement them in reconfigurable hardware such as FPGA. By introducing FPGA runtime monitors into HiL, we provide a rigorous method for runtime verification of long mixed-signal design executions.

We start from requirements expressed in STL [MN13] with *past* and *bounded future* operators. Due to the digital nature of FPGAs we interpret STL formulas over discrete time. Using rigorous procedures we transform the formulas into a form convenient for translation into basic temporal testers [PZ08]. As we explained in Section 4.4, a tester for an STL formula φ is a *transducer* that observes an execution trace w and outputs $\top$ at time t if and only if w satisfies φ at time t. In order to decide the satisfaction of the formula at runtime, we restrict ourselves to *deterministic* testers, which have a natural translation to sequential circuits. We build a top-level monitor in a compositional fashion from basic temporal testers presented in this chapter.

We implement the entire FPGA monitor synthesis and deployment flow – from formal specifications to the lab environment. We demonstrate our approach on two case studies coming from both digital and mixed-signal domain: the bounded stabilization property, and the Serial Peripheral Interface (SPI) communication protocol. The content of this chapter is based on the work published in [JBG+15].

5.1 Hardware Monitor Synthesis

Before defining the algorithm for hardware monitor synthesis, we briefly recall the syntax and semantics of Signal Temporal Logic, which is already given in Section 3.2.1.

Let $P = \{p_1, \ldots, p_m\}$ be the set of boolean variables and $X = \{x_1, \ldots, x_n\}$ the set of variables defined over non-boolean domain $\mathbb{D}$. We denote by $\pi_e(w)$ the projection of w to $e \in P \cup X$. Given a signal w and its projection $\pi_p(w)$ to some $p \in P$, we say that $\pi_p(w)$ has (Δ, l)-variability if within every interval $[i, i + \Delta - 1]$, the value of p changes at most l times.

The syntax of a STL formula φ over $P \cup X$ is defined by the grammar

$$\varphi \quad := \quad p \mid x \sim u \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I \varphi_2 \mid \varphi_1 \,\mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a, b]$ or $[a, \infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the form $[a, a]$, we will use the notation $\{a\}$ instead.

The semantics of an STL formula with respect to a signal w is described via the satisfiability relation $(w, i) \models \varphi$, indicating that the signal w satisfies φ at the time index i, according to the following definition where $\mathbb{T} = [0, |w|]$.

$$\begin{aligned}
(w,i) &\models p && \leftrightarrow \quad \pi_p(w)[i] = \top \\
(w,i) &\models x \sim u && \leftrightarrow \quad \pi_x(w)[i] \sim u \\
(w,i) &\models \neg\varphi && \leftrightarrow \quad (w,i) \not\models \varphi \\
(w,i) &\models \varphi_1 \vee \varphi_2 && \leftrightarrow \quad (w,i) \models \varphi_1 \text{ or } (w,i) \models \varphi_2 \\
(w,i) &\models \varphi_1 \,\mathcal{U}_I \varphi_2 && \leftrightarrow \quad \exists j \in (i + I) \cap \mathbb{T} : (w,j) \models \varphi_2 \text{ and} \\
& && \qquad \forall i < k < j, (w,k) \models \varphi_1 \\
(w,i) &\models \varphi_1 \,\mathcal{S}_I \varphi_2 && \leftrightarrow \quad \exists j \in (i - I) \cap \mathbb{T} : (w,j) \models \varphi_2 \text{ and} \\
& && \qquad \forall j < k < i, (w,k) \models \varphi_1
\end{aligned}$$

In this section, we present the algorithms for translating STL specifications into deterministic temporal testers that can be synthesized on FPGA hardware. In order to solve this problem, we need to address the following challenges: (1) implement numerical predicates over real-valued signal; (2) provide a translation into memory-efficient monitors from timed properties; and (3) find an appropriate approach for evaluating properties with bounded future operators.

Numerical predicates over real-valued signals are rather an implementation issue - hence we will discuss them in more detail in Section 5.2.

We achieve the translation from real-time STL specifications into memory-efficient monitors by exploiting the *bounded variability* property of STL timed operators. Similarly to (Δ, l)-variable signals, we say that a temporal operator has (Δ, l)-variability if within any interval of size Δ its satisfaction value can change l times at most. Instead of memorizing the satisfaction status of the formula at every cycle, this property allows us to record only the changes in the satisfaction of the formula.

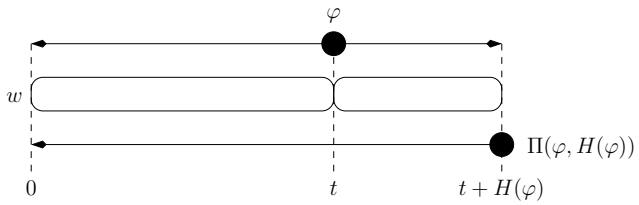


Figure 5.1: From bounded future φ to past $\Pi(\varphi, b)$.

Finally, we handle monitoring of STL specifications with bounded future operators by transforming them into *equisatisfiable* STL formulas that contain only past operators. This transformation eliminates the "predictive" aspect of the original formula by changing the time direction from future to past. This is possible because the formula refers only to a statically pre-computable bounded future horizon. Hence, its evaluation can be delayed until the horizon is reached. Figure 5.1 illustrates the transformation Π from a bounded future STL formula φ into its equisatisfiable past STL counterpart $\Pi(\varphi, H(\varphi))$, where $H(\varphi)$ is the time horizon of φ.

5.1.1 Monitoring Past-Time STL Specifications

In order to efficiently evaluate past-time STL operators, we use instead the following equivalences to first simplify the formulas. We remind the reader about those rewriting rules, which were already shown in Section 4.4.2:

$$
\begin{aligned}
\varphi_1 \, \mathcal{S}_{[0,b]} \varphi_2 &= (\varphi_1 \, \mathcal{S} \varphi_2) \wedge \diamondminus_{[0,b]} \varphi_2 \\
\varphi_1 \, \mathcal{S}_{[a,b]} \varphi_2 &= \boxminus_{[0,a-1]} (\varphi_1 \wedge \ominus (\varphi_1 \, \mathcal{S} \varphi_2)) \wedge \diamondminus_{[a,b]} \varphi_2 \\
\boxminus_{[a,b]} \varphi &= \neg \diamondminus_{[a,b]} (\neg \varphi) \\
\diamondminus_{[a,b]} \varphi &= \diamondminus_{\{a\}} \diamondminus_{[0,b-a]} \varphi
\end{aligned}
$$

As a result of this reduction, we only need to build testers for the $\ominus$, $\mathcal{S}$, $\diamondminus_{[0,b]}$ and $\diamondminus_{\{a\}}$ temporal operators. In what follows, we describe the algorithms for building basic temporal testers for each of these operators.

$\ominus$ and $\mathcal{S}$ operators [HR02]

The temporal tester for $\psi = \ominus\varphi$, shown in Figure 4.6,must satisfy $\neg\psi'$ in the first step, and in every following step executes the transition labeled by $\psi' \leftrightarrow \varphi$.

We notice that $\psi = \varphi_1 \mathcal{S} \varphi_2$ is equivalent to the formula $\varphi_2 \vee (\varphi_1 \wedge \ominus(\varphi_1 \mathcal{S} \varphi_2)$. It follows that the temporal tester for ψ must satisfy $\varphi_2' \leftrightarrow \psi'$ is the first step and $\psi' \leftrightarrow (\varphi_2' \vee (\varphi_1' \wedge \psi))$ in every following step, which is shown in Figure 4.5. Both temporal testers require a single memory element.

$\diamondminus_{[0,a]}$ operator

The temporal tester, shown in Figure 4.9 for $\psi = \diamondminus_{[0,a]}\varphi$ uses a single counter c bounded by $a+1$ to implement a discrete time clock[1] and works as follows. The tester observes the satisfaction of φ over time and moves through its locations, generating the output that follows the satisfaction relation of the operator. Whenever the tester observes the satisfaction of φ, the output of the tester must trivially satisfy ψ. If the tester detects a falling edge in the satisfaction of φ, the counter c is reset to 0. As long as φ is violated and the counter c is smaller or equal to a, the counter is incremented and the output must still satisfy ψ - the property is satisfied since the last observation where φ was true lies within the previous $[0, a]$ interval. If the tester still observes violation of φ while the counter c reaches $a + 1$, it must satisfy the output $\neg\psi$, indicating the violation of the formula. The temporal tester for $\diamondminus_{[0,a]}$ requires a single $\lceil \log_2(a) \rceil$-bit counter.

$\diamondminus_{\{a\}}p$ tester with discrete clocks

We first note that the operator $\diamondminus_{\{a\}}\varphi$ simply shifts the satisfaction of φ by a time steps, i.e. $(w, i) \models \varphi$ if and only if $(w, i + a) \models \diamondminus_{\{a\}}\varphi$. There is a very simple implementation of this formula by observing the following equivalence: $\diamondminus_{\{a\}}\varphi = \underbrace{\ominus\ominus\ldots\ominus}_{a}\varphi$

Despite its simplicity, this direct implementation for the $\diamondminus_{\{a\}}\varphi$ requires a memory registers and may not be optimal if a is large and φ has bounded variability. Hence, we sketch an alternative algorithm, formalized in Section 4.4.2 and illustrated in Figure 5.2, for building a temporal tester for $\psi = \diamondminus_{\{a\}}\varphi$ when φ has (Δ, l)-variability.

Instead of recording the last a values of φ, we instead memorize only the relative times of the last l changes in φ by using discrete counters. In the first a steps, the tester must satisfy the output $\neg\psi$. Whenever a change is observed in φ, a fresh discrete counter c is reset to 0 and incremented in the next a steps. When the counter reaches a, the tester enforces the same change in ψ. The number of active counters required at any point in time is dependent on the variability of the input signal and the bound a - whenever an active counter reaches a, it is deactivated and can be reused. The implementation of this algorithm with discrete clocks requires at most $\lceil \frac{a \cdot l}{\Delta} \rceil \cdot \lceil \log_2 a \rceil$ bits.

[1] From now on, we refer to the clocks from timed automata terminology as counters, in order to avoid confusion with system clock signals in hardware.

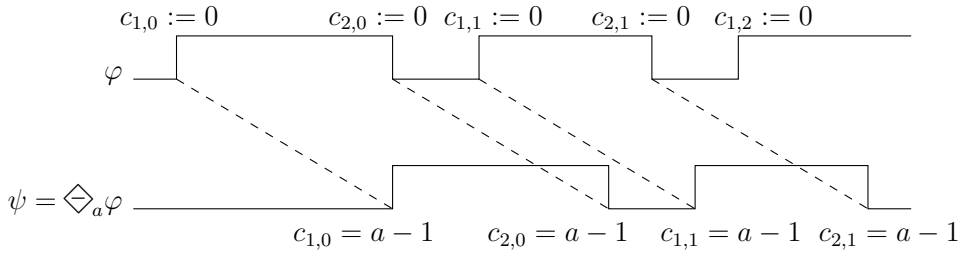
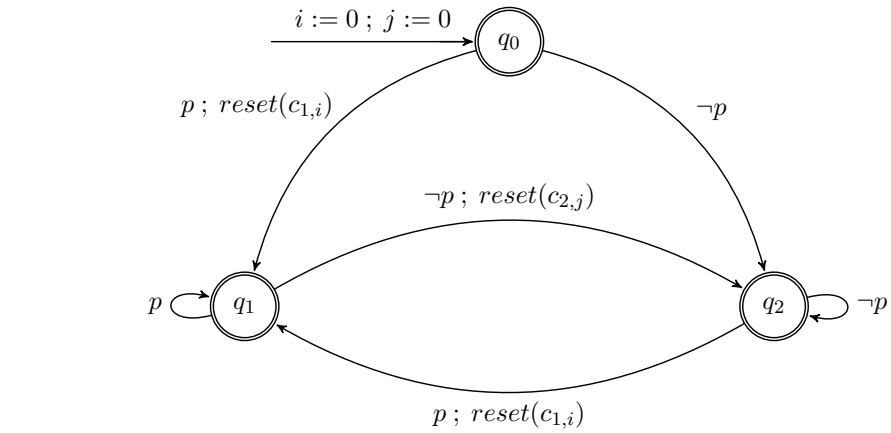
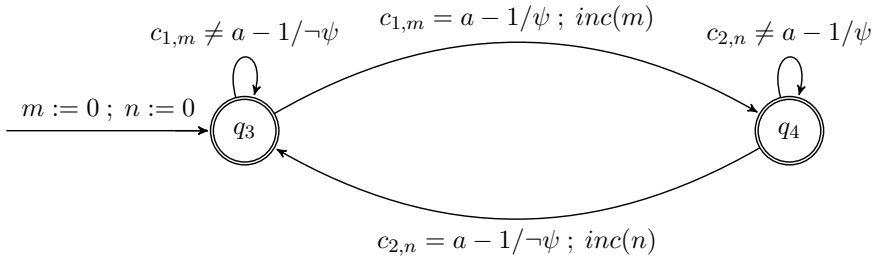


Figure 5.2: Computing $\diamondsuit_a \varphi$ with discrete clocks.

We define an efficient implementation of time delay operator as a temporal tester with discrete clocks and an automaton which detects signal edges, as shown in Figure 5.3.





$$inc(index) \equiv \left\{ index \ := \ (index + 1)\ mod \left\lceil \frac{a \cdot l}{2 \cdot \Delta} \right\rceil \right\}$$

$$reset(c_{k,index}) \equiv \left\{ c_{k,index} \ := \ 0\ ;\ inc(index) \right\}$$

Figure 5.3: General structure of temporal tester $\diamondsuit_{\{a\}} p$ for inputs of (Δ, l) variability.

We follow divide-and-conquer strategy for building $\diamondsuit_{\{a\}}$ tester. We separate responsibilities: one automaton detects input symbol changes and a temporal tester drives the output variable y. The change detection automaton does not set any output variables. It uses two circular indices i and j, to identify a fresh variable in two groups of clock variables, which are dedicated to positive and negative edges of the signal p. The automaton resets a fresh clock variable $c_{1,i}$ for each positive edge of input signal p. Similar is done upon detecting negative edge with a clock variable $c_{2,j}$.

Temporal tester in the lower part of Figure 5.3 is used to drive output variable y when an edge was detected a time units in the past, which is detected with a dedicated clock variable. Whenever a clock variable reaches $a-1$, the tester increments the respective index, in order to identify the next clock variable for consideration. Tester location q_3 is used to drive positive edge of y and q_4 is used to drive the negative edge. It uses two separate circular indices m and n to address clock variables.

We remind the reader that we use $\diamondsuit_{[a,b]}\varphi = \diamondsuit_{\{a\}}\diamondsuit_{[0,b-a]}\varphi$ to decompose the monitoring of arbitrary *once* operators. The satisfaction signal $\psi = \diamondsuit_{[0,b-a]}\varphi$ has the $(b-a+1,2)$-variability. Hence, the algorithm that uses counters needs $\lceil \frac{2\cdot a}{b-a+1} \rceil \cdot \lceil \log_2 a \rceil$ bits to implement the operator. It follows that the direct implementation with the shift registers is more optimal when the input signal has high variability, while the algorithm with the counters works better with low-varying signals. For instance, the direct implementation of $\diamondsuit_{\{500\}}p$ requires 500 registers, while the algorithm with counters needs 500 counters, each having $\lceil \log_2 500 \rceil = 9$ bits. On the other hand, the direct implementation of $\diamondsuit_{[500,1000]}\varphi = \diamondsuit_{\{500\}}\diamondsuit_{[0,500]}\varphi$ requires 500 bits, while the algorithm using counters needs only $2 \cdot \lceil \log_2 500 \rceil = 18$ bits. The optimal implementation choice for the $\diamondsuit_{\{a\}}$ operator can be easily automated by performing a syntactic analysis of the formula.

Correctness conditions for $\diamondsuit_{\{a\}}p$ tester As previously noted, clock variables are incremented with every step. For this reason, it is necessary to initialize them to maximal value. It is necessary to initialize circular counters i, j, m, n to 0. When we reset a clock variable we must increment the respective index in the modulo of $\lceil \frac{a \cdot l}{2 \cdot \Delta} \rceil$. Transitions which reset a clock variable when an input change is detected are allowed to reset only a single variable, either from clock variable group $c_{1,i}$ or $c_{2,j}$. Due to the specific architecture, we do not use this tester directly in parallel composition with other testers.

5.1.2 From Bounded Future to Past STL Specifications

We now address the problem of developing monitors for bounded future STL formulas. The challenge for monitoring such formulas comes from the fact that the satisfaction at time index i depends on the inputs that will become available only in some future (but bounded) horizon $[i, i+h]$. The bound of the future horizon can be syntactically computed from the specification, as shown in [DSS+05b, MNP07]. In order to solve this problem, we adapt the *pastification* procedure from [MNP07], which transforms the bounded future specification φ into an *equisatisfiable* past specification ψ that is evaluated with the delay h. Formally, the two formulas are related as follows: $(w,i) \models \varphi$ iff $(w, i+h) \models \psi$, where h is the *temporal depth* of φ. The temporal depth

h is the maximum size of the input w suffix $[i, i+h]$ needed to determine the satisfaction of φ at the time index i. Formally, the temporal depth $h = H(\varphi)$ of φ is the syntax-dependent upper bound on the actual depth of the formula and is computed using the following recursive definition:

$$\begin{aligned}
H(p) &= 0 \\
H(\neg\varphi) &= H(\varphi) \\
H(\varphi_1 \vee \varphi_2) &= \max\{H(\varphi_1), H(\varphi_2)\} \\
H(\bigcirc\varphi) &= H(\varphi) + 1 \\
H(\varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2) &= b + \max\{H(\varphi_1) - 1, H(\varphi_2)\}
\end{aligned}$$

Consider the formula $\varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2$ and let us interpret φ_1 as the satisfaction signal of its first argument. An arbitrary interval $I = [i, i+b]$ admits a minimal partition $I_1, \ldots, I_n$ such that in every partition I_i the value of φ_1 is constant and in every two adjacent partitions the value of φ_1 differs. It is simple to see that the maximum number of partitions for such arbitrary I is bounded by $z = \lceil \frac{b}{2} \rceil$. We can then decompose φ_1 into z signals $\varphi_1^1, \ldots, \varphi_1^z$ such that $\varphi_1 = \bigcup_{i \in [1,z]} \varphi_1^i$, $\varphi_1^i \wedge \varphi_1^j$ is false for every $i \neq j$ and each φ_1^i has $(b, 2)$-variability with at most one uniform subinterval of $[i, i+b]$ where φ_1^i is true. This decomposition is achieved in practice by letting φ_1^i rise and fall only on the j^{th} rising and falling of φ_1, where $j = i \mod b$. After the decomposition, we have the following simple equivalence:

$$\begin{aligned}
(w, i) \models \varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2 \quad &\leftrightarrow \quad (w, i) \models \textstyle\bigvee_{i=1}^{z} \varphi_1^i \,\mathcal{U}_{[a,b]} \varphi_2 \\
&\leftrightarrow \quad (w, i) \models \textstyle\bigvee_{i=1}^{z} (\bigcirc\varphi_1^i \wedge \diamondsuit_{[a-1,b-1]} (\varphi_1^i \wedge \bigcirc\varphi_2))
\end{aligned}$$

The *pastification* operation Π on the STL formula φ with past and bounded future and its bounded horizon $d = H(\varphi)$ is defined recursively as follows:

$$\begin{aligned}
\Pi(p, d) &= \diamondsuit_{\{d\}} p \\
\Pi(\neg\varphi, d) &= \neg\Pi(\varphi, d) \\
\Pi(\varphi_1 \vee \varphi_2, d) &= \Pi(\varphi_1, d) \vee \Pi(\varphi_2, d) \\
\Pi(\bigcirc\varphi, d) &= \Pi(\varphi, d-1) \\
\Pi(\diamondsuit_{[a,b]}\varphi, d) &= \diamondsuit_{[0,b-a]} \Pi(\varphi, d-b) \\
\Pi(\varphi_1 \,\mathcal{U}_{[a,b]} \varphi_2, d) &\leftrightarrow \textstyle\bigvee_{i=1}^{z} \Pi(\varphi_1^i \,\mathcal{U}_{[a,b]} \varphi_2, d) \\
&\leftrightarrow \textstyle\bigvee_{i=1}^{z} \Pi(\bigcirc\varphi_1^i \wedge \diamondsuit_{[a-1,b-1]} (\varphi_1^i \wedge \bigcirc\varphi_2), d)
\end{aligned}$$

We note that for monitoring $\Pi(p \,\mathcal{U}_I q, d)$ we first need to decompose p into $\frac{d}{2}$ signals, each having $(d, 2)$-variability. Hence, for every decomposed signal, we must use at most 2 active counters of size $\lceil \log_2 d \rceil$. As a result, the monitor for $\Pi(p \,\mathcal{U}_I q, d)$ requires $d \cdot \lceil \log_2 d \rceil$ registers. It follows that the monitor for an arbitrary bounded future formula requires at most $d \cdot \lceil \log_2 d \rceil \cdot |\varphi|$ registers.

5.1.3 Correctness Monitors from TRE specifications

Although STL is our language of choice, it is important to consider algorithms for obtaining hardware monitors using Timed Regular Expressions. We use our monitors in a hardware setting, where signals are sampled in time. Therefore, we interpret our specifications over discrete time. Pnueli and Zaks [PZ08] have shown that satisfaction of a Property Specification Language expression can be evaluated using temporal testers. In a discrete setting, it is possible to reduce TRE to Sequential Extended Regular Expressions which are a part of the PSL and which translate directly to temporal testers.

5.2 Hardware Implementation

We implement our monitors on Zynq7020 All Programmable SoC, a configurable hardware platform. Its main parts are the Processing System and Programmable Logic. The Programmable Logic primarily consists of configurable logic blocks Configurable Logic Block (CLB) that contain lookup tables lookup table (LUT) and flip-flops flip-flop (FF). Such blocks can be arbitrarily connected by programming desired connections in Switch Matrix. Each CLB can implement combinatorial net in LUTs and sequential circuits using FFs. The Zync7020 device contains in total 53200 LUTs and 106400 FFs. It also contains dedicated physical components for specific purpose such as block RAM (bRAM), specific SLICEM blocks for efficient implementation of shift registers and an internal Analog-to-Digital Converter (ADC). We now provide a high-level overview of our implementation flow showing the details of the monitor synthesis.

5.2.1 Implementation Phases

The implementation, illustrated in Figure 5.4, consists of three phases: (1) pre-processing; (2) code generation; and (3) FPGA flow.

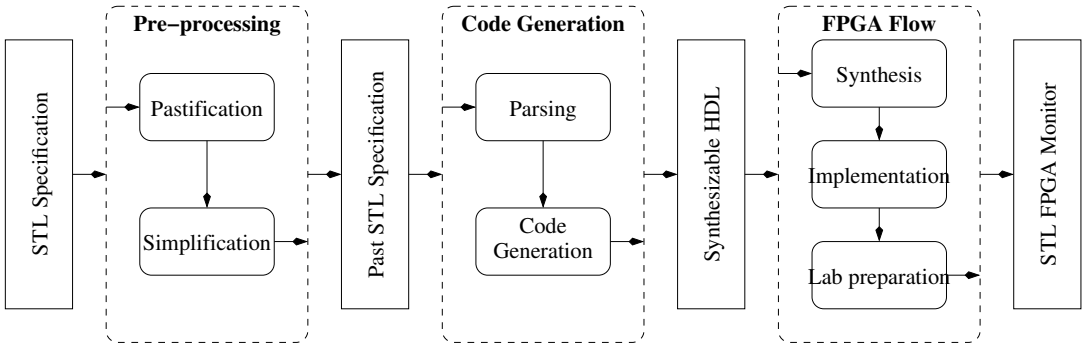


Figure 5.4: Implementation flow.

In the pre-processing phase, we first translate the bounded-future STL formula to its equisatisfiable past counterpart. We then simplify the resulting past formula into an equivalent one which uses only basic $\ominus$, $\mathcal{S}$, $\diamondsuit_{[0,a]}$ and $\diamondsuit_{\{a\}}$ operators. This phase follows the algorithms and rewriting rules from Section 5.1.

For the code generation phase, we deploy a parser in Java with a specific algorithm to extract information about input signals, operators and composition of the formula. Then, the algorithm uses a hash map to convert the parse tree of the formula into a Directed Acyclic Graph (DAG) and eliminate duplicate sub-formulas. From this DAG we generate a deterministic monitor in a compositional way resulting in synthesizable Verilog code.

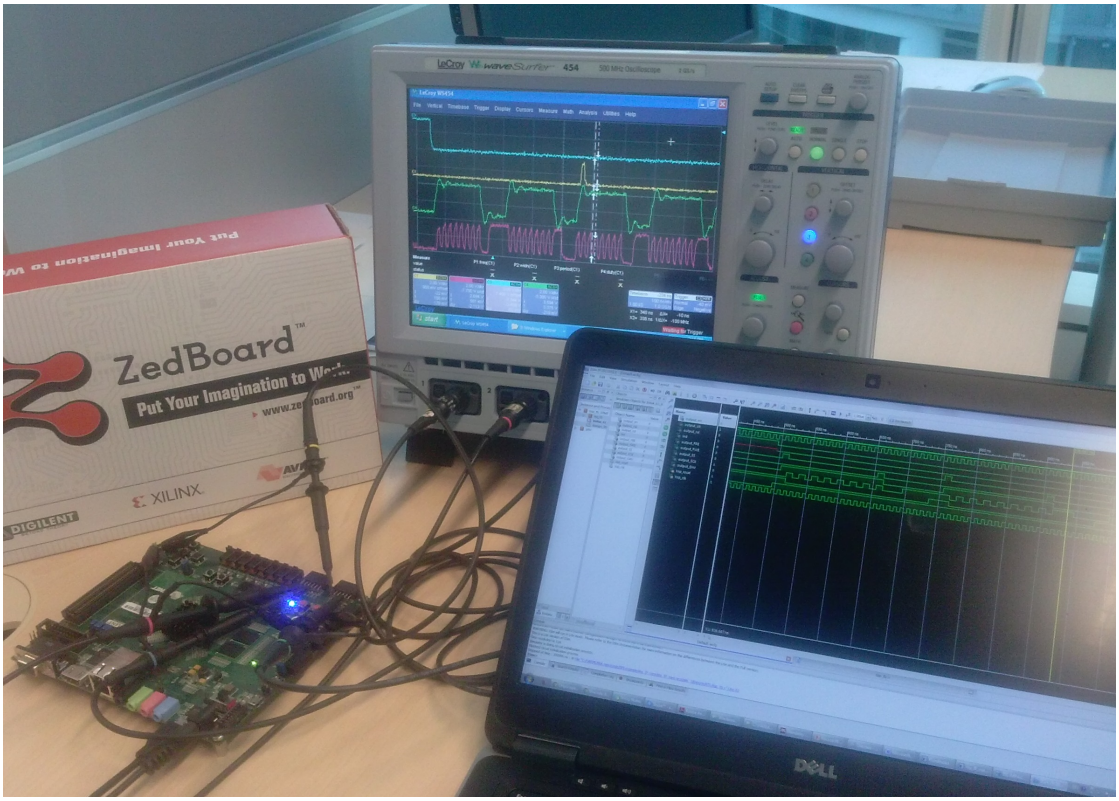


Figure 5.5: STL monitor running in real time in lab environment.

In the final phase, we follow the classical FPGA development flow in order to map the synthesizable monitor to the actual hardware. We use PlanAhead 14.7 and Vivado 2014.4 tools to perform the following steps: (1) synthesis; (2) implementation; and (3) bit-stream generation. We program the Zynq7020 device with generated bitstream and connect it in the lab environment to an oscilloscope to probe the signals of interest. In the digital case, we route the internal signals out from the Zynq7020 - it is thus sufficient to monitor only external pins in the lab. In the analog case, the lab evaluation is more difficult. We created an *internal logic analyzer* block by deploying Xilinx IP core able to record quantized analog signals at runtime. In addition, we dedicate a specific external pin to alert the user on property violation. In Figure 5.5 we show a running FPGA monitor with different signals displayed on an oscilloscope.

5.2.2 Numerical Predicates

In our implementation, we consider the case where the analog device is an external component connected to the STL hardware monitor. We use the Xilinx analog-to-digital converter (XADC) block that is integrated to the Xilinx IP to convert the analog signal to a digital (quantized) one. We then implement synthesizable Verilog code that compares the quantized value of the signal to the threshold from the STL formula and outputs the Boolean signal for further use. We note that the precision of this approach is mainly limited by the characteristics of the ADC: (1) the frequency at which it operates; (2) its resolution; and (3) the maximum voltage that it is able to process. The XADC block from the Xilinx IP operates at a maximal frequency of 1MHz, has a resolution of 12 bits and is able to process signals that have the maximal amplitude of 1V.

5.2.3 Monitor Integration

Monitors implemented on FPGAs are self-contained hardware units. There are different ways to integrate such monitors with the SUT. We first consider the case when both the SUT and monitors are implemented on the same FPGA programmable logic. This architecture is depicted in Figure 5.6. In this situation, both SUT and the monitor are purely digital blocks. The monitor non-intrusively observes relevant SUT signals by connecting to the SUT interface. Based on the observations, the monitor generates a verdict. Both the signals of interest and the verdict can be routed out of the FPGA by making the appropriate connections to the FPGA pin-outs and displayed on an oscilloscope. The monitor and the SUT operate at the same frequency, limited by the maximum achievable frequency of the FPGA. The architecture allows usage of either internal or external clock generator. This architecture can be used to connect the monitor to digital design emulation. In this case, the SUT is either an emulation of a purely digital design or a digital approximation of an analog design.

We also present an alternative architecture, in which the monitor implemented on FPGA is connected to an external device. This architecture is presented in Figure 5.7. In this case, the SUT is an external digital, analog or mixed-signal device. The monitor connects to external digital signals via FPGA pin-outs. The analog signals cannot be directly connected to the monitor - we use the ADC block to quantify the input signal and convert it to the digital domain, as explained in Section 5.2.2. The external visualization of signals of interest and the verdict is done via the FPGA pin-outs, as in the previous case. The typical use case for this architecture is the real-time monitoring of the real mixed-signal devices in the post-silicon verification phase. The main limitation of this architecture is the performance of the AD converter and the inaccuracies introduced during AD conversion.

5.3 Evaluation

We now present the experimental evaluation of our framework. We translate several classes of STL formulas and collect the data on hardware resources allocated to the resulting monitors. For each formula we generate a hardware monitor and report the number of flip-flops flip-flop (FF) and lookup tables lookup table (LUT) it consumes and the maximal achievable clock frequency

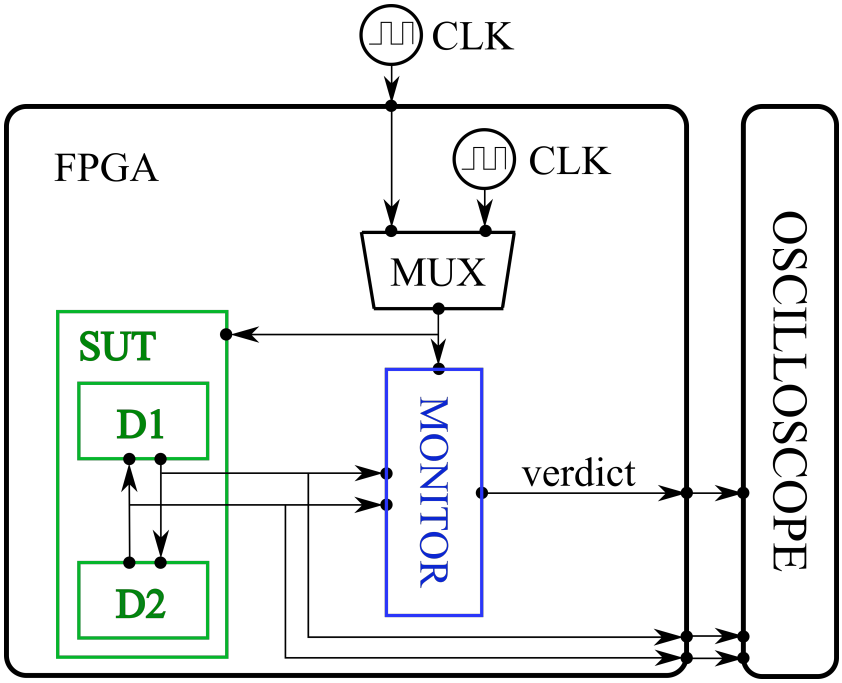


Figure 5.6: Self-contained architecture - a monitor and a SUT on the same FPGA.

STL Formula	#FF	#LUT	MHz
$\Box(p\,\mathcal{S}(q \wedge \boxminus r))$	2	3	346
$\Box(p\,\mathcal{S}(q \wedge \ominus(r\,\mathcal{S}q)))$	2	3	346
$\Box(p \rightarrow (q\,\mathcal{S}r))$	1	2	345
$\Box(\diamondminus p \wedge \diamondminus q \wedge \diamondminus r \wedge \diamondminus s \wedge \diamondminus t)$	5	7	339
$\Box(p \rightarrow (q\,\mathcal{S}(\boxminus r \vee \boxminus s)))$	3	4	346

Table 5.1: Resource benchmark results for untimed past STL formulas. We report on the number of flip-flops consumed by the resulting monitor (#FF), number of lookup tables consumed (#LUT) and achieved maximum circuit frequency in MHz.

in MHz. We note that the maximal achievable clock frequency depends in general on the size of the device implemented on the FPGA, since large resource usage requires more complex routing and internal signal propagation.

In our first experiment we consider only the untimed past fragment of STL. The results of this experiment, shown in Table 7.1, clearly indicate that this fragment of the logic admits monitors with a very small footprint.

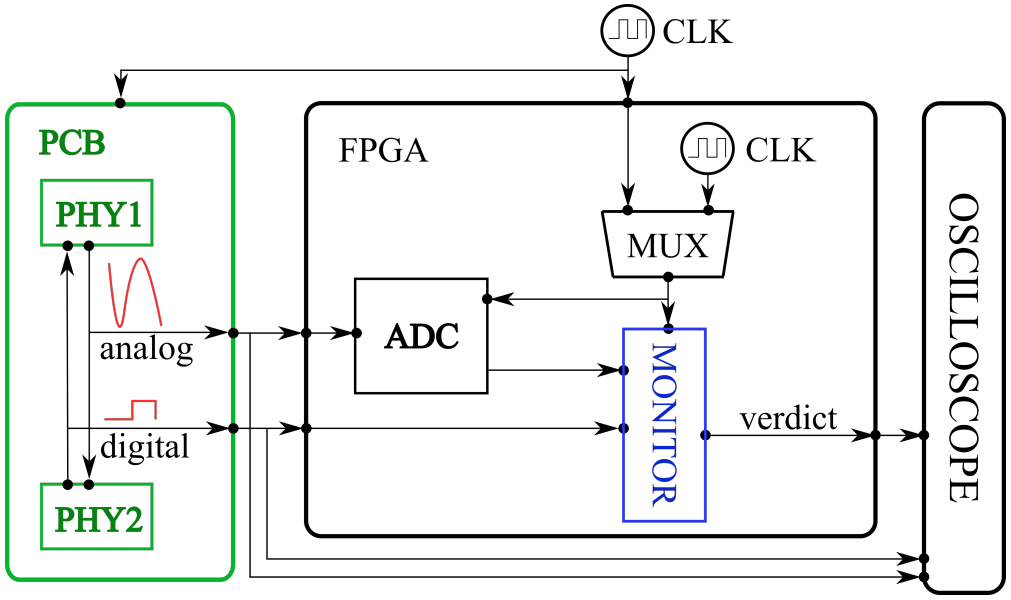


Figure 5.7: Integration of a FPGA monitor to an external device.

STL Formula	b=50			b=500			b=5000		
	#FF	#LUT	MHz	#FF	#LUT	MHz	#FF	#LUT	MHz
$\square(p \leftrightarrow q\,\mathcal{S}_{[0,b]}r)$	13	18	346	16	26	346	20	36	346
$\square(p \leftrightarrow q\,\mathcal{S}_{[b/2,b]}r)$	42	136	329	61	193	321	84	321	317
$\square(p \leftrightarrow q\,\mathcal{S}_{[9b/10,b]}r)$	121	423	286	213	727	250	309	1446	213

Table 5.2: Resource benchmark results for bounded past STL formulas. We report on the number of flip-flops (#FF), lookup tables consumed (#LUT) and achieved maximum circuit frequency in MHz.

In our second experiment (see Table 7.2) we evaluate the scalability of our approach w.r.t. a class of formulas from real time past fragment of Signal Temporal Logic. In particular we explore formulas containing the $\mathcal{S}_{[a,b]}$ and study the impact of bound variations to the resulting size of the monitor. In this experiment all our monitors use the counters algorithm to record real time data. We can first observe that for formulas with lower bound equal to zero, the resulting monitors have a small resource consumption and are insignificantly affected by the size of the upper bound b. This is due to the fact that these monitors require a single counter of size that is logarithmic in b. We can also see that the size of the monitors for this class of formulas are mostly affected by the ratio between the lower and the upper time bounds. The variability of the $\mathcal{S}_{[a,b]}$ operator, which is a function of the aforementioned ratio, directly affects the resource consumption of the monitor.

STL Formula	Register Buffer			Counters Algorithm		
	#FF	#LUT	MHz	#FF	#LUT	MHz
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,50]}r))$	1614	584	178	811	1807	155
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,100]}r))$	5915	1548	202	1751	4069	110
$\square(p \leftrightarrow q\,\mathcal{U}_{[0,200]}r))$	22006	3476	177	3500	7630	103
$\square(p \leftrightarrow q\,\mathcal{U}_{[25,50]}r))$	1588	532	208	779	1747	169
$\square(p \leftrightarrow q\,\mathcal{U}_{[50,100]}r))$	5880	1463	191	1666	3912	105
$\square(p \leftrightarrow q\,\mathcal{U}_{[100,200]}r))$	22001	3484	156	3620	7916	120

Table 5.3: Resource benchmark results for bounded future STL formulas. We report on the number of flip-flops (#FF), lookup tables consumed (#LUT) and achieved maximum circuit frequency in MHz.

The third experiment has two purposes. On one hand we evaluate the size of monitors for bounded future STL. On the other hand we compare the resource requirements of two time event recording algorithms: straightforward register buffering and the counters algorithm. The results are shown in Table 5.3.

We first observe that handling future operators can be very costly. This result is as expected - checking online future formulas requires the determinization of the underlying monitors. In our approach we do this step at the syntactic level using the pastification procedure. We observe that the monitors for future formulas are in particular sensitive to the size of their future horizon. Regarding the two event recording algorithms we can notice that counters algorithm prevents the explosion of memory demands. On the other hands it uses more LUTs than the register buffer. This is due to arithmetical operations performed on the counters. We can also see that the maximum achievable frequency for the register buffering approach is in general higher than the one for the counters approach. We suspect that this results from more combinatorial operations done in the counters algorithm. As a consequence, the monitor computes more operations during a single clock cycle, thus prolonging its minimal duration.

5.3.1 Mixed Signal Bounded Stabilization

In first case study, we adapt the mixed signal bounded stabilization property from Example 1. We monitor a system generating a boolean *trigger* and an analog signal x. Upon the rising edge of the trigger, the analog signal is allowed to get unstable, but is required to stabilize within a specified finite time horizon. The informal specification in natural language is stated as follows: "whenever the trigger is on its rising edge, the analog signal x is allowed to take an arbitrary amplitude, but within $1ms$, the signal must take an amplitude smaller or equal than 0.5V and continuously remain below that threshold for at least $500\mu s$".

We first model the analog signal x with perturbation by using an external pulse generator. We generate a 5 kHz sinusoidal signal in the range of 250 mV -800 mV modulated with a 250 Hz down ramp with 33% maximal decline, in order to obtain damped sine oscillations. This analog signal operates at lower frequency than our ADC. We reduce the clock frequency of the monitor to 200 kHz and thus we avoid unnecessary computations.

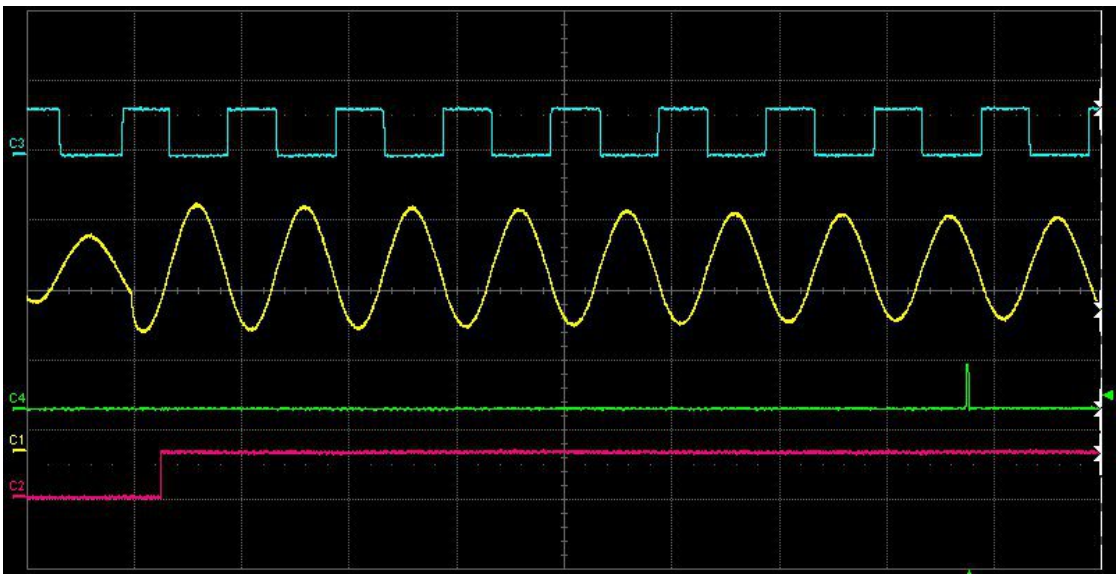


Figure 5.8: Stabilization property signals observed in our lab environment.

The formalized requirement in STL and its past counterpart are as follows:

$$\square(\uparrow \mathit{trigger} \rightarrow \diamondsuit_{[0,200]}\square_{[0,100]}(x \leq 0.5))$$
$$\square(\diamondsuit_{\{300\}} \uparrow \mathit{trigger} \rightarrow \diamondsuit_{[0,200]}\boxminus_{[0,100]}(x \leq 0.5)).$$

Figure 5.8 shows the screenshot from the oscilloscope on which we observe the monitored results, delayed by 300 clock cycles due to formula pastification. The yellow waveform represents the analog input signal x, while the blue signal is the predicate $x \leq 0.5$. The red signal is the boolean *trigger* and finally the green waveform is the assertion verdict, where the peak represents the violation of the property. We can see from Figure 5.8 that the property is violated because the signal x stabilizes too slowly and does not remain below the 0.5 threshold for sufficient time.

5.3.2 Serial Peripheral Interface

Serial Peripheral Interface (SPI) bus is a de facto standard serial communication interface specification used for short distance communication, primarily in embedded systems. SPI provides full duplex communication with a master-slave architecture.

Regular SPI interface consists of the following signals: *SPI clock* (*sck*), *slave select* (*ss*), *master input slave output* (*miso*) and *master output slave input* (*mosi*). In this case study, we use a digital

master device that has additional control signals such as *data available* (*dav*) and *FIFO read enable* (*fre*).

We create a synthesizable TB in Verilog to stimulate the SPI device with several scenarios. In addition to regular scenarios we inject errors to trigger property violations. We synthesize both the test bench and the SPI master on a single Zync 7020 device. The user controls the emulation with a programmed physical interrupt.

We consider two SPI requirements: *nested pulse* and *clock division*. Nested pulse property specifies mutual interaction of control signals *dev* and *fre* - it states that once the FIFO read is done, there should be no more data available for the transfer. Clock division property requests that *sck* is slower than the system clock by a rate determined by the *rate* signal value. We fix *rate* to 0, meaning that *sck* must be twice slower than the system clock. In addition, it is required that the *sck* clock is toggling only when *ss* is asserted. We specify these two requirements in natural language as follows:

1. *fre* pulse must be on its rising edge in the last cycle of the *dav* pulse;
2. Every rising edge of *sck* is (1) the first one within the current asserted interval of *ss* or (2) follows a falling edge of *sck* in the previous cycle. The symmetric property must hold for every falling edge of *sck*.

We formalize these requirements in past STL as follows:

$$\square(\downarrow dav \rightarrow (\downarrow fre \wedge \ominus \uparrow fre)$$
$$\square((\uparrow sck) \rightarrow (\neg sck \ \mathcal{S} \ \uparrow ss \vee \ominus \downarrow sck))$$
$$\square((\downarrow sck) \rightarrow (sck \ \mathcal{S} \ \uparrow ss \vee \ominus \uparrow sck))$$

The resulting monitors for these properties operate at the Zynq system clock frequency of 100MHz and do not additionally constrain the speed of the SUT.

We can observe *nested pulse* property in Figure 5.9. The *data available* signal is shown in red color, and *fifo read enable* in green color. Due to the timing skew between the pulses of these two signals, monitor correctly detects *nested pulse* property violation. This is shown in Figure 5.9 with the violation signal pulse, depicted in yellow.

Figure 5.10 shows the relevant clock division property signals on an oscilloscope in our lab environment. The blue signal represents the SPI master device reset signal and it marks the start of the emulation. The green signal denote the SPI slave select *ss* and the red one represents the SPI clock *sck*. Finally, the yellow signal shows the verdict for the property, where the peak in the signal denotes its violation. As we can see from the Figure 5.10, the property $\square((\uparrow sck) \rightarrow (\neg sck \ \mathcal{S} \ \uparrow ss \vee \ominus \downarrow sck))$ fails because at the moment of its violation (peak in the yellow signal), the first rising edge of *sck* arrives slightly before the rising edge of the new *ss* window, thus falsifying both disjuncts $\neg sck \ \mathcal{S} \ \uparrow ss$ and $\ominus \downarrow sck$ of the right-hand side of the implication.

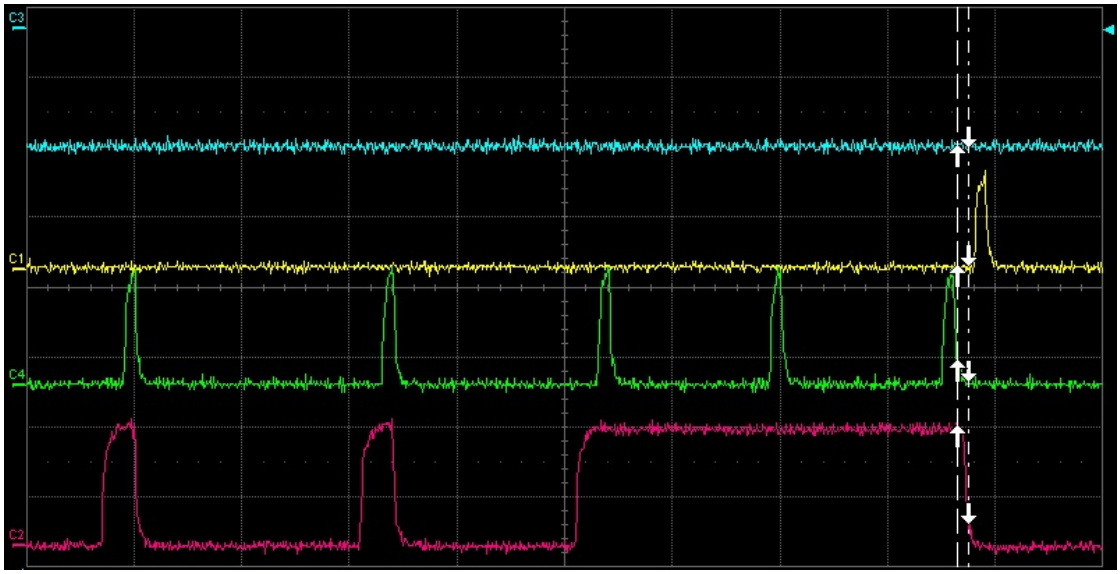


Figure 5.9: Nested pulse property for *data available* (depicted in red) and *fifo read enable* (depicted in green) signals.

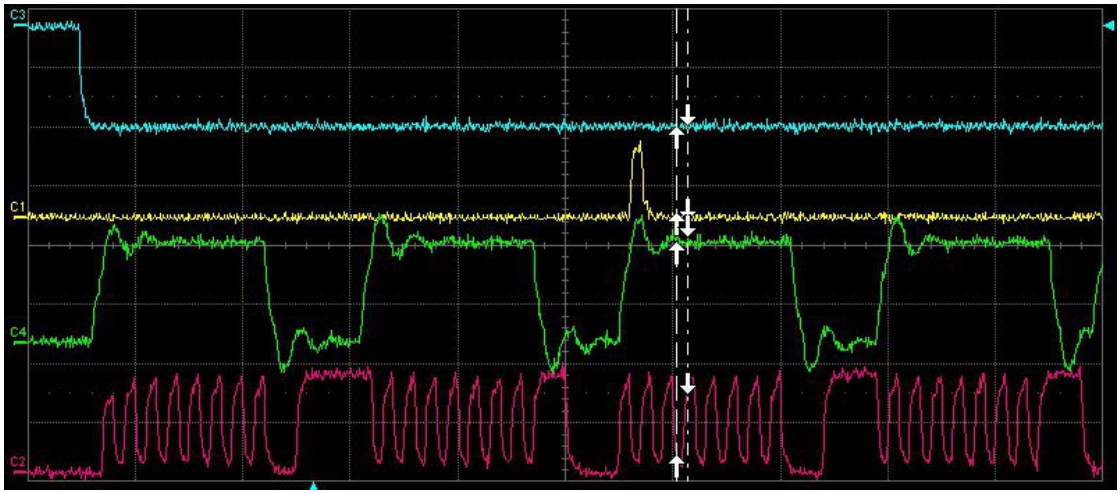


Figure 5.10: SPI clock division property observed in our lab environment.

CHAPTER 6

Algebraic Approach to Runtime Verification

In previous chapter we have seen an efficient method to provide qualitative satisfaction verdicts in real-time, using hardware runtime monitors. The task of evaluating a CPS trace against a specification, frequently requires more than a binary answer. In such context, RV is used to measure the satisfaction robustness of a CPS trace with respect to a specification [AHF+14, DZS+15, ALFS11b]. For this purpose, many measures were developed [FP09, DM10, DFM13, DZS+15, AHF+14, ALFS11b, Don10, RDS+15, DDD+15, BVSF13, AT15, JBGN16, JBG+18, BFMU17], mostly for Signal Temporal Logic. The proliferation of measures reflects the various needs of different applications, and the limitations of STL-based measure definitions. With each new measure being introduced, a new algorithm usually follows.

Algebraic structures, such as *monoids* and *semirings*, guarantee certain axioms to be satisfied over a set and associated generic binary operations: $\oplus$-*addition* and $\otimes$-*multiplication*. For the purpose of performing robustness analysis of a CPS trace, we designate these operations as *choice* and *concatenation*. Measuring a robustness of a trace practically means to evaluate a *distance* of a trace from the specification. Thus, we have observed the structure of a CPS trace which we regard as a sequence of variable valuations, together with *concatenation* operation of a new valuation to the existing trace, to find that this pair satisfies the axioms of a monoid. In addition, we have discovered that the specification, regarded as a set of traces, possesses a semiring structure with respect to *choice* of a trace and *concatenation* of new values to a trace.

These observations opened the door to defining a generic algorithm for measuring robustness of a CPS trace. Since the algorithm uses generic algebraic operations, it allows instantiating arbitrary real-valued functions to choice and concatenation. As a result, we propose Algebraic Runtime Verification (ARV) as a general, semantic approach to measuring a satisfaction robustness of a CPS trace, with respect to a specification automaton.

In this chapter, we start by defining a distance between two points in a metric space in an algebraic fashion, using semirings. Then, we extend the distance definition to valuations and traces and define a trace-specification distance. Finally, we apply the same semiring operations to decompose and calculate valuation-predicate distance, which is a distance of a single valuation from a set of legal valuations specified by a predicate. We leverage these definitions and weighted symbolic automata to define the algorithms behind ARV.

ARV allows any regular specification formalism as an input language. The requirements are translated into a symbolic automaton which is then decorated by the ARV with algebraically-defined weight functions. At the moment, ARV is used to provide quantitative verdicts based only on *space robustness*. A *spatio-temporal* robustness degree based on the Weighted Edit Distance, which can be considered an instantiation and an extension of ARV to *temporal robustness*, is given in Chapter 7.

To illustrate the power of our framework we instantiate our algebraic monitors with different semirings and demonstrate easy switching between qualitative and qualitative monitoring. In the end of this chapter we evaluate ARV on traces generated from benchmark models prominent in the research community. Last but not the least, we compare ARV with widely used research tools. The content of this chapter is based on the work published in [JBGN18].

6.1 Algebraic Approach to Distance

We first introduce the background needed to develop our algebraic runtime verification algorithm. The definition of metric space was given in 4.3.1. In this section we proceed with defining metric spaces and distances in an algebraic fashion. At this point we briefly recall the semiring definition already provided in Section 4.3 and the specific semirings in Table 4.1.

A semiring $\mathcal{S}$ is a tuple $\mathcal{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$, where S is a set equipped with two binary operations *addition* ($\oplus$) and *multiplication* ($\otimes$) and two identity elements $e_\oplus$ and $e_\otimes$ such that:

- $(S, \oplus, e_\oplus)$ is a commutative monoid with an identity element $e_\oplus$;
- $(S, \otimes, e_\otimes)$ is a monoid with an identity element $e_\otimes$;
- $\otimes$ distributes over $\oplus$; and
- $e_\oplus$ is an annihilator element for $\otimes$.

Since we reason about real-valued behaviors and the distances between them, we are interested in semirings defined over (subsets of) reals. Given an element $m \in \mathcal{M}$ from an ordered set $\mathcal{M}$ and a subset $M \subseteq \mathcal{M}$, we can lift the above definition to reason about the distance[1] between m and M to define a Hausdorff-like measure. We use the $\oplus$-addition to combine individual distances between m and the elements in M and fix $e_\otimes$ to 0. We also need a special value when we compare m to an empty set and define $d(m, \emptyset) = e_\oplus$.

[1] Since $d(m, M)$ is comparing an element to a set, strictly speaking it is not a distance.

$$d(m, M) = \begin{cases} e_{\oplus} & \text{if } M \text{ is empty} \\ \oplus_{m' \in M} d(m, m') & \text{otherwise} \end{cases}$$

We define the *robustness degree* $\rho(m, M)$ of m w.r.t. the set M as follows:

$$\rho(m, M) = \begin{cases} d(m, \mathcal{M} \backslash M) & \text{if } m \in M \\ -d(m, M) & \text{otherwise} \end{cases}$$

Let X denote a set of variables defined over a *domain* $\mathbb{D}$. We denote by $v : X \to \mathbb{D}$ the *valuation* function that maps a variable in X to a value in $\mathbb{D}$. We denote by $\tau = v_1, \ldots, v_n$ a *trace* over X and by $\mathcal{T}(X)$ the set of all traces over X. For example, a valuation of variable x is noted as $v(x)$.

A specification φ over a set of variables X, regardless of the formalism used, defines a *language* $L(\varphi) \subseteq \mathcal{T}(X)$ that partitions the set of all traces over X.

Definition 16 (Trace-specification distance)**.** *Let v and v' be two valuations over $\mathbb{D}^{|X|}$, τ and τ' two traces over X of size m and n and φ a specification over X. We then have:*

$$\begin{aligned} d(v, v') &= \otimes_{x \in X} d(v(x), v'(x)) \\ d(\tau, \tau') &= \begin{cases} e_{\oplus} & \textit{if } m \neq n \\ \otimes_{1 \leq i \leq m} d(v_i, v'_i) & \textit{otherwise} \end{cases} \\ d(\tau, \varphi) &= \oplus_{\tau' \models \varphi} d(\tau, \tau'). \end{aligned}$$

In this thesis, we consider specification languages defined over discrete time and real-valued variables that are *regular*. Examples of specification languages that fall into this category are Signal Temporal Logic (STL) and Timed Regular Expressions (TRE), both interpreted over discrete time. We refer the reader to their semantics, which is defined in Chapter 3.

Let $\mathbb{D} = \mathbb{R}$ be the domain of reals and X the set of variables defined over $\mathbb{D}$. We lift the definition of a distance between two valuations to the distance between a valuation and a predicate by $\oplus$-summing the distances between the valuation and the set of valuations defined by a predicate.

Definition 17 (Valuation-predicate distance)**.** *Given a valuation $v \in D^{|X|}$ and a predicate $\psi \in \Psi(X)$, we have that: $d(v, \psi) = \oplus_{v' \models \psi} \otimes_{x \in X} d(v(x), v'(x))$.*

This completes our definitions for computing the distance between an observation and a specification at a single point in time. We now concentrate on the dynamic (temporal) aspect of the specification.

According to definition 8, a WSA is the pair of an automaton $\mathcal{A}$ and a weight function λ. In the following, we assume that the weights are elements of the semiring $(S, \oplus, \otimes, e_{\oplus}, e_{\otimes})$. We use

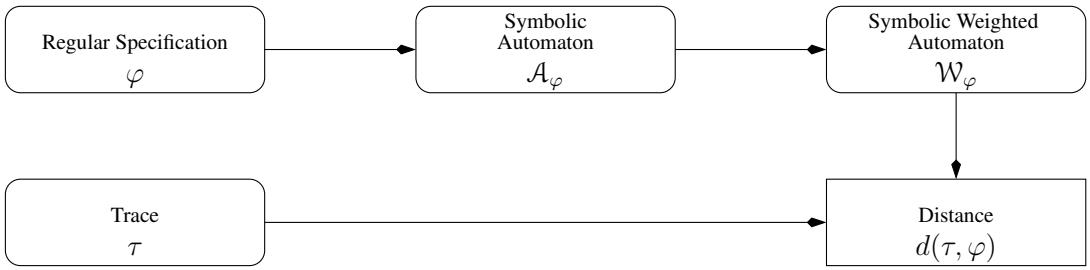


Figure 6.1: Computation of $d(\tau, \varphi)$.

$\otimes$ to compute the weight of a path by $\otimes$-multiplying the weights of the transitions taken along that path. We use $\oplus$ to $\oplus$-sum the path weights induced by an input trace. We now formalize the above notions.

Definition 18 (Path and trace value)**.** *The* path value $\mu(\tau, \pi)$ *of a path* π *induced in* $\mathcal{A}$ *by a trace* τ *is defined as:* $\mu(\tau, \pi) = \otimes_{1 \leq i \leq n} \lambda(v_i, \delta_i)$. *The* trace value $\alpha(\tau, \mathcal{W})$ *of a trace* τ *in* $\mathcal{W}$ *is defined as:* $\alpha(\tau, \mathcal{W}) = \oplus_{\pi \in \Pi(\tau)} \mu(\tau, \pi)$.

6.2 Algebraic Monitors for Correctness and Robustness

In this section, we develop ARV, a novel procedure for abstract computation of the *robustness degree* of a discrete signal with respect to a specification φ. The proposed methods consists of several steps, illustrated in Figure 7.5. We first translate the specification φ into a symbolic automaton $\mathcal{A}_\varphi$ that accepts the same language as the specification. The automaton $\mathcal{A}_\varphi$ treats timing constraints from the formula in an enumerative fashion, but keeps symbolic guards on data variables. We then decorate $\mathcal{A}_\varphi$ with weights on transitions, thus obtaining the *weighted symbolic automaton* $\mathcal{W}_\varphi$. We propose an abstract algorithm for computing the distance between a trace τ and a specification φ by reducing it to the problem of finding the shortest path in $\mathcal{W}_\varphi$ induced by τ. Computing the robustness degree between the trace τ and the specification φ follows from combining the computed distance from the specification φ and its negation $\neg\varphi$.

6.2.1 From Specifications to Weighted Symbolic Automata

Before proceeding with explaining the algorithm details, we remind the reader about the syntax of our two specification languages of choice – Signal Temporal Logic and Timed Regular Expressions. The syntax of a STL formula φ over $P \cup X$ is defined by the following grammar:

$$\varphi \quad := \quad p \mid x \sim u \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \, \mathcal{U}_I \varphi_2 \mid \varphi_1 \, \mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a, b]$ or $[a, \infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the form $[a, a]$, we will use the notation $\{a\}$ instead.

A timed regular expression ψ is defined according to the following syntax [FMNU15]:

$$\psi := \epsilon \mid q \mid rise(p) \mid fall(p) \mid \psi_1 \cdot \psi_2 \mid \psi_1 \cup \psi_2 \mid \psi_1 \cap \psi_2 \mid \psi^* \mid \langle \psi \rangle_I$$

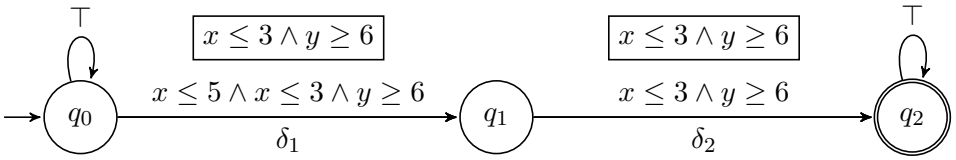


Figure 6.2: Weighted symbolic automaton $\mathcal{W}$ that accepts the language of φ_1 and φ_2.

where q is of the form p, $\neg p$, $x \sim c$ or $\neg(x \sim c)$; I is a time interval $[a, b] \subseteq \mathbb{N}$.

We assume an effective translation T of a regular specification language φ to symbolic automata $\mathcal{A}_\varphi$ that accepts the language of φ. For STL and TRE defined over discrete time, such translation is a moderate adaptation of standard methods, including on-the-fly tableau construction [GPVW96] and temporal testers [PZ08]. During the translation, we decorate the transitions in the symbolic automaton with weight functions that measure the distance between observed valuations and the predicate ψ associated to the transition, i.e. we set instantiate the weight function λ to $\lambda(\delta, v) = d(v, \psi)$, for all $\delta = (q, \psi, q') \in \Delta$.

The Example 9, given in Figure 6.2, shows an automata which is an input for robustness degree algorithm defined by the ARV. The same automata is obtained for two formalizations, one in STL, another in TRE. We have shown that ARV allows for quantitative monitoring of specifications in a formalism of choice, provided there is an efficient translation to automata. However, in case of TRE we would need to transform the resulting timed automata, by enumerating the time and instantiating an appropriate number of states.

Example 9. *We illustrate this step on specifications φ_1 and φ_2 from Example 2:*

$$\varphi_1 \equiv \Diamond(x \leq 5 \wedge \Box_{[0,1]}(x \leq 3 \wedge y > 6))$$
$$\varphi_2 \equiv \top \cdot ((x \leq 5 \cdot \top) \cap \langle x \leq 3 \wedge y \geq 6 \rangle_{[0,1]}) \cdot \top$$

In Figure 6.2 we depict the WSA that accepts the language of φ_1 and φ_2. The $\wedge$-minimal predicates are shown in the boxes above the transitions.

6.2.2 Valuation-Predicate Distance Computation

We propose an effective procedure for computing the distance between a valuation v and a predicate ψ. The procedure is shown in Algorithm 1 and works as follows. The input to the procedure is a valuation v and a predicate ψ in DNF. The computation of the distance between v and ψ is done inductively in the structure of ψ. In the base case when ψ is an atomic predicate, the distance between v and ψ is $e_\otimes$ if v satisfies ψ, and it is equal to $d(v(x), k)$ otherwise. We compute logic $\vee$ and $\wedge$ by interpreting them as $\oplus$-addition and $\otimes$-multiplication.

Algorithm 1 $\mathsf{vpd}(v, \psi)$

Require: Predicate $\psi \equiv \bigvee_i \bigwedge_j p_{ij}$
1: **if** (ψ is UNSAT) **then return** $e_\oplus$
2: **end if**
3: **if** ($\psi = x \preceq k$) or ($\psi = \neg(x \preceq k)$) **then**
4: **if** ($v(x) \models \psi$) **then return** $e_\otimes$
5: **else return** $d(v(x), k)$
6: **end if**
7: **else if** ($\psi = \psi_1 \vee \psi_2$) **then**
8: **return** $\mathsf{vpd}(v, \psi_1) \oplus \mathsf{vpd}(v, \psi_2)$
9: **else if** ($\psi = \psi_1 \wedge \psi_2$) **then**
10: **return** $\mathsf{vpd}(v, \psi_1) \otimes \mathsf{vpd}(v, \psi_2)$
11: **end if**

The procedure presented in Algorithm 1 indeed computes the distance from Definition 17 for (1) idempotent semirings and (2) additively idempotent semirings with the predicate given in $\wedge$-minimal DNF. The transformation of arbitrary predicates to DNF is standard, while we present the $\wedge$-minimization of a predicate in DNF in Algorithm 2. In the following example, we give intuition why predicates must be in $\wedge$-minimal DNF if the semiring is only $\oplus$-idempotent.

Algorithm 2 $\wedge$-$\mathsf{min}(v, \psi)$

Require: Predicate $\psi \equiv \bigvee_i \bigwedge_j p_{ij}$
1: Let $P_i = \{p_{i1}, \ldots, p_{ij}\}$
2: Let $\mathcal{P} = \{P_1, \ldots, P_i\}$
3: **for all** $P \in \mathcal{P}$ **do**
4: **for all** $p_1, p_2 \in P$ s.t. $p_1 \neq p_2$ **do**
5: **if** $p_1 \to p_2$ **then**
6: $P \leftarrow P \backslash \{p_2\}$
7: **end if**
8: **end for**
9: **end for**
10: **return** $\bigvee_{P \in \mathcal{P}} \bigwedge_{p \in P} p$

The procedure presented in Algorithm 1 indeed computes the distance from Definition 17 for (1) idempotent semirings and (2) additively idempotent semirings with the predicate given in $\wedge$-minimal DNF. The transformation of arbitrary predicates to DNF is standard, while we present the $\wedge$-minimization of a predicate in DNF in Algorithm 2. In the following example, we give intuition why predicates must be in $\wedge$-minimal DNF if the semiring is only $\oplus$-idempotent.

Example 10. *Consider the term $x \leq 3 \wedge x \leq 5$ from the transition δ_1 in Figure 6.2 that defines the set of valuations $\{v(x) \mid v(x) \leq 3\}$, its semantically equivalent $\wedge$-minimal representation $x \leq 3$ and the valuation $v(x) = 6$. It is clear that $d(v, x \leq 3 \wedge x \leq 5) = d(v, x \leq 3) = 3$. Let us consider the tropical semiring and the computation of vpd. We illustrate the need for the $\wedge$-minimal predicate in Figure 6.3. We have that $\mathsf{vpd}(v, x \leq 3) = 3$, but $\mathsf{vpd}(v, x \leq 3 \wedge x \leq$*

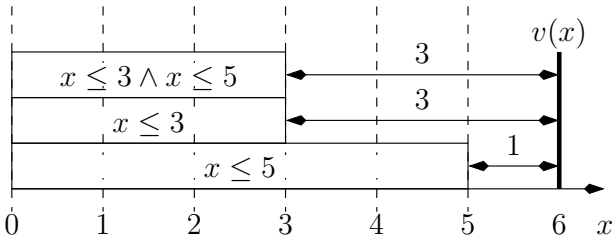


Figure 6.3: Example of a distance between v and ψ.

$5) = 1 \otimes 3 = 1 + 3 = 4$. *Due to the non-minimality of the term and the additive nature of* $+$*, we incorrectly accumulate the distance from* $v(x)$ *to the atomic predicate* $x \leq 5$.

The following Theorem 1 states that the procedure $\mathsf{vpd}(v, \psi)$ described in Algorithm 1 correctly computes the the distance $d(v, \psi)$ between an observation v and a predicate ψ for idempotent (Boolean, MinMax) semirings. In the case of semirings that are only additively idempotent, such as the tropical semiring, the predicate must be in addition translated with the application of Algorithm 2 to the $\wedge$-minimal form. We use this theorem to show that we properly generate weights on the transitions of the symbolic automaton.

Theorem 1. *Given a predicate* ψ *in DNF, a valuation* v *and the distance* $d(v, \psi)$ *defined over a bounded semiring* S*, we have that* $\mathsf{vpd}(v, \psi) = d(v, \psi)$ *if:*

1. S *is idempotent; or*
2. ψ *is in* $\wedge$*-minimal DNF.*

We prove Theorem 1 in Appendix A.1.

6.2.3 Trace Value Computation

In this section, we present a dynamic programming procedure (see Algorithm 3) for computing the value of a trace $\tau = v_1, \ldots, v_n$ in a symbolic weighted automaton $\mathcal{W}$. We assume that the weights are defined over a semiring S. The algorithm first assigns to every state q the initial cost, depending whether q is initial or not. At every step $i \in [1, n]$ and for every state q of $\mathcal{W}$, the procedure computes the cost of reaching q with the prefix of τ which is i symbols long. The procedure uses the $\otimes$-multiplication to aggregate the valuation-predicate distances collected along every path π induced by τ and thus compute the path weight and the $\oplus$-addition to combine the weights of all the accepting paths induced by τ.

Algorithm 3 $\mathsf{val}(\tau, \mathcal{W})$

1: **for all** $q \in Q$ **do**
2: $\quad v(q,0) \leftarrow (q \in I)\ ?\ e_{\otimes}\ :\ e_{\oplus}$
3: **end for**
4: **for** $i = 1\ to\ n$ **do**
5: $\quad v(q,i) \leftarrow \oplus_{(s,\psi,q)\in\Delta}(v(s,i-1) \otimes \mathsf{vpd}(v_i,\psi))$
6: **end for**
7: **return** $\oplus_{q\in F}\ v(q,n)$

The following theorem states the main result of this chapter - Algorithm 4 correctly computes the distance between a specification φ and a trace τ. We note that the theorem works for any $\mathcal{W}$ that has the same language as φ. In particular, the theorem implies that val is invariant to different syntactic, semantically equivalent, representations of the specification and of the associated automaton.

Theorem 2. *Given a specification φ, its associated WSA $\mathcal{W}$ defined over a semiring S such that $L(\varphi) = L(\mathcal{W})$ and a trace τ, we have that $\mathsf{val}(\tau, \mathcal{W}) = \alpha(\tau, \mathcal{W}) = d(\tau, \varphi)$.*

The proofs of Theorem 2 and Theorem 3 are also provided in Appendix A.1.

Algorithm 4 $\mathsf{rob}(\tau, \varphi)$

1: $\mathcal{W}_{\varphi} \leftarrow \mathsf{T}(\varphi)$
2: $\mathcal{W}_{\neg\varphi} \leftarrow \mathsf{T}(\neg\varphi)$
3: $v_1 \leftarrow \mathsf{val}(\tau, \mathcal{W}_{\varphi})$
4: $v_2 \leftarrow \mathsf{val}(\tau, \mathcal{W}_{\neg\varphi})$
5: **if** $v_1 = e_{\otimes}$ **then return** v_2
6: **else return** $-v_1$
7: **end if**

We finally build the monitor that measures the robustness degree between the trace τ and a specification φ by computing the value of τ in $\mathcal{W}_{\varphi}$ and $\mathcal{W}_{\neg\varphi}$. This procedure is summarized in Algorithm 4 that trivially implements the robustness measure ρ. We show that our abstract computation of ρ is sound and complete. In the following theorem we use a distance value which belongs to a domain which, generally speaking, may not have a total order. Thus, we use the partial order symbol $\sqsubset$ for comparing two distances.

Theorem 3 (Soundness and completeness)**.** *Given traces τ and τ', a specification φ and distances $d(\tau, \tau')$, $d(\tau, \neg\varphi)$ defined over a bounded semiring S,*

$$\begin{aligned} \rho(\tau,\varphi) > 0 \quad &\rightarrow \quad \tau \models \varphi \\ \rho(\tau,\varphi) < 0 \quad &\rightarrow \quad \tau \not\models \varphi \\ \tau \models \varphi \text{ and } d(\tau,\tau') \sqsubset d(\tau,\neg\varphi) \quad &\rightarrow \quad \tau' \models \varphi. \end{aligned}$$

We first observe that if $\rho(\tau, \varphi) = 0$, then we do not know (only from that number) whether τ satisfies φ. We illustrate this observation with the formula $x > 0$ and the trace $\tau = 0$ of size one. It is clear that $\tau \not\models \varphi$ but the actual distance between the element in $\{v \mid v > 0\}$ that is closest to 0 and 0 is infinitesimally close to 0. In order to have both directions of the implications in the soundness proof and to guarantee that the robustness degree is never equal to 0, we would need to introduce non-standard reals. Note that even with the current setting, we can easily say whether $\tau \models \varphi$, even when $\rho(\tau, \varphi) = 0$. Second, we do not need to explicitly compute the complement automaton $\mathcal{W}_{\neg\varphi}$ if $\mathcal{W}_\varphi$ is deterministic. In that case, it is sufficient to apply a slight modification of Algorithm 3 on $\mathcal{W}_\varphi$ only. The modification consists in reporting either the minimum value of an accepting or the minimum value of a non-accepting location, depending on whether the trace satisfies φ.

6.2.4 Translation Size and Algorithm Complexity

We observe that the size of the symbolic automaton $\mathcal{W}_\varphi$ is exponential in the size of the STL specification and the largest constant appearing in temporal operators [AH93]. This makes our monitors sensitive both to the size of the formula, but also to the real-time constraints used in the specification. In particular, monitoring a specification with a punctual bound, such as $\Diamond_{[a,a]}p$, results in a combinatorial explosion as it requires at any time step i 2^a locations to predict all possible values of p within the interval $[i, i + a]$. On the other hand, formulas that have a lower bound equal to 0, such as $\Diamond_{[0,a]}p$ behave better, as they need only a locations to keep track of the necessary predictions.

The size of the predicates decorating the transitions in $\mathcal{W}_\varphi$ is also exponential in the number of propositions appearing in the formula, due to the translation of the predicate to DNF form. The algorithm $\mathsf{val}(\tau, \mathcal{W})$ applied to a trace τ of size l, and $\mathcal{W}$ with n locations, m transitions, and maximum size p of the predicates, takes in the order of $l \cdot p \cdot (\max(m, n))$ iterations.

6.2.5 Instantiating Monitors

In Sections 6.2.2 and 6.2.3, we presented an abstract monitoring procedure that measures a robustness degree of a trace τ with respect to a specification φ. We give concrete semantics to these monitors by instantiating the semiring and the distance function. Here we consider three instantiations of the procedure:

1. Boolean semiring with $d(a, b) = 1$ if $a \neq b$ and 0 otherwise
2. Minmax semiring with $d(a, b) = |a - b|$
3. Tropical semiring with $d(a, b) = |a - b|$

The instantiation 1 gives the monitors classical qualitative semantics, where the distance of τ from φ is 0 if τ is in the language of φ and 1 otherwise. The instantiation 2 gives the computation of the robustness degree based on the infinite norm, as defined in [FP09]. The instantiation 3 gives the computation of the robustness degree based on the Hamming distance lifted to the sets.

semiring	state	init	$v_0 = (4, 2)$	$v_1 = (5, 3)$	$v_2 = (2, 5)$	$v_3 = (3, 5)$
Boolean	q_0	0	0	0	0	0
	q_1	1	1	1	1	1
	$\boldsymbol{q_2}$	1	1	1	1	**1**
MinMax	q_0	0	0	0	0	0
	q_1	∞	4	3	1	1
	$\boldsymbol{q_2}$	∞	∞	4	3	**1**
Tropical	q_0	0	0	0	0	0
	q_1	∞	5	5	2	1
	$\boldsymbol{q_2}$	∞	∞	10	7	**3**

Table 6.1: $\mathsf{val}(\tau, \mathcal{W})$ computed on WSA from Figure 6.2 with different semirings.

Example 11. *We illustrate our monitoring procedure instantiated to different semantics in Table 6.1. We choose the trace* $\tau = (4, 2) \cdot (5, 3) \cdot (2, 5) \cdot (3, 5)$ *that violates the specifications* φ_1 *and* φ_2 *from Example 2. We instantiate Algorithm 3 to the specific semirings and apply each instantiation to the* τ *and* $\mathcal{W}$ *from Figure 6.2. We mark in bold the accepting state and the trace value.*

We note that by Theorem 2, our computation of robustness is *precise* with respect to the semantics of the specification, regardless of the instantiated semiring. This is in contrast to the syntactic approaches to robustness [FP09, DM10] that under-approximate the robustness value. In case we instantiate MinMax semiring in ARV, we do not demonstrate imprecision shown in Examples 17,18 from [FP09]. The comparison results from section 6.3.1 confirm this observation.

6.2.6 Implementing Monitors

The ARV framework presented in this section admits an effective and straightforward implementation of both *offline* and *real-time* monitors. Algorithms 3 and 4 describe the offline monitoring procedure in which we assume that the entire input trace τ is available. We use this assumption solely to simplify the presentation. This monitoring procedure is directly extended to its online variant by processing new inputs as they arrive and updating the costs of the automaton locations in an on-the-fly fashion.

The size of the automaton $\mathcal{W}$ obtained from a specification φ is in $\mathcal{O}(2^{b|\varphi|})$, where b is the largest constant appearing in timed modalities $[a, b]$ of φ [AH93]. We note that this is the worst case upper bound. We conjecture that the discrete time adaptation of more recent translations from real-time temporal logics to timed automata [Nič08, DM12] yields automata with size that is exponential in the ratio between the lower bound a and the upper bound b. This significantly reduced the size of automata for many classes of specifications. For example, the automaton

associated to $\Diamond_{[0,b]}p$ has the size in $\mathcal{O}(b)$. On the other hand, translating a punctual formula such as $\Diamond_{[b,b]}p$ without any assumption on the variability of p results indeed into an automaton that reaches the worst case size in $\mathcal{O}(2^b)$.

By definition of Algorithm 3, it is clear that the number of operations required to compute the trace value at every time step is linear in (1) the size of the automaton, (2) the maximum number of incoming transitions that any location in the automaton can have, and (3) the maximum size of the predicates appearing in the transitions of the automaton. More specifically, the number of operations at every time step remains constant with the length of the observed prefix. This results in a predictable monitoring procedure with real-time guarantees that is well-suited for an online implementation.

Algorithm 3 needs to store at every point in time $2n$ real values, where n denotes the number of locations in the automaton - the current and the previous cost for each location. In our theoretical setting, both inputs and costs range over reals. In any practical implementation, the observation of the monitored CPS during its operation is limited by the resolution of the measurement device, which typically quantizes the observed values into a finite set of integer values $[0, k]$. We assume that we use the binary representation of integers. It follows that every value stored in a monitor instantiated with the Boolean or the MinMax semiring is bounded by k and can be represented by $\log k$ bits. As a consequence, Algorithm 3 instantiated with qualitative or infinity-based norm quantitative semantics has memory requirements that are constant in the size of the input trace, another important property for the implementation of real-time monitors.

In the case of monitors instantiated with the tropical semiring, the situation is slightly more involved, since costs are accumulated over time. It follows that for such monitors, the memory requirements grows logarithmically with the length of the input trace. A real-time implementation of such monitors can address this logarithmic growth by using a large enough number of bits for integers that will guarantee correctness for sufficiently long input traces.

6.3 Evaluation

We implemented our approach (ARV) in a prototype tool in Java. In order to determine satisfiability of WSA transition constraints, we used the Z3 solver [DMB08]. We evaluate our framework on two case studies from the automotive domain: The Autonomous Vehicle Control Stack model [ROA+17] and the Automatic Transmission System model [HAF14]. In the first case study we also compare the precision of ARV with relevant tools developed by the RV community.

6.3.1 Autonomous Vehicle Control Stack

The first benchmark is a model of an autonomous vehicle control stack, which is used to solve the trajectory planning problem. The stack consists of three layers, starting from the top: Behavioral Planner (BP), Trajectory Planner (TP) and Trajectory Tracker (TT). The BP provides the coarse-grain trajectory way-points for the Autonomous Vehicle (AV), and supplies them to the underlying TP which calculates fine grain trajectory points, using cubic spline trajectory generation. The

lowest layer is the TT which actuates the AV based on the trajectory points in order to steer it towards the requested path.

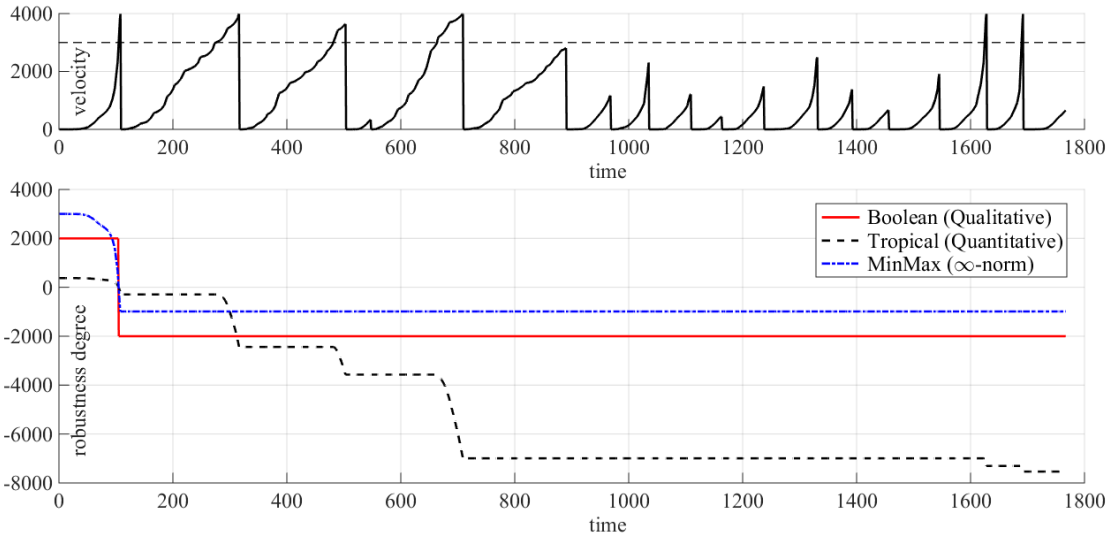


Figure 6.4: Robustness degree $\rho(\tau_{[0,t]}, \varphi_1)$, where $\varphi_1 = \Box(v_{ego} \leq v_{limit})$, based on three different semiring instantiations.

The benchmark supports three distinct layouts: a room with obstacles, a curved road and a roundabout. The obstacles and undesired areas are determined by assigning the specific cost. The autonomous vehicles can simulate 4 scenarios: parallel driving, paths crossing without collision and collision avoidance by either the first or the second vehicle. We ran the scenario of AVs performing parallel driving on roundabout layout and we obtained the traces for the speed v_x and acceleration a_x. We specified two requirements, which model normal operation (w.r.t. acceleration) and traffic rules (speed limit).

In Fig 6.4 we demonstrate various robustness degrees for the following requirement: "The ego vehicle travels at a velocity less than or equal to the speed limit". We formalize this requirement in STL: $\varphi_1 = \Box(v_{ego} \leq v_{limit})$. In Fig 6.5 we monitor the following requirement "If the autonomous vehicle started to accelerate, then it will not start decelerating in the very near future". We formalize this requirement in STL: $\varphi_2 = \Box((a_x \geq \theta) \rightarrow \Box_{(0,\epsilon]} \neg(a_x \leq 0))$. We select $\theta = 5$ and $\epsilon = 3$. For the clarity of the Figure 6.4, some values are (down)scaled.

6.3.2 Comparison with S-TaLiRo and Breach

In this section, we compare our approach to the widely used S-TaLiRo and Breach tools. Both S-TaLiRo and Breach implement robustness monitoring algorithms, measured with infinite norm. The algorithms implemented in these tools are syntactic and work inductively on the structure of the formula without passing via automata. We note that in contrast to our setting, these two tools work with continuous time. In order to enable the comparison, we instantiate our monitors to

the MinMax semiring and we simulate the Autonomous Vehicle Control Stack model with fixed sampling, thus ensuring that the discrete vs. continuous discrepancy does not affect the results.

We first observe that S-TaLiRo and Breach use a different definition of robustness from ARV. Definition of robustness in S-TaLiRo and Breach is inductive in both time and the structure of the specification, and assigns a robustness degree at every point in time to each sub-formula. In contrast, we adopt a more global and semantic definition in which robustness is the distance between the observed trace and the boundary of the set of traces that satisfy/violate the specification.

In contrast to ARV, S-TaLiRo and Breach support a fixed specification language (STL) with specific quantitative semantics. For instance, adding support for regular expressions is not a trivial task. It requires developing new inductive robustness definition for regular expression operators and devising completely new algorithms for computing the robustness degree [BFMU17]. On the other hand, adding regular expressions to ARV only requires an effective procedure that translates the expressions to symbolic automata. In addition, ARV allows plugging-in accumulative semantics without changing the underlying algorithms. We do not see an easy way to adapt the existing algorithms in S-TaLiRo and Breach to accommodate for accumulative quantitative semantics.

We now compare the robustness values obtained by S-TaLiRo, Breach and ARV for semantically equivalent specifications as well as for specifications that are contradictions. In ψ_1 and ψ_2 from Table 6.2 we test the sensitivity of the robustness degree algorithm w.r.t. to the minimal set

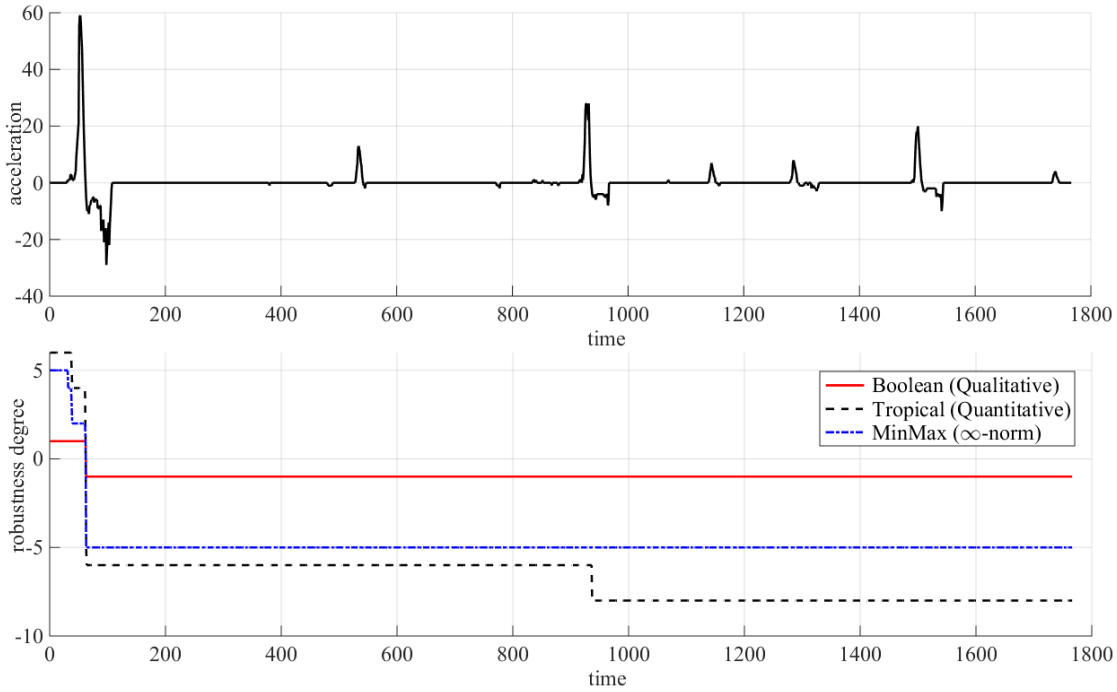


Figure 6.5: Robustness degree $\rho(\tau_{[0,t]}, \varphi_2)$, where $\varphi_2 = \Box((a_x \geq \theta) \rightarrow \Box_{(0,\epsilon]} \neg(a_x \leq 0))$, based on different semiring instantiations.

benchmark formula	S-TaLiRo	Breach	ARV
$\psi_1 = a \geq -30 \wedge a \leq 30$	30	30	30
$\psi_2 = (a \geq -30 \wedge a < 0) \vee (a \geq 0 \wedge a \leq 30)$	0	10^{-13}	30
$\psi_3 = \Diamond(a \geq -10)$	69	69	69
$\psi_4 = \Diamond((a \geq -10 \wedge a \leq 60) \vee (a \geq 55))$	35	35	69
$\psi_5 = \Box(a \geq 5 \wedge a < 5)$	-64	-59	$-\infty$
$\psi_6 = \neg(\Diamond\psi_1 \vee \Diamond(a < -30 \vee a > 30))$	-30	-30	$-\infty$

Table 6.2: Precision comparison between the syntactic-based tools and the semantic approach. We compare the robustness degree results obtained from S-TaLiRo, Breach and ARV.

representation. The formula ψ_1 defines the acceptable range $[-30, 30]$ of values for a, while ψ_2 is a semantically equivalent formula that represents the same set as a disjoint union $[-30, 0) \cup [0, 30]$. Similarly, ψ_3 and ψ_4 represent two semantically equivalent temporal formulas. Finally, both ψ_5 and ψ_6 represent formulas that are unsatisfiable.

We can observe that our approach produces results that are invariant to the syntactic representation of the formula. In particular, our monitoring algorithm is able to detect unsatisfiable formulas. In contrast, neither S-TaLiRo nor Breach can detect unsatisfiable specifications. We can also observe that in some cases, these two tools are also sensitive to the syntactic representation of the specification. This is visible in the computation of the robustness for ψ_1 and ψ_2, where S-TaLiRo computes the inconclusive result 0 for a specification that is satisfied by the trace, while Breach correctly finds that the formula is satisfied, but assigns it a very low robustness degree. These results directly follow from the different ways to define robust semantics in the two approaches.

The flexibility of our approach comes at a price. In contrast to S-TaLiRo and Breach that admit monitoring procedures that are linear in length of the input trace and exponential in the size of the formula [DFM13], ARV requires translating specifications to symbolic automata, and in case of STL it results in a monitoring algorithm that is linear in the length of the trace and exponential in the size of the specification but also in the largest constant appearing in the formula.

6.3.3 Automatic Transmission System

We first consider the Automatic Transmission deterministic model [HAF14] as our system-under-test (SUT). It is a model of a transmission controller that exhibits both continuous and discrete behavior. The system has two inputs – the throttle u_t and the break u_b. The break allows the user to model variable load on the engine. The system has two continuous-time state variables – the speed of the engine ω (RPM), the speed of the vehicle v (mph) and the active gear g_i.

The system is initialized with zero vehicle and engine speed. It follows that the output trajectories depend only on the input signals u_t and u_b, which can take any value between 0 and 100 at any point in time. The Simulink model contains 69 blocks including 2 integrators, 3 look-up tables, 2

two-dimensional look-up tables and a Stateflow chart with 2 concurrently executing finite state machines with 4 and 3 states, respectively. The benchmark [HAF14] defines 8 STL formalized requirements that the system must satisfy.

We select the following requirement for our case study: "The engine and the vehicle speed never reach ω_{max} and v_{max}". We use STL to formalize this requirement as follows: $\varphi_3 = \square((\omega \leq \omega_{max}) \wedge (v \leq v_{max}))$.

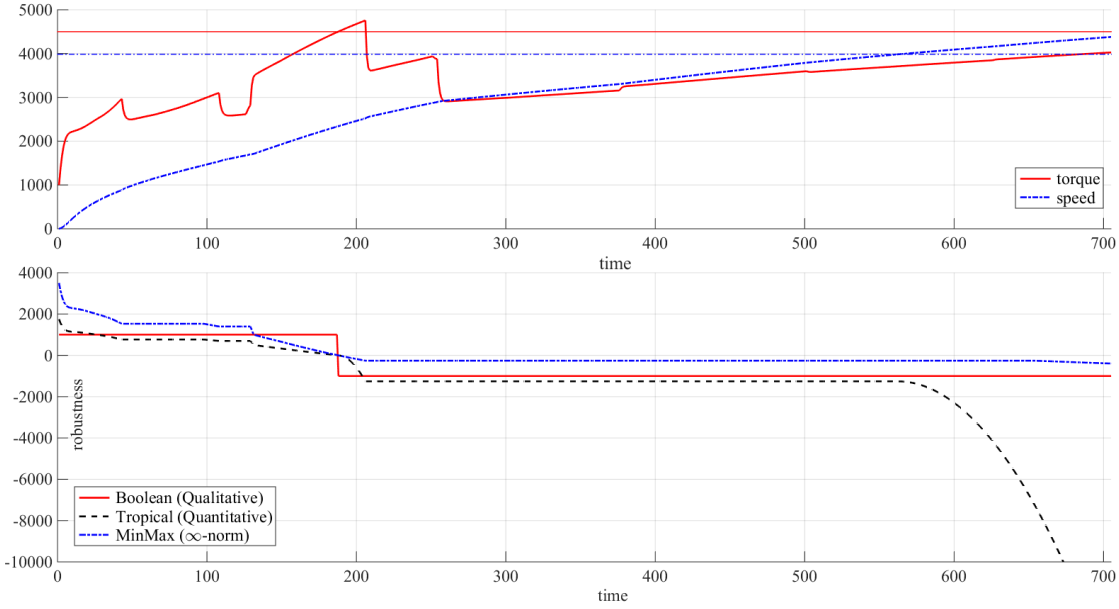


Figure 6.6: Robustness degree $\rho(\tau_{[0,t]}, \varphi_3)$, where $\varphi_3 = \square((\omega \leq \omega_{max}) \wedge (v \leq v_{max}))$, based on different semiring instantiations.

We translate φ_3 into a WSA and instantiate it with three semirings – Boolean, MinMax and Tropical, thus obtaining monitors for qualitative, ∞-norm and Hamming distance based quantitative semantics. We note that the qualitative semantics yields the binary verdicts. In contrast, the quantitative semantics based on the ∞-norm is obtained by instantiating MinMax idempotent semiring. Therefore the robustness degree based of this semantics relies on maximum pointwise distance, without accumulating it over time. For this reason we find it useful for performing worst-case analysis. Finally, the quantitative semantics based on Tropical semiring accumulates the pointwise distances over the entire trace due to non-idempotent $\otimes$ (addition). Thus, the robustness degree in each time instance depends on the robustness of the entire trace prefix, ensuring that no information on robustness is lost over time. We can finally observe that all the quantitative semantics are consistent with qualitative semantics, as stated in Theorem 3. This is observable in Fig 6.6, before time reaches 200.

CHAPTER 7

Quantitative Monitoring with Weighted Edit Distance

In the recent past, property-based runtime monitoring of CPS centered around Signal Temporal Logic (STL) [MN13] and its variants have received considerable attention [FP09, DM10, DFM13, AT15, NN14a, BDSV14, BBS14a]. In its original form, STL allows to distinguish correct from incorrect behaviors, which may not be sufficient for real-valued behaviors. The classical satisfaction relation can be replaced by a quantitative *robustness degree* [FP09, DM10, DFM13] of a behavior with respect to a temporal specification. The robustness degree provides a finer measure of how far is the behavior from satisfying or violating of the specification.

In this chapter we propose a novel quantitative semantics for STL and leverage it to calculate a *robustness degree* sensitive to behavior mismatches in both *space* and *time*. We consider applications in which continuous CPS behaviors are observed by a digital device. In this scenario, continuous behaviors are typically discretized, both in time and space, by an ADC. As a consequence, we interpret STL over discrete-time digitized behaviors.

We first propose the *weighted* edit distance as an appropriate metric for measuring similarity between CPS behaviors. The weighted edit distance has the following desirable characteristics:

1. It is cumulative, hence it can differentiate between a single and multiple deviations from a reference behavior;
2. It combines spatial and temporal aspects, which are both important when reasoning about CPS behaviors; and
3. It is defined in discrete time, which is an important aspect for the applications that we consider.

We then provide the quantitative semantics for STL based on this distance and discuss the effects of sampling and quantization on the distance value. We develop an efficient online algorithm for computing the robustness degree between a behavior and an STL formula. The implemented

procedure can connect to a CPS design simulation or to a real device to monitor its correctness and quality. The algorithm is implemented in Verilog and evaluated on an automotive benchmark.

In Section 7.1 we introduce the notion of Weighted Edit Distance. In Section 7.2 we propose a novel approach for computing WED-based robustness degree of a discrete signal with respect to an STL property. We conclude by demonstrating the approach on two case studies. The content of this chapter is based on work published in [JBGN16, JBG+18].

7.1 Weighted Edit Distance

Measuring the similarity of sequences is important in many application areas, such as information theory, spell checking and bioinformatics. The *Hamming distance* d_H is the most basic and common string measure arising from the information theory. It measures the minimum number of *substitution* operations needed to match equal length sequences. The *edit distance* d_E extends the Hamming distance with two additional operations, *insertion* and *deletion* and is defined as the minimum accumulation of edit operation costs used to transform one sequence into the other.

Neither of these metrics provide satisfactory solution for comparing digitized signals. They are defined over unordered alphabets and associate fixed costs to different kinds of operations. In contrast, the value domain of digital signals admits a natural notion of a distance representing the difference between two signal valuations. In addition, the Hamming distance provides only point-wise comparisons between sequences and consequently does not account for potential timing discrepancies in the sampled signals. Two discrete signals that differ only in a constant time delay will typically have a large Hamming distance. The edit distance addresses this problem by allowing us to bridge the time shifts using insertion and deletion operations.

Inspired by [Moh03] and [SDC13], we propose the *weighted edit distance* as the measure for comparing the similarity of two discrete signals. It adopts the insertion and deletion operations from the edit distance and adapts the substitution operation to the ordered alphabets. Since we consider multi-dimensional signals, we extend the cost of the substitution operation to take into account different variable valuations.

Let X be a finite set of variables defined over some interval domain $\mathbb{D} = [v_{min}, v_{max}]$. Given two valuation vectors $a, b \in \mathbb{D}^{|X|}$ of X, we denote by $d_M(a, b)$ the *Manhattan distance* [Kra12] between a and b, where $d_M(a, b) = \Sigma_{i=0}^{|X|-1}|a_i - b_i|$. Let $w_i, w_d \in \mathbb{Q}$ be weight constants for the insertion and deletion operations. We then define the *costs* of the substitution c_s, insertion c_i and deletion c_d operations as follows: $(1)c_s(a, b) = d_M(a, b); (2)c_i = w_i; (3)c_d = w_d$. The definition of the WED adapts the classical edit distance recursive definition with the new costs.

Definition 19 (Weighted edit distance)**.** *Let* $s_1 : [0, l) \times X \to \mathbb{D}$ *and* $s_2 : [0, l) \times X \to \mathbb{D}$ *be*

discrete-time signals. The weighted edit distance $d_W(s_1, s_2)$ *equals to* $d_{l,l}(s_1, s_2)$:

$$
\begin{aligned}
d_{-1,-1}(s_1, s_2) &= 0 \\
d_{i,-1}(s_1, s_2) &= d_{i-1,-1}(s_1, s_2) + c_i \\
d_{-1,j}(s_1, s_2) &= d_{-1,j-1}(s_1, s_2) + c_d \\
d_{i,j}(s_1, s_2) &= min \begin{cases} d_{i-1,j-1}(s_1, s_2) + c_s(s_1(i, X), s_2(j, X)) \\ d_{i-1,j}(s_1, s_2) + c_i \\ d_{i,j-1}(s_1, s_2) + c_d \end{cases}
\end{aligned}
$$

Proposition 1. *The weighted edit distance is a distance.*

We prove this claim in Appendix A.2.

Remark We chose the Manhattan distance for the substitution cost because it combines the absolute difference of several signal components.

We now further motivate the use of the weighted edit distance and discuss in more depth its characteristics. We do this by comparing the weighted edit distance (d_W) to the Hamming distance (d_H) and to the distance based on the infinity norm (d_{max}). In order to compare these three distances, we record the data from a device implementing an automotive communication protocol. We manually manipulate the data to illustrate specific distance properties. We note that we normalize the two cumulative distances with the total number of data samples, in order to have comparable results.

We first study the cumulative property of WED. Figures 7.1 (a) and (b) depict two scenarios, both consisting of a reference (s_r) and a measured (s_r) behavior. In the first scenario, the two behaviors are equivalent, except for a short spike that happens during each pulse. In the second scenario, the spikes are continuously repeated. Figures 7.1 (c) and (d) show the evolution of the three distances over time, where the distance value at time t corresponds to the distance between the reference and measured behavior prefixes of size t. We can observe that d_{max} measures the maximum deviation between s_r and s_m and hence does not distinguish between a single and multiple deviations. On the other hand, both d_H and d_W are cumulative, and the distance between the reference and the measured behavior increases with the number of deviations.

Figures 7.2 (a), (b) and (c) show three scenarios with the measured signal being equivalent to the reference signal shifted by an increasing amount, respectively. Figures 7.2 (d), (e) and (f) show the evolution of d_W, d_H and d_{max} over time. We first note that d_{max} does not make the distinction between the three scenarios. Second, we can observe that d_W grows slower than d_H over time. This happens because d_W counts a small number of insertion and deletion operations, while d_H accumulates all the pointwise differences between s_r and s_m over time. Finally, we notice that d_H does not make a distinction between the second and the third scenario, despite the different time shifts. This happens because in both scenarios the pulses from the measured signal

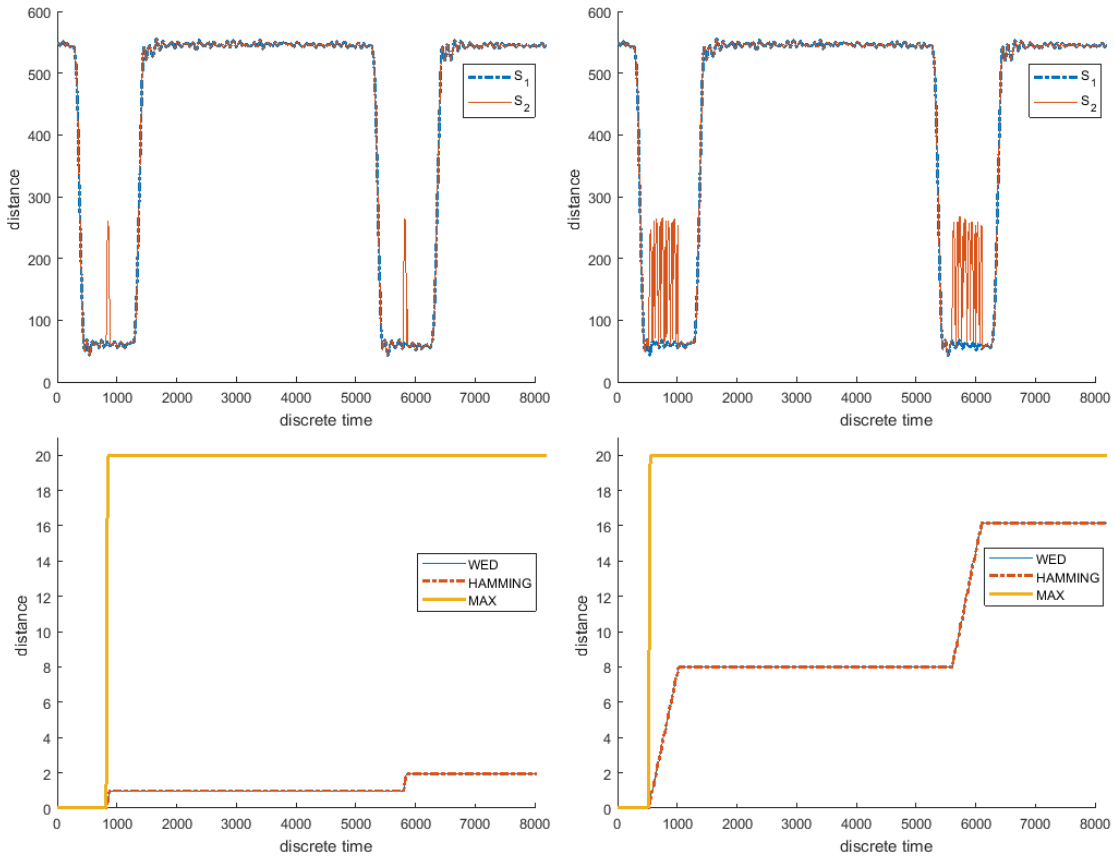


Figure 7.1: Measuring similarity $d_{i,i}(s_1, s_2)$ between a reference $s_1$vand a measured s_2 behavior - single versus multiple deviations.

are superimposed over the the non-pulse segments of the reference behavior. In contrast, d_W makes a distinction between the two situations and assigns a higher distance to the third scenario.

Finally, we illustrate the difference between the weighted edit distance and the classical edit distance (d_E). Figures 7.3 (a) and (b) show two scenarios consisting of a reference and a measured behavior, that are in both cases the same, except for a short spike in each pulse. In the first scenario, the magnitude of the spike is smaller than in the second scenario. Figures 7.3 (c) and (d) depict the evolution of d_W and d_E over time. We can see that in contrast to d_W, d_E cannot distinguish between the two scenarios because its substitution operation has a fixed cost.

7.1.1 Sampling, Quantization and Weighted Edit Distance

We compute the WED between digital signals resulting from physical behavior observations after sampling and quantization. In this section, we discuss the effect of inaccuracies introduced by these operations on the WED.

Let s be an analog signal, T a sampling period and Q a quantization step. We assume that s has

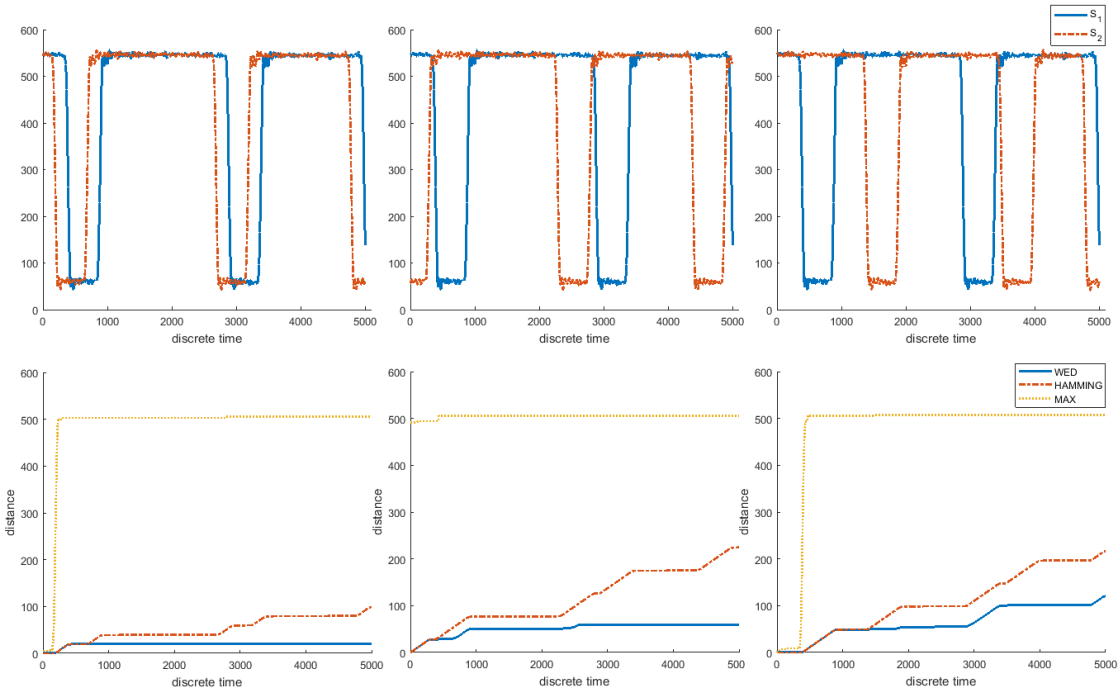


Figure 7.2: Measuring similarity $d_{i,i}(s_1, s_2)$ between a reference s_1 and a measured s_2 behavior - phase shifts

a band limit f_M and $T \leq 1/(2f_M)$. We denote by $s[T]$ the discrete signal obtained from s by sampling with the period T, and by $s[T][\mathsf{Q}]$ the digital signal obtained from $s[T]$ by quantization with the step Q.

We cannot directly relate the WED to the analog signals, because it is not defined in continuous time. However, this distance allows tackling phase shifts in the sampled signals. Consider two analog signals $s_1(t)$ and $s_2(t - \tau)$ such that $\tau = iT$ for some $i \geq 0$ and their sampled variants $s_1[T](t)$ and $s_2[T](t)$. It is clear that with $2 \cdot i$ insertion and deletion operations, $s_2[T]$ can be transformed into $s_1[T]$ such that their remaining substitution cost equals to 0. This situation is illustrated in Figure 7.4 (see signals s_1 and s_2). We see that the distance between the two signals initially grows due to the insertion and deletion operations, but that eventually it becomes perfectly stable.

Now consider another signal $s_3(t) = s_1(t - \tau)$ such that τ is not a multiple of T. In this case, the sampled signal $s_3[T](t)$ cannot be perfectly transformed into $s_1[T](t)$ by using insertion and deletion operations because of the mismatch between the sampling period and the phase shift. As a consequence, the distance between $s_1[T](t)$ and $s_3[T](t)$ will accumulate substitution costs due to this mismatch. This scenario is also depicted in Figure 7.4 (see signals s_1 and s_3). The figure shows that after an initial steep increase of the distance due to the insertion and deletion operations, its value does not converge, but continues slowly increasing due to the accumulation of remaining substitution costs.

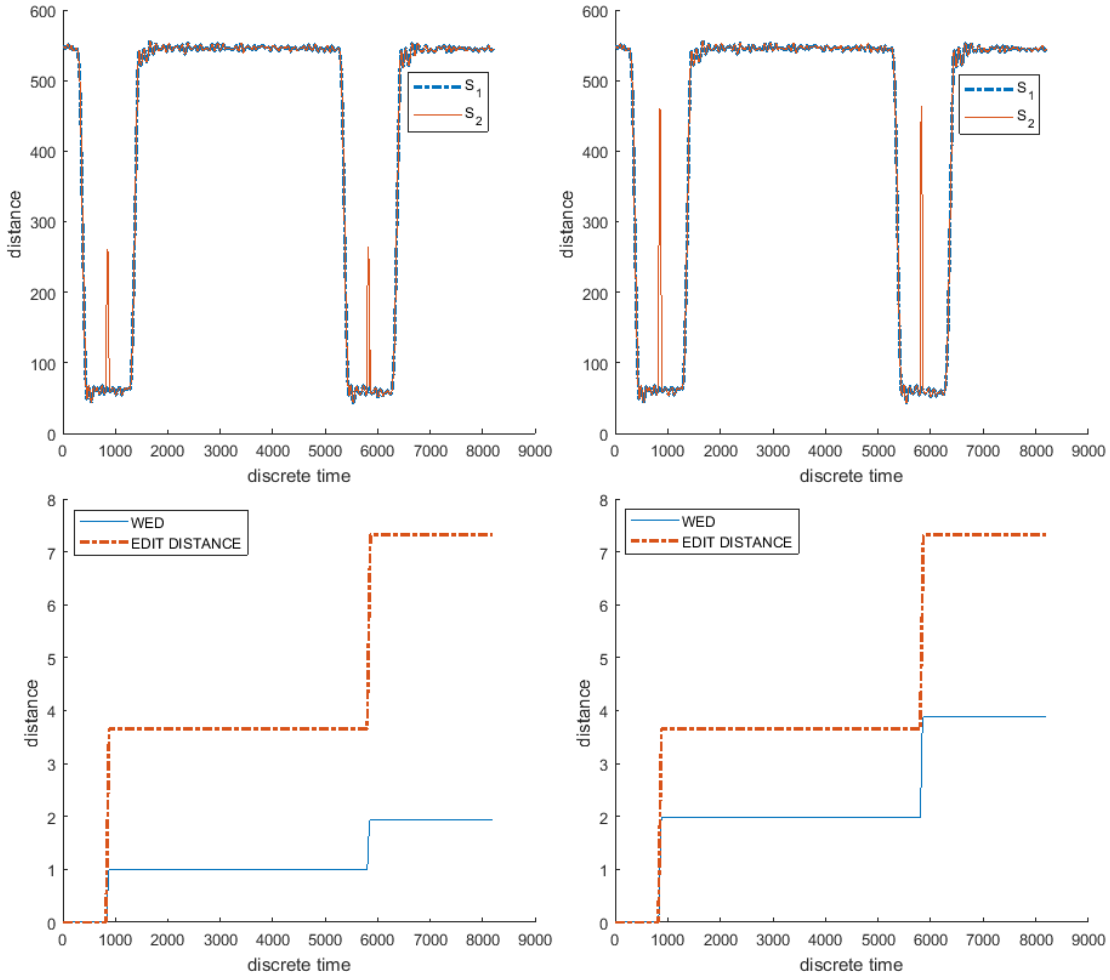


Figure 7.3: Measuring similarity $d_{i,i}(s_1, s_2)$ between a reference s_1 and a measured s_2 behavior - magnitude of deviations.

It is clear that the actual distance between two behaviors is affected by the sampling frequency. We refer to [Uns00] for the survey on the sampling theory, a field that studies the effects of sampling continuous behaviors.

7.1.2 Normalized Weighted Edit Distance

The weighted edit distance is an accumulative distance. It follows that the distance between two behaviors depends on several factors, including: 1) the size of the value domains; 2) the frequency at which the two signals are sampled; and 3) the total duration of the trace. For instance, the comparison of two analog behaviors sampled at two different frequencies can result in completely different absolute distance values. In order to have a more uniform robustness valuation that is less affected by the above factors, we propose *normalizing* the robustness values as follows.

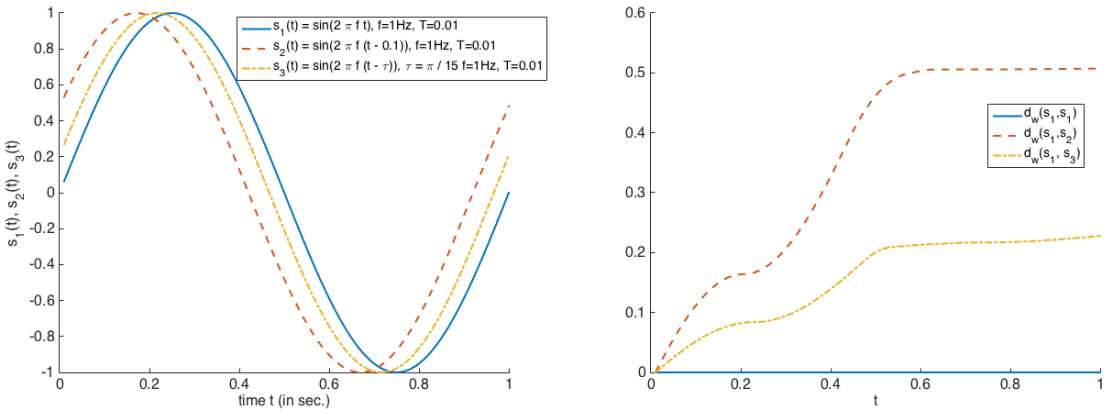


Figure 7.4: Weighted edit distances $d_W(s_1, s_2)$ and $d_W(s_1, s_3)$, where $s_1(t) = \sin(2\pi ft)$, $s_2(t) = \sin(2\pi f(t - 0.1))$, $s_3(t) = \sin(2\pi f(t - \tau))$, $T = 0.01$, $f = 1Hz$ and $\tau = \pi/15$.

Given signals s_1, s_2 of length l defined over X, the value domain $\mathbb{D} = [v_{min}, v_{max}]$, we define the normalized weighted edit distance, which is always bounded by $[0, 1]$ as follows:

$$d_W^{\#}(s_1, s_2) = \frac{d_W(s_1, s_2)}{l|X|(v_{max} - v_{min})}.$$

7.2 Weighted Edit Robustness for STL

In this section we start with extending the definitions from Chapter 3 in order to define *element-set* distance, robustness degree and a value of a signal in symbolic weighted automaton $\mathcal{W}$.

7.2.1 Robustness Degree

A metric space is a set for which distances between all elements in the set are defined.

Given $m \in \mathcal{M}$ and $M \subseteq \mathcal{M}$, we can lift the definition 12 to reason about the distance between an element m of $\mathcal{M}$ and the subset M of $\mathcal{M}$ as follows

$$d(m, M) = \inf_{m' \in M} d(m, m')$$

Definition 20. *We define the* robustness degree $\rho(m, M)$ *of* m *with respect to the set* M *as follows:*

$$\rho(m, M) = \begin{cases} d(m, \mathcal{M} \backslash M) & \text{if } m \in M \\ -d(m, M) & \text{otherwise.} \end{cases}$$

We now build on definition 8 to define the value of a signal s in $\mathcal{W}$.

Let s be a signal of size l and $\pi = q_0 \cdot \delta_0 \cdots \delta_{n-1} \cdot q_n$ a path in $\mathcal{W}$ induced by s. The value of π in $\mathcal{W}$ subject to s, denoted by $v_\pi(s, \mathcal{W})$, is the sum of weights associated to the transitions in the

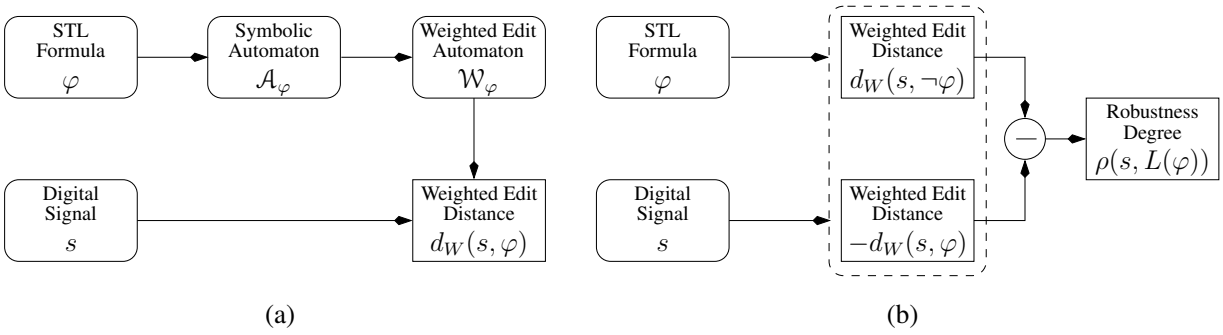


Figure 7.5: Computation of (a) Weighted Edit Distance $d_W(s,\varphi)$ and (b) Robustness Degree $\rho(s,\varphi)$.

path π and subject to the signal s. We define the *value* $v(s,\mathcal{W})$ of s as the *minimum* value from all the paths in $\mathcal{W}$ induced by s, i.e. $v(s,\mathcal{W}) = \min_{\pi\in\Pi(\mathcal{W},s)} v_\pi(s,\mathcal{W})$.

The following section defines a novel procedure for computing the *robustness degree* of a discrete signal with respect to an STL property. In our approach, we set c_i and c_d to be equal to $|X|(v_{max} - v_{min})$. In other words, the deletion and insertion costs are equal to the largest substitution cost. The rational behind this choice is that by inserting/deleting a data point, we can add/remove the maximum value from the domain in the worst case.

7.2.2 From STL to Weighted Edit Automata

Our procedure relies on computing the WED between a signal and a set of signals, defined by the specification. It consists of several steps, illustrated in Figure 7.5. We first translate the STL formula φ into a symbolic automaton $\mathcal{A}_\varphi$ that accepts the same language as the specification. The automaton $\mathcal{A}_\varphi$ treats timing constraints from the formula enumeratively, but keeps symbolic guards on data variables[1]. We then transform $\mathcal{A}_\varphi$ into a *weighted edit automaton* $\mathcal{W}_\varphi$, a weighted symbolic automaton that accepts all the signals but with the value that corresponds to the WED between the signal and the specification (Figure 7.5 (a)). We propose an algorithm for computing this distance. Computing the robustness degree between a signal and an STL specification follows from the calculation of their WED, as shown in Figure 7.5 (b).

Let X be a set of finite variables defined over the domain $\mathbb{D} = [v_{min}, v_{max}] \subseteq \mathbb{N}$. We consider an STL formula φ defined over X. Let $s : [0,l) \times X \to \mathbb{D}$ be a digital signal.

From φ to $\mathcal{A}_\varphi$

In the first step, we translate the STL specification φ into the automaton $\mathcal{A}_\varphi$ such that $L(\varphi) = L(\mathcal{A}_\varphi)$. The translation from STL interpreted over discrete time and finite valued domains to finite automata is standard, and can be achieved by using for instance on-the-fly tableau

[1]The time in $\mathcal{A}_\varphi$ cannot be treated symbolically with digital clocks since every pair of states and clock valuation may behave differently with respect to the WED.

construction [GPVW96] or the temporal testers approach [PZ08]. We note that we need to accommodate these classic constructions to the finitary semantics of the temporal logic by adapting accordingly the acceptance conditions (see for instance [EFH+03] for the interpretation of LTL over finite traces).

Example 12. *Consider the past* STL *formula* $\varphi = \square(x = 4 \rightarrow \diamondsuit(x < 3))$*, where* x *is defined over the domain* $[0, 5]$*. The resulting automaton* $\mathcal{A}_\varphi$ *is shown in Figure 7.6 (a).*

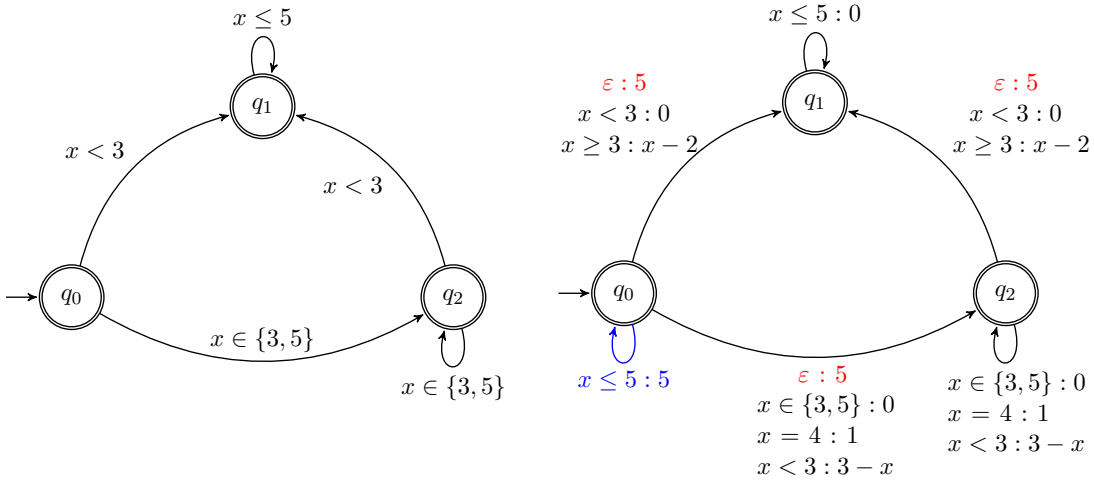


Figure 7.6: (a) A Symbolic Automaton $\mathcal{A}_\varphi$ accepting $L(\varphi)$ (b) a Weighted Edit Automaton $\mathcal{W}_\varphi$.

From $\mathcal{A}_\varphi$ to $\mathcal{W}_\varphi$

In this step, we translate the automaton $\mathcal{A}_\varphi$ to the weighted edit automaton $\mathcal{W}_\varphi$. The automaton $\mathcal{W}_\varphi$ reads an input signal and mimics the weighted edit operations. In essence, $\mathcal{W}_\varphi$ accepts every signal along multiple paths. Each accepting path induced by the signal corresponds to a sequence of weighted edit operations needed to transform the input signal into another one allowed by the specification. The value of the least expensive path corresponds to the weighted edit distance between the input signal and the specification. The weighted automaton $\mathcal{W}_\varphi$ explicitly treats substitution, insertion and deletion operations, by augmenting $\mathcal{A}_\varphi$ with additional transitions and associating to them the appropriate weight function. We now provide details of the translation and describe the handling of weighted edit operations. Let $\mathcal{A}_\varphi = (X, Q, I, F, \Delta)$ be the symbolic automaton accepting the language of the specification φ, where X takes values from a domain $\mathbb{D}$.

Substitution In order to address substitutions in the automaton, we define a new set of *substitution* transitions Δ_s and associate to them the weight function λ_s as follows. Given $q, q' \in Q$, let $\gamma(q, q') = \bigvee_{(q,\psi,q')\in\Delta} \psi$. Then, we have:

- $(q, \top, q') \in \Delta_s$ for all transitions $(q, \psi, q') \in \Delta$; and

- $\lambda_s((q, \top, q'), v) = d_M(v, \gamma(q, q'))$, for all $v \in \mathbb{D}^{|X|}$.

We define the Manhattan distance of valuation v from a $\gamma(q, q')$ as the minimum of Manhattan distances of the valuation v from all the possible valuations that satisfy $\gamma(q, q')$:
$d_M(v, \gamma(q, q')) = \min\{d_M(v, w)\}, \forall w$ such that $w \models \gamma(q, q')$.

Intuitively, we replace all the transitions in $\mathcal{A}_\varphi$ with new ones that have the same source and target states. We relax the guards in the new transitions and make them enabled for *any* input. On the other hand, we control the cost of making a transition with the weight function λ_s, which computes the substitution cost needed to take the transition with a specific input. This cost is the Manhattan distance between the input value and the guard associated to the original transition.

Deletion Addressing deletion operations consists in adding self-loop transitions that consume all the input letters to all the states with the deletion cost $c_d = |X|(v_{max} - v_{min})$, thus mimicking deletion operations. We skip adding a self-loop transition to states that already have the same substitution self-loop transition – according to our definition $c_d \geq c_s(a, X)$ for all a, hence taking the deletion transition instead of the substitution one can never improve the value of a path and is therefore redundant. We define the set of deletion transitions Δ_d and the associated weight function λ_d as follows:

- $(q, \top, q) \in \Delta_d$ if $(q, \top, q) \notin \Delta_s$; and
- $\lambda_d(\delta, v) = c_d$ for all $\delta \in \Delta_d$ and $v \in \mathbb{D}^{|X|}$.

Insertion In order to mimic the insertion operations, we augment the transitions relation of $\mathcal{W}_\varphi$ with silent transitions. For every original transition in Δ, we associate another transition with the same source and target states, but labeled with ϵ and having the insertion cost $c_i = |X|(v_{max} - v_{min})$. Formally, we define the set of insertion transitions Δ_i and the associated weight function λ_i as follows:

- $(q, \epsilon, q') \in \Delta_i$ if $(q, \psi, q') \in \Delta$ for some ψ; and
- $\lambda_i(\delta, \{\epsilon\}) = c_i$ for all $\delta \in \Delta_i$.

Given the symbolic automaton $\mathcal{A}_\varphi = (X, Q, I, F, \Delta)$ accepting the language of the specification φ, its associated Weighted Edit Automaton $\mathcal{W}_\varphi$ is the tuple $(X, Q, I, F, \Delta', \lambda')$, where:

- $\Delta' = \Delta_s \cup \Delta_d \cup \Delta_i$ and
- $\lambda'(\delta, v) = \lambda_s(\delta, v)$ if $\delta \in \Delta_s$ and
- $\lambda'(\delta, v) = \lambda_d(\delta, v)$ if $\delta \in \Delta_d$ and
- $\lambda'(\delta, \epsilon) = \lambda_i(\delta, \epsilon)$ if $\delta \in \Delta_i$.

Example 13. *The weighted edit automaton $\mathcal{W}_\varphi$ obtained from $\mathcal{A}_\varphi$ is illustrated in Figure 7.6 (b). Both automatons from the figure 7.6 (b) use the same input alphabet $\mathbb{D} = \{0, 1, 2, 3, 4, 5\}$. The blue transitions, such as $(A, 0, A)$ with weight 5, correspond to the deletion transitions. The red transitions, such as (A, ϵ, B), correspond to the insertion transitions.*

The resulting weighted automaton $\mathcal{W}_\varphi$ allows determining the weighted edit distance between a signal w and the formula φ, by computing the value of s in $\mathcal{W}_\varphi$.

Theorem 4. $d_W(s, \varphi) = v(s, \mathcal{W}_\varphi)$.

The proof of Theorem 4 is provided in Appendix A.2. The consequence of the Theorem 4 is that two symbolic automata that accept the same language will always give the same distance from the same input.

7.2.3 Computing the Value of a Signal in a Weighted Edit Automaton

We now present an on-the-fly algorithm Val, shown in Algorithm 5, that computes the value of a signal s in a weighted automaton $\mathcal{W}$. In every step i, the algorithm computes the minimum cost of reaching the state q with the prefix of s consisting of its first i values. After reading a prefix of s, we may reach a state $q \in Q$ in different ways with different costs. Note that it is sufficient to keep the state with the minimum value in each iteration. It follows that the algorithm requires book keeping $|Q|$ state value fields in every iteration. We now explain the details of the algorithm. The procedure first initializes the costs of all the states in $\mathcal{W}$ (see Algorithm 6). The initial states are set to 0 and the non-initial ones to ∞. Then, we compute the effect of taking the ϵ transitions without reading any signal value. It is sufficient to iterate this step $|Q|$ times, since within $|Q|$ iterations, one is guaranteed to reach a state q that was already visited with a smaller value v. In every subsequent iteration i, we first update the state values by applying the cost of taking all transitions labeled by $s(i, X)$ and then update the effect of taking ϵ transitions $|Q|$ times. The weight function of a substitution cost is computed as follows: $\lambda(v, x \leq k)$ gives 0 if $v \leq k$, and $v - k$ otherwise; $\lambda(v, \neg(x \leq k))$ is symmetric; $\lambda(v, \varphi_1 \wedge \varphi_2) = \max(\lambda(v, \varphi_1), \lambda(v, \varphi_2))$ and $\lambda(v, \varphi_1 \vee \varphi_2) = \min(\lambda(v, \varphi_1), \lambda(v, \varphi_2))$.

Upon termination, the algorithm returns the minimum cost of reaching an accepting state in the automaton.

Theorem 5. *Val*$(s, \mathcal{W}) = v(s, \mathcal{W})$.

Theorem 6. *Given a signal s of length l defined over X and a weighted automaton $\mathcal{W}$ with n states and m transitions,* Val$(s, \mathcal{W})$ *takes in the order of $\mathcal{O}(lnm))$ iterations to compute the value of s in $\mathcal{W}$, and requires in the order of $\mathcal{O}(n(\lceil log(l(v_{\max} - v_{\min}))\rceil))$ memory.*

We refer the reader to Appendix A.2, where proofs for Theorem 5 and Theorem 6 are presented.

Algorithm 5 $\mathsf{Val}(s, \mathcal{W})$

Require: s and $\mathcal{W}_\psi$
Ensure: v
  $\mathsf{InitVal}(\mathcal{W})$
  for all $i \in [0, l)$ **do**
    for all $\delta = (q, \gamma, q') \in \Delta$ **do**
      $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(s(i, X), \delta))$
    end for
    for $i = 0; i < |Q|; i++$ **do**
      for all $\delta = (q, \epsilon, q') \in \Delta$ **do**
        $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(\delta, \epsilon))$
      end for
      for all $q \in Q$ **do**
        $v(q) \leftarrow v'(q)$
        $v'(q) \leftarrow \infty$
      end for
    end for
  end for
  $v \leftarrow \min_{q \in F} v(q)$
    return v

Algorithm 6 $\mathsf{InitVal}(\mathcal{W})$

  for all $q \in Q$ **do**
    $v(q) \leftarrow (q \in I)\ ?\ 0\ :\ \infty; v'(q) \leftarrow \infty$
  end for
  for $i = 0; i < |Q|; i++$ **do**
    for all $\delta = (q, \epsilon, q') \in \Delta$ **do**
      $v'(q') \leftarrow \min(v'(q'), v(q) + \lambda(\delta, \epsilon))$
    end for
    for all $q \in Q$ **do**
      $v(q) \leftarrow v'(q)$
      $v'(q) \leftarrow \infty$
    end for
  end for

Example 14. *Consider the STL property φ from Example 12, the associated weighted edit automaton $\mathcal{W}_\varphi$ from Figure 7.6 and the signal*[2] $s : [0, 2] \to [0, 5]$ *such that $s(0) = 5$, $s(1) = 5$ and $s(2) = 4$. It is clear that $(s, 0) \not\models \varphi$, since $s(2) = 4$, while there was not a single $0 \leq i < 2$ where $s(i) < 3$. We illustrate in Figure 7.7 the computation of $v(s, \mathcal{W}_\varphi)$. We can see that with the signal s, we can reach one of the accepting states (B or C) with the value 1. This value corresponds to one substitution operation, replacing the value of 4 in $s(2)$ by 5, which allows vacuous satisfaction of the property φ.*

[2] Since s has only one component, we skip the variable name.

		$s(0)=5$		$s(1)=5$		$s(2)=4$		
A	0	0	5	5	10	10	15	15
B	∞	5	3	3	3	3	3	3
C	∞	5	0	0	0	0	1	1
		init	ϵ-init	update	ϵ-update	update	ϵ-update	update	ϵ-update

Figure 7.7: Example - computation of $v(s, \mathcal{W}_\varphi)$.

7.2.4 WED-based robustness and TRE specifications

At this point we discuss the necessary adaptations of the algorithm which would enable calculating WED-based robustness of a trace w.r.t. requirements specified in Timed Regular Expressions. We have seen that our WED-based robustness algorithm works with a weighted edit automaton, which is directly obtained from an acceptor of the language defined by the specification. Hence, the algorithm can be also used for specifications expressed in Timed Regular Expressions [ACM97]. As noted in Chapter 3, we interpret TRE over discrete time due to the hardware setting of our monitors. In that case, we could use the procedure described in [ACM02b], to obtain a timed automaton from a given TRE specification. Since our specifications are interpreted over discrete time, we could simply enumerate the time in the timed automaton and obtain an untimed finite automaton.

In case the input alphabet does not allow for a natural order, we would not use symbolic constraints on our transitions. The enumeration of input-specific transitions, which would be necessary in that case, would yield a bigger automata. Furthermore, we would have to define a specific cost function for substitution operations, rather than using *Manhattan distance* over ordered sets.

Our weighted edit automata use ε-transitions to insert operations on the input trace. Such transitions allow transiting to a state without consuming an input letter. These transitions are introduced during the automaton decoration process, and must be distinguished from the transitions which correspond to consuming an empty string symbol (ε) which is a legal input symbol in TRE. Taking a transition guarded by an empty string symbol would incur zero cost.

7.3 Evaluation

We now evaluate the WED-based quantitative monitors for STL. In order to evaluate our approach, we conducted two automotive benchmarks. We take specification from automotive benchmarks published in [BHF15].

In order to translate STL properties into temporal testers, we start with a parser for STL formulas in Java which we developed using parser generator ANTLR [Par13]. We take temporal testers for basic STL operators and create their parallel composition. Then, we convert such top level

φ_1	$\square(\omega < 4500)$
φ_2	$\square((\omega < 4500) \wedge (v < 120))$
φ_3	$\square((g_2 \wedge \bigcirc g_1) \rightarrow \square_{(0,2.5]} \neg g_2)$
φ_4	$\square((\neg g_1 \wedge \bigcirc g_1) \rightarrow \square_{(0,2.5]} g_1)$
φ_5	$\bigwedge_{i=1}^{4} \square((\neg g_i \wedge \bigcirc g_i) \rightarrow \square_{(0,2.5]} g_i)$
φ_6	$\neg(\diamondsuit_{[0,4]}(v > 120) \wedge \square(\omega < 4500))$
φ_7	$\diamondsuit_{[0,4]}((v > 120) \wedge \square(\omega < 4500))$
φ_8	$((g_1 \,\mathcal{U}\, g_2 \,\mathcal{U}\, g_3 \,\mathcal{U}\, g_4) \wedge \diamondsuit_{[0,10]}(g_4 \wedge \diamondsuit_{[0,2]}(\omega > 4500))) \rightarrow$ $\diamondsuit_{[0,10]}(g_4 \rightarrow \bigcirc(g_4 \,\mathcal{U}_{[0,1]}(v \geq 120)))$

Table 7.1: Automatic Transmission properties [BHF15].

temporal tester into an acceptor automaton, using the approach described in Section 4.4.4. We use JAutomata [jau] library to represent the testers and the acceptors. Finally, we generate the Verilog code of a quantitative monitoring module. The resulting monitor is a hardware implementation of the weighted automata and the underlying algorithm for computing the weighted edit distance. The monitor operates at the frequency limited by the maximum achievable frequency of the FPGA. The programmable logic (FPGA part) of Xilinx Zynq device we used, has a theoretical frequency bound of 866 MHz.

7.3.1 Benchmarks for Automotive Systems

For the evaluation of our approach, we apply it to two benchmarks implemented in Matlab/Simulink and published in [BHF15].

Automatic Transmission System

We first consider the slightly modified Automatic Transmission deterministic Simulink demo provided by Mathworks as our System Under Test. It is a model of an automatic transmission controller that exhibits both continuous and discrete behavior. The system has two inputs – the throttle u_t and the break u_b. The break allows the user to model variable load on the engine. The system has two continuous-time state variables – the speed of the engine ω (RPM), the speed of the vehicle v (mph) and the active gear g_i. The system is initialized with zero vehicle and engine speed. It follows that the output trajectories depend only on the input signals u_t and u_b, which can take any value between 0 and 100 at any point in time. The Simulink model contains 69 blocks including 2 integrators, 3 look-up tables, 2 two-dimensional look-up tables and a Stateflow chart with 2 concurrently executing finite state machines with 4 and 3 states, respectively. The benchmark defines 8 STL formalized requirements that the system shall satisfy, shown in Table 7.1.

We now describe the evaluation setup. We simulated the Simulink model with fixed-step sampling and recorded the results. The obtained traces, as the one shown in Figure 7.8, were then further discretized with the uniform quantization. We have obtained 751 samples from the Simulink model and normalized all variables' value domain to the interval $[0, 5000]$ which is the range of RPM variable, thus achieving fair reasoning about their substitution cost. We designed a testbench in Verilog to stimulate the monitor with generated values from the Simulink model. We used Xilinx Vivado to perform monitor simulation and synthesis.

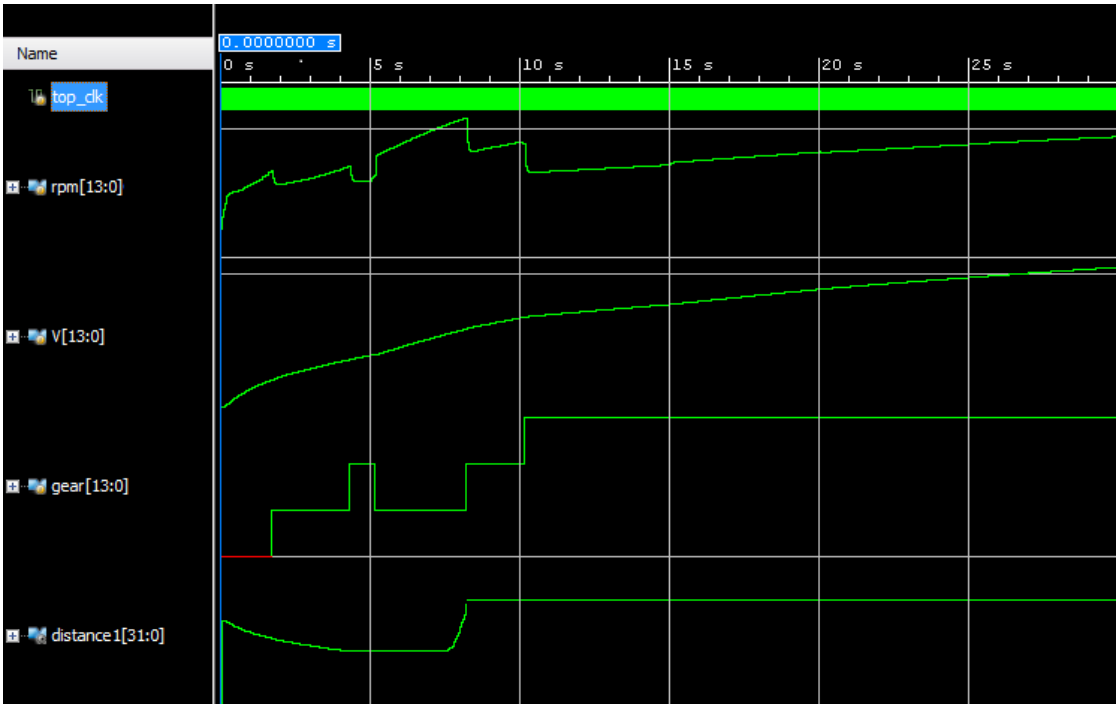


Figure 7.8: A simulation trace s from the Automatic Transmission model and $d_W(s, \neg\varphi_6)$.

Figure 7.8 illustrates the monitoring results for φ_6 on a specific gear input. In the depicted scenario, the speed does not reach $120\ mph$ in 4 seconds, a sufficient condition for the satisfaction of the formula. In order to violate the formula, we need to alter both v and ω signals such that 1) v reaches $120\ mph$ at any moment within the first 4 seconds; and (2) ω remains continuously below $4500\ rpm$. These alterations result in (1) a single substitution happening within the first 4 seconds which is necessary to bring v to $120\ mph$; and (2) the accumulation of substitution costs in the interval between 7 and 8 seconds of the simulation where ω actually exceeds 4500 rpm. Note that the robustness degree decreases in the first 4 seconds. This happens because the actual v increases and the substitution cost needed for v to reach $120\ mph$ is continuously being improved.

The evaluation results are shown in Table 7.2. We tested the correctness of STL to automata translation by generating both acceptors for φ and $\neg\varphi$. The presented robustness degrees are not normalized, which can be statically computed using the formula from Section 7.1. It is clear

		$\mathcal{W}_\varphi$				$\mathcal{W}_{\neg\varphi}$			
φ	ρ	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT
φ_1	-2528	2	2	62	260	4	8	94	657
φ_2	-11423	2	2	75	306	4	11	107	799
φ_3	1000	496	1374	4106	53033	992	2878	8127	106937
φ_4	1000	496	692	3061	22777	992	1445	6025	44968
φ_5	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
φ_6	5337	405	813	6540	66085	409	903	6504	73657
φ_7	-5336	403	903	6504	73766	405	813	6545	66116
φ_8	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

Table 7.2: Evaluation results for the Automatic Transmission benchmark. We report on the resulting robustness degree ρ, the number of states $|Q|$ and transitions $|\Delta|$ of Weighted Edit Automata $\mathcal{W}_\varphi$ and $\mathcal{W}_{\neg\varphi}$, as well as the number of flip-flops (#FF) and lookup tables (#LUT) consumed to implement the monitors.

from our table that either the distance from φ or from its negation is always 0. The dominant type of resources when implementing our monitors on FPGA hardware are LUTs. This is not surprising, due to the large combinatorial and arithmetic requirements of the computation. We can also note that the size of our monitors is sensitive to the timing bounds in the formulas and the sampling period of the input signals. Our monitor automata enumerate clock ticks instead of using a symbolic representation. The enumeration is necessary because state - clock valuation pairs can have different values associated and thus cannot be grouped. We were not able to generate monitors for φ_5 and φ_8 due to the state explosion. However, φ_5 can be decomposed into 4 independent sub-properties. We can see several ways to handle large properties such as φ_8 that we will investigate in the future – by reformulating the specification using both past and future operators, by using larger sampling periods (smaller time bounds in the formula) and by using more powerful FPGA.

Fault-Tolerant Fuel Control System

The second automotive benchmark is based on Fault-Tolerant Fuel Control System model [BHF15, mat]. This system ensures proper air-to-fuel ratio in modern car engines. It must be adaptive to any kind of external failures, such as sensor failures. Since the occurrence of failures is modeled by Poisson stochastic processes, this benchmark will evaluate our quantitative monitors with a model of a Stochastic Cyber Physical System.

The system has throttle as an input which affect failure arrival rates. The change in detected fuel level can be caused either by throttle or a sensor failure. Such change directly affects air-to-fuel ratio λ which is the output of the model. We sample this variable over time in order to create

stimulus for our monitors. We collected 10000 samples from the model output. We rounded double precision output to 2 decimals, and multiplied it by 100 for easier representation in hardware testbench.

The requirement for air-to-fuel ratio λ specifies that no matter what kind of disturbance in system occurs, the value of λ must stabilize within certain value limit in specified time window. We call this a bounded stabilization property and formalize it in STL with the following formula:

$$\varphi_9 = \square(\lambda > V_{limit} \rightarrow \diamondsuit_{[0,1s]}\square_{[0,1s]}(\lambda < V_{limit}))$$

As suggested by the authors of [BHF15], we use $V_{limit} = 1.1 \cdot V_{id}$, where V_{id} corresponds to ideal air-to-fuel ratio when no throttle change or a sensor failure occurs.

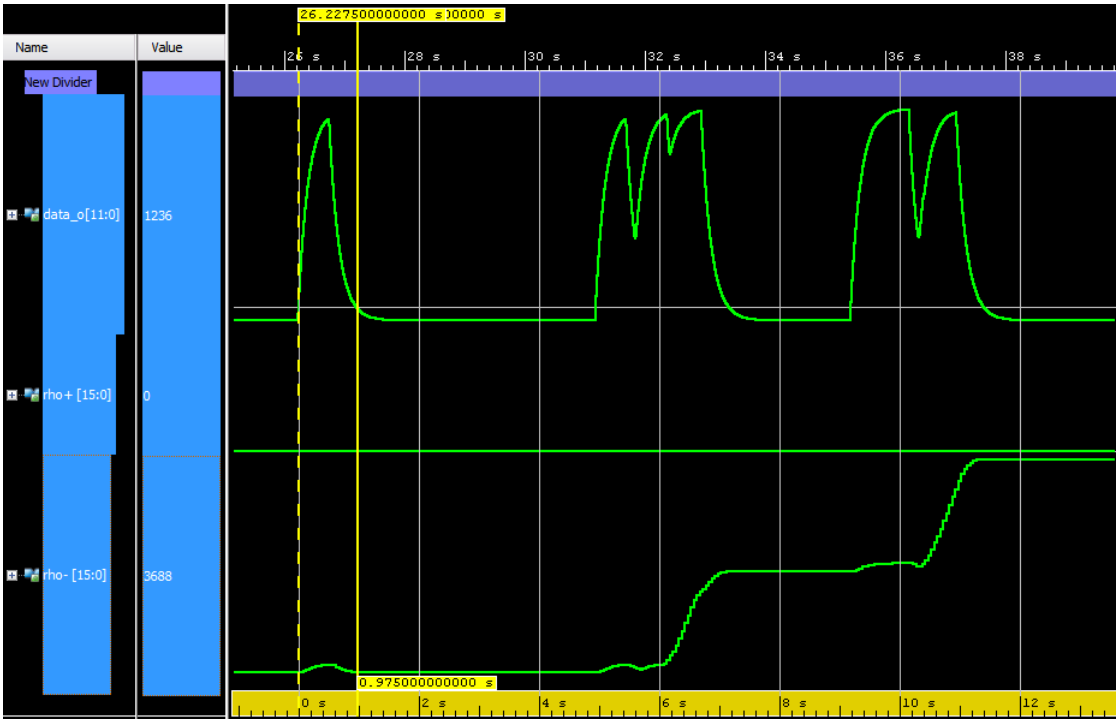


Figure 7.9: Calculated positive and negative robustness for obtained air-to-fuel ratio λ.

In Fig 7.9 we can observe change of λ and robustness values w.r.t. the formula. We see several λ pulses caused by the disturbance in the system. Due to the initial conditions, negative robustness is greater than zero. The first pulse is satisfying the requirement since it stabilizes to required V_{limit} within 1 second time window. Since it satisfies the bounded stabilization property and does not add any WED cost, the negative robustness value remains the same before and after the pulse.

The next disturbance in the system generates more impact on air-to-fuel ratio. In this case the signal does not stabilize fast enough. Therefore the WED algorithm suggests to substitute problematic parts of the trace with correct values. Since the substitution costs accumulate, the

		$\mathcal{W}_\varphi$				$\mathcal{W}_{\neg\varphi}$			
φ	ρ	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT
φ_9	-43878	882	1493	13203	119989	1574	2648	23624	212341

Table 7.3: Evaluation results for Fault-Tolerant Fuel Control System properties [BHF15]. We report on the resulting robustness degree ρ, the number of states $|Q|$ and transitions $|\Delta|$ of Weighted Edit Automata $\mathcal{W}_\varphi$ and $\mathcal{W}_{\neg\varphi}$, as well as the number of flip-flops (#FF) and lookup tables (#LUT) consumed to implement the monitor.

negative robustness keeps increasing. Positive robustness equals zero throughout the simulation due to the fact that the trace is violating the formula from the start.

We report on monitor size and FPGA resource consumption in Table 7.3. The large negative robustness value is due to several bounded stability violations and significantly larger number of samples compared to Automatic Transmission System testbench. Increased resource consumption is consequence of the fact that the formula uses future time STL operators over bounded time intervals.

CHAPTER 8

Industrial case study: SENT protocol

The V&V process for automotive electronic systems involves extensive product testing, such as long duration stress test. Current industry practice relies on hard-crafted checkers, that lack rigor and fail to provide usable diagnostics information. Existing tools for offline trace verification (e.g. the AMT [NM07, NLM+18]) are not directly applicable in this context, due to the excessive size of the resulting traces. Moreover, it is also often the case that a long-term test takes several days of real-time execution.

Runtime monitors based on formal semantics, allow immediate detection of requirement violation, enable monitoring of long traces, regardless of their source, and allow for automatic synthesis of monitors from requirements. Correctness monitors can be used to verify hard requirements, whereas robustness monitors can verify soft design constraints.

In this work, we translate high-level specifications into monitors implemented in an FPGA and run them in parallel with the system under investigation, in the spirit of the approach from Chapter 5. However, this time we are using an industrial tool, called High-Level Synthesis (HLS) to perform the monitor synthesis.

To fully demonstrate the usability of our monitors, we conducted a full-scale case study of an industrial sensor which implements Single Edge Nibble Transmission (SENT) [Int16b] protocol. We start by providing formalization of the SENT requirements in Signal Temporal Logic and Timed Regular Expressions. We focus on qualitative online monitors able to detect multiple violations, recover and log them for further analysis. We verify electrical and timing requirements of the SENT protocol in an online fashion. In addition, we conduct a smaller case study of SENT protocol, where we perform WED-based *robustness* monitoring for selected STL properties. The content of this chapter is mostly based on the work published in [SJN+17].

8.1 Formalization of the SENT Protocol

In this section we introduce the Single Edge Nibble Transmission Protocol (SENT), and then formalize a subset of its electrical and timing requirements. The SENT standard (SAE J2716 [Int16a]) defines unidirectional communication scheme for devices continuously transmitting data, such as sensors connected to Engine Control Unit (ECU). The simplicity and flexibility of this protocol results in reduced cost, making it an attractive choice for Original Equipment Manufacturers (OEMs) in automotive industry.

The protocol is mainly used in automotive applications, for instance, in an Electronic Power Steering (EPS), or an Electronic Braking System (EBS). In these applications, the Hall-effect magnetic sensor is used to detect the angle of the steering wheel or the position of a braking pedal. This solution is popular because it eliminates friction between the mechanical elements, thus making the whole control system more durable. The magnetic sensor is using SENT protocol to transfer data to the ECU. Since these electronic systems are safety-critical, correct information transfer and runtime error detection is of utter importance.

8.1.1 Single Edge Nibble Transmission Protocol

The SENT protocol is an industry standard (SAE J2716 [Int16a]) for transmitting data between a sensor and a controller.

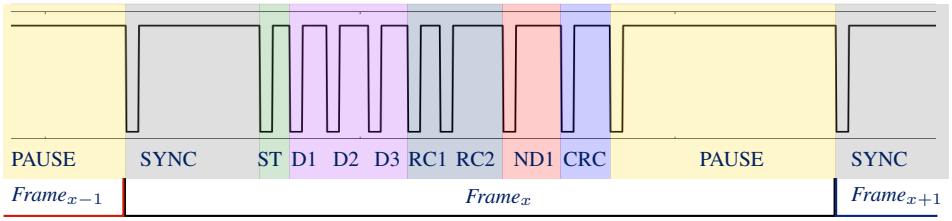


Figure 8.1: A SENT frame starts with a mandatory synchronization pulse (SYNC), followed by a status nibble (ST), data nibbles (D1, D2, D3), rolling counters (RC1, RC2), bit inverse of D1 (ND1), cyclic redundancy check (CRC), and finishes with an optional pause.

SENT communication is unidirectional from a sensor to a controller; the information is partitioned into frames with the structure shown in Fig. 8.1. The transmitted data is split in four-bit data chunks, so-called nibbles, which encode the data in their length. Each nibble has the shape depicted in Fig. 8.2, where the length of the 'H' region determines the transmitted value (from 0 to 15). In the case study we build runtime monitors for magnetic sensors, which transfer angular information encoded in three data nibbles D1-D3.

The SAE J2716 standard admits several frame configurations (e.g. the number of data nibbles may vary). SENT devices are configured prior to operation, and the configuration does not change on-the-fly; we take this into account and also assume that the frame structure is static and cannot change at runtime.

	SENT requirement	TRE specification
1	The fall time from V_1 to V_2 must be no longer than T_{fall} μs	`F`
2	The rise time from V_2 to V_1 must be no longer than T_{rise} μs	`R`
3	The signal stabilization time below low threshold V_1 or above high threshold V_2 must be at least T_{stable} μs	$\texttt{ST}_{\texttt{low}}$, `L`, $\texttt{ST}_{\texttt{high}}$

Table 8.1: SENT Electrical Interface Requirements given in natural language, together with their TRE formalization.

To be able to correctly decode sensor data, a controller needs to receive a signal that satisfies electrical and timing requirements of the SENT protocol. We now state these requirements formally, both in STL and TRE and elaborate on checking the frame correctness. Fig. 8.2 shows a SENT nibble and graphically depicts the requirements to be checked. Tables 8.1, 8.2 present in natural language a subset of electrical and timing requirements of the SENT protocol.

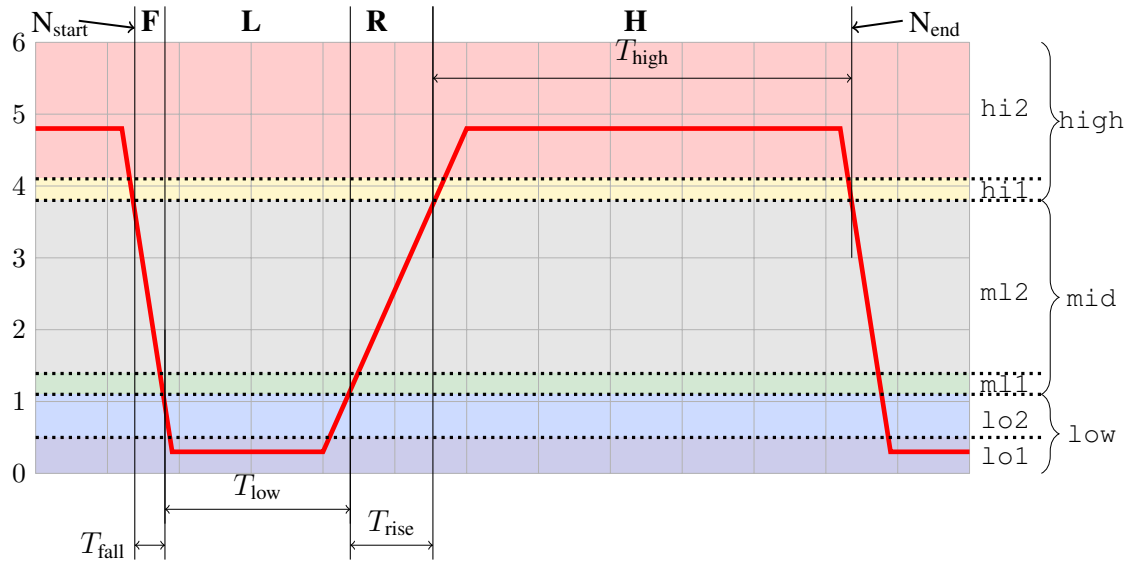


Figure 8.2: SENT nibble pulse: A pulse starts (N_{start}) with a falling edge F, followed by a low region L, followed by a rising edge R, followed by a high region H.

8.1.2 SENT requirements in STL

Before proceeding to the formalization of the protocol, we would like to provide a quick reference of the syntax of Signal Temporal Logic, which is originally defined in Chapter 3. The syntax of a

	SENT requirement	TRE specification
4	The synchronization pulse shall have a nominal period of 56 clock ticks.	SYNC
5	Five clock ticks of the synchronization pulse shall be driven low.	L
6	All the remaining clock ticks of the calibration (synchronization) pulse shall be driven high.	SYNC, H_{sync}
7	Five clock ticks of the nibble pulse shall be driven low.	L
8	All the remaining clock ticks of the nibble pulse shall be driven high.	NIBBLE, H_{nibble}
9	The minimum pulse period of the nibble pulse shall be 12 clock ticks.	NIBBLE, H_{nibble}
10	The maximum pulse period of the nibble pulse shall be 27 clock ticks.	NIBBLE, H_{nibble}

Table 8.2: SENT Transmission Requirements of Synchronization & Nibble Pulse in natural language and formalized in TRE.

STL formula φ over set of predicates $P \cup X$ is defined by the grammar:

$$\varphi \quad := \quad p \mid x \sim u \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \mid \varphi_1 \mathcal{S}_I \varphi_2$$

where $p \in P$, $x \in X$, $\sim \in \{<, \leq\}$, $u \in \mathbb{D}$, I is of the form $[a, b]$ or $[a, \infty)$ such that $a, b \in \mathbb{N}$ and $0 \leq a \leq b$. For intervals of the form $[a, a]$, we will use the notation $\{a\}$ instead.

Electrical Interface Requirements We specify the duration of the slopes, as well as the minimum stable time of the SENT signal. The STL formulas (Eq. 1-4) capture the temporal order in which the signal should cross voltage regions from Fig. 8.2. F and R (Eq. 1, 2) are the formal representations of falling and rising time requirements (Tab. 8.1, Req.1,2). The signal stabilization requirement (Tab. 8.1,Req.3) is mapped to two STL formulas (Eq. 3, 4) that deal separately with both thresholds. The STL formulas are written using past temporal operators: in this type of formulation a consequent should have happened before an antecedent (i.e. the form "whenever at a time step i φ holds, ψ should have held at $(i - I) \cap \mathbb{T}$").

$$\mathtt{F} = \textbf{rise}(\mathtt{low}) \rightarrow \mathtt{mid}\ \mathcal{S}_{[0,T_{\mathrm{fall}}]}\ \textbf{fall}(\mathtt{high}) \tag{1}$$

$$\mathtt{R} = \textbf{rise}(\mathtt{high}) \rightarrow \mathtt{mid}\ \mathcal{S}_{[0,T_{\mathrm{rise}}]}\ \textbf{fall}(\mathtt{low}) \tag{2}$$

$$\mathtt{ST}_{\mathtt{low}} = \textbf{fall}(\mathtt{low}) \rightarrow \boxminus_{[0,T_{stable}]}\ \mathtt{low} \tag{3}$$

$$\mathtt{ST}_{\mathtt{high}} = \textbf{fall}(\mathtt{high}) \rightarrow \boxminus_{[0,T_{stable}]}\ \mathtt{high} \tag{4}$$

Transmission Properties of Synchronization & Nibble Pulses. The synchronization and the nibble pulse requirements (Tab.8.2, 4-6 and 7-10 respectively) describe the timing properties these pulses should adhere to. A synchronization pulse has a pre-defined length and is considered as the start of a frame. The shape of synchronization and nibble pulses is to be checked as well (see Fig.8.2).

To verify the form of the synchronization, nibble, and pause pulses, we split each pulse in regions F, l, r, h (see Fig 8.2) and check temporal precedence of the regions. The total length of the pulses and the length of the low region L are given in "clock ticks" (Tab. 8.2, 4-5, 7, 9-10), which are generated by a sensor's internal clock. Let us denote $\delta = (T_{\text{rise}} + T_{\text{fall}})$, then the allowed durations of the H region for the nibble pulse and synchronization pulse are $[7\text{tick} - \delta, 22\text{tick} - \delta]$ and $(51\text{tick} - \delta)$, respectively. Similarly, the length of the H region of the pause pulse is within the following bounds: $[7\text{tick} - \delta, 122\text{tick} - \delta]$.

Requirements for L and H regions can be written directly in past-STL:

$$\texttt{L} = \texttt{fall(low)} \rightarrow \boxminus_{[0,5\text{ticks}]}\ \texttt{low} \tag{5}$$

$$\texttt{H}_{\text{sync}} = \texttt{fall(high)} \rightarrow \texttt{high}\ \mathcal{S}_{\{51\text{tick}-\delta\}}\ \texttt{rise(high)} \tag{6}$$

$$\texttt{H}_{\text{nibble}} = \texttt{fall(high)} \rightarrow \texttt{high}\ \mathcal{S}_{[7\text{tick}-\delta,\ 22\text{tick}-\delta]}\ \texttt{rise(high)} \tag{7}$$

$$\texttt{H}_{\text{pause}} = \texttt{fall(high)} \rightarrow \texttt{high}\ \mathcal{S}_{[7\text{tick}-\delta,\ 122\text{tick}-\delta]}\ \texttt{rise(high)} \tag{8}$$

The general way of capturing precedence relation in STL is by using the bounded until operator $\mathcal{U}_I$. As we show in [JBG+15] show, the hardware implementation of $\mathcal{U}_I$ is not scalable w.r.t. operator time bounds. In order to overcome this issue, we avoid using nested $\mathcal{U}_I$ operators in the formulation, and reformulate the properties. Each SYNC, NIBBLE, and PAUSE patterns of the SENT protocol are the requirements F, L, R, and the corresponding $\texttt{H}_{\{\text{sync}|\text{nibble}|\text{pause}\}}$ requirement put in a sequence. In order to attain efficient hardware implementation, we (i) re-state assertions from $\varphi \rightarrow \psi$ to $\psi \wedge \varphi$, to capture the events when the corresponding requirement has been satisfied; (ii) we then define precedence relation with following macro: $\varphi_1 \texttt{before}_{[t_1,t_2]} \varphi_2 = \varphi_2 \wedge \boxminus_{[0,t_1]} \neg\varphi_1 \wedge \diamondminus_{[t_1,t_2]} \varphi_1$. This allows to use hardware-cheap bounded historically $\boxminus_{[0,t_1]}$ and bounded once $\diamondminus_{[t_1,t_2]}$ operators and significantly reduce hardware resources.

The requirement for NIBBLE is then defined as follows (STL formulas for SYNC and PAUSE are constructed analogously):

$$\texttt{NIBBLE} = (\texttt{F} \wedge \texttt{rise(low)})\ \texttt{before}_{[t_1,t_2]}\ (\texttt{L} \wedge \texttt{fall(low)})$$
$$\texttt{before}_{[t_3,t_4]}\ (\texttt{R} \wedge \texttt{rise(high)})\ \texttt{before}_{[t_5,t_6]}\ (\texttt{H}_{\text{nibble}} \wedge \texttt{fall(high)})$$

The top-level FRAME requirement captures precedence relation between SYNC, NIBBLEs, and the PAUSE. The monitor construction is compositional: a frame correctness is reported only when all the lower-level requirements for all the frame components (SYNC, NIBBLE, PAUSE) are met.

8.1.3 SENT requirements in TRE

For improved readability, we provide a reader with a quick reference to Timed Regular Expressions syntax, which is originally defined in Chapter 3. A timed regular expression ψ is defined

according to the following syntax [FMNU15]:

$$\psi := \epsilon \mid q \mid rise(p) \mid fall(p) \mid \psi_1 \cdot \psi_2 \mid \psi_1 \cup \psi_2 \mid \psi_1 \cap \psi_2 \mid \psi^* \mid \langle \psi \rangle_I$$

where q is of the form p, $\neg p$, $x \sim c$ or $\neg(x \sim c)$; I is a time interval $[a, b] \subseteq \mathbb{N}$.

Although it is possible to formulate TREs in an STL-like style and express the same behavior (e.g. the requirements $\texttt{F}^\dagger$ and $\texttt{R}^\dagger$ match falling and rising times), we use the syntax TRE to compose the requirements hierarchically. Hence, we obtain a concise and clear formalization for the requirements of interest. F and R regions (Eq. 9-10) are defined as follows:

$$\texttt{F} = \langle \texttt{mid} \rangle_{[0,T_{\texttt{fall}}]}; \qquad \texttt{F}^\dagger = \textbf{fall}(\texttt{high}) \cdot \langle \texttt{mid} \rangle_{[0,T_{\texttt{fall}}]} \cdot \textbf{rise}(\texttt{low}) \tag{9}$$

$$\texttt{R} = \langle \texttt{mid} \rangle_{[0,T_{\texttt{rise}}]}; \qquad \texttt{R}^\dagger = \textbf{fall}(\texttt{low}) \cdot \langle \texttt{mid} \rangle_{[0,T_{\texttt{rise}}]} \cdot \textbf{rise}(\texttt{high}) \tag{10}$$

The L TRE (Eq. 11) combines the requirements 3 and 5 from Tables 8.1, 8.2. The H TRE (Eq. 12) will match when the requirement 3 is fulfilled. The two are the necessary building blocks for checking the shape of pulses:

$$\texttt{L} = \langle \texttt{low} \rangle_{[T_{\texttt{stable}},5\texttt{tick}]} \tag{11}$$

$$\texttt{H} = \langle \texttt{high} \rangle_{[T_{\texttt{stable}},127]} \tag{12}$$

We are now able to define the TRE for synchronization, nibble, and pause pulses as a concatenation of regions, restricting the length of the pulses with appropriate time bounds. The SYNC TRE (Eq. 13) will match only when the requirements 1-6 (Tab. 8.1, 8.2) are met. The sensor signal will match the NIBBLE TRE (Eq. 14) if the requirements 1-3, 7-10 are fulfilled.

$$\texttt{SYNC} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{\{56\texttt{tick}\}} \tag{13}$$

$$\texttt{NIBBLE} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{[12\texttt{tick},27\texttt{tick}]} \tag{14}$$

$$\texttt{PAUSE} = \langle \texttt{F} \cdot \texttt{L} \cdot \texttt{R} \cdot \texttt{H} \rangle_{[12\texttt{tick},127\texttt{tick}]} \tag{15}$$

The frame and protocol requirements in TRE are formulated as follows:

$$\texttt{SENT\_FRAME} = \texttt{SYNC} \cdot \texttt{NIBBLE}^8 \cdot \texttt{PAUSE} \tag{16}$$

$$\texttt{SENT\_PROTOCOL} = (\texttt{SENT\_FRAME})^+ \tag{17}$$

8.2 Runtime Monitoring with Recovery

A runtime monitor typically partitions the execution traces in those that either satisfy or violate system's specification, possibly providing a quantitative metric of satisfaction (violation). However, for data-driven applications, such as serial protocols, test executions may last for hours and it

is required to continue monitoring even after detecting errors. Similarly to compilers, a monitor in such a case must be able to recover after observing a violation, collect the encountered errors, and report them to the user.

For a class of serial protocols, the asynchronous serial protocols (e.g. SENT [Int16a], RS-232 [Axe07], DMX512 [(R208], etc.), we propose a procedure to construct monitors with *error recovery*. To apply monitoring with recovery, the protocol must fulfill the following requirement: the devices communicate over a single line, where synchronization symbol, control and payload data, respectively, are multiplexed in time. As control signals are absent, the devices rely on the synchronization symbol to successfully capture the beginning of a useful portion of a frame.

By creating runtime monitors with recovery, we are able to: (i) Continue monitoring after detecting violations; (ii) Collect the errors and report them together with their violation type.

8.2.1 STL Monitors with Recovery

The STL monitors are transducers (temporal testers [PZ08]) by construction and are composed hierarchically to output the satisfaction signal of the top-level requirement. The sketch of construction procedure for monitoring with recovery is as follows: (i) we first formalize the `START`, `PAYLOAD`, and `STOP` patterns in STL; (ii) we then change the semantic meaning of STL assertions from (1) $\varphi \rightarrow \psi$ to (2) $\varphi \wedge \psi$: in the first formulation the transducer outputs '1' even if the requirement has never been checked, and '0' when the requirement has been violated (e.g. the F requirement from Sec. 8.1.2 is fulfilled even the line stays always at '1'); the second case the transducer manifests with the signal the precise time stamp when the requirement has been satisfied (i.e. outputting '1' when the correct falling edge occurred); (iii) for each requirement we identify a set of possible violations and assign an error code $\texttt{err}_i$ to each violation type. Each violation is guarded by an STL assertion $\varphi \wedge \neg\psi \wedge v_i$, where v_i identifies a violation type (e.g. $\texttt{mid}\ \mathcal{S}_{[T_{\text{fall}}+1,\infty)}\ \texttt{exit(high)}$ is a v_i clause to capture the violation of the type "too slow falling time" for the STL assertion F from Sec. 8.1.2).

Finally we check the temporal precedence of the `START`, n `PAYLOAD` sequences and the `STOP` pattern with the $\texttt{before}_{[t_1,t_2]}$ macro defined in Sec. 8.1.2. Using temporal testers allows to monitor all the requirements in parallel, and extending with violation clauses v_i provides the necessary debugging information.

8.2.2 TRE Monitors with Recovery

In the case of asynchronous serial protocols, the devices communicate with sequences that form certain patterns over time; the communication is cyclic, where the data is split in subsequently following frames. These protocols admit a natural formalization in TREs: A frame begins with a unique synchronization pattern (`START`), followed by n `PAYLOAD` patterns, and ends with a `STOP` pattern. The asynchronous serial protocol is then defined as a sequence of frames:

$$\texttt{ASYNC\_SERIAL\_PROTOCOL} = \texttt{FRAME}^{+}, \tag{18}$$

$$\texttt{FRAME} = \texttt{START} \cdot \texttt{PAYLOAD}^{n} \cdot \texttt{STOP}. \tag{19}$$

The above expression exactly generalizes the TRE formalization of the SENT protocol from Section 8.1.1. It is important to mention that the Kleene star (*) operator should not be used in the specification of START, PAYLOAD and STOP in TREs, as these patterns are finite sequences of symbols; we use the Kleene star operator only at the top TRE (i.e. Eq. 18).

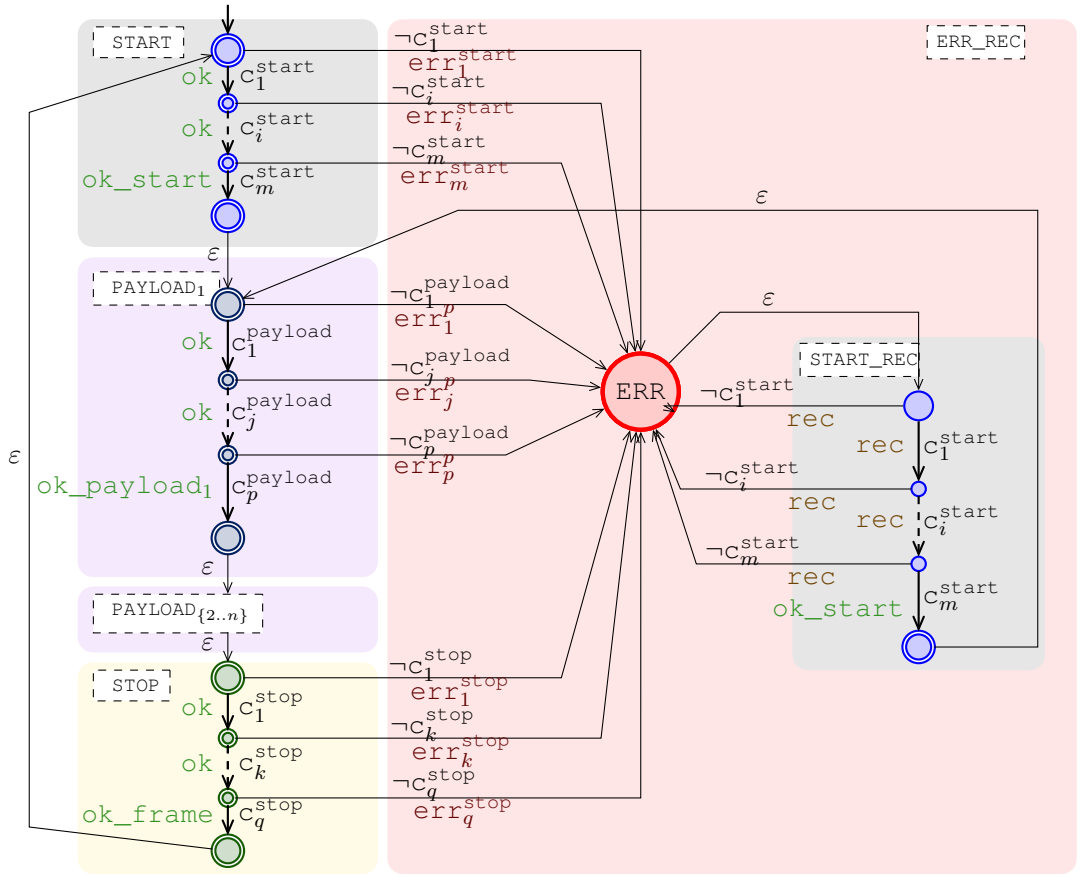


Figure 8.3: General procedure for obtaining monitors with recovery[1]

The sketch of construction procedure for a monitor with recovery is shown in Fig. 8.3. For each of the START, PAYLOAD, and STOP patterns, we construct the corresponding automata with discrete-time clocks $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$, respectively. We also create an additional copy of $\mathcal{A}_{\texttt{start}}$, called $\mathcal{A}_{\texttt{rec}}$, which enables the runtime monitor to recover from an error. In this work we take an optimistic approach, and use a weak interpretation of regular expression over finite traces. In case when a trace ends and only a prefix of the regular expression is matched, we decide to accept the input sequence. Therefore all the states in $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$ are accepting. The automaton-construction procedure from a given TRE, is adopted from [ACM02a] to the discrete interpretation of time. The state transitions are protected by a set C of symbolic transition guards C, where $C = \{c_1^{\texttt{start}}, \ldots, c_m^{\texttt{start}}, c_1^{\texttt{payload}}, \ldots, c_p^{\texttt{payload}}, c_1^{\texttt{stop}}, \ldots, c_q^{\texttt{stop}}\}$.

For each $c_i \in C$ we associate a complementary transition $\neg c_i$ to the global error state. The error state silently transitions to the starting state of the recovery automaton $\mathcal{A}_{\texttt{rec}}$ which consumes garbage symbols until a correct synchronization symbol is observed. The correct `START` pattern is a necessary pre-requisite for a monitor to analyze subsequent frames, and for the decoder to analyze the transferred data: as long as the synchronization symbol of the next frame is not received, the recovery automaton $\mathcal{A}_{\texttt{rec}}$ goes back to the error state.

We introduce a diagnostic variable `out`, defined over a finite set of symbolic values: $\{$`ok`, `ok_start`, $\texttt{ok\_payload}_{1,\ldots,N}$, `ok_frame`, `rec`, $\texttt{err}_{1,\ldots,m}\}$. The values have the following meaning: `ok`: the trace has been correct so far; `ok_start`: the starting synchronization symbol has been matched; $\texttt{ok\_payload}_i$: the i^{th} payload symbol has been matched; `ok_frame`: the frame has met all the requirements; `rec`: the monitor is in the recovery state; $\texttt{err}_i$: the specification is violated by an error of type i.

We then transform $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, $\mathcal{A}_{\texttt{stop}}$ and $\mathcal{A}_{\texttt{rec}}$ to transducers $\mathcal{A}'_{\texttt{start}}$, $\mathcal{A}'_{\texttt{payload}}$, $\mathcal{A}'_{\texttt{stop}}$ and $\mathcal{A}'_{\texttt{rec}}$ as follows: (i) For each transition in $\mathcal{A}_{\texttt{i}}$, we output `ok` value; (ii) For each transition leading to a sink state, we output appropriate `ok_{start|payload|frame}` value; (iii) For each transition guarded by $\neg c_i$ we output $\texttt{err}_i$; (iv) For each recovery automaton transition, except the synchronization symbol matching transition, we associate `rec` value. The transition in $\mathcal{A}'_{\texttt{rec}}$ which matches synchronization symbol outputs `ok_start` (see Fig. 8.3). For the top-level expression `FRAME`, we create the automaton $\mathcal{A}_{\texttt{frame}}$ by concatenating the $\mathcal{A}_{\texttt{start}}$, $\mathcal{A}_{\texttt{payload}}$, and $\mathcal{A}_{\texttt{stop}}$ with ε transitions. This way the user is capable to receive the information about the number of frames that meet the specification, as well as errors and their type.

8.3 Runtime Monitoring of the SENT protocol

This section describes building runtime monitors in FPGA and evaluating the results in industrial environment. A general overview of the framework is followed by implementation and evaluation details.

8.3.1 From Requirements to Hardware Monitors

In this section we summarize the process of creating runtime monitors; the proposed framework is not limited to the SENT, and can be applied for other protocols as well.

Requirements Formalization. The initial step for creating runtime monitors is to obtain formal representation of the system requirements. Formal semantics allows to eliminate ambiguities in interpretations and precisely define what is to be monitored. In order to evaluate the power of different formalisms, and to eliminate "single source of truth" from the system we use Signal Temporal Logic (STL) and Timed Regular Expressions (TRE) as specification languages. This phase results in a set of formulae (STL & TRE) which describe natural-language requirements.

[1] For clarity of the presentation, we keep ε-transitions in the Figure 8.3; these transitions are removed in implementation though keeping the monitor deterministic.

For STL requirements we admit an automated pre-processing step to obtain formulae that allow efficient hardware realization: on the parse tree of the formula we (i) eliminate duplicate sub-trees (Simplification); (ii) apply a recursive procedure from [MNP07] to convert bounded future STL temporal operators to an equisatisfiable past operators, resulting in a causal formula with the past temporal operators only (Pastification). The second step is achieved by (i) calculating the temporal depth $\mathcal{D}$ of the formula; (ii) re-writing a formula with past operators which results in postponing a monitoring verdict by $\mathcal{D}$.

Offline Evaluation. In this phase we evaluate monitors offline on short trace fragments, previously recorded from an oscilloscope or an ADC via the Chipscope [Inc16] in order to speed-up debugging and identify implementation bottlenecks.

The monitors for STL formulas are built compositionally from the formula parse tree [PZ08]. With each node of the STL parse tree, which represents either a temporal or a logical operator, we associate a transducer $\mathcal{T}$ which takes as inputs satisfaction signals of its child nodes and outputs the satisfaction signal for the corresponding operator. The satisfaction signal of the root node produces output of the monitor. *Behavioral STL Lib SystemC* is a SystemC implementation of STL transducers, which are used to obtain a monitor. We use SystemC simulation kernel to run the monitor on the pre-recorded traces.

The runtime monitors for the TRE requirements are also implemented in hierarchical fashion: the $\mathcal{A}'_{\texttt{sync}}$, $\mathcal{A}'_{\texttt{nibble}}$, and $\mathcal{A}'_{\texttt{stop}}$ transducers are combined in the top-level recovery automaton described in Section 8.2.2. We use Vivado Behavioral Simulation to evaluate VHDL code of the top-level $\mathcal{A}'_{\texttt{frame}}$ transducer.

Runtime Monitoring in FPGA is the final phase; the monitors are synthesized in a digital reconfigurable hardware and evaluated in the lab environment. After the off-line phase we obtain the validated monitors for STL and TRE, which follow different paths of hardware implementation.

In case of STL monitoring, we use High-Level Synthesis (HLS) [Inc] to generate Hardware Description Language (HDL) code for monitors written in SystemC. During the HLS step, the SystemC monitors are transformed to an equivalent synthesizable VHDL or Verilog. We use an alternative implementation of transducers (*Synthesizable STL Lib SystemC*), which is suitable for HDL code generation. *Behavioral* and *Synthesizable* implementations are functionally equivalent, but HLS imposes constraints on the SystemC code to be hardware-synthesizeable. Keeping *behavioral* and *synthesizable* versions allows quick prototyping using all `C++` features and then produce a hardware-optimized *synthesized* version.

Since transducers $\mathcal{A}'_{\texttt{sync}}$, $\mathcal{A}'_{\texttt{nibble}}$, $\mathcal{A}'_{\texttt{stop}}$, and $\mathcal{A}'_{\texttt{frame}}$ in the TRE approach are implemented in VHDL, we directly use Vivado Synthesis, Logic & Power Optimization, Place & Route tools to obtain a bitstream for FPGA programming.

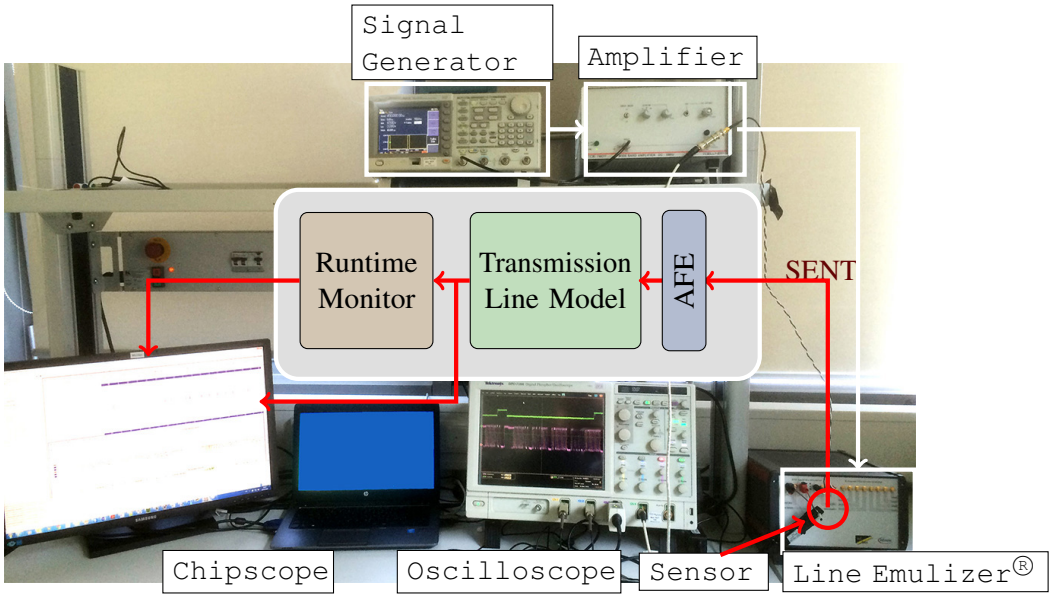


Figure 8.4: Runtime Monitoring of the SENT: Hardware Setup

8.3.2 FPGA Implementation

We implemented runtime monitors for the SENT protocol in Xilinx Virtex 7 FPGA. The monitors are embedded in the *Line Emulizer* hardware (Fig. 8.4). It combines an Analog Front-End (AFE) capable to interface various sensors with a high-performance Virtex 7 FPGA. This hardware also models a transmission line with adjustable parameters between a sensor and an ECU.

The signal from the SENT sensor (see Fig. 8.4) comes to the *Line Emulizer*, where it is passed through the AFE and sampled with a high-speed ADC, which results in its finite value representation. During operation in a car, a sensor and an ECU are placed in different locations, hence the sensor signal is affected by a transmission line. To take into account the effects of physical wires, the sensor signal is passed through a digital model of a transmission line. We attach the STL and TRE runtime monitors at the end of the transmission line model (see Fig. 8.4), to be able to report specification conformance at the receiver side, which is important for proper signal decoding.

The STL and TRE monitors observe at 70 MHz the sensor signal affected by the physical line, calculate verdicts at every clock cycle and output the result to the user via the Chipscope (Fig. 8.4). We performed experiments with different models of the line, and conclude that the appropriate line parameters are critical for ensuring the specification compliance. The sensor signal passed through a line with a higher capacitance violates the specification, since the falling and rising times are not met, which is directly observed from the monitor.

Table 8.3 reports the estimated FPGA resources (flip-flops, FF & look-up tables, LUT), and the estimated maximum clock period of the runtime monitors. For each HLS-generated monitor we also present its generation time and peak memory usage during HDL-code generation.

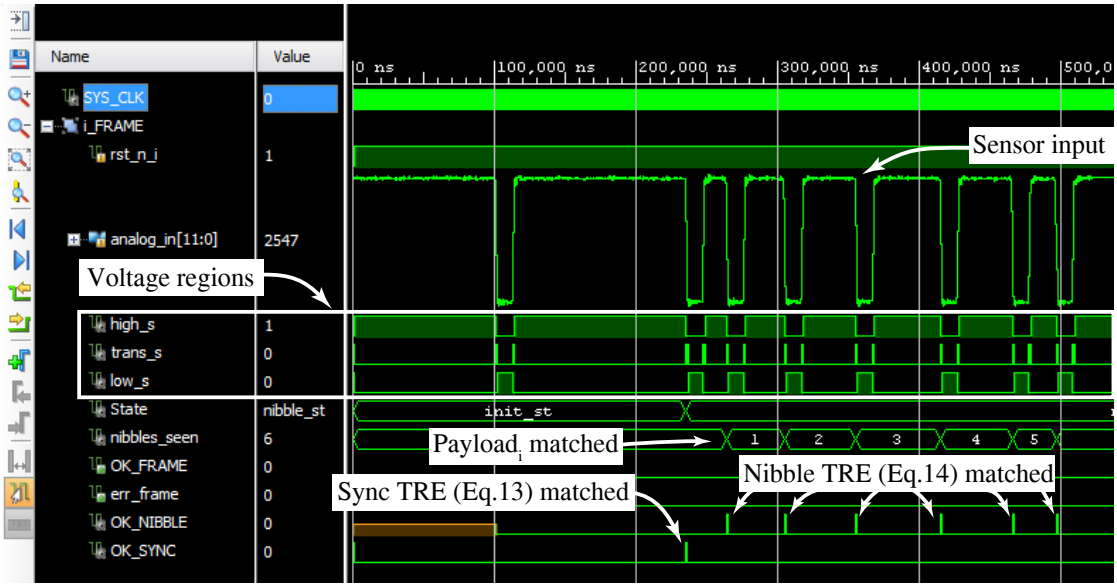


Figure 8.5: Runtime TRE monitoring: Vivado RTL functional simulation

The monitors in High-Level Synthesis are constructed in a hierarchical fashion, hence the FRAME monitor (see Tab. 8.3) subsumes monitors for other requirements and results the highest hardware footprint. The last row of the Tab. 8.3 reports the total hardware resources consumed by the top-level TRE monitor: the direct hardware implementation results in an order of magnitude lower footprint.

Fig. 8.5 shows a result of offline evaluation for TRE requirements. The original SENT signal is observed by the monitor, which outputs OK_NIBBLE, OK_SYNC and the corresponding ERR signals. The figure depicts a nominal case, where all the requirements are met.

Runtime monitoring of the SENT signal against STL requirements is shown in the Fig. 8.6. For this test case the optional pause pulse was deactivated, hence the correct frame is manifested after observing eight correct nibbles (signals OK_NIBBLE, OK_SYNC, OK_FRAME). The OK_NIBBLE signal is asserted when the corresponding precedence between the requirements F, L, R, and H is met. The output of the monitors F, L, R, and H, and the corresponding sub-formulas are presented in the lower part of the Fig. 8.6.

8.4 Monitoring SENT with WED

In order to demonstrate robustness monitoring of real industrial devices using our hardware monitors, we monitor selected SENT requirements using WED-based robustness.

Requirement		#FF	#LUT	Clock	HLS: Time	HLS: Memory
F	HLS	61	118	5.81ns	114.203s	225MB
L		53	85	4.24ns	96.490s	159MB
R		61	113	5.81ns	109.784s	224MB
H_{nibble}		125	249	5.81ns	175.716s	225MB
H_{sync}		28	407	5.81ns	253.507s	224MB
H_{pause}		73	98	4.24ns	162.637s	212MB
NIBBLE		435	1123	7.7ns	394.671s	611MB
SYNC		207	1062	7.7ns	723.690s	605MB
PAUSE		217	710	7.7ns	206.767s	317MB
FRAME		1198	4322	7.7ns	1675.52s	1.39GB
FRAME	TRE	68	350	4.5ns	-	-

Table 8.3: Monitor Synthesis from STL and TRE formalizations. We provide FPGA synthesis results: flip-flops (#FF) and lookup tables (#LUT) consumed and minimum clock frequency achieved. In addition, we provide time and memory consumed by HLS procedure.

8.4.1 Formalized Requirements

In order to communicate without errors, SENT transmitter must comply to timing and electrical requirements specified by the standard. We focus on monitoring timing requirements of the rising and falling edges of a pulse. If these timing constraints are not met, it is not guaranteed that the controller will be able to decode the data from the pulse.

The timing requirements of interest can be stated in natural language as follows: the fall/rise time from V_1 to V_2 must be no longer than T_{fall}/T_{rise} μs. Before applying our approach we formalize the requirements with following STL formulas:

$$\begin{aligned} \varphi_{10} &= \square(\downarrow \mathit{high} \rightarrow \mathit{mid}\, \mathcal{U}_{[0,T_{fall}]} \uparrow \mathit{low}) \\ \varphi_{11} &= \square(\downarrow \mathit{low} \rightarrow \mathit{mid}\, \mathcal{U}_{[0,T_{rise}]} \uparrow \mathit{high}) \end{aligned}$$

where $rise$ and $fall$ operators are defined by $\uparrow p = \ominus(\neg p) \wedge p$ and $\downarrow p = \ominus(p) \wedge \neg p$, respectively. The SENT standard allows the following values: $T_{fall} \leq 6.5\mu s$ and $T_{rise} \leq 18\mu s$. Voltage levels are also specified in the standard, however in our experiments they are scaled to the Analog-to-Digital converter output range.

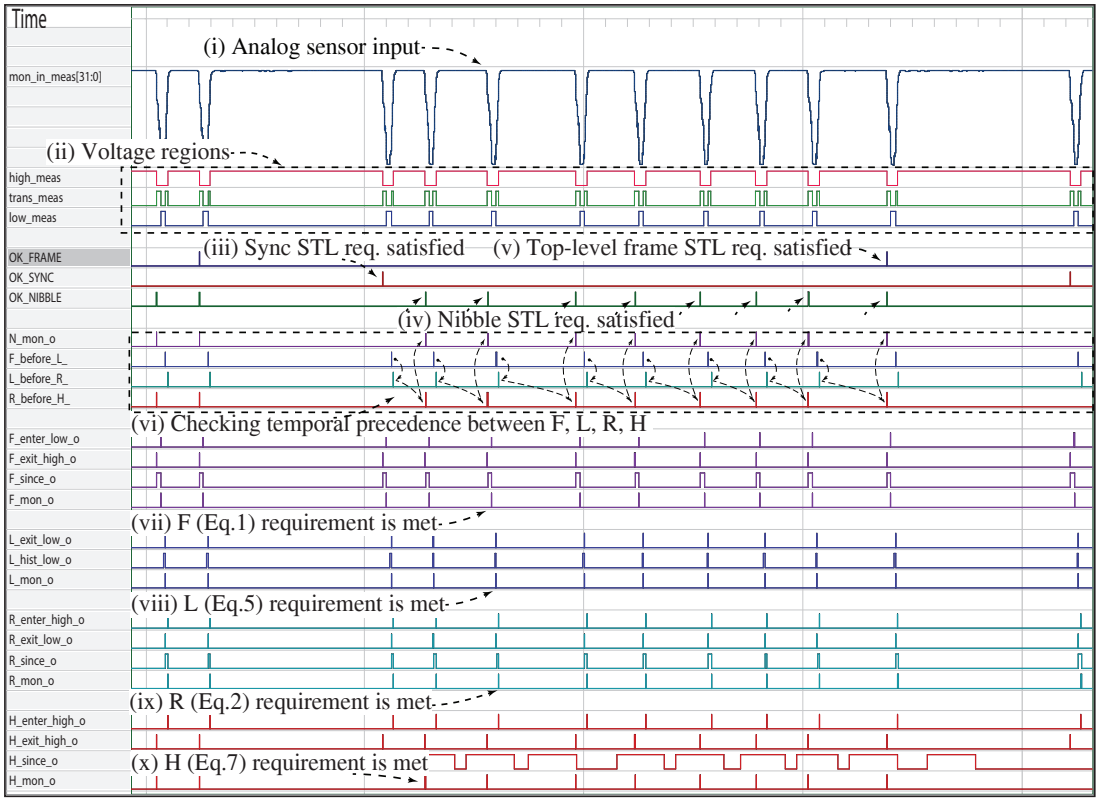


Figure 8.6: Runtime monitoring of the STL requirements

8.4.2 Evaluation Results

In order to test the monitors with realistic data, we recorded output from a real magnetic sensor which implements the SENT protocol. We used the Hall-effect sensor with SENT interface from Infineon Technologies. The Hall-effect cell in this sensor measures the magnetic flux. Such information can be used for linear and angular position sensing. In the automotive domain, this sensor is used to sense steering torque and pedal and throttle position.

According to the SENT standard, devices are configured prior to operation. Therefore, we are allowed to assume that the configuration of SENT frame is static and its structure cannot change during runtime.

In Fig. 8.7 we can see the first SENT requirement monitored on a trace which represents a correct SENT pulse falling edge. For this pulse we compute both positive and negative robustness degree. In the beginning of the trace, the left hand side of the implication is not satisfied, therefore the entire formula is trivially satisfied and the negative robustness is zero. In contrast, the positive robustness is equal to the WED cost of creating a violating trace - which can be done simply by substituting *high* sample with *mid*, thus making $\downarrow$ *high* condition true and the entire formula

false. We note that the positive robustness decreases in the course of the execution - this happens because the robustness algorithm dynamically discovers a cheaper way to transform the trace into a violating one.

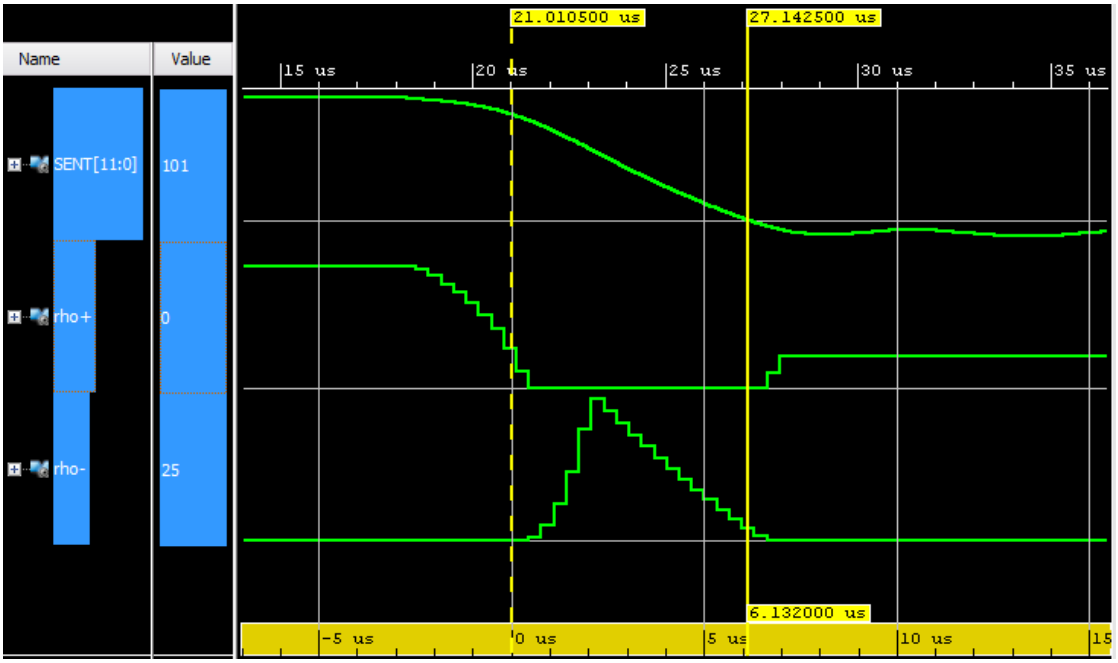


Figure 8.7: Calculated positive and negative robustness for SENT pulse falling edge which satisfies T_{fall} requirement.

We now analyze these results in more detail. At the moment when the left hand side of the implication becomes satisfied (dashed yellow marker in 8.7, the right hand side of the formula is not yet satisfied. This results in an increase of the negative robustness that comes from the accumulated WED substitution costs needed to disarm the $\downarrow$ *high* condition. After observing a sufficient number of trace samples, the robustness algorithm realizes that it is cheaper to perform substitutions at the low value end of the falling edge in order to make the right hand side of the formula hold. As a consequence, the negative robustness starts also decreasing. Finally, the monitor starts observing the samples that satisfy the right hand side of the implication, thus also satisfying the entire formula. This results in the negative robustness dropping to zero, but also in an increase of the positive robustness (see the trace segment after the yellow mark in Fig. 8.7). The small positive robustness degree conveys two important messages: (1) the observed execution satisfies the timing requirements; and (2) a small change of the trace could violate the requirement.

In Fig. 8.8 we can see the rising edge timing requirement monitored on a trace which represents a violating SENT pulse. The violating pulse was artificially created from a correct trace which was recorded from the actual sensor. The violation was created by replaying the correct recorded values at a slower speed, which prolonged the rising edge length.

In this case the evolution of positive and negative robustness degree over time is converse to the previous case. The obvious difference is that the final value of the positive robustness is zero and the negative robustness degree is non-negative. This is valid result because the trace is violating the rising edge timing requirement $T_{rise} \leq 18\ \mu s$. The negative robustness is larger than the positive robustness degree of the previous example in Fig. 8.7, due to the larger cost of compensating for timing violation of the rising edge.

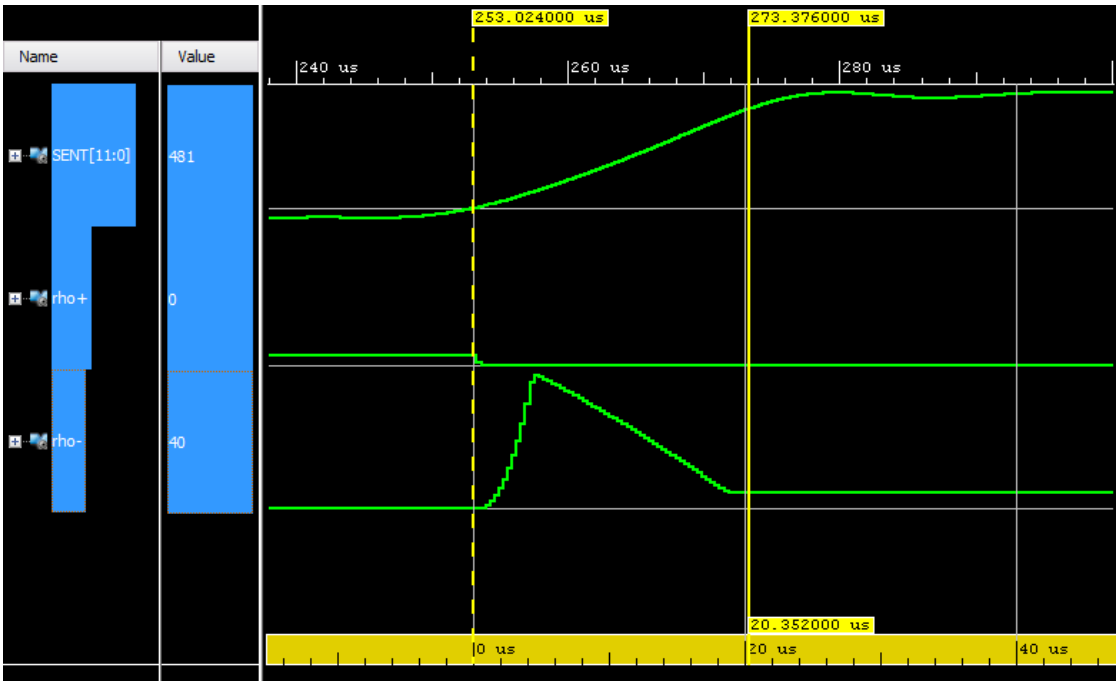


Figure 8.8: Calculated positive and negative robustness for SENT pulse rising edge which violates T_{rise} requirement.

In Table 8.4 we report on FPGA resource consumption of the generated monitors. A flip-flop (FF) represents a memory element which is implementing a sequential circuit, in this case it is used to implement an automaton state. A lookup table (LUT) implements a function which returns a *minimum* element. These functions are used in the dynamic programming algorithm for WED.

The linear dependency between increase in number of transitions of automaton and amount of LUTs used can be observed in Table 8.4. Same conclusion can be drawn for the number of states and the number of FFs consumed.

	SAT		$\mathcal{W}_\varphi$				$\mathcal{W}_{\neg\varphi}$			
	trace	ρ	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT	$\lvert Q\rvert$	$\lvert\Delta\rvert$	#FF	#LUT
φ_{10}	YES	11	208	627	3272	50046	498	1945	7745	148852
φ_{11}	NO	-41	558	1677	8865	136321	1338	5224	21191	405604

Table 8.4: Evaluation results for SENT protocol properties. We report on the resulting robustness degree ρ, the number of states $|Q|$ and transitions $|\Delta|$ of Weighted Edit Automata $\mathcal{W}_\varphi$ and $\mathcal{W}_{\neg\varphi}$, as well as the number of flip-flops (#FF) and lookup tables (#LUT) consumed to implement the monitors.

CHAPTER 9

Conclusion and Future Work

9.1 Summary

In this thesis, we have introduced theoretical concepts and practical solutions for obtaining correctness and robustness real-time monitors for specifications expressed in STL. We provide a general, unified approach to calculating correctness and robustness with Algebraic Runtime Verification. To perform spatio-temporal robustness analysis, we introduce robustness monitors based on Weighted Edit Distance, and implement them in hardware. We evaluate our monitors on a real-scale case study of a protocol used for automotive sensor applications. To improve usability of monitors in an industrial setting, we define monitors with recovery.

We proposed an efficient procedure for real-time monitoring of CPS with STL correctness monitors implemented on FPGAs. The introduced theoretical novelties, such as temporal testers with discrete clocks for signals of bounded variability, allowed us to minimize the size of our monitors. We provide RTL implementation of these testers in Verilog HDL. We have implemented our monitors in a hardware development board, and demonstrated the benefits for real-time monitoring of Cyber-Physical Systems.

We presented Algebraic Runtime Verification, our generic framework for monitoring both correctness and robustness of CPS. Simply by instantiating different semirings, ARV allows users to evaluate different types of robustness degree. We demonstrated the flexibility of ARV with respect to specification languages and semantics. Our work showed that automata-based ARV enables precise robustness measurement of a trace w.r.t. a specification, compared to syntactic-based approaches.

In order to capture spatio-temporal robustness in a meaningful way, we defined a novel procedure for evaluating robustness degree of STL properties based on weighted edit distance. The distance is cumulative by definition, which allows robustness degree to be sensitive on the number of violations of the formula. Although we focus on the quantitative semantics of STL, the weighted

edit distance can be applied to other specification languages over finite signals, in the spirit of Algebraic Runtime Verification.

Our FPGA implementation of WED-based robustness monitors allows quantifying the distance to the violation of safety requirements in real-time on emulated hardware or real silicon. We have successfully demonstrated our approach to check relevant safety properties in the automotive domain, by monitoring the behavior of a car engine through observation of essential signals such as air-to-fuel ratio. Furthermore, we show that WED is also suitable for verification of well-established industrial standards such as SENT protocol.

The industrial case study focuses on assessing STL and TRE for formalizing requirements of the SENT protocol and obtaining hardware monitors for these requirements. We evaluate these specification languages in terms of expressivness w.r.t. SENT protocol requirements, consumption of hardware resources, and monitor reuse.

Hardware resource consumption for the case of SENT protocol shows that (i) both approaches can be easily mapped to state-of-the-art FPGAs, (ii) STL-based monitors consume an order of magnitude more resources than the TRE monitors. Implementing STL hardware monitors in SystemC makes them applicable not only for low-level monitoring of hardware, but also for monitoring of models written in SystemC. However, such monitors require an intermediate transformation using High-Level Synthesis which results in larger hardware footprint. TRE specifications of serial protocols can be directly translated to automata with recovery which admit efficient hardware realization.

9.2 Research Motivation Revisited

In this section we briefly reiterate through research problems which motivated our work and explain how contributions of the thesis address those problems.

Emerging Verification Challenges Our aim was to offer a solution for new problems in verification, which may emerge due to the change in verification paradigm. A perfect example where we would have to shift from exhaustive verification to real-time monitoring are autonomous vehicles with data-driven controllers.

Because of their speed, responsiveness, versatility and black-box approach to the SUT, we promote real-time monitors for correctness and robustness as a natural solution to new verification challenges. Correctness real-time monitors in hardware are presented in Chapter 5. For soft constraints, we propose deploying robustness monitors which were presented in Chapters 6 and 7. The basis of our correctness and robustness monitors are automata and temporal testers, which are introduced in Chapter 4.

Hardware-accelerated Verification The growing trend in the industrial verification practice is to offload simulation engines by adopting emulation-based or hardware-in-the-loop approach. However, such methods lack a systematic, rigorous and automatic approach.

As a solution to this problem we see correctness monitors implemented in FPGAs, described in Chapter 5. Due to their compositional architecture based on temporal testers, these correctness monitors are able to produce verdicts in real-time, running at the same speed as the system under test. We propose an automatic procedure to translate requirements specified as STL assertions, into hardware-ready implementation in Verilog HDL.

An Abundance of Quantitative Metrics Quantitative monitoring offers a plethora of different quantitative metrics, each developed for a specific purpose and each defining a specific evaluation algorithm. This situation makes it harder to identify which metric is appropriate to a specific kind of problem. Some of the mainstream algorithms which provide quantitative verdicts use syntactic-based approach, which does solve the problem of quantitative monitoring at a reduced price. We have shown in Chapter 6 that such approach *imprecise*, in general case.

We offer a unified and general approach ARV, which allows switching between different distance metrics simply by replacing a set of operations in the algorithm. Monitoring algorithm clearly separates concerns between specification representation and quantitative verdict calculation. This way we clarify the problem of quantitative monitoring and unify different semantics within a single framework. We presented Algebraic Runtime Verification in Chapter 6, where we also demonstrate precision compared to syntactic-based approach.

Beyond Runtime Verification Runtime correctness monitoring provides qualitative verdicts, which are not sufficient for real-valued behaviours characteristic for CPS. *Robustness analysis* is a quantitative method which provides a notion of distance of a trace from a set of desired behaviors defined by a specification.

A method which solves the problem of obtaining specification from set of traces, which is especially useful for requirement engineering of data-driven systems or legacy systems lacking specification, is *specification mining*.

Finally, for the systems with complex behavior simulation it is neither possible to achieve reasonable verification coverage nor exercise longer scenarios. In such cases *falsification testing*, which is iteratively optimizing systems' parameters in order to falsify the requirement, is an approach that can be used as a powerful testing tool.

All the mentioned approaches rely on the notion of meaningful and precise *robustness degree*. We investigate general notion of robustness in Chapter 6 and define and evaluate spatio-temporal robustness degree based on Weighted Edit Distance in Chapter 7.

Fostering RV Methods in the Industry We highly prioritize the applicability of our monitors, which use lightweight formal methods, on problems from the industrial context. For this reason we develop procedures which promote automation of RV. We conduct a large industrial case study, presented in Chapter 8. In Chapter 7, we also evaluate WED-based robustness on Simulink models taken from automotive domain.

In Chapters 5 and 7 we implement our monitors on FPGAs, which are frequently used in the industry for prototyping mixed-signal designs and implementing HiL testbenches. Furthermore,

we equip our monitors with a generic recovery procedure in Chapter 8. This is particularly valuable to practicioners, since the monitors can recover and continue monitoring long streams of data, typically found in asynchronous serial protocols. Our monitors can also classify errors, which directly allows logging of violations. We also show in Chapter 8 how our monitors can be reused in different abstraction levels (high-level models, functional simulation or pure silicon) and with different specification languages: STL and TRE.

9.3 Future Work

We believe that the results presented in this thesis may open several research and development avenues. At the moment, the Algebraic Runtime Verification framework adopts an enumerative approach to time. We plan to investigate the effect of symbolic representation of time to our automata-based approach ARV. We also plan to leverage different kinds of robustness degrees from our ARV framework, to improve the efficiency of falsification testing approach.

The precision of our semantic ARV approach comes at a price – an exponential blow up in the size of the specification, but also in the largest constant appearing in the formula. So far, we studied the worst-case complexity of our translation. However, we believe that for many applications the effective translation will yield monitors that are much smaller from the worst case. This requires a comprehensive experimental study with an optimized implementation. To achieve this goal, we will explore different optimization strategies, including the use of the algorithms implemented in the symbolic automata library[1] such as its minimization procedure [DV14]. We will implement several code generators (interpreted Java, Simulink S-functions, embedded C, FPGA, etc.) and investigate the reuse of specifications and monitors across stages in the development cycle.

In this thesis, we restricted ourselves to Hausdorff-like measures and additive idempotent semirings. We would like to study in the future the extension of our framework to non-idempotent $\oplus$-addition and its application to RV: for instance, to a probabilistic semiring. We would also like to investigate whether we can use our approach to support other types of semantics.

Treating the value domain symbolically is natural and we exploit this fact in our work on WED-based robustness. On the other hand, combining this quantitative semantics with symbolic time is not straightforward. In the qualitative case, representing the time symbolically can be done because there is a certain equivalence between states that have the same discrete location and different clock valuations, and such states can be grouped together. In the quantitative setting, this is not the case – two states with the same discrete location and different clock valuations will in general have different values and hence cannot be grouped together. Such a symbolic representation of quantitative states might be possible if some accuracy can be dropped. We will consider extending our WED robustness algorithm to automata with discrete clocks.

We see an attractive research direction in integrating our monitors in *autonomous vehicles*. In this setting, our monitors would leverage their real-time nature, rigor and unobtrusiveness to provide an additional layer of safety. Furthermore, we believe that deploying machine learning techniques would allow our monitors to gain predictive ability and foster them in system health

[1] `https://github.com/lorisdanto/symbolicautomata`

and fail-aware applications. Such monitors could be used to track the evolution of critical system parameters, and provide alerts on possible violations even before they occur. Further applications of real-time correctness and robustness monitors could be found in the medical domain, following the work presented in [BBS14a, BBS$^+$14b].

APPENDIX A

Theorem Proofs

A.1 Theorem Proofs: Algebraic Runtime Verification

Lemma 3. *Let $\mathbb{S} = (S, \oplus, \otimes, e_\oplus, e_\otimes)$ be an additively idempotent, negative and monotonic semiring. Then, for all $a \in S$, $e_\otimes \sqsubseteq a \sqsubseteq e_\oplus$.*

Proof. Consider an arbitrary $a \in S$. By Definition 11 and assumption that S is negative, we have that $e_\otimes \sqsubseteq e_\oplus$. By Definition 11 and assumption that S is monotonic, we have that $e_\otimes \otimes a \sqsubseteq e_\oplus \otimes a$ and $e_\otimes \oplus a \sqsubseteq e_\oplus \oplus a$. By Definition 9, $e_\otimes \sqsubseteq a$ and $a \sqsubseteq e_\oplus$, which concludes the proof. □

Theorem 1. *Given a predicate ψ in DNF, a valuation v and the distance $d(v, \psi)$ defined over a bounded semiring S, we have that $\mathsf{vpd}(v, \psi) = d(v, \psi)$ if: (1) S is idempotent; or (2) ψ is in $\wedge$-minimal DNF.*

Proof. We prove the theorem by induction on the structure of the predicate. If ψ is not satisfiable, we have by by Definition 12 and Algorithm 1that $d(v, \psi) = \mathsf{vpd}(v, \psi) = e_\oplus$. Hence, from now on, we consider only satisfiable ψ. We first prove the theorem for an arbitrary term ψ^c, and then prove it for general DNF formulas ψ. We start with the proof for ψ^c.

Base cases: We have 3 base cases to consider - $\psi^l \equiv \top$, $\psi^l \equiv y \preceq k$ and $\psi^l \equiv y \succeq k$.

- *Case $\psi^l \equiv \top$:*

$$
\begin{array}{rcll}
\mathsf{vpd}(v, \top) & = & e_\otimes & \text{by Algorithm 1} \\
d(v, \top) & = & \oplus_{v' \models \top} \otimes_{x \in X} d(v(x), v'(x)) & \text{by Definition 17.} \\
& = & \otimes_{x \in X} d(v(x), v(x)) \oplus \oplus_{v' \neq v} \otimes_{x \in X} d(v(x), v'(x)) & \text{by semantics of } \top \\
& = & e_\otimes \oplus \oplus_{v' \neq v} \otimes_{x \in X} d(v(x), v'(x)) & \text{by Definition 12} \\
& = & e_\otimes & \text{by Definition 9.}
\end{array}
$$

- *Case* $\psi^l \equiv y \preceq k$: We consider two sub-cases, $v(y) \preceq k$ and $v(y) \succ k$.
  - *Case* $v(y) \preceq k$:

$$\begin{aligned}
\mathsf{vpd}(v, y \preceq k) &= e_\otimes && \text{by Algorithm 1}\\
d(v, y \preceq k) &= \oplus_{v' \models y \preceq k} \otimes_{x \in X} d(v(x), v'(x)) && \text{by Definition 17}\\
&= \otimes_{x \in X} d(v(x), v(x)) \oplus \\
&\quad \oplus_{v' \models y \preceq k, v' \neq v} \otimes_{x \in X} d(v(x), v'(x)) && \text{by associativity of } \oplus\\
&= e_\otimes \\
&\quad \oplus_{v' \models y \preceq k, v' \neq v''} \otimes_{x \in X} d(v(x), v'(x)) && \text{by Definition 12 } \oplus\\
&= e_\otimes && \text{by boundedness of } S.
\end{aligned}$$

  - *Case* $v(y) \succ k$: Consider arbitrary $a \preceq k$ and $b \succ k$, and arbitrary valuations v such that $v(y) = b$ and v' such that $v'(y) = a$ and for all $x \in X \backslash \{y\}$, $v'(x) = v(x)$. We have that

$$\begin{aligned}
d(v, v') &= \otimes_{x \in X} d(v(x), v'(x)) && \text{by Definition 17}\\
&= d(v(y), v'(y)) \otimes \otimes_{x \in X \backslash \{y\}} d(v(x), v'(x)) \\
&= d(b, a) \otimes \otimes_{x \in X \backslash \{y\}} d(v(x), v'(x)) && \text{by assumption}\\
&= d(b, a) \otimes e_\otimes && \text{by Definition 12.}
\end{aligned}$$

    Consider an arbitrary valuation valuation v'' and variable $z \in X \backslash \{, x, y\}$ such that $v''(x) = v'(x)$ for all $x \in X \backslash \{z\}$. We have that:

$$\begin{aligned}
e_\otimes &\sqsubseteq d(v(z), v''(z)) && \text{by Lemma 3}\\
d(v, v') &= \otimes_{x \in X} d(v(x), v'(x)) && \text{by Definition 17}\\
&= d(v(y), v'(y)) \otimes d(v(z), v''(z)) \otimes_{x \in X \backslash \{y,z\}} d(v(x), v'(x)) \\
&= d(b, a) \otimes d(v(z), v''(z)) \otimes \otimes_{x \in X \backslash \{y,z\}} d(v(x), v'(x)) && \text{by assumption}\\
&= d(b, a) \otimes d(v(z), v''(z)) && \text{by Definition 12.}
\end{aligned}$$

    Combining the facts that $e_\otimes \sqsubseteq d(v(z), v''(z))$ and $d(b, a) \otimes e_\otimes \sqsubseteq d(b, a) \otimes d(v(z), v''(z))$ (Definition 11 and Lemma 2), we conclude that $d(v, v') \sqsubseteq d(v, v'')$. By Definition 10, it follows that $d(v, v') \oplus d(v, v'') = d(v, v')$, and hence that $\oplus_{v' \models y = a \text{s.t. } v'(y) = b} d(v, v') = d(b, a) = d(v'(y), a)$.
    Consider two arbitrary $a \preceq k$ and $a' \preceq k$ such that $a < a'$. We have that $\oplus_{v' \models y = a \text{s.t. } v'(y) = b} d(v, v') = d(v'(y), a)$ and $\oplus_{v' \models y = a' \text{s.t. } v'(y) = b} d(v, v') = d(v'(y), a')$.

Following the fact that $d(v'(y), a') \sqsubseteq d(v'(y), a)$, we conclude that $d(v, y \preceq k) = d(v(y), k)$. By Algorithm 1, we have that $\mathsf{vpd}(v, y \preceq k) = d(v(y), k)$, hence $\mathsf{vpd}(v, y \preceq k) = d(v, y \preceq k)$.

- *Case* $\psi^l \equiv \neg(x \preceq k)$: Symmetric to $\psi^l \equiv x \preceq k$.

Inductive hypothesis: $\psi^c \equiv \psi^c_1 \wedge \psi^l$.

We have 3 base cases to consider - $\psi^l \equiv \top$, $\psi^l \equiv y \preceq k$ and $\psi^l \equiv \neg(y \preceq k)$.

- *Case* $\psi^l \equiv \top$:

$$\begin{aligned} \mathsf{vpd}(v, \psi^c \wedge \top) &= \mathsf{vpd}(v, \psi^c) \otimes e_\otimes && \text{by Algorithm 1} \\ &= \mathsf{vpd}(v, \psi^c) \\ &= d(v, \psi^c) && \text{by inductive hypothesis} \\ &= d(v, \psi^c \wedge \top) && \text{by definition of } \top \end{aligned}$$

- *Case* $\psi^l \equiv y \preceq k$: We consider two sub-cases, $v(y) \preceq k$ and $\neg(v(y) \preceq k)$.
  - *Case* $v(y) \preceq k$:

$$\begin{aligned} \mathsf{vpd}(v, \psi^c \wedge y \preceq k) &= \mathsf{vpd}(v, \psi^c) \otimes e_\otimes && \text{by Algorithm 1} \\ &= \mathsf{vpd}(v, \psi^c) \\ &= d(v, \psi^c) && \text{by inductive hypothesis} \end{aligned}$$

    Consider an arbitrary $a \preceq k$ and $b \succ k$. Consider two arbitrary valuations v' and v'' such that $v'(y) = a$, $v''(y) = b$ and for all $x \in X \backslash \{y\}$, $v'(x) = v''(y)$. We have that $d(v'(y), y \preceq k) = e_\otimes$ and $d(v''(y), y \preceq k) = d(v''(y), k)$ by the proof of the base cases, hence $d(v'(y), y \preceq k) \sqsubseteq d(v''(y), y \preceq k)$. By Definition 11, we have that $d(v, \psi^c \wedge y \preceq k) \sqsubseteq d(v, \psi^c \wedge \neg(y \preceq k))$. We also have that:

$$\begin{aligned} d(v, \psi^c) &= d(v, \psi^c \wedge y \preceq k) \oplus d(v, \psi^c \wedge \neg(y \preceq k)) && \text{by simple rewriting} \\ &= d(v, \psi^c \wedge y \preceq k) && \text{by } d(v, \psi^c \wedge y \preceq k) \sqsubseteq d(v, \psi^c \wedge \neg(y \preceq k)) \end{aligned}$$

  - *Case* $\neg(v(y) \preceq k)$: We consider two cases, when S is multiplicatively idempotent and when $\psi^c \wedge y \preceq k$ is in $\wedge$-minimal DNF form.
    - *Case S is multiplicatively idempotent*: We first recall that $d(v, y \preceq k) = d(v, y \preceq k) \oplus d(v, \psi^c \wedge y \preceq k)$, hence by Definition 11, $d(y \preceq k) \sqsubseteq d(v, \psi^c \wedge y \preceq k)$.

$$
\begin{aligned}
\mathsf{vpd}(v, \psi^c \wedge y \preceq k) &= \mathsf{vpd}(v, \psi^c) \otimes \mathsf{vpd}(v, y \preceq k) && \text{by Algorithm 1} \\
&= d(v, \psi^c) \otimes d(v, y \preceq k) && \text{by inductive hypothesis} \\
&= (d(v, \psi^c \wedge y \preceq k) \\
&\quad \oplus d(v, \psi^c \wedge \neg(y \preceq k))) \\
&\quad \otimes d(v, y \preceq k) \\
&= d(v, \psi^c \wedge y \preceq k) \otimes d(v, y \preceq k) && \text{by assumption that } \neg(y \preceq k) \\
&= d(v, \psi^c \wedge y \preceq k) && \text{by } \otimes\text{ - idempotence of } S \\
& && \text{and Definitions 10 and 11}
\end{aligned}
$$

  - *Case* ψ^c is in $\wedge$-minimal DNF: By this assumption, ψ^c does not contain any conjunct in the form $y \preceq k'$, although it might contain a conjunct of the form $\neg(y \preceq k')$ for some $k' \leq k$. However, even if that is the case, by assumption that $v(y) \succ k$, we have that the contribution of y in $d(v, \psi^c)$ is $e_{\otimes}$. Hence, we have that $d(v, \psi^c \wedge y \preceq k)$ consists of computing $d(v, \psi^c \wedge)$ and $\otimes$-multiplying it with the effect of the $y \preceq k$ constraint, that is with $d(v, y \preceq k) = d(v(y), k)$.

- *Case* $\psi^l \equiv \neg(y \preceq k)$: symmetric to the previous case.

We are now ready to prove that $\mathsf{vpd}(v, \psi \vee \psi^c) = d(v, \psi \vee \psi^c)$.

Base case: The first part of the proof establishes that $\mathsf{vpd}(v, \psi^c) = d(v, \psi^c)$

Inductive hypothesis:

$$
\begin{aligned}
\mathsf{vpd}(v, \psi) &= \mathsf{vpd}(v, \psi) \oplus \mathsf{vpd}(v, \psi^c) && \text{by Algorithm 1} \\
&= d(v, \psi) \oplus d(v, \psi^c) && \text{by inductive hypothesis} \\
&= \oplus_{v' \models \psi} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \psi^c} \otimes_{x \in X} d(v, v') && \text{by Definition 5} \\
&= \oplus_{v' \models \psi \wedge \neg\psi^c} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \psi \wedge \psi^c} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \psi \wedge \psi^c} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \neg\psi \wedge \psi^c} \otimes_{x \in X} d(v, v') && \text{by partition of sets} \\
&= \oplus_{v' \models \psi \wedge \neg\psi^c} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \psi \wedge \psi^c} \otimes_{x \in X} d(v, v') \oplus \\
&\quad \oplus_{v' \models \neg\psi \wedge \psi^c} \otimes_{x \in X} d(v, v') && \text{by idempotence of } \oplus \\
&= \oplus_{v' \models \psi \vee \psi^c} \otimes_{x \in X} d(v, v') && \text{by union of disjoint sets} \\
&= d(v, \psi \vee \psi^c) && \text{by Definition 5.}
\end{aligned}
$$

□

Theorem 2. *Given a specification φ, its associated WSA $\mathcal{W}$ defined over a semiring S and a trace τ, we have that* $\mathsf{val}(\tau, \mathcal{W}) = \alpha(\tau, \mathcal{W}) = d(\tau, \varphi)$.

Proof. The proof follows from the monotonicity of the natural order in additively idempotent semirings. This propery allow us to merge the values of all paths π induced by a prefix of size n of the trace τ and ending in a location q into a single representative value that we represent as the cost $c(q, n)$ of the location at time n and that is used to compute the value of all the extensions. (we recall that for all $a, b, c \in S$, if $a \sqsubseteq b$, then $a \otimes c \sqsubseteq b \otimes c$). Following the definition of the trace value, the cost $c(q, n+1)$ of q at time $n+1$ is then the $\oplus$-summation of the individual effects of taking all possible transitions (s, ψ, q) from s to q with the new valuation v_{n+1} (the cost $c(s, n)$ $\otimes$-multiplied by the predicate-value distance $\mathsf{vpd}(v_{n+1}, \psi)$).

□

Theorem 3. *Given traces τ and τ', a specification φ and distances $d(\tau, \tau')$, $d(\tau, \neg\varphi)$ defined over a bounded semiring S,*

$$\begin{aligned} \rho(\tau, \varphi) > 0 &\rightarrow \tau \models \varphi \\ \rho(\tau, \varphi) < 0 &\rightarrow \tau \not\models \varphi \\ \tau \models \varphi \text{ and } d(\tau, \tau') \sqsubset d(\tau, \neg\varphi) &\rightarrow \tau' \models \varphi. \end{aligned}$$

Proof. The proof for the first two implications is trivial from the definitions of $\rho(\tau, \varphi)$ and $d(\tau, \varphi)$. We prove the third implication by contradiction. Assume that $\tau \models \varphi$, $d(\tau, \tau') \sqsubset d(\tau, \neg\varphi)$ and $\tau' \not\models \varphi$. Then, by definition of the distance, we have that $d(\tau', \neg\varphi) = e_{\otimes}$. By the additive idempotence of S and the definition of $d(\tau', \neg\varphi)$, there exists $\tau'' \models \neg\varphi$ such that $d(\tau', \tau'') = e_{\otimes}$, hence $\tau' = \tau''$. However, in that case we have that $d(\tau, \tau') = d(\tau, \neg\varphi)$, which is a contradiction. □

A.2 Theorem proofs: Quantitative Monitoring with WED

Proposition 1. *The weighted edit distance is a distance.*

Proof. The proof is similar to the proof that edit distance is a distance, and is given for completeness reason.

1. $d_W(s, s) = 0$: in order to transform s to itself, no insertions or deletions are needed, and all substitutions have cost 0.
2. $d_W(s_1, s_2) = d_W(s_2, s_1)$: insertions and deletions are inverses of each other, while substitution is symmetric.
3. $d_W(s_1, s_2) \geq 0$: by definition, all costs are greater or equal to 0.

4. $d_W(s_1, s_2) \leq d_W(s_1, s) + d_W(s, s_2)$: by definition, $d_W(s_1, s_2)$ is the minimum summation of insertion, deletion and substitution costs needed to transform s_1 to s_2. Transforming s_1 to s, and then s to s_2 is one way of transforming s_1 to s_2, and hence cannot be cheaper than $d_W(s_1, s_2)$.

□

For the next two theorems, we first prove an auxiliary lemma.

Lemma 4. *Let s be a signal, $\mathcal{W}$ a weighted symbolic automaton and π_1 and π_2 two paths in $\mathcal{W}$ induced by s such that both paths terminate in the same state $q \in Q$ and $v_{\pi_1}(s, \mathcal{W}) < v_{\pi_2}(s, \mathcal{W})$. Then, for all continuations s' of s and all paths $\pi_1 \cdot \pi$ and $\pi_2 \cdot \pi$ induced by $s \cdot s'$, the following relation holds: $v_{\pi_1 \cdot \pi}(s \cdot s', \mathcal{W}) < v_{\pi_2 \cdot \pi}(s \cdot s', \mathcal{W})$.*

Proof. The proof follows directly from the definition of path and the definition of path value in weighted automaton $\mathcal{W}$. Each continuation path of either π_1 or π_2 must start from the state q. Since both π_1 and π_2 terminates at state q it means that all symbols from s are consumed once a state q' is reached. For this reason, it is clear that only s' affects transition weights in π. As a consequence, and by the definition of $v_\pi(s, \mathcal{W})$ the following holds:

$v_{\pi_i \cdot \pi}(s \cdot s', \mathcal{W}) = v_{\pi_i}(s, \mathcal{W}) + v_\pi(s', \mathcal{W})$.

By definition, $v_\pi(s, \mathcal{W})$ is a sum of non-negative transition weights. Consequently, the following relation holds: $v_{\pi_1}(s, \mathcal{W}) + v_\pi(s', \mathcal{W}) < v_{\pi_2}(s, \mathcal{W}) + v_\pi(s', \mathcal{W})$.

□

Theorem 4. $d_W(s, \varphi) = v(s, \mathcal{W}_\varphi)$.

Due to the length of the proof, it is presented in separate appendix below.

Theorem 5. $\mathit{Val}(s, \mathcal{W}) = v(s, \mathcal{W})$.

Proof. The algorithm iteratively explores the paths in $\mathcal{W}$ induced by s and computes their values after reading the current prefix. Although the number of paths grows with the number of iterations, by Lemma 4 it is sufficient to keep in every iteration only the minimum value of reaching each state in Q after reading the current prefix. We also bound the number of consecutive silent transitions (insertion operations) to $|Q|$ in every iteration – it is guaranteed that if a state q' is reachable from state q, it can be reached within $|Q|$ steps. When the main loop of the algorithm terminates, the values associated to each state clearly correspond to the minimal values of reaching them with the signal s – the minimum value of an accepting state hence corresponds to $v(s, \mathcal{W})$.

□

Theorem 6. *Given a signal s of length l defined over X and a weighted automaton $\mathcal{W}$ with n states and m transitions,* $\textsc{Val}(s, \mathcal{W})$ *takes in the order of $\mathcal{O}(l(nm))$ iterations to compute the value of s in $\mathcal{W}$, and requires in the order of $\mathcal{O}(n(\lceil log(l(v_{\max} - v_{\min}))\rceil))$ memory.*

Proof. In Algorithm 1, there are l main iterations, and in each iterations one needs to do m updates due to substitutions/deletions and mn updates due to ϵ-transition propagation. For the space complexity, we need to keep for each state a value, that can be at most $l(v_{max} - v_{min})$ and can be encoded in binary. □

A.2.1 Proof of Theorem 4

In this section, we provide the proof that $d(s, \varphi) = v(s, \mathcal{W}_\varphi)$. In order to achieve this goal, we decompose the problem into the following smaller instances:

$$
\begin{array}{lrcll}
 & d_W(s,\varphi) & = & v(s,\mathcal{W}_\varphi) & \Leftrightarrow \\
(1) & d_W(s,L(\varphi)) & = & v(s, \bigcup\limits_{s'\in L(\varphi)} \mathcal{W}_\varphi^{s'}) & \Leftrightarrow \\
(2) & \min\limits_{s'\in L(\varphi)} d_W(s,s') & = & \min\limits_{s'\in L(\varphi)} v(s,\mathcal{W}_{s'}) & \Leftrightarrow \\
(3) & d_W(s,s') & = & v(s,\mathcal{W}_{s'}) &
\end{array}
$$

where s' represents an arbitrary word from the language of formula φ.

In (1), we first decompose $\mathcal{W}_\varphi$ into a (possibly infinite) union of weighted symbolic automata $\mathcal{W}^{s'}$, where each $\mathcal{W}^{s'}$ models all possible edit operations that are applicable to transform an arbitrary signal s into another signal $s' \in L(\varphi)$. In (2), we then show that finding $s' \in L(\varphi)$ that minimizes the distance $d_W(s, s')$ is equivalent to finding the $s' \in L(\varphi)$ that minimizes the value $v(s, \mathcal{W}^{s'})$. Finally, in (3) we show that for an arbitrary s', the distance $d_W(s, s')$ equals to the value of the value $v(s, \mathcal{W}^{s'})$. We proceed in the bottom up fashion, by proving first (3), i.e. $d(s, s') = v(s, \mathcal{W}^{s'})$, where $\mathcal{W}^{s'}$ is a weighted edit automaton obtained by applying the procedure from Section 7.2.2 to the automaton $\mathcal{A}^{s'}$ that accepts only the trace s'.

Definition 21. *Given a signal s, we define a* single word acceptor *(SWA) for s, $\mathcal{A}^s$, as the minimal automaton such that $L(\mathcal{A}^s) = \{s\}$ holds. We denote by $\mathcal{W}^s$ the* single word weighted acceptor *(SWWA) that is constructed from $\mathcal{A}^s$ by applying the procedure from Section 7.2.2.*

We note that for every s of size n, both $\mathcal{A}^s$ and $\mathcal{W}^s$ consist of a sequence of locations $q_0, \ldots, q_n$, where q_0 is the only initial location, q_n is the only final location, and for all $0 < i \leq n$, the incoming transitions to q_i have a source either in q_{i-1} or in q_i. We define by $P(s)$ the set of all prefixes of a signal s, where $s[0, i)$ denotes the prefix of s of size i, where $0 \leq i \leq |s|$. In Figure A.1, we illustrate the array of weighted edit automata $\mathcal{W}^{s[0,i)}$ for the prefixes of the signal s of size 4.

Lemma 5. *Let s and s' be two arbitrary signals of size m and n, respectively. Then, for all $0 \leq i \leq m$ and $0 \leq j \leq n$, we have that $d_W(s[0, i), s'[0, j)) = v(s[0, i), \mathcal{W}^{s'[0,j)})$.*

Proof. We prove this lemma by induction.

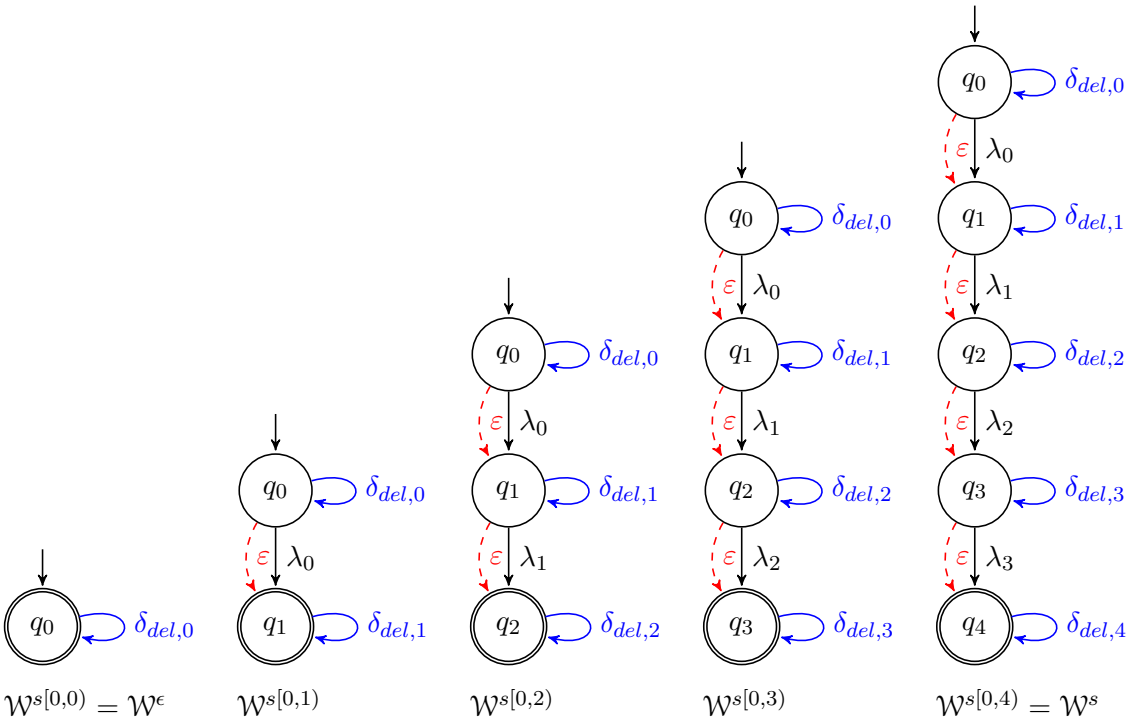


Figure A.1: An array of SWWAs $\mathcal{W}^{s[0,j)}$ that model all possible edit operations on each prefix $s[0,j) \in P(s)$ of $s \in L(\varphi)$.

Base case: We first prove that (1) $d_W(\epsilon, s'[0,j)) = v(\epsilon, \mathcal{W}^{s'[0,j)})$ for all $0 \leq j \leq n$ and (2) $d_W(s[0,i), \epsilon) = v(s[0,i), \mathcal{W}^{\epsilon})$ for all $0 \leq i \leq m$. In the case (1), by the definition of the weighted edit distance $d_W(\epsilon, s'[0,j)) = jc_i$. The cheapest path from the initial to the finite location in $\mathcal{W}^{s'[0,j)}$ induced by an empty word is by taking j consecutive ϵ (insertion) transition, each inducing a cost of c_i. In the case (2), by the definition of the weighted edit distance $d_W(s[0,i), \epsilon) = ic_d$. The cheapest path from the initial to the accepting state in $\mathcal{W}^{\epsilon}$ induced by $s[0,i)$ is to consume the i letters by consecutive self-loop (deletion) transition, each inducing a cost of c_d.

Inductive step: By inductive hypothesis, we assume that $d(s[0,i-1), s'[0,j-1)) = v(s[0,i-1), \mathcal{W}^{s'[0,j-1)})$, $d(s[0,i-1), s'[0,j)) = v(s[0,i-1), \mathcal{W}^{s'[0,j)})$ and $d(s[0,i), s'[0,j-1)) = v(s[0,i), \mathcal{W}^{s'[0,j-1)})$. We now prove that $d(s[0,i), s'[0,j)) = v(s[0,i), \mathcal{W}^{s'[0,j)})$. By definition of the weighted edit distance, we have that

$$d_W(s[0,i), s'[0,j)) \quad = \quad min \begin{cases} d_W(s[0,i-1), s'[0,j-1)) & +c_s(s(i), s'(j)) \\ d_W(s[0,i), s'[0,j-1)) & +c_i \\ d_W(s[0,i-1), s'[0,j)) & +c_d \end{cases}$$

We now prove that

$$v(s[0,i), \mathcal{W}^{s'[0,j)}) \quad = \quad min \begin{cases} v(s[0,i-1), \mathcal{W}^{s'[0,j-1)}) & +c_s(s(i), s'(j)) \\ v(s[0,i), \mathcal{W}^{s'[0,j-1)}) & +c_i \\ v(s[0,i-1), \mathcal{W}^{s'[0,j)}) & +c_d \end{cases}$$

We first observe that $\mathcal{W}^{s'[0,j)}$ has only one final location q_j. By the definition of the weighted symbolic automata, path values are non-negative and additive, and by the definition of $\mathcal{W}^{s'[0,j)}$, any location $q_{j'}$, where $0 < j' \leq j$ can be reached in one step only from $q_{j'}$ or $q_{j'-1}$. It follows that it is sufficient to consider $s[0,i)$ and $s[0,i-1)$ and q_j and q_{j-1} in order to prove $d(s[0,i), s'[0,j)) = v(s[0,i), \mathcal{W}^{s'[0,j)})$. Let $\pi = \pi' \cdot \delta \cdot q_j$ be the path with minimum value induced by $s[0,i)$ in $\mathcal{W}^{s'[0,j)}$. By the the definition of $\mathcal{W}^{s'[0,j)}$, q_j has 3 incoming transitions: (1) a substitution transition from q_{j-1} to q_j; (2) an ϵ (insertion) transition from q_{j-1} to q_j; and (3) a self-loop (deletion) transition in from q_j to q_j.

By the definition of the value in a weighted symbolic automaton, the value of π corresponds to the value of π' to which the cost of the last transition δ is added, which is the minimum of the above three cases. In the case (1), π' reaches q_{j-1} with $s[0,i-1)$ consumed. The value accumulated by π' corresponds to the value of π' induced by $s[0,i-1)$ in $\mathcal{W}^{s'[0,j-1)}$, i.e. $v(s[0,i-1), \mathcal{W}^{s'[0,j-1)})$. The added cost of the last transition corresponds to $c_s(s(i), s'(j))$. In the case (2), π' reaches q_{j-1} with $s[0,i)$ consumed. The value accumulated by π' corresponds to the value of π' induced by $s[0,i)$ in $\mathcal{W}^{s'[0,j-1)}$, i.e. $v(s[0,i), \mathcal{W}^{s'[0,j-1)})$. The added cost of the last transition corresponds to the cost c_i of an insertion. In the case (3), π' reaches q_j with $s[0,i-1)$ consumed. The value accumulated by π' corresponds to the value of π' induced by $s[0,i-1)$ in $\mathcal{W}^{s'[0,j)}$, i.e. $v(s[0,i-1), \mathcal{W}^{s'[0,j)})$. The added cost of the last transition corresponds to the cost c_d of a deletion.

□

Corollary 1. $d_W(s, s') = v(s, \mathcal{W}^{s'})$

Corollary 1 is a special case of Lemma 5, where $i = m$ and $j = n$. We can now generalize Corollary 1 with the following lemma, in order to take into account all the (possibly infinite number of) signals that are in a language of an STL formula φ.

Lemma 6. $\min_{s' \in L(\varphi)} d_W(s, s') = \min_{s' \in L(\varphi)} v(s, \mathcal{W}^{s'})$.

Proof. Follows directly from Corollary 1 and the definition of a minimum. □

Corollary 2. $d_W(s, \varphi) = v(s, \bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'})$

The above corollary follows from the definition of the weighted edit distance, and the fact that $\min$ distributes over the union. We finally need to show that the decomposition of $\mathcal{W}_\varphi$ into $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ preserves the value induced by s. We first show that the above decomposition

preserves paths, that is all possible sequences of edit operations that transform s into any $s' \in L(\varphi)$ and the values of these paths.

Lemma 7. *Consider an arbitrary trace s and an STL formula φ. For every path π in $\mathcal{W}_\varphi$ induced by s, there exists π' in $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ induced by s, and for every path π' in $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ induced by s, there exists a path π in $\mathcal{W}_\varphi$ induced by s, such that $v(s, \pi, \mathcal{W}_\varphi) = v_{\pi'}(s, \bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'})$.*

Proof. We create a SWWA $\mathcal{W}^{s'}$ for every trace $s' \in L(\varphi)$. It follows that $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ contains the paths that model all the edit operations that transform an arbitrary s into an arbitrary $s' \in L(\varphi)$. By the construction in Section 7.2.2, $\mathcal{W}_\varphi$ also models all the edit operations that transform an arbitrary s into an arbitrary $s' \in L(\varphi)$. The substitution operations are by definition preserved in the decomposition of $\mathcal{W}_\varphi$ into $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$, while the insertion and deletion transitions are systematically added in both $\mathcal{W}_\varphi$ and $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ with the same costs. It follows that for an arbitrary s, both $\mathcal{W}_\varphi$ and $\bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'}$ contain paths π and π' induced by s that model the same edit operations with the same value. □

Lemma 8. $v(s, \mathcal{W}_\varphi) = v(s, \bigcup_{s' \in L(\varphi)} \mathcal{W}^{s'})$.

Proof. Follows directly from the preservation of all paths and their values proved in Lemma 7 and the definition of a value in a weighted symbolic automata. □

The combination of Lemma 8, Corollary 2, Lemma 6 and Corollary 1 constitutes the proof of Theorem 4.

APPENDIX B

Hardware Monitors in Verilog

In the following section we provide source code of our hardware monitors, implemented in Verilog HDL. The listing below shows an implementation of a temporal logic operator $\diamondsuit_{[a,b]}p$, which is using a temporal tester for $\diamondsuit_{[0,a]}p$ presented in Figure 4.9, and a set of small one-clock automata presented in Figure 5.3.

```
1  //////////////////////////////////////////////////
2  // Company: AIT/TUW
3  // Author: Stefan Jaksic
4  // Create Date: 04/03/2015 11:03:19 AM
5  // Module Name: once_a_b_clocks_alg
6  // Target Devices: Zedboard with ZYNQ 7020
7  // Description: This code implements template code for monitor for
8  // "eventually in the past" operator with bounds [a,b]
9  // Dependencies: once_MTLB_monitor.v
10 //////////////////////////////////////////////////
11
12 // params calculated from the STL formula by monitor synthesis tool
13 //__CLK_REG_WIDTH__ - clock var register width = log2 (a)
14 //__CLK_REG_NUM__ - clock var register num = 2*ceil(b)/a   N = 2 * ceil(a/(b-
       a+1))
15
16 module __ONCE_A_B_CLK_MON_NAME__ #(parameter A=0, B=1) (
17    input clk,
18    input rst,
19    input XXX,
20    output err
21    );
22
23    reg [__CLK_REG_WIDTH__- 1:0] clocks_array [0:__CLK_REG_NUM__- 1];
24
25    //indices of clocks in clock arrays
26    reg [__CLK_REG_WIDTH__- 1:0] i;
27    reg [__CLK_REG_WIDTH__- 1:0] j;
28
```

```
29   //loop counters
30   integer m;
31   integer n;
32
33   //output signal - registered
34   reg reg_ev_out;
35   wire ev_out;
36
37   localparam VARIABILITY = B - A;
38
39   //instance of eventually_in_the_past[0,t] monitor
40   once_MTLB_monitor#(VARIABILITY) ev_mon (
41       .clk(clk),
42       .rst(rst),
43       .XXX(XXX),
44       .err(ev_out)
45   );
46
47   always @(posedge clk) begin
48    if (rst == 1'b1) begin
49  reg_ev_out <= ev_out; //init reg
50
51    for (n = 0; n < __CLK_REG_NUM__; n = n+1) begin //init clocks to max_val
52       clocks_array[n] <= {__CLK_REG_WIDTH__{1'b1}} - 2'd2;
53    end
54
55    j <= 3'b000;
56    end
57    else begin
58      reg_ev_out <= ev_out;
59
60    for (m = 0; m < ____CLK_REG_NUM____; m = m+1) begin //inc array of clocks
61      if (m != j) begin
62        if (clocks_array[m] != (A - 1))
63              clocks_array[m] <= (clocks_array[m] + 1'b1);
64      end
65    end //for
66
67    if(ev_out != reg_ev_out) begin
68      clocks_array[j] <= {__CLK_REG_WIDTH__{1'b0}};
69
70    if (j == (__CLK_REG_NUM__ - 1'b1)) begin
71      j <= {__CLK_REG_WIDTH__{1'b0}}; //artificial overflow
72    end
73    else begin
74      j <= j+1; //next clock
75    end
76      end
77    else begin
78    if (clocks_array[j] != (A - 1))
79      clocks_array[j] <= (clocks_array[j] + 1'b1);
80    end
81  end
```

```
  end

  always @(posedge clk) begin
    if (rst == 1'b1) begin
    i <= {__CLK_REG_WIDTH__{1'b0}};   //pointer init
    end
    else begin
      if (clocks_array[i] == A-2) begin
         if (i == (__CLK_REG_NUM__- 1'b1))
           i <= {__CLK_REG_WIDTH__{1'b0}};  //artificial overflow
         else
           i <= i+1; //next clock
      end
    end
  end

assign err = i[0:0] ^ 1'b1;

endmodule
```

Listing B.1: Hardware monitor for STL formula $\diamondsuit_{[a,b]}p$ implemented in Verilog

In the following Verilog source code we can see a temporal tester implementation for operator $\diamondsuit_{[0,a]}p$, as shown in Figure 4.9.

```
////////////////////////////////////////////////
// Company: AIT/TUW
// Engineer: Stefan Jaksic
// Create Date: 03/30/2015 11:24:01 AM
// Module Name: once_MTLB_monitor
// Target Devices: Zedboard with ZYNQ 7020
// Description: Verilog template of the monitor for
// "once in the past" operator
// in Signal Temporal Logic past time. The bounds are always between [0,TAU].
// Dependencies:
////////////////////////////////////////////////

module __ONCE_0_TAU_MONITOR_NAME__ #(parameter TAU=1)(
    input clk,
    input rst,
    input XXX,
    output err
    );

    reg [1:0] state;
    reg [__ONCE_0_TAU_CNTR_MSB__:0] counter;
    reg u;

    always @(posedge clk) begin

      if(rst === 1'b1) begin            //init block
        counter <= {__ONCE_0_TAU_CNTR_MSB__{1'b0}};
        state <= 2'b00;
        u <= 1'b0 ;
```

```
30      end
31      else begin
32
33      case(state)
34        2'b00: begin          //virtual starting state
35            if (XXX == 1'b1) begin
36                state <= 2'b01;
37                u <= 1'b1;
38            end
39            else begin
40                state <= 2'b11;
41                u <= 1'b0;
42            end
43        end
44        2'b01: begin
45            if ((XXX == 1'b0)) begin
46                state <= 2'b10;
47                u <= 1'b1;
48                counter <= counter + 1'b1;
49            end
50            else
51                counter <= 16'h0000;
52        end
53        2'b10: begin
54            counter <= counter + 1'b1;
55
56            if (XXX == 1'b1) begin
57                state <= 2'b01;
58                counter <= 16'h0000;
59                u <= 1'b1;
60            end
61            else if (counter >= TAU) begin
62                state <= 2'b11;
63                counter <= 16'h0000;
64                u <= 1'b0;
65            end
66        end
67        2'b11: begin
68            if (XXX == 1'b1) begin
69                state <= 2'b01;
70                counter <= 16'h0000;
71                u <= 1'b1;
72            end
73        end
74      endcase
75    end
76    end
77
78  assign err = !u;
79
80 endmodule
```

Listing B.2: Hardware monitor for STL formula $\diamondsuit_{[0,a]} p$ implemented in Verilog

Acronyms

ABS Anti-lock Break System. xvii, 31, 32

ABV Assertion-Based Verification. 2, 6, 7

AD Analog to Digital. 60

ADC Analog-to-Digital Converter. 58, 60, 83, 111

AFE Analog Front-End. 111

AHB Advanced High-performance Bus. xiv, xvii, xxi, 17, 23, 28–30, 32, 33

AMBA Advanced Microcontroller Bus Architecture. 17, 23, 28

AMS Analog and Mixed-Signal. 17, 18

AMT Analog Monitoring Tool. 6

ARM Arm Holdings. 23, 28

ARV Algebraic Runtime Verification. xvi, 12–14, 21, 22, 67, 68, 71, 79, 80, 119–122, 125, 127

CDV Coverage Driven Verification. 2–6

CEGAR Counterexample-guided Abstraction Refinement. 2

CLB Configurable Logic Block. 58

CPS Cyber-Physical System. ix, xi, xiii, 1, 2, 4, 5, 7, 10, 12, 13, 18, 19, 25, 67, 83, 119, 121

CPU Central Processing Unit. 2, 12, 23

DAG Directed Acyclic Graph. 59

DDR Double Data Rate. 6

DNF Disjunctive Normal Form. 36

EBS Electronic Braking System. 102

ECU Engine Control Unit. 102, 111

EPS Electronic Power Steering. 102

eRM *e* Reuse Methodology. 4

FF flip-flop. 58, 60, 111, 116

FPGA Field Programmable Gate Array. xvii, xxii, 9, 12, 14, 25, 35, 51, 52, 58–62, 96, 101, 111, 113, 119–121

FT Falsification Testing. 2, 3

HDL Hardware Description Language. 9, 110, 119, 121, 135

HiL Hardware-in-the-Loop. ix, xi, 9, 10, 12, 51, 121

HLS High-Level Synthesis. xxii, 101, 110, 112, 113, 120

IC Integrated Circuit. 51

IEEE Institute of Electrical and Electronics Engineers. 4

IP Intellectual Property. 3, 7, 59, 60, 141

LTL Linear Temporal Logic. 7, 17, 21

LUT lookup table. 58, 60, 63, 111, 116

MTL Metric Temporal Logic. 11, 18, 25

NASA National Aeronautics and Space Administration. 1, 6

OEM Original Equipment Manufacturers. 102

OVM Open Verification Methodology. 4

PSL Property Specification Language. 6, 7, 17, 30, 58

RAM Random-access Memory. 58

RTL register transfer level. xviii, 4, 8, 9, 112, 119

RV Runtime Verification. xiii, xiv, 2, 5–7, 11–13, 21, 121

SA Symbolic Automaton. ix, xi, 37

SAT boolean satisfiability problem. 2

SENT Single Edge Nibble Transmission. x, xi, xv, 14, 101–106, 108–110, 112–117, 120

SMT satisfiability modulo theory. 18

SoC System on Chip. 9, 58

SPI Serial Peripheral Interface. 52, 64, 65

SRE Signal Regular Expressions. 10

STL Signal Temporal Logic. ix, xi, xiii–xv, xxi, xxii, 5, 6, 10, 12–19, 23, 25, 27–30, 32–34, 44, 47, 51–53, 61–63, 67, 69–71, 83, 84, 89–91, 93, 95, 101, 103, 106, 107, 109, 111, 113, 119–122, 137, 138

SUT System Under Test. xvii, 2, 4, 6, 7, 10, 12, 18, 20, 60, 61, 96, 120

SVA System Verilog Assertions. 6, 7, 30

TB test bench. 65

TRE Timed Regular Expressions. xiv, xv, xxi, xxii, 14, 17, 18, 23, 25, 30–34, 58, 69–71, 95, 101, 103–111, 113, 120, 122

UVM Universal Verification Methodology. 4, 9

UxAS Unmanned Systems Autonomy Services. 5

V&V Verification and Validation. 51, 101

VHDL VHSIC Hardware Description Language. 6, 110

VHSIC Very High Speed Integrated Circuit. 141

VIP Verification IP. 3, 4

WED Weighted Edit Distance. ix, xi, xv, xvi, 13, 14, 68, 84, 95, 101, 112, 113, 115–117, 119–122, 129, 131, 133

WSA Weighted Symbolic Automata. xv, xvii, xxi, 13, 14, 21, 35, 38, 39, 69–71, 74, 76, 77, 81, 129

XADC Xilinx analog-to-digital converter. 60

Bibliography

[ABB16] Lacramioara Astefanoaei, Saddek Bensalem, and Marius Bozga. A compositional approach to the verification of hybrid systems. In *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*, pages 88–103, 2016.

[ACH+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid systems*, pages 209–229. Springer, 1993.

[ACM97] Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Logic in Computer Science (LICS)*, pages 160–171, 1997.

[ACM02a] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[ACM02b] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.

[ADF+05] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, J. Sanchez-Gasca, R. Schulz, A. Stankovic, C. Taylor, and V. Vittal. Causes of the 2003 major grid blackouts in north america and europe, and recommended means to improve system dynamic performance. *IEEE Transactions on Power Systems*, 20(4):1922–1928, Nov 2005.

[AFJ13] Roy Armoni, Dana Fisman, and Naiyong Jin. SVA and PSL local variables - A practical approach. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 197–212, 2013.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[AHF+14] Houssam Abbas, Bardh Hoxha, Georgios E. Fainekos, Jyotirmoy V. Deshmukh, James Kapinski, and Koichi Ueda. Conformance testing as falsification for cyber-physical systems. *CoRR*, abs/1401.5200, 2014.

[ALFS11a] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 254–257, 2011.

[ALFS11b] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Proc. of TACAS 2011: the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[AM09] Cyril Allauzen and Mehryar Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. *CoRR*, abs/0904.4686, 2009.

[AMF14] Houssam Abbas, Hans D. Mittelmann, and Georgios E. Fainekos. Formal property verification in a conformance testing framework. In *Proc. of MEMOCODE 2014: the Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 155–164. IEEE, 2014.

[ARK+13] Matthias Althoff, Akshay Rajhans, Bruce H. Krogh, Soner Yaldiz, Xin Li, and Larry T. Pileggi. Formal verification of phase-locked loops using reachability analysis and continuization. *Commun. ACM*, 56(10):97–104, 2013.

[AT15] Takumi Akazaki and Ichiro Tasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *Computer Aided Verification, 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2011, Proceedings*, 2015.

[Axe07] Jan Axelson. *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems; 2nd ed.* Lakeview Research, Madison, WI, 2007.

[BBCT14] Saddek Bensalem, Marius Bozga, Jacques Combaz, and Ahlem Triki. Rigorous system design flow for autonomous systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 184–198, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[BBKT05] Saddek Bensalem, Marius Bozga, Moez Krichen, and Stavros Tripakis. Testing conformance of real-time applications by automatic generation of observers. *Electronic Notes in Theoretical Computer Science*, 113:23 – 43, 2005. Proceedings of the Fourth Workshop on Runtime Verification (RV 2004).

[BBLN17] Ezio Bartocci, Luca Bortolussi, Michele Loreti, and Laura Nenzi. Monitoring mobile and spatially distributed cyber-physical systems. In *Proc.of MEMOCODE 2017: the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 146–155. ACM, 2017.

[BBS14a] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. Data-driven statistical learning of temporal logic properties. In *Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings*, pages 23–37, 2014.

[BBS+14b] Sara Bufo, Ezio Bartocci, Guido Sanguinetti, Massimo Borelli, Umberto Lucangelo, and Luca Bortolussi. Temporal logic based monitoring of assisted ventilation in intensive care patients. In *Proc. of ISoLA 2014: the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Part II*, volume 8803 of *Lecture Notes in Computer Science*, pages 391–403. Springer, 2014.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in computers*, 58:117–148, 2003.

[BCG+11] R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, and B. Jobstmann. Specification-centered robustness. In *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, pages 176–185, June 2011.

[BCM+92] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[BDSV14] Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. STL*: Extending Signal Temporal Logic with signal-value freezing operator. *Inf. Comput.*, 236:52–67, 2014.

[Bel58] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[BFMU17] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. On the quantitative semantics of regular expressions over real-valued signals. In *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, pages 189–206, 2017.

[BGHS04] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Program monitoring with LTL in EAGLE. In *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*, 2004.

[BGJ+07] Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, compile, run: Hardware from psl. *Electronic*

Notes in Theoretical Computer Science, 190(4):3 – 16, 2007. Proceedings of the Workshop on Compiler Optimization meets Compiler Verification (COCV 2007).

[BHF15] Houssam Abbas Bardh Hoxha and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. *Proc. of ARCH@CPSWeek 2014 and ARCH@CPSWeek 2015: the 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, 34, 2015.

[BHW+13] R. Backasch, C. Hochberger, A. Weiss, M. Leucker, and R. Lasslop. Runtime verification for multicore soc with high-quality trace data. *ACM Transactions on Design Automation of Electronic Systems*, 18(2), 2013.

[BJK14] Roderick Bloem, Swen Jacobs, and Ayrat Khalimov. Parameterized synthesis case study: AMBA AHB. In *Proceedings 3rd Workshop on Synthesis, SYNT 2014, Vienna, Austria, July 23-24, 2014.*, pages 68–83, 2014.

[BKKW15] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis:. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 533–548, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[BLL+95] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop, Ruttgers University, NJ, USA*, pages 232–243, 1995.

[BLMA+05] D. Borrione, Miao Liu, K. Morin-Allory, P. Ostier, and L. Fesquet. On-line assertion-based verification with proven correct monitors. In *Proc. of ITI 2005: the 3rd International Conference on Information and Communications Technology*, pages 125–143, 2005.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[BVSF13] Lubos Brim, Tomas Vejpustek, David Safránek, and Jana Fabriková. Robustness analysis for value-freezing signal temporal logic. In *Proceedings Second International Workshop on Hybrid Systems and Biology, HSB 2013, Taormina, Italy, 2nd September 2013.*, pages 20–36, 2013.

[BZ05] M. Boulé and Z. Zilic. Incorporating efficient assertion checkers into hardware emulation. In *Proc. of ICCD*, pages 221–228. IEEE Computer Society Press, 2005.

[BZ06] M. Boulé and Z. Zilic. Efficient automata-based assertion-checker synthesis of PSL properties. In *Proc. of HLDVT*, pages 69–76. IEEE, 2006.

[BZ08] M. Boulé and Z. Zilic. Automata-based assertion-checker synthesis of PSL properties. *ACM Transactions on Design Automation of Electronic Systems*, 13(1), 2008.

[CDH08] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. In *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, pages 385–400, 2008.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, May 1981*, pages 52–71, 1981.

[CES13] K. Claessen, N. Een, and B. Sterin. A circuit approach to ltl model checking. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 53–60, Oct 2013.

[CGJ+00] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[CR05] Feng Chen and Grigore Roşu. Java-mop: A monitoring oriented programming environment for java. In Nicolas Halbwachs and Lenore D. Zuck, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 546–550, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[CSM91] Pak K Chan, Martine DF Schlagy, and Marcelo Martinzx. Borg: A reconfigurable prototyping board using field-programmable gate arrays. *algorithms*, 3:4, 1991.

[CSRB13] Chih-Hong Cheng, Natarajan Shankar, Harald Ruess, and Saddek Bensalem. EF-SMT: A logical framework for cyber-physical systems. *CoRR*, abs/1306.3456, 2013.

[Dav09] Jennifer M. Davoren. Epsilon-tubes and generalized skorokhod metrics for hybrid paths spaces. In *Proc. of HSCC 2009: the 12th International Conference on Hybrid Systems: Computation and Control*, volume 5469 of *LNCS*, pages 135–149. Springer, 2009.

[DDD+15] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, pages 127–142, 2015.

[DDG+17] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017.

[DDM04] Thao Dang, Alexandre Donzé, and Oded Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, pages 21–36, 2004.

[DDS17] Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, pages 357–372, 2017.

[DFM13] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *Computer Aided Verification (CAV)*, pages 264–279, 2013.

[DFM+15] Philip Daian, Yliès Falcone, Patrick Meredith, Traian Florin Şerbănuţă, Shin'ichi Shiriashi, Akihito Iwai, and Grigore Rosu. Rv-android: Efficient parametric android runtime verification, a brief tutorial. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, pages 342–357, Cham, 2015. Springer International Publishing.

[DGG+05] A. Dahan, D. Geist, L. Gluhovsky, D. Pidan, G. Shapir, Y. Wolfsthal, L. Benalycherif, R. Kamidem, and Y. Lahbib. Combining system level modeling with assertion based verification. In *Proc. of ISQED 2005: Sixth International Symposium on Quality of Electronic Design*, pages 310–315. IEEE, 2005.

[DHF14] Adel Dokhanchi, Bardh Hoxha, and Georgios E. Fainekos. On-line monitoring for temporal logic robustness. In *Proc. RV 2014: the 5th International Conference on Runtime Verification*, volume 8734 of *Lecture Notes in Computer Science*, pages 231–246. Springer, 2014.

[DJDS15] A. Donzé, X. Jin, J. V. Deshmukh, and S. A. Seshia. Automotive systems requirement mining using breach. In *2015 American Control Conference (ACC)*, pages 4097–4097, July 2015.

[DJKM15] Jyotirmoy Deshmukh, Xiaoqing Jin, James Kapinski, and Oded Maler. Stochastic local search for falsification of hybrid systems. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Automated Technology for Verification and Analysis*, pages 500–517, Cham, 2015. Springer International Publishing.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer-Verlag Berlin Heidelberg, 2009.

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 92–106, 2010.

[DM12] Deepak D'Souza and Raj Mohan Matteplackel. A compositional hierarchical monitoring automaton construction for LTL. In *Theoretical Aspects of Computing -*

ICTAC 2012 - 9th International Colloquium, Bangalore, India, September 24-27, 2012. Proceedings, pages 16–29, 2012.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.

[DMP15] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the skorokhod metric (full version). *CoRR*, abs/1505.05832, 2015.

[Don10] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 167–170, 2010.

[Dow97] Mark Dowson. The ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, 22(2):84, 1997.

[DSS+05a] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*, pages 166–174, 2005.

[DSS+05b] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: Runtime monitoring of synchronous systems. In *Proceedings of the 12th International Symposium of Temporal Representation and Reasoning (TIME 2005)*, pages 166–174. IEEE Computer Society Press, 2005.

[DV14] Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 541–554, 2014.

[DV15] Loris D'Antoni and Margus Veanes. Extended symbolic finite automata and transducers. *Formal Methods in System Design*, 47(1):93–119, 2015.

[DV17] Loris D'Antoni and Margus Veanes. The power of symbolic automata and transducers. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 47–67, 2017.

[DZS+15] Adel Dokhanchi, Aditya Zutshi, Rahul T. Sriniva, Sriram Sankaranarayanan, and Georgios E. Fainekos. Requirements driven falsification with coverage metrics. In *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*, pages 31–40, 2015.

[EF06] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Series on Integrated Circuits and Systems. Springer, 2006.

[EF08] Cindy Eisner and Dana Fisman. Augmenting a regular expression-based temporal logic with local variables. In *Formal Methods in Computer-Aided Design, FMCAD 2008, Portland, Oregon, USA, 17-20 November 2008*, pages 1–8, 2008.

[EF18] Cindy Eisner and Dana Fisman. Temporal logic made practical. *Handbook of Model Checking, Springer*, 2018, 2018.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, pages 27–39, 2003.

[EKK05] Joel M. Esposito, Jongwoo Kim, and Vijay Kumar. *Adaptive RRTs for Validating Hybrid Robotic Control Systems*, pages 107–121. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[FGD+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 379–395, 2011.

[FHR13] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. In *Engineering Dependable Software Systems*, pages 141–175. 2013.

[FJN+11] Yliès Falcone, Mohamad Jaber, Thanh-Hung Nguyen, Marius Bozga, and Saddek Bensalem. Runtime verification of component-based systems. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods*, pages 204–220, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[FK09] B. Finkbeiner and L. Kuhtz. Monitor circuits for ltl with bounded and unbounded future. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5779 LNCS:60–75, 2009.

[FKR06] Goran Frehse, Bruce H. Krogh, and Rob A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*, pages 257–262, 2006.

[FMFR11] Yliès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods in System Design*, 38(3):223–262, 2011.

[FMNU15] Thomas Ferrère, Oded Maler, Dejan Ničković, and Dogan Ulus. *Measuring with Timed Patterns*, pages 322–337. Springer International Publishing, Cham, 2015.

[FP09] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[FSIG09] Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Robustness of model-based simulations. In *Proc. of RTSS 2009: the 30th IEEE Real-Time Systems Symposium*, pages 345–354. IEEE Computer Society, 2009.

[FSUY12] Georgios E. Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using S-TaLiRo. In *American Control Conference, ACC 2012, Montreal, QC, Canada, June 27-29, 2012*, pages 3567–3572, 2012.

[Gla09] Mark Glasser. *Open Verification Methodology Cookbook*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[GPVW96] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proc. of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification*, volume 38 of *IFIP Conference Proceedings*, pages 3–18. Chapman & Hall, 1996.

[HAF14] Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015.*, pages 25–30, 2014.

[HDF18] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1):79–93, 2018.

[HKPV95] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 373–382, 1995.

[HKWW17] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 3–29, 2017.

[HMN01] Y. Hollander, M. Morley, and A. Noy. The e language: a fresh separation of concerns. In *Proceedings Technology of Object-Oriented Languages and Systems. TOOLS 38*, pages 41–50, 2001.

[HR02] Klaus Havelund and Grigore Rosu. Synthesizing monitors for safety properties. In *Proc. of TACAS 2002: the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *LNCS*, pages 342–356. Springer, 2002.

[HR04] Klaus Havelund and Grigore Roşu. An overview of the runtime verification tool java pathexplorer. *Formal Methods in System Design*, 24(2):189–215, Mar 2004.

[HV16] Luisa Herrmann and Heiko Vogler. Weighted symbolic automata with data storage. In *Proc. of DLT 2016: the 20th International Conference on Developments in Language Theory*, volume 9840 of *LNCS*, pages 203–215. Springer, 2016.

[IEE17] Ieee standard for universal verification methodology language reference manual. *IEEE Std 1800.2-2017*, pages 1–472, May 2017.

[IJ13] S. Iman and S. Joshi. *The e Hardware Verification Language*. Springer US, 2013.

[Inc] Xilinx Inc. Vivado High-Level Synthesis. http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html *(Accessed 18.01.2017)*.

[Inc16] Xilinx Inc. Vivado Design Suite Tutorial, Programming and Debugging. `http://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug936-vivado-tutorial-programming-debugging.pdf`, 2016. [Online; Accessed 12-January-2017].

[Int16a] SAE International. SENT - Single Edge Nibble Transmission for Automotive Applications, J2716, Standard. `http://standards.sae.org/j2716_201001/`, 2016. [Online; Accessed 21-January-2017].

[Int16b] SAE International. SENT - Single Edge Nibble Transmission for Automotive Applications, J2716, Standard. `http://standards.sae.org/j2716_201001/`, 2016. [Online; Accessed 21-January-2017].

[ISO] ISO 26262:2011. *Road Vehicles – Functional Safety*. ISO, Geneva, Switzerland.

[jau] `http://jautomata.sourceforge.net/`.

[JBG+15] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Reinhard Kloibhofer, Thang Nguyen, and Dejan Ničković. From signal temporal logic to FPGA monitors. In *13. ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2015, Austin, TX, USA, September 21-23, 2015*, pages 218–227, 2015.

[JBG+18] Stefan Jakšić, Ezio Bartocci, Radu Grosu, Thang Nguyen, and Dejan Ničković. Quantitative monitoring of stl with edit distance. *Formal Methods in System Design*, 53(1):83–112, Aug 2018.

[JBGN16] Stefan Jakšić, Ezio Bartocci, Radu Grosu, and Dejan Ničković. Quantitative monitoring of STL with edit distance. In *Runtime Verification - 16th International Conference, RV 2016, Madrid, Spain, September 23-30, 2016, Proceedings*, pages 201–218, 2016.

[JBGN18] Stefan Jakšić, Ezio Bartocci, Radu Grosu, and Dejan Ničković. An algebraic framework for runtime verification. In *International Conference on Embedded Software, EMSOFT 2018, Torino, Italy: (to appear)*, 2018.

[JDDS15] 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*, 34(11):1704–1717, Nov 2015.

[JKN10] Kevin D. Jones, Victor Konrad, and Dejan Ničković. Analog property checkers: a ddr2 case study. *Formal Methods in System Design*, 36(2):114–130, Jun 2010.

[KBD+17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017.

[KCDK15] Aaron Kane, Omar Chowdhury, Anupam Datta, and Philip Koopman. A case study on runtime monitoring of an autonomous research vehicle (arv) system. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, pages 102–117, Cham, 2015. Springer International Publishing.

[KDJ+16] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems Magazine*, 36(6):45–64, Dec 2016.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G Griswold. An overview of aspectj. In *European Conference on Object-Oriented Programming*, pages 327–354. Springer, 2001.

[KK13] Yoko Kubota and B Klayman. Japan carmakers recall 3.4 million vehicles for takata airbag flaw. *Reuter, April*, 11, 2013.

[Kon07] Stavros Konstantinidis. Computing the edit distance of a regular language. *Inf. Comput.*, 205(9):1307–1316, 2007.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KPR98] Yonit Kesten, Amir Pnueli, and Li-on Raviv. Algorithmic verification of linear temporal logic specifications. In *International Colloquium on Automata, Languages, and Programming*, pages 1–16. Springer, 1998.

[Kra12] Eugene F Krause. *Taxicab geometry: An adventure in non-Euclidean geometry*. Courier Corporation, 2012.

[KW17] P. Koopman and M. Wagner. Autonomous vehicle safety: An interdisciplinary challenge. *IEEE Intelligent Transportation Systems Magazine*, 9(1):90–96, Spring 2017.

[Lev66] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, February 1966.

[Lev04] V. Levin. Static driver verifier, a formal verification tool for windows device drivers. In *Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proceedings. Second ACM and IEEE International Conference on*, pages 151–, June 2004.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

[LT93] Nancy G Leveson and Clark S Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[Ltd06] ARM Ltd. AMBA 3 AHB-Lite Protocol v1.0. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0033a/index.html`, 2006. [Online; Accessed 22-April-2018].

[LW06] Jarred Adam Ligatti and David P Walker. *Policy enforcement via program monitoring*. Princeton University, 2006.

[LWJ+10] Scott Little, David Walter, Kevin R. Jones, Chris J. Myers, and Alper Sen. Analog/mixed-signal circuit verification using models generated from simulation traces. *Int. J. Found. Comput. Sci.*, 21(2):191–210, 2010.

[Mal16] Oded Maler. Some thoughts on runtime verification. In Yliès Falcone and César Sánchez, editors, *Runtime Verification*, pages 3–14, Cham, 2016. Springer International Publishing.

[mat] `http://www.mathworks.com/products/demos/stateflow/fuelsys.html`.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, pages 447–484, 1991.

[MN04] Oded Maler and Dejan Ničković. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*, pages 152–166, 2004.

[MN13] Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.

[MNP07] Oded Maler, Dejan Ničković, and Amir Pnueli. On synthesizing controllers from bounded-response properties. In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, pages 95–107, 2007.

[Moh02] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[Moh03] Mehryar Mohri. Edit-distance of weighted automata: General definitions and algorithms. *Int. J. Found. Comput. Sci.*, 14(6):957–982, 2003.

[MP16] Rupak Majumdar and Vinayak S. Prabhu. Computing distances between reach flowpipes. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, pages 267–276, 2016.

[MRS17] Patrick Moosbrugger, Kristin Y. Rozier, and Johann Schumann. R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Formal Methods in System Design*, 51(1):31–61, 2017.

[Nič08] Dejan Ničković. *Checking Timed and Hybrid Properties: Theory and Applications. (Vérification de propriétés temporisées et hybrides: théorie et applications)*. PhD thesis, Joseph Fourier University, Grenoble, France, 2008.

[NLM+18] Dejan Ničković, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. AMT 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, pages 303–319, 2018.

[NM07] Dejan Ničković and Oded Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*, pages 304–319, 2007.

[NN14a] Thang Nguyen and Dejan Ničković. Assertion-based monitoring in practice - checking correctness of an automotive sensor interface. In *Formal Methods for Industrial Critical Systems (FMICS)*, pages 16–32, 2014.

[NN14b] Thang Nguyen and Dejan Ničković. Assertion-based monitoring in practice - checking correctness of an automotive sensor interface. In *Formal Methods for Industrial Critical Systems (FMICS)*, pages 16–32, 2014.

[NW14] Thang Nguyen and Stuart Wooters. FPGA-based development for sophisticated automotive embedded safety critical system. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 7(1):125–132, 2014.

[Par13] Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.

[PKV07] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Hybrid systems: From verification to falsification. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification*, pages 463–476, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[PKV09] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Falsification of ltl safety properties in hybrid systems. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 368–382, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[Pri95] Dick Price. Pentium fdiv flaw-lessons learned. *IEEE Micro*, 15(2):86–88, 1995.

[PSL10] Ieee standard for property specification language (psl). *IEEE Std 1850-2010 (Revision of IEEE Std 1850-2005)*, pages 1–182, April 2010.

[PZ08] A. Pnueli and A. Zaks. On the Merits of Temporal Testers. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*, pages 172–195. Springer Berlin Heidelberg, 2008.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, pages 337–351, 1982.

[Que13] Jan-David Quesel. *Similarity, Logic, and Games - Bridging Modeling Layers of Hybrid Systems*. PhD thesis, Universität Oldenburg, 2013.

[(R208] ANSI E1.11-2008 (R2013). Entertainment Technology – USITT DMX512-A – Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories . `http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+E1.11-2008+(R2013)`, 2008. [Online; Accessed 20-January-2017].

[RBFS08] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *Computational Methods in Systems Biology, 6th International Conference, CMSB 2008, Rostock, Germany, October 12-15, 2008. Proceedings*, pages 251–268, 2008.

[RBNG16] Alena Rodionova, Ezio Bartocci, Dejan Ničković, and Radu Grosu. Temporal logic as filtering. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 11–20. ACM, 2016.

[RCD+16] Alastair Reid, Rick Chen, Anastasios Deligiannis, David Gilday, David Hoyes, Will Keen, Ashan Pathirane, Owen Shepherd, Peter Vrabel, and Ali Zaidi. End-to-end verification of processors with isa-formal. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 42–58, 2016.

[RDS+15] 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, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 239–248, 2015.

[RFB14] T. Reinbacher, M. Függer, and J. Brauer. Runtime verification of embedded real-time systems. *Formal Methods in System Design*, 44(3):230–239, 2014.

[ROA+17] Alena Rodionova, Matthew O'Kelly, Houssam Abbas, Vincent Pacelli, and Rahul Mangharam. An autonomous vehicle control stack. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek) on April 17, 2017 in Pittsburgh, PA, USA*, pages 44–51, 2017.

[RRS14] Thomas Reinbacher, Kristin Y. Rozier, and Johann Schumann. Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proc. of TACAS 2014*, volume 8413 of *LNCS*, pages 357–372. Springer-Verlag, 2014.

[RSM+99] S Raman, N Sivashankar, W Milam, W Stuart, and S Nabi. Design and implementation of hil simulators for powertrain control system software development. In *American Control Conference, 1999. Proceedings of the 1999*, volume 1, pages 709–713. IEEE, 1999.

[SB06] Volker Stolz and Eric Bodden. Temporal assertions using aspectj. *Electr. Notes Theor. Comput. Sci.*, 144(4):109–124, 2006.

[SDC13] Roopsha Samanta, Jyotirmoy V. Deshmukh, and Swarat Chaudhuri. Robustness analysis of string transducers. In *Proc. of ATVA 2013: the 11th International Symposium on Automated Technology for Verification and Analysis*, volume 8172 of *LNCS*, pages 427–441. Springer, 2013.

[SH08] Sebastian Steinhorst and Lars Hedrich. Model checking of analog systems using an analog specification language. In *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*, pages 324–329, 2008.

[Shi11] AL Shimpi. The source of intel's cougar point sata bug. *World Wide Web*, 2011.

[SJN+17] Konstantin Selyunin, Stefan Jakšić, Thang Nguyen, Christian Reidl, Udo Hafner, Ezio Bartocci, Dejan Ničković, and Radu Grosu. Runtime monitoring with recovery of the SENT communication protocol. In *Proc. of CAV 2017: the 29th International Conference on Computer Aided Verification*, volume 10426 of *LNCS*, pages 336–355. Springer, 2017.

[SKC+17] Sriram Sankaranarayanan, Suhas Akshar Kumar, Faye Cameron, B. Wayne Bequette, Georgios E. Fainekos, and David M. Maahs. Model-based falsification of an artificial pancreas control system. *SIGBED Review*, 14(2):24–33, 2017.

[Sko56] Anatolii Vladimirovich Skorokhod. Limit theorems for stochastic processes. *Theory of Probability & Its Applications*, 1(3):261–290, 1956.

[SM] U. Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1):67–85.

[SNBG16] Konstantin Selyunin, Thang Nguyen, Ezio Bartocci, and Radu Grosu. *Applying Runtime Monitoring for Automotive Electronic Development*, pages 462–469. Springer International Publishing, 2016.

[SVs13] Ieee standard for systemverilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, pages 1–1315, Feb 2013.

[sys] SystemC.

[THD+18] Cumhur Erkan Tuncali, Bardh Hoxha, Guohui Ding, Georgios E. Fainekos, and Sriram Sankaranarayanan. Experience report: Application of falsification methods on the uxas system. In *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings*, pages 452–459, 2018.

[TKID18] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V. Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 30:1–30:6, New York, NY, USA, 2018. ACM.

[Uns00] Michael Unser. Sampling 50 years after shannon. *Proceedings of the IEEE*, 88(4):569–587, 2000.

[VBdM10] Margus Veanes, Nikolaj Bjørner, and Leonardo Mendonça de Moura. Symbolic automata constraint solving. In *Proc. of LPAR-17: the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 of *LNCS*, pages 640–654. Springer, 2010.

[vdSY15] Hans van der Schoot and Ahmed Yehia. Uvm and emulation: How to get your ultimate testbench acceleration speed-up. In *Design and Verification Europe 2015, DVCON 2015, Proceedings*, 2015.

[VHL+12] Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjørner. Symbolic finite state transducers: algorithms and applications. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 137–150, 2012.

[VKP15] S. Venkataramanan, A. Kumari, and L. Piper. *SystemVerilog Assertions Handbook, 4th Edition: ... for Dynamic and Formal Verification*. CreateSpace Independent Publishing Platform, 2015.

[Wag74] Robert A. Wagner. Order-n correction for regular languages. *Commun. ACM*, 17(5):265–268, May 1974.

[WBKW07] Thomas Witkowski, Nicolas Blanc, Daniel Kroening, and Georg Weissenbacher. Model checking concurrent linux device drivers. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 501–504, New York, NY, USA, 2007. ACM.

[Yeu04] Ping Yeung. The four pillars of assertion-based verification. *Mentor Graphics Corporation*, 2004.

[YHF12] Hengyi Yang, Bardh Hoxha, and Georgios Fainekos. Querying parametric temporal logic properties on embedded systems. In *IFIP International Conference on Testing Software and Systems*, pages 136–151. Springer, 2012.