UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Thomas Ferrère

Thèse dirigée par **Oded Maler**

préparée au sein du laboratoire **Verimag**
et de l'école doctorale **Mathématiques, Sciences et Technologies de l'Information et l'Informatique**

Assertions and Measurements for Mixed-Signal Simulation

Thèse soutenue publiquement le **28 octobre 2016**,
devant le jury composé de :

Prof. Ahmed Bouajjani
Université Paris 7, Président
Dr. Jyotirmoy Deshmukh
Toyota, Rapporteur
Prof. Javier Esparza
Université technique de Munich, Rapporteur
Prof. Thomas Henzinger
IST Austria, Examinateur
Dr. Dejan Nickovic
Austrian Institute of Technology, Examinateur
Prof. Alberto Sangiovanni-Vincentelli
Université de Californie à Berkeley, Examinateur
Dr. Oded Maler
CNRS, Directeur de thèse
Dr. Ernst Christen
Mentor Graphics, Co-encadrant

Acknowledgements

I must start by thanking my supervisor, Oded Maler, whose benevolent guidance, support, encouragement, and trust were essential to this thesis being what it is – tautology intended. Working alongside Oded was a stimulating and rewarding experience, and I learned a great deal in the process, from theoretical to practical aspects of research.

I am also indebted to my co-advisor, Ernst Christen, who spared countless hours explaining, discussing, debating, and challenging conceptions that I came to form about the mixed-signal domain. Ernst's top-down approach to research ensured a complementary, somewhat reverse-engineered flow from practice to theory. I thank both Ernst and Oded for the attention they devoted to proof-reading manuscript drafts.

I was privileged to be read and heard by a jury totaling eight researchers and professors, to whom I am very grateful. I thank Ahmed Bouajjani, Jyo Deshmukh, and Javier Esparza for their careful reading and commenting of the manuscript, which gave me the opportunity to further improve its quality. I thank Tom Henzinger, Dejan Nickovic, and Alberto Sangiovanni-Vincentelli for their in-depth questions and remarks, some of which already led to new research directions being investigated.

Let me take this opportunity to thank other colleagues in relation with my two host institutions: the ADMS team of Mentor Graphics Corporation, and the Verimag laboratory of Grenoble University.

In Mentor Graphics, I would like to thank Jean-Marc Talbot for providing both high-level oversight and in-depth remarks during regular progress meetings. I am grateful to Martin Vlach for contributing to the initial research proposal. Polen Kission was responsible for the early orientation of the applicative focus of my research, and furthermore took part throughout in ensuring its timely progression. Thank you Polen for enabling the thesis to run smoothly. Among other Mentor colleagues, the help of Serge Garcia-Sabiro and Eric Jeandeau was instrumental in its provision of realistic use-cases and the sharing of their related applicative research. Serge also gave feedback on my own research efforts. I thank all the ADMS team for their availability in providing technical help and creating the friendly atmosphere, in which it was very enjoyable to work.

In Verimag, my research benefited from the following interactions with present or (distant) past members. I thank Alexandre Donzé for involving me into what would become a first scientific publication. Dejan Nickovic collaborated formally and informally on many projects. Thank you Dejan for also inviting me to the Austrian Institute of Technology in July 2015. I am grateful to Eugene Asarin for making accessible his knowledge in various domains, and for giving detailed feedback on several early research drafts. I would also like to thank Dogan Ulus for our fruitful collaboration; proving me wrong more than once, and seeing opportunity where I was doubtful of my own solution.

In general I must acknowledge the important role of other fellow PhD candidates in improving the quality of my scientific and social life! I thank all Verimag researchers, in particular its successive directors Nicolas Halbwachs and Florence Maraninchi for providing all the assistance I ever needed.

This is also for me the occasion to thank, and acknowledge the role of previous professors who introduced me to the domain of formal verification and monitoring, Arnaud Durand and Nicolas Rapin.

Science sans conscience n'est que ruine de l'âme... To finish I warmly thank all my family, who continuously supported me through this (sometimes consuming) process. Most especially to my wife Lorna, thank you for standing by my side all those years.

Abstract

This thesis is concerned with the monitoring of mixed-signal circuit simulations. In the field of hardware verification, the use of declarative property languages in combination with simulation is now standard practice. However the lack of features to specify asynchronous behavior, or the insufficient integration of verification results, makes existing assertion and measurement languages unusable in the mixed-signal setting.

We propose several theoretical and practical tools for the description and automatic monitoring of such behaviors, that feature both discrete and continuous aspects. For this we build on previous work on real-time extensions of temporal logic and regular expressions as follows.

Efficient algorithms for computing the distance from some simulation trace to temporal logic specifications are given, with complexity comparable to that of traditional monitoring. An original diagnostic procedure is provided for the systematic debugging of such traces. The monitoring of continuous behaviors is also extended to other forms of assertions based on regular expressions. These expressions form the basis of our measurement language, which describes conjointly a measure and the patterns over which that measure should be taken.

We finish by showing how measurements currently implemented in analog circuits simulators can be ported to digital descriptions, this way extending structured verification methodologies used for digital designs toward mixed-signal.

Résumé

L'objet de cette thèse est le monitorage des simulations de circuit en signaux mixtes analogique / digital. Dans le domaine de la vérification de matériel, l'utilisation de langages déclaratifs pour la description de propriété, combinés avec la simulation, est dorénavant pratique courante. Cependant, par absence de fonctionalités dédiées à la spécification de comportement asynchrones, ou dû à l'intégration insuffisante des résultats de vérification, les assertions et mesures telles que couramment implantées sont presque inutilisables dans un contexte de signaux mixtes.

Nous proposons plusieurs outils théoriques et pratiques pour la formalisation et le monitorage de tels comportements, à la jonction de mondes discret et continu. Pour ce faire, nous nous appuyons sur des recherches antérieures dans le domaine des spécifications temps-réel, particulièrement en ce qui concerne la logique temporelle et les expressions régulières, et obtenons les résultats suivants.

Des algorithmes pour calculer la distance d'une trace de simulation à une propriété de logique temporelle sont décrits; ils garantissent une complexité comparable à celle du problème de monitorage correspondant. Une procédure originale pour le débogage systématique d'une telle trace relativement à une formule de logique temporelle est donnée. Le monitorage des comportements continus est ensuite étendu à d'autres formes d'assertions, basées sur les expressions régulières. Ces expressions forment la base de notre langage de description de mesures, qui permet de spécifier conjointement des intervalles temporels à observer et un certain type de mesure à appliquer.

Nous montrons enfin comment étendre le champ d'application des outils de mesure existant dans un simulateur de circuits analogiques, pour une utilisation dans les descriptions digitales. Ce faisant, nous rendons possible l'utilisation des méthodologies de vérification hiérarchiques venant du monde digital dans un contexte de signaux mixtes.

Contents

Introduction

Context

Complex engineered systems such as integrated circuits are designed using *models* that describe the behavior of their components in a mechanistic way. These models are then used to simulate various scenarios of the behavior of the system in question; designers monitor the simulated behavior in order to evaluate the system's performance and conformance to specifications. As systems tend to be larger and more complex, simulation becomes more expensive, the number of possible scenarios grows, and the criteria according to which behaviors are evaluated also become more complex and less intuitive. This part of the design process, called *verification*, becomes costly and error-prone, demanding the designer or verification engineer to do many long simulations and check the results against complex requirements.

This thesis is concerned with making the monitoring of simulations more efficient and reliable, as automatic as possible. It is centered around formalisms for specifying requirements, expected properties and performance measures that can be the basis for automatic monitoring, liberating the engineers from these tedious tasks. A large part of the thesis consists of novel theoretical and algorithmic contributions, related to formalisms based on temporal logic and regular expressions. These results are closer to the 'R' side of the R&D pipeline. The thesis also has a concrete applicative ingredient, namely, the implementation of such ideas in the context of existing industrial tools in the domain of electronic design automation (EDA) and more specifically for the design of analog and mixed-signal (AMS) circuits.

In general, *mixed-signal* verification of integrated circuits is an increasing source of challenges. This is due to the vastly different concerns, cultures, and modes of operation of digital and analog parts of the design and verification effort. As a result the amount of time spent on mixed-signal verification in a design project is becoming longer, with design errors escaping verification due to incompatible tools and approaches. To some extent, structured digital methodologies can be applied on mixed-signal circuits, by integrating analog verification results. However the further automation of mixed-signal verification, especially regarding mixed behaviors, requires novel ideas.

This thesis was conducted under the CIFRE framework in collaboration with Mentor Graphics, a leading software provider in EDA. In particular the work has taken place as part of the team in charge of developing Questa® ADMS$^{\text{TM}}$, a mixed-signal simulator. The collaboration gave invaluable insight into the domain, and also inspired several research directions.

Overview

Monitoring The verification of systems by means of simulation involves the checking of each trace for correctness. In practice this monitoring activity can be entirely manual, proceeding by simple visual inspection of simulation traces in some waveform viewer. However for systems over a certain complexity, the verification effort always requires some automation. This can be achieved in one of several ways. For electronic circuits, dedicated *monitors* can be programmed as additional components. These monitors are integrated along with design blocks, or other verification components in the simulated testbench. In the case of analog monitors there are specific concerns of accuracy and repeatability, which we attempt to address. Another possibility is to use dedicated measurement and assertion languages. These high level languages are compiled separately into software monitors, that operate transparently along the simulation.

Declarative Languages Programming monitors that check simulation results can be a repetitive and error-prone task for designers and verification engineers. The use of declarative languages that are high-level and concise can help overcome such issues. Digital assertion languages enable using temporal logic and regular expressions, separately or in combination, to describe the system's sequential behavior. These assertions are compiled into procedural monitors by the simulation environment, and their violations are reported alongside other simulation results. In addition to automating the monitoring of properties, assertions enable further analysis such as marking the simulation trace for time intervals where some fault occurs. We provide extensions to the digital assertion framework towards the mixed-signal domain.

Continuous-Time Properties In the mixed-signal setting, we are primarily interested in the time-domain behavior of circuits. While in this regard digital circuits are mainly synchronous, analog circuits are mainly asynchronous. However after quantizing analog signals according to some thresholds, time-domain behaviors can usually be expressed as a combination of sequential behavior and timing constraints. The theory of real-time systems modeling and specification led to the development of specification languages based on temporal logic and regular expressions enhancing formal language theory with real-time aspects. Such extensions are praticularly relevant in our context. Real-time specifications do not however easily translate to effective, computational descriptions. We give algorithms for the monitoring of real-time specification languages.

Quantitative Aspects Some properties of systems called *specifications* are evaluated on a given simulation as being true or false, while other properties called *measures* yield real values. A measure can form a specification when asserted to lie in some range of values, as in "the average value of x is less than 2". Conversely a specification is turned into a measure by considering the distance from the given simulation trace to satisfying it, or violating it. This distance is also called the *robustness* of the simulation trace relative to the property. We give efficient algorithms for evaluating this distance, in the case of temporal logic specifications. In general the formalization of measures does not need to rely on strong theoretical notions, as digital assertions do with formal language theory.

Measurement languages are used routinely in analog circuits verification, and provide similar benefits to those obtained through the use of digital assertions. In a mixed-signal setting it makes sense to only measure some behavior over periods of time the circuit operates in some mode, that we in turn detect according to a sequence of events. We show how measures can be formalized this way, enabling further automation of mixed-signal verification.

Interaction with Simulation The monitoring process takes place either during simulation, we say *online*, or after the simulation, we say *offline*. Offline monitoring is done by storing all the signals associated with some variable appearing in the property under consideration. Having the option to process these signals moving back and forth in time, despite causing an overhead in memory consumption often simplifies treatment. Online monitoring offers a lot of possibilities in terms of interaction with the simulation. For instance a simulation may be stopped as soon as some assertion is violated, saving precious time for designers. Measurements performed in an online fashion can also take part in the stimulus generation process, improving the coverage or driving the design towards erroneous behavior. We give a framework to generalize the implementation of analog measurements into digital testbenches in an online fashion.

Contributions

Monitoring Timed Behaviors Timed Regular Expressions (TRE) [25] are a convenient specification language for continuous-time behaviors. We extend the monitoring of continuous-time behaviors to specifications written using TRE. Our monitoring approach is similar in spirit to that of [83] for Metric Temporal Logic (MTL) [73]. While for temporal logic the domain to explore is the set of times at which the formula is satisfied, for regular expressions the domain to explore is the set of start time and end time pairs at which the expression is matched. We show that this set of time pairs can be decomposed into *zones*, simple convex sets used in timed systems formal verification. Results appear here as published in [110], but with additional focus on monitoring and integration with temporal logic. Additional results, namely the extension of this work to *online* monitoring, appear in [111].

Robustness of Continuous Behaviors The specification language Signal Temporal Logic (STL) [83], designed for mixed-signal behaviors, can be supplemented with a robustness indicator [53]. An important application of this robustness indicator is its use as a continuous variable that can be subject to optimization; this allows to detect bugs by varying simulation parameters towards minimizing the robustness value as in [67]. We develop new algorithms for the *robust monitoring* of Signal Temporal Logic (STL) based on a piecewise linear representation of signals. Among other results we use an algorithm described in [79] to guarantee a complexity comparable to that of the non-robust monitoring, and in particular linear in the length of the trace. This work was first exposed in [45] considering the case of (continuous) real signals; here we consider the more general case of traces with both Boolean and real signals.

Diagnostics of Timed Behaviors Metric Temporal Logic has been the object of many theoretical investigations, and also had some applicative success with its extension to real signals STL. However the continuous-time interpretation of MTL is sometimes considered not intuitive by verification practitioners. To facilitate further adoption of MTL and its variants we investigate the diagnostics problem. The method that we propose makes it possible to debug a trace relative some specification, by finding a (minimal) set of segments of that trace sufficient to cause the violation of the specification. Solving this problem gives new insight into causality in continuous-time behaviors, with the notion of *temporal implicants*. This research also appears in [56]; it is presented here focusing on continuous-time traces, a characteristic specific to mixed-signal behaviors.

Declarative Measurements Measurements as defined in analog simulators provide a convenient way to extract performance indicators. For mixed-signal behaviors we found it advantageous to separate the measure itself from the description of time periods over which that measure takes place. We developed following this principle a declarative measurement language based on Signal Regular Expressions (SRE), an extension of TRE to real signals for this purpose, and basic continuous measures such as duration, minimum, or maximum of some signal. Our language is particularly applicable to mixed-signal simulations, for which measure time periods tend to be delimited by sequences of events corresponding discrete mode changes. This work initially appeared in [57], and is reproduced here using a simpler yet more expressive formalism.

Testbench Measurements A notable difficulty in mixed-signal verification is that structured verification methodologies rely on a digital testbench, while measurements are performed using simulator command located elsewhere, in analog descriptions and simulation command files. We propose to make the functionality of these commands available in the form of system tasks, modules and classes in hardware description (or verification) languages, allowing to access the same algorithms. We use a uniform representation of input, output, and control signals in a measurement, and obtain the basis of a reusable library of measurement for mixed-signal verification.

Organization

The thesis is organized as follows. Chapter 1 describes the context, motivation and state of the art in the mixed-signal verification domain. Chapters 2–3 lay down the foundations of our approach. In particular Chapter 2 demonstrates how discrete and continuous signals, sampling clocks, and timing checks can be integrated in the same assertion framework, while Chapter 3 exposes a monitoring algorithm for metric temporal logic that we later extend. Chapters 4–7 present research contributions, namely diagnosing metric temporal logic specifications and measuring their robustness, monitoring timed regular expressions and measuring continuous behaviors using such expressions. Readers who are not interested in the specific application domain can restrict themselves to these chapters while those interested primarily in AMS verification can read them lightly. Chapter 8 returns to the applicative domain and demonstrates how measurements defined by analog simulators can be brought into a digital testbench.

Notation

We use the following notational conventions. Lower-case letters typically stand for numbers $(a, b, \ldots)$ or functions $(f, g, \ldots)$. Upper-case letters stand for sets $(A, B, \ldots)$, and calligraphic letters stand for sets of sets $(\mathscr{A}, \mathscr{B}, \ldots)$. As customary we denote by $\mathbb{N}$, and $\mathbb{R}$ the set of natural, and real numbers respectively. We use similar notation for sets $(\mathbb{A}, \mathbb{B}, \ldots)$ that are fixed throughout a section or chapter.

The set of functions from A to B is denoted B^A. The powerset (set of subsets) of A is denoted 2^A as usual. We denote by $\cup\mathscr{A}$ the union of $\mathscr{A}$, with by definition $\cup\mathscr{A} = \bigcup_{A \in \mathscr{A}} A$. Truth values are represented using Boolean numbers 0 and 1. Functions from A to $\{0, 1\}$ are called *predicates* over A. Predicates P over A are identified with subsets of A, that is, we do not distinguish $\{0, 1\}^A$ from 2^A, and for some $a \in A$ we write equivalently $P(a) = 1$ in place of $a \in P$. Similarly, binary relations over A are identified with predicates over $A^2 = A \times A$, equivalently with subsets of A^2. For function f continuous at $a \in \mathbb{R}$ we let $f(a^+)$ and $f(a^-)$ stand for the right-limit and left-limit of f at a, respectively.

A *valuation* is a mapping of variable symbols to some value domain. Variable symbols are lower-case letters $(p, q, \ldots)$, and valuations are written using bold letters $(\boldsymbol{u}, \boldsymbol{v}, \ldots)$. We denote $q_{\boldsymbol{v}}$ the value of variable q according to valuation $\boldsymbol{v}$. Formal statements such as formulas and expressions are built from variable symbols and operators according to some grammar. We use Greek letters $(\varphi, \psi, \ldots)$ to denote such formal statements and associated syntactic categories.

Algorithms are written in pseudo-code using lower-case letters for built-in routines, small capitals for user-defined routines, and bold letters for control flow instructions. We write mathematical symbols directly in pseudo-code when the implementation of the operation they denote is trivial or discussed elsewhere. Event-driven concurrent procedures are written using a syntax loosely based on Verilog, and their implementation discussed using Verilog constructs. This is purely conventional and does not intend to restrict the implementability to one particular language.

1

Mixed-Signal Verification

In this chapter we present the applicative context of this thesis in Electronic Design Automation (EDA). The production of integrated circuits requires a particular phase that involves the use of *mixed-signal simulation*. The conjoint use of digital and analog models is in particular necessary to analyze the interaction between parts of the circuit implementing the two kinds of functionality. This enables modeling concisely Boolean and finite-state aspects in conjunction with some electrical effects. The simulation proceeds by synchronized execution of digital and analog simulators. Thus in a mixed-signal context all the usual analog and digital checks can performed; however the interaction of digital and analog parts creates the need for a specific verification effort. We briefly present assertions and measurements used in practice, and their limitations for the specification of mixed-signal behaviors.

1.1 Circuit Design

An increasing number of applications require a computerized system to interact with its physical environment in a non-trivial manner. In such cyber-physical systems [15], most functionality is sometimes implemented on a single integrated circuit. This trend is reflected in the notion of system on chip (SoC). In the presence of both digital and analog functionality, we talk of *mixed-signal* circuit [75]. The design of digital or mixed signal integrated circuits[1] involves a simulation technique also dubbed *mixed-signal*.

The correct operation of an integrated circuit is usually verified by means of repeated simulations. During the specification phase the system is divided into large blocks, each implementing a particular function, and such that their composition implements all the desired functionality of the system. Blocks may be designed specifically for a given system, or reused from previous systems designed either by the same manufacturer or purchased from an external provider. In general, the process of creating an integrated circuit can be decomposed into the following phases: (1) specification; (2) design; (3) verification; (4) production; (5) testing. To some extent, the specification, design, and verification phases

[1]Digital-intent circuits can also be considered as mixed-signal, when one takes into account the analog sub-circuits needed for their own operation, in particular for supplying power, or generating clock signals.

can be performed concurrently on all sub-systems. Such phases are then repeated on a larger system made of such blocks, see [60]. Verification of a sub-circuit relative to its environment typically ensures that its input-output relation conforms to some protocol or “contract”, such as a range of operation for analog circuits.

Naturally, it is preferable to perform the verification as early as possible in the development process, and it has been shown that the cost of fixing an error increases exponentially with time [61]. When an error is only observed on the end product, fixing it requires identifying where the error occurred and applying again downstream phases after the error has been fixed. The methods we propose in this thesis can be applied for the early detection of errors in mixed-signal circuit designs.

1.2 Modeling

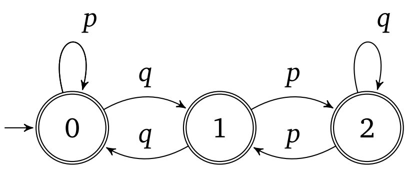
A circuit can be modeled in dedicated hardware description languages (HDL), or written in some other format specific to analog simulators. These computerized descriptions are used for simulation, but also as input to other software tools for synthesis, place and route, etc. to generate the blueprint of the silicon chip itself. In the course of the complete design cycle of some integrated circuit, a given sub-circuit can have several models. Various *levels of abstractions* can be considered: real-number, behavioral, register transfer level, gate level, or electrical, to name but a few. The use of more than one abstraction enables considering necessary trade-offs between simulation speed and accuracy. Abstract models, ignoring some details or internal aspects of the circuit behavior, can be advantageous for the performance of the simulation. Their use is essential in simulating larger portions of a design. In this scenario, the aim is to explore the circuit behavior in sufficient detail to ensure that the practical realization will conform to the specifications.

Hardware description languages such as Verilog [1] and VHDL [4] allow a representation of digital circuits close to that of concurrent computer programs. The state of the system is stored in some Boolean, integer, and floating-point variables. Such variables are updated via the traditional programming control structure with branching and looping constructs. An example of digital description appears Figure 1.1. In general, a digital description is decomposed into modules, which represent sub-circuits. Modules declare a list of ports, describing the interface, and procedural code, describing how port quantities and the internal state are updated based on events and deterministic delays. The procedural code has concurrent execution semantics. The synchronization of instructions is handled by events, which correspond to the change of state of some variable. A statement is executed on the occurrence of some event. It may in turn generate another event at a later time, according to some scheduling policy or some explicit delay.

The *de facto* standard for simulating analog circuits is known under the name of Simulation Program with Integrated Circuit Emphasis (SPICE) [92]. In this framework, the circuit description is that of a network of devices. The devices can be for example resistors, capacitors, or transistors. Their interconnection forms the circuit, and they are implicitly translated to differential equations for simulation. Alternatively the analog circuit equations can be given explicitly, using analog and mixed-signal modeling languages such as Verilog-AMS [9] and VHDL-AMS [2]. These languages enable decomposing the functionality in modules, similarly to digital description languages. Analog modules

```
always @($p$)
  if $s = 1$ then $s := 2$
  else if $s = 2$ then $s := 1$
  end if
end always
always @($q$)
  if $s = 0$ then $s := 1$
  else if $s = 1$ then $s := 0$
  end if
end always
```

(a)

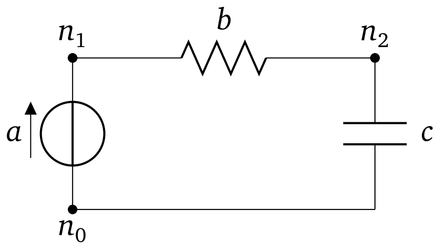


(b)

Figure 1.1: Finite Automaton: (a) Digital description; (b) Transition graph.

```
analog
  $\mathrm{V}(n_0, n_1) = a$
  $\mathrm{V}(n_2, n_1) = b\,\mathrm{I}(n_2, n_1)$
  $\mathrm{I}(n_0, n_2) = c\,\frac{\mathrm{d}\mathrm{V}(n_0, n_2)}{\mathrm{d}t}$
end analog
```

(a)



(b)

Figure 1.2: Linear electrical circuit: (a) AMS description; (b) Circuit schematic.

consist of a list of electrical ports, and procedural and equational analog statements. An example of an analog description appears in Figure 1.2. Equations describing the behavior of the model are collected across all analog statements. These languages also allow combining analog and digital statements, describing discrete and continuous dynamics conjointly in a mixed-signal model. Such models may naturally represent switched systems, continuous systems with discrete control; they may represent interface elements such as analog to digital, and digital to analog converters.

Digital descriptions can be split into two parts. One part belongs to the design under test (DUT), while the other part belongs to the *testbench*. In the DUT variables are static, they exist throughout the simulation. The testbench can be thought of as representing additional hardware blocks surrounding the DUT, or the software executing on the DUT. This later part of the code is used to generate stimuli, observe and process the response. Functionality in the testbench can be written using object oriented programming, in some hardware verification languages (HVL) such as SystemVerilog [7]. In addition to modules, testbench descriptions use constructs such as classes and interfaces. Variables in these parts of a testbench can be dynamic, created and deleted during simulation. Similar remarks can be made for analog descriptions. Part of the code, such as directed current or voltage source, or measurement functions, is not part of the design. These descriptions differ in that contrarily to the design they describe ideal behavior.

1.3 Simulation

A given DUT and environment featuring both digital and analog parts can be simulated in a software tool such as Mentor Graphics' Questa® ADMS^{TM} [11]. Mixed-signal simulators are usually formed by assembling an event-driven simulator with an electrical circuit simulator [29]. Specific functionality enables splitting circuit descriptions between analog and digital parts, potentially creating additional components not present in the circuit description when some module is this way split. In simulation, digital and analog parts interact through *boundary elements*. These elements are inserted automatically at the interface between digital and analog parts. A digital to analog boundary element can be realized as a digitally controlled source (possibly with load); an analog to digital boundary element may be implemented as a quantizer (possibly with hysteresis).

The event-driven simulator compiles digital HDL instructions and executes the resulting program in a similar fashion to that for software environments. Additional features of hardware models are taken into account. The scheduling of events follows a standardized scheme as, for example, with Verilog [1]. Hardware descriptions allow for concurrent events, which can be executed in arbitrary order. This may create race conditions, in which the update of two quantities can happen in a non-deterministic order.

The electrical circuit simulator gathers equations from the given description, and provides a numerical solution up to the desired precision [93]. Such equations are derived from *devices*, and *connectivity*. Device equations are given explicitly in a model. Connectivity equations are determined according to Kirchhoff's current and voltage laws at each circuit node. The system of equations obtained is discretized according to a numerical integration scheme, such as Backward Euler, or Gear [59]. Discrete equations are solved at each time step via iterative methods such as Newton-Raphson.

A mixed-signal simulator alternates analog and digital *time steps*, in which analog and digital quantities are updated for after some given amount of time has elapsed. The size of digital time steps is determined by events scheduled according to the execution of digital modules, with fixed delays that are multiple of some time precision. The size of analog time steps depends on the numerical stability of the equations to solve, the estimated accuracy of the current solution, and user-provided thresholds for which the times of crossing need to be precisely detected. In particular, a different set of equations can be used when quantity x is below, and when it is above some threshold c, due to analog modeling or mixed-signal interaction. This can be indicated to the simulator using *threshold crossing* events, that we may denote $\uparrow(x \geq c)$ for instance, whose effect is to force the simulator to make a step at the time where some quantity x crosses a predefined threshold c, here crossing it upward.

In this thesis we only consider *time-domain* properties; in a purely analog setting, other types of properties such as frequency-domain can be considered. The outcome of a time-domain simulation is a *trace* recording the evolution over time of digital and analog quantities of interest. Digital quantities are considered piecewise constant, following event-driven semantics. Analog quantities are considered piecewise polynomial. The interpolation scheme (naturally associated with the numerical integration scheme) may vary along the simulation – potentially requiring polynomials of degree more than one. However more samples can be added to the trace in order to ensure that in offline analysis, linear interpolation is always sufficient.

1.4 Verification

The verification of mixed-signal circuits often proceeds with purely digital and purely analog verification methods, based on what type of property is observed. Let us first introduce the verification practice in each separate domain, and then discuss mixed-signal verification specifically.

1.4.1 Digital

Simulation-based (dynamic) verification and formal (static) verification of digital circuits are well-established in industrial practice [76]. Formal verification of digital circuits was initially based on theorem proving [33], and now mostly relies on *model checking* [35]. In the model checking activity, the system model is checked against some formal statement or property, also called an *assertion*. Formal methods are limited in application to (critical) sub-circuits with manageable size, the mainstream approach to verification remains simulation-based. Like formal verification, simulation-based verification can make use of formal property languages. The objective here is to check that all simulations conform to the specification. The use of formal property languages in the perspective of checking simulations is known as *assertion-based verification*.

In order to ensure that most or all aspects of the circuit's functionality are exercised in simulation, the generation of stimuli to the circuit can be based on coverage indicators. We talk in that case of *coverage-driven verification*, and the use of randomization in that perspective leads to the notion of *constrained-random* simulation. Due to the time investment needed to develop this kind of testbench, it should be as reusable as possible. For this one may use object-oriented programming constructs. Combining all such techniques can be done using the Universal Verification Methodology (UVM), developed in collaboration between the major electronic design automation suppliers [8]. This effort resulted in a set of SystemVerilog classes to easily implement verification techniques in a digital testbench.

Checking that the DUT reacts correctly to the given stimuli can be done by implementing *monitors* directly as modules in HDL or classes in HVL. An orthogonal approach is to use assertions [39]. In hardware design, an assertion is a piece of code specifying a sequential property, ranging across several clock cycles. The two principal assertion languages for digital circuit design are called SystemVerilog Assertions (SVA) [1] and Property Specification Language (PSL) [6]. The former is integrated in the hardware description language SystemVerilog, while the latter may be used together with any hardware description language, according to simulator support.

1.4.2 Analog

Analog verification proceeds by repeated simulations under varying parameter values, with the aim of exhibiting the quality of a single behavior. This can take several forms, with a so-called *corners* simulation exploring extreme values, Monte Carlo simulation using randomized values, or more elaborate scenarios using intermediate simulations results to guide the choice of parameters. This practice dominates the verification of small analog blocks and tends to be less present for larger blocks or at system level.

Analog verification usually features a first phase, in which meaningful quantities are extracted, according to the circuit intent. We talk of measurements, by analogy with physical measurements taking place during the testing phase – here we mean virtual measurements, applied to simulated values. The quantities extracted from the simulation include period, slew rate, duty cycle, and rise time. Generic functions such as integral, maximum, can also be used for that purpose. Measured values can be checked against bounds, or composed into more elaborate indicators. Measurement commands reside in the run management file, or may directly feature in a SPICE netlist.

Once a meaningful quantity has been extracted, the verification primarily consists in checking that it is within the correct range, also called Safe Operating Area (SOA). The notion of SOA initially applied to the analysis of voltages and currents at transistors ports, and by extension it applies to the analysis of arbitrary quantities measured in the context of a simulation. Analog simulators allow the designer to check such safety properties automatically, via dedicated commands. Such specifications can handle conditional statements based on arithmetic expressions and comparison operators, and timing relaxation in which the unsafe area may be entered for a specified maximum amount of time.

1.4.3 Mixed-Signal

Mixed-signal functionality may relate the time-domain behavior of digital signals and analog signals. This is the case for analog-to-digital or digital-to-analog converter circuits. There are established techniques to specify and evaluate the performance of this particular class of circuits. Digital signals may have lower rate of change than analog, when they act as control for the analog part. In other situations analog signals may be the ones with a lower rate of change. In such cases one part may be approximated as constant, and the verification is then repeated over many modes, or parameter values. In that sense, outside of converters, in mixed-signal circuits the temporal interaction of analog and digital signals is relatively rare.

The verification of a mixed-signal design is said to be *analog-centric*, or *digital-centric* depending on the principal intent of the design. The verification of analog circuits is less structured, and in analog-centric verification mixed-signal does not require a special treatment. On the other hand, digital circuits typically undergo a structured verification process. Support for verifying analog functionality in a digital-centric environment is crucial to the relevance of digital techniques. Some steps have been done in that direction in some commercial simulators:

- SOA violations reported alongside digital assertions and counted in coverage;
- analog quantities can be used in expressions within digital assertions;
- libraries of analog probes and sources are available in AMS languages.

With analog extensions of hardware design languages such as Verilog-AMS, it is also possible to extend the digital practice towards incorporating analog features. For example [115] and [30] propose analog monitors implemented in this fashion. A major difficulty with implementing an analog and mixed-signal testbench is that while HDLs have mixed-signal capabilities, this is not the case of major HVLs such as SystemVerilog.

A current effort from software providers is the merging of Verilog-AMS with SystemVerilog. This will facilitate the access of analog quantities from within a digital testbench, and may be a first step towards a general purpose mixed-signal description and verification language [82].

1.5 Measurements

The role of measurements is to process simulation traces in order to extract relevant quantities, or *measures*. Such measures can be obtained by taking a maximum, an average value, or computing delays and slopes of signals in a variety of ways. Analog verification heavily relies on measurements; typically the verification is a two-layer process that extracts some measures, and checks that they are within the expected range. Measurements can be written in dedicated languages such as the MEAS library initially implemented by Synopsys [3], and EXTRACT library implemented by Mentor Graphics [10]. These libraries provide statements that can appear in SPICE descriptions, or alongside the run-management part of a simulation.

In the context of measurements, a signal can be seen either as a discrete-time sequence of values, or as a continuous-time piecewise-linear / piecewise-constant function of time. In general, we may define a measurement as a mapping from signals to signals. This covers the case of continuous-time signals via interpolation, discrete-time signals, and single values seen as sequences of length one. The time component of a measure is usually defined as the instant at which the measure was extracted.

Example 1.1. *Consider the measurement* $y = \text{average}(\text{period}(p))$*, illustrated in Figure 1.3. The period is simply defined as the amount of time elapsed between consecutive rising edges of p. This measure can be taken several times over a signal p featuring more than two rising edges. The computation of* $y = \text{average}(x)$ *involves representing x as a set of discrete values, and produces another set of discrete values; typically only the last one would be of interest. We may also want to check that the period of p does not go above value* 2 *for too long, for which we would interpret signal x as continuous and could use interpolation.*

Some measurements are restricted to time periods during which the system is in a certain state. In many situations, entering or leaving a some state can be detected by simple trigger and target events, such as some signal crossing a given threshold. Measurements of this kind can be specified by the MEAS library. In other situations, the measurement can be specified using an absolute time window, for instance when a particular stimulus is applied, and the behavior is expected to occur within a fixed time window. Measurements of that kind are easily described using the EXTRACT library.

1.6 Assertions

Digital assertions enable specifying sequential behaviors. They can be written in dedicated languages PSL and SVA, which feature two main layers: (1) regular expressions, and (2) temporal logic. The regular expression layer is used to describe sequences of events, in order to detect that the design enters a given state. The temporal logic layer

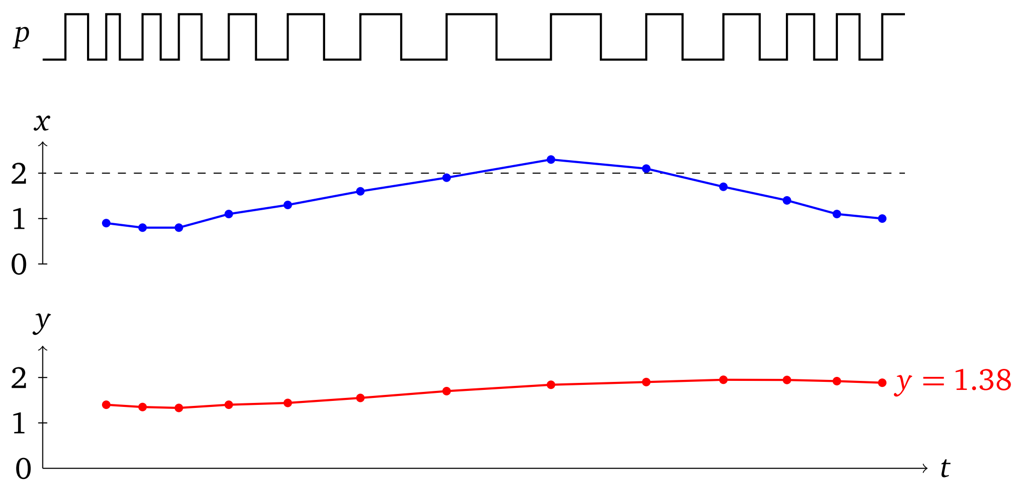


Figure 1.3: Extracting measure $x = \mathrm{period}(p)$, checking x against threshold 2, and extracting $y = \mathrm{average}(\mathrm{period}(p))$ in turn.

is used to describe invariants using operator *always* denoted $\square$, and reactive properties based on such sequences, using suffix implication denoted $\circ\!\!\rightarrow$. Typical assertions are of the form $\square\,\varphi$, with $\varphi = \rho_1 \circ\!\!\rightarrow \rho_2$ given ρ_1 and ρ_2 some regular expressions. Such an expression reads "ρ_1 is always followed by ρ_2", and its meaning is as follows: any segment of the trace that matches ρ_1 is immediately followed by another segment that matches ρ_2.

Digital circuits are synchronized by clocks, and assertions are often evaluated under the associated sampling. The sampling may be implicit, or may apply to some assertion φ in the from of some event $\uparrow r$, with the assertion written $\varphi\,@\!\uparrow r$. Properties of interest sometimes involve more than one clock, typically when describing the interaction between parts of the circuit driven by different clocks. One may specify such properties using multi-clock assertions, which feature a regular expression or temporal logic statement split between two sampling clocks. For example, one can specify a sequence of events $(\rho_1\,@\!\uparrow r_1)\cdot(\rho_2\,@\!\uparrow r_1)$ that describes a segment of the trace matching ρ_1 with sampling given by $\uparrow r_1$ and followed by another segment matching ρ_2 with sampling given by $\uparrow r_2$.

Example 1.2. *Consider the property $\varphi = (p\cdot p\cdot p\cdot p \circ\!\!\rightarrow q \vee \neg p\cdot\neg p)\,@\!\uparrow r$. It uses for sampling the rising edges of signal r. Based on this sampling it requires that each sequence of four consecutive occurrences of p should be followed either by an occurrence of q or two consecutive occurrences of $\neg p$. The monitoring of this assertion on a given simulation trace is illustrated in Figure 1.4.*

Analog assertions can be formed in one of two ways. The first way to specify analog behaviors is to use CHECKSOA instructions, dedicated to checking the safe operation of electrical circuits. Such statements allow to check if some simulated or measured quantity stays within some range (safe area), and in a more general way, for how long does it leave the designated range. This is particularly suited for checking correctness of some analog behavior of the circuit, focusing on the stable values (discarding short spikes). Safe operating area checks can also include preconditions in the form of arbitrary

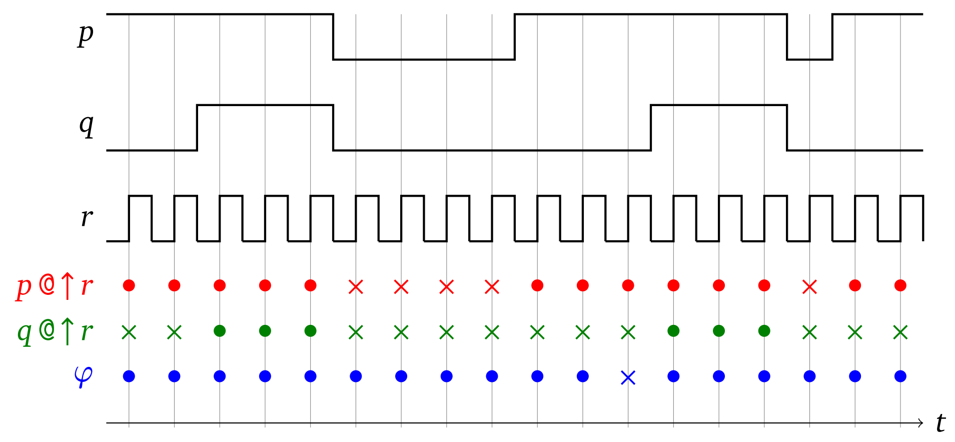


Figure 1.4: Evaluation of concurrent assertion $\varphi = (p \cdot p \cdot p \cdot p \circ\!\!\rightarrow q \vee \neg p \cdot \neg p)\,@\!\uparrow r$ on a given simulation trace.

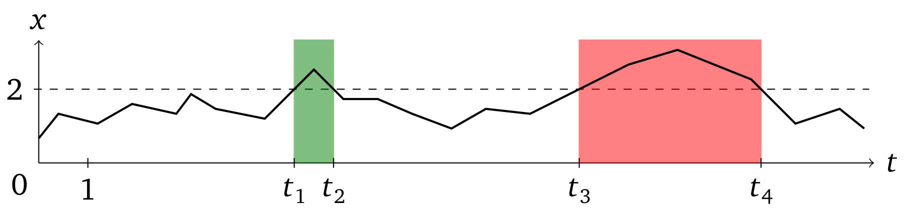


Figure 1.5: Checking safe operating area of x using a continuous time representation. A spike above 2 is safe if it lasts less than 3 time units ($t_2 - t_1 \leq 3$), but is unsafe when it lasts 3 time units or longer ($t_4 - t_3 > 3$).

propositional formulas over propositions of the form $x \leq c$ or $x < c$ for some measured quantity x and threshold c. Outside timing relaxation, automated safe operating area checks do not support the specification of sequential behaviors.

Example 1.3. *Consider the property according to which that signal x either stays less than 2, or does not stay above threshold 2 for more than 3 time units; see Figure 1.5 for an illustration. Signal x can be a primitive quantity simulated in the circuit such as the voltage at some node, or a measured quantity such as the period of some signal, as in Figure 1.3.*

The second way to specify analog behaviors is to use digital assertion languages over analog quantities. Unlike safe operating area checks, digital assertions can specify sequential behaviors. Continuous signals can be incorporated into the regular and temporal layers of digital assertions via two mechanisms. The first mechanism consists in replacing digital variables $p, q, \ldots$ by threshold propositions over analog variables $x > 1, y \leq 2, \ldots$ and using a digital sampling clock. One drawback is that glitches occurring between sampling points are ignored. This phenomenon is exemplified in Figure 1.6. The second mechanism consists in replacing the digital sampling event $\uparrow r, \downarrow s, \ldots$ by some analog event such as $\uparrow(x > 1), \downarrow(y \leq 2), \ldots$ and using simple true or false conditions. A difficulty associated with this solution is that the sampling cannot be used to measure the elapsed time, as with a periodic digital sampling clock. There one may resort to storing

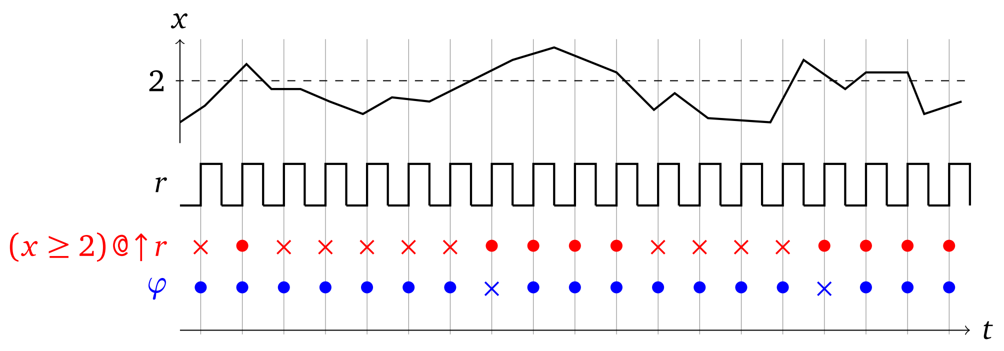


Figure 1.6: Checking safe operating area of x using a discrete time approximation $\varphi = (\neg(p \cdot p \cdot p \cdot p))\,@\!\uparrow r$. The specification appears violated at the end of the trace (four consecutive sampled values of x above 2) whereas it is marginally satisfied (x does not stay above 2 for more than 2 time units).

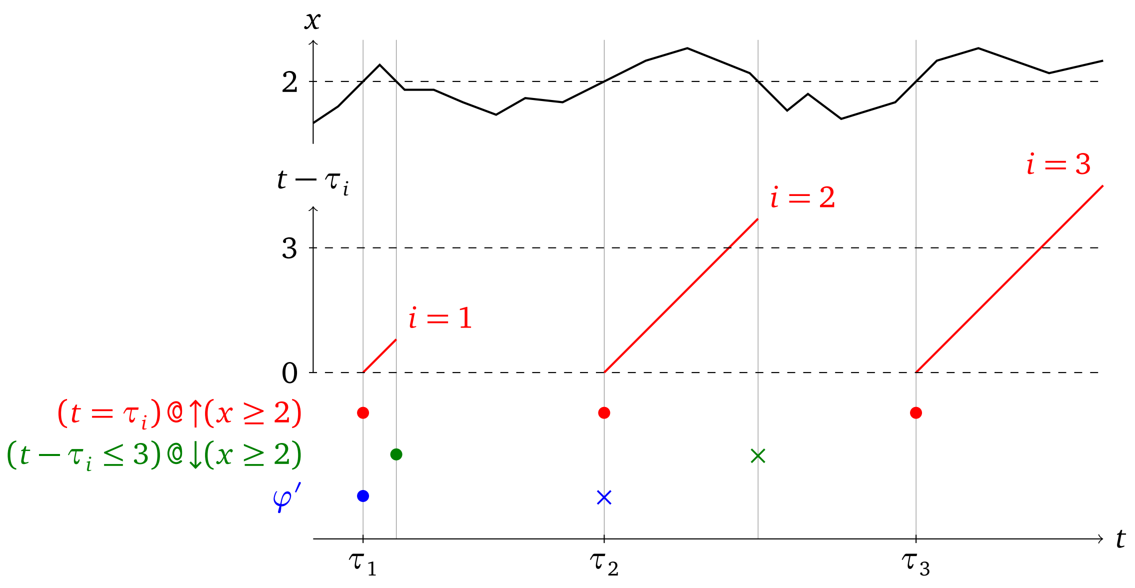


Figure 1.7: Checking safe operating area of x using analog events, local variable based assertion $\varphi' = \forall\tau\,(\tau = t)\,@\!\uparrow(x \geq 2) \circ\!\!\rightarrow (t - \tau \leq 3)\,@\!\downarrow(x \geq 2)$. The last event marking the exit of x from safe area is not matched by an event marking its return, so that the violation goes undetected.

the absolute time of events via a local variable, and computing delays by subtracting such times. The drawback of using this encoding is that the presence of some analog event cannot be asserted unless one uses the notion of *strong* and *weak* expressions as standardized in [113], which are not implemented in all commercial simulators. The default semantics is that of weak expressions, according to which the absence of clock ticks will not cause the assertion to fail as illustrated Figure 1.7. Here the clock tick is defined as the occurrence of some analog event.

2

Specifying Discrete and Continuous Properties

This chapter introduces a framework for the specification of mixed-signal behaviors, obtained by extending standard hardware assertion languages towards continuous behaviors. A system design features several variables representing quantities and their evolution in time. Not all quantities are defined at all times; we consider two models of time: discrete and continuous. The monitoring process computes additional quantities of particular interest on a given simulation: propositions, events, properties, and measures. We give formal definitions for specification languages of temporal logic and regular expressions. Along with *suffix implication* they are the backbone of hardware assertion languages. For digital behaviors, assertions feature a sampling event, based on some clock signal. We redefine the semantics of sampling operators as changing the underlying temporal domain. For analog and mixed-signal behaviors, we advocate the use of real-time constraints, along with continuous checks. We recall previously proposed extensions of temporal logic and regular expressions with such constraints. Our definitions demonstrate how these extensions naturally integrate in the existing digital framework, leading to a unified mixed-signal assertion language.

2.1 Introduction

The specification of mixed properties, involving both digital and analog behaviors, is still an active research topic. Not all mixed-signal circuits exhibit such behaviors. Often, the behavior of the circuit can be split between analog and digital parts, which do not interact in a dynamic fashion. Yet the verification of mixed-signal interaction still remains a bottleneck in the design of integrated circuits. A unified language for specifying analog and digital behaviors would have the advantage of concepts reuse. In particular it is possible to use the same operators to specify sequential aspects, those of regular expressions and temporal logic. However as seen, the nature of the temporal domain in digital and analog circuits is fundamentally different. In the case of digital behaviors, correct timing consists in a range of clock cycles. In the case of analog behavior, correct timing consists in a range of delay values. Sampling with a digital clock of fixed period, the number of clock cycles also stands for a time delay. Yet the specificity of analog behaviors however is that events arrive in an asynchronous manner. Thus using a periodic sampling clock on analog signals results in reduced accuracy. Continuous time specification languages, as with Metric Temporal Logic (MTL) [73] and Timed Regular Expressions (TRE) [25], do not suffer from this limitation, as they treat time as a real value. Another important aspect of analog behaviors is that the state of some analog circuit cannot always be taken as some signal value. The state of some analog circuit is better represented by some measure applied to continuous signals produced by the circuit. We can also add time domain measurements to our framework. Such measurements can easily be integrated to assertions by considering the values it produces as forming another signal produced by the simulation.

Using digital assertions languages such as SVA or PSL for specifying analog behaviors is already possible in some commercial simulators, see [98]. The work [63] studies to problem of integrating timing constraints in the regular expression layer of SVA. For this they mainly rely on definitions of timed regular expressions set in [25], which they adapt to the setting of digital assertions. Various aspects are considered there, and in particular giving continuous-time interpretations. Our framework retains several propositions made in [63]. The use of MTL to the specification of continuous signals was first proposed in [83]. Further applications of this work to analog and mixed-signal circuits have been proposed by the same authors, and are summarized in [84]. A similar solution is proposed in [90], where the authors study additional problems such as analog simulator synchronization and performance aspects. They include a further case study demonstrating the interest of using MTL as a base for mixed-signal specifications. In the context of temporal logic based specification of analog behaviors, [88] demonstrates the benefits of using real-time constraints over fine-grained sampling. This work also points to the interest of deeper integration of analog assertions with the simulator in order to adapt the precision of simulation according to the temporal property under consideration. The limitations of using only timing constraints and threshold predicates is also emphasized in [112]. Another directions include the use of analog measures between digital sampling points [77]. We feel that this approach and similar are only applicable in particular verification scenarios, and thus focus on the core problem of treating real-time aspects. In most cases quantitative properties of analog signals can be measured almost independently of other temporal specifications.

2.2 Simulation Traces

A simulation trace can be defined as a set of *signals*, or equivalently as a valuation of variable and statement symbols to concrete values that are a function of time. In particular we consider Boolean and real variables, and statements such as propositions, events, properties, and measures. We will always assume the (simulated) time to range over a bounded *temporal domain*. In the remainder of this section, we fix a temporal domain $\mathbb{T} = [0, d]$. For a simulation $\boldsymbol{w}$ over $\mathbb{T}$, we denote by $\boldsymbol{w}[t, t']$ its restriction to some interval of time $[t, t'] \subseteq \mathbb{T}$. We use similar notations for the restriction of some trace to an open or semi-open time interval.

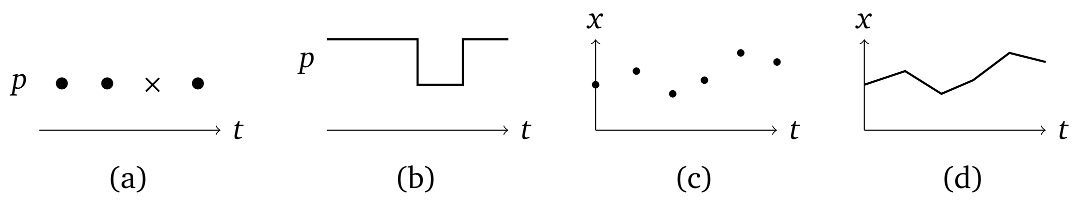


Figure 2.1: Graphical representation: (a) discrete-time Boolean signal; (b) continuous-time Boolean signal; (c) discrete-time real signal; (d) continuous-time real signal.

2.2.1 Signals

We use the term *signal* to refer to any function from the temporal domain to some value domain. Quantities appearing in a trace are defined over a specific temporal domain T. Some quantities are defined over the whole temporal domain $T = \mathbb{T}$, while other quantities are only defined over a finite set of times $T \subset \mathbb{T}$. We talk of *discrete-time* and *continuous-time* signals, respectively. Independently of the nature of its temporal domain T, a signal also has a type, according to the domain in which it takes its values. We consider two types of quantities: *Boolean* values in $\{0,1\}$ and *real* values in $\mathbb{R}$, occurring respectively in digital and analog circuits. In full rigor digital quantities can take their value in domains other than $\{0,1\}$, typically the set of values $\{0,1,\mathrm{X},\mathrm{Z}\}$ where 'X' stands for an undetermined state, and 'Z' for a high impedance state. This is not essential to our results, and we ignore this aspect.[1] We obtain four different kind of signals, according to their temporal domain and value domain as illustrated in Figure 2.1.

Definition 2.1 (Variability). *We say that a continuous-time signal is* finitely variable *when its number of discontinuities is finite. In that case the* variability *of this signal is the maximum number of its discontinuities over an arbitrary unit-length interval.*

In the rest of this thesis, we can safely assume that continuous-time Boolean and real signals, as present in simulation traces, have finite variability.[2]

2.2.2 Variables

In a specification or system description, quantities are typically defined by using abstract variables. Assume a given set of *Boolean variables* $\mathbb{P}$, and a set of *real variables* $\mathbb{X}$. Let $\boldsymbol{w}$ be a simulation trace, and let $\mathbb{T}$ stand for the temporal domain of $\boldsymbol{w}$. In the context of $\boldsymbol{w}$, a Boolean variable $p \in \mathbb{P}$ represents the function $p_w \in \{0,1\}^{\mathbb{T}}$, and a real variable $x \in \mathbb{X}$ represents the function $x_w \in \mathbb{R}^{\mathbb{T}}$. As expected, their value at time $t \in \mathbb{T}$ is denoted $p_w(t) \in \{0,1\}$ and $x_w(t) \in \mathbb{R}$, respectively. For Boolean signals we do not make the distinction between a predicate over $\mathbb{T}$ and a function $\{0,1\}^{\mathbb{T}}$, thus writing indifferently $t \in p_w$ and $p_w(t) = 1$ to denote the fact that p holds at time t over trace $\boldsymbol{w}$.

[1]Finite sets of values occuring in digital design can be emulated using powers of the set $\{0,1\}$.

[2]No reasonable simulator can produce signals with infinitely many discontinuities.

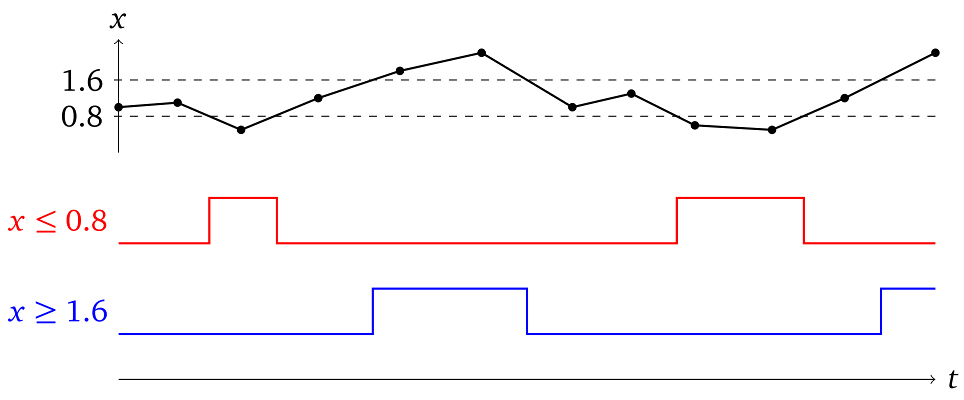


Figure 2.2: Quantization of signal x according to thresholds 0.8 and 1.2.

2.2.3 Propositions

A *proposition* is some property of the system that can be checked instantaneously. For some Boolean variable $p \in \mathbb{P}$, the fact that p is true constitutes a proposition; we also talk of Boolean variables as propositional variables. For a value of $x \in \mathbb{X}$, a typical instantaneous property is its position relative to some threshold $c \in \mathbb{R}$. We consider *threshold propositions* of the form $x \bowtie c$ where $\bowtie \in \{<, >, \leq, \geq\}$ is a comparison operator. Such propositions have a truth value denoted $[x \bowtie c]_w(t)$ at time $t \in \mathbb{T}$ according to trace $\boldsymbol{w}$. For example it holds $[x \leq 2]_w(t) = 1$ if and only if $x_w(t) \leq 2$.

The evolution of the truth value $x \bowtie c$ over time therefore constitutes a Boolean signal. Going from the real signal denoted x to the Boolean signal denoted $x \bowtie c$ for some thresholds c is referred to as *Booleanization*, or *quantization* when several thresholds are involved. Such an operation is a common preprocessing step in the analysis of analog behaviors typically represented by continuous-time real signals, which makes continuous-time Boolean signals of particular interest in this context. Figure 2.2 exemplifies the quantization of some continuous real signal x.

Propositions can be combined using Boolean operators $\vee$ and $\neg$. We obtain propositions in the general form $p \vee q$, and $\neg p$, standing for the disjunction of proposition p and proposition q, and the negation of proposition p respectively. The truth value at time t of these propositions is given by the trace $\boldsymbol{w}$, according to the following equations:

$$[p \vee q]_w(t) = \max\{p_w(t), q_w(t)\} \qquad\qquad [\neg p]_w(t) = 1 - p_w(t)$$

We consider that such formulas are readily available to higher level specifications, and sometimes assume their truth value given as part of the simulation trace.

2.2.4 Events

Now take a proposition p defined over a continuous temporal domain $\mathbb{T}$. The change of value of variable p constitutes an instantaneous property that we call an *event*. We denote $\uparrow p$ the rising edges of p, occurring when p goes from 0 to 1. In the context of simulation trace $\boldsymbol{w}$, event p has the semantics of a Boolean signal $[\uparrow p]_w$ that is true at points where p has a rising edge and false elsewhere. Formally, at any time $t \in \mathbb{T}$ we

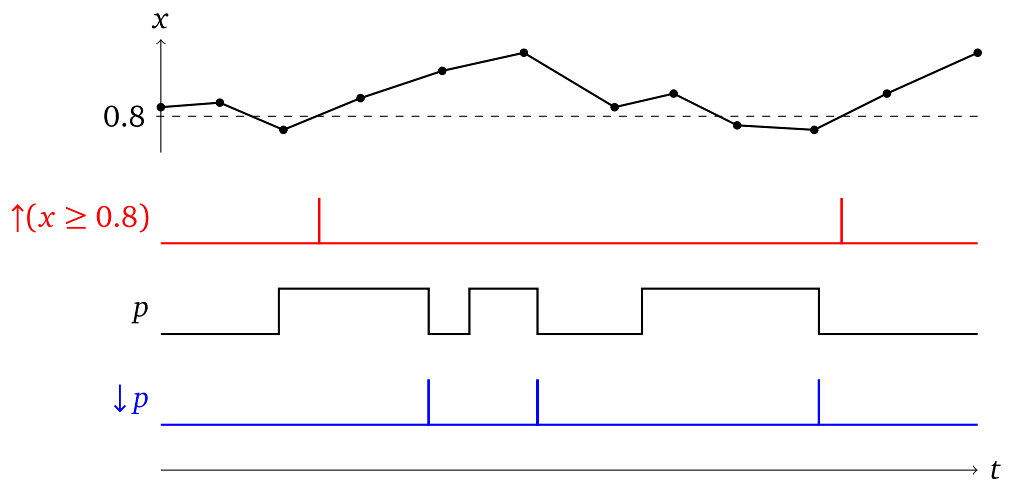


Figure 2.3: Some events associated to real variable x and Boolean variable p.

let $[\uparrow p]_w(t) = 1$ if and only if $p_w(t^-) = 0$ and $p_w(t^+) = 1$. We denote $\downarrow p$ the *falling edges* of p and denote $\updownarrow p$ the *edges* of p, defined by letting $\downarrow p = \uparrow \neg p$ and $\updownarrow p = \uparrow p \vee \downarrow p$, respectively.

An event over some Boolean variable corresponds to a discontinuity in some signal, and by finite variability hypothesis it may only occur at finitely many times. Note that we can specify events occurring on real variables using a threshold comparison, for example $\uparrow(x \geq 1)$. We also assume that an event of the form $x \bowtie c$ only occurs finitely many times in a given simulation. In analog simulation, when x represents an analog quantity, detecting such events may involve the computation of a numerical solution to find the precise time of crossing, or alternatively the times of occurrence of this event can be computed using interpolation. Examples of events appear in Figure 2.3.

2.2.5 Properties and Measures

Several quantities can be computed on a simulation trace, and in turn added to the trace itself. In general we consider two types of quantities: properties and measures. Fix $\boldsymbol{w}$ a simulation trace. A measure μ associates a *real* value with $\boldsymbol{w}$. A property φ is either true or false, and associates a *Boolean* value with $\boldsymbol{w}$. It is also interesting to monitor the evolution of the truth value of φ, or the real value of μ across some simulation as follows.

Let φ be a property. Associating a truth value $[\varphi]_w(t)$ of φ with times $t \in \mathbb{T}$ can be done by considering whether φ holds over the prefix $\boldsymbol{w}[0,t]$. This definition, which is that of a *past* property, enables monitoring the truth value of φ with every time instant based on preceding values of $\boldsymbol{w}$. The monitoring of φ as a past property may be done online, as the dependency of its value upon those of $\boldsymbol{w}$ respect some form of temporal causality. Another way to associate a truth value of φ with time t is to consider whether the property holds over the suffix $\boldsymbol{w}[t,d]$ where $d = \sup \mathbb{T}$. Such a definition would not enable in general a monitor to decide the value of φ in real time, as this truth value may depend on signal values not yet acquired. We call this of *future* property.

In the context of trace $\boldsymbol{w}$, the value of some measurement μ can be associated with some time t with some real value following similar principles. We usually denote by $[\![\mu]\!]_w(t)$ the value of μ at time t, relative to trace $\boldsymbol{w}$. This measure can then be used as additional real variable in assertions or other measurements likewise.

It is also relevant to apply a measure, or check a property for various *segments* of the trace $\boldsymbol{w}$. For convenience we represent an arbitrary segment $\boldsymbol{w}[t, t']$ of the trace for $t \leq t'$ as a pair of times $(t, t') \in \mathbb{T}^2$. The action of some proposition p holding continuously over some segments (t, t') is a typical temporal property. Similarly one may compute the real value of the measure μ for a set of segments (t, t') of the input trace. In what follows we demonstrate how properties and measures can be defined in this fashion.

2.3 Declarative Property Languages

In this section we present the declarative languages that form the basis of assertions, namely temporal logic and regular expressions. We define their semantics on both discrete and continuous temporal domains in a uniform fashion. Features specific to digital assertions, such as *sampling clocks*, can be integrated to enable the specification of discrete-time behaviors. In the other direction, *real-time constraints* can also be integrated, enabling the specification of continuous-time behaviors. We show that these two types of features can indeed be handled in the same framework.

2.3.1 Temporal Logic

Temporal logic was initially proposed in Philosophy and studied under the name of *tense logic* [102]. It was introduced into the verification of reactive systems in [101] and used in formal verification ever since. In contrast to classical logic, in temporal logic atomic propositions are not interpreted as true or false constants, but as predicates over the temporal domain. Temporal logic formulas enable asserting what is true about the current state of the system through propositional logic operators, and what is true about future or past states of the system through additional temporal operators, also called *modalities*. In this thesis we conform to the practice in digital assertions and only provide *future* modalities; symmetrical *past* modalities [81] can be considered without additional difficulty. In dynamic verification, a trace considers only one possible future and is *linear*, which is emphasized by considering Linear Temporal Logic (LTL). In formal verification, other forms of temporal logic can be considered such as Computation Tree Logic (CTL) [37], interpreted over branching time traces.

Let $\mathbb{P}$ be a set of atomic propositions. The syntax of temporal logic is given by the grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathrm{U}\, \varphi$$

Conjunction and implication can be defined as abbreviations with $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$ and $\varphi \rightarrow \psi = \neg\varphi \vee \psi$, respectively. Falsehood is by definition $\bot = \neg\top$.

Following Section 2.2, we consider that a trace $\boldsymbol{w}$ provides a valuation of atomic propositions, assigning to each $p \in \mathbb{P}$ a Boolean signal p_w over $\mathbb{T}$. This Boolean signal, or predicate, is equivalently defined as the subset of times in $\mathbb{T}$ where p holds. A trace $\boldsymbol{w}$

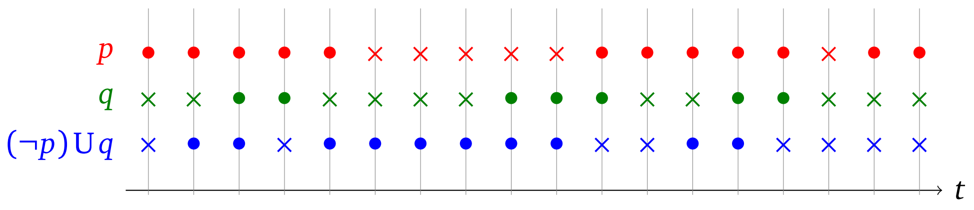


Figure 2.4: Formula $(\neg p)\,\mathrm{U}\,q$ evaluated at every time point of the trace.

satisfies φ at time t, denoted $(w,t) \models \varphi$, according to the following inductive definitions:

$$
\begin{array}{lll}
(w,t) \models \top & & \\
(w,t) \models p & \text{iff} & t \in p_w \\
(w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \,\mathrm{U}\, \varphi_2 & \text{iff} & \exists t' \in (t,\infty) \cap \mathbb{T},\, (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t,t') \cap \mathbb{T},\, (w,t'') \models \varphi_1
\end{array}
$$

We let $w \models \varphi$, and say that w satisfies φ, when $(w,0) \models \varphi$.

For discrete temporal domains[3] we define the *next* operator, denoted X, by letting $\mathrm{X}\,\varphi = \bot\,\mathrm{U}\,\varphi$. The semantics of *next* operator are such that $(w,t) \models \mathrm{X}\,\varphi$ if and only if for the smallest $t' > t$ in $\mathbb{T}$ we have $(w,t') \models \varphi$.

Note that we use *strict future* semantics for operator *until*, such that the truth value of an *until* formula is independent of the value of its arguments at the current time. The *non-strict until* operator, denoted $\tilde{\mathrm{U}}$, can be defined as the abbreviation $\varphi\,\tilde{\mathrm{U}}\,\psi$ given by $\varphi\,\tilde{\mathrm{U}}\,\psi = \psi \vee (\varphi\,\mathrm{U}\,\psi)$. This new operator has the same semantics as U, except that it allows for its right argument to occur at the current time.[4] Operators *eventually*, denoted $\Diamond$, and *always*, denoted $\Box$, are introduced as abbreviations with $\Diamond\,\varphi = \top\,\tilde{\mathrm{U}}\,\varphi$ and $\Box\,\varphi = \neg\Diamond\neg\varphi$, respectively. The *eventually* operator is such that $\Diamond\,\varphi$ holds at time t if and only if there exists $t' \geq t$ such that φ holds at time t'. The *always* operator is dual, and such that $\Box\,\varphi$ holds at time t if and only if φ holds at all times $t' \geq t$.

An example of temporal logic formula, interpreted over a discrete-time trace appears in Figure 2.4.

2.3.2 Regular Expressions

Regular expressions were proposed by Kleene for the analysis of sequences of events [70]. The so-called Kleene algebra can be seen in various mathematical contexts [74], and regular expressions have many applications, notably text processing [66]. Regular expressions may describe a phenomenon as a set of possible sequences, built using atomic expressions corresponding to the occurrence of a single event. Complex phenomena may then be described as the *union*, *intersection*, *concatenation*, or *iteration* of simpler ones.

[3]In a continuous temporal domain, there is no such a time hence in that setting $\mathrm{X}\,\varphi \iff \bot$ for all φ.

[4]Another more common definition considers a different *until* operator, strict in neither of its arguments φ and ψ, which meaning is $\psi \vee (\varphi \wedge \varphi\,\mathrm{U}\,\psi)$. Out of all variants among the choices of closed, open, or semi-open intervals in the semantics of *until*, its strict version is the most general and can express all other variants. On the contrary, *strict until* cannot in general be expressed in terms of *non-strict until*.

Considering that a sequence of events has a start time $t \in \mathbb{T}$, and end time $t' \in \mathbb{T}$, such combinatorial operations (denoted $\vee$, $\wedge$, $\cdot$, and $*$ respectively) can be understood as applied to binary relations over $\mathbb{T}$. We adopt this view in the following.

The syntax of regular expressions is given according the grammar:

$$\rho ::= a \mid \epsilon \mid \rho \vee \rho \mid \rho \wedge \rho \mid \rho \cdot \rho \mid \rho^*$$

where a is an atomic expression.

We assume that a trace $\boldsymbol{w}$ defines a valuation of atomic expression a as binary relation over $\mathbb{T}$, equivalently a predicate over $\mathbb{T}^2$. This valuation denoted a_w has the intuitive meaning that $(t, t') \in a_w$ when action a occurs between times t and t' in the simulation trace $\boldsymbol{w}$. Such atomic expressions are constructed from a given set of propositions $\mathbb{P}$. For a discrete temporal domain $\mathbb{T}$, we define atomic expression $\dot{p}$ such that $(t, t') \in \dot{p}_w$ if and only if $(t, t'] \cap \mathbb{T} = \{t'\}$ and $t' \in p_w$. Action $\dot{p}$ occurs over the segment (t, t') when t' is the next time instant after t and proposition p holds at time t'. We omit elsewhere this decoration, writing p instead of $\dot{p}$ when clear from the context.[5] For continuous temporal domain $\mathbb{T}$, we will use other types of expressions that will be defined in the following.

The symbol ϵ stands for the empty word; the atomic expression ϵ is matched by any zero-length segment (t, t). Boolean combinations are defined as expected. A concatenation requires some segment (t, t') to be split into (t, t'') matching the first argument and (t'', t') matching the second. A Kleene star allows the splitting of some segment in arbitrarily (but finitely) many segments, all of which match the expression.

Formally, a trace $\boldsymbol{w}$ *matches* ρ between times t and t', which we denote $(\boldsymbol{w}, t, t') \models\!\equiv \rho$ according to the following inductive[6] definitions:

$(\boldsymbol{w}, t, t') \models\!\equiv \epsilon$	iff	$t' = t$
$(\boldsymbol{w}, t, t') \models\!\equiv a$	iff	$(t, t') \in a_w$
$(\boldsymbol{w}, t, t') \models\!\equiv \rho_1 \vee \rho_2$	iff	$(\boldsymbol{w}, t, t') \models\!\equiv \rho_1$ or $(\boldsymbol{w}, t, t') \models\!\equiv \rho_2$
$(\boldsymbol{w}, t, t') \models\!\equiv \rho_1 \wedge \rho_2$	iff	$(\boldsymbol{w}, t, t') \models\!\equiv \rho_1$ and $(\boldsymbol{w}, t, t') \models\!\equiv \rho_2$
$(\boldsymbol{w}, t, t') \models\!\equiv \rho_1 \cdot \rho_2$	iff	$\exists t''$, $(\boldsymbol{w}, t, t'') \models\!\equiv \rho_1$ and $(\boldsymbol{w}, t'', t') \models\!\equiv \rho_2$
$(\boldsymbol{w}, t, t') \models\!\equiv \rho^*$	iff	$(\boldsymbol{w}, t, t') \models\!\equiv \epsilon$ or $(\boldsymbol{w}, t, t') \models\!\equiv \rho \cdot \rho^*$

A segment (t, t') such that $(\boldsymbol{w}, t, t') \models\!\equiv \rho$ is called a *match* of ρ (relative to $\boldsymbol{w}$).
We define the satisfaction of some regular expression ρ by trace $\boldsymbol{w}$ by letting $\boldsymbol{w} \models\!\equiv \rho$ when $(\boldsymbol{w}, 0, d) \models\!\equiv \rho$ given $[0, d]$ the temporal domain of $\boldsymbol{w}$.

2.3.3 Temporal-Regular Assertions

Regular expressions can also be used to specify relations between patterns in the trace by using them as propositions within temporal formulas. To this end we introduce the *suffix*

[5] Note however that propositional operators apply to variable symbols and not atomic expressions. In particular atomic expression $\neg p$ denotes a signal segment with a single event where p does not hold. Signal segments with more than one event are matched neither by p nor by $\neg p$.

[6] Strictly speaking, because of $*$ we should say that relation $\models\!\equiv$ is the *smallest* satisfying the following equivalences; in that case the *only if* direction becomes redundant. Otherwise the semantics of expressions such as ϵ^* is not correctly defined. One could have $(\boldsymbol{w}, t, t') \models\!\equiv \epsilon^*$ iff $t \leq t'$, or $(\boldsymbol{w}, t, t') \models\!\equiv \epsilon^*$ iff $t = t'$, since both satisfy the condition $(\boldsymbol{w}, t, t') \models\!\equiv \epsilon^*$ iff $t = t'$ or $(\boldsymbol{w}, t, t') \models\!\equiv \epsilon^*$.

implication operator, denoted $\circ\!\!\rightarrow$. Suffix implication derives from the less commonly used *suffix conjunction* operator, itself denoted $\circ$. Such operators associate a regular expression ρ and a temporal formula φ into compound temporal formulas $\rho \circ\!\!\rightarrow \varphi$, and $\rho \circ \varphi$, respectively. The formula $\rho \circ\!\!\rightarrow \varphi$ is satisfied when for all prefixes matching ρ the associated suffix satisfies φ. The formula $\rho \circ \varphi$ is satisfied when there exists one prefix matching ρ and associated suffix satisfying φ.

The syntax of temporal logic can this way be enriched with a clause $\varphi ::= \rho \circ \varphi$, where ρ is a regular expression. Given an expression ρ and formula φ, the semantics of suffix conjunction is as follows:

$$(\boldsymbol{w}, t) \models \rho \circ \varphi \qquad \text{iff} \qquad \exists t', (\boldsymbol{w}, t, t') \mathrel{|\!\!\equiv} \rho \text{ and } (\boldsymbol{w}, t') \models \varphi$$

The suffix implication operator is defined as the abbreviation $\rho \circ\!\!\rightarrow \varphi = \neg(\rho \circ \neg\varphi)$. Its semantics can be derived as follows:

$$(\boldsymbol{w}, t) \models \rho \circ\!\!\rightarrow \varphi \qquad \text{iff} \qquad \forall t', (\boldsymbol{w}, t, t') \mathrel{|\!\!\equiv} \rho \text{ and } (\boldsymbol{w}, t') \models \varphi$$

When a regular expression ρ is used directly as a formula, it implicitly stands for the formula $\rho \circ \top$. An expression ρ is satisfied at time t if and only if there exists $t' \in \mathbb{T}$ such that the segment (t, t') matches ρ. This way the assertion $\varphi = \Box \rho_1 \circ\!\!\rightarrow \rho_2$ featuring regular expressions ρ_1 and ρ_2 stands for $\Box \rho_1 \circ\!\!\rightarrow (\rho_2 \circ \top)$. Assertion φ is satisfied if and only if for all segment (t, t') matching ρ_1 there exists a segment (t', t'') matching ρ_2.

2.3.4 Sampled Properties

Hardware (digital) assertions are define over a discrete temporal domain, based on a sampling clock. This sampling clock is given in the form of an event, for example $\uparrow r$. Following the practice in hardware assertion languages, we introduce an explicit sampling operator @. Integrating this operator in temporal logic has been the object of the study [49].

Let p and r be atomic propositions. Assume given a trace $\boldsymbol{w}$, and let $T = [\uparrow r]_{\boldsymbol{w}}$ be the set of times at which r has a rising edge. The statement $p \,@\! \uparrow r$ denotes the signal $[p \,@\! \uparrow r]_{\boldsymbol{w}}$ such that $[p \,@\! \uparrow r]_{\boldsymbol{w}}(t) = [p]_{\boldsymbol{w}}(t)$ for all $t \in T$, and undefined for $t \in \mathbb{T} \setminus T$. This definition departs from simulation semantics, according to which operator @ simply means “at the next event”. In the context of mixed-signal assertions, we must make the difference between discrete-time signals and their piecewise constant interpolation.

A sampling clock can also be attached to some expression or formula φ by using the syntax $\varphi \,@\! \uparrow r$. The semantics of $@\! \uparrow r$ seen as a postfix operator can be thought of as setting the temporal domain of the property to the set of times where $\uparrow r$ occurs. Some properties involve more than one sampling clock; this occurs for instance when properties range over signals driven by different clocks.

Sampled expressions and formulas are defined by adding to regular expressions the clause $\rho ::= \rho \,@\! \uparrow r$ and to temporal logic the clause $\varphi ::= \varphi \,@\! \uparrow r$. The match and satisfaction relations are now made dependent on the sampling $T \subseteq \mathbb{T}$; these are written $\mathrel{|\!\!\equiv}_T$ and $\models_T$. Inductive definitions of $\mathrel{|\!\!\equiv}_T$ and $\models_T$ are identical to those of $\mathrel{|\!\!\equiv}$ and $\models$, respectively. The effect of the sampling in regular expression is limited to atomic expressions:

$$(\boldsymbol{w}, t, t') \models_T \dot{p} \qquad \text{iff} \qquad (t, t'] \cap T = t' \text{ and } t' \in p_{\boldsymbol{w}}$$

For temporal formulas the sampling influences semantics of the *until* operator:

$$(\boldsymbol{w}, t) \models_T \varphi \, \mathrm{U} \psi \quad \text{iff} \quad \exists t' \in (t, \infty) \cap T, (w, t') \models \psi \text{ and } \forall t'' \in (t, t') \cap T, (w, t'') \models \varphi$$

The inductive definition of other temporal and regular operators' semantics is unaffected. The role of operator $@\!\uparrow r$, in both expressions and formulas, is simply to change the sampling; its semantics are given by

$$\begin{aligned} (\boldsymbol{w}, t, t') \models_T \rho \, @\!\uparrow r & \qquad \text{iff} \qquad (\boldsymbol{w}, t, t') \models_S \rho \\ (\boldsymbol{w}, t) \models_T \varphi \, @\!\uparrow r & \qquad \text{iff} \qquad (\boldsymbol{w}, t, t') \models_S \varphi \end{aligned}$$

where $S = [\uparrow r]_{\boldsymbol{w}}$.

Let us now assume a periodic sampling T. Timing constraints can be expressed by counting the number of repetitions of atomic expressions, or by nesting the *next* operator for temporal formulas. Counting operators, denoted using exponentiation are introduced as abbreviations with $\rho^0 = \epsilon$, $\rho^i = \rho^{i-1} \cdot \rho$, $\rho^{i,i} = \rho^i$, and $\rho^{i,j} = \rho^{i,j-1} \cdot (\epsilon \vee \rho)$ for all integers $0 \leq i < j$. For temporal logic, consider the nesting of *next* operators with abbreviations $\mathrm{X}^0 \varphi = \varphi$, $\mathrm{X}^i \varphi = \mathrm{X}\mathrm{X}^{i-1} \varphi$, $\mathrm{X}^{i,i} \varphi = \mathrm{X}^i \varphi$, and $\mathrm{X}^{i,j} \varphi = \mathrm{X}^{i,j-1}(\varphi \vee \mathrm{X}\varphi)$ for all integers $0 \leq i < j$. A constrained discrete delay can be specified in temporal logic as $\mathrm{X}^{m,n} \varphi$, which means φ will hold sometime in the next m to n clock cycles. It may be specified in a regular expression with $\rho_1 \cdot \dot{\top}^{m,n} \cdot \rho_2$, which describes a segment starting with ρ_1, followed by a delay of m to n clock cycles, and finishing with ρ_2.

2.3.5 Timed Properties

To specify continuous delays, that is, bounds on real-time duration of behaviors, we consider so-called *timed properties*. Such specifications extend finite-state properties with timing constraints. They were studied extensively, primarily based on the timed automaton model [17]; we also refer the reader to [21] for an early survey of specification techniques applicable to real-time systems. In such properties, the constraints are specified as an interval of delays that are deemed acceptable between two events, or two points in time. Both temporal logic and regular expressions can be endowed with additional operators to specify these constraints, leading to specification languages of Metric Temporal Logic (MTL) [73], and Timed Regular Expressions (TRE) [25] respectively. The temporal domain of traces can be either discrete or continuous, as long as real-time information is present. In this thesis we favor continuous temporal domains as a more general case: in temporal logic discrete-time properties can be encoded in continuous-time [21]. Also, while it may be possible to approximate timed specifications using periodic-time counters, it is usually not simple as we have seen in Section 1.6. We propose to overcome such limitations or inconvenience by using timed properties as a basis for the analysis of mixed-signal assertions.

Metric Temporal Logic Timing constraints can introduced into temporal logic by considering a family of *timed until* operators, instead of the usual *until*, which we now call *untimed until*. *Timed until* operators are denoted U_I with some interval I as subscript. The resulting logic, introduced by Koymans in [73] is called Metric Temporal Logic (MTL). The syntax of MTL is the same as that of temporal logic, with the additional clause $\varphi ::= \varphi \,\mathrm{U}_I\, \varphi$, where I is an interval of $\mathbb{R}_{\geq 0}$. Let us write $\oplus$ and $\ominus$ for the Minkowski sum and difference operations, defined between arbitrary sets $R, S \subseteq \mathbb{R}$ as follows:

$$R \oplus S = \{r + s : r \in R, s \in S\} \qquad\qquad R \ominus S = \{r - s : r \in R, s \in S\}$$

In the case where R is a singleton we abuse the notation and write $r \oplus S$ and $r \ominus S$ in place of $\{r\} \oplus S$ and $\{r\} \ominus S$ respectively. The satisfaction relation $\models$ for MTL is as given in Section 2.3.1, with an additional inductive case for timed until given as follows:

$$\begin{array}{lll} (\boldsymbol{w}, t) \models \varphi_1 \,\mathrm{U}_I\, \varphi_2 & \text{iff} & \exists t' \in (t \oplus I) \cap \mathbb{T},\, (\boldsymbol{w}, t') \models \varphi_2 \text{ and} \\ & & \forall t'' \in (t, t') \cap \mathbb{T},\, (\boldsymbol{w}, t'') \models \varphi_1 \end{array}$$

Operator *until* is viewed a special case of timed until, with $\varphi \,\mathrm{U}\, \psi = \varphi \,\mathrm{U}_{(0,\infty)}\, \psi$. Metric Temporal Logic can be defined on a discrete temporal domain ($\mathbb{T}$ is finite) or on a dense temporal domain ($\mathbb{T}$ an interval of $\mathbb{R}_{\geq 0}$).

Timed Regular Expressions Timing constraints can be introduced in regular expressions by considering a family of new operators that restrict the duration of some expression to some fixed interval. Such duration constraint are denoted $\langle\rangle_I$. Given a timing interval I, this unary operator requires that the expression, which it applies to has a duration within I. The resulting specification language, introduced by Asarin, Caspi and Maler in [24], is called Timed Regular Expressions (TRE). The syntax of TRE is the same as the syntax of regular expressions, with the additional clause $\rho ::= \langle\rho\rangle_I$, where I is an interval of $\mathbb{R}_{\geq 0}$. The semantics of duration constraints are given by

$$(\boldsymbol{w}, t, t') \models\!\equiv \langle\rho\rangle_I \qquad \text{iff} \qquad t' - t \in I \text{ and } (\boldsymbol{w}, t, t') \models\!\equiv \rho$$

Timed regular expressions can be defined on a discrete or continuous temporal domain $\mathbb{T}$, similarly as with MTL. For a discrete temporal domain, one considers atomic expressions $\dot{p}$ described in Section 2.3.2. For a continuous temporal domain, we define atomic expressions $\underline{p}$ such that $(t, t') \in \underline{p}_{\boldsymbol{w}}$ if and only if $t'' \in p_{\boldsymbol{w}}$ for all $t'' \in (t, t')$. Action $\underline{p}$ occurs over the segment (t, t') when proposition p holds continuously over the open interval of time (t, t').

The specification languages of MTL and TRE, when defined over Boolean variables and threshold propositions form practical languages for the specification of continuous and mixed-signal behaviors. Using the conventions set out in this section, we can see that they can naturally integrate to hardware assertions as they exists in practice. For the purpose of allowing specifications mixing continuous-time parts along with discrete-time, we can simply further allow the constant *true* as a sampling clock. The semantics of $@\top$ have the effect of reseting the sampling to the entire temporal domain $\mathbb{T}$, enforcing a continuous-time interpretation of the subformulas or subexpressions it applies to. Whether a full integration of timed specifications with sampled ones is desirable should be the subject of further practical investigations. We did not encounter properties of mixed-signal circuits that required the full expressiveness of such an integrated language.

3

Temporal Logic Monitoring

In this chapter we present a procedure to monitor Metric Temporal Logic over continuous-time traces. This form of temporal logic adds timing constraints to the *until* operator. It has been successfully applied to the specification of analog and mixed-signal behaviors, in particular those featuring both sequential and timing aspects. As the Boolean signals that we consider typically come from quantization of analog quantities, we assume a continuous temporal domain. A temporal logic formula can be given a truth value at any point in the temporal domain; the resulting Boolean signal is known as a *satisfaction signal*. Knowing the truth value of direct subformulas for every time point is sufficient to deduce the truth value of the formula also at every time point. This is the basis of the approach due to [83] that we present in this chapter. Analyzing the complexity of resulting algorithms, we are able to show that MTL monitoring has a computation time cost linear in the size of the input trace, and at most quadratic in that of the input formula.

3.1 Introduction

Metric Temporal Logic [73], as introduced in Section 2.3.5 is a convenient language for the specification or real-time behaviors of discrete and continuous systems. Consider the simple safe operating area property of Example 1.3 in Section 1.6. It requires that signal x does not stay above threshold 2 for a period lasting more than 3 time units. This property can be specified as the following MTL formula: $\Box \Diamond_{[0,3]} x \leq 2$. It reads as follows: "always eventually $x \leq 2$ within $[0,3]$". Generally in MTL, sequential and duration constraints can be considered together. Take for example the formula $p \, \mathrm{U}_I (q \, \mathrm{U}_J (r \, \mathrm{U}_K \top))$, with proposition p, q, r and intervals I, J, K, that reads "p until q within I, until r within J, until true within K". This formula requires that p holds with duration in I, followed by q holding with duration in K, followed by r holding with duration in K. Propositional logic operators can also be used freely within a formula.

MTL has been applied to the specification of analog and mixed-signal behaviors with success in two empirical evaluations, reported in [85]. It can be readily incorporated into digital assertion framework by introducing the timed operators of MTL, while keeping the discrete-time representation of digital assertions, as presented in Chapter 2. Further integration of MTL with sampled properties can be achieved by using a continuous-time domain representation. Analogously to operator $@\uparrow p$ setting the sampling according to rising edges of p, we proposed to use the form $@\top$ in order to set the sampling to the entire, continuous temporal domain. This, along with the *timed until* operator would suffice for making MTL available in existing digital assertion languages.

The monitoring problem consists in deciding if a simulation trace satisfies a given specification. Monitoring temporal logic on discrete behaviors is a well-studied topic, with many applications in run-time verification of software systems [80]. In this chapter we consider the monitoring of Metric Temporal Logic. This problem has been studied in [108] for discrete-time traces, and [83] for continuous-time traces. The generalization of [87], with applications to model checking, consists in deciding whether an ultimately-periodic trace satisfies an MTL formula. In what follows we expose the *offline* monitoring algorithms of [83], which proceeds by evaluating the satisfaction of all subformulas over the entire temporal domain. This is done via a double induction, on the structure of the formula, and over time. For each subformula we show that the temporal domain can be decomposed into intervals in which the satisfaction status of formulas is constant. The relation between intervals of satisfaction of a formula and those of its subformulas can be expressed using usual operations such as union, complement, and Minkowski sum. We obtain a simple and efficient algorithm for monitoring MTL formulas. Online monitoring of MTL can also be achieved at the cost of small modifications to the algorithms presented here, as exposed in [86].

3.2 Metric Temporal Logic

Let us specialize the definitions of Chapter 2 as follows. We fix a set of Boolean variables (or propositions) $\mathbb{P}$, and a temporal domain $\mathbb{T} = [0, d]$ for some $d > 0$. A trace $\boldsymbol{w}$ is a valuation of propositions in $\mathbb{P}$. This valuation maps each proposition $p \in \mathbb{P}$ to a predicate $p_{\boldsymbol{w}} \in 2^{\mathbb{T}}$. Minkowski sum and difference operations are restricted to the temporal domain: for any $T \subseteq \mathbb{T}$ and interval $I \subseteq \mathbb{R}_{\geq 0}$, we let $T \oplus I = \{t + c \in \mathbb{T} : t \in T, c \in I\}$, and $T \ominus I = \{t - c \in \mathbb{T} : t \in T, c \in I\}$.

3.2.1 Syntax and Semantics

We recall from Section 2.3.5 the definition of Metric Temporal Logic. The syntax of MTL formulas is given according to the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathrm{U}_I\, \varphi$$

where $p \in \mathbb{P}$ and I is an interval of $\mathbb{R}_{\geq 0}$. *Eventually* and *always* operators can also be timed by some interval I, and denoted $\Diamond_I$ and $\Box_I$ respectively. They are given as the following abbreviations: $\Diamond_I \varphi = \top \,\mathrm{U}_I\, \varphi$ and $\Box_I \varphi = \neg \Diamond_I \neg\varphi$. We use the following priorities between MTL operators, in increasing order: $\neg$, temporal operators, $\wedge$, and $\vee$. Operator *until* gives priority to the left: the formula $\varphi \,\mathrm{U}_I\, \psi \,\mathrm{U}_J\, \gamma$ stands for $\varphi \,\mathrm{U}_I (\psi \,\mathrm{U}_J\, \gamma)$.

The semantics of MTL formulas relative to trace $\boldsymbol{w}$ are given by induction as follows:

$$\begin{array}{lll}
(\boldsymbol{w},t) \models p & \text{iff} & t \in p_{\boldsymbol{w}} \\
(\boldsymbol{w},t) \models \neg\varphi & \text{iff} & (\boldsymbol{w},t) \not\models \varphi \\
(\boldsymbol{w},t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (\boldsymbol{w},t) \models \varphi_1 \text{ or } (\boldsymbol{w},t) \models \varphi_2 \\
(\boldsymbol{w},t) \models \varphi_1 \, \mathrm{U}_I \, \varphi_2 & \text{iff} & \exists t' \in t \oplus I,\, (\boldsymbol{w},t') \models \varphi_2 \text{ and } \forall t'' \in (t,t'),\, (\boldsymbol{w},t'') \models \varphi_1
\end{array}$$

We let $\boldsymbol{w} \models \varphi$ when $(\boldsymbol{w},0) \models \varphi$. The notation $\varphi \Leftrightarrow \psi$ signifies that formulas φ and ψ agree on all traces: for all traces w, it holds that $(\boldsymbol{w},t) \models \varphi$ if and only if $(\boldsymbol{w},t) \models \psi$.

The semantics of derived operators $\Diamond_I$ and $\Box_I$ can be made explicit as follows. For *timed eventually*, it holds $(\boldsymbol{w},t) \models \Diamond_I \varphi$ if and only if $\exists t' \in t \oplus I$, $(w,t') \models \varphi$. For *timed always*, it holds $(\boldsymbol{w},t) \models \Box_I \varphi$ if and only if $\forall t' \in t \oplus I$, $(w,t') \models \varphi$.

Timing interval I may be omitted in *eventually* and *always* operators when equal to $[0,\infty)$. It may also be omitted in the *until* operator, however when equal to $(0,\infty)$. Formally, we define operator *untimed until* U as the abbreviation $\varphi \, \mathrm{U} \, \psi = \varphi \, \mathrm{U}_{(0,\infty)} \, \psi$. The *non-strict until* operator, denoted $\tilde{\mathrm{U}}$, is such that $\varphi \, \tilde{\mathrm{U}} \, \psi \Leftrightarrow \varphi \, \mathrm{U}_{[0,\infty)} \, \psi$, and can be defined by letting $\varphi \, \tilde{\mathrm{U}} \, \psi = \psi \vee (\psi \, \mathrm{U} \, \varphi)$. In such cases we fall back to the definitions of Section 2.3.1.

3.2.2 Until Rewrite

A noteworthy property of MTL that we call the *until rewrite*, is that timing aspects can be separated from sequential aspects. This observation has been used in many places [19, 47, 65], and appears in full details in [95]. The *until rewrite* rules state that *timed until* can be rewritten only using temporal operators *timed eventually* and *untimed until*. These rules greatly simplify the analysis in a number of situations.

Proposition 3.1 (Until Rewrite). *For any formulas φ, ψ and constants $0 \leq a \leq b$ the following equivalences hold:*

$$\varphi \, \mathrm{U}_{[a,b]} \, \psi \Leftrightarrow \Diamond_{[a,b]} \, \psi \wedge \varphi \, \mathrm{U}_{[a,\infty)} \, \psi \qquad\qquad \varphi \, \mathrm{U}_{[a,\infty)} \, \psi \Leftrightarrow \Box_{[0,a]} (\varphi \, \tilde{\mathrm{U}} \, \psi)$$

Such properties extend to the case of open and semi-open intervals. For any interval I of $\mathbb{R}_{\geq 0}$ it holds $\varphi \, \mathrm{U}_I \, \psi \Leftrightarrow \Diamond_I \, \psi \wedge \varphi \, \mathrm{U}_{I \cup [\sup I,\infty)} \, \psi$. For any real-value $a \geq 0$ it also holds $\varphi \, \mathrm{U}_{(a,\infty)} \, \psi \Leftrightarrow \Box_{[0,a]} (\varphi \, \mathrm{U} \, \psi)$. Thus any MTL formula can be rewritten using temporal operators of the form $\Diamond_I$ and U only.

3.2.3 Satisfaction Signals

The satisfaction signal $[\varphi]_w$ of formula φ relative to $\boldsymbol{w}$ is characterized by $[\varphi]_w(t) = 1$ if $(\boldsymbol{w},t) \models \varphi$, 0 otherwise, for all $t \in \mathbb{T}$. The offline monitoring algorithm that we present is recursive on the formula structure. As MTL semantics are inductive, knowing the satisfaction signals of direct subformulas shall be sufficient to compute the satisfaction signal of any formula. Each operator among $\vee$, $\neg$, U, and $\Diamond_I$ is now seen as processing satisfaction signals. The crucial property of each operator is that it preserves *finite variability*: if the satisfaction signal of subformulas has finitely many discontinuities, so will have the satisfaction signal of the main formula. The computation of satisfaction signals is trivial for atomic propositions. For a Boolean operation, the resulting satisfaction signal has a number of discontinuities at most the sum of that of its arguments. A formula of the form $\Diamond_{[a,b]} \varphi$ has a satisfaction signal with at most the same number of discontinuities as that of φ. Discontinuities can only appear at $t - b$ for each rising edge of φ at t, and at $t - a$ for each falling edge of φ at t. Let us now give a more detailed intuition as to why the satisfaction signal of an *until* formula also has finite variability, and how to compute it.

Firstly, for any open interval (t, t') of trace $\boldsymbol{w}$ such that the satisfaction signal of φ and ψ is constant, then the satisfaction signal of $\varphi \, \mathrm{U} \, \psi$ relative to $\boldsymbol{w}$ does not change either. Furthermore assuming φ and ψ constant over (t, t'), irrespectively of the value of φ and ψ at time t, its satisfaction status is the same at t.

Proposition 3.2 (Right-Continuity).

$$[\varphi \, \mathrm{U} \, \psi]_w(t) = [\varphi \, \mathrm{U} \, \psi]_w(t^+)$$

The value of $\varphi \, \mathrm{U} \, \psi$ in the preceding instant can be determined as follows.

Proposition 3.3 (Induction).

$$[\varphi \, \mathrm{U} \, \psi]_w(t^-) = \begin{cases} 1 & \textit{if} \quad [\varphi]_w(t^-) = 1 \textit{ and } [\psi]_w(t^-) = 1 \\ 1 & \textit{if} \quad [\varphi]_w(t^-) = 1 \textit{ and } [\psi]_w(t) = 1 \\ 0 & \textit{if} \quad [\varphi]_w(t^-) = 0 \\ [\varphi \, \mathrm{U} \, \psi]_w(t) & \textit{otherwise} \end{cases}$$

To compute the satisfaction signal we may split the temporal domain $\mathbb{T} = [0, d]$ into $[t_1, t_1], (t_1, t_2), [t_2, t_2], (t_2, t_3), \ldots [t_n, t_n]$ with $t_1 = 0$, $t_n = d$, such that $[\varphi]_w$ and $[\psi]_w$ are constant over each segment (t_i, t_{i+1}). We represent signal $f = [\varphi]_w$ by its successive values $f(0)$ and $f(t_i^-)$, $f(t_i)$ for $i = 2..n$, similarly for $g = [\psi]_w$. Following the semantics of operator *untimed until*, the signal $h = [\varphi \, \mathrm{U} \, \psi]_w$ is also constant over each such interval and representable by its values $h(0)$ and $h(t_i^-)$, $h(t_i)$ for $i = 2..n$. As the satisfaction signal of an *until* formula is right-continuous, for all $i = 2..n$ it holds $h(t_{i-1}) = h(t_i^-)$. Also at the last instant the *until* formula cannot hold due to the operator's strict future semantics, so that $h(t_n) = 0$. The computation may then proceed by backward induction, thanks to the Proposition 3.3, obtaining $h(t_i^-)$ from $f(t_i^-)$, $g(t_i^-)$, $f(t_i)$, $g(t_i)$, and $h(t_i)$.

3.3 Interval Marking

The original presentation of MTL monitoring algorithms emphasized the notion of satisfaction signal [83], as previously described. Instead of reasoning on satisfaction signals, we found it more convenient to reason purely in terms of *satisfaction sets*. Identifying predicates with Boolean signals, we now view $[\varphi]_w$ as a set of times in $\mathbb{T}$ at which the formula is satisfied.

Definition 3.4 (Satisfaction Set). *The satisfaction set of formula φ relative to trace $\boldsymbol{w}$, denoted $[\varphi]_w$, is defined as follows:*

$$[\varphi]_w = \{t \in \mathbb{T} : (\boldsymbol{w}, t) \models \varphi\}$$

In that setting we establish that finite variability input signals induce satisfaction sets with finitely many intervals.

Theorem 3.5 (Satisfaction Set Decomposition)**.** *Given a finite variability trace $\mathbf{w}$ and an MTL formula φ, the set $[\varphi]_w$ is a finite union of intervals.*

We use the *until rewrite* rule, and consider without loss of generality that formulas are constructed using operators $\vee$, $\neg$, $\Diamond_I$, and U. The proof then proceeds by induction on the formula structure.

For atomic propositions, this follows directly from the finite variability hypothesis. We have $[p]_w = p_w$ which admits a decomposition in finitely many intervals.

For disjunction, the satisfaction set of formula $\varphi \vee \psi$ is obtained from those of φ and ψ as follows:

$$[\varphi \vee \psi]_w = [\varphi]_w \cup [\psi]_w$$

For the negation operator, we have:

$$[\neg\varphi]_w = \mathbb{T} \setminus [\varphi]_w$$

For the *timed eventually* operator, it stems from definitions that:

$$[\Diamond_I \varphi]_w = [\varphi]_w \ominus I$$

As the Minkowski difference of two intervals $T \ominus I$ is an interval, and the Minkowski difference distributes over the union, it follows by induction that $[\Diamond_I \varphi]_w$ is itself a union of intervals.

For the *untimed until* $\varphi \, \mathrm{U} \, \psi$, consider the induction hypothesis according to which $[\varphi]_w$ and $[\psi]_w$ are finite unions of intervals of $\mathbb{T}$. We further assume that $[\varphi]_w$ and $[\psi]_w$ are given in the form of their minimal representations, as finite unions of *separated* intervals of $\mathbb{T}$. Each pairs of intervals of $[\varphi]_w$ and $[\psi]_w$ can be treated independently, according to the following facts. We begin by noticing that operator *until* is right-distributive relative to disjunction:

$$\varphi \, \mathrm{U}(\psi_1 \vee \psi_2) \Leftrightarrow (\varphi \, \mathrm{U} \, \psi_1) \vee (\varphi \, \mathrm{U} \, \psi_2)$$

This is straightforward from semantic definitions. We may thus assume without loss of generality, that $[\psi]_w$ is made of a single interval J. Otherwise we treat several intervals in isolation using left-distributivity. Now take T_1, T_2 two separated subsets of $\mathbb{T}$ such that $[\varphi]_w = T_1 \cup T_2$. According to the semantics of *until* it holds $t \in [\varphi \, \mathrm{U} \, \psi]_w$ if and only if $\exists t' \in [\psi]_w$, such that $t' > t$ and $\forall t'' \in (t, t')$ either $t'' \in J_1$ or $t'' \in J_2$. We notice that as J_1 and J_2 are separated, for any t' we cannot have both $(t, t') \subseteq J_1$ and $(t, t') \subseteq J_2$. We may thus assume that $[\varphi]_w$ is a single interval I, and otherwise treat several intervals separately taking the union of the result. For $[\varphi]_w = I$ and $[\varphi]_w = J$ given I, J intervals of $\mathbb{T}$, the satisfaction set $[\varphi \, \mathrm{U} \, \psi]_w$ is an interval K of $\mathbb{T}$ such that

$$K = \begin{cases} [\inf I, \sup I) \cap [0, \sup J) & \text{if } [0, \sup I] \cap J \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

We have shown that assuming $[\varphi]_w$ and $[\psi]_w$ are finite unions of intervals, $[\varphi \, \mathrm{U} \, \psi]_w$ is also a finite union of intervals. This concludes the last inductive case in the proof of Theorem 3.5.

3.4 Algorithms

The proof of Theorem 3.5 already gives us a sketch of algorithm for the monitoring of MTL. The first step of the procedure, which is implicit in the following, is to rewrite subformulas using operator U_I into formulas using $\Diamond_I$ and U following Proposition 3.1. The monitoring algorithm proceeds recursively on the formula structure, computing satisfaction signals of all its subformulas. It uses a sub-routine COMBINE to compute the satisfaction signal of a formula based on that of its direct subformulas, applying specific treatments for each MTL operator $\bullet$ among $\neg$, $\vee$, $\Diamond_I$, and U. The skeleton of the procedure is given in Algorithm 3.1. We use a representation of a satisfaction set $[\varphi]_w$ as a set of intervals $\mathscr{I}_\varphi$.

In order to implement basic operations efficiently it is beneficial to ensure a minimal representation where no intervals overlap or are adjacent. This also allows to consider sets of intervals as implicitly ordered according to their left- or right- end points indifferently. When adding an interval to a set, and in general merging two sets of intervals we shall always fuse pairs of overlapping or adjacent intervals. Let $\mathscr{J}$ and $\mathscr{K}$ be two sets of intervals. We denote $\mathscr{J} \Cup \mathscr{K}$ their merging, defined as the smallest set of intervals such that $\cup(\mathscr{J} \Cup \mathscr{K}) = \cup(\mathscr{J} \cup \mathscr{K})$. This enables an optimal representation of any given satisfaction set, in general of any set definable as finite union of intervals.

Algorithm 3.1 INTERVALS($\varphi, \boldsymbol{w}$)

```
select $\varphi$
case $p$:
    $\mathscr{I}_\varphi := \text{ATOM}(\varphi, \boldsymbol{w})$
case $\bullet\ \psi$:
    $\mathscr{I}_\psi := \text{INTERVALS}(\psi, \boldsymbol{w})$
    $\mathscr{I}_\varphi := \text{COMBINE}(\bullet, \mathscr{I}_\psi)$
case $\psi_1 \bullet \psi_2$:
    $\mathscr{I}_{\psi_1} := \text{INTERVALS}(\psi_1, \boldsymbol{w})$
    $\mathscr{I}_{\psi_2} := \text{INTERVALS}(\psi_2, \boldsymbol{w})$
    $\mathscr{I}_\varphi := \text{COMBINE}(\bullet, \mathscr{I}_{\psi_1}, \mathscr{I}_{\psi_2})$
end select
return $\mathscr{I}_\varphi$
```

Boolean Operations For disjunction, the implementation of subroutine COMBINE is straightforward. We simply merge the ordered lists of intervals $\mathscr{I}_\varphi$ and $\mathscr{I}_\psi$ into a new ordered list $\mathscr{I}_{\varphi\vee\psi} = \mathscr{I}_\varphi \Cup \mathscr{I}_\psi$. Note that this can be achieved in time linear in $|\mathscr{I}_\varphi| + |\mathscr{I}_\psi| \geq |\mathscr{I}_{\varphi\vee\psi}|$. The pseudo-code appears in Algorithm 3.2, and preserves the ordering with $|\mathscr{I}_\varphi| + |\mathscr{I}_\psi|$ comparisons by ensuring intervals are inserted in $\mathscr{I}_{\varphi\vee\psi}$ in the same order.

For negation, we need to cover the complement of $\mathscr{I}_\varphi$. Assuming $\mathscr{I}_\varphi = \{I_1, I_2, \ldots, I_n\}$ is a minimal, ordered list of intervals representing $[\varphi]_w$, we obtain an ordered list of intervals representing $\mathbb{T} \setminus [\varphi]_w$ by considering adjacent time values among 0, $\inf I_1$, $\sup I_1$, $\inf I_2$, $\sup I_2$, …, $\inf I_n$, $\sup I_n$, d. We represent $[\neg\varphi]_w$ using at most $n+1$ intervals, and the time-complexity is also linear in $|\mathscr{I}_{\neg\varphi}| \leq |\mathscr{I}_\varphi| + 1$.

Algorithm 3.2 COMBINE($\vee, \mathscr{J}, \mathscr{K}$)

```
Require: $\mathscr{J}, \mathscr{K}$ sorted
Ensure: $\mathscr{L}$ minimal, sorted
  while $\mathscr{J}, \mathscr{K} \neq \emptyset$ do
    $J :=$ first($\mathscr{J}$)
    $K :=$ first($\mathscr{K}$)
    if $J$ before $K$ then
      $\mathscr{L} := \mathscr{L} \Cup \{K\}$
      $\mathscr{K} := \mathscr{K} \setminus K$
    else
      $\mathscr{L} := \mathscr{L} \Cup \{J\}$
      $\mathscr{J} := \mathscr{J} \setminus J$
    end if
  end while
  return $\mathscr{L}$
```

Timed Eventually The case of *timed eventually* does not pose any difficulty. Given a minimal, ordered list $\mathscr{I}_\varphi = \{J_1, J_2, \ldots, J_n\}$ of intervals we get a list of intervals $\mathscr{I}_{\Diamond_I \varphi} = \{J_1 \ominus I, J_2 \ominus I, \ldots, J_n \ominus I\}$. It is ordered according to both left and right end points. Note that the Minkowski difference between two separated intervals J_i, J_{i+1}, and some interval I can create adjacent or overlapping intervals $I \ominus J_i$ and $I \ominus J_{i+1}$. In that case we may merge the resulting intervals. The pseudo-code for treating operator *timed eventually* appears in Algorithm 3.3. We have $|\mathscr{I}_{\Diamond_I \varphi}| \leq |\mathscr{I}_\varphi|$, and the time-complexity of this algorithm is linear in $|\mathscr{I}_\varphi|$.

Algorithm 3.3 COMBINE($\Diamond_I, \mathscr{J}$)

```
Require: $\mathscr{J}$ sorted
Ensure: $\mathscr{K}$ minimal, sorted
  while $\mathscr{J} \neq \emptyset$ do
    $J :=$ first($\mathscr{J}$); $\mathscr{J} := \mathscr{J} \setminus J$
    $\mathscr{K} := \mathscr{K} \Cup \{J \ominus I\}$
  end while
  return $\mathscr{K}$
```

Untimed Until For the *until* operator we proceed as follows. The COMBINE algorithm is called with two lists of intervals $\mathscr{J}$ and $\mathscr{K}$ as arguments, representing the satisfaction sets of two subformulas φ and ψ, respectively. We aim to produce a list of intervals $\mathscr{L}$ that represents the satisfaction set of $\varphi \mathbin{\mathrm{U}} \psi$. We make the additional hypotheses that $\mathscr{J}$ and $\mathscr{K}$ are minimal, i.e. all their intervals are separated, and ordered in decreasing time order. In such conditions we can directly apply the argument used in Section 3.3 to process intervals pairwise. The procedure to compute the decomposition of some *untimed until* satisfaction signal appears in Algorithm 3.4. This procedure has time-complexity linear in $|\mathscr{J}| + |\mathscr{K}| \geq |\mathscr{L}|$.

Algorithm 3.4 COMBINE(U, $\mathscr{J}$, $\mathscr{K}$)

```
Require: $\mathscr{J},\mathscr{K}$ minimal, sorted decreasing
Ensure: $\mathscr{L}$ minimal, sorted decreasing
  while $\mathscr{J} \neq \emptyset$ and $\mathscr{K} \neq \emptyset$ do
    $J := \text{first}(\mathscr{J})$
    $K := \text{first}(\mathscr{K})$
    $L := [\inf J, \sup J) \cap [0, \sup K)$
    if $L = \emptyset$ then
      $\mathscr{J} := \mathscr{J} \setminus J$
    else if $[0, \sup J] \cap K = \emptyset$ then
      $\mathscr{K} := \mathscr{K} \setminus K$
    else
      $\mathscr{L} := \mathscr{L} \Cup L$
      $\mathscr{J} := \mathscr{J} \setminus J$
    end if
  end while
```

We define the size of some formula φ, denoted $|\varphi|$, as the size of its syntactic tree. The size of some trace $|\boldsymbol{w}|$ is defined as the total number discontinuities in its Boolean signals. From all intermediate results in this section, we obtain that the monitoring of MTL can be done with a time complexity quadratic in the size of the formula (number of its subformulas) and linear in the size of the trace (number of its discontinuities). The quadratic dependency on the formula size is due to the fact that size of satisfaction signals may grow linearly with the size of the formula.

Theorem 3.6. *For any formula φ and trace $\boldsymbol{w}$, the signal $[\varphi]_w$ has a size at most $|\varphi| \cdot |\boldsymbol{w}|$.*

Proof. Let $\boldsymbol{w}$ be a trace; we prove the result by induction on φ. The property is trivial for atomic formulas. For negation, we have $|[\neg\varphi]_w| \leq |[\varphi]_w| + 1 \leq |\varphi| \cdot |\boldsymbol{w}| + 1 \leq (|\varphi| + 1) \cdot |\boldsymbol{w}| = |\neg\varphi| \cdot |\boldsymbol{w}|$, and for disjunction we have $|[\varphi \vee \psi]_w| \leq |[\varphi]_w| + |[\psi]_w| \leq |\varphi| \cdot |\boldsymbol{w}| + |\psi| \cdot |\boldsymbol{w}| = (|\varphi| + |\psi|) \cdot |\boldsymbol{w}| = |\varphi \vee \psi| \cdot |\boldsymbol{w}|$. The case of *until* operator is similar. ∎

Corollary 3.7. *The algorithm* INTERVALS$(\varphi, \boldsymbol{w})$ *has time-complexity in $O(|\varphi|^2 \cdot |\boldsymbol{w}|)$.*

Proof. The computation of the satisfaction signal of a formula ψ from its direct subformulas takes time in $c \cdot |\psi| \cdot |\boldsymbol{w}|$ for some constant c. For any subformula ψ of φ this is less than $c \cdot |\varphi| \cdot |\boldsymbol{w}|$; there are at most $|\varphi|$ subformulas to consider, so that the computation of all satisfaction signals takes time less than $c \cdot |\varphi|^2 \cdot |\boldsymbol{w}|$. ∎

4

Diagnostics

In this chapter we consider the problem of explaining, given a simulation trace and a failing MTL formula, why the trace violates the formula. We propose the definition of temporal implicants, which represent subsets of simulation traces that are sufficient to account for the violation of a formula φ, or dually the satisfaction of $\neg\varphi$. The diagnostic problem is defined as finding temporal implicants of a formula satisfied by a given trace. In order to solve this novel problem, we had to overcome mathematical issues that come from the density of the temporal domain. Our main result is an inductive diagnostic generation scheme for MTL which produces focused sub-traces sufficient to explain a given violation. A crucial ingredient of the procedure is the elimination of disjunctive operations by the introduction of selection functions similar in spirit to Skolem functions used to eliminate existential quantification. Applying this technique to analyze the result of simulation traces makes MTL readily available to the non-specialist, by making semantics of the specification explicit on a given trace. The debugging effort is reduced in general, and the fault is easier to locate whether it lies in the specification or in the simulated model.

4.1 Introduction

4.1.1 Motivation

Despite the expressive advantages of MTL and its efficient monitoring procedure described in Chapter 3, there are still many obstacles to its further adoption by electronic design practitioners and others. One particular obstacle is the use of a continuous-time representation of signals. It can be counter-intuitive to reason about the truth of some proposition or formula in between sampling points associated with a given simulation. Another obstacle is inherent in the way temporal logic describes sequential behavior through the *until* operator. This operator does not share the intuitive properties of concatenation, in particular *until* is not associative: $(p \,\mathcal{U}\, q)\,\mathcal{U}\, r$ is not equivalent to $p\,\mathcal{U}(q\,\mathcal{U}\, r)$.

Regardless of the specification language that one choses to use, the output of monitoring can be a long trace (the temporal domain features many events) with a large number of variables (the value domain has a high dimension). Analyzing the causes of some assertion violation is not intuitive. There is a need to make the result of temporal logic, and assertion checking in general, more comprehensible to users of simulation-based verification techniques. This problem is particularly acute in the setting of continuous-time properties, in which some timing violation may not stem from an observable event in the trace, but may also stem from the *absence* of some event in the trace under consideration.

4.1.2 Our Approach

Consider the temporal logic formula $\square(p \rightarrow \lozenge_{[1,2]} q)$. It requires that for any time where p holds there exists a future time, within 1 to 2 from the present, where q holds. The behavior depicted in Figure 4.1 violates this temporal property. The violation can be explained by the fact that p holds at time t, and q does not hold throughout $[t+1, t+2]$. Such a concise piece of information will increase our confidence in monitoring procedures, and promote their further acceptance.

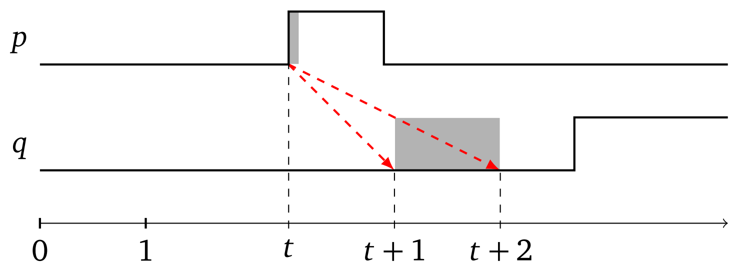


Figure 4.1: A behavior that violates $\square(p \rightarrow \lozenge_{[1,2]} q)$. Shaded area gives one possible explanation.

Finding an explanatory sub-model in the propositional case, is strongly related to the concept of *implicants* of a formula. In the setting of temporal logic, we define the notion of *temporal* implicant, as a set of signal segments that account for the violation. This set of segments is not unique, and our aim is to provide a minimal set of segments, such that strict subsets of these segments are not sufficient to explain the violation. We will see that this minimality condition is too strong, making the computation of such *prime* implicants not practical. Instead we proceed inductively on the formula structure, and we find implicants that are only locally minimal: they are prime if subformulas are atomic.

The benefits of this approach are not only in terms of computation cost. Proceeding inductively on the formula structure also enables us to produce a hierarchical diagnostic, in which the truth or falsehood of some formula can be traced back to atomic formulas via all its intermediate subformulas. In the presence of multiple causes of a fault, at each stage of the process the user can be given the responsibility to choose one alternative. This results in a fully interactive diagnostic procedure. Alternatively the choice can be automated, with the aim of reaching a concise diagnostic. We believe our notion of diagnostic is well suited for the analysis of monitoring results, and will improve the usability of temporal logic in general.

4.1.3 Related Work

The problem of understanding a counter-example by finding the reason for the failure of a temporal logic formula in the trace itself was studied in [28]. This work differs from ours in several aspects. It adopts a different notion of failures based on Halpern and Pearl causality [62] and considers only Linear Temporal Logic (LTL) and discrete temporal domains. The authors are interested in the detection of the earliest failure in a trace. In our work we provide more flexibility by means of selection functions, enabling to choose between several possible failures.

A procedure computing a *minimal debugging window* for traces that violate an MTL formula was proposed in [89]. While this work is similar to ours in spirit, the resulting analysis is coarser – it allows the analysis to focus on a smaller temporal interval where the cause of violation can be found, but does not exhibit the parts of the input signals that explain the violation. The minimal debugging window is a single time interval of the whole trace, while our diagnostic is made of several time intervals, applying to individual signals in the trace.

4.2 Propositional Foundations

Consider the problem of explaining why a formula φ is *violated* by a given execution $\boldsymbol{w}$ of some system, seen as finding the part of the execution $\boldsymbol{w}$ that causes φ to be violated. Note that through negation this is equivalent[1] to solving the dual problem of explaining why some formula is *satisfied*. We first introduce and study the problem in the simple setting of propositional logic, where models are valuations, and put classical notions in the diagnostics perspective.

4.2.1 Problem Statement

Let $\mathbb{P}$ be a *finite* set of atomic propositions. A valuation $\boldsymbol{v}$ is a mapping from $\mathbb{P}$ to $\{0,1\}$, which we write $\boldsymbol{v} \in 2^{\mathbb{P}}$. We define propositional formulas over $\mathbb{P}$ using constant $\top$ and operators $\neg$ and $\vee$ the usual way. The Boolean value of formula φ under valuation $\boldsymbol{v}$ is denoted $[\varphi]_{\boldsymbol{v}}$. We write $\boldsymbol{v} \models \varphi$, and say that $\boldsymbol{v}$ is a *model* of φ when $[\varphi]_{\boldsymbol{v}} = 1$. For φ and ψ two formulas we write $\varphi \Rightarrow \psi$ when $[\varphi]_{\boldsymbol{v}} \leq [\psi]_{\boldsymbol{v}}$ for all valuations $\boldsymbol{v}$, and $\varphi \Leftrightarrow \psi$ when $[\varphi]_{\boldsymbol{v}} = [\psi]_{\boldsymbol{v}}$ for all valuations $\boldsymbol{v}$. Note that implication ($\Rightarrow$) induces a partial order over classes of equivalent ($\Leftrightarrow$) formulas.

In some sense, we are looking for an interpolant between the valuation $\boldsymbol{v}$ and the formula φ. That is, a formula γ such that $\boldsymbol{v} \models \gamma$ and $\gamma \Rightarrow \varphi$. The least general explanation of φ relative to some trace $\boldsymbol{v} \models \varphi$ is a term representing the entire valuation $\boldsymbol{v}$. It is intuitively clear, however that we opt for explanations that are smaller and more general. Conversely φ itself is the most general explanation for $\boldsymbol{v} \models \varphi$, and is too general in some sense.

Intuitively, we would simply like to omit “don’t care” variables, those whose valuation can be changed without changing the value of the formula [103]. We aim at providing explanations that use small subsets of “do care” variables. The notion of interpolant

[1]This presupposes that the specification language has a complement operation.

does not capture precisely this intuition of an explanation. An interpolant may be of disjunctive nature, while our explanation must be unique. A better definition of an explanation requires that the explanation is also purely conjunctive, in other words an implicant of φ satisfied by $\boldsymbol{v}$. The most general explanations are then the prime implicants of φ. Let us recall the definition of implicants of a formula.

Definition 4.1 (Literals, Terms, Implicants and Prime Implicants). *A* literal *is an atomic proposition or its negation. A* term *is a conjunction of literals. An* implicant *of formula φ is a term γ such that $\gamma \Rightarrow \varphi$. A* prime implicant *of φ is an implicant maximal relative to $\Rightarrow$.*

The diagnostics problem can the be defined as follows.

Problem ((Minimal) Diagnostics). *Given a formula φ and valuation $\boldsymbol{v} \models \varphi$, find a (prime) implicant γ of φ such that $\boldsymbol{v} \models \gamma$. Given a formula φ and valuation $\boldsymbol{v} \not\models \varphi$, find a (prime) implicant γ of $\neg\varphi$ such that $\boldsymbol{v} \models \gamma$.*

4.2.2 Syntactic and Semantic Formulations

Take φ a formula, $\boldsymbol{v}$ a model of φ and γ a solution to the corresponding diagnostics problem. As $\gamma \Rightarrow \varphi$, there exists a proof of φ under hypothesis γ; a correct algorithm producing the diagnostics is implicitly constructing that proof. The more general the implicant is, the more complex the associated proof can be.

Example 4.1. *Let $\varphi = (p \wedge q) \vee (p \wedge \neg q)$ and $\boldsymbol{v} = \{p \mapsto 1, q \mapsto 0, r \mapsto 0\}$. The formulas $\alpha = p$ and $\beta = p \wedge \neg q$ are both implicants of φ, and satisfied by $\boldsymbol{v}$ with α being a prime implicant of φ.*

Let us say that $\boldsymbol{u}$ is a partial valuation of $\mathbb{P}$ when $\boldsymbol{u}$ is a valuation of some set $U \subseteq \mathbb{P}$. We now propose some semantic counter-parts of implicants, beginning with a refinement relation $\sqsubseteq$ between valuations.

Definition 4.2 (Valuation Refinement). *Let $U, V \subseteq \mathbb{P}$ be subsets of variables. For partial valuations $\boldsymbol{u} \in 2^U$ and $\boldsymbol{v} \in 2^V$ we write $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ when $U \subseteq V$ and $p_{\boldsymbol{u}} = p_{\boldsymbol{v}}$ for all $p \in U$.*

The space of partial valuations of $\mathbb{P}$ is a semi-lattice with respect to $\sqsubseteq$ with a *meet* operation denoted $\sqcap$ and a least element denoted $\mathbf{0}$. Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be some valuations of variables in U and V respectively. The valuation $\boldsymbol{u} \sqcap \boldsymbol{v}$ has domain $\{p \in U \cap V : p_{\boldsymbol{u}} = p_{\boldsymbol{v}}\}$ and value $p_{\boldsymbol{u} \sqcap \boldsymbol{v}} = p_{\boldsymbol{u}} (= p_{\boldsymbol{v}})$ where defined. The least element $\mathbf{0}$ is the nowhere-defined valuation.

One can think of a valuation $\boldsymbol{v}$ over $V \subseteq \mathbb{P}$ as a compact representation for all valuations $\boldsymbol{w}$ over $\mathbb{P}$ such that $\boldsymbol{v} \sqsubseteq \boldsymbol{w}$. A valuation $\boldsymbol{v}$ corresponds to a term $\gamma(\boldsymbol{v})$, the conjunction of literals true according to $\boldsymbol{v}$, and reciprocally any satisfiable term γ corresponds to a valuation $\boldsymbol{v}(\gamma)$, that assigns a value to variables according to the literals in γ.

Definition 4.3 (Sub-Model). *A partial valuation $\boldsymbol{v}$ is a sub-model of φ if for all valuations $\boldsymbol{w}$ over $\mathbb{P}$ such that $\boldsymbol{v} \sqsubseteq \boldsymbol{w}$ we have $\boldsymbol{w} \models \varphi$; if moreover $\boldsymbol{v}$ is minimal with respect to $\sqsubseteq$ we talk of minimal sub-model.*

A valuation $\boldsymbol{v}$ is a sub-model of φ if and only if $\gamma(\boldsymbol{v})$ is an implicant of φ. The (minimal) diagnostics problem for φ relative to $\boldsymbol{w}$ can thus be formulated equivalently as the problem of finding a (minimal) sub-model of φ contained in $\boldsymbol{w}$.

4.2.3 Practical Solution

Note that the minimal diagnostics problem is at least as hard as satisfiability, since tautologies can be recognized by the fact that they admit a unique prime implicant, the empty term $\top$. However if we relax the minimality assumption, knowing the truth value of each subformula of φ on $\boldsymbol{w}$ enables us to construct implicants γ such that $\boldsymbol{w} \models \gamma$ in a simple, top-down fashion. We construct an implicant of every formula by combination of implicants of its subformulas; these implicants are also satisfied by $\boldsymbol{w}$. Accordingly we define an *explanation* operator E_w (and its *falsification* dual F_w) that for a given formula φ returns an implicant of φ (respectively of $\neg\varphi$) which under suitable assumptions is satisfied by $\boldsymbol{w}$. The diagnostic of φ is defined as

$$D_w(\varphi) = \begin{cases} E_w(\varphi) & \text{if } \boldsymbol{w} \models \varphi \\ F_w(\varphi) & \text{otherwise} \end{cases}$$

with

$$\begin{aligned} E_w(\top) &= \top & F_w(\top) &= \bot \\ E_w(p) &= p & F_w(p) &= \neg p \\ E_w(\neg\varphi) &= F_w(\varphi) & F_w(\neg\varphi) &= E_w(\varphi) \\ E_w(\varphi_1 \vee \varphi_2) &= E_w(\xi_w(\varphi_1 \vee \varphi_2)) & F_w(\varphi_1 \vee \varphi_2) &= F_w(\varphi_1) \wedge F_w(\varphi_2) \end{aligned}$$

where ξ_w is a *selection function* satisfying $\xi_w(\varphi_1 \vee \varphi_2) \in \{\varphi_1, \varphi_2\}$. We say that ξ_w is *correct* with respect to $\boldsymbol{w}$ if for any formula $\varphi_1 \vee \varphi_2$ such that $\boldsymbol{w} \models \varphi_1 \vee \varphi_2$ it holds $\boldsymbol{w} \models \xi_w(\varphi_1 \vee \varphi_2)$. We can take for example

$$\xi_w(\varphi_1 \vee \varphi_2) = \begin{cases} \varphi_1 & \text{if } \boldsymbol{w} \models \varphi_1 \\ \varphi_2 & \text{otherwise} \end{cases}$$

This may represent the user's intent of giving priority to the left disjunct. Under the assumption that ξ_w is correct with respect to $\boldsymbol{w}$, the formula $D_w(\varphi)$ is a solution to the diagnostics problem associated with φ and $\boldsymbol{w}$.

In the case of Example 4.1, applying the procedure to φ and $\boldsymbol{w}$ yields the explanation β (only one selection function for the disjunction in φ is correct relative to $\boldsymbol{w}$).

4.3 Temporal Implicants

4.3.1 Semantics

Let us fix a temporal domain $\mathbb{T} = [0, d]$, and define a trace $\boldsymbol{w}$ as an application from $\mathbb{T} \times \mathbb{P}$ to $\{0, 1\}$, which we write $\boldsymbol{w} \in 2^{\mathbb{T}\times\mathbb{P}}$. Notations are otherwise unchanged, since up to isomorphism, $(2^{\mathbb{P}})^{\mathbb{T}} = (2^{\mathbb{T}})^{\mathbb{P}} = 2^{\mathbb{T}\times\mathbb{P}}$.

We now introduce, similarly to partial valuations in the propositional case, the notion of *sub-trace*. A sub-trace is simply a trace with domain $V \subseteq \mathbb{T} \times \mathbb{P}$. We define over sub-traces a partial order denoted $\sqsubseteq$ as follows. Sub-traces $\boldsymbol{u}$ and $\boldsymbol{v}$ with respective domains U and V verify $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ if and only if $U \subseteq V$ and $p_{\boldsymbol{u}}(t) = p_{\boldsymbol{v}}(t)$ for all $(t,p) \in U$. Given formula φ and sub-trace $\boldsymbol{v}$, we say that $\boldsymbol{v}$ is a *sub-model* of φ if $\boldsymbol{w} \models \varphi$ for all traces $\boldsymbol{w}$ such that $\boldsymbol{v} \sqsubseteq \boldsymbol{w}$. Meet operation $\sqcap$ and least element $\mathbf{0}$ are defined as with propositional valuations.

We denote $\mathbb{L} = \mathbb{P} \cup \{\neg p : p \in \mathbb{P}\}$ the set of literals formed with propositions in $\mathbb{P}$. To ensure finite representation we place similar restrictions on sub-traces as we do for full traces. A finite variability sub-trace $\boldsymbol{v}$ is such that the satisfaction set $[\ell]_{\boldsymbol{v}}$ of every literal $\ell \in \mathbb{L}$ can be written as finite union of intervals of $\mathbb{T}$. In what follows we assume all traces and sub-traces have finite variability. Unfortunately due to simple topological considerations, bounding the variability in a uniform manner does not guarantee the existence of a minimal sub-model, as we see in the following example.

Example 4.2. *The formula $\varphi = p \,\mathrm{U}\, \top$ has no minimal sub-model over the dense temporal domain $\mathbb{T} = [0,1]$. Consider the monotone sequence $(\boldsymbol{v}_i)$ of sub-models of φ with variability 1, and domain $(0, \frac{1}{i}) \times \{p\}$. The sub-trace $\bigsqcap_{i=1}^{\infty} \boldsymbol{v}_i = \mathbf{0}$ is not a sub-model of φ.*

To overcome such problems we extend the temporal domain $\mathbb{T} = [0,d]$ with some additional numbers in the set $\mathbb{T}^+ = \{t^+ : t \in [0,d)\}$ and $\mathbb{T}^- = \{t^- : t \in (0,d]\}$, letting $\mathbb{T}^\pm = \mathbb{T} \cup \mathbb{T}^+ \cup \mathbb{T}^-$. We use such *non-standard* numbers as customary, denoting $f(t^+)$ the right limit of some signal f at time t, and $f(t^-)$ its left limit. Any finite variability signal over $\mathbb{T}$ naturally extends to a signal over $\mathbb{T}^\pm$.

4.3.2 Syntax

We now introduce sentences based on (possibly infinite) conjunctions of unary predicates $p(t)$ and their negation $\neg p(t)$ for some (non-standard) real in $t \in \mathbb{T}^\pm$.

Definition 4.4 (Terms, Implicants and Prime Implicants)**.** *Temporal terms are are defined using the grammar*

$$\gamma ::= p(t) \mid \neg p(t) \mid \gamma \wedge \gamma \mid \bigwedge_{t \in T} \theta(t)$$

where $p \in \mathbb{P}$ is a propositional variable, t is a time in $\mathbb{T}^\pm$, T is a subset of $\mathbb{T}^\pm$, and θ a mapping from $\mathbb{T}^\pm$ to temporal terms. When θ is a constant term γ with one free variable t, we write the last form $\bigwedge_{t \in T} \gamma(t)$. The semantics $\models$ of temporal terms relative to a trace $\boldsymbol{w}$ are given by

$$\begin{array}{lll}
\boldsymbol{w} \models p(t) & \textit{iff} & p_{\boldsymbol{w}}(t) = 1 \\
\boldsymbol{w} \models \neg p(t) & \textit{iff} & p_{\boldsymbol{w}}(t) = 0 \\
\boldsymbol{w} \models \gamma_1 \wedge \gamma_2 & \textit{iff} & \boldsymbol{w} \models \gamma_1 \textit{ and } \boldsymbol{w} \models \gamma_2 \\
\boldsymbol{w} \models \bigwedge_{t \in T} \theta(t) & \textit{iff} & \forall t \in T, \boldsymbol{w} \models \theta(t)
\end{array}$$

An implicant *of some temporal formula φ is a temporal term γ such that $\gamma \Rightarrow \varphi$. We talk of* prime implicant *when γ is maximal with respect to $\Rightarrow$.*

The above definition of temporal terms allows arbitrary mappings θ under infinite conjunctions, which is convenient for inductive manipulations. However temporal terms can always be written in a simpler normal form as follows.

Proposition 4.5 (Normal Form). *For every temporal term γ there is an equivalent temporal term of the form $\bigwedge_{\ell \in \mathbb{L}} \bigwedge_{t \in T(\ell,\gamma)} \ell(t)$. Given an ordering of literals in $\mathbb{L}$ this normal form is unique.*

Proof. Straightforward by induction on the structure of the term γ. We collect $T(\ell,\gamma)$ from the sub-terms of γ by replacing conjunctions of terms with unions of their temporal domains, for each literal ℓ. Now, assuming that conjunctions in γ appear in the order according that over $\mathbb{L}$, we can easily show that for two distinct terms γ, γ' of the form above, there exists a literal ℓ and time t such that $t \in T(\ell,\gamma)$ while $t \notin T(\ell,\gamma')$ and thus $\gamma' \not\Rightarrow \gamma$, or vice-versa. ∎

For any term γ we will now use the notation $\bigwedge_{\ell \in \mathbb{L}} \bigwedge_{t \in T(\ell,\gamma)} \ell(t)$ for its normal form, with function T associating each literal and term with a subset of $\mathbb{T}$. Intuitively $T(\ell,\gamma)$ is the part the temporal domain where implicant γ states that ℓ should hold.

It is clear that normal form temporal terms are isomorphic to sub-traces over $\mathbb{T}^{\pm}$. Notably the relation $\Rightarrow$ defines a partial order over normal form terms. Given arbitrary terms α and β, one can check that it holds $\alpha \Rightarrow \beta$ if and only if $T(\ell,\beta) \subseteq T(\ell,\alpha)$ for all $\ell \in \mathbb{L}$.

4.3.3 Minimality

We have seen in Example 4.2 that some formulas impose conditions on limit values of satisfaction signals. It is clear that introducing non-standard reals will address the problem occurring in Example 4.2, and related problems in which finitely many limit conditions are involved. Due to density of the domain, some formula can place an infinite number of such conditions. However as the temporal domain is bounded, we still obtain the existence of at least one prime implicant for every satisfiable formula.

Theorem 4.6 (Existence of Prime Implicants). *For any formula φ and trace $\boldsymbol{w}$ such that $\boldsymbol{w} \models \varphi$ there exists a prime implicant γ of φ over $\mathbb{T}^{\pm}$ such that $\boldsymbol{w} \models \gamma$.*

This result crucially relies on two assumptions on the input trace $\boldsymbol{w}$: (1) the temporal domain $\mathbb{T}$ of $\boldsymbol{w}$ is bounded; (2) at every point in $\mathbb{T}$ the left and right limits of $\boldsymbol{w}$ are defined. Recall that we use finite variability semantics over $\mathbb{T} = [0, d]$, which clearly entails point (2). We will use the boundedness of the temporal domain to apply Bolzano-Weierstrass Theorem, in order to systematically eliminate potential counter-examples in the form of Example 4.2. We state below this famous theorem, in its simplest form. Let (r_i) and (s_i) be two infinite sequences, indexed by natural numbers. Recall that (s_i) is a *subsequence* of (r_i) if there exists a strictly increasing sequence of natural numbers (k_i) such that $s_i = r_{k_i}$ for all i.

Bolzano-Weierstrass Theorem. *Any bounded sequence of real values has a monotone and convergent subsequence.*

In order resolve issues with partial ordering of implicants, we will exercise the axiom of choice under the guise of the following well-known statement.

Zorn's Lemma. *Any partially ordered set, containing upper bounds for each of its totally ordered subsets, contains at least one maximal element.*

We are now ready to establish the main result of this section.

Proof of Theorem 4.6. Let us denote by Γ the set of implicants γ of φ such that $\boldsymbol{w} \models \gamma$, that we assume in normal form. The trace $\boldsymbol{w}$ seen as a temporal term is itself an implicant of φ. This gives us $\Gamma \neq \emptyset$. We now demonstrate the existence of a maximal element of Γ relative to $\Rightarrow$ by application of Zorn's Lemma as follows. Consider Δ an arbitrary totally ordered subset of Γ. First we can see that Δ is bounded by the temporal term α such that $T(\ell,\alpha) = \bigcap_{\gamma\in\Delta} \mathrm{cl}(T(\ell,\gamma))$, where $\mathrm{cl}(T(\ell,\gamma))$ denotes the closure of $T(\ell,\gamma)$ in $\mathbb{T}^{\pm}$. This does not pose any difficulty. We then show that moreover $\alpha \in \Gamma$, so that α is indeed an upper bound of Δ in Γ. For that we need to show that $\boldsymbol{w} \models \alpha$ which is trivial, and that $\alpha \Rightarrow \varphi$. To demonstrate the latter fact we consider an arbitrary model $\boldsymbol{v}$ of α and prove the existence of some $\gamma \in \Delta$ such that $\boldsymbol{v} \models \gamma$. As $\Delta \subseteq \Gamma$ this will in turn grant $\boldsymbol{v} \models \varphi$, by definition of Γ.

Let $\boldsymbol{v} \models \alpha$. Assume, in search of a contradiction that $\boldsymbol{v} \not\models \gamma$ for all $\gamma \in \Delta$. For each $\gamma \in \Delta$ there exists $\ell \in \mathbb{L}$ and $t \in T(\ell,\gamma)$ such that $[\ell]_{\boldsymbol{v}}(t) = 0$. We may construct a sequence (γ_i, ℓ_i, t_i) of $\Delta \times \mathbb{L} \times \mathbb{T}^{\pm}$ such that $t_i \in T(\ell_i,\gamma)$ and $[\ell_i]_{\boldsymbol{v}}(t_i) = 0$ for all $i \in \mathbb{N}$, and such that (γ_i) is monotone and diverging, that is $\gamma_i \Rightarrow \gamma_j$ if $i \leq j$, and for all $\gamma \in \Delta$ there exists $i \in \mathbb{N}$ such that $\gamma \Rightarrow \gamma_i$. We take $s_i \in \mathbb{T}$ the standard part of t_i, that is $t_i \in \{s_i^+, s_i^-, s_i\}$. As $\mathbb{L}$ is finite, we can safely assume that the sequence (ℓ_i) is constant. As $\mathbb{T}$ is bounded, by Bolzano-Weierstrass Theorem we may in turn assume that the sequence (s_i) is monotone and convergent, an assumption that we extend to (t_i). Let us write ℓ the value of (ℓ_i), and t the limit of (t_i). As (γ_i) is monotone, the subsequence of times $(t_j)_{j\geq i}$ has all its values in $T(\ell,\gamma_i,)$, so that $t \in \mathrm{cl}(T(\ell,\gamma_i))$. In particular $t \in \bigcap_{i\in\mathbb{N}} \mathrm{cl}(T(\ell,\gamma_i)) = \bigcap_{\gamma\in\Delta} \mathrm{cl}(T(\ell,\gamma)) = T(\ell,\alpha)$ given that (γ_i) is diverging. Then $t \in T(\ell,\alpha)$ yields $[\ell]_{\boldsymbol{v}}(t) = 1$. By finite variability of $\boldsymbol{v}$, as (t_i) converges to t there exists i such that $[\ell]_{\boldsymbol{v}}(t_i) = 1$. Yet $[\ell]_{\boldsymbol{v}}(t_i) = 0$ by hypothesis. Contradiction! Therefore there exists $\gamma \in \Delta$ such that $\boldsymbol{v} \models \gamma$. It follows that $\boldsymbol{v} \models \varphi$.

This holds for every model $\boldsymbol{v}$ of α, therefore $\alpha \Rightarrow \varphi$. Implicant α is a maximal element of Δ. This holds for every totally ordered subset of Γ. By Zorn's Lemma the set Γ has a maximal element relative to $\Rightarrow$, in other words formula φ has a prime implicant satisfied by trace $\boldsymbol{w}$. ■

4.4 Computation

In this section, we propose an effective procedure to compute implicants of an MTL formula φ relative to a finite trace $\boldsymbol{w}$ of length d. First, note that the satisfaction signal $[\varphi]_{\boldsymbol{w}}$ of a given formula φ relative to a finite variability trace $\boldsymbol{w}$ has itself finite variability, the variability of satisfaction signals growing at most quadratically with the size of the formula. Like satisfaction, an explanation for a temporal formula is *time dependent* and should be a function from the temporal domain to formulas that explain satisfaction or violation from some time t. Analogously to the notion of satisfaction signal $[\varphi]_w \in 2^{\mathbb{T}}$ we define the notion of *explanation signal* denoted $E_w(\varphi)$ such that $E_w(\varphi)(t)$ explains the satisfaction of φ by $\boldsymbol{w}$ from time $t \in \mathbb{T}$. We then construct explanations through definitions of $E_w(\varphi)(t)$ and its dual $F_w(\varphi)(t)$, which are inductive on the structure of formula φ, and on the times t at which explanations of its subformulas are required. We are able to guarantee finite representation by producing finite variability explanation signals. We use selection functions ξ_w^φ to relate the truth of some formula φ at time t with the truth of its direct subformulas at some time $\xi_w^\varphi(t)$. Arbitrary selection functions may yet lead to explanations which are almost as large as the trace itself, however we can find selection functions that allow best "explanation sharing". For instance given a non-singular interval I the same t' may belong to $t \oplus I$ for every t in some interval T. Hence a selection function satisfying $\xi_w^\varphi(t) = t'$ for every $t \in T$ will use only one point to witness the satisfaction of $\varphi = \Diamond_I \psi$ throughout T.

4.4.1 Metric Temporal Logic

In this section MTL formulas are given by the following syntax:

$$\varphi ::= \top \mid p \mid \varphi \vee \varphi \mid \neg\varphi \mid \Diamond_I \varphi \mid \varphi \,\mathrm{U}\, \varphi$$

where p is a proposition in $\mathbb{P}$, and I is an interval in $\mathbb{R}_{\geq 0}$. We further assume that all such intervals $I \subseteq \mathbb{R}_{\geq 0}$ has integer bounds, which can always be achieved by scaling considering formulas with rational timing constants. Semantics of MTL formulas are given inductively as follows:

$$\begin{array}{lll}
(w,t) \models p & \text{iff} & t \in p_w \\
(w,t) \models \neg\varphi & \text{iff} & (w,t) \not\models \varphi \\
(w,t) \models \varphi_1 \vee \varphi_2 & \text{iff} & (w,t) \models \varphi_1 \text{ or } (w,t) \models \varphi_2 \\
(w,t) \models \varphi_1 \,\mathrm{U}\, \varphi_2 & \text{iff} & \exists t' > t,\, (w,t') \models \varphi_2 \text{ and } \forall t'' \in (t,t'),\, (w,t'') \models \varphi_1 \\
(w,t) \models \Diamond_I \varphi & \text{iff} & \exists t' \in t \oplus I,\, (w,t') \models \varphi
\end{array}$$

Thanks to the *until rewrite* rules of Proposition 3.1, these definitions of MTL are equivalent to those of Section 3.2.1 as far as expressiveness is concerned.

We extend the notion of satisfaction signal $[\varphi]_w$ to sets of formulas Ψ, by letting $[\Psi]_w \in 2^{\mathbb{T}\times\Psi}$ be a multi-dimensional signal featuring the corresponding $|\Psi|$ satisfaction signals $[\psi]_w$ for $\psi \in \Psi$. The satisfaction signals of φ and of all its subformulas ψ are assumed to be given as the result of applying a monitoring procedure, such as the one described in Section 3.2.3, to $\boldsymbol{w}$ and φ.

For the purpose of handling the negation of an *until* formula we introduce its dual operation *release* defined by $\varphi \,\mathrm{R}\, \psi = \varphi \,\mathrm{U} \top \;\vee\; \psi \,\mathrm{U}(\psi \wedge \varphi \,\tilde{\mathrm{U}}\, \varphi) \;\vee\; \square_{(0,\infty)} \psi$. The *release* and *until* operators verify the following property.

Proposition 4.7 (Until Duality). *For any formulas φ, ψ it holds $\neg(\varphi \,\mathrm{U}\, \psi) \Leftrightarrow \neg\varphi \,\mathrm{R}\, \neg\psi$.*

The negation of an *until* formula is explained this way: $\varphi \,\mathrm{U}\, \psi$ does not hold if φ is immediately false, or if φ becomes false before (or immediately when) ψ becomes true, or if ψ never holds in the future.

The remainder of this section is organized as follows. We begin by extending the semantics of MTL to handle limits so as to conveniently represent earliest, and latest witnesses. Informally, a *witness* of some formula is some time at which a subformula holds and which is sufficient to account for the formula's satisfaction. The definition of explanation operators is then given in full. Following developments describe a procedure to compute an instance of selection functions correct with respect to a given signal. An example and further remarks conclude this section.

4.4.2 Non-Standard Semantics

To facilitate the definitions of witnesses of a formula, we extend MTL semantics to time domains with non-standard reals t^+ and t^-, representing limit points as previously. The relation $<$ over the reals of $\mathbb{T} = [0, d]$ extends into a transitive, antisymmetric relation over $\mathbb{T}^\pm$ using the following additional equations, for all $s \in (0, d]$ and $t \in [0, d)$:

$$\begin{aligned} s^- &< s & \qquad t &< t^+ \\ s^- &< s^- & \qquad t^+ &< t^+ \end{aligned}$$

and taking the transitive closure. Note that with this definition, relation $<$ is not rigorously speaking an order relation over $\mathbb{T}^\pm$: it is reflexive (strict) over the reals, but antireflexive (non-strict) over non-standard reals. Non-strict order relation $\leq$ is defined over $\mathbb{T}^\pm$ by letting $t \leq t'$ if and only if $t < t'$ or $t = t'$ for all $t, t' \in \mathbb{T}^\pm$. We use interval notations $[t, t']$, (t, t'), $(t, t']$, and $[t, t')$ with their usual meaning in terms of inequalities $<$ and $\leq$ over $\mathbb{T}^\pm$.

The sum of a symbolic limit t^+, respectively t^-, and a real number a is taken as $(t + a)^+$, respectively $(t + a)^-$. The sum $T \oplus I$ of an interval $T \subseteq \mathbb{T}^\pm$ and an interval I of $\mathbb{R}_{\geq 0}$ is defined as the convex hull in $\mathbb{T}^\pm$ of the Minkowski sum $\{t + a \in \mathbb{T}^\pm : t \in T, a \in I\}$, and similarly for the difference $T \ominus I$. We use abbreviations $t \oplus I$ and $t \ominus I$ as previously.

Recall from Section 4.3.1 that any trace $\boldsymbol{w}$ over $\mathbb{T}$ extends to a trace over $\mathbb{T}^\pm$ by defining its value in t^+ and t^- in terms of right- and right- limits respectively. Let us write $(\boldsymbol{w}, t) \models^\pm \varphi$ to signify that the extension to $\mathbb{T}^\pm$ of some trace $\boldsymbol{w}$ over $\mathbb{T}$ satisfies formula φ, using $\mathbb{T}^\pm$ as temporal domain. The following proposition states that such a non-standard semantics is consistent with the standard one.

Proposition 4.8. *Let $\boldsymbol{w}$ be a finitely variable trace over $\mathbb{T}$, and φ an MTL formula.*

- *For any $t \in \mathbb{T}$, $(\boldsymbol{w}, t) \models^\pm \varphi$ if and only if $(\boldsymbol{w}, t) \models \varphi$.*
- *For any $t \in \mathbb{T}^+$, $(\boldsymbol{w}, t) \models^\pm \varphi$ if and only if $\forall \delta > 0, \exists \epsilon \in (0, \delta), (\boldsymbol{w}, t + \epsilon) \models \varphi$.*
- *For any $t \in \mathbb{T}^-$, $(\boldsymbol{w}, t) \models^\pm \varphi$ if and only if $\forall \delta > 0, \exists \epsilon \in (0, \delta), (\boldsymbol{w}, t - \epsilon) \models \varphi$.*

Proof (sketch). The proof is by induction on the formula structure. We show that for finitely variable traces, the limit $[\varphi]_{\boldsymbol{w}}(t^+)$ is defined everywhere in $[0, d)$ and $[\varphi]_{\boldsymbol{w}}(t^-)$ everywhere in $(0, d]$. In particular the negation gives us the *only if* part, and can be proved using the following fact: given f a signal admitting a right-limit at t, it holds $\forall \delta > 0, \exists \epsilon \in (0, \delta), f(t + \epsilon) = 1$ if and only if $\exists \delta > 0, \forall \epsilon \in (0, \delta), f(t + \epsilon) = 1$, and similar for t^-. ∎

Hence we may simplify the notation, and safely write $(w,t) \models \varphi$ in place of $(w,t) \models^{\pm} \varphi$ for any trace w, formula φ, and (non-standard) time $t \in \mathbb{T}^{\pm}$.

4.4.3 Explanation Operators

We may now define $E_w(\varphi)(t)$ and $F_w(\varphi)(t)$ providing explanations of φ, or $\neg\varphi$, relative to trace w at time $t \in \mathbb{T}^{\pm}$ in the form of temporal terms. The diagnostic of formula φ relative to trace w is then given by the value at time 0 of $E_w(\varphi)$ in case of satisfaction, or that of $F_w(\varphi)$ in case of violation. We let

$$D_w(\varphi) = \begin{cases} E_w(\varphi)(0) & \text{if } (w,0) \models \varphi \\ F_w(\varphi)(0) & \text{otherwise} \end{cases}$$

with

$$\begin{aligned}
E_w(\top)(t) &= \top & F_w(\top)(t) &= \bot \\
E_w(p)(t) &= p(t) & F_w(p)(t) &= \neg p(t) \\
E_w(\neg\varphi)(t) &= F_w(\varphi)(t) & F_w(\neg\varphi)(t) &= E_w(\varphi)(t) \\
E_w(\varphi \vee \psi)(t) &= E_w(\xi_w^{\varphi\vee\psi}(t))(t) & F_w(\varphi \vee \psi)(t) &= F_w(\varphi)(t) \wedge F_w(\psi)(t) \\
E_w(\Diamond_I \varphi)(t) &= \begin{cases} E_w(\varphi)(t+a) & \text{if } I = [a,a] \\ E_w(\varphi)(\xi_w^{\Diamond_I \varphi}(t)) & \text{otherwise} \end{cases} & F_w(\Diamond_I \varphi)(t) &= \bigwedge_{t' \in t \oplus I} F_w(\varphi)(t') \\
E_w(\varphi \,\mathrm{U}\, \psi)(t) &= E_w(\psi)(\xi_w^{\varphi \mathrm{U} \psi}(t)) \wedge \bigwedge_{t' \in (t, \xi_w^{\varphi \mathrm{U} \psi}(t))} E_w(\varphi)(t') & F_w(\varphi \,\mathrm{U}\, \psi)(t) &= E_w(\neg\varphi \,\mathrm{R}\, \neg\psi)(t)
\end{aligned}$$

where the *selection functions* $\xi_w^{\varphi\vee\psi}$ from $\mathbb{T}^{\pm}$ to formulas, $\xi_w^{\Diamond_I \varphi}$ and $\xi_w^{\varphi \mathrm{U} \psi}$ from $\mathbb{T}^{\pm}$ to $\mathbb{T}^{\pm}$ are such that for all $t \in \mathbb{T}^{\pm}$, it holds $\xi_w^{\varphi\vee\psi}(t) \in \{\varphi, \psi\}$, $\xi_w^{\Diamond_I \varphi}(t) \in t \oplus I$ and $\xi_w^{\varphi \mathrm{U} \psi}(t) > t$.

We say that a selection function ξ_w^{φ} is correct with respect to w if for all $t \in \mathbb{T}^{\pm}$ such that $(w,t) \models \varphi$ we have

- $(w,t) \models \xi_w^{\varphi}(t)$ when φ is of the form $\psi_1 \vee \psi_2$;
- $(w, \xi_w^{\varphi}(t)) \models \psi$ when φ is of the form $\Diamond_I \psi$;
- $(w, \xi_w^{\varphi}(t)) \models \psi_2$ and $\forall t' \in (t, \xi_w^{\varphi}(t))$, $(w,t') \models \psi_1$ when φ is of the form $\psi_1 \,\mathrm{U}\, \psi_2$.

The following theorem, that states our approach is consistent, is straightforward to establish by structural induction. The proof consists in checking the logical dependency between a formula and its subformula for propositional operators, and ensuring that witnesses provided by selection function are sufficient to account for the satisfaction or violation of the main formula for temporal operators.

Theorem 4.9 (Soundness). *A term $D_w(\varphi)$, correct with respect to trace w is a solution to the diagnostics problem of φ with respect to w.*

Given finite variability selection functions, the diagnostic $D_w(\varphi)$ can be effectively represented.

Proposition 4.10 (Finite Representation). *Assuming all selection functions are finitely variable, the term $D_w(\varphi)$ has a normal form $\bigwedge_{\ell\in\mathbb{L}}\bigwedge_{t'\in T_\ell}\ell(t')$ such that each T_ℓ is a* finite union of intervals *of* $\mathbb{T}^\pm$.

Proof. We prove the property by induction on the formula structure. First notice that singular conditions $\ell(t)$ are equivalent to $\bigwedge_{t'\in[t,t]}\ell(t')$. This takes care of atomic formulas. The negation case is follows directly from the induction hypothesis and duality of explanation and falsification. For all other MTL operators it is sufficient to consider the expansion of the term produced by our procedure under a single conjunction ranging over an interval T. Namely we show that any term $\bigwedge_{t\in T}\theta(t)$ where θ is a diagnostic (explanation or falsification) for φ can be replaced with a finite conjunction of terms $\bigwedge_{t\in T'}\theta'(t)$ or simply $\theta'(t')$ for $t'\in\mathbb{T}^\pm$, T' an interval of $\mathbb{T}^\pm$, and θ' a diagnostic for subformulas of φ.

Explaining a disjunction produces

$$\bigwedge_{t\in T}E(\varphi\vee\psi)(t)=\bigwedge_{t\in T}E(\xi_{\varphi\vee\psi}(t))(t)\Leftrightarrow\bigwedge_{i=1}^{k}\bigwedge_{t\in T_i\cap T}E(\varphi)(t)\wedge\bigwedge_{i=1}^{n}\bigwedge_{t\in T_i'}E(\psi)(t)$$

where T_i, T_i' are intervals of $\mathbb{T}^\pm$ that can be obtained according to the finite-variability of ξ. The falsification of a disjunction is direct. For the falsification of an *eventually* formula we obtain

$$\bigwedge_{t\in T}F(\Diamond_I\varphi)(t)=\bigwedge_{t\in T}\bigwedge_{t'\in t+I}F(\varphi)(t')\Leftrightarrow\bigwedge_{t'\in T+I}F(\varphi)(t')$$

where $T+I=\bigcup_{t\in T}t+I$ is an interval of $\mathbb{T}^\pm$. The explanation of an *eventually* formula is treated as follows:

$$\bigwedge_{t\in T}E(\Diamond_I\varphi)(t)=\bigwedge_{t\in T}E(\varphi)(\xi(t))\Leftrightarrow\bigwedge_{i=1}^{k}E(\varphi)(t_i)$$

for some finite sequence of times $(t_i)_{i\leq k}$ thanks to the finite-variability of ξ. For the explanation of an *until* formula we write

$$\bigwedge_{t\in T}E(\varphi\,\mathrm{U}\,\psi)(t)=\bigwedge_{t\in T}E(\psi)(\xi(t))\wedge\bigwedge_{t'\in(t,\xi(t))}E(\varphi)(t')\Leftrightarrow\bigwedge_{i=1}^{k}E(\psi)(t_i)\wedge\bigwedge_{t'\in(s_{i-1},t_i)}E(\varphi)(t')$$

given $(t_i)_{i=1..k}$ the finite sequence of switching points of ξ over T, $s_0=\min T$, and s_i the standard part of t_i for all $i>0$. The falsification of an *until* formula is handled by rewriting and goes through cases already verified. ∎

4.4.4 Selection Functions

We now describe some procedures defining explicit instances of selection functions, which satisfy the correctness and finite variability criteria. The explanation operators can then be made constructive when the normalization of the terms they produce is interleaved with the instantiation of selection functions over intervals appearing in the normalization process. It is indeed sufficient to define selection functions piecewise, on closed intervals T of $\mathbb{T}^\pm$. Furthermore we can assume that for such intervals T, formula φ holds for all $t\in T$ as the correctness assumption is void outside such intervals, with the finite variability assumption then trivial to fulfill.

Disjunction

Consider the formula $\varphi \vee \psi$, and a trace $\boldsymbol{w}$. A finitely variable selection function $\xi_{\boldsymbol{w}}^{\varphi\vee\psi}$ correct with respect to $\boldsymbol{w}$ can be constructed as follows. By finite variability hypothesis on $\boldsymbol{w}$, the satisfaction signal $[\varphi, \psi]_w$ has finite variability over any interval T where $\varphi \vee \psi$ holds. We partition T in k maximally uniform intervals T_i, and take $\xi_{\boldsymbol{w}}^{\varphi\vee\psi}(t) = \varphi$ over intervals T_i where $(\boldsymbol{w}, t) \models \varphi$, and $\xi_{\boldsymbol{w}}^{\varphi\vee\psi}(t) = \psi$ over other intervals. The function $\xi_{\boldsymbol{w}}^{\varphi\vee\psi}$ is uniform over all T_i, hence it has finite variability.

Timed Eventually

Now consider the formula $\Diamond_I \varphi$ for a non-singular interval I, and a trace $\boldsymbol{w}$. Assume that φ is satisfied by $\boldsymbol{w}$ continuously over some time interval T. We build a procedure that generates a small set of witnesses of φ accounting for the satisfaction of $\Diamond_I \varphi$ by $\boldsymbol{w}$ over T. The satisfaction of $\Diamond_I \varphi$ over T can be explained by the satisfaction of φ at some time points (witnesses) in $T \oplus I$, and in particular the satisfaction of φ at some $s \in T \oplus I$ provides a sufficient explanation for the satisfaction of $\Diamond_I \varphi$ for all $t \in (s \ominus I) \cap T$. We use these two observations to generate a piecewise constant selection function $\xi_{\boldsymbol{w}}^{\Diamond_I \varphi}$ defined over T and correct relative to a given trace $\boldsymbol{w}$.

Algorithm 4.1 WITNESSES($\Diamond_I \varphi, \boldsymbol{w}$)

```
 1: $\xi_{\boldsymbol{w}}^{\Diamond_I \varphi} := \emptyset$
 2: **while** $T \neq \emptyset$ **do**
 3:     $t := \min(T)$
 4:     $S := (t \oplus I) \cap \{t' : (\boldsymbol{w}, t') \models \varphi\}$
 5:     $s := \max(S)$
 6:     $R := (s \ominus I) \cap T$
 7:     $\xi_{\boldsymbol{w}}^{\Diamond_I \varphi} := \xi_{\boldsymbol{w}}^{\Diamond_I \varphi} \cup (R \times \{s\})$
 8:     $T := T \setminus R$
 9: **end while**
10: **return** $\xi_{\boldsymbol{w}}^{\Diamond_I \varphi}$
```

We present the procedure in Algorithm 4.1; it works as follows. The selection function is initialized (line 1) as nowhere defined. In every iteration of the main while loop (line 2), we find a temporal domain $S = (t \oplus I) \cap \{t' : (\boldsymbol{w}, t') \models \varphi\}$ such that φ is satisfied inside S and any point in S provides a sufficient explanation for the satisfaction of $\Diamond_I \varphi$ at t taken as the earliest time of T. Such set S is obtained directly from the satisfaction signal $[\varphi]_w$ assumed to be already computed by the monitoring procedure. We then take s the latest time of S, which constitutes a minimal subset of S sufficient to explain the satisfaction of $\Diamond_I \varphi$ throughout the domain $s \ominus I$; when intersected with T it gives R, a prefix of T. At the end of the iteration, the definition of $\xi_{\boldsymbol{w}}^{\Diamond_I \varphi}$ over the interval R is taken as s, which we may write $R \times \{s\}$ identifying selection functions with subsets of $\mathbb{T}^{\pm} \times \mathbb{T}^{\pm}$ (line 7). The covered prefix R can be removed from T (line 8). The procedure terminates when sufficient witnesses of φ are found, i.e. when T the domain remaining to cover becomes empty.

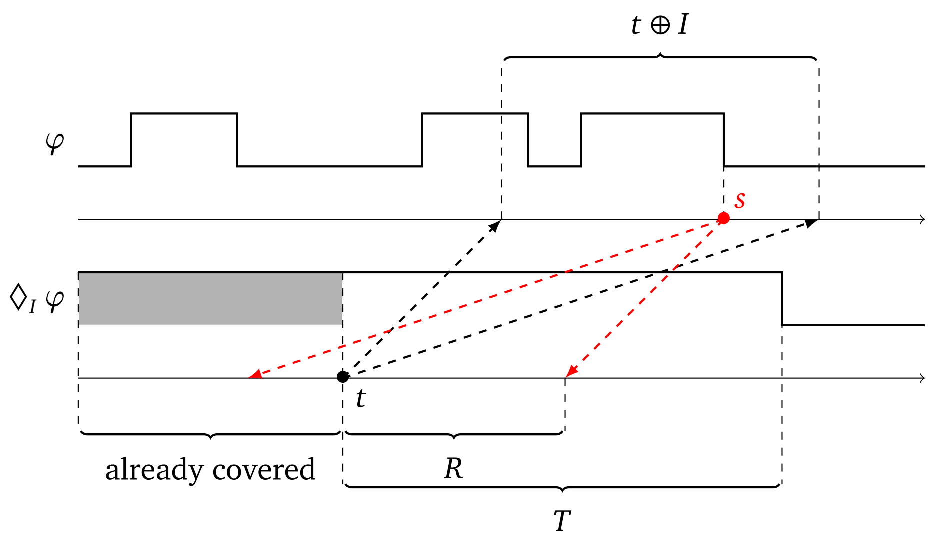


Figure 4.2: Example of R and s computation for $\Diamond_I \varphi$.

Untimed Until

Consider the formula $\varphi \,\mathrm{U}\, \psi$ and a trace w, and assume that the formula is satisfied over T, taken without loss of generality to be a closed interval of $\mathbb{T}^{\pm}$. For $t \in \mathbb{T}^{\pm}$, similarly to the case of *timed eventually* a single witness $t' > t$ of ψ along with a uniform interval (t, t') where φ holds is sufficient to explain the satisfaction of $\varphi \,\mathrm{U}\, \psi$ over the whole interval $[t, t')$. With such observations we generate a piecewise constant selection function $\xi_w^{\varphi \mathrm{U} \psi}$ correct with respect to w and defined over T. We make use of a subroutine $Z_w(\varphi, \psi, t)$ that returns the set of witnesses of ψ in trace w that are sufficient to explain $\varphi \,\mathrm{U}\, \psi$ at time $t \in \mathbb{T}^{\pm}$ where $\varphi \,\mathrm{U}\, \psi$ holds. We have

$$Z_w(\varphi, \psi, t) = \{t' > t : (w, t') \models \psi \text{ and } \forall t'' \in (t, t'), (w, t'') \models \varphi\}$$

Assuming the satisfaction signals $[\varphi]_w$ and $[\psi]_w$ given by the monitoring algorithm, the procedure $Z_w(\varphi, \psi, t)$ can be realized as follows. First write the domain $\{t' \in \mathbb{T}^{\pm} : t' > t\}$ as a finite partition into uniform intervals of $\mathbb{T}^{\pm}$ with respect to $[\varphi, \psi]_w$ that we can assume of the form $[t_i, t_i]$ and (t_i, t_{i+1}) with $(t_i)_{i \le n}$ an ordered sequence of times, as explained in Section 3.2.3. Start from the interval containing $t_0 = t$ and iterate through the intervals, accumulating intervals where ψ holds, until φ stops holding at $[t_i, t_i]$ or (t_i, t_{i+1}), or we reach the interval containing $t_n = d$ marking the end of the temporal domain.

We present the main procedure to compute the selection function in Algorithm 4.2-(a). The procedure first assigns $\xi_w^{\varphi \mathrm{U} \psi}$ the empty function $\emptyset$ (line 1). In every iteration of the while loop, we compute an interval S whose elements s are witnesses of ψ providing a sufficient explanation for the satisfaction of the $\varphi \,\mathrm{U}\, \psi$ throughout $[t, s)$. When S lies entirely outside T we define s as the earliest suitable witness of ψ, so as not to impose a condition on φ beyond it (line 5). When S intersects with T we look for the latest suitable witness of ψ in their intersection (line 6); this witness explains the *until* formula for all times that precede it (line 8). The interval $R = [t, s) \cap T$ is now accounted for,

Algorithm 4.2 WITNESSES($\varphi \,\mathrm{U}\, \psi, w$)

```
1: $\xi_w^{\varphi \mathrm{U} \psi} := \emptyset$
2: while $T \neq \emptyset$ do
3:     $t := \min(T)$
4:     $S := Z_w(\varphi, \psi, t)$
5:     if $S \cap T = \emptyset$ then
6:         $s := \min(S)$
7:     else
8:         $s := \max(S \cap T)$
9:     end if
10:    $R := [t, s) \cap T$
11:    $\xi_w^{\varphi \mathrm{U} \psi} := \xi_w^{\varphi \mathrm{U} \psi} \cup (R \times \{s\})$
12:    $T := T \setminus R$
13: end while
14: return $\xi_w^{\varphi \mathrm{U} \psi}$
```

hence we define $\xi_w^{\varphi \mathrm{U} \psi}$ as taking the value s over interval R (line 8). Eventually R can be removed from T for the next iteration (line 9), and the procedure terminates when T becomes empty.

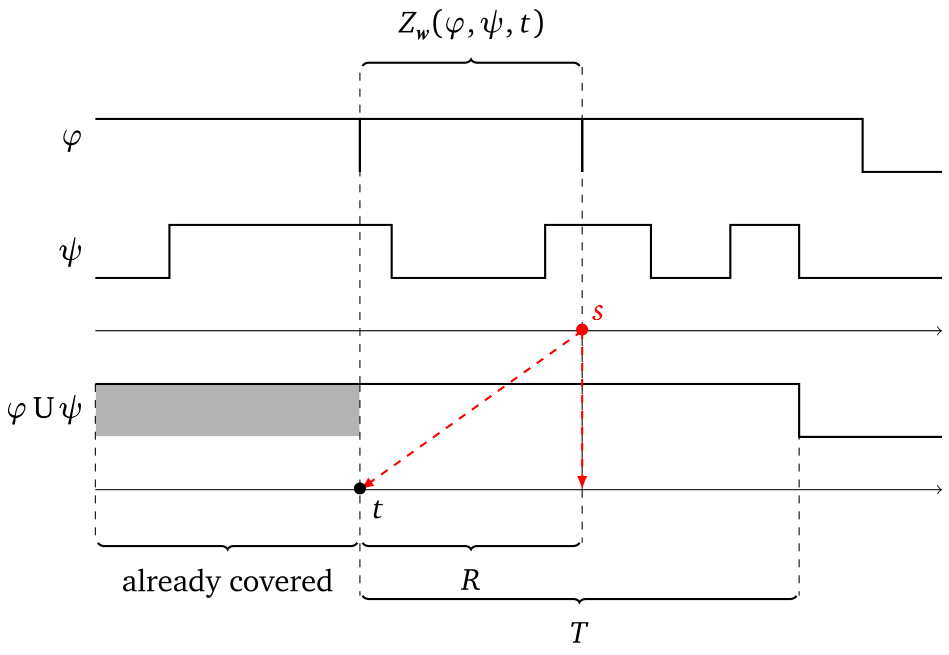


Figure 4.3: Example of R and s computation for $\varphi \,\mathrm{U}\, \psi$.

4.4.5 Evaluation

Let us illustrate the overall process of deriving an explanation for a non-trivial formula.

Example 4.3. *Let $\varphi = \Box(\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q)$, and let $\mathbf{w}$ be the signal appearing in Figure 4.4. The diagnostic $D_w(\varphi)$ is illustrated in terms of sub-signals, inductively extracted from satisfaction signals; its computation is as follows:*

$$\begin{aligned}
D_w(\varphi) &= F_w\left(\Box(\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q)\right)(0)\\
&= F_w\left(\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q\right)(9.7)\\
&= F_w\left(\Box_{[0,2]}\, p\right) \wedge F_w\left(\Diamond_{[4,5]}\, q\right)(9.7)\\
&= \bigwedge_{t\in[9.7,11.7]} F_w(\neg p)(t) \;\wedge\; F_w\left(\Diamond_{[4,5]}\, q\right)(9.7)\\
&= \bigwedge_{t\in[9.7,11.7]} E_w(p)(t) \;\wedge\; F_w\left(\Diamond_{[4,5]}\, q\right)(9.7)\\
&= \bigwedge_{t\in[9.7,11.7]} p(t) \;\wedge\; F_w\left(\Diamond_{[4,5]}\, q\right)(9.7)\\
&= \bigwedge_{t\in[9.7,11.7]} p(t) \;\wedge \bigwedge_{t\in[13.7,14.7]} F_w(q)(t)\\
&= \bigwedge_{t\in[9.7,11.7]} p(t) \;\wedge \bigwedge_{t\in[13.7,14.7]} \neg q(t)
\end{aligned}$$

According to $D_w(\varphi)$, formula φ is violated by trace $\mathbf{w}$ because p is true over $[9.7, 11.7]$ and q is false over $[13.7, 14.7]$.

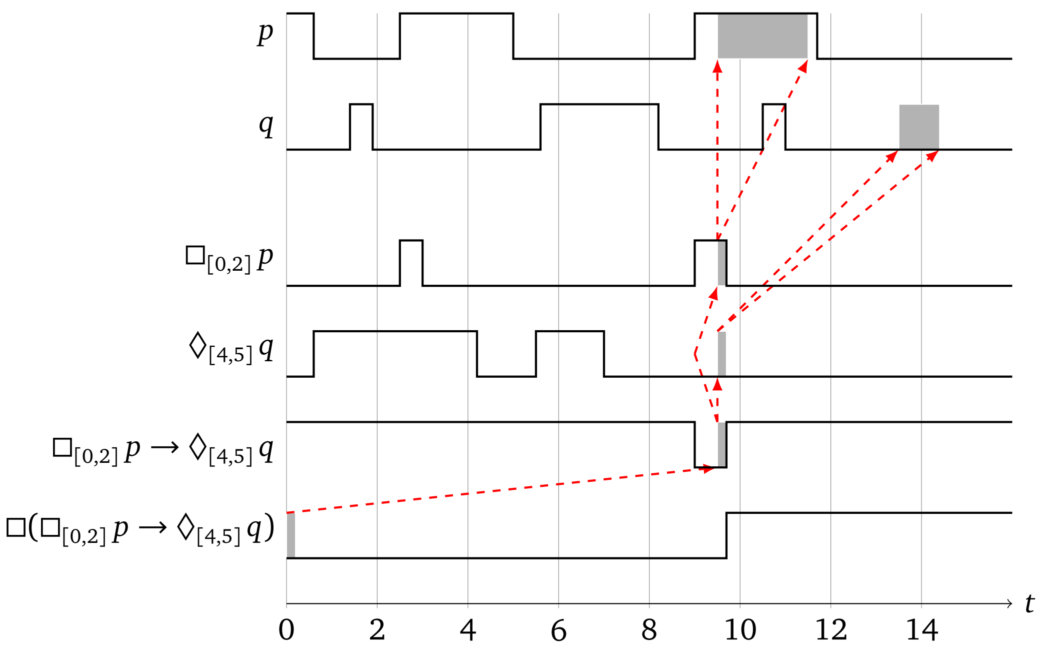

p
q
$\Box_{[0,2]}\, p$
$\Diamond_{[4,5]}\, q$
$\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q$
$\Box(\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q)$
t
0
2
4
6
8
10
12
14

Figure 4.4: The trace $\mathbf{w}$ and diagnostic of $\varphi = \Box(\Box_{[0,2]}\, p \rightarrow \Diamond_{[4,5]}\, q)$ produced by our algorithms: trace, satisfaction signals, and shaded sub-signals. Red arrows exhibit the dependencies between implicants of some formula and those of its direct subformulas.

We now examine the complexity of our algorithm. Let us define the size of formulas and traces as in Chapter 3. We make the additional assumption that all traces have a variability bounded by some constant c, that is all signals appearing in a trace $\boldsymbol{w}$ have at most c discontinuities per time unit.

Theorem 4.11. *Computing $D_w(\varphi)$ with our algorithms has time-complexity in $O(|\varphi|^2 \cdot |\boldsymbol{w}|)$.*

Proof (sketch). The proof proceeds by induction on the formula structure. We show similarly to Theorem 3.6 that the number of terms in the explanation of a formula is linear in $|\varphi| \cdot |\boldsymbol{w}|$. Note that for a timed eventually operator $\Diamond_I$, the number of witnesses needed is at most twice the size of trace divided by the variability c, given that interval I has integer bounds. Computing the satisfaction signal and diagnostics with such a complexity for each subformula creates an additional factor $|\varphi|$. ∎

As mentioned earlier, our procedure does not guarantee minimality, as it does not recognize tautologies. However we obtain some form of temporal minimality through the proposed construction for selection functions. Intuitively each time a witness is required we select the furthest away, which maximizes the interval over which that witness is valid. Let φ be an *eventually* or *until* formula, $\boldsymbol{w}$ a trace such that φ holds for $\boldsymbol{w}$ on some domain T. One can see that selection functions ξ_w^φ constructed by our algorithms choose a set of witnesses $\xi_w^\varphi(T) = \{\xi_w^\varphi(t) : t \in T\}$, which is minimal relative to $\boldsymbol{w}$.

The main advantage of our explanation principle is its hierarchical character: every subformula has its own explanation, which is used in turn to account for the satisfaction or violation of its super-formulas. The process of fault-finding is progressive: if the fault lies in the specification then it can be localized syntactically, and otherwise the parts of the specification that explain the failure give additional insight. In the presences of multiple explanations, a user of our procedure can be given the choice between several alternatives. The hierarchical decomposition we propose also enables us to solve the diagnostics problem efficiently. Under a uniform bounded variability assumption, computing $D_w(\varphi)$ with our algorithms takes time quadratic in the size of the formula and linear in the size of the input trace. The minimal diagnostics problem has a higher complexity. By reduction to the satisfiability of MTL over bounded time [99], minimal diagnostics is EXPSPACE-hard in the size of the formula.

5 Robustness Analysis

Traditionally, monitoring some temporal formula φ only gives a pass or fail verdict for a given trace $\boldsymbol{w}$. The robustness value, defined in what follows, enables in addition to estimate the distance from $\boldsymbol{w}$ to the boundary between satisfaction and violation of φ. In this chapter we solve the robust monitoring problem for a variant of Metric Temporal Logic. This variant is obtained by considering MTL over continuous-time Boolean and real signals, and using propositions $x > c$ for real signals x and constants c. The resulting specification language, dubbed Signal Temporal Logic (STL), provides a convenient framework for specifying properties of continuous-time signals. An important application of robust monitoring is to help find the stimulus or parameter set that leads to the violation of a property; this can be done using an off-the-shelf continuous optimization routine in order to minimize the robustness value. The method that we propose operates directly on piecewise linear signals, and thus does not introduce additional error due to sampling. The efficiency of resulting algorithms is comparable to that of standard (pass/fail) monitoring algorithms, in terms of theoretical complexity and practical performance.

5.1 Introduction

Signal Temporal Logic [83, 86] is becoming an established formalism for the specification of continuous-time behaviors. A monitoring tool called AMT [96] has been developed and used in the context of analog and mixed-signal circuits [68, 85]. In assertion based verification, and more specifically in the analog and mixed signal domain, a pass/fail status only provides partial information and could be augmented with *quantitative* information about the satisfaction, so as to provide a better basis for decision making. To illustrate, consider the safety condition $x \leq c$ for constant c and a real variable x. The condition splits the values $\boldsymbol{v}$ of x into safe ones $S = \{\boldsymbol{v} : x_{\boldsymbol{v}} \leq c\}$ and unsafe ones $\overline{S} = \{\boldsymbol{v} : x_{\boldsymbol{v}} > c\}$. For a given value $\boldsymbol{v}$ of x, the answer to the satisfaction query $x \leq c$ depends on the membership of $\boldsymbol{v}$ in S but *not* on its relative position inside or outside S. The *robustness* of the formula $x \leq c$ relative to $\boldsymbol{v}$ should tell us whether $\boldsymbol{v}$ satisfies the property by far ($x_{\boldsymbol{v}} = c - \delta$ for a large δ) or very marginally ($x_{\boldsymbol{v}} = c - \delta$ for a small δ). For this example, the robustness degree is captured by $c - x_{\boldsymbol{v}}$ whose *sign* indicates satisfaction/violation and its *magnitude* indicates the distance between x_v and the boundary between S and $\overline{S}$.

Of course the computation of this distance is trivial for a single proposition $x \leq c$ and fixed valuation $\boldsymbol{v}$. Signal Temporal Logic on the other hand allows to specify propositional combinations of such threshold propositions, and their temporal evolution. For example consider the formula $\varphi = \Box(x > c) \rightarrow \Diamond_{[0,b]}(y \leq d)$, which reads "always when x is above value c, eventually within b time units, y is at most d". Computing the distance of some trace $\boldsymbol{w}$ with real variables x and y violating φ to a nearest trace satisfying φ, is not as simple. This distance depends on the distance from x and y to c and d respectively, and also on their evolution in time. In this chapter we solve this problem in an approximate fashion by providing a robustness *estimate*, which is a guaranteed under-approximation of the real distance to satisfaction (or violation in the opposite case).

The notion of robustness has been introduced into temporal logic by Fainekos and Pappas [52] for MTL, and by Fages and Rizk [104] for LTL over real-valued sequences. The robustness information is useful to assess the severity of an assertion violation, when part of the assertion ranges over real valued quantities. It can also increase the confidence in the results of verification when the coverage is small, if it so happens that all sampled behaviors satisfy the requirements robustly. In [46] the notion of robustness was studied both in the space and time dimension; Donzé and Maler provided an algorithm for computing this robustness degree with respect to a given trace. A related problem is that of computing the distance between two traces according to the Skorokhod metric, which integrates both the time and space dimensions [42].

The contribution of this chapter is a new, optimal algorithm that computes the robustness degree in time linear with respect to the signal size. As customary, real valued signals are represented as sequences of timed-stamped values (discrete-time real signals) with piecewise-linear interpolation.[1] Our algorithms guarantee that the overhead added by monitoring to the simulation process is acceptable, thus making robustness-based monitoring a feasible technology that can be used routinely as an add-on for commercial simulators. This low complexity is due to two key ideas:

- The use of the optimal streaming algorithm of Lemire [79] to compute the min and max of a discrete time signal over a sliding window;

- The rewriting of the *timed until* operator as a conjunction of simpler timed and untimed operators, as explained in Section 3.2.2.

The algorithm has been implemented at the core of Breach [44] which is a toolbox for simulation-based analysis of hybrid systems developed by Donzé. It has notably been applied to mine requirements from Simulink models in the automotive industry [67]. Our implementation outperforms the tool S-TaLiRo [23], which at the time the experiments were conducted was the only other tool implementing quantitative semantics for continuous-time.

[1]Minor adaptations of our procedures are sufficient to handle the case of piecewise-polynomial signals.

5.2 Signal Temporal Logic

In this section we extend the framework of Chapter 3 for the specification of continuous-time Boolean and real signals. This framework forms the basis of Signal Temporal Logic, for which we will give *quantitative* semantics, directly inspired from the one proposed by [52] for MTL. We adopt conventions similar to those of Chapter 2 and 3 as follows.

We fix a (continuous) temporal domain $\mathbb{T} = [0, d]$, a set of Boolean variables $\mathbb{P}$, and a set of real variables $\mathbb{X}$. Let $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$ be the totally ordered set of real numbers extended with a smallest element $-\infty$ and a greatest element $+\infty$. A *real signal* is now defined as a function in $(\mathbb{R}_\infty)^\mathbb{T}$, mapping times in $\mathbb{T}$ to real or infinite values in $\mathbb{R}_\infty$. We expect signals given by a simulation trace $\boldsymbol{w}$ not to contain infinite values. However some derived signals will be taking such values, and thus we adopt this simplifying convention.[2] By convention $\inf \mathbb{R} = \sup \emptyset = -\infty$, and symmetrically. A *Boolean signal* is as previously a function in $\{0,1\}^\mathbb{T}$. A trace $\boldsymbol{w}$ is a valuation of real variables $x \in \mathbb{X}$ and Boolean variables $p \in \mathbb{P}$ into real signals x_w and Boolean signals p_w defined over the temporal domain $\mathbb{T}$. Such a trace can be quantized through a set of threshold propositions of the form $x \leq c$ and their negation. Formally, the syntax of STL is as follows:

$$\varphi ::= \top \mid p \mid x \leq c \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathrm{U}_I\, \varphi$$

for p Boolean variable in $\mathbb{P}$, x real variable in $\mathbb{X}$, c a constant in $\mathbb{R}$, and I interval of $\mathbb{R}_{\geq 0}$.

5.2.1 Qualitative Semantics

For a trace $\boldsymbol{w}$, the validity of an STL formula φ at a given time $t \in \mathbb{T}$ is set according to the inductive definitions of Chapter 2. In particular for threshold propositions we have by definition

$$(\boldsymbol{w}, t) \models x \leq c \quad \text{iff } x_w(t) \leq c$$

Let us recall the definition of *satisfaction signal* $[\varphi]_w$ by letting

$$[\varphi]_w(t) = \begin{cases} 1 \text{ if } (w,t) \models \varphi \\ 0 \text{ otherwise} \end{cases}$$

for all $t \in \mathbb{T}$.

As we have seen in Chapter 3, monitoring the satisfaction of a formula φ can be done by computing for each subformula ψ of φ the entire satisfaction signal $[\psi]_w$. Translating the semantics of each operator using the operations min, max, and involution in the canonical Boolean algebra $\{0,1\}$, we obtain the equivalent characterization of

[2]Standard double-precision floating-point numbers have special values representing $-\infty$ and $+\infty$. Electrical, and other physical quantities typically have a bounded range, so that such infinite values are merely place holders for some value whose magnitude exceeds that of any other present quantity.

(qualitative) semantics of STL as follows. For all $t \in \mathbb{T}$ we have

$$\begin{aligned}
[\top]_w(t) &= 1 \\
[p]_w(t) &= p_w(t) \\
[x \leq c]_w(t) &= \begin{cases} 1 \text{ if } x_w(t) \leq c, \\ 0 \text{ otherwise} \end{cases} \\
[\neg\varphi]_w(t) &= 1 - [\varphi]_w(t) \\
[\varphi \vee \psi]_w(t) &= \max\{[\varphi]_w(t), [\psi]_w(t)\} \\
[\varphi \, \mathrm{U}_I \, \psi]_w(t) &= \sup_{t' \in t \oplus I} \min\{[\psi]_w(t'), \inf_{t'' \in (t,t')} [\varphi]_w(t'')\}
\end{aligned}$$

Here each operator is interpreted as a signal transducer.

5.2.2 Quantitative Semantics

The previous inductive definition of satisfaction signals only needs a minor modification to operate over real rather than Boolean values. Given a formula φ, trace $\boldsymbol{w}$, and time $t \in \mathbb{T}$, we define the quantitative semantics $[\![\varphi]\!]_w(t)$ by induction as follows:

$$\begin{aligned}
[\![\top]\!]_w(t) &= +\infty \\
[\![p]\!]_w(t) &= \begin{cases} +\infty \text{ if } p_w(t) = 1, \\ -\infty \text{ otherwise} \end{cases} \\
[\![x \leq c]\!]_w(t) &= c - x_w(t) \\
[\![\neg\varphi]\!]_w(t) &= -[\![\varphi]\!]_w(t) \\
[\![\varphi \vee \psi]\!]_w(t) &= \max\{[\![\varphi]\!]_w(t), [\![\psi]\!]_w(t)\} \\
[\![\varphi \, \mathrm{U}_I \, \psi]\!]_w(t) &= \sup_{t' \in t \oplus I} \min\{[\![\psi]\!]_w(t'), \inf_{t'' \in (t,t')} [\![\varphi]\!]_w(t'')\}
\end{aligned}$$

It is worth noting that the above inductive rules only differ from the qualitative ones in the interpretation of the propositions. In the quantitative semantics, threshold propositions $x_i \leq c$ do not evaluate to $+\infty$ or $-\infty$ but give a real value representing the distance to satisfaction or to violation, which is then propagated in the formula using operations min, max, and $-$ over $\mathbb{R}_\infty$.

Let us write $\varphi \Leftrightarrow \psi$ when $[\varphi]_w = [\psi]_w$ for all traces $\boldsymbol{w}$, and write $\varphi \Leftrightarrow \psi$ when $[\![\varphi]\!]_w = [\![\psi]\!]_w$ for all traces $\boldsymbol{w}$. From the lattice properties of $(\mathbb{R}_\infty, <)$, we are granted the axioms of associativity, commutativity, neutral element, and distributivity. The minus function remains involutive, which gives us the usual dualities of temporal logic: $\neg(\varphi \wedge \psi) \Leftrightarrow \neg\varphi \vee \neg\psi$ and $\neg \Diamond_I \varphi \Leftrightarrow \Box_I \neg\varphi$. Derived operators enjoy the same natural interpretation as in the Boolean semantics:

$$\begin{aligned}
[\![\varphi \wedge \psi]\!]_w(t) &= \min\{[\![\varphi]\!]_w(t), [\![\psi]\!]_w(t)\} \\
[\![\Diamond_I \varphi]\!]_w(t) &= \sup_{t' \in t \oplus I} [\![\varphi]\!]_w(t') \\
[\![\Box_I \varphi]\!]_w(t) &= \inf_{t' \in t \oplus I} [\![\varphi]\!]_w(t')
\end{aligned}$$

However we do not have over $\mathbb{R}_\infty$ the whole Boolean algebra, as the axiom of excluded middle does not hold. For instance $x \leq 0 \vee x > 0$, whose quantitative semantics is the absolute value of x, is not equivalent to $\top$, whose quantitative semantics is $+\infty$. In general, two STL formulas φ and ψ equivalent in the usual, qualitative semantics may no longer be equivalent in the present, quantitative semantics.

5.2.3 Robustness Estimate

We first define the distance $\mathrm{d}(f,g)$ between two Boolean signals f and g as follows, inherently based on a discrete metric:

$$\mathrm{d}(f,g) = \begin{cases} 0 & \text{if } \forall t \in \mathbb{T}, f(t) = g(t) \\ +\infty & \text{otherwise} \end{cases}$$

We then define the distance $\mathrm{d}(f,g)$ between two real signals f and g based on the infinity norm, and the usual metric on $\mathbb{R}$ extended to $\mathbb{R}_\infty$ as follows:

$$\mathrm{d}(f,g) = \sup_{t\in\mathbb{T}} |f(t) - g(t)|$$

Finally, the distance $\mathrm{d}(\boldsymbol{u},\boldsymbol{v})$ between two traces $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined as the maximal distance between their respective signals, with $\mathrm{d}(\boldsymbol{u},\boldsymbol{v}) = \max_{q\in\mathbb{P}\cup\mathbb{X}} \mathrm{d}(q_{\boldsymbol{u}}, q_{\boldsymbol{v}})$.

The quantitative semantics of STL has two fundamental properties proved in [52], and restated below. Firstly, whenever $[\![\varphi]\!]_{\boldsymbol{w}}(t) \neq 0$ its sign indicates the satisfaction status.

Theorem 5.1 (Soundness). *Let φ be an STL formula, $\boldsymbol{w}$ a trace and t a time in $\mathbb{T}$.*

$$\begin{array}{l} \text{if } [\![\varphi]\!]_{\boldsymbol{w}}(t) > 0 \text{ then } (\boldsymbol{w},t) \models \varphi \\ \text{if } [\![\varphi]\!]_{\boldsymbol{w}}(t) < 0 \text{ then } (\boldsymbol{w},t) \not\models \varphi \end{array}$$

In particular, observe that if $[\![\varphi]\!]_{\boldsymbol{w}}(0) > 0$ then $\boldsymbol{w} \models \varphi$ and symmetrically. Secondly, if $\boldsymbol{u}$ satisfies φ, any other trace $\boldsymbol{v}$ whose pointwise distance from $\boldsymbol{u}$ is smaller than the quantitative semantics of φ at time 0 also satisfies φ.

Theorem 5.2 (Correctness). *Let φ be an STL formula, and $\boldsymbol{u}$ and $\boldsymbol{v}$ two traces.*

$$\begin{array}{r} \text{if } \boldsymbol{u} \models \varphi \text{ and } \mathrm{d}(\boldsymbol{u},\boldsymbol{v}) < [\![\varphi]\!]_{\boldsymbol{u}}(0) \text{ then } \boldsymbol{v} \models \varphi \\ \text{if } \boldsymbol{u} \not\models \varphi \text{ and } \mathrm{d}(\boldsymbol{u},\boldsymbol{v}) < -[\![\varphi]\!]_{\boldsymbol{u}}(0) \text{ then } \boldsymbol{v} \not\models \varphi \end{array}$$

On these grounds we now talk of $[\![\varphi]\!]_{\boldsymbol{w}}(t)$ as the *robustness estimate* at time t, and also call $[\![\varphi]\!]_{\boldsymbol{w}}$ the *robustness signal* of φ with respect to $\boldsymbol{w}$.

5.2.4 Until Rewrite

The equivalences $\varphi\, \mathrm{U}_{[a,b]}\, \psi \iff \Diamond_{[a,b]}\psi \wedge \varphi\, \mathrm{U}_{[a,+\infty)}\, \psi$ and $\varphi\, \mathrm{U}_{[a,+\infty)}\, \psi \iff \Box_{[0,a]}(\varphi\, \tilde{\mathrm{U}}\, \psi)$ granted by Proposition 3.1 extend from qualitative to quantitative semantics. Such equivalences constitute generalized de Morgan laws, in some way. It is indeed possible to show, according to our proof principle, that any equivalence between *negation-free* formulas that holds in the qualitative semantics, also holds in the quantitative semantics.

Proposition 5.3. *For any STL formulas φ and ψ, and constants $0 \leq a \leq b$ it holds*

$$\varphi \, \mathrm{U}_{[a,b]} \, \psi \iff \Diamond_{[a,b]} \psi \wedge \varphi \, \mathrm{U}_{[a,+\infty)} \, \psi \tag{5.1}$$

$$\varphi \, \mathrm{U}_{[a,+\infty)} \, \psi \iff \Box_{[0,a]} (\varphi \, \tilde{\mathrm{U}} \, \psi) \tag{5.2}$$

Proof. We only prove rule (5.1); rule (5.2) follows from the same argument. Fix φ and ψ two formulas, $\boldsymbol{w}$ a trace, and t a time in $\mathbb{T}$. Let c and d stand for robustness estimates of left-hand, and right-hand formulas of (5.1) at time t, respectively. By definition

$$\begin{aligned} c &= \sup_{r \in [t+a, t+b]} \min\{g(r), \inf_{s \in (t,r)} f(s)\} \\ d &= \min\{ \sup_{s \in [t+a,t+b]} g(s), \sup_{r \in [t+a,+\infty)} \min\{g(r), \inf_{u \in (t,r)} f(u)\}\} \end{aligned}$$

Suppose, in search of a contradiction, that $c \neq d$, for instance $c < d$. Now consider the trace $\boldsymbol{v}$ with two real signals defined as $x_{\boldsymbol{v}} = f$ and $y_{\boldsymbol{v}} = g$. Define the STL formulas α and β as follows:

$$\begin{aligned} \alpha &= (x > \tfrac{c+d}{2}) \, \mathrm{U}_{[a,b]} (y > \tfrac{c+d}{2}) \\ \beta &= \Diamond_{[a,b]} (y > \tfrac{c+d}{2}) \wedge (x > \tfrac{c+d}{2}) \, \mathrm{U}_{[a,+\infty)} (y > \tfrac{c+d}{2}) \end{aligned}$$

By commuting the sum of constant $-\frac{c+d}{2}$ with operations sup and inf, and factoring for min, we get $[\![\alpha]\!]_{\boldsymbol{v}}(t) = c - \frac{c+d}{2} < 0$ and $[\![\beta]\!]_{\boldsymbol{v}}(t) = d - \frac{c+d}{2} > 0$. Hence by Theorem 5.1 we have $(\boldsymbol{v}, t) \not\models \alpha$, and $(\boldsymbol{v}, t) \models \beta$ respectively. Yet according to Proposition 3.1 in the usual, qualitative semantics $\alpha \iff \beta$. Contradiction! Therefore $c = d$; we have shown that $[\![\varphi \, \mathrm{U}_{[a,b]} \, \psi]\!]_{\boldsymbol{w}}(t) = [\![\Diamond_{[a,b]} \psi \wedge \varphi \, \mathrm{U}_{[a,+\infty)} \, \psi]\!]_{\boldsymbol{w}}(t)$ for arbitrary time t and trace $\boldsymbol{w}$, thus Equation (5.1) holds. ■

Other cases of *until* rewrite for open, or semi-open intervals exposed in Section 3.2.2 also extend to quantitative semantics for the same reasons.

5.3 Piecewise Linear Decomposition

In a simulation trace, continuous-time signals are represented as sequences of values over a discrete temporal domain. The value of a real signal in between two consecutive samples is given by linear interpolation.[3] While real signals typically do not have discontinuities, that is not the case for Boolean signals. In turn robustness signals may also inherit these discontinuities; thus we consider the general case of piecewise linear signals, with zero or finitely many discontinuities.

Definition 5.4 (Piecewise Linear Signals). *A signal $f \in (\mathbb{R}_\infty)^T$ over some interval T is said to be* piecewise linear *if there exists a finite sequence $t_0, t_1, \ldots, t_n$ with $t_0 = \inf T$, $t_n = \sup T$ such that f is affine on segments (t_i, t_{i+1}) for all $i < n$.*

[3]For other interpolation orders, the property that interests us is that signals and their derivatives are piecewise monotone.

The sequence $(t_i)_{i\leq n}$ is called *time sequence* of f; we denote by $|f| = n+1$ the number of its elements, and also call it the *size* of signal f. Such a signal will be represented by its time sequence $t_0, t_1, \ldots, t_n$ and its value sequence

$$f(t_0), f(t_0^+), f(t_1^-), f(t_1), f(t_1^+), f(t_2^-), \ldots, f(t_n)$$

By extension we say that a trace is piecewise linear if all its signals are piecewise linear. In what follows, we assume piecewise linear traces.

The key property behind our approach, is that quantitative semantics preserves the piecewise linear property of signals.[4]

Theorem 5.5. *Let $\boldsymbol{w}$ be a piecewise linear trace. For any formula φ the robustness signal $[\![\varphi]\!]_w$ is also piecewise linear.*

The proof of Theorem 5.5 proceeds by induction on the formula structure. We treat the case of propositional operators, and expose the general principle for treating the case of temporal operators. The full analysis of temporal operators is postponed to the next section, which gives detailed algorithms for computing their robustness signals.

5.3.1 Propositional Operators

Fix $\boldsymbol{w}$ a piecewise linear trace, and φ, ψ some temporal formulas. Let us write $f = [\![\varphi]\!]_w$ and $g = [\![\psi]\!]_w$ their respective robustness signals.

For negation, we would like to compute a signal denoted h such that $h = [\![\neg\varphi]\!]_w = -[\![\varphi]\!]_w = -f$. Let $(t_i)_{i\leq n}$ be the time sequence of f. Over time sequence of f we have $h(t_i) = -f(t_i)$, $h(t_i^+) = -f(t_i^+)$, and $h(t_i^-) = -f(t_i^-)$ where defined. Negation commutes with the interpolation, $h = -[\varphi]_w$. The number of sampling points is unchanged, that is $|h| = |f|$.

For disjunction, we would like to compute a signal denoted h such that $h = [\![\varphi \vee \psi]\!]_w = \max\{[\![\varphi]\!]_w, [\![\psi]\!]_w\} = \max\{f, g\}$. We first build a sequence $(s_i)_{i\leq m}$ containing the sampling points of both f and g. Now for any time pair s_i, s_{i+1} such that $f(s_i^+) - g(s_i^+)$ and $f(s_{i+1}^-) - g(s_{i+1}^-)$ do not share the same sign, we add a new time point at the intersection of f and g in the segment (s_i, s_{i+1}). We denote by $(t_i)_{i\leq n}$ the resulting time sequence, such that h is linear over every interval (t_i, t_{i+1}). We have $h(t_i) = \max\{f(t_i), g(t_i)\}$, $h(t_i^+) = \max\{f(t_i^+), g(t_i^+)\}$, and $h(t_i^-) = \max\{f(t_i^-), g(t_i^-)\}$ where defined. There are less than $|f| + |g|$ additional points in the time sequence of h, and at most $|f| + |g|$ sampling points already present in f and g, so that we have $|h| \leq 4\max\{|f|, |g|\}$. The robustness computation for disjunction is illustrated in Figure 5.1.

5.3.2 Temporal Operators

We treat operator *until* by rewriting it into *untimed until* and *timed eventually*. This decomposition has been successfully applied to the classical, qualitative monitoring problem in Chapter 3. Let us focus on the case where the timing constraint is a closed

[4]In addition, *continuity* of signals is preserved by the sup and inf operations; thus traces featuring only continuous (real) signals yield a continuous robustness signal for any formula. Details on how to take advantage of this fact can be found in [45].

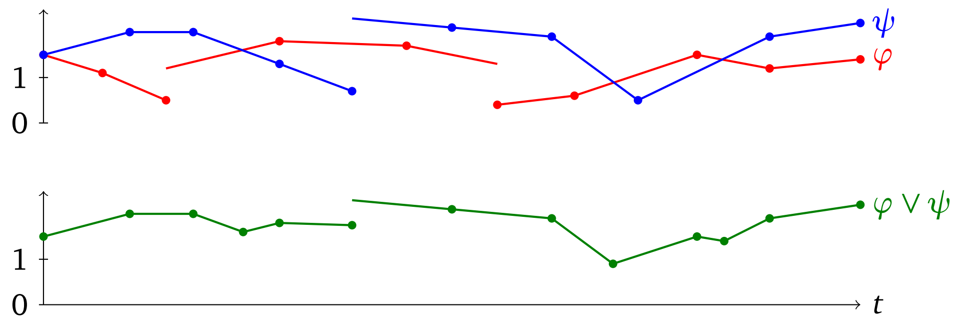


Figure 5.1: Computation of the robustness signal of $\varphi \vee \psi$, defined as the pointwise maximum of robustness signals of φ and ψ.

interval; open and semi-open intervals do not pose further difficulty. We have proved in Proposition 5.3 that the *until rewrite* rules also hold in the quantitative semantics. For an *unbounded until* operator $\mathrm{U}_{[a,+\infty)}$ we can use the rewrite rule (5.2) directly, whereas for *bounded until* we have in the quantitative semantics $\varphi \, \mathrm{U}_{[a,b]} \, \psi \Longleftrightarrow \Diamond_{[a,b]} \psi \wedge \Box_{[0,a]}(\varphi \, \tilde{\mathrm{U}} \, \psi)$ by chaining rewrite rules (5.1) and (5.2). Operator *always* being the dual of operator *eventually*, it only remains to treat the case of $\Diamond_{[a,b]}$ with $[a, b]$ a non-singular interval, and to provide an algorithm for the *untimed until*. We show in the coming section that these temporal operators preserve piecewise-linearity, while demonstrating how their semantics can be computed.

5.4 Algorithms

Algorithm 5.1 SIGNALS($\varphi, \boldsymbol{w}$)

```
  select $\varphi$
  case $\top, p$:
    return $t \mapsto +\infty \cdot (2[\varphi]_w - 1)$
  case $x > c$:
    return $x_w - c$
  case $\bullet\psi$:
    $f$ := SIGNALS($\psi, \boldsymbol{w}$)
    return COMBINE($\bullet, f$)
  case $\psi_1 \bullet \psi_2$:
    $f$ := SIGNALS($\psi_1, \boldsymbol{w}$)
    $g$ := SIGNALS($\psi_2, \boldsymbol{w}$)
    return COMBINE($\bullet, f, g$)
  end select
```

The main robustness computation procedure is given in Algorithm 5.4. It inputs a formula φ and a trace $\boldsymbol{w}$, and outputs a piecewise linear function that represents the robustness signal of φ relative to $\boldsymbol{w}$. For this the procedure uses a different routine

COMBINE to treat each operator, which compute the robustness signal of some formula based on those of its direct subformulas.

From the segment decomposition of operators $\neg$, and $\vee$ just described, we can immediately derive the corresponding COMBINE algorithms with time-complexity linear in the number of samples of input signals. By duality we also have an algorithm for $\wedge$ with the same property. We now give detailed algorithms for the remaining operators: U, and $\Diamond_I$ for I bounded interval. To this end we will need four basic operations on signals: *shift*, *pointwise maximum*, *pointwise minimum*, and *suffix supremum*. All such operations correspond to a robustness computation for simpler operators.

The shift of signal f by some constant $c \geq 0$ produces a signal g such that $g(t) = f(t+c)$ if f is defined at $t-c$, $-\infty$ otherwise. This can be achieved by shifting all sampling points and appending a segment with value $-\infty$. Note that the *shift* operation corresponds to the quantitative semantics of $\Diamond_{[a,a]}$. We denote its computation by $g :=$ COMBINE$(\Diamond_{[a,a]}, f)$ in pseudo-code.

The pointwise maximum (respectively minimum) of signals f and g produces a signal h such that $h(t) = \min\{f(t), g(t)\}$ (resp. $h(t) = \max\{f(t), g(t)\}$). This corresponds to the quantitative semantics of operator $\vee$ (respectively $\wedge$). We denote its computation by $h :=$ COMBINE$(\vee, f, g)$ (resp. $h :=$ COMBINE$(\wedge, f, g)$) in pseudo-code.

The suffix supremum of signal f is produces a signal g such that $g(t) = \sup_{t'>t} f(t')$. This can be achieved by backward induction, as follows. Assume that the temporal domain of f is of the form $T = [c, d]$. We let $g(d) = -\infty$, and $g(d^-) = \max\{f(d), f(d^-)\}$. Then for $s < t$ it holds $g(s) = \min\{g(t), \inf_{r \in (s,t)} f(r)\}$. Thus g can be represented using the same time sequence as f. Its values over this time sequence are computed going backward in time using the previous equations. Note that suffix supremum corresponds to the quantitative semantics of $\Diamond$. We denote its computation by $g :=$ COMBINE$(\Diamond, f)$ in pseudo-code.

Temporal operators are handled by applying such operations either over the whole input signals, or a subset thereof, possibly single linear segments. Given some signal f we write $f[t, t']$ the restriction of f to the interval $[t, t']$, and similarly for temporal domains that are open or semi-open intervals. We sometimes abuse the notation and also write $f(c)$ to denote a constant signal $t \mapsto f(c)$ over some temporal domain clear from the context.

5.4.1 Timed Eventually

Let f be the robustness signal of some formula φ with respect to some trace $\boldsymbol{w}$, with $(t_i)_{i \leq |f|}$ its time sequence. Let $I = [a, b]$ be a closed interval of $\mathbb{R}_{\geq 0}$ for some given $a < b$; open and semi-open intervals constraint can be handled similarly. We would like to compute $h = [\Diamond_I \varphi]_{\boldsymbol{w}}$, the robustness signal of $\Diamond_I \varphi$ with respect to $\boldsymbol{w}$, such that $h(t) = \sup_{t' \in t \oplus I} f(t')$ for all $t \in \mathbb{T}$. Let us first assume that f is a continuous signal, and postpone the analysis of signals with discontinuities.

As f is piecewise monotone, it is sufficient to consider candidates for the maximum in the time sequence of f, along with $t + a$ and $t + b$. Let us denotes by S the set of sampling points of f, in symbols $S = \{t_i : i \leq n\}$. We have

$$\sup_{s \in t \oplus [a,b]} f(s) = \max\{f(t+a), f(t+b), \max_{s \in S \cap (t+a, t+b)} f(s)\}$$

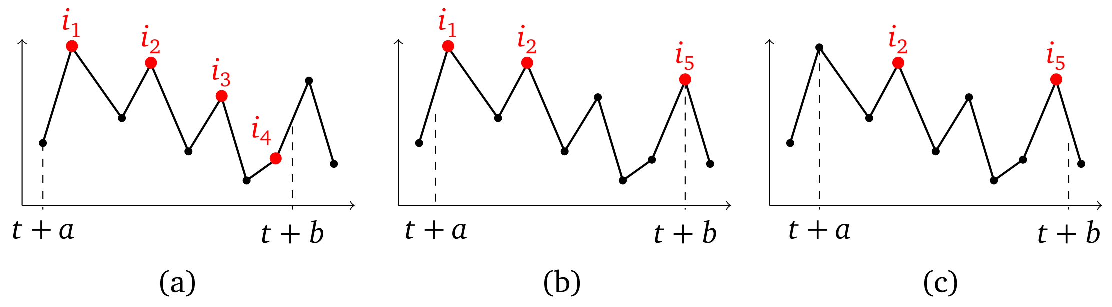


Figure 5.2: Three steps of Lemire's algorithm: (a) The set M contains samples i_1, i_2, i_3, and i_4; (b) New sample i_5 appears at the interval upper-bound $t+b$, and sample i_5 inserted in M, which causes i_4 and then i_3 to be removed; (c) The interval lower-bound $t+a$ reaches sample i_1, and sample i_1 is removed from M.

The problem of computing h is reduced to the computation of the maximum of the set $\{f(s) : s \in S \cap (t+a, t+b)\}$, followed by a pointwise maximum with both $f(t+a)$ and $f(t+b)$ where defined. On the one hand, time intervals where $S \cap (t+a, t+b)$ provide the maximum are "plateau" phases, where the supremum is reached at a point in the interior of the interval $t \oplus I$ and the derivative is constant. On the other hand, intervals where $t+a$ or $t+b$ provide the maximum are "descending" and "ascending" phases, where the slope follows that at some endpoint of $t \oplus I$.

The signal $t \mapsto \max_{s \in S \cap (t+a, t+b)} f(s)$ can be computed by a straightforward adaptation of the running maximum filter algorithm given in [79]. This work addresses the problem of computing, for discrete-time signals, the maximum over a shifting time window consisting of k elements. Lemire's algorithm is the first such algorithm with time complexity linear in the length of the sequence and *independent* of the window size k. We generalize this algorithm to the case of discrete-time signals with variable time-step, and in turn to piecewise linear signals.

The main idea is to maintain, as we increase t, a set of indices M, such that for each indice $i \in M$ the corresponding value $f(t_i)$ dominates all $f(t_j)$ for $j > i$ and t_j in $t \oplus (a, b)$. Formally we require that

$$i \in M \qquad \text{iff} \qquad t_i \in t \oplus (a,b) \text{ and } \forall j > i, \text{ if } t_j \in t \oplus (a,b) \text{ then } f(t_j) < f(t_i)$$

In particular for any given t, if $M \neq \emptyset$ we have $f(t_{\min M}) = \max\{f(t_i) : t_i \in t + (a, b]\}$. Assume M is known for a given time s, and is non-empty. We begin by finding the first $t > s$ such that either a new point appears at $t+b$, or some maximum candidate in M disappears at $t+a$, or both. We update M accordingly: if $t_{\min M} = t+a$ we remove $\min M$, the first index from M. Then if $t+b = t_i$ for a given i then we compare $f(t_i)$ with $f(t_k)$ for $k \in M$ in decreasing order of k, starting with the last candidate $f(t_{\max M})$. If $k \in M$ is such that $t_k \leq t_i$ then t_k is removed from M as dominated by t_i, otherwise we stop. At this point i is inserted as the new last element of M. We now have in M an ordered set of maximum candidates in $t \oplus I$; we can output $f(t_{\min M})$ and repeat the procedure for the next event. The algorithm steps are illustrated in Figure 5.4.1.

The pseudo-code of Algorithm 5.2 exposes in detail our generalization of Lemire's algorithm. For simplicity sake we ignore the case where $M = \emptyset$ (occurs if $[a, b]$ is shorter than the delay between two consecutive sampling points).

Algorithm 5.2 LEMIRE$(f, (t_i)_{i \leq n}, a, b)$

```
$s := t_0 - b,\ t := s,\ i := 0,\ M := \{0\}$
while $t + a < t_n$ do
  $t := \min\{t_{\min M} - a, t_{i+1} - b\}$
  if $t = t_{\min M} - a$ then
    $M := M \setminus \{\min M\}$
    $g(t) := f(t_{\min M})$
    $s := t$
  end if
  if $t = t_{i+1} - b$ then
    while $f(t_{i+1}) \geq f(t_{\max M})$ and $M \neq \emptyset$ do
      $M := M \setminus \{\max M\}$
    end while
    $M := M \cup \{i + 1\}$
    $i := i + 1$
  end if
  if $s \geq t_0$ then
    $g(s, t) := f(t_{\min M})$
  end if
end while
return $g$
```

Proposition 5.6. *The time-complexity of Algorithm 5.2 is linear in* $|f|$.

Proof. We store M as a doubly-linked queue, that we keep sorted in increasing index order; the elements of M can be this way be accessed with cost $O(1)$. The cost of maintaining M becomes proportional to the number of value comparisons involved. With the same argument as [79] we notice that on the whole run, there are $|f|$ elements entering M and thus $|f|$ elements are leaving M. Each time the comparison $f(t_{i+1} - b) \geq f(t_{\max M})$ evaluates to *true*, an element is removed from M so there can only be $|f|$ such comparisons that evaluate to *true*. When it evaluates to *false* we leave the inner *while* loop, hence there are at most $|f|$ such comparisons that evaluate to *false*. Over the whole execution Algorithm 5.3 uses at most $2 \cdot |f|$ such value comparisons, making its execution time linear in $|f|$. ■

We obtain the full algorithm, for piecewise linear signals, by integrating this value with $f(t + a)$ and $f(t + b)$. Now if signal f has discontinuities, we use the same principle to compute the running maximum of the left and right limits of f that lie inside $t \oplus (a, b)$. Let us denote g (resp. g^+, g^-) signal whose value at time t is determined by the maximum of f at sampling points in S (resp. right-limits in S^+, left-limits in S^-) in the window $t \oplus I$. We also denote f_a (respectively f_b) the signal f shifted by a (respectively b). The robustness signal h is computed as the pointwise maximum of g, g^+, g^-, f_a, and f_b. The pseudo-code appears in Algorithm 5.3.

Lemma 5.7. *The time-complexity of Algorithm 5.3 is in* $O(|f|)$.

Algorithm 5.3 COMBINE($\Diamond_{[a,b]}, f$)

```
$f_a :=$ COMBINE($\Diamond_{[a,a]}, f$)
$h := f_a$
$f_b :=$ COMBINE($\Diamond_{[b,b]}, f$)
$h :=$ COMBINE($\vee, f_b, h$)
$g :=$ LEMIRE($f, (t_i)_{i \leq n}, a, b$)
$h :=$ COMBINE($\vee, g, h$)
$g^+ :=$ LEMIRE($f, (t_i^+)_{i<n}, a, b$)
$h :=$ COMBINE($\vee, g^+, h$)
$g^- :=$ LEMIRE($f, (t_i^-)_{1 \leq i \leq n}, a, b$)
$h :=$ COMBINE($\vee, g^-, h$)
return $h$
```

Proof. Notice that the computation of signals f_a, f_b, g, g^+, and g^- use algorithms with time-complexity linear in $|f|$. The computation of pointwise maximum also has time-complexity linear in the input size, thus every signal in the algorithm has a size linear in $|f|$, and there is a fixed number of signals. ■

5.4.2 Untimed Until

Let $f = [\![\varphi]\!]_w$ and $g = [\![\psi]\!]_w$ be the robustness signals of some formulas φ and ψ relative to some trace $\boldsymbol{w}$, and let $(t_i)_{i \leq |f|}$ and $(t'_i)_{i \leq |g|}$ be their respective time sequences. We would like to compute $h = [\![\varphi \,\mathrm{U}\, \psi]\!]_w$, the robustness signal of $\varphi \,\mathrm{U}\, \psi$ relative to $\boldsymbol{w}$. By definition we have $h(t) = \sup_{r \in (t,+\infty)} \min\{g(r), \inf_{u \in (t,r)} f(u)\}$. The computation can be done by backward induction, similarly to the case of qualitative semantics as exposed in Section 3.2.3. In what follows we give a quantitative analogue to propositions 3.2 and 3.3.

First, we observe that following its definition, the robustness signal of an *until* formula is right-continuous. We have $h(t) = h(t^+)$ for all $t \in \mathbb{T} \setminus \{\sup \mathbb{T}\}$.

Second, let $s < t$ be two times in $\mathbb{T}$, and define the signal $h_t(s)$ as follows:

$$h_t(s) = \sup_{r \in (s,t]} \min\{g(r), \inf_{u \in (s,r)} f(u)\}$$

According to general properties of sup and inf, we obtain the following inductive formula:

$$h(s) = \max\left\{h_t(s), \min\{\inf_{u \in (s,t)} f(u), f(t), h(t)\}\right\}$$

Suppose that f is linear on the interval $[s,t)$. We have two possibilities according to whether f is decreasing or increasing on that interval. If $f(s^+) \geq f(t^-)$ then $\inf_{u \in (s,r)} f(u) = f(r^-) = f(r)$ for all $r \in (s,t)$. In this case, we have:

$$h_t(s) = \sup_{r \in (s,t]} \min\{g(r), f(r)\} \qquad h(s) = \max\left\{h_t(s), \min\{f(t^-), f(t), h(t)\}\right\}$$

Otherwise $\inf_{u \in (s,r)} f(u) = f(s^+) = f(s)$ for all $r \in (s,t)$. In that case, we have:

$$h_t(s) = \min\{f(s), \sup_{r \in (s,t]} g(r)\} \qquad h(s) = \max\left\{h_t(s), \min\{f(s), f(t), h(t)\}\right\}$$

We let $t = t_i$ in the above, and obtain a characterization of the value of h on the segment (t_i, t_{i+1}) as a function of s. Computation can now be expressed as operations previously discussed. The pseudo-code detailing such operations appears in Algorithm 5.4.

Algorithm 5.4 COMBINE(U, f, g)

```
$i := |f| - 1$
$h(t_{i+1}) = -\infty$
while $i \geq 0$ do
    if $f(t_i^+) \geq f(t_{i+1}^-)$ then
        $h_1 :=$ COMBINE($\Diamond, g(t_i, t_{i+1})$)
        $h_2 :=$ COMBINE($\wedge, h_1, f(t_i, t_{i+1})$)
        $h_3 := \min\{f(t_{i+1}^-), f(t_{i+1}), h(t_{i+1})\}$
        $h(t_i, t_{i+1}) :=$ COMBINE($\vee, h_2, h_3$)
    else
        $h_1 :=$ COMBINE($\wedge, g(t_i, t_{i+1}), f(t_i, t_{i+1})$)
        $h_2 :=$ COMBINE($\Diamond, h_1$)
        $h_3 := \min\{f(t_{i+1}), h(t_{i+1})\}$
        $h_4 :=$ COMBINE($\wedge, f(t_i, t_{i+1}), h_3$)
        $h(t_i, t_{i+1}) :=$ COMBINE($\vee, h_2, h_4$)
    end if
    $h(t_i) := h(t_i^+)$
    $i := i - 1$
end while
return $h$
```

Lemma 5.8. *The time-complexity of Algorithm 5.4 is in* $O(\max\{|f|, |g|\})$.

Proof. The algorithm takes $|f|$ steps. Each step i computes the segment of signal h over the interval $[t_i, t_{i+1})$ from segments $f(t_i, t_{i+1}]$ and $g(t_i, t_{i+1}]$, and $h(t_{i+1})$. The execution of a single step takes time linear in the sum of the size of the input segments; the size of inputs summed over all steps is less than $2|f| + |g|$. Thus the total execution time is linear in $\max\{|f|, |g|\}$. ■

5.5 Evaluation

5.5.1 Complexity

We are interested in the time complexity of the robustness computation relative to both the trace size and the formula size. The size of a trace $\boldsymbol{w}$ is defined as $|\boldsymbol{w}| = \sum_{q \in \mathbb{P} \cup \mathbb{X}} |q_w|$, the sum of all its signals sizes. The size of a formula φ, denoted $|\varphi|$, is the number nodes in its syntactic tree.

Theorem 5.9. *For any formula φ and trace $\boldsymbol{w}$, the signal $[\![\varphi]\!]_w$ has a size in* $2^{O(|\varphi|)} \cdot |\boldsymbol{w}|$

Proof. From lemmas 5.7 and 5.8, and remarks of Section 5.3 we deduce the following. Given any signals f, g, and h such that either $h : t \mapsto -f(t)$, $h : t \mapsto \min\{f(t), g(t)\}$

or $h : t \mapsto \sup_{r \in t+[a,b]} \min\{g(r), \inf_{u \in (t,r)} f(u)\}$, there exists a constant $c > 1$ such that $|h| \leq c \max\{|f|, |g|\}$. We now prove the property $|\,[\![\varphi]\!]_w\,| \leq c^{|\varphi|} \cdot |w|$ by structural induction. Atomic formulas have size 1, while their robustness signal has a number of sampling points at most the size of the input trace. For the negation, we have $|\neg\varphi| = |\varphi| + 1$ while $|\,[\![\neg\varphi]\!]_w\,| = |\,[\![\varphi]\!]_w\,|$, and we conclude using the inductive hypothesis. For binary operations, consider formula $\gamma = \varphi \wedge \psi$ or $\gamma = \varphi \,\mathrm{U}\, \psi$; in either case $|\gamma| = |\varphi| + |\psi| + 1$. Let f, g and h be the robustness signals relative to w of φ, ψ and γ respectively. By induction hypothesis, $|f| \leq c^{|\varphi|} \cdot |w|$, and $|g| \leq c^{|\psi|} \cdot |w|$. We have $|h| \leq c(c^{|\varphi|} \cdot |w| + c^{|\psi|} \cdot |w|) = c^{|\varphi|+|\psi|+1} \cdot |w| = c^{|\gamma|} \cdot |w|$. ■

Corollary 5.10. *The algorithm* SIGNALS(φ, w) *has time-complexity in* $2^{O(|\varphi|)} \cdot |w|$.

Proof. The complexity results of previous sections can be summed up by stating the following: there exists a constant c such that for any signals f, g and any operator $\neg, \wedge$ or U_I, the corresponding COMBINE algorithm takes execution time at most $c(|f| + |g|)$. Now let φ be an arbitrary formula, and w an arbitrary trace. By Theorem 5.9, each subformula ψ of φ has a robustness signal with at most $c^{|\psi|} \cdot |w| \leq c^{|\varphi|} \cdot |w|$ sampling points. Thus for each subformula ψ of φ, algorithm SIGNALS(ψ, w) algorithm takes execution time linear in $c^{|\varphi|} \cdot |w|$. There are exactly $|\varphi|$ such nodes, so that the main robustness computation of φ with respect to w is linear in $|\varphi| \cdot c^{|\varphi|} \cdot |w|$, and linear in $2^{d|\varphi|} \cdot |w|$ for any $d > \log(c)$. ■

The complexity of robust (quantitative) monitoring, like that of classical (qualitative) monitoring, is linear in the size of the input trace. However the exponential complexity of robust monitoring relative to the size of the formula contrasts with that of classical monitoring, which is only quadratic. This is explained by the fact that the size of the satisfaction signal of some formula is at most the sum of that of its direct subformulas. For robust monitoring, the size of the robustness signal of some formula may be greater than the sum of that of its direct subformulas.

5.5.2 Experiments

The proposed algorithm has been implemented in C++ and interfaced with the STL parser of Breach, a Matlab toolbox for the testing and verification of hybrid system. We computed the robustness estimate for three formulas of size 1, 25 and 50 for random signals of varying size. We could confirm that for each formula, the computation time is linear with respect to the signal size, processing on average 42735, 6135, and 4081 input samples per second respectively. We then set the signal size to 1000, and generated formulas of varying size. For each pair of formula and signal, we computed the robust signal, and the ratio between its size and that of the input signal. We observed a high variability of this ratio, and could not draw definitive conclusions, other than an exponential behavior in the *height* of the formula's syntactic tree. We believe that human-written specifications are unlikely to contain the deeply nested temporal operators that would be necessary to yield this exponential behavior.

Next we compared the performance of our algorithm, as implemented in Breach, with that of the DP-TaLiRo [54], based on a dynamic programming approach and implemented in S-TaLiRo version 1.3. In particular we compared the monitoring of $\Diamond_I p$ and $p \,\mathrm{U}_I\, q$

relative to traces of various sizes, and for different time intervals I. I. The results are given in Table 5.1. Except for signals of small size, our algorithm as implemented in Breach is consistently faster by several orders of magnitude.

Table 5.1: Computation time (s) of robustness estimates for eventually and until formulas; comparison with DP-Taliro algorithms.

	DP-Taliro				Breach			
$\lvert \boldsymbol{w} \rvert$	10^2	10^3	10^4	10^5	10^2	10^3	10^4	10^5
$\Diamond_{[1,2]}$	0.00109	0.00278	0.176	16.6	0.00312	0.00302	0.004	0.0193
$\Diamond_{[1,11]}$	0.00068	0.00304	0.212	20.4	0.00286	0.00262	0.00391	0.0173
$\Diamond_{[1,21]}$	0.00071	0.00334	0.253	24.3	0.00268	0.00269	0.00412	0.0185
$\Diamond_{[1,31]}$	0.00070	0.0038	0.278	27.3	0.00302	0.00281	0.00409	0.0208
$\mathrm{U}_{[1,2]}$	0.523	4.72	46.8	486	0.00577	0.00766	0.0268	0.228
$\mathrm{U}_{[1,11]}$	0.482	4.55	47.1	493	0.00567	0.00743	0.0269	0.223
$\mathrm{U}_{[1,21]}$	0.468	4.59	46.2	499	0.00545	0.00722	0.0268	0.229
$\mathrm{U}_{[1,31]}$	0.462	4.7	46.7	505	0.00567	0.0073	0.0274	0.222

One partial explanation could be the fact that in the framework of TaLiRo, the robust satisfaction of a predicate is obtained through the computation of a distance function, which is done by the monitoring algorithm. This may lead to an additional cost [54], whereas in our case, that computation is separated from the monitoring. However, all the predicates in our experiments are such that the distance function should be trivial to compute, so this alone cannot account for the difference in performance. Also, our results confirmed that the computation time for bounded time operators does not depend on the size of the time interval, as in [79]. This dramatically improves upon the observed complexity of the DP-TaLiRo algorithm.

6

Regular Expressions Monitoring

This chapter solves the problem of monitoring Timed Regular Expressions (TRE). These expressions are formed by adding to the syntax of classical regular expressions an operator that constrains the *duration* of subexpressions. This extensions enables specifying mixed-signal behaviors in a natural way, particularly those where timing aspects interact with other non-trivial sequential conditions. In the context of regular expressions, the notion of satisfaction signal used for temporal logic is not suitable for an inductive monitoring procedure. Semantics of regular expressions are formulated in terms of trace segments, which are represented for a given trace by pairs of *start* and *end* times. Instead of simple *membership*, we solve the more general problem of *pattern matching*, which consists in computing the entire set of matches (the match set) of a given expression on some trace. Seeing segments as points in a two dimensional space, we show that the match set admits a finite decomposition into simple convex polyhedra using vertical, horizontal and diagonal constraints (elsewhere called *zones*). The performance of the resulting TRE monitoring procedure appears comparable to that of MTL.

6.1 Introduction

6.1.1 Motivation

Timed Regular Expressions were introduced in [24, 25] as a language (and theoretical tool) for the specification (and study) of continuous-time behaviors. Their expressiveness matches that of Timed Automata [17], if renaming of action symbols is authorized. Thus TRE provide a strong theoretical basis for the specification of continuous-time behaviors. Note that TRE contain classical regular expressions, and thus enable the specification of *modulo counting* properties, which cannot be expressed using temporal logic unless one resorts to some form of second order quantification.

A practical advantage of using regular expressions instead of temporal logic, is found in the ease with which they enable to describe behaviors with a combination of timing and sequential aspects. Consider the property, according to which p holds for some time in I, followed by q holding for some time in J. It can be specified with the MTL formula $\varphi = p\,\mathrm{U}_I(q\,\mathrm{U}_J\,\top)$ and the TRE $\rho = \langle\underline{p}\rangle_I \cdot \langle\underline{q}\rangle_J$. Imagine that we aim to further specify that after this sequence, proposition r should also be holding for some time in interval K. To achieve this with MTL, we must write $\varphi' = p \, \mathrm{U}_I(q \, \mathrm{U}_J(r \, \mathrm{U}_K \top))$. Note that operator *until* is not associative, and in particular φ is not a subformula of φ'. Using a TRE, we can write $\rho' = \langle \underline{p} \rangle_I \cdot \langle \underline{q} \rangle_J \cdot \langle \underline{r} \rangle_K = \rho \cdot \langle \underline{r} \rangle_K$, which bears no ambiguity, as concatenation is associative. We can also place further constraints on the duration of subparts of the sequence, with for example $\langle \langle \underline{p} \rangle_I \cdot \langle \underline{q} \rangle_J \rangle_H \cdot \langle \underline{r} \rangle_K$. Properties of this kind, with constraints ranging over several events or actions, cannot in general be expressed using MTL [32].

6.1.2 Our Approach

The algorithm we propose to monitor TRE follows a similar principle to the monitoring of MTL by interval marking. It proceeds by computing recursively the satisfaction of direct subexpressions and, given some algorithm for each operator, can deduce the satisfaction of any expression. Satisfaction sets however are not an appropriate representation for the purpose of monitoring regular expressions.

Fix the temporal domain $\mathbb{T} = [0, d]$. Define for argument's sake the satisfaction set of some expression φ relative to $\boldsymbol{w}$, as the set of times t such that $(\boldsymbol{w}, t, d) \models\equiv \varphi$. Let $\boldsymbol{w}$ be a trace with Boolean variable p such that $p_{\boldsymbol{w}} = [0, c)$ for some c with $0 < c < d$, and consider the expressions $\varphi = \underline{p} \cdot \underline{\neg p}$ and $\psi = \underline{p} \cdot (\epsilon \vee \underline{\neg p})$. The satisfaction sets of expressions φ and ψ are both equal to $[0, c)$. However the satisfaction set of $\varphi \cdot \varphi$ is empty, while that of $\psi \cdot \psi$ is equal to $[0, c)$. Hence that notion of satisfaction set is not sufficient to decide satisfaction of TRE inductively: it does not distinguish φ from ψ, yet it distinguishes $\varphi \cdot \varphi$ from $\psi \cdot \psi$.

For some expression φ, we propose instead to explore the set of pairs (t, t') such that the segment of the trace under consideration matches φ between t and t'. We call this subset of $\mathbb{T}^2$ the *match set* of φ. Note that the match set is potentially uncountable, as the temporal domain is an interval of the reals. We show that the match set of some TRE can always be decomposed into *zones*, simple convex sets already known in timed automata formal verification [16]. We observe that every regular operation corresponds to a simple operation on match sets, and in particular timing operators $\langle \, \rangle_I$ specific to TRE correspond to an intersection with a single diagonal zone. The computation of match sets then proceeds by structural induction as with MTL for satisfaction sets. Using this two-dimensional representation, the cost of certain binary regular operations like concatenation becomes quadratic, resulting in a potentially exponential complexity in the length of the trace. Experiments seem to indicate that a suitable ordering of zones prevents such behaviors from occurring in practice.

In order to solve the *membership* problem, that is to decide whether $\boldsymbol{w} \models\equiv \varphi$ defined as $(\boldsymbol{w}, 0, d) \models\equiv \varphi$, we indeed solve the *pattern matching* problem, that is to find all segments (t, t') such that $(\boldsymbol{w}, t, t') \models\equiv \varphi$. An interesting application of our algorithms is the monitoring of timed assertions in which TRE ρ can appear in temporal formulas φ via suffix implication $\rho \circ\!\!\rightarrow \varphi$. We can combine our pattern matching and temporal monitoring algorithms to solve this problem. This demonstrates the feasibility of providing timed operators in digital assertions, interpreted in continuous-time.

6.1.3 Related Work

Monitoring procedures for *temporal logic* formulas are well-studied and have been extended successfully to continuous-time with STL [83, 86] notably. An extension of STL to *freeze quantification* is defined in [43]; semantic definitions of this extension require an additional time parameter. The set of time pairs that satisfy such a formula is computed using a two-dimensional representation akin to ours, but which makes use of a decomposition into arbitrary polyhedra. The work of [12] proposes an application of TRE to mixed-signal specification, with the notion of feature-indented assertions, in which no Kleene star may be used and all duration constraints apply to atomic expressions. An interval-based algorithm for monitoring such expressions is presented there. Standard assertion languages used in the semi-conductor industry such as PSL [48, 38] and SVA [113, 106] combine temporal logic with regular expressions. We extend monitoring procedures toward timed extensions of such specification languages, similar to that of [63].

Pattern matching is a fundamental operation in searching over texts and elsewhere. The basic string matching algorithms for single words [71, 34] as well as specialized data structures such as suffix trees [116] and later advancements can be found in [107, 40]. Regular expressions variants are implemented in many software tools ranging from the *grep* [109] family to modern programming languages, notably Perl and Python. Besides texts, pattern matching has important applications in Biology [14] and in database querying, especially temporal databases [55]. A straightforward application of the classical translations of regular expressions to automata can be used to detect whether the *prefix* of a string matches the pattern. The classical algorithm of Thompson [109] adapts the automaton construction for pattern matching but still, in the discrete case, finding *all* matches of an expression in a string is considered a very difficult problem and is not part of the mainstream; some exceptions are [100] and [78]. One reason might be that without a symbolic representation, which is necessary for the timed case, the set of matches may be prohibitively large to represent.

6.2 Timed Regular Expressions

6.2.1 Syntax and Semantics

Let us recall and specialize the definitions of Chapter 2 as follows. We fix a temporal domain $\mathbb{T} = [0, d]$ and a set of propositions $\mathbb{P}$. We consider atomic expressions of the form $\underline{p}$, standing for a trace segment such that proposition p holds everywhere. Timing constraints are assumed to have finite resolution; for the sake of simplicity we assume that all time constants in the specification are given as integers (rationals can be handled by scaling). We say that some an interval I of $\mathbb{R}_{\geq 0}$ is *integer-bounded* when it verifies $\inf I, \sup I \in \mathbb{N}$. The syntax of Timed Regular Expressions is given by the following grammar:

$$\varphi ::= \underline{p} \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \cdot \varphi \mid \varphi^* \mid \langle \varphi \rangle_I$$

where p is a proposition in $\mathbb{P}$, and I is an integer-bounded interval.

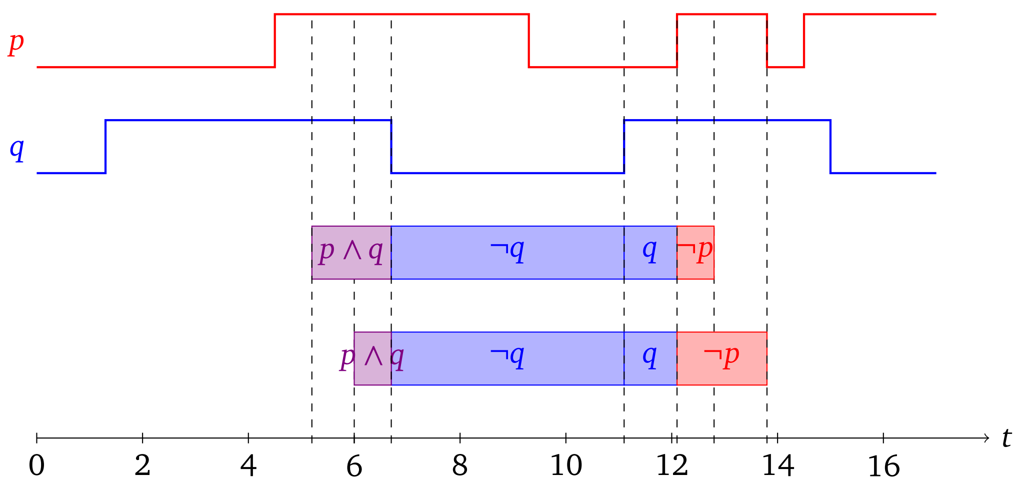


Figure 6.1: Two matches of expression $\langle \underline{p \wedge q} \cdot \underline{\neg q} \cdot \underline{q} \cdot \underline{p} \rangle_{[5,10]}$ on a given simulation trace.

A trace $\boldsymbol{w}$ is a valuation of propositions in $p \in \mathbb{P}$ as Boolean signals p_w over the temporal domain $\mathbb{T}$. In the context of some trace $\boldsymbol{w}$, atomic expressions $\underline{p}$ are interpreted as predicates $\underline{p}_w$ over $\mathbb{T}^2$, representing segments of $\boldsymbol{w}$ that match $\underline{p}$. Expression $\underline{p}$ is matched by segments such that p holds continuously in the corresponding open interval of time. We let $(t, t') \in \underline{p}_w$ if and only if $p_w[t''] = 1$ for all $t \in (t, t')$.

We say that trace $\boldsymbol{w}$ *matches* expression φ between times $t \leq t'$ in $\mathbb{T}$, denoted $(\boldsymbol{w}, t, t') \models \varphi$, when the following inductive definitions hold:

$$
\begin{array}{lll}
(\boldsymbol{w}, t, t') \models \epsilon & \text{iff} & t' = t \\
(\boldsymbol{w}, t, t') \models \underline{p} & \text{iff} & (t, t') \in \underline{p}_w \\
(\boldsymbol{w}, t, t') \models \varphi_1 \vee \varphi_2 & \text{iff} & (\boldsymbol{w}, t, t') \models \varphi_1 \text{ or } (\boldsymbol{w}, t, t') \models \varphi_2 \\
(\boldsymbol{w}, t, t') \models \varphi_1 \wedge \varphi_2 & \text{iff} & (\boldsymbol{w}, t, t') \models \varphi_1 \text{ and } (\boldsymbol{w}, t, t') \models \varphi_2 \\
(\boldsymbol{w}, t, t') \models \varphi_1 \cdot \varphi_2 & \text{iff} & \exists t'', (\boldsymbol{w}, t, t'') \models \varphi_1 \text{ and } (\boldsymbol{w}, t'', t') \models \varphi_2 \\
(\boldsymbol{w}, t, t') \models \varphi^* & \text{iff} & (\boldsymbol{w}, t, t') \models \epsilon \text{ or } (\boldsymbol{w}, t, t') \models \varphi \cdot \varphi^* \\
(\boldsymbol{w}, t, t') \models \langle \varphi \rangle_I & \text{iff} & t' - t \in I \text{ and } (\boldsymbol{w}, t, t') \models \varphi
\end{array}
$$

Technically speaking, $\models$ is the least such relation, see Footnote 6 of Section 2.3.2. We say that $\boldsymbol{w}$ matches φ, and write $\boldsymbol{w} \models \varphi$ when $(\boldsymbol{w}, 0, d) \models \varphi$ given $\mathbb{T} = [0, d]$ the temporal domain of $\boldsymbol{w}$. An example of expression and some of its matches on a given signal appear in Figure 6.1.

6.2.2 Match Set

Strictly speaking the monitoring problem for TRE asks, for a given trace $\boldsymbol{w}$ and expression φ, whether $\boldsymbol{w} \models \varphi$. In order to reason inductively about the validity of expression, the suitable notion is that of *match set*, defined as follows.

Definition 6.1. *The match set of φ according to $\boldsymbol{w}$, denoted $[\varphi]_w$ is the set of segments of $\boldsymbol{w}$ that match φ:*

$$[\varphi]_w = \{(t, t') \in \mathbb{T}^2 : (\boldsymbol{w}, t, t') \models \varphi\}$$

To compute the match set of some expression φ, it is sufficient to simply know the match sets of direct subexpressions of φ. Clearly knowing the match set of φ enables to decide its satisfaction by some trace $\boldsymbol{w}$, as by definition $\boldsymbol{w} \models\!\equiv \varphi$ if and only if $(0, d) \in [\varphi]_{\boldsymbol{w}}$. In the course of solving the simple monitoring problem for TRE, that is deciding if $\boldsymbol{w} \models\!\equiv \varphi$ for some trace $\boldsymbol{w}$ and expression φ, we will solve as a by-product a more general problem, that of pattern matching. The pattern matching problem aims at finding all the matches of expression φ on $\boldsymbol{w}$, that is the entire match set $[\varphi]_{\boldsymbol{w}}$.

6.3 Zone Marking

Central to our approach is a decomposition of the match set into a finite union of *zones*. We define *zones* as a class of convex polyhedra given by constraints of the form $t_i \bowtie a_i$, $t_i \bowtie b_i$, and $t_i - t_j \bowtie c_{i,j}$ with $\bowtie \in \{<, \leq, \geq, >\}$. By defining 0 as a time constant, zones can be viewed as a special case of formulas in the logical theory of difference constraint, where all atomic formulas are of the form $t_i - t_j \bowtie c_{i,j}$. Such a theory admits quantifier elimination [72].

Zones were introduced (and are still used extensively) to represent real-time clock values in the analysis of timed automata [16]. In the monitoring context we use them to represent absolute time values in a match set. On the one hand diagonal constraints arise directly from timing constraint in the specification, and can be taken as integers (modulo scaling). On the other hand orthogonal constraints may have non-integral parts inherited from the time stamps of events in $\boldsymbol{w}$.

We denote by $\pi_1(Z)$, $\pi_2(Z)$ and $\pi_{2,1}(Z)$ the orthogonal projections of a given zone Z on horizontal, vertical, and diagonal axes, respectively. These are intervals with respective endpoints written $\pi_1^-(Z)$, $\pi_1^+(Z)$, $\pi_2^-(Z)$, $\pi_2^+(Z)$, $\pi_{2,1}^-(Z)$, and $\pi_{2,1}^+(Z)$. Typically we have

$$(t, t') \in Z \quad \text{if and only if} \quad \left\{ \begin{array}{rcl} \pi_1^-(Z) \leq & t & \leq \pi_1^+(Z) \\ \pi_2^-(Z) \leq & t' & \leq \pi_2^+(Z) \\ \pi_{2,1}^-(Z) \leq & t' - t & \leq \pi_{2,1}^+(Z) \end{array} \right.$$

We also consider zones that are open or part-open with the same canonical representation yet featuring strict inequalities.[1]

We now prove the central result of this chapter, according to which the match set can be represented as a finite union of zones. Since operations $\wedge$, $\langle\rangle_I$ and $\cdot$ distribute over $\vee$, similarly to the temporal logic case, it is sufficient to prove closure of zones under their associated match set operations, and the closure of unions of zones will immediately follow. The closure is straightforward to obtain, except in the case of starred expressions φ^* where we show that the match set of φ^* is computed in terms of a finite number of iterations of φ. The convergence to a fixed point $\bigcup_{n=0}^{k+1} [\varphi^n]_w = \bigcup_{n=0}^{k} [\varphi^n]_w$ can be intuitively understood as follows. There are finitely many time constants in the trace and expression (and finitely many combinations thereof), thus there are finitely many time constants in the zone constraints encountered while computing the closure. However a bound obtained by an argument along these lines would be overly pessimistic.

[1]This can be prevented by slight modifications of our definitions, if only closed intervals are allowed in timing constraints. However strict constraints do not create a particular difficulty.

It may only be used to show that the complexity relative to the expression is linear. What interests us most is the behavior of the algorithm relative to the length of the trace. In what follows we will use other proofs to compute more relevant bounds on the number of iterations in the computation of the match set of starred expressions.

Theorem 6.2 (Match Set Decomposition). *Given a finite variability signal $\mathbf{w}$ and a TRE φ, the set $[\varphi]_w$ is a finite union of zones.*

For the empty word, notice that the match set of ϵ is the identity relation $Id = \{(t,t') \in \mathbb{T}^2 : t = t'\}$ over $\mathbb{T}$. Moreover this relation is a zone, hence the "decomposition" of ϵ is given by

$$[\epsilon]_w = Id$$

In the case of atomic expressions $\underline{p}$, the match set is a disjoint union of triangles touching the diagonal, see Figure 6.2-(a). Assuming rising edges and falling edges of p occur at times $(r_i)_{i\leq n}$ and $(s_i)_{i\leq n}$ respectively, we have

$$[\underline{p}]_w = \bigcup_{i=0}^{n} \{(t,t') : r_i \leq t < t' \leq s_i\}$$

For Boolean operations, the match sets for disjunction and conjunction satisfy

$$[\varphi \vee \psi]_w = [\varphi]_w \cup [\psi]_w \qquad\qquad [\varphi \wedge \psi]_w = [\varphi]_w \cap [\psi]_w$$

and finite unions of zones are closed under Boolean operations. The match set of an expression φ restricted in its duration to interval I is obtained as follows.

$$[\langle\varphi\rangle_I]_w = [\varphi]_w \cap \{(t,t') \in \mathbb{T}^2 : t' - t \in I\}$$

This is just an intersection with a zone and the result remains a union of zones, see Figure 6.2-(b).

In the case of concatenation, thanks to our semantics formulation we can see directly that the match set for concatenation is a relational composition of the corresponding match sets. Let $R, S \subseteq \mathbb{T}$ be two binary relations over $\mathbb{T}$, also known as predicates over $\mathbb{T}^2$ elsewhere in the text. We denote by $R \mathbin{⨟} S = \{(t,t') : \exists t'', (t,t'') \in R \text{ and } (t'',t') \in S\}$ the composition of binary relations R and S. Following definitions it holds

$$[\varphi \cdot \psi]_w = [\varphi]_w \mathbin{⨟} [\psi]_w$$

as illustrated in Figure 6.3-(a). It turns out that the zones are well behaved relative to composition, which is one of the two key lemmas to get our result:

Lemma 6.3. *The composition of two zones is a zone.*

Proof. Let $\alpha[t,t']$ and $\beta[t,t']$ be conjunctions of difference constraints defining zones X and Y respectively. From the definition or relational composition, the set $Z = X \mathbin{⨟} Y$ is characterized by the formula $\gamma = \exists t'', \alpha[t,t''] \wedge \beta[t'',t']$. Eliminating t'' from γ using the Fourier-Motzkin procedure we get an equivalent, quantifier-free formula. The elimination proceeds by comparing the lower bounds and upper bounds on variable t'', having rewritten $t'' - t \bowtie c$ as $t'' \bowtie c + t$ and symmetrically. The resulting constraints can be written back in the form $t \bowtie a$, $t' \bowtie b$, and $t' - t \bowtie c$. Thus γ is equivalent to a conjunction of difference constraints, hence Z is a zone. ∎

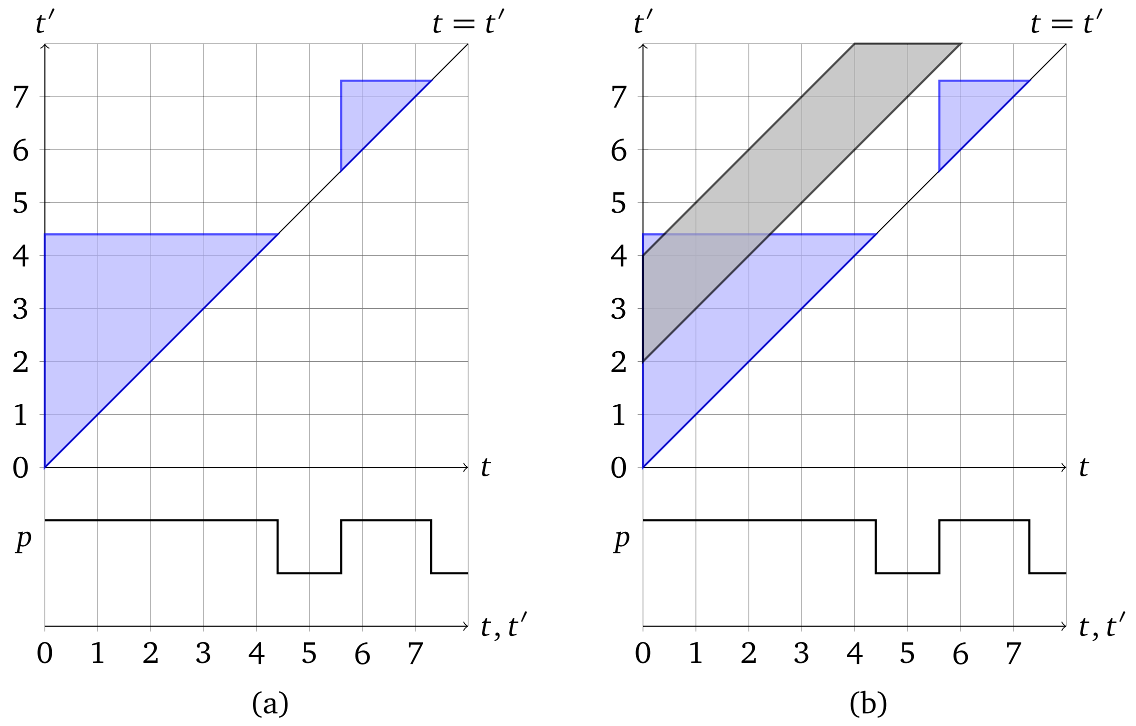


Figure 6.2: (a) The match set of a signal with respect to an atomic expression $\underline{p}$; (b) The effect of time restriction with $\langle \underline{p} \rangle_{[2,4]} \Leftrightarrow \underline{p} \wedge \langle \underline{\top} \rangle_{[2,4]}$.

Geometrically speaking, $X \fatsemi Y$ can be seen as inverse-projecting X and Y into a 3-dimensional space with axes labeled by t, t' and t'', intersecting these two sets and projecting back on the plane t, t'. Another intuition in dimension 2 is given Figure 6.3-(b).

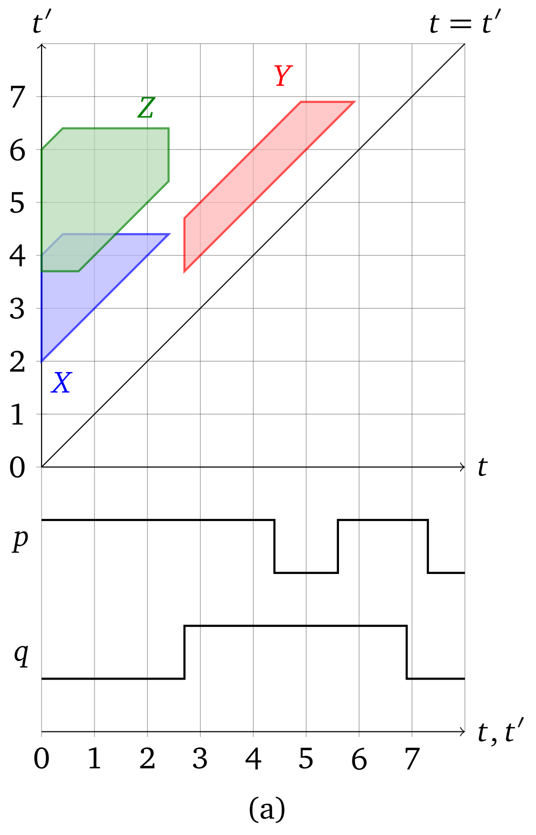
The next lemma that we will prove states that it is sufficient to consider the relation of finitely many zones to obtain the match set of a starred expression. First we present a simpler but more direct argument. This argument uses the finite variability of the trace to bound the number of iterations. Essentially we prove that the match set of the star can be computed by a finite number of concatenations.

Let us say that an open interval (t, t') is *unitary* with respect to $\boldsymbol{w}$ if $t' - t < 1$ and $\boldsymbol{w}$ is constant throughout (t, t'). We recall that all intervals in expressions have integer bounds; in particular 1 is the smallest non-zero duration constant. The following property of unitary intervals can be proved straightforwardly by structural induction on φ.

Lemma 6.4. *Let (t, t') be a unitary interval with respect to $\boldsymbol{w}$. For all intervals $(s, s') \subseteq (t, t')$ we have $(\boldsymbol{w}, s, s') \models \varphi$ if and only if $(\boldsymbol{w}, t, t') \models \varphi$.*

Let $\|\boldsymbol{w}\|$ be the least k such that $\boldsymbol{w}$ can be covered by k unitary intervals, that is, there exists a sequence of intervals $(0, t_1), (t_1, t_2), \ldots, (t_{k-1}, d)$, all unitary with respect to $\boldsymbol{w}$. A key property of $k = \|\boldsymbol{w}\|$ is the following.

Lemma 6.5. *For any $n > 2k + 1$, if $(\boldsymbol{w}, t, t') \models \varphi^n$ then $(\boldsymbol{w}, t, t') \models \varphi^{n-1}$.*



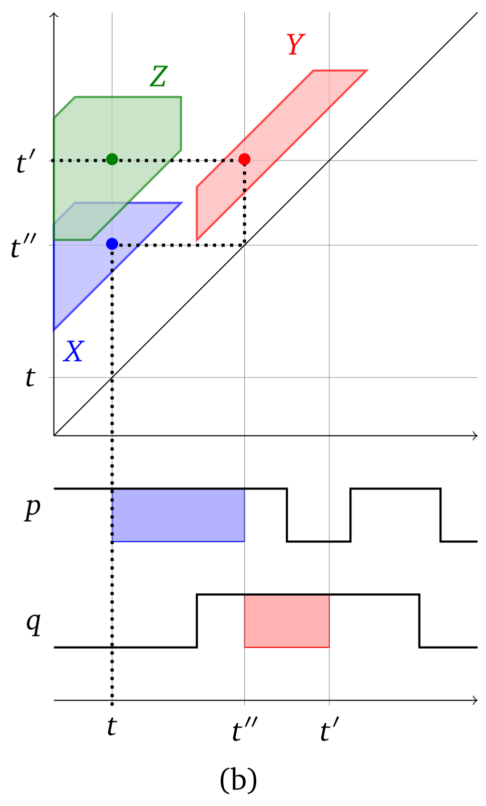


Figure 6.3: (a) Match sets X, Y and Z of expressions $\varphi = \langle \underline{p} \rangle_{[2,4]}$, $\psi = \langle \underline{q} \rangle_{[1,2]}$ and their concatenation $\varphi \cdot \psi$, respectively; (b) Every match $(t, t') \in Z$ corresponds to a path from t on the abscissa to t' on the ordinate via a match $(t, t'') \in X$, the point t'' on the diagonal, and a match $(t'', t') \in Y$.

Proof. Let $(0, t_1), (t_1, t_2), \ldots, (t_{k-1}, d)$ be a sequence of unitary intervals with respect to $\boldsymbol{w}$. If $(\boldsymbol{w}, t, t') \models \varphi^n$ then there exists a sequence of time points $t = s_0 \leq s_1 \leq \ldots \leq s_n = t'$ such that for $i = 1..n$

$$(\boldsymbol{w}, s_{i-1}, s_i) \models \varphi \tag{6.1}$$

When $n > 2k+1$, by the pigeonhole principle, among time points $r_0, \ldots, r_n$ there are three consecutive points, denoted by s_{i-1}, s_i, s_{i+1}, within the same unitary interval (t_{j-1}, t_j) of $\boldsymbol{w}$. By Lemma 6.4 it holds that $(\boldsymbol{w}, s_{i-1}, s_{i+1}) \models \varphi$, thus the time point s_i can be excluded from $s_0, \ldots, s_n$ still preserving (6.1). Hence $(\boldsymbol{w}, t, t') \models \varphi^{n-1}$. ∎

Corollary 6.6. *For any expression φ and any signal $\boldsymbol{w}$ with $\|\boldsymbol{w}\| = k$ it holds that $[\varphi^*]_{\boldsymbol{w}} = \bigcup_{n=1}^{2k+1} [\varphi^n]_{\boldsymbol{w}}$.*

We have demonstrated our second key lemma, addressing the inductive step for Kleene star. This concludes the proof of Theorem 6.2.

6.4 Algorithms

6.4.1 Zones Computation

A procedure for computing match-sets can be readily derived from the proof of Theorem 6.2. This procedure, sketched in Algorithm 6.1, recursively calls a subroutine COMBINE that takes as arguments the topmost operator of the expression $\bullet$ among $\cdot$, $\vee$, $\wedge$, $*$, or $\langle\,\rangle_I$ for some I, along with the match sets of the subexpressions and applies the operation corresponding to the specific operator $\bullet$.

Algorithm 6.1 ZONES($\varphi, \boldsymbol{w}$)

```
  select $\varphi$
  case $\epsilon$, $p$:
    $\mathscr{Z}_\varphi :=$ ATOM($\varphi, \boldsymbol{w}$)
  case $\bullet\ \psi$:
    $\mathscr{Z}_\psi :=$ ZONES($\psi, \boldsymbol{w}$)
    $\mathscr{Z}_\varphi :=$ COMBINE($\bullet, \mathscr{Z}_\psi$)
  case $\psi_1 \bullet \psi_2$:
    $\mathscr{Z}_{\psi_1} :=$ ZONES($\psi_1, \boldsymbol{w}$)
    $\mathscr{Z}_{\psi_2} :=$ ZONES($\psi_2, \boldsymbol{w}$)
    $\mathscr{Z}_\varphi :=$ COMBINE($\bullet, \mathscr{Z}_{\psi_1}, \mathscr{Z}_{\psi_2}$)
  end select
  return $\mathscr{Z}_\varphi$
```

Our algorithm intensively performs various binary operations over sets of zones by in turn applying this operation to every pair of zones that can be formed from those two sets. In addition to the operations defined in the expressions, in various stages we eliminate redundancy by checking *pairwise inclusion* of zones covering a match-set. This kind of filtering consists in removing from $\mathscr{Z}$ all zones strictly included in other zones in $\mathscr{Z}$. As the purpose of $\mathscr{Z}$ is to cover some match set taking its union, this operation has no effect on the result but it is an important ingredient of an efficient implementation. This filtering is systematically applied when merging two sets of zones; accordingly we define the pairwise union $\Cup$ by letting

$$\mathscr{Z} \Cup \mathscr{Z}' = \{Z \in \mathscr{Z} : \forall Z' \in \mathscr{Z}', Z \not\subset Z'\} \cup \{Z' \in \mathscr{Z}' : \forall Z \in \mathscr{Z}, Z' \not\subset Z\}$$

We also define the pairwise inclusion $\Subset$ by

$$\mathscr{Z} \Subset \mathscr{Z}' \quad \text{if and only if} \quad \forall Z \in \mathscr{Z}, \exists Z' \in \mathscr{Z}', Z \subseteq Z'$$

That is $\mathscr{Z} \Subset \mathscr{Z}'$ when each zone in $\mathscr{Z}$ is included in a zone of $\mathscr{Z}'$. Note that $\mathscr{Z} \Subset \mathscr{Z}'$ if and only if $\mathscr{Z} \Cup \mathscr{Z}' = \mathscr{Z}'$. [2]

A naive implementation of a binary operation on sets of zones with n elements would need $O(n^2)$ operations. In practice, many of these operations yield an empty set and

[2]In itself, filtering $\mathscr{Z}$ could be defined as taking the smallest subset of $\mathscr{Z}$ that pairwise contains $\mathscr{Z}$. Note that this does not yield a minimal representation of a finite union of zones. For this one would need to consider inclusion of a union of zones in a zone and vice-versa, which are costly combinatorial operations.

can be avoided by exploiting inherent ordering between zones. In particular we can implement this idea through the *plane sweep* algorithm [97], used as is for intersection, and specialized for concatenation.

Algorithm 6.2 COMBINE($\wedge, \mathscr{X}, \mathscr{Y}$)

```
Require: $\mathscr{X}, \mathscr{Y}$ sorted by $\pi_1^-$
Ensure: $\mathscr{Z}$ sorted by $\pi_1^-$
  $\mathscr{X}' := \mathscr{Y}' := \mathscr{Z} := \emptyset$
  while $\mathscr{X} \neq \emptyset \wedge \mathscr{Y} \neq \emptyset$ do
    $X := \text{first}(\mathscr{X})$
    $Y := \text{first}(\mathscr{Y})$
    if $\pi_1^-(X) < \pi_1^-(Y)$ then
      $\mathscr{X} := \mathscr{X} \setminus \{X\}$
      $\mathscr{X}' := \mathscr{X}' \cup \{X\}$
      $\mathscr{Y}' := \{Y' \in \mathscr{Y}' : \pi_1^+(Y') \geq \pi_1^-(X)\}$
      for all $Y \in \mathscr{Y}'$ do
        $Z := X \cap Y$
        $\mathscr{Z} := \mathscr{Z} \Cup \{Z\}$
      end for
    else
      $\mathscr{Y} := \mathscr{Y} \setminus \{Y\}$
      $\mathscr{Y}' := \mathscr{Y}' \cup \{Y\}$
      $\mathscr{X}' := \{X \in \mathscr{X}' : \pi_1^+(X) \geq \pi_1^-(Y)\}$
      for all $X \in \mathscr{X}'$ do
        $Z := X \cap Y$
        $\mathscr{Z} := \mathscr{Z} \Cup \{Z\}$
      end for
    end if
  end while
  return $\mathscr{Z}$
```

For intersection, Algorithm 6.2 keeps X and Y sorted according to π_1^-. It maintains two active sets X' and Y' which contain candidates for intersection. Elements are successively moved to the active sets and removed from the inactive sets when it is clear they will not participate in further (non-empty) intersections. This happens for $X \in X'$ such that $\pi_1^+(X) < \pi_1^-(Y)$ for every $Y \in Y'$ and vice versa. We avoiding the quadratic blow up by considering candidates for inclusion in an ordered manner. For concatenation, this is achieved by computing a pointwise operation with $\mathscr{Z} = \{X \fatsemi Y : X \in \mathscr{X}, Y \in \mathscr{Y}\}$, observe that $X \fatsemi Y \neq \emptyset$ if and only if $\pi_2(X) \cap \pi_1(Y) \neq \emptyset$. Hence we can apply an algorithm similar to Algorithm 6.2, where $\mathscr{X}$ is this time sorted according to π_2^- and $\mathscr{Y}$ is sorted according to π_1^-.

For the star operation, we tried two different approaches. In the *incremental* approach we compose the input set $\mathscr{X}$ with an accumulating set $\mathscr{Y}$ initialized to $\mathscr{X}$, computing the sequence $\cup\mathscr{X}, (\cup\mathscr{X})^2, (\cup\mathscr{X})^3, \ldots, (\cup\mathscr{X})^n$. In the *squaring* approach we compose the accumulating set $\mathscr{Y}$ with itself, computing the sequence $\cup\mathscr{X}, (\cup\mathscr{X})^2, (\cup\mathscr{X})^4, \ldots, (\cup\mathscr{X})^{2^k}$. The squaring approach is more efficient for sets of zones that converge slowly to a fixpoint.

Algorithm 6.3 depicts our implementation of this approach. In order not to compose the same sequence of zones twice, we maintain two sets of zones $\mathscr{Y}_k$ and $\mathscr{Z}_k$ such that at the end of iteration k we have $\cup\mathscr{Y}_k = (\cup\mathscr{X})^{2^k}$ and $\cup\mathscr{Z}_k = \bigcup_{n=0}^{2^k-1}(\cup\mathscr{X})^n$. According to Corollary 6.6, we may stop at the first k such that $2^k \geq 2\cdot\|w\|+1$; however under realistic assumptions a fixpoint $\mathscr{Y}_k$ can be reached in much fewer iterations. For performance reasons we do not check the inclusion $\cup\mathscr{Y}_k \subseteq \cup\mathscr{Z}_k$, but use the pairwise inclusion test $\mathscr{Y}_k \Subset \mathscr{Z}_k$. In the sequel we give an upper-bound on the number of iterations k until this condition is met.

Algorithm 6.3 COMBINE$(*, \mathscr{X})$

```
$\mathscr{Z} := \{Id\}$
$\mathscr{Y} := \mathscr{X}$
while $\mathscr{Y} \not\Subset \mathscr{Z}$ do
    $\mathscr{Z} := \mathscr{Z} \Cup$ COMBINE$(\cdot, \mathscr{Z}, \mathscr{Y})$
    $\mathscr{Y} :=$ COMBINE$(\cdot, \mathscr{Y}, \mathscr{Y})$
end while
return $\mathscr{Z}$
```

6.4.2 A Bound for the Kleene Star

The argument of Section 6.3 is based on properties of the expression relative to the length of the trace, and not on the match set $\cup\mathscr{Z}$ over which the reflexive-transitive closure is computed. Here we provide an alternative upper bound, based on properties of the match set decomposition. The intuition behind this argument can be given as follows.

On the one hand, when zones in the set $\mathscr{Z}$ have a positive minimal duration, there is an obvious bound on the number of zones that can appear in a sequence whose composition is not empty. In general the composition of two zones has a minimum duration which is at least the sum of that of both zones: $\langle \underline{p}\rangle_{[a,\infty)} \cdot \langle \underline{q}\rangle_{[b,\infty)} \Rightarrow \langle \underline{p}\cdot\underline{q}\rangle_{[a+b,\infty)}$. The temporal domain is bounded; however we place no restriction on the minimal duration of atomic predicates so the argument is void. On the other hand, triangular zones resulting from atomic expressions are stable under composition. In the absence of singular discontinuities, $\underline{p} \Leftrightarrow \underline{p}\cdot\underline{p}$, and upper diagonal constraints over such a zone will be lifted when star is applied: $(\langle \underline{p}\rangle_{[0,a]})^* \Leftrightarrow \epsilon \vee \underline{p}$ for $a > 0$. These observations are still not sufficient, we use a more detailed argument. Composing longer sequences of zones, the overall duration of matches in the resulting zone will tend to increase. Indeed we show that every repetition of some zone in a sequence causes the maximal duration to increase of at least one time unit (assuming integer diagonal bounds). There will be a certain sequence length above which the maximal duration is that of the temporal domain, and concatenating one more zone can only make the resulting composition a smaller set of matches. The computation can stop.

Fix $\mathscr{Z}$ a set of zones with $|\mathscr{Z}|$ elements. Let $c = \max_{Z,Z'\in\mathscr{Z}} \pi_2^+(Z') - \pi_1^-(Z)$ be the time-span of $\mathscr{Z}$. We show that computing the transitive-reflexive closure of $\mathscr{Z}$ using Algorithm 6.3, the pairwise inclusion test is met before $\lfloor\log(|\mathscr{Z}|+c)\rfloor+1$ iterations. A sequence of zones $Z_1,\ldots,Z_n$ is said to be *redundant* if there exists $1 \leq i < j \leq n$ such that

$Z_1 \fatsemi \dots \fatsemi Z_j \subseteq Z_1 \fatsemi \dots \fatsemi Z_i$. Notice that the star algorithm eliminates redundant sequences, since for any such $Z_1, \dots, Z_n$ we have $Z_1 \fatsemi \dots \fatsemi Z_n \subseteq Z_1 \fatsemi \dots \fatsemi Z_i \fatsemi Z_{j+1} \fatsemi \dots \fatsemi Z_n$ by transitivity. We first see that in a non-redundant sequence the maximal duration never decreases:

Lemma 6.7. *For any X, Y such that $X \fatsemi Y \not\subseteq X$ we have $\pi^+_{2,1}(X \fatsemi Y) \geq \pi^+_{2,1}(X)$.*

Proof. The propagation of difference constraints gives us $\pi^+_{2,1}(X \fatsemi Y) = \min\{\pi^+_{2,1}(X) + \pi^+_{2,1}(Y), \pi^+_2(Y) - \pi^-_1(X)\}$. Suppose $\pi^+_{2,1}(X \fatsemi Y) < \pi^+_{2,1}(X)$ and show $X \fatsemi Y \subseteq X$. First note that $\pi_1(X \fatsemi Y) \subseteq \pi_1(X)$. By hypothesis $\pi^+_2(Y) - \pi^-_1(X) < \pi^+_{2,1}(X)$, yet $\pi^+_{2,1}(X) \leq \pi^+_2(Y) - \pi^-_1(X)$ so that $\pi^+_2(Y) < \pi^+_2(X)$. This implies that $\pi_2(X \fatsemi Y) \subseteq \pi_2(X)$. Finally the hypothesis $\pi^+_{2,1}(X \fatsemi Y) < \pi^+_{2,1}(X)$ gives us $\pi_{2,1}(X \fatsemi Y) \subseteq \pi_{2,1}(X)$. ■

Let us call *repeated* a position i in the sequence $X, \dots, X_n$ such that there exists $j > i$ with $X_i = X_j$. A zone that has the ability to produce a non-redundant sequence when repeated, induces a simple sum on the maximal duration:

Lemma 6.8. *For any X, Y such that there exists Z with $X \fatsemi Y \fatsemi Z \fatsemi Y \not\subseteq X \fatsemi Y$ we have $\pi^+_{2,1}(X \fatsemi Y) = \pi^+_{2,1}(X) + \pi^+_{2,1}(Y)$.*

Proof. Suppose $\pi^+_{2,1}(X \fatsemi Y) < \pi^+_{2,1}(X) + \pi^+_{2,1}(Y)$, and show that $X \fatsemi Y \fatsemi Z \fatsemi Y \subseteq X \fatsemi Y$ for any zone Z. Similarly to the proof of Lemma 6.7 it is sufficient to show that π^+_2 and $\pi^+_{2,1}$ do not increase. On the one hand $\pi^+_2(X \fatsemi Y \fatsemi Z \fatsemi Y) \leq \pi^+_2(Y) = \pi^+_2(X \fatsemi Y)$, and on the other hand $\pi^+_{2,1}(X \fatsemi Y \fatsemi Z \fatsemi Y) \leq \pi^+_2(Y) - \pi^-_1(X) = \pi^+_{2,1}(X \fatsemi Y)$. ■

Remark 6.1. *Following our assumption that all timing intervals in expressions have integer bounds, we may assume that in zones for which the maximal duration is less than 1, the upper diagonal constraint is always redundant (consequence of other constraints). By a straightforward induction on the expression, it can be shown that a zone Z' produced by our algorithms and such that $\delta^+(Z') < 1$ always verifies $Z' = \pi_1(Z') \times \pi_2(Z') \cap \{(t, t') : t' - t > 0\}$. Thus if such a zone Z' was repeated it would make the corresponding sequence redundant; under the conditions of Lemma 6.8 we indeed have $\delta^+(Z \fatsemi Z') \geq \delta^+(Z) + 1$.*

Theorem 6.9. *Algorithm 6.3 stops within $\lfloor \log(|\mathscr{Z}| + c) \rfloor + 1$ iterations of its* while *loop.*

Proof. We first show that any non-redundant sequence of zones in $\mathscr{Z}$ with m repetitions has a duration of at least m time units. Fix $Z_1, \dots, Z_n$ a sequence of zones of $\mathscr{Z}$ with m repetitions, and show $\pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_n) \geq m$. Let i be a position in the sequence. If Z_i is repeated, there exists $j > i$ with $Z_i = Z_j$. Factoring the composition of $Z_1, \dots, Z_j$ into $(Z_1 \fatsemi \dots \fatsemi Z_{i-1}) \fatsemi Z_i \fatsemi (Z_{i+1} \fatsemi \dots \fatsemi Z_{j-1}) \fatsemi Z_j$ we see by Lemma 6.8 that $\pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_i) = \pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_{i-1}) + \pi^+_{2,1}(Z_i)$. Following Remark 6.1 it holds $\pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_i) \geq \pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_{i-1}) + 1$, the maximal duration increases by 1 or more. Else Z_i is not repeated and by Lemma 6.7 we ensure $\pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_i) \geq \pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_{i-1})$, the maximal duration does not decrease. With m repeated zones, the sequence $Z_1, \dots, Z_n$ has maximal duration of at least m.

Now suppose that the algorithm reaches iteration $k = \lfloor \log(|\mathscr{Z}| + c) \rfloor + 1$ and does not stop. There exists a zone Z in $\mathscr{Z}_k$ and a non-redundant sequence $Z_1, \dots, Z_n$ such that $Z = Z_1 \fatsemi \dots \fatsemi Z_n$ for some n with $2^{n+1} > k \geq 2^n$. Such a sequence has at least $n - |\mathscr{Z}|$ repeated zones, and maximal duration $\pi^+_{2,1}(Z_1 \fatsemi \dots \fatsemi Z_n) \leq c$. Hence $n - |\mathscr{Z}| \leq c$, giving us $|\mathscr{Z}| + c - |\mathscr{Z}| < c$. Contradiction! Thus the algorithm stops at iteration $\lfloor \log(|\mathscr{Z}| + c) \rfloor + 1$ or earlier. ■

6.5 Integration with Temporal Logic

We defined the membership problem as that of checking whether $\boldsymbol{w} \models\!\equiv \varphi$, and the pattern matching problem as that of finding all segments (t, t') such that $(\boldsymbol{w}, t, t') \models\!\equiv \varphi$. Clearly pattern matching algorithms as described can readily be used for monitoring of TRE defined as a membership problem. For example we can specify with expression φ as follows that the trace should begin by p holding for 3 time units, followed by q switching from false to true with period between 4 and 5 for an even number of times, and finish with r holding for 2 time units:

$$\langle \underline{p} \rangle_{[3,3]} \cdot (\langle \underline{\neg q} \cdot \underline{q} \rangle_{[4,5]})^{2*} \cdot \langle \underline{p} \rangle_{[2,2]}$$

Monitoring this property consists in deciding $\boldsymbol{w} \models\!\equiv \varphi$, equivalently if $(0, d) \in [\varphi]_{w}$ where $d = \sup \mathbb{T}$. In this setting, the knowledge of the entire match set of φ is superflous.

Not all temporal behaviors however can be easily specified in this fashion. Some behaviors are better described using invariants or reactive properties, as with temporal logic. Consider the simple case of the safety property according to which p should not hold continuously for more than 2 time units. This property can be written as the assertion

$$\psi = \neg(\underline{\top} \cdot \langle \underline{p} \rangle_{(2,\infty)} \cdot \underline{\top})$$

In its TRE form this property is more complicated to express, with

$$\psi' = (\underline{\neg p} \vee \epsilon) \cdot (\langle \underline{p} \rangle_{[0,2]} \cdot \underline{\neg p})^{*} \cdot (\langle \underline{p} \rangle_{[0,2]} \vee \epsilon)$$

Complex assertions cannot always be expressed without resorting to negation or temporal operators.[3]

Temporal operators enable specifying the intent of the regular expression. For example, a safety property should be monitored according to the semantics of a temporal *always* operator. This feature is available in assertion languages, where a regular expression may be evaluated in the context of an arbitrary temporal logic formula as sketched in Chapter 2. A simple way to add this kind of functionality is to consider the operator *suffix implication*. This operator, denoted $\circ\!\!\rightarrow$, is defined as through abbreviation $\rho \circ\!\!\rightarrow \varphi = \neg(\rho \circ \neg\varphi)$, where $\circ$ denotes the *suffix conjunction* operator. In turn that operator has semantics such that $(\boldsymbol{w}, t) \models \rho \circ \varphi$ if and only if there exists t', $(\boldsymbol{w}, t, t') \models\!\equiv \rho$ and $(\boldsymbol{w}, t') \models \varphi$. The form "$\rho\, \circ$" seen as a prefix operator[4] can be handled via a simple additional case in the setting of our monitoring procedure of Chapter 3.

Let us define *timed assertions*, with syntax given by the following grammar:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \,\mathrm{U}\, \varphi \mid \rho \circ \varphi$$

where p is a proposition in $\mathbb{P}$, and ρ is a TRE using propositions in $\mathbb{P}$. Semantics are defined as previously. The *timed eventually* operator can be obtained through the following abbreviation: $\Diamond_I\, \varphi = \langle \epsilon \vee \underline{\top} \rangle_I \circ \varphi$. Adding *timed until* operator to such assertions does

[3]Negation could be introduced as a regular expression primitive. However looking for the *absence* of a match requires to complement a match set; this may be expensive to compute.

[4]This operator was first introduced in dynamic logic [58].

Algorithm 6.4 INTERVALS($\varphi, \boldsymbol{w}$)

```
select $\varphi$
...
case $\rho \circ \psi$:
    $\mathscr{I}_\psi :=$ INTERVALS($\psi, \boldsymbol{w}$)
    $\mathscr{Z}_\rho :=$ ZONES($\rho, \boldsymbol{w}$)
    $\mathscr{I}_\varphi :=$ COMBINE($\circ, \mathscr{Z}_\rho, \mathscr{I}_\psi$)
end select
return $\mathscr{I}_\varphi$
```

not increase their expressiveness, as this operator may be obtained using the *until rewrite* rule of Proposition 3.1.

For the monitoring of timed assertions, we proceed as follows. We first extend the composition operation $⨟$ to accept *predicates* in its right-hand argument. For an arbitrary relation R and predicate P, let $R ⨟ P = \{t \in \mathbb{T} : \exists t',\ (t, t') \in R \text{ and } t' \in P\}$.

Proposition 6.10. *For any formula ψ, expression ρ, and trace $\boldsymbol{w}$ we have $[\rho \circ \psi]_w = [\rho]_w ⨟ [\psi]_w$.*

As we have seen in Lemma 6.3, zones are closed under composition. Notice that an interval is a one-dimensional zone. Suffix conjunction commutes with disjunction, so that it can be done for all combinations of zone in $[\rho]_w$ and interval in $[\varphi]_w$. The temporal logic monitoring procedure of Chapter 3 is adapted to timed assertions by adding a single inductive case as appears in Algorithm 6.4. The implementation of COMBINE($\mathscr{Z}_\rho, \mathscr{I}_\psi$) does not pose particular difficulty; it proceeds elementwise in an ordered fashion and may use standard zone algorithms to perform intersection and projection.

6.6 Evaluation

We benchmark our algorithms on the following example, of intermediate complexity. We keep expression φ fixed while changing the signal characteristics. Our traces define two boolean signals p and q. We define an expression which is satisfied when the two signals oscillate rapidly together for some amount of time:

$$\varphi = \langle ((\underline{p} \cdot \underline{\neg p}\rangle_{[0,10]})^* \wedge (\langle \underline{q} \cdot \underline{\neg q}\rangle_{[0,10]})^* \rangle_{[80,\infty]}$$

We then generate input signals with n segments, where segments have a length 400 time units. For each segment, we draw a time sequence of switching points using an exponential distribution with expected delay $\frac{1}{v}$ where the random variable v controls the expected variability. This ensures that there tends to be less switching at the end of each segment, favoring the stabilization case. We monitor the expression φ on signals of different length (number of segments n), and variability (parameter v). The results are depicted in Table 6.1. We also report the number switching points $|\boldsymbol{w}|$ in the trace $\boldsymbol{w}$ along with the number of zones found $|\mathscr{Z}_\varphi|$. These results are consistent with simpler examples, indicating that one can monitor complex expressions without facing a blow-up in computation time.

Table 6.1: Computation time (s) as a function of expected variability v and length n (number of segments). Size of the trace (actual number of events) is shown as $|\mathbf{w}|$, and number of zones as $|\mathscr{Z}_\varphi|$.

v	n	$\lvert\mathbf{w}\rvert$	$\lvert\mathscr{Z}_\varphi\rvert$	time
0.025	100	1893	0	0.08
0.025	200	3825	0	0.17
0.025	400	7642	0	0.37
0.05	100	3654	0	0.27
0.05	200	7305	0	0.60
0.05	400	14614	0	1.27
0.075	100	5131	1	0.64
0.075	200	10476	4	1.40
0.075	400	21200	5	2.88
0.1	100	6715	10	1.35
0.1	200	13306	23	2.73
0.1	400	26652	47	5.83

In other experiments reported in [110], we evaluated the computation of Kleene star by iterative, and squaring algorithms. We saw that the squaring performs better than incremental algorithm except cases where n is small. This can be explained by the fact that in the squaring approach, the effect of filtering is multiplied in the following sense. Every sequence that we discard may have appeared in several factorizations of longer sequences; squaring will reuse sequences as subsequences in many places which the incremental approach does not do. Another effect that we observed is that the number of zones covering sequences of length n does not explode but rather seem to stay constant over iterations.

We have demonstrated the feasibility of the offline monitoring of TRE and their integration with temporal logic. In the context of dynamic verification, it is useful to monitor assertions during simulation so as to possibly stop early in case a violation. Making our algorithm work online would require to re-order the operations on zones according to some notion of time. The duration restriction $\langle\rangle_I$ operator may also enjoy specific treatment so as to cancel matches beyond the maximum duration in I. An online approach to timed regular expression monitoring appears in [111].

Regular expressions can also specify actions that represent *events*; this also was the model of the timed regular expressions of [25]. In our signals framework, we can also handle events such as rise and fall of signals. Events only induce finitely many matches and can be represented by punctual zones. In the setting of event-based regular expressions, specification languages have concatenation operators allowing to wait a (non-deterministic) number of discrete time units between events; the generalization of SVA proposed in [63] shows that this concept is easily lifted to the real-time setting. The extension of timed regular expressions to events will be demonstrated in Chapter 7.

7

Pattern-Based Measurements

This chapter is concerned with the specification of quantitative properties, also known as measurements, and their computation on several segments of a given trace. In this context we propose a *declarative* specification language for measurements, with a particular focus on the specification of mixed-signal behaviors. Timed regular expressions are extended with events and conditional expressions, and used for defining segments of the simulation trace over which measurements are to be taken. We conjoin measure specifications to such expressions, in order to describe a particular type of aggregation (maximum, minimum, average, etc.) to be done over the matched segments of the trace. The resulting language enables expressive and versatile specification of measurement objectives. This measurement framework was implemented, and applied to a case study based on an analog communication protocol for automotive embedded systems. Experiments demonstrate that the proposed technique is usable with a very low overhead compared to a typical simulation.

7.1 Introduction

7.1.1 Motivation

In the AMS context, various measures are associated with systems and their executions. Measures are computed by applying various operations such as summation/integration, arithmetical operations, minimum/maximum to certain segments of the simulation trace. The endpoints of these segments are defined according to the occurrence of certain *events*. Simple measures can be realized by inserting additional observer blocks to the system model, but when they are more complex, they are extracted using manually-written (and error-prone) procedural scripts that perform computations over the traces.

Measures typically take place over particular intervals of time, where a given behavior can be observed, and the measure can thus be taken. The semantics of a regular expression φ relative to $\boldsymbol{w}$ is not defined for a single time point like in temporal logic, but for a *pair* of points (t, t') such that the segment of $\boldsymbol{w}$ between t and t' matches the expression. This makes regular expressions ideal for defining sets of time segments. Once a time interval is isolated, the measure often consists in a simple aggregating operation such as taking a minimum, maximum, or integral of some signal in the trace. In the approach we propose, one gains the expressiveness of the language of Timed Regular Expressions (TRE), which allows to detect complex sequences of events and states in the trace. This facilitates repeated measurements (over several intervals) according to a sequence of events, while maintaining a clean separation of the behavior description from the measure itself.

7.1.2 Our Approach

We propose a declarative *measurement specification language* for automatically extracting continuous measures from mixed-signal traces. Expressions are used to define the scope of measurements, using a variant of TRE specially adapted for this purpose, by adding conditional expressions and atomic events. An additional language layer enables defining the particular measures applied to the matching segments. The extraction of measures takes advantage of the pattern matching procedure introduced in Chapter 6 for computing the set of segments of a trace that match a timed regular expression.

The problem of matching TRE over Boolean signals was studied Chapter 6. In order to use such expressions for real signals we add threshold propositions. To further gain in expressiveness we also introduce operators that condition the match of a TRE according to some behavior occurring before, or after the matched segment. We also define *events* of zero duration such as rising and falling edges. The resulting specification language, called Signal Regular Expressions (SRE), can be monitored against mixed-signal behaviors. By wrapping such specifications with some aggregating operator, we get our definition of declarative measurements. The resulting framework is a first step toward making the practice of measurement extraction more rigorous and automated.

7.1.3 Related Work

Quantitative Regular Expressions of [20] enable the definition of streaming algorithms, computing some aggregated value based on regular properties of the input trace. Assertion-based *features* of [41, 13] proposed independently are similar in spirit to ours. The authors propose an approach for quantitative evaluation of mixed-signal properties expressed as regular expressions. In contrast to our work, the regular expressions are extended with local variables, which are used to explicitly store values of interest, such as the beginning and the end time of a matched pattern. In [41], emphasis is placed on measuring features (quantitative properties, in our terminology) of hybrid automata models using formal methods. We also mention the extension to TRE proposed in [63] that combines specification of real-time events and states occurring in continuous-time signals. Their syntax and primitive constructs are inspired by and extend industrial standards PSL and SVA. This work focuses on a translation from TRE to timed automata acceptors, but does not address the problem of pattern matching an expression on a concrete trace.

The use of pattern matching in verification has been considered in the setting of formal verification of concurrent programs in [51]. In the context of modeling resource-constrained computations, quantitative languages [36] were studied as generalizations of formal languages in which traces are associated with a real number rather than a

Boolean value. The idea of quantitative languages are further extended in [64], by defining the *model measuring* problem. The model checking problems of TCTL and LTL are extended in [114, 50, 18] to a model measuring paradigm by parameterizing the bounds of the temporal operators. The authors propose algorithms for identifying minimum and maximum parameter values for which the model satisfies the temporal formula. A similar extension is proposed in [26] for Signal Temporal Logic, where both the temporal bounds and real-valued thresholds are written as parameters and inferred from signals. Robust interpretation of temporal logic specifications, presented in Chapter 5, is another way to associate numbers with traces according to how strongly they satisfy or violate a property.

7.2 Signal Regular Expressions

In this section we define Signal Regular Expressions (SRE). This specification language is based on TRE interpreted over continuous-time signals, with atomic expressions build from Boolean variables and real variables x, that are checked using threshold conditions of the form $x \leq c$ for c constant in $\mathbb{R}$. In addition to previous operators we also consider conditional operators, that enable checking the occurrence of some subexpression without changing position in time. Such an operator improves the expressiveness when expressions are used to specify segments of a given trace.

7.2.1 Syntax and Semantics

We let $\mathbb{T} = [0, d]$ be the temporal domain of simulation traces. A *proposition* is taken to be either a Boolean variable, a condition $x \leq c$ for some real variable $x \in \mathbb{X}$ and constant c, the negation $\neg p$ of some proposition p, or the disjunction $p \vee q$ of propositions p and q. Let us assume $\mathbb{P}$ a set of propositions conforming to this description. The semantics of propositional operators is that of predicates over $\mathbb{T}$, denoted $[p]_w$, or simply p_w in the absence of ambiguity for some $p \in \mathbb{P}$.

The syntax of signal regular expressions is given according to the following grammar:

$$\varphi ::= \epsilon \mid \underline{p} \mid \uparrow p \mid \varphi? \mid \varphi! \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \cdot \varphi \mid \varphi^* \mid \langle \varphi \rangle_I$$

for p proposition in $\mathbb{P}$, and I an interval of $\mathbb{R}_{\geq 0}$.

In Signal Regular Expressions we consider two types of actions (atomic expressions) associated with a proposition: the proposition holds continuously over some time interval, or the time interval is singular and is located at a discontinuity of the proposition. Actions form the atomic expressions in our pattern language; their semantics are that of predicates over $\mathbb{T}^2$. On the one hand, $\underline{p}$ denotes the action of p holding for some positive time, with semantics defined by $(t, t') \in \underline{p}_w$ if and only if $t < t'$ and $t'' \in p_w$ for all $t'' \in (t, t')$. On the other hand, $\uparrow p$ denotes the instantaneous action of p becoming true, with semantics defined by $(t, t') \in [\uparrow p]_w$ if and only if $t = t'$, $p(t^-) = 0$, and $p(t^+) = 1$. As with signals we define abbreviations $\downarrow p = \uparrow \neg p$ and $\updownarrow p = \uparrow p \vee \downarrow p$.

The *future* operator '?' is borrowed from dynamic logic [58]; operator '!' is its *past* version.[1] The semantics of clauses $\varphi?$ and $\varphi!$ is given as follows:

$$\begin{array}{lll} (\boldsymbol{w}, t, t') \models\equiv \varphi? & \text{if and only if} & t = t' \text{ and } \exists t'' \geq t,\ (\boldsymbol{w}, t, t'') \models \varphi \\ (\boldsymbol{w}, t, t') \models\equiv \varphi! & \text{if and only if} & t = t' \text{ and } \exists t'' \leq t,\ (\boldsymbol{w}, t'', t) \models \varphi \end{array}$$

The semantics of regular and timing operators is defined as in Section 6.2.1.

7.2.2 Match Set

Following the definitions in Chapter 6, we characterize φ by the set of segments of $\boldsymbol{w}$ that match it, which is called the *match set* of φ. The match set of a signal regular expression φ over $\boldsymbol{w}$ is the set of all pairs (t, t'), denoted $[\varphi]_w$, such that the segment of w between t and t' matches φ. Formally, we let as previously:

$$[\varphi]_w = \{(t, t') \in \mathbb{T}^2 : (\boldsymbol{w}, t, t') \models\equiv \varphi)\}$$

The computation of a match set for a SRE φ can be performed as presented in Chapter 6 following the same argument:

Theorem 7.1 (Match Set Decomposition). *For any finite variability trace* $\boldsymbol{w}$ *and SRE* φ*, the set* $[\varphi]_w$ *is a finite union of zones.*

Proof (sketch). By induction on the expression structure. For the inductive step $\varphi = \uparrow p$, just notice that each rise event of p is represented by a punctual zone in the match set of $\uparrow p$. Formally we have $[\uparrow p]_w = \{(t, t) : p_w(t^-) = 0 \text{ and } p_w(t^+) = 1\}$. By finite variability assumption the number of discontinuities of p_w is finite. For the inductive step of $\varphi?$, notice that $[\varphi?]_w$ is obtained by projecting $[\varphi]_w$ vertically on the diagonal. Therefore it is finite union of sets of the form $\{(t, t) \in \mathbb{T}^2 : t \in I\}$ for some interval I, which are zones. Operator '!' follows from a similar argument. Other inductive cases are treated in detail in Section 6.3. ∎

7.3 Measurement Expressions

7.3.1 Event-Bounded Expressions

We first consider a subclass of SRE that may only be matched by trace segments that begin and end with events. Such expressions, called *event-bounded*, are well-behaving in the following sense: given an arbitrary trace $\boldsymbol{w}$ with finite variability, an event-bounded SRE can be matched in $\boldsymbol{w}$ only a finite number of times. Formally, event-bounded SRE are a syntactic fragment of SRE given by the following grammar:

$$\psi ::= \uparrow p \mid \psi? \mid \psi! \mid \psi \cdot \varphi \cdot \psi \mid \psi \vee \psi \mid \psi \wedge \varphi$$

where p is a proposition, φ is an SRE, and ψ is an event-bounded SRE.

[1]In dynamic logic, a so-called *test expression*, denoted $\varphi?$, occurs over zero-length segments (t, t) such that φ holds at time t.

The match set of an event-bounded SRE ψ relative to an arbitrary (finite variablity) trace $\boldsymbol{w}$ consists of a finite number of segments (t, t'), for which t and t' are some occurence times for some event appearing in ψ.

Theorem 7.2 (Match Set Finiteness). *Given an event-bounded SRE ψ and a trace $\boldsymbol{w}$, their associated match set $[\psi]_w$ is finite.*

Proof. By induction on the expression structure. Consider an arbitrary trace $\boldsymbol{w}$ and an event $\uparrow p$; by finite variability assumption there are finitely many time points in $\boldsymbol{w}$ where $\uparrow p$ occurs, so that its match set relatively to $\boldsymbol{w}$ is finite. Now let ψ be an event-bounded SRE of the form $\psi = \psi_1 \cdot \varphi \cdot \psi_2$. The trace $\boldsymbol{w}$ matches ψ on the segment (t, t') if and only if there exists some times s and s' such that $\boldsymbol{w}$ matches ψ_1 on (t, s) and matches ψ_2 on (s', t'). By induction hypothesis there are finitely many such times t, t', s and s' so that ψ itself has a finite number of matches. The case of $\psi?$ and $\psi!$ follows directly from induction; one can also see that the finiteness of the match set is preserved by $\psi_1 \vee \psi_2$ and $\psi \wedge \varphi$, which concludes our proof. ■

7.3.2 Quantitative Operations

Once the associated match set $[\varphi]_w$ is computed, we propose a two stage analysis of signals. In the first step, we compute a scalar value for each segment of $\boldsymbol{w}$ that matches φ, either from absolute times of that match, or from the values of a real signal x in $\boldsymbol{w}$ during that match. Each value is associated with the start point of the match to distinguish multiple values, and provide information on when the measure was taken.[2] A measure expression is then given using sampling or aggregating operators according to the following grammar:

$$\mu ::= \text{duration}(\varphi) \mid \min(x, \varphi) \mid \max(x, \varphi) \mid \text{integral}(x, \varphi) \mid \text{average}(x, \varphi)$$

where φ is a signal regular expression, and x is a real variable.

Let φ be an event-bounded SRE and $\boldsymbol{w}$ a trace. The semantics $[\![\varphi]\!]_w$ of φ relative to $\boldsymbol{w}$ is the set of values in $\mathbb{T} \times \mathbb{R}$ defined by induction as follows:

$$\begin{array}{lcl}
[\![\text{duration}(\varphi)]\!]_w & = & \{(t, t'-t) : (t, t') \in [\varphi]_w\} \\
[\![\min(x, \varphi)]\!]_w & = & \{(t, \min_{t \le t'' \le t'} x_w(t'')) : (t, t') \in [\varphi]_w\} \\
[\![\max(x, \varphi)]\!]_w & = & \{(t, \max_{t \le t'' \le t'} x_w(t'')) : (t, t') \in [\varphi]_w\} \\
[\![\text{integral}(x, \varphi)]\!]_w & = & \left\{\left(t, \int_t^{t'} x_w(t'')\text{d}t''\right) : (t, t') \in [\varphi]_w\right\} \\
[\![\text{average}(x, \varphi)]\!]_w & = & \left\{\left(t, \frac{1}{t'-t} \int_t^{t'} x_w(t'')\text{d}t''\right) : (t, t') \in [\varphi]_w\right\}
\end{array}$$

Notice that if φ cannot be matched by overlapping segments, then the resulting set represents a discrete-time signal. The measure μ may provide exactly the level of information needed, or require a second processing phase. In this second step of the analysis, this measure can be passed upon to other measurements, and in particular it may be further

[2]Here we adopt a future view of the measure, where its value at time t is defined by values of signals after t. The past view is also possible; it consists in associating the value of the measure with the end time of the segment, denoted t' in our equations.

aggregated into a single "representative" value. Typically that consists in computing standard statistical indicators over $[\![\mu]\!]_w$, such as the average, maximum, minimum or standard deviation, etc.

We illustrate the use of our measurement language in the specification of the *pseudo-period* measure, common in the electronic domain.

Example 7.1 (Pseudo-Period). *For oscillatory real signals x the instantaneous period is usually computed as the temporal distance between two successive positive edges. Let v_{mid} be the nominal intermediate level of signal x; we define the pseudo-period by*

$$\text{period}(x, v_{mid}) = \text{duration}(\uparrow(x \geq v_{mid}) \cdot (x > v_{mid}) \cdot (x < v_{mid}) \cdot \uparrow(x \geq v_{mid}))$$

Other common electrical measures such as rise time or duty cycle can also be defined in our measurement language.

7.4 Case Study

7.4.1 Distributed Systems Interface

Distributed systems interface (DSI3) is a flexible and powerful bus standard [5] developed by the automotive industry. It is designed to interconnect multiple remote sensor and actuator devices to a controller. The controller interacts with the sensor devices via so-called *voltage* and *current lines*. In this work we focus on two phases of the DSI3 protocol:

- The initialization phase called the *discovery mode*;
- One of the stationary phases called the *command and response mode*.

In the discovery mode, prior to any interaction the power is turned on, resulting in a voltage ramp from 0V to v_{high}. The communication is initiated by the controller that probes the presence/absence of sensors by emitting analog pulses on the voltage line. Connected sensor devices respond in turn with another pulse sent over the current line. At the end of this interaction, a final short pulse is sent to the sensors interfaces, marking the end of the discovery mode.

In the command-and-response mode, the controller sends a command to the sensor as a series of pulses (or pulse train) on the voltage line, which transmits its response by another pulse train on the current line. For power-demanding applications the command-response pairs are followed by a power pulse, which goes above v_{high}. This allows the sensor to load a capacitor used for powering its internal operation.

The DSI3 standard provides a number of ordering and timing requirements that determine correct communication between the controller and the sensor devices: (1) minimal time between the power turned on and first discovery pulse; (2) maximal duration of discovery mode; (3) expected time between two consecutive discovery pulses; (4) expected time between command and response. Figure 7.1 illustrates the discovery mode in the DSI3 protocol and provides a high-level overview of its ordering and timing requirements. In this example, the controller probes five sensor interfaces.

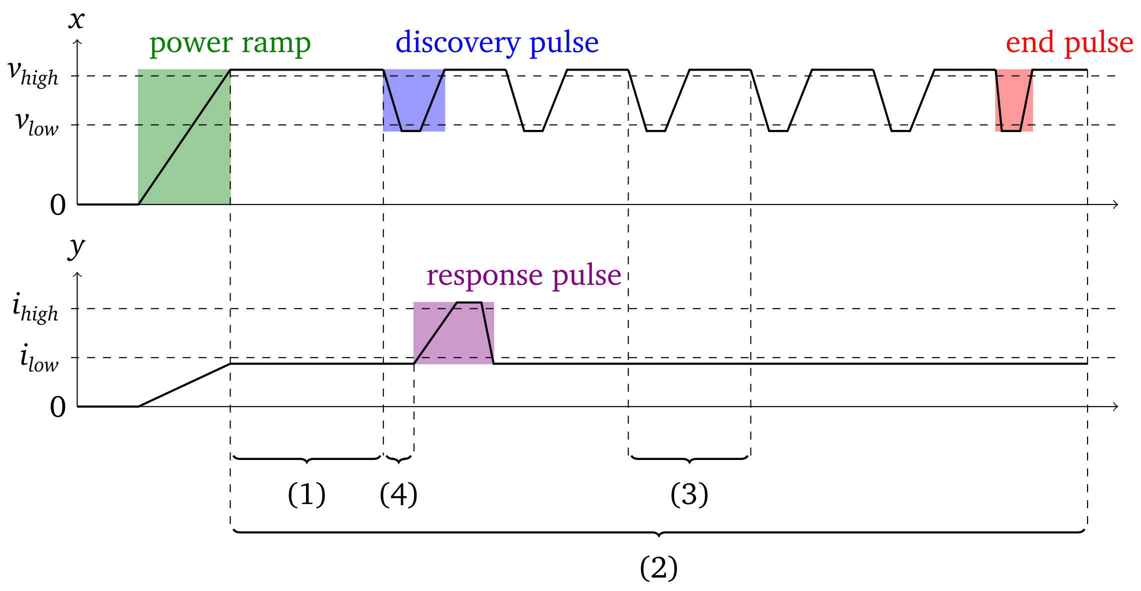


Figure 7.1: Overview of DSI3 discovery mode.

The correctness of a DSI3 protocol implementation in an automotive airbag system was studied in [94]. The above requirements were formalized as assertions expressed in STL and the monitoring tool AMT [96] was used to evaluate the simulation traces. We use instead Signal Regular Expressions to specify signal segments of interest and define several measures within the framework introduced in Section 7.2. We study two specific measures: (1) the time between consecutive discovery pulses; and (2) the amount of energy transmitted to the sensor through power pulses.

In order to generate simulation traces, we model our system as follows: the controller is a voltage-source, and the sensor is a current-source in parallel with a resistive-capacitive load. The schematic is shown in Figure 7.2. During the discovery phase the load is disabled; the voltage source generates randomized pulses in which the time between two discovery pulses has a Gaussian distribution with a mean of 250µs and a standard deviation of 3.65µs. For each power pulse of the command-and-response mode, the load is enabled and randomized with fixed capacitance value of $c = 120$nF and resistance value r uniformly distributed in the range $[25\Omega, 35\Omega]$. Threshold levels are 4.6V low, 7.8V high, 8.3V power, and 11.5V idle.

7.4.2 Time between Consecutive Discovery Pulses

To characterize a discovery pulse, we first define three regions of interest – when the voltage x is (1) below v_{low}; (2) between v_{low} and v_{high}; and (3) above v_{high}. We specify these regions with the following propositions:

$$\begin{aligned} q_{low} &= (x \leq v_{low}) \\ q_{mid} &= (v_{low} \leq x \leq v_{high}) \\ q_{high} &= (x \geq v_{high}) \end{aligned}$$

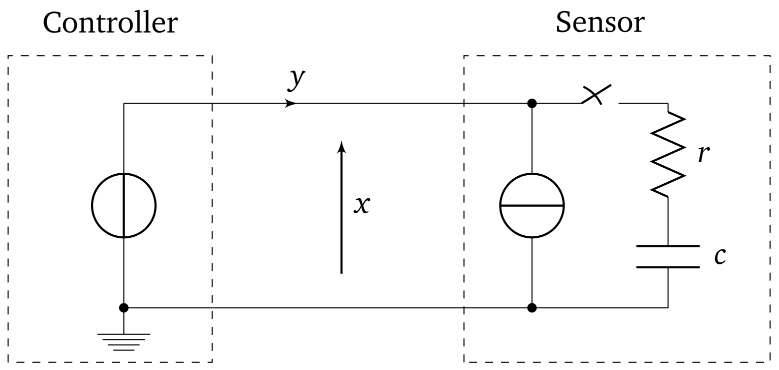


Figure 7.2: Electrical model of the system.

Next, we describe the shape of a discovery pulse. Such a pulse starts at the moment when the value of x moves from q_{high} to q_{mid}. The signal must then go into q_{low}, q_{mid} and finally come back to q_{high}. In addition to its shape, the DSI3 specification requires the discovery pulse to have a certain duration within timing interval $[a, b]$ with $a = 245\mu s$ and $b = 255\mu s$. This requirement allows distinguishing a discovery pulse from other pulses, such as the end-of-discovery pulse. We illustrate the requirements for a discovery pulse in Figure 7.3 and formalize it with the following event-bounded SRE:

$$\psi = \downarrow q_{high} \cdot \langle \underline{q_{mid}} \cdot \underline{q_{low}} \cdot \underline{q_{mid}} \rangle_{[c,d]} \cdot \uparrow q_{high}$$

In order to measure the time between consecutive discovery pulses, we need to characterize signal segments that we want to measure. The associated pattern shall start at the beginning of a discovery pulse and end at the beginning of the next one, as depicted in Figure 7.3. It consists of a discovery pulse ψ, followed by the voltage signal being in the q_{high} region, and terminating when the voltage leaves q_{high}. This description is not sufficient – we also need to ensure that this segment is effectively followed by another discovery pulse. Indeed the standard does not constrain the time between the last discovery pulse, and the end pulse. For this we concatenate a (future) conditional expression, that specifies this additional constraint. The signal regular expression is formalized as $\varphi_1 = \psi \cdot \underline{q_{high}} \cdot \psi?$. Finally, we evaluate the measure expression $\mu_1 = \text{duration}(\varphi_1)$ over the simulation trace.

7.4.3 Energy Transfer from Controller to Sensor

In the DSI3 protocol, the discovery mode can be followed by a stationary command and respond mode. A command and respond mode sequence is a pulse train that consists of a command subsequence in the form of potential pulses between v_{high} and v_{low}, a response subsequence by means of current pulses between 0 and i_{resp}, and finishes by a power pulse rising from potential v_{pwr} to potential v_{idle} in which a large current can be drawn by the sensor. We ignore the communication part and focus on this later power phase. We characterize the power pulse as follows: it occurs when the voltage goes from below v_{pwr} to above v_{idle}, and back under v_{pwr}. The three regions of interest are specified with

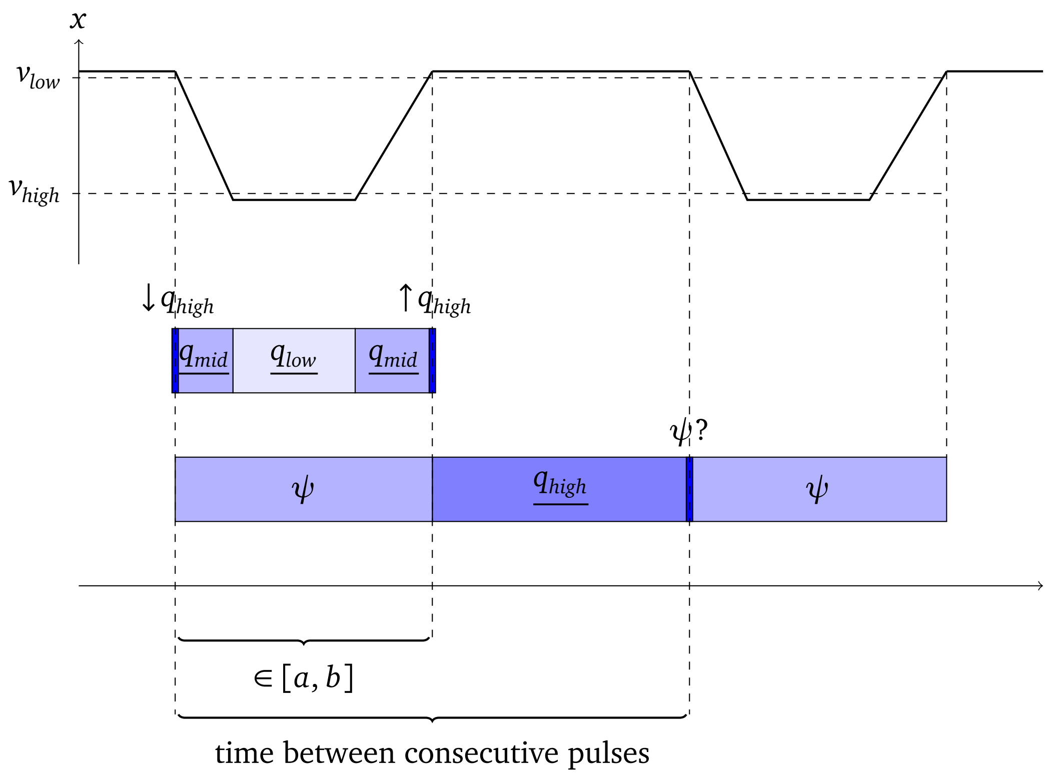


Figure 7.3: Consecutive discovery pulses with timing.

the following propositions:

$$\begin{aligned} q_{pwr} &= (x \geq v_{pwr}) \\ q_{tran} &= (v_{pwr} \leq x \leq v_{idle}) \\ q_{idle} &= (x \geq v_{idle}) \end{aligned}$$

Hence the pattern specifying a power pulse is expressed as

$$\varphi_2 = \uparrow q_{pwr} \cdot \underline{q_{tran}} \cdot \underline{q_{pwr}} \cdot \underline{q_{tran}} \cdot \downarrow q_{pwr}$$

The signal regular expression does not make use of further tests. Given x the voltage across and y the current thorough the communication line, the energy transfered to the sensor is given by the area under the power signal xy. We assume that such a signal is given in the simulation trace, and evaluate the measure expression $\mu_2 = \text{integral}(xy, \varphi_2)$ directly.

7.4.4 Results

For our experiment we apply a scenario according to which our electrical model is switched on/off 100 times in sequence to stress the discovery mode of DSI3. We implemented a behavioral model of the circuit in VerilogAMS, using a UVM constrained random stimuli to drive the current and potential sources. Simulation traces were generated using

Table 7.1: Break down of computation times (s) for measures μ_1 and μ_2

	Measure μ_1				Measure μ_2			
$\|w\|$	quantize	match	extract	total	quantize	match	extract	total
$1 \cdot 10^6$	0.047	0.617	0.000	0.664	0.009	0.004	0.011	0.024
$5 \cdot 10^6$	0.197	0.612	0.000	0.809	0.050	0.005	0.047	0.103
$1 \cdot 10^7$	0.386	0.606	0.000	0.992	0.101	0.005	0.100	0.216
$2 \cdot 10^7$	0.759	0.609	0.000	1.368	0.203	0.005	0.260	0.468

Mentor Graphics' Questa® ADMS™ simulator. The traces we generate conform to the discovery, and command-and-response modes of the protocol. We compute match sets for properties presented in Section 7.3.2 over these simulation traces using our prototype implementation.

We then compared the execution times to compute measurements, using a periodic sampling with different sampling rates. Note that our framework supports variable step sampling, but we used periodic sampling as a way to easily assess the influence of the number of samples. The computation times are given in Table 7.1 with the detailed computation time needed for evaluation of threshold propositions (quantize), match set computation (match), measure aggregation (extract) and total computation time (total). Computation of match sets does not depend on the number of samples but on the number of uniform intervals of atomic propositions. The evaluation of threshold propositions using interpolation, and the computation of measures like an integral can be done in time linear in the number of samples.

The case study confirms the practicality of our approach, in terms of performance and expressiveness. Its applicability to other types of circuits, or other engineering domains would require further investigations.

8

Analog Measurements in the Digital Testbench

In this chapter, we study the problem of measuring analog and mixed behaviors by inserting verification code in a digital testbench. The monitoring of analog behaviors, unlike digital ones, relies heavily on the use of measures, either as a preprocessing step, or as the final outcome of the simulation. A sound verification of mixed-signal designs featuring large digital portions often relies on a *structured* digital testbench, where intermediate verification results interact in a complex way. To this end we aim to provide digital components with means to access analog measurement algorithms as they already exist, in a unified fashion. This is realized by using a discrete representation of continuous-time signals in terms of a sequence of time-stamped values, which can also be implemented using a sampling clock and real variable. Several types of measurement wrappers, calling the same core algorithms are considered according to the verification context. Using measurements implemented in the simulator code, and not written manually is expected to both provide more automation and improve the accuracy of the results. We implemented a prototype of this framework and evaluated it on a case study.

8.1 Introduction

8.1.1 Motivation

Analog and digital verification follow different procedures. Digital verification relies on a testbench which consists in a collection of HDL/HVL elements surrounding the design. The testbench drives the input stimuli to the DUT and monitors the output response conjointly in an online fashion. Analog verification also contains a testbench, which consists of template sources that drive the design and measurements that are mainly done offline. We consider in particular the case of digital-centric verification, in which the predominant part of the circuit is digital and the verification is structured according to digital methodologies. Currently, analog monitors need to be implemented by verification engineers as modules, or as classes. It is possible to develop libraries dedicated to probing and analyzing analog signals in a testbench. However such libraries are bound to a given testbench architecture and their definition may not conform to that of simulator internal measurements. When implemented as simulator commands, measurements can also take advantage of information not available to the digital simulator, such as the correct interpolation method.

Notice that the sole use of simulator internal measurements by calling run management primitives, or SPICE commands, is not entirely satisfactory. If the DUT changes, for example a new component is added, or a SPICE model is replaced by an HDL one, measurements will not automatically transfer across simulation environments. These measurements can be essential in establishing correct operation of some circuit, or verifying some assumption (or guarantee) relative to input (or output) signals of some sub-circuit. Overlooking the need to ensure continuous verification of AMS behaviors (going along during various development stages) can cause failure to identify design bugs. We propose a framework for making analog measurements available to the digital testbench, for the verification of analog and mixed-signal behaviors in a digital-centric context.

8.1.2 Our Approach

The framework that we propose aims at extracting the same measures, independently of the surrounding verification code. From a user perspective, the measurement is an HDL/HVL statement or sentence, and can interact with its digital environment. That statement or sentence is one aspect of the measurement that we call the *measurement view*. Another aspect of a measurement is its actual implementation, that we call the *measurement core*. Every measurement view, for example a task or module, relies on the same measurement core to perform the computation. What ties together the view and the core of a measurement is called a *wrapper*, and consists in additional code, possibly hidden to the user, ensuring interaction of core algorithms with the visible object. We ensure this way that analog measurements in digital testbenches and other contexts are accurate and repeatable.

Every context surrounding a measurement comes with a different set of constraints, relative to how the chosen measurement view can interact with HDL/HVL constructs. For example, inside a *class*, one can instantiate another class, call a function, but one cannot instantiate a module. Independently of the context, the expected functionality of a measurement is summarized as follows. The measurement can be instantiated with specific control values, for example a threshold crossing detection is parameterized by the threshold itself as a control value. It can be applied to portions of the trace specified using surrounding HDL/HVL constructs. Eventually its output is accessible to its immediate environment, in particular other measurements. This last point is important in order to create complex measure from basic ones. We use for each measurement view the same constructs for input and output signals, and this way ensure easy composition of measurements.

Following the mode of operation of digital verification, measurements are implemented in an online fashion. To represent both discrete-time and continuous-time signals uniformly, we view the output of a measurement as a sequence of time-stamped values. The interpolation method is assumed to be taken care of by measurements algorithms. For HVL models, we use the ability to create objects during simulation, and represent each sample directly using two real *dynamic* variables, holding the measured value, and the absolute time at which it was obtained. Every time some new value is measured, a sample is created in memory, and passed on to a downstream function for processing. For HDL models, we represent the signal using *static* real and Boolean variables. The real variable represents the measured value, and the Boolean variable has an edge every time some new measure is obtained, and is called the clock. A discrete measure is obtained by sampling of the output quantity by the output clock; a continuous measure is obtained by interpolation of the output quantity between two output clock edges.

8.1.3 Related Work

A library of Verilog-AMS monitors is proposed in [30], performing usual analog measurements such as threshold crossing detection. The work of [115] presents a case study around analog monitors in the mixed-signal setting, with emphasis on the Open Verification Methodology (OVM), a precursor to UVM [8]. We provide similar measurements in the form of Verilog modules, but accessing analog quantities via system commands. The integration of analog monitors in the UVM verification environment is considered in [69]. Due to analog signals not being visible in HVL, the monitors have to be split between a HVL part and a HDL part, with the HDL part featuring the analog and mixed-signal functionality. The problem of accessing analog quantities and real variables from HVL uniformly is solved in [105] in the context of analog assertions. The advantage of bringing continuous, analog information to the digital testbench is outlined in [31], under the banner of coverage analysis. A complete, practical study of integrating analog assertions in a digital environment is proposed in [91].

8.2 Preliminaries

8.2.1 Online Measurements

In this section we give an abstract definition of an online measurement in terms of its features: the causality or dependence on past values, the update or control, the instantiation for a given signal or real parameter, and the input – output composition.

A measurement μ is a statement that, given a trace $\boldsymbol{w}$ and a time t in its definition domain, takes a value $[\![\mu]\!]_{\boldsymbol{w}}(t) \in \mathbb{R}$. Let us define a *past* measurement μ as verifying $[\![\mu]\!]_{\boldsymbol{u}}(t) = [\![\mu]\!]_{\boldsymbol{v}}(t)$ for all traces $\boldsymbol{u}$ and $\boldsymbol{v}$ over $\mathbb{T}$ such that $\boldsymbol{u}[0,t] = \boldsymbol{v}[0,t]$. The value of a past measurement at time $t \in \mathbb{T}$ only depends on the portion of the trace $\boldsymbol{w}[0,t]$. In what follows we assume that all measurements are *past*; this enables us to compute them online. For this we assume $\boldsymbol{v}$ to be a short, unitary trace segment (for example a linear piece in the case of piecewise linear traces).

In the setting of measurement instantiated in a digital context, we only consider sampled signals.[1] Fix $\mathbb{T}$ a discrete (finite) temporal domain. We let $\mathbb{U} = \mathbb{T} \times \mathbb{R}$ denote the set of possible signal samples; a signal is now a sequence of samples. We describe the online computation of a past measurement μ in terms of some set of functions operating

[1]For continuous signals, consider the interpolation scheme as being part of the measurement itself. The only limitation is that measurement values can only be produced at sampling points, and not in between.

over the possible state of the measurement algorithm, in terms of initial value, update, and computation of the measure itself. A measurement μ is defined in terms of:

- A state domain $\mathbb{S}$;
- An initial state $a_\mu \in \mathbb{S}$;
- An evaluation function $r_\mu \in \mathbb{R}^{\mathbb{S}}$;
- An aggregating function $s_\mu \in \mathbb{S}^{\mathbb{S}\times\mathbb{U}}$.

Let us denote by $y_w(t)$ the value of the measurement state at time t. The signal y_w has a forward inductive definition:

$$y_w(0) = a_\mu \qquad\qquad y_w(t') = s_\mu(y_w(t), \boldsymbol{w}(t'))$$

for t' the sampling point following t. The value of measurement μ at time $t \in \mathbb{T}$ is given by $[\![\mu]\!]_w(t) = r_\mu(y_w(t))$.

We place no restriction on the size of the domain $\mathbb{S}$, leaving the possiblity to store the entire prefix $\boldsymbol{w}[0,t]$ in $y_w(t)$. Hence every past measurement can be given in terms of the equations above. In practice most measurements only require a bounded amount of memory. For example, the online computation of the maximum of a real variable x can be described by defining r_μ as the identity over the reals, and letting $a_\mu = -\infty$ and $s_\mu : (M, \boldsymbol{v}) \mapsto \max\{M, \sup_{t\in\mathbb{T}} x_v(t)\}$. Here a single (floating point) real value is sufficient to store the state of the measurement.

There are two extreme cases of state updates that can occur in a measurement, when processing a new sample t after having processed trace $\boldsymbol{w}$ up to time t:

- $y_w(t') = y_w(t)$: sample t' is ignored, and the measure remains that at time t;
- $y_w(t') = a_\mu$: prefix up to t is forgotten, and the measure is that over some empty segment.

In the first case we could say that the measurement is *inactive*, while in the second case we may say that the measure is *reset*.

In general we consider abstract measurements, that can be *instantiated* when some real value c (or real variable x) is given as a parameter. Let μ be an abstract measurement; we write $\mu(c)$ (respectively $\mu(x)$) to denote the corresponding instance of μ. Creating instances of abstract measurements is required in many contexts. For example, a threshold crossing measurement, that measures the absolute time some quantity crosses some threshold takes two parameters: the variable it applies to, and the real-valued threshold. We can also consider abstract measurements with several parameters following the same principle.

Given an abstract measurement μ_2 with real variable parameter, and a measurement μ_1, we can also define $\mu_1(\mu_2)$ the measurement produced by their input – output composition. For arbitrary traces $\boldsymbol{u}$ and $\boldsymbol{v}$ such that $x_{\boldsymbol{u}} = [\![\mu_2]\!]_{\boldsymbol{v}}$ with variable x fresh in $\boldsymbol{v}$, and $y_{\boldsymbol{u}} = y_{\boldsymbol{v}}$ for all variables $y \neq x$, it holds that $[\![\mu_1(x)]\!]_{\boldsymbol{u}} = [\![\mu_1(\mu_2)]\!]_{\boldsymbol{v}}$.

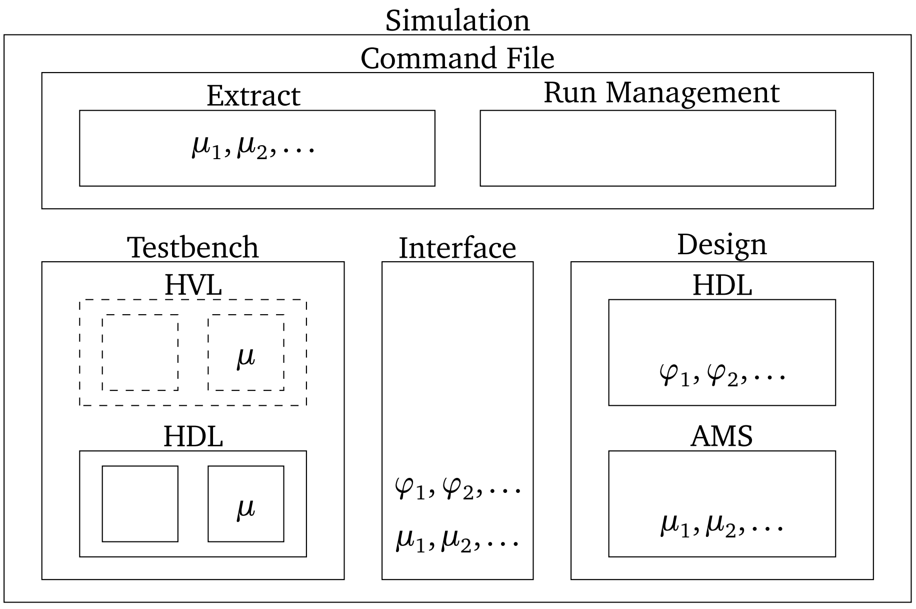


Figure 8.1: Insertion of a measurement view μ, along with measurement statements $\mu_1, \mu_2, \dots$ and assertion statements $\varphi_1, \varphi_2, \dots$ in a mixed-signal simulation. Dynamic constructs appear dashed, static constructs appear plain.

8.2.2 Interaction with Environment

In this section we explain what type of interaction measurements can have in general with the verification environment. Let us first briefly characterize HDL/HVL constructs relevant in the context of the declaration or instantiation of a measurement.

The smallest, and most general object to contain functionality is a *task*. A special case of task in known as a *function*. General tasks can spread their operation over simulated time, functions cannot. A task can be called at any time in the simulation and return at any later time. A task has the possibility to read and write quantities across logical time by passing as argument such quantities by reference. An *assertion* can call a task or function upon completion of some *sequence* (the construct that corresponds to regular expressions in SVA), or upon violation or satisfaction of some property.

In HDL, functionality is often grouped into a *module*, with input and output ports and control parameters. An important aspect of modules is that all its variables are created at the start of simulation; we call such variables *static*. In HVL, functionality can be grouped in a *class*, which may be instantiated during the simulation. In that context we may not use a module to perform a measurement: modules cannot be instantiated after the start of simulation. Variables of a class are *dynamic*; they can be created in the course of the simulation, hence a class can thus implement behaviors requiring unbounded memory.

To connect several modules and class instances, one can use an HVL *interface*. This is a construct specific to SystemVerilog, however in different languages other types of objects can be used to achieve the same functionality. An interface is a unit of code that aims to ensure proper connections of several modules or class instances. It can be used between the testbench and the DUT, or between two modules of the DUT. Assertions are usually placed in an interface to check some behavior observable at input and output ports of modules and larger blocks, for example properties of some protocol between sub-circuits.

Measurements views ought to be instantiated in some part of the testbench, where the quantity they apply to is being watched. For low level behaviors (realized by one sub-circuit) or some interaction between components, measurements can be instantiated in some interface. For high level behaviors (realized by several sub-circuits) measurements can be placed in a dedicated monitor in the testbench. In other situations measurement statements can also be placed in the run management file, or in the DUT modules. We do not address those situations here, as measurement commands like extracts are already implemented in most commercial simulators. Figure 8.1 summarizes the relevant places where assertion and measurements can be inserted in a mixed-signal simulation.

In contrast to absolute time windows used in existing analog measurement languages, we allow digital signals to control directly the operation of the measurement, based on three control events *activate*, *deactivate*, and *reset* loosely corresponding to the type of state update previously discussed. The measurement takes one signal as input and produces another signal as output.[2] Control events can be created by making the measurement code sensitive to the change of value of some Boolean variable. Alternatively such events can correspond to function calls. Similar remarks can be made for signals. The successive values of some signal can be transmitted via some real variable that drives some measured quantity, or passed on to the measurement by some driving code calling a dedicated function.[3] In either case parameters of the measurement can be passed upon instantiation of the measurement construct.

8.3 Visible Objects

We now introduce our solution to the problem of testbench measurements, which comes in the form of four measurement views. Their operation is described in each programming context. We use a syntax akin to the Verilog language; what is described here can be implemented in Verilog, VHDL, or in a software programming language like C.

Consider $\mu(x, c)$ a measurement with real variable x and constant c as parameters. Our four measurement views of $\mu(x, c)$ consist of a set of functions, a task, a module, and a class. Each of these views gives us *the same* functionality, yet is available in different contexts according to HDL and HVL restrictions.

8.3.1 Set of Functions

According to this view, a measurement is a set of seven functions that closely follow the mathematical description of Section 8.2.1. There are specificities, however, coming from procedural languages. In particular, the same abstract measurement can be applied to different signals, or with different constant parameters. We distinguish instances by

[2] Several input signals could be considered; a single one is sufficient for the purpose of introducing our approach.

[3] We find that the first option is suitable for static settings, where the lifetime of all variables is the whole simulation. Conversely the second option suits best in dynamic settings, where the variables lifetime can be shorter than the simulation.

numbering them in a unique fashion. This numbering scheme allows the user to specify, for each action, to which measurement instance it applies. That number can be for example a long integer, or of any other data type sufficiently large to distinguish between measurement instances. The state of the measure is stored implicitly and does not need to be handled by the user.

The first step in using a measure set of functions is to declare this number, with the statement **longint** *number* for example. The initialization of measurement μ proceeds with a dedicated function INITIALIZE_μ, declared as follows:

function $\text{INITIALIZE}_\mu(number, c)$

The update of measurement μ with a new value of the signal denoted x is obtained by function UPDATE_μ, declared as follows:

function $\text{UPDATE}_\mu(number, x)$

The control of the measure is realized by three functions denoted ACTIVATE_μ, DEACTIVATE_μ, and RESET_μ, which require *number* as argument. Note that the passage of time may affect the value of the measure hence deactivating the measure, and not updating the measure are not equivalent.

The measure μ is not necessarily defined at all times, for example in the case of a discrete measure. Whether the measure is defined at the current time is answered by function VALID_μ by calling $\text{VALID}_\mu(number)$, which returns the corresponding Boolean value. At time when μ is defined, its value is obtained via function EVALUATE_μ declared as follows:

function $\text{EVALUATE}_\mu(number)$

The implementation can ensure that a measurement is initialized before it is used, and returns an error if the measurement is evaluated when not defined.

8.3.2 Task

In this view, the measurement $\mu(x, c)$ consumes values by reading the input quantity x according to a sampling clock p, and produces values by writing some quantity y and producing an edge in sampling clock q. These sampling clocks are simply Boolean signals, with the associated time sequence given by both of their edges, rising and falling.

The use of input and output sampling clocks becomes necessary when such quantities represent discrete-time signals, and thus are not everywhere defined. For continuous-time signals, the time sequence given by such clocks may also be necessary when using a representation in terms of samples and interpolation. Note that quantities managed by the digital simulator are interpreted as piecewise constant, and thus in this case one may use the event associated with their change of value as a sampling clock.

The measurement is controlled by additional Boolean variables e and f as follows. Variable e is used as an enable signal, which activates the measurement on its rising edges, and deactivates it on its falling edges. Variable f is the restart signal, that causes the measure to be reset on the occurrence of both its edges, rising and falling.

The prototype of a measurement task is as follows:

task $\text{WRAPPERTASK}_\mu(x, p, y, q, e, f, c)$

where x, p are the input signal and clock, y, q are the output signal and clock, e is the enable signal, f is the restart signal, and c is a constant parameter argument to the measurement. The task is called by the user at the beginning of the simulation, and runs for its entire duration. This task consumes the simulation time and updates its output signals whenever the input signals they depend upon change.

8.3.3 Module

The use model of a measurement module is similar to that of a measurement task. The module takes a signal as input and produces a signal as output. Signals are represented as a pair of variables, real and Boolean. Note that this representation is necessary for discrete-time signals, that may have identical consecutive values. Under the semantics of digital simulation, the piecewise interpolation of such signals merges such values in one linear segment, and the original discrete-time signal cannot be reconstructed.[4]

We treat measurement constant parameters as proper parameters (not changing during the entire simulation), so as to guarantee a consistent behavior. The measurement module has the following prototype:

```
**module** WRAPPERMODULE$_\mu$ #$(c)$ $(x, p, y, q, e, f)$
```

As for the task view, we denote x, p the input signal and clock, y, q the output signal and clock, and e, f the enable and restart signals. Constant c is a parameter argument to the measurement. The measurement defines a set of sampling points with value given by y on the edges of signal q.

8.3.4 Class

In this view, the measurement is a single class that provides as set of functions for computing the measure. We rely for this on the decomposition of measurements of Section 8.2.1, and define measurement classes following the UVM standard for the definition of monitors. The input and output of monitors are transactions, with a dedicated type also known as UVM sequence item. In the case of real signals such as the input signal and the output measured value, transactions represent signal samples. This class has two fields, containing absolute time t and real value v:

```
class sample extends sequence_item
    real $t$
    real $v$
end class
```

The control part of the measurement uses empty transactions declared with class names *activate, deactivate,* and *reinitalize,* whose typing is sufficient to command the corresponding action. This measurement class has the structure of a UVM monitor. The class is given ports corresponding to each of these inputs and outputs: x for the input samples, y for the output samples, d, e for the deactivate and activate transactions, and f for the reset transactions. Its constructor takes as argument the constant real c that parameterizes the measure. The instantiation of the measurement then simply consists

[4]In VHDL, there are ways to detect assignments even when they do not change a value; that is not the case in Verilog.

in instantiating the measurement class. The declaration of the measurement class for μ is as follows.

```
class WRAPPERCLASS$_\mu$ extends monitor
    port #(sample) $x, y$
    port #(deactivate) $d$
    port #(activate) $e$
    port #(reset) $f$
end class
```

In order for the measurement class instance to observe electrical or real quantity x, it will need a sampling agent to transform it into a series of transactions of type *sample*. This sampling agent consists in a UVM monitor that samples values of signal x according to an input clock – the sampling clock may be derived from the input signal itself. In return it will produce transactions holding the time-stamped values of the observed signal.

8.4 Implementation

In this section we propose a framework for implementing measurements core algorithms, and sketch an implementation of wrappers to obtain the measurement views of Section 8.3 in this context.

8.4.1 Core

Let us fix a measurement μ. We assume that the measurement computation is broken down into functions r_μ and s_μ, and constant a_μ as described in Section 8.2.1. Those functions may use the additional argument c for cases where the measure is a template parameterized by some constant c. In that case the constant a_μ becomes a function of c. The internal state of μ is stored in a data structure M, whose type depends on the state space $\mathbb{S}$ of measurement μ. In addition let us define a function $v_\mu \in \{0,1\}^{\mathbb{T}}$, such that $v_\mu(M) = 1$ when the measure μ is defined (discrete measure) or produces a sampling point (continuous measure), 0 otherwise.

Algorithms for measurement μ can always be implemented this way. Constant a_μ holds the initial value of M. Whenever a new input sample (or segment) $x_{\boldsymbol{u}}(t)$ is acquired, the function s_μ aggregates the new value of x into M, according to an update of the from $M := s_\mu(M, (t, x_{\boldsymbol{u}}(t)))$. Function v_μ is a Boolean function that indicates whether the measurement is defined in that instant (whether the property μ is measurable). Finally r_μ gives the measured value based on the current state M. If the measure is undefined the function r_μ produces an estimate, or an incoherent value.

Let us take for example a threshold crossing measurement.

Example 8.1. *Measurement μ with parameters x, c computes the times at which some real signal x crosses the threshold c upward. The state of the measurement core is a tuple $M = (m_0, m_1, m_2, m_3, m_4)$ of real values m_0, m_1, m_2, m_3 storing the current time, current value, previous time, previous value, and real value m_4 storing the threshold c. We initialize the measurement state by*

$$a_\mu(c) := (0, 0, 0, 0, c)$$

The state is updated by

$$s_\mu(M,(t,r)) := (t,r,m_0,m_1,m_4)$$

with (m_0,m_1) *the previous sample. The time of crossing is computed based on the state M by*

$$r_\mu(M) := \begin{cases} m_2 + (m_0 - m_2)\dfrac{m_4 - m_3}{m_1 - m_3} & \textit{if } m_1 - m_3 \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

Such a time of crossing is only relevant when the last two samples lie at opposite sides of the threshold, that is

$$v_\mu(M) := \begin{cases} 1 & \textit{if } m_3 \leq m_4 \leq m_1 \\ 0 & \textit{otherwise} \end{cases}$$

8.4.2 Wrappers

We now describe the how measurements views are connected to the measurement core.

Set of Functions The implementation of some measurement μ as the previously described set of functions INITIALIZE$_\mu$, UPDATE$_\mu$, ACTIVATE$_\mu$, DEACTIVATE$_\mu$, RESET$_\mu$, and EVALUATE$_\mu$ is transparent. Such HDL system functions simply call directly the measurement core implementation, managing the different instances of the same measure accordingly.

Task For measurement tasks, the functionality of the wrapper consists of the following parts: (1) initialize the measurement core; (2) sample the input signal according to the input clock; (3) update the measurement core when a new sample is acquired; (4) activate, deactivate, or reset the measurement core when corresponding events occur. These functions can be achieved by the following event-driven code:

```
task WRAPPERTASK$_\mu(x,p,y,q,e,f,c)$
  $M := a_\mu(c)$
  forever @($\updownarrow p$ or $\updownarrow e$ or $\updownarrow f$)
    if $\updownarrow f$ then
      $M := a_\mu(c)$
    end if
    if $\updownarrow p$ and $e$ then
      $M := s_\mu(M,x)$
      $y := r_\mu(M)$
      if $v_\mu(M)$ then
        $q = \neg q$
      end if
    end if
  end forever
end task
```

In our prototype implementation the tasks wrappers are programmed using the Verilog Procedural Interface (VPI) [1]. This enables using an unbounded amount of memory, and makes it possible for the code to call analog simulator internals.

Module The interface of a measurement module is similar to that of the task. We build a wrapper that calls the measurement task, instead of the measurement core. This demonstrates the flexibility in this approach. A verification engineer may want to define a different interface to a measurement module, or additional control signals. To achieve that effect it is sufficient to modify the wrapper that we provide. The module wrapper simply calls the task wrapper at the start of simulation:

```
module $\text{WRAPPERMODULE}_\mu$#$(c)$ $(x, p, y, q, e, f)$
    real $x, y$
    logic $p, q, e, f$
    input $x, p, e, f$
    output $y, q$
    initial $\text{WRAPPERTASK}_\mu(x, p, y, q, e, f, c)$
end module
```

Class Wrappers for measurement classes are built using the view of a measurement as a set of functions. Our implementation of task wrappers is based on the structure of UVM; the measurement uses transactions to interact with its environment. When measuring some analog signal $V(n)$ such transactions are obtained via an additional agent that we define via the class *analog monitor* with the following skeleton:

```
class analog_monitor extends monitor #(sample)
    port $x$
    ...
    task body
        forever @$(p)$
            sample $m = \mathrm{V}(n)$
            $x$.write$(m)$
        end forever
    end task
end class
```

This agent is then connected via its analysis port to the measurement class instance. The implementation of the class wrapper consist in monitoring each port and taking the corresponding action by calling core functions; details are language specific and we omit the pseudo-code.

8.5 Case Study

A Phased-Locked Loop (PLL) is a circuit that generates an *output* periodic signal synchronized with some *input* periodic signal. We use a PLL design based on the textbook example of [27], such that the output period is $\frac{1}{16}$ times the input period. Two other signals are of interest. The *locked* signal is high if and only if the PLL output signal has a stable period that closely satisfies the specification. The *supply voltage* signal also influences the correct operation of the circuit. We denote x the input signal, y the output signal, p the locked signal, and z the supply voltage. Variables x, y, and z are interpreted as real signals, while variable p is interpreted as a Boolean signal.

Two versions of the design are considered in alternation: a real number model, and a SPICE model. The real number model only emulates the voltage and control behavior of the PLL. These two version of the DUT can be evaluated using the same testbench, which indicates that the proposed methodology can be used across abstraction levels.

The design is given some random input stimuli parameterized by:

- period of x
- duty cycle of x
- constant voltage at z.

The testbench is written in SystemVerilog and using UVM, and we develop measurement classes following the definitions of Section 8.4.2. We aim to take two measures during the simulation: the *relative jitter* and the *locking time*, and check some operating area safety condition.

Relative Jitter We define the relative jitter as the standard deviation of the pseudo-period in the output voltage y of the PLL, computed over a sliding window of 20 values when the p signal is high. The period is taken as the time difference between two successive crossings of threshold v_{th}, some real value representing a middle threshold voltage. To perform the measurement we use a *period* measurement class. We create an instance of this measurement class that we add to the environment, along with an analog monitor that we connect to y by hierarchical name. We chain the period measurement with a jitter measurement class using an input – output composition.

Locking Time The locking time is defined as the time between a parameter change in the input stimuli generator and the subsequent positive edge of the signal p at the output. We define a *duration* measurement class, controlled by an activation event that triggers a continuous time clock to measure the time elapsed, a deactivation event whose effect is to fix the output value to the current value, and a resetting event that sets the clock to 0. The measurement class instance is controlled by the following events. The activation event is given as the disjunction of all events attached to the change of input parameter values. The deactivation event is defined by the rising edges of p. The resetting event is tied to the activation event.

Safe Operating Area The PLL is considered safe if z is not above v_{unsafe} for more than d_{max} consecutive time units. We use the duration measurement agent with the following control signals: the *reset* and *activate* events are tied to upward crossings of v_{unsafe} by z, and the *deactivate* event is defined by negative downward crossings of v_{unsafe}. We use a SystemVerilog assertion to check that the duration is within acceptable range d_{max}.

The overhead caused by such measurements is not observable relative to the simulation. The simulation of the real-number model is almost instantaneous, while the simulation of the SPICE version of the design takes time ≈ 300 seconds. The computational performance of our algorithms is due to the fact that measurements do not interfere with the analog simulation. In the setting of mixed-signal simulation, this means that no additional analog solution points are requested from the analog solver.

Conversely, performing such measurements *online* can save simulation time by stopping the simulation early in case of violation. This is already the case for other forms of measurements already implemented in commercial simulators, in particular safe operating area checks. What our method enables, is to extend the scope of online measurement to complex ones, formed by input output composition or other forms of combination. By allowing the measurements to be inserted in interfaces, modules, or classes of the testbench, one also makes sure they are reused considering the same circuit at a different level of abstraction, something we also confirmed is possible.

To sum up, we improved the state of the art in measuring analog behavior in a digital mixed-signal context, through the following principles. The first key aspect of our approach, is the ability to control a measurement using events. Such events come directly from the digital simulation process; the measurement is causal and can thus be done in an online fashion. Measurements are accessible through various views, this is the second key point of our approach. That way it can be used in several contexts: in the modeling code, testbench code or configuration code. Here we focused on the testbench part, which is novel. The third ingredient, is that algorithms used to compute the measure are the same irrespectively of which view of the measurement is considered. This enables perfect integration with existing digital verification practice, while retaining most of the guarantees of simulator-implemented measurements.

Conclusion

Summary

The verification of systems featuring some interaction between continuous and discrete behaviors, such as mixed-signal integrated circuits, poses a great source of challenges. One obvious flaw of existing tools in the hardware domain is the lack of specification language encompassing both features simultaneously. We began by describing how real-time extensions of temporal logic and regular expressions can integrate naturally to discrete-time specification languages, prolonging the work of [63] to digital assertions. Contrarily to the majority of previous research in mixed-signal verification, primarily focused on the integration of discrete and continuous aspects, we postulate that existing real-time specification languages can be sufficient for the purpose of specifying continuous and mixed behavior. This assumption is backed by previous research on Metric Temporal Logic (MTL); we built upon results obtained in [95] and elsewhere in several ways.

First we extend the monitoring of temporal logic with a diagnostic procedure. That procedure is original in the setting of simulation-based verification; it does not attempt to explain the property violation other than by looking at the trace and the formula. Second we improve on previous algorithms for the robust monitoring of Signal Temporal Logic. The robustness of some specification relative to some trace is an indicator of the distance to violation. Our algorithms have been used in numerous situations and form the core of an efficient tool for the falsification of mixed-signal properties [44]. Third we propose a monitoring procedure similar to that of MTL for the monitoring of Timed Regular Expressions (TRE). The syntax of TRE is intuitive proceeding by simple extension of classical regular expressions; the expressiveness of TRE is incomparable to that of MTL, as witnessed by the ability of TRE to generate interleaved sequential and timing checks. Having proposed aforementioned extensions, we feel that a solid base is available for the specification of mixed-signal behaviors.

The monitoring of continuous or mixed-signal systems is not excursively concerned with correctness. Performance values, confidence estimates or other quantitative indicators are also routinely used (under the name of measurements) in analog verification. We invented a declarative measurement language that is based on a decomposition between behavior identification, using TRE, and aggregation of values, using simple continuous operations. This measurement language is well suited for the specification of mixed-signal properties, which feature both sequential conditions and continuous aspects. We finally study the problem of further integrating analog and digital verification practice. In this regard, previous research attempted to extend digital constructs towards continuous behavior but largely ignored the current practice in analog verification, which makes for a reliable framework for the analysis of continuous behaviors. In opposition, we aimed at importing analog measurements in a flexible way into the digital domain. Our solution provides the basic architecture to convert measures into verification modules, classes, tasks, and other digital constructs. This enables reusing analog verification artifacts in digital verification environments, and further automate the checking of analog properties throughout the development cycle.

Perspectives

Continuous-Time Signals We identified the opposition between *asynchronous* and *synchronous* as the main differentiator between analog and digital circuits temporal behavior. In order to treat asynchronous aspects we favored a continuous-time representation of signals, and duration constraints based on interval of timing values. We argue this choice allows to faithfully represent continuous systems as having a state defined at every time instant. The use of a fixed sampling in standard assertion languages prevents the detection of errors occurring in between sampling points.

Symbolic Representation A key to analyzing continuous-time signals is to represent timing values in a symbolic fashion. Symbolic techniques are used routinely in formal verification, and in particular with Timed Automata and other real-time extensions of finite-state systems [22]. We show that the same representation, namely that of *zones* can be used profitably in the monitoring of TRE. We make the explicit comparison with MTL, where the symbolic representation uses time *intervals*. Enhancements of MTL monitoring that we consider use different symbolic representations. Robust monitoring is based on piecewise-linear signals, while diagnostics relies on what we may call *explanation signals*, which map time to implicants in a piecewise-algebraic fashion.

Inductive Procedures Our general solution to problems related to the monitoring of temporal specifications proceeds inductively on the structure of the specification. This requires first to identify the type of information required by the induction, for instance the satisfaction set for temporal logic, the match set for regular expressions, and other types of data represented symbolically as mentioned. Then the computation relies on one procedure per operator in the specification language under consideration. This makes for a simple approach, and on that can be reused for the solution of many related problems. It would be straightforward in particular to extend robust monitoring and diagnostics to TRE following such remarks.

Modularity and Approximation The advantage of algorithms working inductively on the specification structure, is that they easily extend to accommodate new specification operators. A limitation in the case of robustness and diagnostic computation, is that by treating subformulas independently where they share atomic propositions, the outcome ignores some logical dependencies. The quantitative semantics provides an under-estimate of the distance to violation, while the diagnostic provides a small set of segments causing the violation and not a minimal one.

Complexity and Online Computation In the setting of simple monitoring, the outcome of inductive computations is exact, but does not exploit the intricate temporal dependency of some formula (respectively expression) over its subformulas (resp. subexpressions). This does not affect the worst-case complexity of MTL monitoring algorithms. Computing the entire match set of TRE however has a worst-case complexity super-linear in the length of the trace. In order to guarantee linear complexity we can proceed instead inductively on the “structure of time”, and compute the satisfaction each subexpression interval per interval. We contributed to development of this principle, fully exposed in [111], and obtain an online monitoring algorithm for TRE. Further investigations are needed to provide a fully online approach to monitoring real-time extensions of digital assertions.

Interaction of Discrete and Continuous In our treatment of analog signals with declarative measurements, we adopt the point of view according to which mixed-signal behaviors feature sequential aspects that are finite-state, and continuous aspects that aggregate time or values via summation or maximum / minimum operations. The interplay of both aspects could be subjected to further investigations in the line of [20]. In particular we would like to define the composition of two measurements in this setting, by allowing some real value computed by the first to be used in the definition of the second. In our view the domain of mixed-signal measures is still lacking a simple, well-behaved computational model as a foundation as real-time systems do with timed automata, something we would like to investigate further.

Future Measurements In order to improve existing verification technology in the specific context of mixed-signal circuits, we aimed at providing verification engineers with tools usable in the setting of a digital testbench, yet providing the same guarantees as established analog measurements algorithms. We followed the practice of analog measurements, defined with a *past* temporal view in which the value of some measure at time t corresponds to the prefix up to t. Note that a similar approach can be taken for qualitative properties, and is another way to obtain online monitoring algorithms. Further integration of measurements with existing assertions would require to consider future-oriented quantitative properties. In that setting, the non-determinism that can occur in measurements as we defined them, would need new techniques in order to be resolved.

Bibliography

[1] IEEE standard for verilog hardware description language. *IEEE Std 1364-2005*, pages 1–560, 2006.

[2] IEEE standard VHDL analog and mixed-signal extensions. *IEEE Std 1076.1-2007*, pages 1–328, 2007.

[3] HSPICE reference manual: Commands and control options. *Synopsys*, 2008.

[4] IEEE standard for VHDL language reference manual. *IEEE Std 1076-2008*, pages 1–626, 2009.

[5] DSI3 bus standard. *DSI Consortium*, 2011.

[6] IEEE standard for property specification language (PSL). *IEEE Std 1850-2010*, pages 1–184, 2012.

[7] IEEE standard for SystemVerilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2012*, pages 1–1315, 2013.

[8] Standard universal verification methodology class reference. *Accellera Std UVM 1.2*, pages 1–852, 2014.

[9] Verilog-AMS language reference manual. *Accellera Std Verilog-AMS 2.4*, pages 1–437, 2014.

[10] Eldo reference manual. *Mentor Graphics*, 2016.

[11] Questa ADMS reference manual. *Mentor Graphics*, 2016.

[12] Antara Ain, Antonio Bruto da Costa, and Pallab Dasgupta. Feature indented assertions for analog and mixed-signal validation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[13] Antara Ain and Pallab Dasgupta. Monitoring AMS simulation: from assertions to features. In *VLSI Design*, pages 429–434, 2015.

[14] Stephen Altschul, Warren Gish, Webb Miller, Eugene Myers, and David Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, pages 403–410, 1990.

[15] Rajeev Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.

[16] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, David L Dill, and Howard Wong-Toi. Minimization of timed transition systems. In *Concurrency Theory*, pages 340–354. Springer, 1992.

[17] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[18] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". *ACM Transactions on Computational Logic*, 2(3):388–407, 2001.

[19] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[20] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. regular programming for quantitative properties of data streams. In *European Symposium on Programming Languages and Systems*, pages 15–40. Springer, 2016.

[21] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, pages 74–106. Springer, 1992.

[22] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *Computer Aided Verification*, pages 419–434, 2000.

[23] Yashwanth Annpureddy, Che Liu, Georgios 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*, pages 254–257, 2011.

[24] Eugene Asarin, Paul Caspi, and Oded Maler. A Kleene theorem for timed automata. In *Logic in Computer Science*, pages 160–171, 1997.

[25] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *Journal of ACM*, 49(2):172–206, 2002.

[26] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Ničković. Parametric identification of temporal properties. In *Runtime Verification*, pages 147–160, 2011.

[27] R. Jacob Baker. *CMOS: circuit design, layout, and simulation*, volume 1. John Wiley & Sons, 2008.

[28] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni, and Richard J. Trefler. Explaining counterexamples using causality. In *Computer Aided Verification*, pages 94–108, 2009.

[29] Joël Besnard, Pascal Bolcato, Dézai Glao, and Hervé Guegan. Simulation des circuits analogiques et mixtes. *Techniques de l'ingénieur*, 6(E3450), 2009.

[30] Bishnupriya Bhattacharya, John Decker, Gary Hall, Nick Heaton, Yaron Kashai, Neyaz Khan, Zeev Kirshenbaum, and Efrat Shneydor. *Advanced verification topics*. Cadence Design Systems, 2012.

[31] Prabal K. Bhattacharya, Swapnajit Chakraborti, Scott Little, Donald O'Riordan, and Vaibhav Bhutani. Bringing continous domain into SystemVerilog covergroups. In *Design and Verification Conference*, pages 1–8, 2012.

[32] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *Foundations of Software Technology and Theoretical Computer Science*, pages 432–443. Springer, 2005.

[33] Robert Boyer, Matt Kaufmann, and J. Strother Moore. The boyer-moore theorem prover and its interactive environment. *Computers & Mathematics with Applications*, 29(2):27–62, 1995.

[34] Stephen Boyer and J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 1977.

[35] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, and David L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conference*, pages 46–51. IEEE, 1990.

[36] Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Transactions on Computational Logic*, 11(4):23, 2010.

[37] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*, pages 52–71. Springer, 1981.

[38] Ben Cohen, Srinivasan Venkataramanan, and Ajeetha Kumari. *Using PSL/Sugar for formal and dynamic verification: Guide to Property Specification Language for Assertion-based Verification*. VhdlCohen Publishing, 2004.

[39] Ben Cohen, Srinivasan Venkataramanan, and Ajeetha Kumari. SystemVerilog assertions handbook: for formal and dynamic verification. 2005.

[40] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology*. World Scientific, 2002.

[41] Antonio Anastasio Bruto da Costa and Pallab Dasgupta. Formal interpretation of assertion-based features on AMS designs. *IEEE Design & Test*, 32(1):9–17, 2015.

[42] Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the skorokhod metric. In *Computer Aided Verification*, pages 234–250. Springer, 2015.

[43] Petr Dluhos, Lubos Brim, and David Safránek. On expressing and monitoring oscillatory dynamics. In *Hybrid Systems Biology*, pages 73–87, 2012.

[44] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170, 2010.

[45] Alexandre Donzé, Thomas Ferrère, and Oded Maler. Efficient robust monitoring for STL. In *Computer Aided Verification*, pages 264–279. Springer, 2013.

[46] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems*, pages 92–106, 2010.

[47] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *Formal Modeling and Analysis of Timed Systems*, 2004.

[48] Cindy Eisner and Dana Fisman. *A practical introduction to PSL*. Springer, 2006.

[49] Cindy Eisner, Dana Fisman, John Havlicek, Anthony McIsaac, and David Van Campenhout. The definition of a temporal clock operator. In *International Colloquium on Automata, Languages, and Programming*, pages 857–870. Springer, 2003.

[50] E. Allen Emerson and Richard J. Trefler. Parametric quantitative temporal reasoning. In *Logic in Computer Science*, pages 336–343, 1999.

[51] Javier Esparza, Pierre Ganty, and Tomáš Poch. Pattern-based verification for multithreaded programs. *ACM Transactions on Programming Languages and Systems*, 36(3):9:1–9:29, September 2014.

[52] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In *Formal Approaches to Software Testing and Runtime Verification*, pages 178–192. Springer, 2006.

[53] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[54] Georgios E. Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using S-TaLiRo. In *American Control Conference*, pages 3567–3572, 2012.

[55] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Conference on Management of Data*, 1994.

[56] Thomas Ferrère, Oded Maler, and Dejan Ničković. Trace diagnostics using temporal implicants. In *Automated Technology for Verification and Analysis*, pages 241–258. Springer, 2015.

[57] Thomas Ferrère, Oded Maler, Dejan Ničković, and Dogan Ulus. Measuring with timed patterns. In *Computer Aided Verification*, pages 322–337. Springer, 2015.

[58] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of computer and system sciences*, 18(2):194–211, 1979.

[59] C. William Gear. The automatic integration of ordinary differential equations. *Communications of the ACM*, 14(3):176–179, 1971.

[60] Georges G.E. Gielen and Rob A. Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825–1854, 2000.

[61] Aarti Gupta. Assertion-based verification turns the corner. *IEEE Design & Test of Computers*, 19(4):131–132, 2002.

[62] Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach: Part 1: Causes. In *Uncertainty in Artificial Intelligence*, pages 194–202, 2001.

[63] John Havlicek and Scott Little. Realtime regular expressions for analog and mixed-signal assertions. In *Formal Methods in Computer Aided Design*, pages 155–162. FMCAD Inc, 2011.

[64] Thomas A. Henzinger and Jan Otop. From model checking to model measuring. In *Concurrency Theory*, pages 273–287, 2013.

[65] Yoram Hirshfeld and Alexander Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62(1):1–28, 2004.

[66] John E Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman, 1st edition, 1990.

[67] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit Seshia. Mining requirements from closed-loop control models. In *Hybrid Systems: Computation and Control*, 2013.

[68] Kevin D. Jones, Victor Konrad, and Dejan Ničković. Analog property checkers: a DDR2 case study. *Formal Methods in System Design*, 36(2):114–130, 2010.

[69] Neyaz Khan, Yaron Kashai, and Hao Fang. Metric driven verification of mixed-signal designs. In *Design and Verification Conference*, 2011.

[70] Stephen Cole Kleene. Representation of events in nerve nets and finite automata. Technical report, DTIC Document, 1951.

[71] Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, pages 323–350, 1977.

[72] Manolis Koubarakis. Complexity results for first-order theories of temporal constraints. In *Principles of Knowledge Representation and Reasoning*, volume 94, pages 379–390, 1994.

[73] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[74] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[75] Ken Kundert, Henry Chang, Dan Jefferies, Gilles Lamant, Enrico Malavasi, and Fred Sendig. Design of mixed-signal systems-on-a-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1561–1571, 2000.

[76] William K. Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall PTR, 2005.

[77] Stefan Lämmermann, Alexander Jesser, Martin Rathgeber, Jürgen Ruf, Lars Hedrich, Thomas Kropf, and Wolfgang Rosenstiel. Checking heterogeneous signal characteristics applying assertion-based verification. *Frontiers in Analog CAD*, 2009.

[78] Ville Laurikari. NFAs with tagged transitions, their conversion to deterministic automata and application to regular expressions. In *String Processing and Information Retrieval*, pages 181–187, 2000.

[79] Daniel Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *Nordic Journal of Computation*, 13(4):328–339, 2006.

[80] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[81] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. The glory of the past. In *Workshop on Logic of Programs*, pages 196–218. Springer, 1985.

[82] Scott Little and Martin Vlach. SystemVerilog-AMS: The future of analog/mixed-signal modeling. In *Design and Verification Conference*, 2016.

[83] Oded Maler and Dejan Ničković. Monitoring temporal properties of continuous signals. In *Formal Modeling and Analysis of Timed Systems*, pages 71–76, 2004.

[84] Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *STTT*, 15(3):247–268, 2013.

[85] Oded Maler and Dejan Ničković. Monitoring properties of analog and mixed-signal circuits. *Software Tools for Technology Transfer (STTT)*, 15(3):247–268, 2013.

[86] Oded Maler, Dejan Ničković, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of Computer Science*, pages 475–505, 2008.

[87] Nicolas Markey and Jean-François Raskin. Model checking restricted sets of timed paths. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 432–447, 2004.

[88] Subhankar Mukherjee and Pallab Dasgupta. Assertion aware sampling refinement: a mixed signal perspective. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, November 2012.

[89] Subhankar Mukherjee and Pallab Dasgupta. Computing minimal debugging windows in failure traces of AMS assertions. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(11):1776–1781, 2012.

[90] Subhankar Mukherjee, Pallab Dasgupta, Siddhartha Mukhopadhyay, Scott Little, John Havlicek, and Srikanth Chandrasekaran. Synchronizing ams assertions with ams simulation: From theory to practice. *ACM Transactions on Design Automation of Electronic Systems*, 17(4):38:1–38:25, 2012.

[91] Rajdeep Mukhopadhyay, S.K. Panda, Pallab Dasgupta, and John Gough. Instrumenting AMS assertion verification on commercial platforms. *ACM transactions on Design Automation of Electronic Systems*, 14(2):21:1–21:47, April 2009.

[92] Laurence William Nagel and Donald O. Pederson. *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California, 1973.

[93] Farid N. Najm. *Circuit Simulation*. John Wiley & Sons, 2010.

[94] Thang Nguyen and Dejan Ničković. Assertion-based monitoring in practice – checking correctness of an automotive sensor interface. In *Formal Methods for Industrial Critical Systems*, pages 16–32, 2014.

[95] Dejan Ničković. *Checking Timed and Hybrid Properties: Theory and Applications*. PhD thesis, Joseph Fourier University, Grenoble, 2008.

[96] Dejan Ničković and Oded Maler. AMT: A property-based monitoring tool for analog systems. In *Formal Modeling and Analysis of Timed Systems*, pages 304–319, 2007.

[97] Jürg Nievergelt and Franco P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25(10):739–747, 1982.

[98] Donald O'Riordan and Prabal Bhattacharya. PSL/SVA assertions in SPICE. In *The Design and Verification Conference and Exhibition*, 2012.

[99] Joël Ouaknine, Alexander Rabinovich, and James Worrell. Time-bounded verification. In *Concurrency Theory*, pages 496–510. Springer, 2009.

[100] Rob Pike. The text editor sam. *Software: Practice and Experience*, 17(11):813–845, 1987.

[101] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science*.

[102] Arthur N. Prior. *Past, present and future*, volume 154. Clarendon Press Oxford, 1967.

[103] Thomas W. Reps, Alexey Loginov, and Shmuel Sagiv. Semantic minimization of 3-valued propositional formulae. In *Logic in Computer Science*, pages 40–51. IEEE Computer Society, 2002.

[104] 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*, pages 251–268. Springer, 2008.

[105] Regis Santonja. Re-usable continuous-time analog sva assertions. In *CNDLive! EMEA*, 2012.

[106] Chris Spear. *SystemVerilog for Verification*. Springer, 2006.

[107] Graham A. Stephen. *String searching algorithms*. World Scientific, 1994.

[108] Prasanna Thati and Grigore Roşu. Monitoring algorithms for metric temporal logic specifications. *Electronic Notes in Theoretical Computer Science*, 113:145–162, 2005.

[109] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, pages 419–422, 1968.

[110] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Timed pattern matching. In *Formal Modeling and Analysis of Timed Systems*, pages 222–236, 2014.

[111] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Online timed pattern matching using derivatives. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 736–751. Springer, 2016.

[112] Dogan Ulus and Alper Sen. Using haloes in mixed-signal assertion based verification. In *IEEE International High Level Design Validation and Test Workshop*, 2012.

[113] Srikanth Vijayaraghavan and Meyyappan Ramanathan. *A practical guide for SystemVerilog assertions*. Springer, 2006.

[114] Farn Wang. Parametric timing analysis for real-time systems. *Information and Computation*, 130(2):131–150, 1996.

[115] Harry Wang, Wessam El-naji, and Kenneth Bakalar. Experience with OVM-based mixed-signal verification of the impedance calibration block for a DDR interface. In *Design and Verification Conference*, 2012.

[116] Peter Weiner. Linear pattern matching algorithms. *Switching and Automata Theory*, 1973.

Abstract This thesis is concerned with the monitoring of mixed-signal circuit simulations. In this setting we make several theoretical and practical contributions as follow, with particular emphasis in the area of timed specification languages. We give efficient algorithms for computing the distance from some simulation traces to temporal logic formulas. An original diagnostic procedure is provided for the systematic debugging of such traces. The monitoring of continuous behaviors is also extended to other forms of assertions based on regular expressions, forming the basis of our novel measurement language. We then show how analog measurements can be implemented in existing digital frameworks, overall extending current verification methodologies toward the mixed-signal domain.

Résumé L’objet de cette thèse est le monitorage de simulation de circuit en signaux mixtes analogique / digital. Dans ce contexte, et plus particulièrement dans le cadre des langages de spécification temporisée, nous apportons les contributions théoriques et pratiques suivantes. Nous proposons des algorithmes pour calculer la distance d’une trace de simulation à une formule de logique temporelle. Une procédure originale est donnée pour le débogage systématique d’une telle trace relativement à une formule de logique temporelle. Le monitorage des comportements continus est ensuite étendu à d’autres formes d’assertions, basées sur les expressions régulières, aussi à la base de notre langage de description de mesures. Nous montrons enfin comment en général des mesures analogiques peuvent être implantées dans l’environnement digital existant. Ce faisant, étendons le champ d’application des méthodologies existantes aux circuits en signaux mixtes.