THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques et Informatique**

Arrêté ministérial : 25 mai 2016

Présentée par

Dogan Ulus

Thèse dirigée par **Oded Maler**

préparée au sein **Laboratoire VERIMAG**
et de **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Pattern Matching with Time: Theory and Applications

Filtrage par Motif Temporisé:
Théorie et Applications

Thèse soutenue publiquement le **15 Janvier 2018**,
devant le jury composé de :

ABSTRACT

Dynamical systems exhibit temporal behaviors that can be expressed in various sequential forms such as signals, waveforms, time series, and event sequences. Detecting patterns over such temporal behaviors is a fundamental task for understanding and assessing these systems. Since many system behaviors involve certain timing characteristics, the need to specify and detect patterns of behaviors that involve quantitative timing requirements, called *timed patterns*, is evident. However, this is a non-trivial task due to a number of reasons including the concurrency of subsystems and the density of time.

The key contribution of this thesis is in introducing and developing *timed pattern matching*, that is, the act of identifying segments of a given behavior that satisfy a timed pattern. We propose timed regular expressions (TREs) and metric compass logic (MCL) as timed pattern specification languages. We first develop a novel framework, the algebra of timed relations, which abstracts the computation of time-related aspects. Then we provide offline matching algorithms for TRE and MCL over discrete-valued dense-time behaviors using this framework and study some practical extensions.

It is necessary for some application areas such as runtime verification that pattern matching needs to be performed during the actual execution of the system. For that, we provide an online matching algorithm for TREs based on the classical technique of derivatives of regular expressions. We believe that the underlying technique which combines derivatives and timed relations constitutes another major conceptual contribution to timed systems research.

Furthermore, we present an open-source tool MONTRE that implements our ideas and algorithms. We explore diverse applications of timed pattern matching over several case studies using MONTRE. Finally we discuss future directions and several open questions that emerged as a result of this thesis.

RESUME

Les systèmes dynamiques présentent des comportements temporels qui peuvent être exprimés sous diverses formes séquentielles telles que des signaux, des ondes, des séries chronologiques et des suites d'événements. Détecter des motifs sur de tels comportements temporels est une tâche fondamentale pour comprendre et évaluer ces systèmes. Étant donné que de nombreux comportements du système impliquent certaines caractéristiques temporelles, le besoin de spécifier et de détecter des motifs de comportements qui implique des exigences de synchronisation, appelées motifs temporisés, est évidente. Cependant, il s'agit d'une tâche non triviale due à un certain nombre de raisons, notamment la concomitance des sous-systèmes et la densité de temps.

La contribution principale de cette thèse est l'introduction et le développement du *filtrage par motif temporisé*, c'est-à-dire l'identification des segments d'un comportement donné qui satisfont un motif temporisé. Nous proposons des expressions rationnelles temporisées (TRE) et la logique de la boussole métrique (MCL) comme langages de spécification pour motifs temporisés. Nous développons d'abord un nouveau cadre qui abstraite le calcul des aspects liés au temps appelé l'algèbre des relations temporisées. Ensuite, nous fournissons des algorithmes du filtrage hors ligne pour TREs et MCL sur des comportements à temps dense à valeurs discrètes en utilisant ce cadre et étudions quelques extensions pratiques.

Il est nécessaire pour certains domaines d'application tels que la vérification dynamique que le filtrage par motif doit être effectué pendant l'exécution réelle du système. Pour cela, nous fournissons un algorithme du filtrage en ligne pour expressions rationnelles temporisées basé sur la technique classique des dérivées d'expressions rationnelles. Nous croyons que la technique sous-jacente qui combine les dérivées et les relations temporisées constitue une autre contribution conceptuelle majeure pour la recherche sur les systèmes temporisés.

Nous présentons un logiciel libre MONTRE qui implémente nos idées et algorithmes. Nous explorons diverses applications du filtrage par motif temporisé par l'intermédiaire de plusieurs études de cas. Enfin, nous discutons des orientations futures et plusieurs questions ouvertes qui ont émergé à la suite de cette thèse.

ACKNOWLEDGMENTS

My doctoral studies have been satisfying and lucrative from many aspects. My laboratory Verimag, with its highly international environment, was a great place for all those years. I have indebted to many people and must thank each of them while margins or even full pages are not enough to express my gratitude fully.

First and foremost, I would like to thank my supervisor, Oded Maler, as I have learned many things from him directly, indirectly, mathematically, and culturally. All our discussions, all the books he gave me to read, and all witty remarks have contributed to my development in a particular way. Then, it was a great pleasure to work with Eugene Asarin, Thomas Ferrère, and Dejan Ničković. Without our fruitful discussions and their contributions, it would be much harder to achieve the results presented in this thesis. I am also thankful to Rajeev Alur and his group at UPenn for their hospitality during my valuable summer visit.

For this paragraph, I have reserved a truly big thanks for my colleagues and friends Irini, Jan, Abhinav, Stefano, Nikos, Alexandre, Marcelo, and Özgün as well as all other PhD students and the staff of Verimag I couldn't name now. All together we created a unique environment that I will always remember.

Finally I would like to thank to my parents, Ulviye and Merih, who always gave me their love and full support in life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

There are only patterns,
patterns on top of patterns,
patterns that affect other patterns.
Patterns hidden by patterns.
Patterns within patterns.

— Survivor, Chuck Palahniuk

Time, it needs time.

— Still Loving You, Klaus Meine

This thesis concerns patterns and how to detect them in temporal behaviors generated by various systems. These can be known patterns, good or bad patterns, correct or incorrect patterns, simple or complex patterns, but, more precisely, these are *timed patterns* expressed in some intuitive and well-defined formalisms, namely timed regular expressions and metric compass logic.

Let's assume what good, bad, correct, and incorrect patterns mean is known in the context. Then a few types of composition regulate how to form complex patterns out of simpler patterns in general. Particularly this process resembles how things formed in the nature from atoms to very large complex systems; therefore, it should not be surprising that a small set of composition rules over a set of elementary patterns can express many real-life phenomena. In order to convince yourself that patterns are everywhere in our lives, you may look at this page and count how many times the letter p is followed by other fellow letters, a, t, t, e, r, n, and s in the sequential order given. Indeed simple sequences constitute one of the most basic but extremely useful class of patterns, and sequences of textual characters (letters, punctuation marks, whitespaces, . . .) are commonly known as strings. You are lucky if you read this thesis on a computer so you can use the word search functionality[1] of your document reader to find and count all instances of the string "patterns". Such a computation of searching and locating strings over long texts is the classical example of string matching or —in more general— *pattern matching*.

Patterns in general would not be formed not only by sequential composition but also other types of compositions. A remarkable fact shown by Kleene in [60] is that an extremely important class of patterns, called regular patterns, can be represented using finite number of composition operations of three types: sequential composition (concatenation), alternative choice (union), and indefinite repetition (Kleene star). In the same paper Kleene proved that all and only regular patterns are recognized by finite-state automata [27, 77, 78, 93].

1 Usually Ctrl+F or Cmd+F starts the program. Currently 26 instances of the string "patterns" on this page (case insensitive) including this footnote.

This was the birth of the formalism of *regular expressions* and led to many theoretical and practical consequences. Two appraoches for automata construction from regular expressions [26, 76] emerged early in the following and it is shown later in [18] that both approaches are very closely related. In [107] Thompson presented a noteworthy implementation of these automata-theoretic concepts to search patterns expressed as regular expressions. Since then tools for regular expression matching have become established in various domains of computer science ranging from compilers to verification and biology.

Time, usually regarded as a sequence of temporal entities (time instants or periods), can be very naturally modeled using finite-state automata and regular expressions. Patterns on the time axis is similar to patterns in text if we think of letters as events and their positions in text as timestamps. Consider a sequence $\underset{1}{c}\underset{2}{a}\underset{3}{b}\underset{4}{c}\underset{5}{c}\underset{6}{a}\underset{7}{b}\underset{8}{c}$ of events observed at each second of time and a pattern $\varphi = ab$ meaning that an event a is followed by b after 1 second according to the time granularity given. Then, using pattern matching techniques, we can say that the pattern φ occurred over time periods $(2, 3)$ and $(6, 7)$. Thus we have learned that the pattern φ occurred twice and when these occurrences happened exactly in time. That's already a useful technique to reason about time and temporal information. Yet, observe, in general, that temporal events and patterns can occur simultaneously and many interesting patterns can be defined over such events and patterns happening at the same time. Therefore, parallel composition (intersection) of patterns is far more ubiquitous and important for patterns in time than texts. Particularly a specialized case of parallel composition can express constraints about durations of patterns, and we'll talk about duration constraints for patterns while explaining quantitative aspects of time. But, before that, we move to another important formalism for pattern specification involving time.

Temporal logic concerns reasoning about time and all kinds of temporal information in a logical framework. In the narrow sense, however, we denote by temporal logic the modal-logic approach introduced by Prior [91, 92] over a simple linear time model. Prior's logic, then, is a way to formalize tense aspects of common sentences such as "an event will be eventually followed by another event" or "a situation has been always the case". In his seminal paper [89], Pnueli imported the concept into computer science and proposed to use temporal logic to express properties over execution traces of (reactive, concurrent) programs (also see [65]). Afterwards, temporal logic quickly was adopted by the community as a major specification formalism in modern verification technology for model checking [31] and runtime verification [51, 68, 71] (sometimes accompanied with regular expressions [29, 40]). Model checking consists in exploring a mathematical model of a system exhaustively and providing a formal proof of correctness that all behaviors of the system satisfy (temporal logic) specifications. The downside, however, is that we often do not have a faithful model of large complex systems at the first place and, even when we have such a model, the exhaustive exploration over the model is usually infeasible. Being a lighter approach, runtime verification checks behaviors individually against specifications. It does not necessitate a model; therefore, it is applicable for a larger class of systems (including black-box systems) in practice. Although the formal proof of correctness cannot be provided in runtime verification (unless exhaustive), it can be approximated by employing some coverage-based and statistical metrics.

From the point of this thesis, temporal logics deliver yet another means to express patterns in time. For example, Pnueli's linear-time temporal logic (LTL) is proved to be very convenient to specify infinite-duration patterns of intentionally-never-terminating programs. The most important classes of these patterns are safety (bad things will never occur) and liveness (good things will always recur) properties. Checking these properties over system behaviors, as in runtime verification, can be easily viewed as pattern matching using point-based temporal logics. For example, notice that the satisfaction at a time point t by an LTL formula means that there is an instance of the pattern that occurs between t and the end of time. Therefore, existing verification techniques using point-based temporal logics provide a significant source of inspiration for design choices made and techniques developed in this thesis.

However, point-based temporal logics are not naturally suitable to express local patterns with durations (but aforementioned global patterns of safety and liveness). Although we acknowledge that time points are quite established in some disciplines of science such as physics, time periods seem better fit for reasoning about time in general. The topic can lead us to interesting debates (and see [17] for a comparative study) but it is clear for us that a choice has to be made when it comes to realizing these concepts elegantly and efficiently. Therefore, a philosophical and practical position that favors time periods instead of time points is going to be prevalent in this thesis, thus in accordance with Allen's theory of action and time [2, 3].

Considering time periods as primitive entities, Allen introduced thirteen basic relations between two time periods, and proposed a first-order logic over time periods that employs a predicate for each of the so-called Allen relations. However, first-order logic with all its explicit variables and binding mechanisms is much more complicated than modal logic systems. Besides its full expressiveness is usually not needed since modal logics are extremely well to express real-life structures, which are composed of smaller structures. Therefore, we consider modal logics over time periods as excellent logical frameworks to express temporal patterns while providing a good compromise between expressiveness and simplicity. We are then interested in the modal logic, called HS-logic or Compass Logic, introduced by Halpern and Shoham in [48]. Extending Prior's modal approach towards time periods and based on Allen's relations, it provides simple constructs to express some complex patterns such as a pattern that *precedes*, *meets*, *starts*, *ends*, *overlaps*, or occurs *during* another pattern. We also note that extending HS-system with concatenation would be very useful for pattern matching purposes and such an extension leads to an increase in the expressiveness as shown by Venema in [113].

One major contribution of this thesis is in introducing *timed pattern matching*, that is, pattern matching under the notion of quantitative time and timing constraints. For that we propose to use appropriate timed variants of regular expressions and temporal logics as pattern specification languages and we develop pattern matching techniques for these formalisms. It is possible to trace timed formalisms back to Prior's writings. However, the most influential works in the literature are about timed automata [4] that extend finite-state automata with clock (timer) constraints and metric temporal logic (MTL) [62] that extends LTL with bounds on temporal distances. Many variants of both formalisms are extensively studied and several implementations have appeared in last decades mostly for verification purposes.

These timed formalisms are generally motivated by the need to verify safety-critical real-time systems, which need to act correctly and timely for all possible situations without exception. Surely air traffic controllers, nuclear reactors, and autonomous vehicles are such systems and their correct operation may be deduced by an exhaustive analysis of their behaviors that exhibit many instances of different timed patterns, desirable or undesirable. On the other hand, we are also interested in exploring possible applications beyond the safety-critical domain. As sensors, monitors, and video cameras collect huge amounts of temporal data nowadays, it is crucial to extract meaningful information from temporal databases and streams. Clearly timing information and constraints are also important for possible soft applications over temporal data such as monitoring customer behaviors in stores and analyzing human and vehicle mobility in cities. As seen, timed patterns are a part of many safety-critical and not-so-critical systems and applications and we believe timed pattern matching would be an important tool when reasoning about them.

Following the line of thought of previous paragraphs, the first formalism we propose for timed patterns is the formalism of *timed regular expressions* (TRE) [10, 11] that extends regular expressions with duration constraints and intersection. Similar to Kleene's original theorem, the equivalence can be shown between timed regular expressions and timed automata. On the other hand, the lack of a nice algebraic characterization for timed automata has convinced us to make our pattern matching procedures free of timed automata and their clocks. Instead we favor the operation of duration restriction, which is a specialized form of intersection, rather than explicit clock variables. Therefore time periods are crucial in our system as they implicitly express duration without any need of an additional timekeeping mechanism. As a result, together with other composition operations, notably concatenation, we obtain a very nicely behaving algebraic system as we call the algebra of timed relations. Secondly, we introduce *metric compass logic* (MCL), a metric extension of compass logic with time bounds in a similar way that MTL extends LTL. Although there are some works extending some fragments of compass logic with timing, we consider full compass logic. For that, we extend the algebra of timed relations with quantitative versions of modalities. Using such an algebra under the hood, we represent quantitative temporal information and perform timed pattern matching for both timed regular expressions, metric compass logic, and even a free mix of their operators.

The introduction of *derivatives of timed regular expressions* is the other major contribution of this thesis. In the classical theory, derivatives of regular expressions, introduced by Brzozowski in [26], constitute a very elegant technique to evaluate a regular expression with respect to a word, thus simulate underlying finite-state automaton. We then lift the notion of derivative to timed languages without disrupting the simplicity and elegance of the technique. The use of timed derivatives to match timed patterns online is an immediate application as the ability to react as soon as a pattern is detected is an important feature.

Besides theoretical contributions mentioned above, we implemented all the concepts, ideas, and algorithms to perform timed pattern matching online and offline over timed behaviors of systems and the environment. We provide the tool MONTRE that contains these implementations with an easy-to-use interface and open it for the general public. Finally we demonstrate and evaluate our tool with examples and case studies from diverse application areas of timed pattern matching.

Contributions. We summarize the contributions of this thesis as follows:

- We introduce the algebra of timed relations, which simplifies the timed theory and implementations.
- We propose timed regular expressions as a concise, intuitive, and highly-expressive timed pattern specification language.
- We introduce metric compass logic, and propose it as another timed pattern specification language that can be used standalone or together with timed regular expressions.
- We provide efficient offline matching algorithms for patterns expressed in timed regular expressions and metric compass logic.
- We introduce derivatives of timed regular expressions and provide an online timed pattern matching algorithm based on derivatives.
- We implement timed pattern matching algorithms and package them as the command-line tool MONTRE. We present some examples and case studies.

Outline. The thesis is organized as follows. Chapter 2 gives the necessary background in classical automata and language theory without quantitative time as well as in temporal logics. Chapter 3 presents the algebra of timed relations and its computational aspects. Chapter 4 defines timed pattern matching, presents syntax and semantics of timed regular expressions and metric compass logic, and provides an offline pattern matching algorithm. Chapter 5 is devoted for timed derivatives and online computation. The tool MONTRE and case studies are described in Chapter 6 before concluding in Chapter 7.

2 PRELIMINARIES

Surely a moment's reflection,
and a single instance from common life,
must convince every one that
our social system is based upon Regularity.

— Flatland, Edwin A. Abbott

This chapter gives a brief background in classical formal languages, automata theory, and temporal logics. We start by providing the general terminology on the topic. Then we review parts of the theory of regular languages, regular expressions, and automata related to the thesis. In particular, we are mostly interested in Kleene's theorem [60] as well as Brzozowski's derivatives of regular expressions [26]. Thanks to many great subsequent work, the relation between languages, expressions, and automata is now very well-known as they represent semantic, syntactic, and operational facets of sequential properties. For a much broader survey on the topic, readers may consult monographs [54, 101, 102].

By temporal logic, we refer to the modal-logic approach introduced by Prior under the name of tense logic [91, 92]. It allows reasoning about temporal situations (states, events, processes) using (temporal) operators in a formal framework. The approach and formalism imported into the verification community by Pnueli to specify temporal behaviors of programs in [89]. Since then, it is considerably developed, extended, and diversified by many logicians and computer scientists. Here we will only review temporal logics over linear time flows related to this thesis and we suggest [47] for a broader survey and references therein.

2.1 WORDS AND LANGUAGES

In this section, we briefly present standard definitions and notations of letters, words, languages, and operations over them in computer science. Shared terminology with linguistics shows the original close connection between two disciplines and the subsequent success of text processing using formal languages and automata nailed these terms in every textbook. These connotations are nice as long as they are not taken too literally. In particular, we emphasize that a letter just means a basic building block that we're interested in their sequences. This can be truly a letter from the Latin alphabet but also a labeled time period, a symbolic expression, a function, etc.. Then let's start by words, which are finite sequences of letters.

Words. An alphabet is a set of letters. A word over an alphabet Σ is a finite (possibly empty) sequence of letters of Σ and the empty word is denoted by ϵ. The length $|w|$ of the word w is the number of letters in sequence and $|\epsilon| = 0$. The set of all words (resp. all nonempty words) over an alphabet Σ is denoted by Σ^* (resp. Σ^+). Given two words $u = a_1, \dots, a_m$ and $v = b_1, \dots, b_n$ over Σ, their concatenation $u \cdot v$ is given as $u \cdot v = a_1, \dots, a_m, b_1, \dots, b_n$. Concatenation is an associative operation as $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ for all $x, y, z \in \Sigma^*$. Also we often denote concatenation by juxtaposition. The empty word ϵ is the identitiy element of concatenation as ϵ satisfies the condition $w = \epsilon \cdot w = w \cdot \epsilon$ for all $w \in \Sigma^*$. For some $x, z \in \Sigma^*$, a word y is a prefix of a word w if $y \cdot z = w$, a suffix if $x \cdot y = w$, and a factor if $x \cdot y \cdot z = w$.

Languages. A language L over an alphabet Σ is a subset of Σ^*, that is, any finite or infinite set of words over Σ. On languages, set-theoretic operations of union $L_1 \cup L_2$, intersection $L_1 \cap L_2$, and complementation $(\overline{L})$ are defined in the usual way.

$$\begin{aligned} \overline{L} &= \{w \in \Sigma^* \mid w \notin L\} \\ L_1 \cup L_2 &= \{w \mid w \in L_1 \text{ or } w \in L_2\} \\ L_1 \cap L_2 &= \{w \mid w \in L_1 \text{ and } w \in L_2\} \end{aligned}$$

The concatenation $L_1 \cdot L_2$ of two languages L_1 and L_2 is naturally extended as follows.

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}.$$

Furthermore, the k-th power L^k of a language L is defined recursively such that $L^k = L^{k-1} \cdot L$ where $L^0 = \{\epsilon\}$. The union of all nonnegative powers (Kleene star) and the union of all positive powers (Kleene plus) of a language L are respectively denoted by

$$L^* = \bigcup_{k \geqslant 0} L^k \quad \text{and} \quad L^+ = \bigcup_{k \geqslant 1} L^k.$$

Another important operation on languages that we are interested in is the left quotient of a language L_1 by L_2. Intuitively speaking, the left quotient is an operation to obtain all words such that their concatenation with some words in L_2 would produce a word in L_1. The other way round, it is an operation to delete prefixes of words in L_1 if the prefix in L_2 and get the word that remains after, which may be empty as well. More precisely, the left quotient $L_2 \backslash L_1$ of a languages L_1 by L_2 is defined as

$$L_2 \backslash L_1 = \{v \mid \exists u \in L_2.\ uv \in L_1\}$$

A specialization of the (left) quotient operation by a word u —in the strict sense, by a singleton language $\{u\}$— is called the (left) derivative operation. The (left) derivative $D_u(L)$ of a language L with respect to u is defined as

$$\{u\} \backslash L = D_u(L) = \{v \mid uv \in L\}$$

It is often the case that the derivative operation is given with respect to single letters since a derivative operation with respect to a word $w = a_1 \dots a_n$ can be decomposed into successive derivations by single letters of w such that $D_{a_1 \dots a_n}(L) = D_{a_n} \dots D_{a_1}(L)$.

2.2 REGULAR EXPRESSIONS AND AUTOMATA

Given an alphabet Σ, the languages $\varnothing$, $\{\epsilon\}$ and $\{a\}$, where $a \in \Sigma$, are called basic languages. Then a language $L \subseteq \Sigma^*$ is said to be regular if it can be obtained from basic languages by applying finitely many times the operations of union, concatenation, and Kleene star as first introduced in [60]. These three (regular) operations form an algebra on languages called the regular algebra whose properties have extensively studied in [32]. Subsequently it is shown that the class of regular languages is also closed under intersection and complementation, sometimes called extended regular operations.

Kleene showed that the class of regular languages is precisely the class of languages recognized by finite automata, which is usually known as Kleene's theorem, that makes regular languages an extremely important class of formal languages. Other early characterizations of regular languages are given by Copi et al. in [33] from an operational perspective, by Nerode [82] from an algebraic perspective, and by Büchi in [28] from a logical perspective. In the context of this thesis, we do not delve into such algebraic and logical aspects much and mostly employ a language-theoretic perspective.

Then, we start with the definition of regular expressions, which are syntactic representations of regular languages. The syntax of regular expressions over Σ is given by the following grammar:

$$\varphi := \varnothing \mid \epsilon \mid a \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cdot \varphi_2 \mid \varphi^*$$

where $a \in \Sigma$. A regular expression φ specifies a regular language $[\![\varphi]\!]$, inductively defined as follows.

$$\begin{array}{rcl@{\qquad}rcl}
[\![\varnothing]\!] & = & \varnothing & [\![\varphi_1 \cup \varphi_2]\!] & = & [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
[\![\epsilon]\!] & = & \{\epsilon\} & [\![\varphi_1 \cdot \varphi_2]\!] & = & [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \\
[\![a]\!] & = & \{a\} & [\![\varphi^*]\!] & = & [\![\varphi]\!]^*
\end{array}$$

Then two regular expressions are said to be equivalent if they represent the same language. Moreover, when there is no ambiguity, we use languages and expressions interchangeably and usually use some easy simplifications (e.g. properties of absorbing and identity elements) without mentioning explicitly.

We know that the membership test $w \in L$ of a word w in a language L is equivalent to $\epsilon \in D_w(L)$ by definition. Hence, the problem of testing whether a word w is contained in a language can be reduced to a computation for the derivative D_w and a check for the empty word containment. For this purpose, an empty word extraction function ν (also known as the nullability predicate or output function) is first defined as

$$\nu(\varphi) = \begin{cases} \epsilon & \text{if } \epsilon \in [\![\varphi]\!] \\ \varnothing & \text{otherwise} \end{cases}$$

The function ν, which extracts ϵ from φ if it is contained in $[\![\varphi]\!]$, can be computed inductively for regular expressions by the following rules:

$$\begin{aligned} \nu(\varnothing) &= \varnothing & \nu(\varphi_1 \cdot \varphi_2) &= \nu(\varphi_1) \cap \nu(\varphi_2) \\ \nu(\epsilon) &= \epsilon & \nu(\varphi_1 \cup \varphi_2) &= \nu(\varphi_1) \cup \nu(\varphi_2) \\ \nu(a) &= \varnothing & \nu(\varphi^*) &= \epsilon \end{aligned}$$

In his seminal paper [26], Brzozowski applied the notion of derivatives to regular expressions and proved that the derivative $D_a(\varphi)$ of an expression φ with respect to a letter a can be computed recursively using the following syntactic rewrite rules:

$$\begin{aligned} D_a(\varnothing) &= \varnothing & D_a(\varphi_1 \cup \varphi_2) &= D_a(\varphi_1) \cup D_a(\varphi_2) \\ D_a(\epsilon) &= \varnothing & D_a(\varphi_1 \cdot \varphi_2) &= D_a(\varphi_1) \cdot \varphi_2 \;\cup\; \nu(\varphi_1) \cdot D_a(\varphi_2) \\ D_a(a) &= \epsilon & D_a(\varphi^*) &= D_a(\varphi) \cdot \varphi^* \\ D_a(b) &= \varnothing \end{aligned}$$

These rules are extended for words by letting $D_{a \cdot w}(r) = D_w(D_a(r))$. Hence to check, for example, whether abc is in the language of the expression $\varphi = a^* \cdot (b \cdot c)^*$ we compute $D_{abc}(\varphi) = D_c(D_b(D_a(\varphi)))) = (b \cdot c)^*$ as follows.

$$a^* \cdot (b \cdot c)^* \xrightarrow[D_a]{} a^* \cdot (b \cdot c)^* \xrightarrow[D_b]{} c \cdot (b \cdot c)^* \xrightarrow[D_c]{} (b \cdot c)^*$$

and then we know $abc \in [\![\varphi]\!]$ since $\nu((b \cdot c)^*) = \epsilon$. It is of course not a coincidence that this procedure resembles the reading of the word by an automaton where derivatives correspond to states and those that contain ϵ correspond to accepting states. Let us look at automata more closely.

In the most general sense, an automaton is an abstract computing device or machine that operates according to a set of predefined rules and external input symbols (letters). A state of the automaton is the representation of a particular situation in which the automaton can be at a specific time. An automaton processes a sequence of letters and moves from one state to another at each step. An automaton is called finite if it has a finite number of states and deterministic if there exists one and only one state to which the automaton can move from its current state upon processing a specific letter.

We denote the fact that a deterministic finite automaton (DFA) moves from the current state q upon processing the next letter a to the next state q' by $q \xrightarrow[a]{} q'$. Alternatively, it can be said that the letter a transforms the state q into q' and it is denoted by $\delta_a(q) = q'$ and, in this case, δ is called the transition function of the automaton. Some states of the automaton can be designated as initial and final states. Then, a sequence of letters, a word, that drive the automaton from the initial state to a final state is said to be successful or accepted by the automaton. The language $L \subseteq \Sigma^*$ recognized by an automaton $\mathcal{A}$ is the set of all accepted words are denoted by $[\![\mathcal{A}]\!]$. More precisely, a DFA $\mathcal{A}$ is defined as a 5-tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$ where Σ is a finite

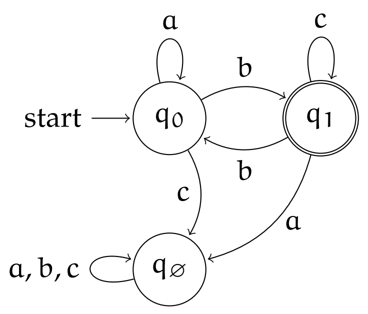


Figure 1: An example deterministic finite automaton over $\Sigma = \{a, b, c\}$.

alphabet, Q is a finite set of states, $\delta : \Sigma \times Q \to Q$ is the (deterministic) transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. It is often convenient to depict a DFA as a directed graph where edges are labeled with letters of the alphabet. For example, we depict an automaton $(\{a, b, c\}, \{q_0, q_1, q_\varnothing\}, \delta, q_0, \{q_1\})$ in Figure 1 where the state $q_\varnothing$ is usually called the dead state.

Now we give a very brief review of language equations for DFAS. For a more detailed treatment, we suggest the monograph [67] that also includes generalizations of automata — nondeterministic finite automata (NFA) [93] and boolean (alternating) finite automata [27]. Since we can declare every state in the automaton as an initial state, every state $q \in Q$ is associated with a unique language L_q, that is, the set of all words leading to the acceptance from the state q. In particular, the language L_q contains the empty word ϵ if the state q is a final state. Moreover, the language L_q contains $a \cdot L_{q'}$ where $L_{q'}$ is the language of the next state q' by the letter a such that $\delta_a(q) = q'$. This is simply because the word $a \cdot w$ leads to the acceptance from the state q if w does so from the next state q' by a.

Suppose that the states Q of a deterministic finite automaton $\mathcal{A}$ is given as $Q = \{q_0, q_1, \ldots, q_n\}$ and q_0 is the initial state, we can represent the automaton as a system $L_{q_0}, L_{q_1}, \ldots, L_{q_n}$ of $n + 1$ language equations obtained for each state. More precisely, the system of language equations associated with the automaton $\mathcal{A}$ is given as follows.

$$L_q = \bigcup_{q \xrightarrow{a} q'} a \cdot L_{q'} \cup \nu(q) \qquad \text{for all } q \in Q$$

where ν is the empty word extraction function previously defined. For example, consider the automaton depicted in Figure 1 and we then write the corresponding system of equations:

$$\begin{aligned} L_{q_0} &= a \cdot L_{q_0} \cup b \cdot L_{q_1} \\ L_{q_1} &= b \cdot L_{q_0} \cup c \cdot L_{q_1} \cup \epsilon \\ L_{q_\varnothing} &= \varnothing \end{aligned} \tag{1}$$

A system of equations can be solved for any state by using standard elimination techniques and, in particular, the language of the automaton is obtained by solving the system for the initial state q_0. Recursive

Table 1: Transition function of the automaton.

δ	a	b	c
φ_0	φ_0	φ_1	$\varnothing$
φ_1	$\varnothing$	φ_0	φ_1
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

language equations can be solved by using Arden's lemma [9], which states that

$$L_q = A \cdot L_q \cup B = A^* \cdot B$$

where A and B are arbitrary languages and the solution is unique if $\epsilon \notin A$. For example, we obtain the language $L_{q_0} = (a \cup bc^*b)^*bc^*$ by solving for q_0 the system (1) of equations above. Similarly we obtain the language $L_{q_1} = c^*(\epsilon \cup b(a \cup bc^*b)^*bc^*)$. Solving the system of equations shows that the language of every DFA can be obtained in such a way from basic languages by applying finitely many times the operations of union, concatenation, and Kleene star. Therefore, the language of a DFA is regular. This result gives one direction of Kleene's theorem from automata to regular languages. For the other direction, it would suffice to show that we can construct a deterministic finite automaton from any regular expression using derivatives of regular expressions as reviewed in the following.

For every regular expression φ, a DFA that recognizes $[\![\varphi]\!]$ can be constructed by the following procedure. Since every derivative characterizes a state of the automaton, two regular expressions are equivalent if they represent the same language. Then we compute derivatives starting from the initial state for each letter in the alphabet and keep track of new states (distinct derivatives) and continue with computing the derivatives for new states. This process of computing the derivatives will terminate after a finite number of steps since the number of distinct derivatives is finite as proved in [26]. For example, consider a regular expression $\varphi_0 = (a \cup bc^*b)^*bc^*$ over a three-letter alphabet $\Sigma = \{a, b, c\}$ and the expression φ_0 to be our initial state.

$$\begin{array}{llll} \varphi_0: \; (a \cup bc^*b)^*bc^* & \xrightarrow[D_a]{} & (a \cup bc^*b)^*bc^* & (\varphi_0) \\ & \xrightarrow[D_b]{} & c^*b(a \cup bc^*b)^*bc^* \; \cup \; c^* & (\text{New state: } \varphi_1) \\ & \xrightarrow[D_c]{} & \varnothing & (\text{New state: } \varnothing) \end{array}$$

Since we discovered new states distinct from the previously known, we need to repeat the process for each of them until no new state found. Then we have

$$\begin{array}{llll} \varphi_1: \; c^*b(a \cup bc^*b)^*bc^* \; \cup \; c^* & \xrightarrow[D_a]{} & \varnothing & (\varnothing) \\ & \xrightarrow[D_b]{} & (a \cup bc^*b)^*bc^* & (\varphi_0) \\ & \xrightarrow[D_c]{} & c^*b(a \cup bc^*b)^*bc^* \; \cup \; c^* & (\varphi_1) \end{array}$$

and derivations of $\varnothing$ are simply $\varnothing$. Then we tabulate the transition function δ of the automaton in Table 1, which is equivalent to the automaton depicted in Figure 1. Finally, since detecting equivalence between regular expressions is computationally expensive, a simpler check can be employed by sacrificing the minimality of the automaton. It is also shown in [26] that the number of distinct derivatives is finite even when syntactic similarity (based on associativity, commutativity, and idempotence properties) is considered.

2.3 TEMPORAL LOGICS

In this section, we review temporal logics based on time points as well as time periods. By temporal logic, we mean the modal approach introduced by Prior in [91, 92] under the name of tense logic with two temporal modalities, namely *some time in the past* and *some time in the future*. These modalities are based on time points and easily seen that they implicitly refer to time points less than the current time point and time points greater than the current time point, respectively. An important extension for Prior's logic, the introduction of two binary modalities *since* and *until*, is proposed by Kamp in [59]. Later Pnueli imported the concept into the computer science and proposed the use of temporal logic to specify infinite-duration temporal properties [89]. Afterwards, temporal logic was quickly adopted by the community as a major specification formalism in modern verification technology.

Here we first give the syntax of semantics of Pnueli's linear-time temporal logic with past and future operators.

Linear-time Temporal Logic. Linear-time Temporal Logic (LTL) extends propositional logic with future temporal modalities such as NEXT ($\bigcirc$), EVENTUALLY ($\Diamond$), ALWAYS ($\Box$), and UNTIL ($\mathcal{U}$) as well as past temporal modalities PREVIOUSLY (●), ONCE ($\blacklozenge$), HISTORICALLY ($\blacksquare$), and SINCE ($\mathcal{S}$). These modalities have the following intuitive meanings over a logical formula φ.

$\bigcirc\varphi$	[● φ]	φ holds at the next [previous] time point.
$\Diamond\varphi$	[$\blacklozenge\varphi$]	φ holds for some future [past] time point.
$\Box\varphi$	[$\blacksquare\varphi$]	φ holds for all future [past] time points.
$\varphi_1\ \mathcal{U}\ \varphi_2$	[$\varphi_1\ \mathcal{S}\ \varphi_2$]	φ_1 holds for all future [past] time points until [since] φ_2 holds for some future [past] time point.

where we refer the past versions in brackets. In the following, we present the (minimal) syntax and (strict) semantics of LTL formulas over finite or infinite behavior [41, 46, 69, 73, 89].

Let $N = [0, n)$ be an interval of integer time line. Given a set $P = \{p_1, \ldots, p_m\}$ of atomic propositions, a (discrete-time) behavior $w : N \to \mathbb{B}^m$ is a function that assigns a Boolean value $\{\text{FALSE}, \text{TRUE}\}$

for each proposition $p \in P$ at each time step. The syntax of LTL with future and past modalities is given over P by the following grammar.

$$\varphi \;=\; p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{S}\, \varphi_2$$

where $p \in P$. Then the satisfaction relation $\vdash$ of an LTL formula φ over a (finite or infinite) discrete-time behavior w is inductively defined as follows.

$$\begin{array}{lll}
(w,i) & \vdash p & \leftrightarrow \; w_p(i) = \text{TRUE} \\
(w,i) & \vdash \neg\varphi & \leftrightarrow \; (w,i) \nvdash \varphi \\
(w,i) & \vdash \varphi_1 \vee \varphi_2 & \leftrightarrow \; (w,i) \vdash \varphi_1 \text{ or } (w,i) \vdash \varphi_2 \\
(w,i) & \vdash \varphi_1 \,\mathcal{U}\, \varphi_2 & \leftrightarrow \; \exists j \in (i,|w|).(w,j) \vdash \varphi_2 \text{ and} \\
 & & \qquad \forall k \in (i,j).(w,k) \vdash \varphi_1 \\
(w,i) & \vdash \varphi_1 \,\mathcal{S}\, \varphi_2 & \leftrightarrow \; \exists j \in [0,i).(w,j) \vdash \varphi_2 \text{ and} \\
 & & \qquad \forall k \in (j,i).(w,k) \vdash \varphi_1
\end{array}$$

The other operators are derived by letting

$$\begin{array}{rclrcl}
\bigcirc\varphi & = & \text{FALSE}\,\mathcal{U}\,\varphi & \bullet\varphi & = & \text{FALSE}\,\mathcal{S}\,\varphi \\
\Diamond\varphi & = & \text{TRUE}\,\mathcal{U}\,\varphi & \blacklozenge\varphi & = & \text{TRUE}\,\mathcal{S}\,\varphi \\
\Box\varphi & = & \neg\Diamond\neg\varphi & \blacksquare\varphi & = & \neg\blacklozenge\neg\varphi
\end{array}$$

Metric Temporal Logic. Metric temporal logic (MTL) extends LTL with the notion of dense-time and timing constraints on the temporal modalities. Let T be an interval of dense real time line. Given a set $P = \{p_1, \ldots, p_m\}$ of atomic propositions, a dense-time behavior $w : T \to \mathbb{B}^m$ is a function that assigns a Boolean value $\{\text{FALSE}, \text{TRUE}\}$ for each proposition $p \in P$ at each time point. The syntax of MTL with future and past LTL modalities augmented with temporal bounds is given over a set P of atomic propositions by the following grammar.

$$\varphi \;=\; p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}_I\, \varphi_2 \mid \varphi_1 \,\mathcal{S}_I\, \varphi_2$$

where $p \in P$ and I is an interval of duration values. Then the satisfaction relation $\vdash$ of an MTL formula φ over a (finite or infinite) dense-time behavior $w : T \to \mathbb{B}^m$ is inductively defined as follows.

$$\begin{array}{lll}
(w,t) & \vdash p & \leftrightarrow \; w_p(t) = \text{TRUE} \\
(w,t) & \vdash \neg\varphi & \leftrightarrow \; (w,t) \nvdash \varphi \\
(w,t) & \vdash \varphi_1 \vee \varphi_2 & \leftrightarrow \; (w,t) \vdash \varphi_1 \text{ or } (w,t) \vdash \varphi_2 \\
(w,t) & \vdash \varphi_1 \,\mathcal{U}_I\, \varphi_2 & \leftrightarrow \; \exists t' \in T.\ (w,t') \vdash \varphi_2, \\
 & & \qquad \forall t'' \in (t,t').(w,t'') \vdash \varphi_1, \text{ and} \\
 & & \qquad t' - t \in I \\
(w,t) & \vdash \varphi_1 \,\mathcal{S}_I\, \varphi_2 & \leftrightarrow \; \exists t' \in T.\ (w,t') \vdash \varphi_2, \\
 & & \qquad \forall t'' \in (t',t).(w,t'') \vdash \varphi_1, \text{ and} \\
 & & \qquad t - t' \in I
\end{array}$$

The modalities $\Diamond_I$,$\Box_I$, $\blacklozenge_I$, and $\blacksquare_I$ can be derived in the same way as in LTL. This is not the case for NEXT and PREVIOUSLY, which have no

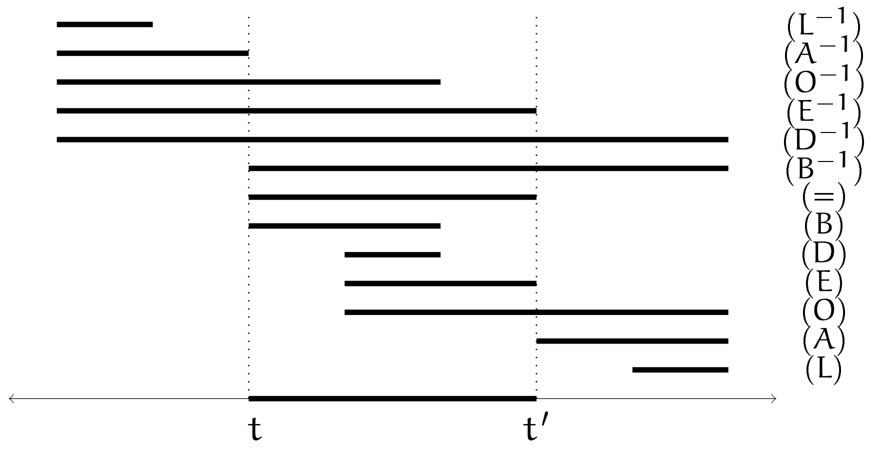


Figure 2: Allen's relations between time periods.

meaning under dense time. Observe that their derivation formulas FALSE $\mathcal{U}_I$ φ and FALSE $\mathcal{S}_I$ φ evaluate to FALSE for all formulas φ and bounds I due to the density.

Logics of time periods. Point-based temporal logics are not naturally suitable to express local patterns with finite durations. For these specifications, temporal logics based on time periods are argued to be a better formalism. Then, considering time periods as primitive entities, Allen introduced 13 basic relations between two time periods to represent high-level temporal knowledge in [2]. The set of so-called Allen's relations consists of relations *met-by* (A), *begins* (B), *ends* (E), *during* (D), *overlaps* (O), and *later* (L) as well as their inverses and the equality (=). In Figure 2 we illustrate these relations in a way that a depicted time period (horizontal lines) and the time period (t, t') is in the specified relation given at the right. In [48], Halpern and Shoham applied Prior's modal approach over time periods and proposed a temporal logic that features a modality for each Allen's relation.

It is shown that six certain temporal modalities of the HS logic can express others under strict semantics. Here we use the compass notation introduced by Venema in [112] (with our slight extension) since it has nice geometric connotations on the two-dimensional plane. The basic set consists of six modalities (diamonds) denoted by $\Diamond, \Diamond, \Diamond, \Diamond, \Diamond, \Diamond$, respectively corresponding to relations $A, A^{-1}, B, B^{-1}, E, E^{-1}$ between time periods. These (diamond) modalities have the following intuitive meanings over a formula φ over time periods.

$\Diamond$	φ	φ holds at a period met by the current one.
$\Diamond$	φ	φ holds at a period that meets the current one.
$\Diamond$	φ	φ holds at a period that begins the current one.
$\Diamond$	φ	φ holds at a period begun by the current one.
$\Diamond$	φ	φ holds at a period that ends the current one.
$\Diamond$	φ	φ holds at a period ended by the current one.

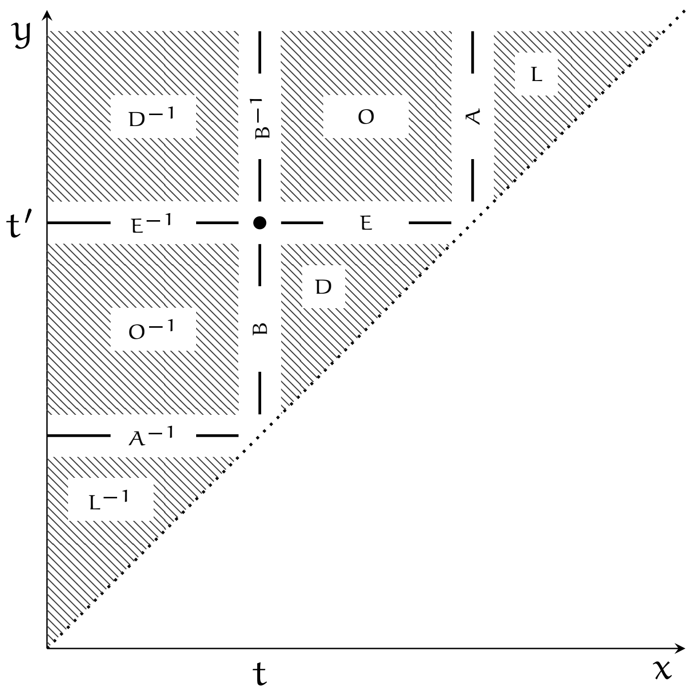


Figure 3: Accessibility regions for each Allen relation with respect to a time period (t, t').

As shown, using these basic set of modalities, we can derive more modalities for the remaining relations. For example, $\Diamond\varphi \equiv \Diamond\Diamond\varphi$ corresponds to the relation L, $\Diamond\varphi \equiv \Diamond\Diamond\varphi \equiv \Diamond\Diamond\varphi$ to the relation D, and $\Diamond\varphi \equiv \Diamond\Diamond\varphi$ to the relation O. Moreover, the dual (box) modalities are defined as usual such that $\Box\varphi = \overline{\Diamond\overline{\varphi}}$ where $\Diamond \in \{\Diamond, \Diamond, \Diamond, \Diamond, \Diamond, \Diamond\}$ and $\Box$ matches the decoration. From another point of view, each modality accesses a different region on the two-dimensional plane with respect to the current period and quantifies φ over the accessed region. In Figure 3, we illustrate accessed regions for each Allen's relation with respect to a period (t, t').

Finally, we mention the chop modality, which corresponds to the concatenation operator of regular expressions and can be added on the top of temporal logics [30, 79, 97, 113]. It is known that the expressiveness of HS logic can be increased further by the addition of the chop modality [113].

2.4 MONITORING AND PATTERN MATCHING

In this section, we overview regular expression matching and temporal logic monitoring. It is worth to note in the beginning that the precise meaning of the terms monitoring and matching is very fluid in formal methods literature. Historically, the term monitoring is used to check the satisfaction of a temporal behavior against point-based temporal logics. In general, however, we use the term monitoring to denote any act of observing and evaluating individual temporal behaviors of systems. Pattern matching is then a monitoring task defined to be an act of identifying the segments of observed behaviors that satisfy a pattern.

Given a pattern and a word, we can then ask several questions in the context of pattern matching: (1) Does the whole word match the pattern? (acceptance), (2) Does a prefix of word match the pattern? (prefix matching), (3) Does a suffix of the word match the pattern? (suffix matching), and (4) Which factors of the word match the pattern? (two-sided matching). For example, consider a word $w = \mathtt{abcab}$ and a (string) pattern $\varphi = \mathtt{ab}$. Then we give the following answers: (1) No, the whole word w does not match φ, (2) Yes, there is a prefix ending at the position 2, (3) Yes, there is a suffix beginning at the position 4, and (4) These are segments from the position 1 to 2 and from 4 to 5. Most of the work in pattern matching literature can be viewed as techniques to answer some or all of these questions for some formalism. Strings and regular expressions are the most commonly used formalisms for pattern specifications. Although string patterns are not our focus in this thesis, we note that string matching algorithms [35, 81] can be useful in optimizing regular expression matching [116].

The pioneering work on regular expression matching was due to Thompson [107] who constructed a non-deterministic finite automaton, for a given regular expression, to be executed with respect to the input word. This idea implemented in the archetypal tool GREP, which is still actively used program in UNIX systems to search patterns over texts. Since searching and manipulating texts is such a ubiquitous task, some programming languages such as AWK and PERL has been designed to perform regular expression matching. Besides, virtually any programming language provides a functionality to match regular expressions via their standard libraries.

On the other hand, we do not see such a rich picture of algorithms and implementations for regular expressions extended with intersection and complementation. It is known that the complexity of standard automata-based solutions significantly increases in the presence of these operations. Alternative approaches proposed so far are based on dynamic programming [54, 61], derivatives [100], and some variants of automata [55, 64, 98, 117].

Temporal logic monitoring over individual behaviors has been initially proposed as a lightweight approach for program and hardware verification under the name of runtime verification. Complexities of model checking and the practical need to check individual behaviors using formal specifications motivated developments in the field. We suggest [111] for a historical review on the rationale and development of temporal logics for monitoring purposes, and [74] for a complexity comparison between model checking and monitoring of several temporal logics.

For LTL monitoring, a variety of algorithms has been proposed based on formula rewriting systems [52, 99], sequential circuits [44, 53, 90], and finite automata [45]. Algorithms for MTL monitoring usually follows LTL monitoring. Among them we distinguish ones based on dynamic programming [72] and formula rewriting [16, 106]. We also note the existence of an interesting branch of monitoring over data traces using formal specifications [5, 15, 38, 42, 50], which essentially replace the underlying Boolean algebra of classical formalisms with various algebras (of some data domains). For a broader survey which contains other monitoring approaches such as (domain-specific) programming languages, please refer to [49].

3 ALGEBRA OF TIMED RELATIONS

Ne içindeyim zamanın, ne de büsbütün dışında.
Yekpare, geniş bir anın parçalanmaz akışında.

I am neither inside time, nor am I completely outside.
In the indivisible flow of a moment, atomic and wide.

— Poems, Ahmet Hamdi Tanpinar

This chapter introduces the algebra of timed relations. We define *timed relations* to be finitely representable subsets of the set of all time periods. We are interested in various algebraic (boolean, sequential, and temporal) operations on timed relations represented by symbolic and geometric means.

The study of the algebra of logic and relations began with Boole and DeMorgan in the nineteenth century and continued by Peirce and Schröder [103]. Geometric representations of relations can be traced back to Tarski's calculus of relations [105]. Other related classical works include Blake's canonical expressions in boolean algebras [21], Stone's representation theorem [104], and boolean algebras with operators [57]. The latter is closely related to modal (temporal) logics [19, 20], thus it completes a cycle of mathematical results.

We focus on concrete geometric representations specific to timed relations in this chapter. We propose algorithms to compute boolean, sequential, and temporal operations over such representations based on techniques from computational geometry. In short, we establish a computational framework here for the following chapters, which relies on classical results of the algebra of logic and relations.

3.1 DEFINITIONS AND NORMAL FORMS

We start by defining the underlying time domain we will use throughout the thesis, which is a set of time points on a dense, linear, and bounded time line.

Definition 3.1 (Time domain). *A (dense, linear, bounded) time domain* T *is a dense interval of time points with rational bounds admitting a strict linear order* $<$.

We consider $T = (0, d)$ to be an interval of real or rational numbers denoting time points on the time axis. Although we have just mentioned time points, we now abandon them in favor of time periods defined as follows.

Definition 3.2 (Time period). *A time period* (t, t') *is a pair of begin and end boundaries on a time domain* T *such that* $t < t'$ *with a non-zero duration of* $t' - t$. *We denote the set of all time periods over* T *by* $\Omega(T)$.

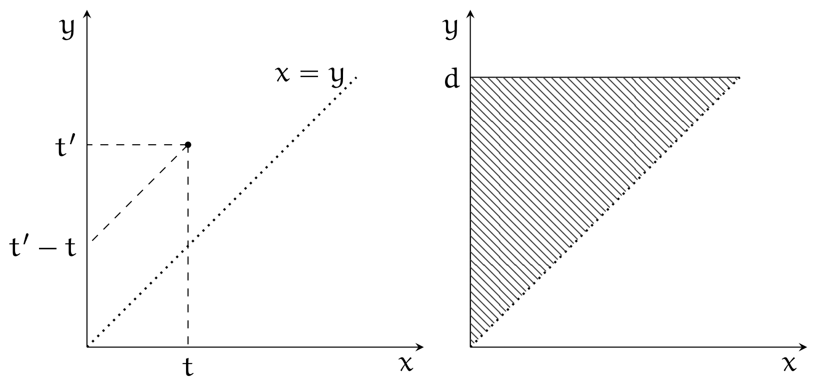


Figure 4: Geometric representations of a time period (t, t') and the set $\Omega(T)$.

We say that a time period (t_1, t_1') meets another time period (t_2, t_2') if the end of the first equals to the beginning of the second such that $t_1' = t_2$. A sequence $S = (t_0, t_1), (t_1, t_2), \dots, (t_{n-1}, t_n)$ of meeting time periods is simply called a sequence of time periods. We say that the sequence S begins at t_0, ends at t_n, and has a duration of $t_n - t_0$ and a length of n.

Geometrically speaking, a time period (t, t') can be viewed as a single point on the standard two-dimensional xy-plane. On the left of Figure 4, we illustrate attributes of a time period (t, t'), its beginning t, its end t', and its duration $t' - t$. Then we are interested in a very specific set of linear inequalities that correspond to constraints on beginnings, endings, and durations of time periods, often called vertical, horizontal, and diagonal half-planes. In particular, the set of all time periods $\Omega(T)$ can be seen as a (triangular) set of points $\{(t, t') \mid t \geqslant 0,\ t' \leqslant d,\ t' - t > 0\}$ on the plane as depicted on the right of Figure 4.

In the following, we closely relate sets of time periods to Boolean functions [24, 34]. Boolean (set-theoretic) operations of union ($\cup$), intersection ($\cap$), and complementation ($\overline{\ }$) as well as the inclusion relation ($\subseteq$) over sets of time periods are defined as usual with the empty set $\varnothing$ and the universal set $\Omega(T)$. We call a set of time periods a *timed relation* if and only if it can be expressed as a finite boolean formula over vertical, horizontal, and diagonal half-planes.

Definition 3.3 (Timed Relation)**.** *A timed relation $Z \subseteq \Omega(T)$ is a (possibly uncountable) set of time periods that can be represented on the xy-plane by a boolean combination of finitely many half-planes having one of six forms (1) $x \prec c$, (2) $y \prec c$, (3) $y - x \prec c$, (4) $c \prec y - x$, (5) $c \prec y$, and (6) $c \prec x$ with rational constants where $\prec \in \{<, \leqslant\}$.*

Observe that the empty set, $\Omega(T)$, and all finite sets of time periods are timed relations. A timed relation that can be formed only by intersections is called a convex timed relation. We denote by $\mathcal{Z}$ and $\mathcal{Z}^\cap$ the set of all timed relations and the set of all convex timed relations over $\Omega(T)$, respectively. We differentiate the six types of half-planes in the definition by annotating superscripted numbers $(1-6)$ such as h^1. The complement of an open [closed] half-plane h^k is a closed [open] half-plane h^{7-k} with the same constant c. Intersections or unions of any number of half-planes $h^k_1, \dots, h^k_n$ of the same type would be implied by one of the half-planes h^k_i, $i \in 1, \dots, n$. Therefore, every convex timed relation $z \in \mathcal{Z}^\cap$ can be formed by an intersection of six half-planes $h^1, h^2, h^3, h^4, h^5, h^6$ of each type such that

$$\bigcap_{k=1\dots 6} h^k = \{(x,y) \mid c_6 \prec x \prec c_1 \ \wedge \ c_5 \prec y \prec c_2 \ \wedge \ c_4 \prec y - x \prec c_3\}$$

where $c_1, \dots, c_6 \in \mathbb{R} \cup \{-\infty, +\infty\}$. Consequently, we can represent a convex timed relation as a six-tuple $(h^1, h^2, h^3, h^4, h^5, h^6)$ of half-planes. Notice that the inequalities defining these half-planes are not totally independent of each other and the arithmetic addition of other two certain inequalities may imply a tighter constraint than the one in the representation. Every non-empty convex timed relation $z \in \mathcal{Z}^\cap$ has a unique normal representation such that all constraints are tight. More precisely, given a representation $(h^1, h^2, h^3, h^4, h^5, h^6)$ of z that the tight representation of z can be computed as follows:

$$\begin{aligned}\textsc{tighten}(z) = \Big(&h^1 \cap (h^2 + h^4),\\ &h^2 \cap (h^1 + h^3),\\ &h^3 \cap (h^2 + h^6),\\ &h^4 \cap (h^1 + h^5),\\ &h^5 \cap (h^4 + h^6),\\ &h^6 \cap (h^3 + h^5)\Big)\end{aligned}$$

where $+$ denotes arithmetic addition of inequalities. In Figure 5, we illustrate the most general (hexagon) case for a convex timed relation where one can see, for example, that two half-planes $h^1 : x < c_1$ and $h^3 : y - x \leqslant c_3$ imply a constraint $h^1 + h^3 : y < c_1 + c_3$, which can imply or be implied by h^2. Importantly, an inclusion test between two convex timed relations z_1 and z_2 can be performed over their tight representations such that

$$z_1 \subseteq z_2 \ \longleftrightarrow \ \bigwedge_{i=1,\dots,6} h^i_1 \subseteq h^i_2$$

From now on, we consider all representations of convex timed relations to be tightened according to the definition above and we use the term *zone* both for a convex timed relation and its tight representation. We say that a zone z_1 is implied by another zone z_2 if $z_1 \subseteq z_2$. Note that zones (possibly in higher dimensions) are commonly employed for timed systems research and admit efficient data structures called difference bound matrices (DBMs) [22, 37]. Here we use many two-dimensional zones to represent time periods rather than one or a few high-dimensional zones to represent clock valuations.

We represent a timed relation as a finite union of zones similar to the disjunctive normal form of Boolean functions. By definition and by DeMorgan's laws, every timed relation Z can be represented by a union over a finite set of non-empty zones $R_Z = \{z_1, z_2, \dots, z_n\}$ such

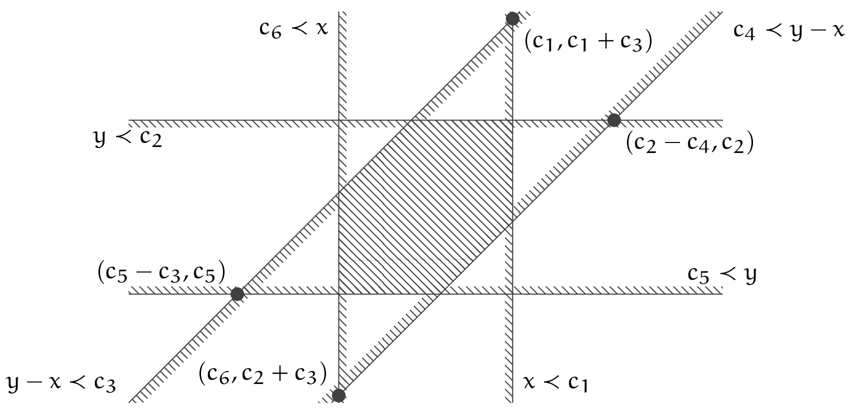


Figure 5: Dependencies between constraints of a convex timed relation.

that $Z = z_1 \cup z_2 \cup \cdots \cup z_n$. Colloquially, we say *timed relation* when we want to emphasize semantic aspects whereas *union of zones* to emphasize syntactic aspects. Now we define some important properties of representations for timed relations.

Definition 3.4 (Absorption)**.** *A union of zones R_Z is absorptive if and only if no zone in R_Z is implied by any other zone in R_Z.*

For any union of zones R_Z, we can obtain an equivalent absorptive union of zones, denoted by ABSORB(R_Z), by removing all absorbed zones from the representation. Obviously, there may be different representations of a timed relation Z but an absorptive representation ABSORB(R_Z) for a given R_Z is unique. Unless specified otherwise, we consider all unions of zones to be absorptive throughout the thesis. We then adapt Blake's syllogistic theory of Boolean functions [21, 24] to the case of timed relations in a direct fashion in the following.

Definition 3.5 (Syllogism)**.** *A union of zones R_Z is syllogistic if and only if every zone $z \subseteq Z$ is included in some zone in R_Z.*

Let us now define a syntactic inclusion test between two unions of zones R_{Z_1} and R_{Z_2} as follows:

$$R_{Z_1} \sqsubseteq R_{Z_2} \longleftrightarrow \forall z_1 \in R_{Z_1}.\ \exists z_2 \in R_{Z_2}.\ z_1 \subseteq z_2$$

An important result for syllogistic sets of zones is that syntactic inclusion is implied by semantic inclusion. That is to say, an inclusion test $Z_1 \subseteq Z_2$ between two timed relations Z_1 and Z_2 can be replaced by a syntactic inclusion test between their representations if R_{Z_2} is syllogistic.

Lemma 3.1. *Let R_{Z_1} and R_{Z_2} be two unions of zones. If R_{Z_2} is syllogistic, we have the equivalence $Z_1 \subseteq Z_2 \longleftrightarrow R_{Z_1} \sqsubseteq R_{Z_2}$.*

Proof. Assume R_{Z_2} is syllogistic. One direction ($\leftarrow$) of the equivalence is trivial. In the other direction, assume a zone z of R_{Z_1} is not included in any zone of R_{Z_2} and then it implies $z \nsubseteq Z_2$ by definition. Hence we have ($\rightarrow$) by contraposition. □

It is also easily seen that ABSORB(R_Z) is syllogistic if and only if R_Z is syllogistic. Below we denote by $R_{Z_1} \cap R_{Z_2}$ a union of zones developed by intersecting zones from both sets using the distributive laws.

Lemma 3.2. *Let $R_{Z_1}, \ldots, R_{Z_n}$ be syllogistic unions of zones. Then we have that $R_{Z_1} \cap \cdots \cap R_{Z_n}$ is syllogistic.*

Proof. Let z be a zone of $R_{Z_1} \cap \cdots \cap R_{Z_n}$. Then $z \subseteq Z_i$ for $i = 1 \ldots n$. Since R_{Z_i} are syllogistic, there exists a zone $z'_i \in R_{Z_i}$ such that $z \subseteq z'_i$ and then we have $z \subseteq z'_1 \cap \cdots \cap z'_n$. See that $z'_1 \cap \cdots \cap z'_n$ is a zone of $R_{Z_1} \cap \cdots \cap R_{Z_n}$, hence $R_{Z_1} \cap \cdots \cap R_{Z_n}$ is syllogistic. □

Next we define a canonical representation for timed relations, which is the analogue of Blake's canonical form for Boolean functions. The maximal normal form of a timed relation Z is defined to be a union of all maximal zones as follows.

Definition 3.6 (Maximal Zone). *Let Z be a timed relation. We say that a zone $z \subseteq Z$ is a maximal zone of Z if there is no other zone z' that satisfies $z \subset z' \subseteq Z$.*

Definition 3.7 (Maximal Normal Form). *A union of zones R_Z is in the maximal normal form if it is the union of all maximal zones of Z.*

Lemma 3.3. *A union of zones R_Z is in the maximal normal form of Z if and only if it is absorptive and syllogistic.*

Proof. The maximal normal form of Z is absorptive and syllogistic by definition. For the other direction, suppose R_Z is syllogistic and absorptive. Let z be a maximal zone of Z. Then z has to be in R_z since there is no zone z' such that $z \subset z' \subseteq Z$. Since R_Z is absorptive, it does not contain any zone z'' such that $z'' \subset z$. Hence R_Z contains all maximal zones, and no other zones. □

3.2 OPERATIONS ON TIMED RELATIONS

When describing operations on timed relations, we first give the semantic definition. Then we demonstrate the operation over single zones and extend it towards unions of zones. We overload operators for both semantic and syntactic operations if there is no ambiguity.

Duration restriction. We consider that the operation of duration restriction is a first-class operation for the algebra of timed relations. A timed relation Z whose elements restricted to have a duration value within an interval $[a, b]$ of non-negative reals is denoted by $\langle Z \rangle_{[a,b]}$ and defined as follows.

$$\langle Z \rangle_{[a,b]} = \{(x, y) \mid (x, y) \in Z \text{ and } y - x \in [a, b]\}$$

Over a zone, we directly apply duration restriction by letting

$$\langle z \rangle_{[a,b]} = \text{TIGHTEN}(h^1, h^2, h^3 \cap (y - x \leqslant b), h^4 \cap (a \leqslant y - x), h^5, h^6)$$

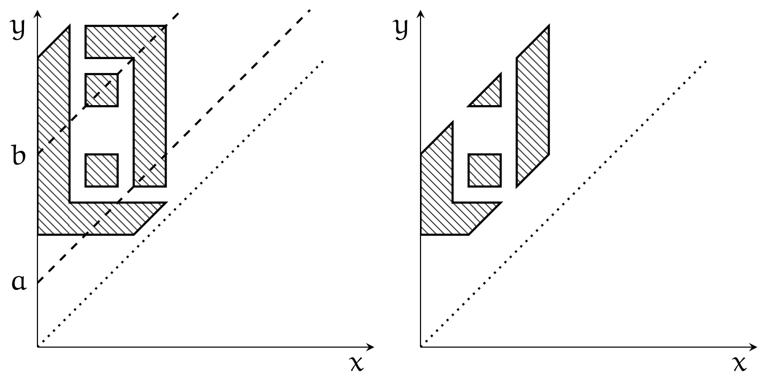


Figure 6: Duration restriction.

and extend it towards unions of zones as

$$\langle R_Z \rangle_{[a,b]} = \{\langle z \rangle_{[a,b]} \mid z \in R_Z\}$$

In Figure 6, we depict a timed relation Z on the left and $\langle Z \rangle_{[a,b]}$ on the right, which can be viewed as a diagonal slice of Z on the plane.

Intersection. The operation of intersection between timed relations is defined as usual.

$$Z_1 \cap Z_2 = \{(x, y) \mid (x, y) \in Z_1 \text{ and } (x, y) \in Z_2\}$$

Observe that duration restriction is a special case of intersection with a (constant) timed relation $C = \langle \Omega(T) \rangle_{[a,b]}$ such that $\langle Z \rangle_{[a,b]} = Z \cap C$. We intersect two zones by letting

$$z_1 \cap z_2 = \text{TIGHTEN}(h_1^1 \cap h_2^1, h_1^2 \cap h_2^2, h_1^3 \cap h_2^3, h_1^4 \cap h_2^4, h_1^5 \cap h_2^5, h_1^6 \cap h_2^6)$$

and extend it towards union of zones as

$$R_{Z_1} \cap R_{Z_2} = \{z_1 \cap z_2 \mid z_1 \in R_{Z_1} \text{ and } z_2 \in R_{Z_2}\}$$

which indicates a quadratic complexity for the worst case.

Complementation. The complement $\overline{Z}$ of a timed relation Z with respect to the universal set $\Omega(T)$ is given as follows:

$$\overline{Z} = \{(x, y) \in \Omega(T) \mid (x, y) \notin Z\}$$

See that the complement of a timed relation Z is unique and the double complement of Z is equivalent to Z. The statements $Z \cap \overline{Z} = \emptyset$ and $Z \cup \overline{Z} = \Omega(T)$ hold for every timed relation Z. Since both DeMorgan laws hold for timed relations, we complement a zone z and a union of zones R_Z, respectively as follows.

$$\overline{R_z} = \{\overline{h_z^1}, \overline{h_z^2}, \overline{h_z^3}, \overline{h_z^4}, \overline{h_z^5}, \overline{h_z^6}\} \quad (2) \qquad \overline{R_Z} = \bigcap_{z \in R_Z} \overline{R_z} \quad (3)$$

Observe that the complement of a single zone as in Equation 2 is a syllogistic union of zones. Then, by Lemma 3.2, we also have that the complement of a timed relation as in Equation 3 are syllogistic. Moreover we can obtain the maximal normal form of $\overline{Z}$ by removing absorbed zones from $\overline{R_Z}$. It follows that double complementation can be used to obtain the maximal normal form of any timed relation.

Finally we note that Equation 3 can be developed into a union of zones in an incremental and more efficient manner such that

$$\overline{R^i_Z} = \text{ABSORB}\left(\overline{R^{i-1}_Z} \cap \overline{R_{z_i}}\right)$$

where $\overline{R^i_Z}$ is the complement of the subset $\{z_1, z_2, \ldots, z_i\}$ of a union of zones with n elements for $1 \leqslant i \leqslant n$ and $\overline{R^0_Z} = \Omega(T)$.

Union. Given two timed relations Z_1 and Z_2, the union of two timed relations is defined as usual.

$$Z_1 \cup Z_2 = \{(x,y) \mid (x,y) \in Z_1 \text{ or } (x,y) \in Z_2\}$$

The class of zones (convex timed relations) is not closed under union and the operation $R_{Z_1} \cup R_{Z_2}$ simply corresponds to that the union of member zones of both representations. We note, however, timed relations of $R_{Z_1} \cup R_{Z_2}$ and $\overline{\overline{R_{Z_1}} \cap \overline{R_{Z_2}}}$ are equivalent but they may have different representations. The latter would produce the maximal normal form of $Z_1 \cup Z_2$ whereas the former does not necessarily.

Composition. Given two timed relations Z_1 and Z_2, their (sequential) composition, equivalently their concatenation, is defined by

$$Z_1 \circ Z_2 = \{(x,y) \mid \exists r.\ (x,r) \in Z_1 \text{ and } (r,y) \in Z_2\}$$

In Figure 7 we illustrate the composition of two singleton timed relations $\{(t,t'')\} \circ \{(t'',t')\} = \{(t,t')\}$ on the left side. Notice that a time period (t_1, t'_1) can be sequentially composed (concatenated) with another time period (t_2, t'_2) only if the end of the first meets the beginning of the second such that $t'_1 = t_2$. Otherwise the composition $\{(t_1,t'_1)\} \circ \{(t_2,t'_2)\} = \varnothing$ for $t'_1 \neq t_2$. Next we have that the class of zones (convex timed relations) is closed under composition.

Proposition 3.4. *The composition of two zones is a zone.*

Proof. Following the semantics of composition we have

$$(t,t') \in z_1 \circ z_2 \text{ iff } \exists t''.\ t < t'' < t',\ (t,t'') \in z_1, \text{ and } (t'',t') \in z_2$$

which translates to $\exists t'' \in (t,t')$ such that

$$\left\{\begin{array}{ccccc} \underline{b}_1 & \prec & t & \prec & \overline{b}_1 \\ \underline{e}_1 & \prec & t'' & \prec & \overline{e}_1 \\ \underline{d}_1 & \prec & t''-t & \prec & \overline{d}_1 \end{array}\right\} \text{ and } \left\{\begin{array}{ccccc} \underline{b}_2 & \prec & t'' & \prec & \overline{b}_2 \\ \underline{e}_2 & \prec & t' & \prec & \overline{e}_2 \\ \underline{d}_2 & \prec & t'-t'' & \prec & \overline{d}_2 \end{array}\right\}$$

By eliminating the quantifier, we obtain that $z_1 \circ z_2$ equals to a zone

$$\left\{\begin{array}{ccccc} \max(\underline{b}_1,\ \underline{b}_2 - \overline{d}_1) & \prec & t & \prec & \min(\overline{b}_1,\ \overline{b}_2 - \underline{d}_1) \\ \max(\underline{e}_2,\ \underline{e}_1 + \underline{d}_2) & \prec & t' & \prec & \min(\overline{e}_2,\ \overline{e}_1 + \overline{d}_2) \\ \underline{d}_1 + \underline{d}_2 & \prec & t'-t & \prec & \overline{d}_1 + \overline{d}_2 \end{array}\right\}$$

if $\underline{e}_1 \prec \overline{b}_2$ and $\underline{b}_2 \prec \overline{e}_1$, the empty set otherwise. □

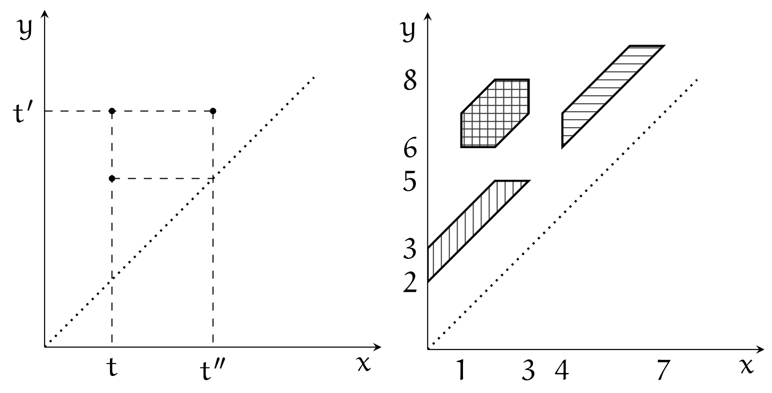


Figure 7: Composition of timed relations.

Following the constructive proof above, we have an algorithm to compute concatenation between two zones and extend it towards unions of zones as

$$R_{Z_1} \circ R_{Z_2} = \{z_1 \circ z_2 \mid z_1 \in R_{Z_1} \text{ and } z_2 \in R_{Z_2}\}$$

In Figure 7, we illustrate the composition operation

$$z_1 \circ z_2 = \{(x,y) \mid 1 \leqslant x \leqslant 3 \ \cap \ 6 \leqslant y \leqslant 8 \ \cap \ 4 \leqslant y - x \leqslant 6\}$$

of two zones $z_1 = \{(x,y) \mid 0 \leqslant x \leqslant 3 \ \cap \ 2 \leqslant y \leqslant 5 \ \cap \ 2 \leqslant y - x \leqslant 3\}$ and $z_2 = \{(x,y) \mid 4 \leqslant x \leqslant 7 \ \cap \ 6 \leqslant y \leqslant 9 \ \cap \ 2 \leqslant y - x \leqslant 3\}$ on the right side.

Transitive Closure. The i-th power of a timed relation for $i \geqslant 1$ and transitive closure Z^+ are defined, respectively as follows.

$$Z^i = \underbrace{Z \circ Z \circ \cdots \circ Z}_{i \text{ times}} \qquad\qquad Z^+ = \bigcup_{i \geqslant 1} Z^i$$

Now we show that the transitive closure of any timed relation is representable by a finite number of composition operation, thus timed relations are closed under transitive closure operation.

Proposition 3.5. *Over a bounded time domain* $T = (0, d)$, *for all* $Z \in \mathcal{Z}$, *there exists an integer* k *that satisfies the equality* $Z^+ = \bigcup_{1 \leqslant i \leqslant k} Z^i$.

Proof. Let H be a finite partition of $\Omega(T)$ with a form of right-triangular grid generated by integer multiples ($t_i = ic$ for $i = 0, \ldots, m = d/c$) of the greatest common divisor c and constant parts of inequalities defining Z and $\Omega(T)$. Each cell in H defines a set of equivalent time segments with respect to Z^+ membership such that $\forall (x,y),(x',y') \in C.\ (x,y) \in Z^+ \leftrightarrow (x',y') \in Z^+$. In particular we use the equivalence in cells $t_i < x < y < t_{i+1}$ to show the original claim in the following.

Assume a segment $(r_0, r_k) \in Z^{k+1}$ and then there exists a sequence of time segments $(r_0, r_1), (r_1, r_2), \ldots (r_k, r_{k+1})$ such that each segment $(r_i, r_{i+1}) \in Z$. When $k > 3m - 2$, by the pigeonhole principle, there are two consecutive segments, denoted by $(r_{i-1}, r_i), (r_i, r_{i+1})$, within the same cell $t_i < x < y < t_{i+1}$. By the equivalence inside a cell, the sequence $(r_{i-1}, r_i), (r_i, r_{i+1})$ can be replaced by (r_{i-1}, r_{i+1}) thus $(r_0, r_k) \in Z^k$, and we conclude that the test $Z^{k+1} \subseteq Z^k$ holds and we have that $Z^+ = \bigcup_{1 \leqslant i \leqslant k} Z^i$. $\square$

Temporal modalities. An important extension for the algebra of timed relations is a set of temporal modalities based on Allen's relations [2] on time periods and introduced by Halpern and Shoham in [48]. Here we use the compass notation introduced by Venema in [112] with slight modifications since it has nice geometric connotations on the two-dimensional plane. The basic set consists of six existential modalities (diamonds) denoted by $\Diamond \in \{\diamondN, \diamondS, \diamondW, \diamondE, \diamondWbar, \diamondEbar\}$. We talk in more detail about the meaning of these modalities in Chapter 4.3. Now we introduce metric compass operators $\Diamond_I$ on timed relations obtained by constraining the range of quantification in an amount specified by an interval I as follows:

$$
\begin{array}{lcl}
\diamondS_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ x < r < y,\ y - r \in I, \text{ and } (x, r) \in Z\} \\
\diamondN_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ x < y < r,\ r - y \in I, \text{ and } (x, r) \in Z\} \\
\diamondE_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ x < r < y,\ r - x \in I, \text{ and } (r, y) \in Z\} \\
\diamondW_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ r < x < y,\ x - r \in I, \text{ and } (r, y) \in Z\} \\
\diamondEbar_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ x < y < r,\ r - y \in I, \text{ and } (y, r) \in Z\} \\
\diamondWbar_I\, Z & = & \{(x, y) \in \Omega(T) \mid \exists r.\ r < x < y,\ x - r \in I, \text{ and } (r, x) \in Z\}
\end{array}
$$

In Figure 8, we illustrate each diamond operator over a timed relation containing a single time period. Intuitively speaking, a diamond operator shifts a time period in the specific direction on the plane by an allowed amount. This operation, called back-shifting (of time points) [72, 83], is used to evaluate the timed eventually modality of metric temporal logic. Notice that the shift of time points can be viewed as a degenerate case of that of timed periods and there are surely more directions to move in two dimensions. Next we note that metric compass operators possess two important algebraic properties of normality and additivity such that

$$
\begin{aligned}
\Diamond_I\, \varnothing &= \varnothing \\
\Diamond_I (Z_1 \cup Z_2) &= \Diamond_I\, Z_1 \cup \Diamond_I\, Z_2
\end{aligned}
$$

Therefore, the algebra of timed relations with compass operators forms a boolean algebra with operators in the sense of [57] that provides an alternative (algebraic) semantics for various modal logics including temporal logics. The close connection between modal logics and boolean algebras with operators is extensively studied in several monographs [19, 20, 75].

In the following, we show the class of zones is also closed under metric compass operators. Consequently, we have that applying a metric compass operation on a zone results in another zone whose bounds are shifted according to the direction given by the type $\Diamond$ and the metric constraint I of the compass operation $\Diamond_I$.

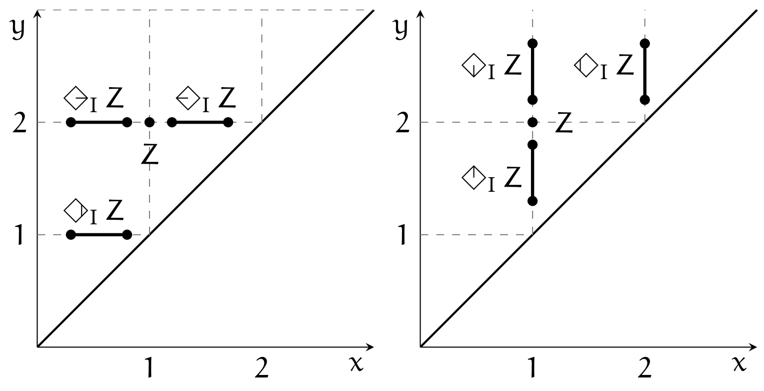


Figure 8: Metric compass operations $\Diamond_I$ on a timed relation $Z = \{(1,2)\}$ for a temporal constraint $I = [0.2, 0, 7]$.

Proposition 3.6. *Given a zone z, the timed relation $\Diamond_I z$ is a zone.*

Proof. We only show the case of $\Diamond_{[m,n]}$ as other cases are symmetric and we assume $T = (0, d)$ without loss of generality. Following the semantics of $\Diamond_{[m,n]}$,

$$(x, y) \in \Diamond_{[m,n]} \text{ iff } \exists r \in (x, y).\ r - x \in [m, n], \text{ and } (r, y) \in z$$

which translates to $\exists r \in (x, y)$ such that

$$\left\{ \begin{array}{ccccc} \underline{b} & \prec & r & \prec & \overline{b} \\ \underline{e} & \prec & y & \prec & \overline{e} \\ \underline{d} & \prec & y - r & \prec & \overline{d} \end{array} \right\}$$

By eliminating the quantifier, we obtain that $\Diamond_{[m,n]}$ equals to a zone

$$\left\{ \begin{array}{ccccc} \underline{b} - n & \prec & x & \prec & \overline{b} - m \\ \underline{e} & \prec & y & \prec & \overline{e} \\ \underline{d} + m & \prec & y - x & \prec & \overline{d} + n \end{array} \right\}$$

□

Following the proof above, we apply metric compass operations over zones directly and extend it towards unions of zones as

$$\Diamond_I R_Z = \{\Diamond_I z \mid z \in R_Z\}$$

Finally, we illustrate the application of metric compass operations in Figure 9. Consider the zone $z = \{(x, y) \mid 3 \leqslant x \leqslant 5 \ \cap\ 5 \leqslant y \leqslant 7 \ \cap\ 3 \leqslant y - x \leqslant 5\}$ on the left of the figure. Then we have $\Diamond_{[1,2]} z = \{(x, y) \mid 1 \leqslant x \leqslant 4 \ \cap\ 5 \leqslant y \leqslant 7 \ \cap\ 4 \leqslant y - x \leqslant 7\}$ on the right, which is obtained by shifting z to the left accordingly. Equivalently, this operation can be viewed as a Minkowski sum $z \oplus S_{\mathtt{left}}$ of the zone z and a left-shifting set $S_{\mathtt{left}} = \{(-t, 0) \mid t \in [1, 2]\}$ with respect to $\Omega(T)$.

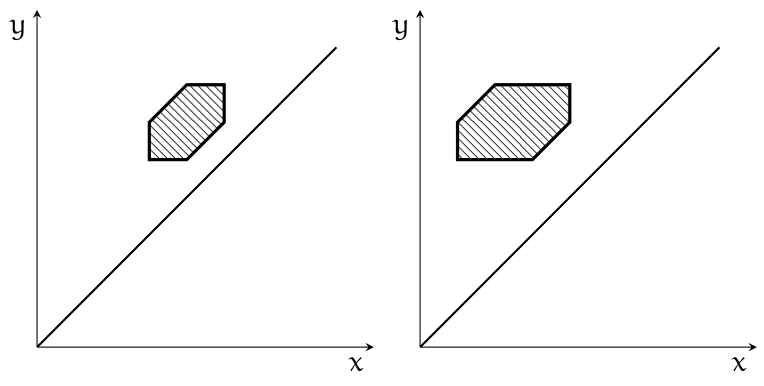


Figure 9: A zone z (left) and its left-shift $\diamondsuit_{[1,2]}\, z$ (right).

3.3 ALGORITHMS AND COMPLEXITY

In this section, we present our algorithms to compute operations on timed relations represented as unions of zones. Unary operations of duration restriction and temporal modalities as well as union operation are implemented in a straightforward way followed by an absorption operation. Binary operations of intersection, concatenation, and absorption (as it is an operation on two copies of the same set) on unions of zones with n elements requires $\mathcal{O}(n^2)$ in the worst case. Operations of transitive closure and complementation are more expensive; the worst case complexity of transitive closure can be bounded from above by $\mathcal{O}(n^3)$ and that of complementation by $\mathcal{O}(n^4)$ using simple arguments. Although tighter bounds can be shown for these operations, our focus here is to develop algorithms that exploit intrinsic relations and achieve linear or quasi-linear (time and space) complexity in practice. The key assumption we make is that time periods of interest and zones representing them would be dispersed on the time axis and their (maximum) durations are much smaller than the entire time domain. Therefore, most operations between two zones would be redundant since they are far away from each other (e.g. their intersection/concatenation are trivially empty) and can be avoided by sorting zones and applying the operations to zones that are temporally closer to each other according to the sort order. Since we already view time as a space and timed relations as geometric objects, we propose the use of efficient computational geometry techniques to perform operations between unions of zones in the following. In particular, we employ the plane sweep technique [85] since it performed better in our initial tests than other spatial join techniques surveyed in [56] such as R-tree-based spatial queries.

Plane Sweep Technique. Given two sets of geometric objects in an Euclidean space, the plane sweep is one of spatial join techniques that finds all pairs of objects satisfying a given relation between their spatial components. The rough idea behind the technique is to move a virtual line across the plane, keep track of objects that are in relation with this sweeping line, and perform an spatial operation on these

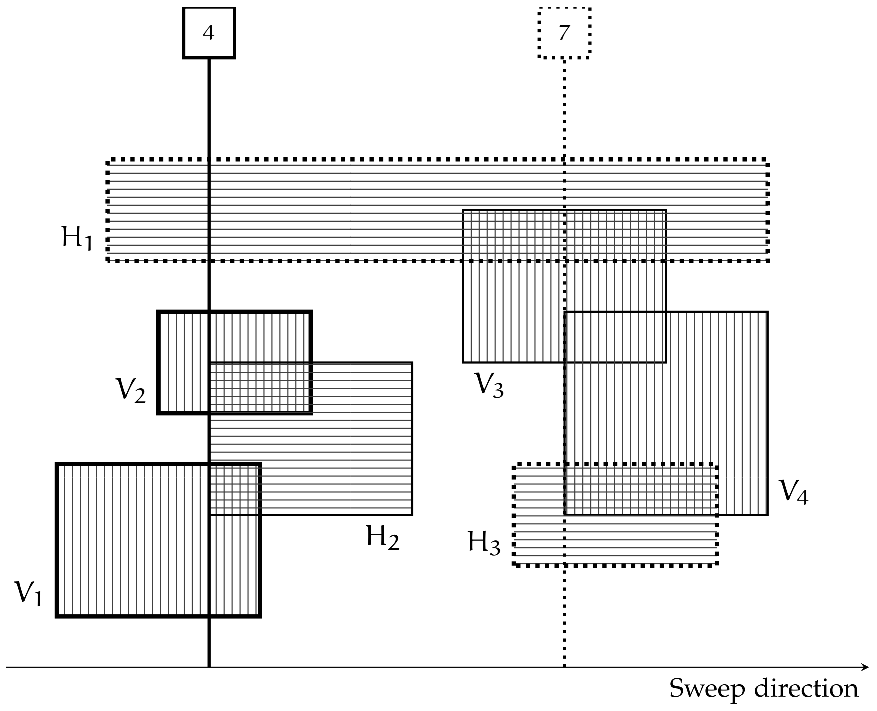


Figure 10: Illustration of plane sweep technique.

objects. More precisely, assuming that objects are sorted according to the sweep direction, a plane sweep algorithm sweeps the virtual line through the sorted list from left to right, pauses at each object, performs the actual operation between the current object and other objects in relation (usually intersects) with the sweep line.

We illustrate the technique for the intersection of rectangles and consider two sets of rectangles, $H = \{H_1, H_2, H_3\}$ and $V = \{V_1, V_2, V_3, V_4\}$, of horizontal and vertical stripes in Figure 10. The plane sweep algorithm first sorts all the rectangle according to their leftmost points so that it processes the list $(V_1, H_1, V_2, H_2, V_3, H_3, V_4)$ in order and moves a line from the leftmost point of one rectangle to that of the next at each step. Rectangles from the other set that intersect the line are considered active and we know that these are the only rectangle that may intersect with the current rectangle and processed earlier. Therefore the algorithm computes an intersection operation between these rectangles and the current rectangle and proceeds. For example, at the fourth step, the current rectangle is H_2 and the sweep line (depicted as a solid line) intersects rectangles V_1 and V_2 from the other set, and the algorithm computes $H_2 \cap V_1$ and $H_2 \cap V_2$, which are non-empty sets. Similarly the algorithm only computes $V_4 \cap H_1$ and $V_4 \cap H_3$ at the seventh and final step where the sweep line is depicted as dotted. □

Now we explain our algorithms for operations of absorption, intersection, and concatenation that use the plane sweep technique. For the absorption operation, Algorithm 1 keeps an initially empty set A of active zones and selects the next zone z from the union of zones of R sorted by h^6. Thus we sweep the plane starting from the zone with

Algorithm 1 ABSORB(R) *assume* R *sorted by* h^6

```
$A := R' := \varnothing$
foreach $z \in R$ and not $\bigcup_{z' \in A} z \subseteq z'$ loop
  $A := A \backslash \{z' \in A \mid z' \subseteq z\} \cup \{z\}$
  foreach $z' \in A$ loop
    Move $z'$ from $A$ to $R'$ if $h^6_z \subset h^6_{z'}$
  end loop
end loop
return $R' \cup A$
```

the earliest begin point. The algorithm skips to the next zone if z is included in some zone in A, removes zones in A that are included in z, and adds Z to A otherwise. Then it moves zones in the active set A to the output set R' if their earliest begin value is less than the earliest begin value of z (thus the sweep line) since upcoming zones cannot include those. Once every zone in the original set has been processed, the algorithm yields the remaining zones in A as output zones in addition to previously yielded zones.

For the intersection operation, Algorithm 2 similarly sorts R and R' by h^6 and keeps two active sets of zones A and A' for zones in R and R', respectively. Zones are successively moved to their corresponding active lists and are removed from them when it is clear they will not participate in further non-empty intersections. Note that the function

Algorithm 2 INTERSECT(R, R') *assume* R, R' *sorted by* h^6

```
$A := A' := Y := \varnothing$
while $R \neq \varnothing$ or $R' \neq \varnothing$ do
  $z :=$ FIRST$(R)$
  $c :=$ CONST$(h^6_z)$
  $z' :=$ FIRST$(R')$
  $c' :=$ CONST$(h^6_{z'})$
  if $c < c'$ then
    Move $z$ from $R$ to $A$
    $A' := \{a' \in A' \mid$ CONST$(h^1_{a'}) \geqslant c\}$
    foreach $a' \in A'$ loop
      $z'' := z \cap a'$
      $Y := Y \cup \{z''\}$
    end loop
  else
    Move $z'$ from $R'$ to $A'$
    $A := \{a \in A \mid$ CONST$(h^1_a) \geqslant c'\}$
    foreach $a \in A$ loop
      $z'' := a \cap z'$
      $Y := Y \cup \{z''\}$
    end loop
  end if
end while
return ABSORB$(Y)$
```

Algorithm 3 CONCATENATE(R, R') *assume* R *sorted by* h^5 *assume* R' *sorted by* h^6

```
$A := A' := Y := \varnothing$
while $R \neq \varnothing$ or $R' \neq \varnothing$ do
  $z :=$ FIRST$(R)$
  $c :=$ CONST$(h^5_z)$
  $z' :=$ FIRST$(R')$
  $c' :=$ CONST$(h^6_{z'})$
  if $c < c'$ then
    Move $z$ from $R$ to $A$
    $A' := \{a' \in A' \mid$ CONST$(h^1_{a'}) \geqslant c\}$
    foreach $a' \in A'$ loop
      $z'' := z \circ a'$
      $Y := Y \cup \{z''\}$
    end loop
  else
    Move $z'$ from $R'$ to $A'$
    $A := \{a \in A \mid$ CONST$(h^2_a) \geqslant c'\}$
    foreach $a \in A$ loop
      $z'' := a \circ z'$
      $Y := Y \cup \{z''\}$
    end loop
  end if
end while
return ABSORB$(Y)$
```

FIRST(R) denotes the the first element of R and CONST(h) denotes the constant part of the inequality that defines the half-plane h.

For the concatenation operation, Algorithm 3 is very similar to the intersection except that the first set R is sorted by h^5 (that is, the earliest end) and the second set R' by h^6 (that is, the earliest begin). Therefore plane sweep technique finds all pairs of zones from two sets such that the end interval of the first one intersects with the begin interval of the second, which is a necessary condition for a non-empty concatenation.

Our complementation procedure presented in Algorithm 4 follows the definition given in the previous section. However, notice that we perform an absorption operation for each step while developing the complement. This reduces the number of zones yielded in the intermediate stages; and therefore, it is more efficient than multiplying out all complemented zones and then performing the absorption.

Algorithm 4 COMPLEMENT(R)

```
$Y := \Omega$
foreach $z \in R$ loop
  $Y :=$ ABSORB$(Y \cap$ COMPLEMENT$(z))$
end loop
return $Y$
```

Algorithm 5 CLOSURE(R)

```
$Y := R$
$X :=$ CONCATENATE$(R, R)$
while $X \nsubseteq Y$ do
  $Y :=$ ABSORB$(X \cup Y)$
  $X :=$ CONCATENATE$(X, R)$
end while
return $Y$
```

Algorithm 6 CLOSURE2(R)

```
$Y := R$
$X :=$ CONCATENATE$(R, R)$
while $X \nsubseteq Y$ do
  $Y :=$ ABSORB$(X \cup Y \cup$ CONCATENATE$(X, Y))$
  $X :=$ CONCATENATE$(X, X)$
end while
return $Y$
```

Finally, we present two algorithms for the transitive closure, one incremental and one based on so-called squaring, given in Algorithm 5 and Algorithm 6, respectively. Our tests did not show a clear winner between two approaches, however, it seems that the squaring approach performs better for sets whose fixed point index is large and the incremental approach is better when the index is small.

4 TIMED PATTERN MATCHING

You are so young.
So many years left,
so many languages to acquire!

— Kató Lomb

In this chapter, we introduce timed pattern matching and propose timed regular expressions (TRE) and metric compass logic (MCL) as pattern specification languages. These formalisms provide a strong theoretical background and can express many real-life patterns intuitively and concisely. We work on timed (symbolic) behaviors that are sequences of qualitative observations of systems and their environment over continuous time. Timed behaviors are also known as (discrete-valued, continuous-time) signals and observation sequences in the literature [6, 10, 95] and are argued to be the most natural model for timed formalisms. Timed regular expressions and metric compass logic formulas are built over atomic propositions that correspond to such observations using connectives defined in their respective syntaxes. Patterns specified using TRE and MCL denote sets of timed behaviors, which are instances of the pattern. For a given input behavior w, we say that a segment $w(t, t')$ of w from t to t' matches or satisfies a pattern if $w(t, t')$ is an instance thereof. Hence, the satisfaction of timed regular expressions and metric compass logic formulas is relative to an input behavior w and a time period (t, t'). We denote such satisfaction relations by the relational semantics as usual. Then we define the problem of timed pattern matching as finding and reporting all segments of an input behavior that satisfy a given timed pattern.

We solve timed pattern matching by evaluating a timed pattern φ over an input behavior w to a timed relation Z such that each time period $(t, t') \in Z$ stands for a matching segment $w(t, t')$. The evaluation is done in the standard way. We consider every atomic expression and logical formula to be evaluated to a timed relation and every connective to be treated as an operation on timed relations. Given an input behavior, this mechanism is captured by a valuation function. Using the valuation function and the algebra of timed relations, we define the algebraic semantics of TRE and MCL and show that the algebraic semantics agrees with the relational semantics.

Finally we consider some practical extensions of our timed pattern matching framework towards non-homogeneous atomic propositions, timed event sequences, and a richer timed pattern language obtained by a free mix of operators from TRE and MCL at end of the chapter.

4.1 DEFINITIONS

Let $P = \{p_1, \ldots, p_m\}$ be a finite set of (atomic) propositional variables over a time domain T that correspond to some qualitative states and activities of some real-time systems and the environment. Examples include the proposition *Alice is running* denoting an activity of Alice and *the temperature is higher than 23 degrees*, which denotes a qualitative state over a physical quantity. We consider such propositions are observed continuously for a finite amount of time and model the evolution of corresponding activities and states as timed behaviors. To this end, we first define an alphabet $\Sigma = \mathbb{B}^m$ of observations expressed as Boolean vectors of dimension m where $\mathbb{B} = \{0, 1\}$. Then a timed behavior is a finite sequence of time periods such that each period (t_{k-1}, t_k) is associated with a Boolean vector $a_k \in \Sigma$ such that $a_k(i) = 1$ if the proposition p_i holds on the time period (t_{k-1}, t_k) and $a_k(i) = 0$ if it does not.

Definition 4.1 (Timed Behavior). *A timed behavior w over a set P of atomic propositions on a time domain $T = (t_0, t_n)$ is a finite sequence such that*

$$w = (t_0, t_1, a_1), (t_1, t_2, a_2), \ldots, (t_{n-1}, t_n, a_n)$$

where $a_k \in \Sigma$ and $t_{k-1} < t_k$ for $k \in 1 \ldots n$. A timed behavior begins at t_0, ends at t_n, has a duration of $t_n - t_0$ and a length of n.

We use $w_p(t, t')$ to denote the restriction of w to an atomic proposition p and a time domain (t, t'). The concatenation $w_1 \cdot w_2$ is defined only if w_1 meets w_2, that is, w_1 ends at the point where w_2 begins. Alternatively, a timed behavior can be given as a sequence of pairs of duration values and symbols from the alphabet Σ such that

$$w = (t_1 - t_0, a_1), (t_2 - t_1, a_2), \ldots, (t_n - t_{n-1}, a_n)$$

In this notation, we assume the beginning time $t_0 = 0$ unless stated otherwise. As an example, we depict in Figure 11 an evolution of two atomic propositions $P = \{p_1, p_2\}$ over time where the alphabet is $\Sigma = \{(0,0), (0,1), (1,0), (1,1)\}$. Then a timed behavior w can represent this evolution such that

$$\begin{aligned} w = {} & (0, 2, (1,1)), (2, 4, (1,0)), (4, 6, (0,0)), (6, 7, (0,1)), (7, 8, (1,1)), \\ & (8, 9, (0,1)), (9, 10, (1,1)), (10, 11, (1,0)), (11, 12, (0,0)) \end{aligned}$$

or, using duration-symbol notation,

$$\begin{aligned} w = {} & (2, (1,1)), (2, (1,0)), (2, (0,0)), (1, (0,1)), (1, (1,1)), \\ & (1, (0,1)), (1, (1,1)), (1, (1,0)), (1, (0,0)) \end{aligned}$$

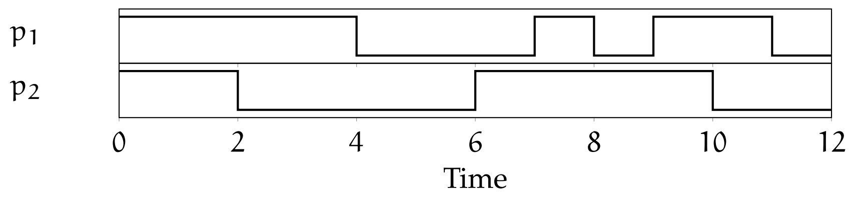


Figure 11: An evolution of atomic propositions p_1 and p_2.

Notice that such representations are not unique; any time period associated with a value can be divided into shorter periods of the same value and the resulting behaviors would be equivalent. Sometimes these successive periods of the same value are called *stuttering* periods [11, 65]. Stuttering periods are often avoidable and it is more efficient to work with the stutter-free behaviors obtained by merging them. For example, if we restrict w above to the proposition p_2, we can directly write w_{p_2} as

$$\begin{aligned} w_{p_2} = {} & (0,2,(1)),(2,4,(0)),(4,6,(0)),(6,7,(1)),(7,8,(1)),\\ & (8,9,(1)),(9,10,(1)),(10,11,(0)),(11,12,(0)) \end{aligned}$$

which, after elimination of stuttering, becomes

$$w_{p_2} = (0,2,(1)),(2,6,(0)),(6,10,(1)),(10,12,(0))$$

However, there are some applications like online monitoring where one does not know what will be observed in the next period but want to react immediately. For such cases, stuttering in behaviors may arise naturally. Therefore, we (have to) allow stuttering in our definitions and procedures. The correctness of our results does not depend on stutter-freeness. Next, notice that the length n of a timed behavior is minimal when the timed behavior is stutter-free and we call this minimal length the *variability* of the behavior. We use the notation $\Sigma^{(n)}$ to denote the set of all timed behaviors of variability n over an alphabet Σ. In particular, $\Sigma^{(1)}$ is the set of all uniform timed behaviors and $\Sigma^{(+)}$ the set of all timed behaviors. We call subsets of $\Sigma^{(+)}$ timed (behavior) languages in this thesis. We do not consider subsets of $\Sigma^{(*)}$ since the theory becomes asymmetric in the simultaneous presence of the empty word and the complement operation. This is related to the exclusion of singular time points in our theory for which we provide additional motivation in the following remark.

Remark. *We can equate timed behaviors to a (well-behaving) class of continuous time functions, however, we should emphasize time period nature of behaviors. A time period* (t, t') *with a value* a *can be interpreted as a constant continuous-time function that maps its domain* D *to the value* a *where* D *is either* (t, t'), $[t, t')$, $(t, t']$, *or* $[t, t']$. *One can restrict oneself to càdlàg (left-closed, right-open) or càglàd (left-open, right-closed) intervals and obtain a notion of continuity for timed behaviors in the traditional sense, however, this is not an essential consideration. Indeed we simply do not care about the value of singular points and the existence of gaps. Attempts to support such features would bring many complications and no practical benefit as seen in the history of timed systems research; therefore, we insist on leaving them out.*

Next we define our period-based temporal structures that we use to evaluate propositions over timed behaviors. Given a timed behavior w on a time domain $T = (t_0, t_n)$, we define a temporal structure $W = (\Omega(T), V)$ induced by w such that $\Omega(T)$ is the set of all time periods over T and V is a valuation function that assigns every proposition to a set of time periods on which it holds with respect to w. It is an

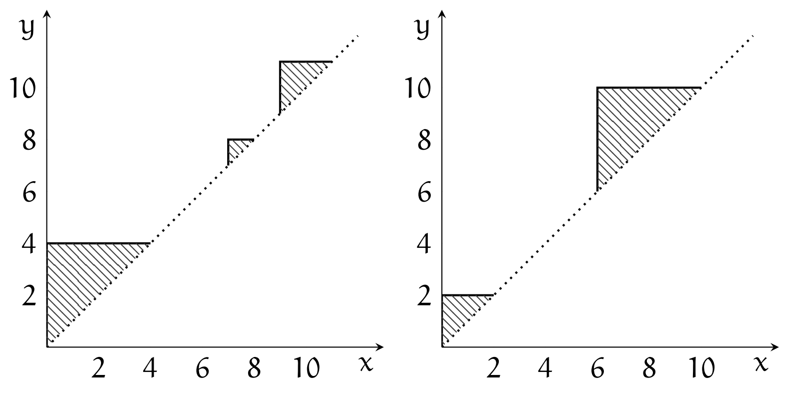


Figure 12: Valuations of atomic propositions.

important point that atomic propositions are homogeneous, that is to say, an atomic proposition $p \in P$ holds on a time period if and only if it holds on its all sub-periods, which is captured by the formula

$$(t, t') \in V(p) \longleftrightarrow \forall r, r'. \ (t < r < r' < t') \rightarrow (r, r') \in V(p) \quad \text{(HOM)}$$

Since a timed behavior w is a finite sequence, a valuation $V(p)$ of an atomic proposition p in W is a finite union of triangle zones that reside along the diagonal, thus a timed relation. We illustrate $V(p_1)$ and $V(p_2)$ in Figure 12 for the timed behavior of Figure 11. Notice that valuations of atomic propositions are closed under transitive closure such that $V(p) = V(p)^+$.

4.2 TIMED REGULAR EXPRESSIONS

In this section, we present the syntax and semantics of timed regular expressions over a set of propositional variables P. The variant presented here admits standard regular connectives of union, concatenation, and (one-or-more) iteration as well as intersection and duration restriction in the syntax given by the following grammar:

$$\varphi \ := \ p \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \varphi_1 \cdot \varphi_2 \mid \varphi^+ \mid \langle \varphi \rangle_I$$

where $p \in P$ and I is an interval of duration values. The duration restriction $\langle \varphi \rangle_I$ of an expression φ is the characteristic feature of TRE [10, 11]. We use an exponent notation with $\varphi^1 = \varphi$, $\varphi^k = \varphi^{k-1} \cdot \varphi$ for $k \geqslant 2$ and sometimes juxtaposition for concatenation. The Kleene plus connective φ^+ stands for an infinite sum $\bigcup_{k \geqslant 1} \varphi^k$ as usual. We do not include the Kleene star as a unary connective but it can be added as a binary one (as in Kleene's original paper) such that $\varphi_1 {}^* \varphi_2 = \varphi_1^+ \varphi_2 \cup \varphi_2$. Given a timed behavior w, the satisfaction of a timed regular expression at a time period (t, t') is defined inductively in the following.

Definition 4.2 (Relational Semantics). *The satisfaction relation* $\models$ *of a timed regular expression* φ *in a temporal structure* $W = (\Omega(T), V)$ *induced by a timed behavior* w *with a time domain* T*, relative to a time period* $(t, t') \in \Omega(T)$ *is defined as follows:*

$$\begin{array}{lll}
(W, t, t') & \models p & \leftrightarrow \quad (t, t') \in V(p) \\
(W, t, t') & \models \varphi_1 \cap \varphi_2 & \leftrightarrow \quad (W, t, t') \models \varphi_1 \textit{ and } (W, t, t') \models \varphi_2 \\
(W, t, t') & \models \varphi_1 \cdot \varphi_2 & \leftrightarrow \quad \exists t''. (W, t, t'') \models \varphi \textit{ and } (W, t'', t') \models \psi \\
(W, t, t') & \models \varphi_1 \cup \varphi_2 & \leftrightarrow \quad (W, t, t') \models \varphi_1 \textit{ or } (W, t, t') \models \varphi_2 \\
(W, t, t') & \models \varphi^+ & \leftrightarrow \quad \exists k \geqslant 1. (W, t, t') \models \varphi^k \\
(W, t, t') & \models \langle \varphi \rangle_I & \leftrightarrow \quad (W, t, t') \models \varphi \textit{ and } t' - t \in I
\end{array}$$

We illustrate some matches on an example timed behavior in Figure 13 such that the time period $(1, 4)$ satisfies an expression $p_1 \cap p_2$ since $(1, 4)$ satisfies an expression p_1 and p_2. Similarly, the time period $(6, 11)$ satisfies the expression $p_1 \cdot p_2$ since it is sufficient that $(6, 9)$ satisfies p_1 and $(9, 11)$ satisfies p_2.

Notice that atomic expressions in TRE are more symbolic than in classical regular expressions in the sense that an atomic expression $p_i \in P$ stands for all observations $a \in \Sigma$ such that $a(i) = 1$ and the duration of the observation period can be of any value. For comparison, atomic expressions of traditional regular expressions stand for single letters from the alphabet and have the length of 1. Such approach leads to more natural and concise specifications when propositions are concurrent; and consequently, commonly used in concurrent system specification languages.

Let us continue the comparison with a running example. Consider a basic pattern φ over a set of two propositions $P = \{p_1, p_2\}$ and the alphabet $\Sigma = \{(0,0), (0,1), (1,0), (1,1)\}$ such that a period where p_1 holds is followed by a period where p_2 holds. Assuming a base time unit for the classical case, we may express the pattern φ either as a classical regular expression φ_{RE} or a timed regular expression φ_{TRE} as given in the following.

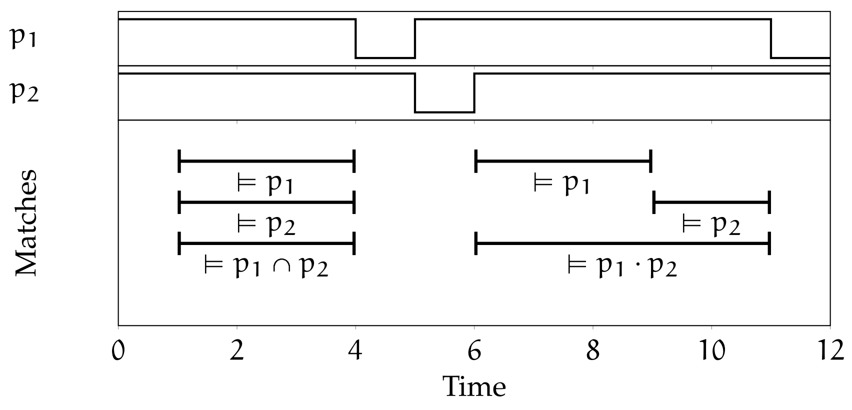


Figure 13: Some matches for timed regular expressions over atomic propositions p_1 and p_2.

$$\varphi_{\mathrm{RE}} = \big((1,0) \cup (1,1)\big)^{+} \cdot \big((0,1) \cup (1,1)\big)^{+} \qquad \varphi_{\mathrm{TRE}} = p_1 \cdot p_2$$

Moreover, a duration (or length) restriction by an interval I on the pattern φ, denoted by φ', could be expressed by letting

$$\varphi'_{\mathrm{RE}} = \varphi_{\mathrm{RE}} \ \cap \ \textstyle\bigcup_{k \in I} \Sigma^{k} \qquad \varphi'_{\mathrm{TRE}} = \langle p_1 \cdot p_2 \rangle_I$$

As illustrated, the specification φ_{RE} gets cumbersome quickly especially in the presence of several propositions. On the other hand, the specification φ_{TRE} conveys the intended meaning of pattern concisely. More importantly, notice how classical techniques to match φ'_{RE} might be very inefficient when numbers a and b are large. A minimal deterministic automaton constructed from φ'_{RE} if $I = [100, 104]$ has around 200 states. For a little more complex pattern $\varphi'_{\mathrm{RE}} \cdot \varphi'_{\mathrm{RE}}$, the number of states[1] surpasses 1200. It is clear that the essential problem here is the naive enumeration of time. Therefore, timing aspects should be represented in a quantitative and symbolic manner.

For this reason, we associate expressions with the algebra of timed relations in the following. We use this algebra together with propositional valuations to evaluate timed regular expressions in a compositional way. More precisely, we define the algebraic semantics of timed regular expressions as follows.

Definition 4.3 (Algebraic Semantics). *For a temporal structure $\mathcal{W} = (\Omega(T), V)$ induced by a timed behavior w, the valuation function V is extended to timed regular expressions as follows:*

$$\begin{array}{rclrcl} V(p) & = & V(p) \text{ for } p \in P & V(\varphi_1 \cdot \varphi_2) & = & V(\varphi_1) \cdot V(\varphi_2) \\ V(\varphi_1 \cup \varphi_2) & = & V(\varphi_1) \cup V(\varphi_2) & V(\varphi^{+}) & = & V(\varphi)^{+} \\ V(\varphi_1 \cap \varphi_2) & = & V(\varphi_1) \cap V(\varphi_2) & V(\langle \varphi \rangle_I) & = & \langle V(\varphi) \rangle_I \end{array}$$

An important property is that the algebraic semantics of timed regular expressions agrees with the relational semantics.

Proposition 4.1 (Semantic Agreement). *For every temporal structure $\mathcal{W} = (\Omega(T), V)$ induced by a timed behavior w and every timed regular expression φ, the statement $(\mathcal{W}, t, t') \models \varphi \leftrightarrow (t, t') \in V(\varphi)$ holds.*

Proof. By induction on the structure. The case of propositions holds by definition. For the concatenation $\varphi_1 \cdot \varphi_2$, we directly have

$$\begin{aligned} (\mathcal{W}, t, t') \models \varphi_1 \cdot \varphi_2 & \leftrightarrow (t, t') \in V(\varphi_1 \cdot \varphi_2) \\ & \leftrightarrow \exists t''.\ (t, t'') \in V(\varphi_1) \text{ and } (t'', t') \in V(\varphi_2) \\ & \leftrightarrow \exists t''.\ (\mathcal{W}, t, t'') \models \varphi_1 \text{ and } (\mathcal{W}, t'', t') \models \varphi_2 \end{aligned}$$

Cases for Boolean connectives and the plus are similar.

□

1 Some current industrial-level implementations such as Google's RE2 regex engine (https://github.com/google/re2) limit by default the maximum number of DFA states by 10000.

4.3 METRIC COMPASS LOGIC

In this section, we present the syntax and semantics of metric compass logic over a set of propositional variables P and follow a similar path as we just did for timed regular expressions. We introduce metric compass logic (MCL) to be a metric extension of the modal system HS of time periods developed by Halpern and Shoham in [48]. The HS system of six temporal modalities for relations A (adjacent/meets), B (begins), E (ends), and their inverses, A^{-1}, B^{-1}, E^{-1} is shown to be sufficient to have a full reasoning based on Allen's algebra on time periods [2]. We use the compass notation introduced by Venema in [112] and augment it with temporal constraints similar to the way that metric temporal logic (MTL) [62, 87] is extended from linear temporal logic [46]. Note that we slightly differ from original definitions by excluding degenerate time periods and employing irreflexive versions of modalities. It is known that the system HS is very expressive and its validity problem is undecidable. Therefore, much of previous efforts were devoted to find decidable and sufficiently expressive fragments; see the survey [36] for more details. Among such fragments, propositional neighborhood logic (PNL), which only includes modalities for meets and met-by relations of Allen, is extended with metric constraints [23] but there is no more expressive metric extension to our knowledge. It may be the case that undecidability results (of the validity problem) prevented any further investigation on metric extensions. However, since we propose metric compass logic to be yet another timed pattern specification language, we are only interested in evaluating an MCL formula for a given timed behavior. The problem of evaluation is easier than the validity; therefore, we do not have any reason to restrict ourselves to a fragment; thus, we extend the HS system in its full generality.

The syntax of metric compass logic that admits usual Boolean connectives and metric compass modalities is given by the following grammar:

$$\varphi \ := \ p \mid \overline{\varphi} \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \Diamond_I \, \varphi$$

where $p \in P$, a compass modality is $\Diamond \in \{\Diamond^{\uparrow}, \Diamond^{\downarrow}, \Diamond^{\leftarrow}, \Diamond^{\rightarrow}, \Diamond^{\lhd}, \Diamond^{\rhd}\}$ and I is an interval of duration values. As usual we define dual operators as $\Box_I \varphi = \overline{\Diamond_I \, \overline{\varphi}}$ and we omit the interval if $I = [0, \infty)$ or $I = (0, \infty)$. Given a timed behavior w, the satisfaction of a metric compass logic formula for a time period (t, t') is defined inductively.

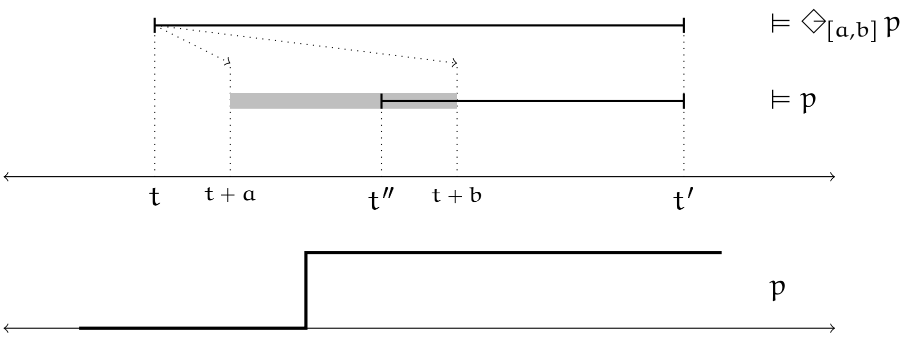


Figure 14: One dimensional illustration of $\diamondright_{[a,b]}$.

Definition 4.4 (Relational Semantics). *The satisfaction relation* $\models$ *of a metric compass logic formula* φ *in a temporal structure* $\mathcal{W} = (\Omega(T), V)$ *induced by a timed behavior* w *with a time domain* T, *relative to a time period* $(t,t') \in \Omega(T)$ *is defined as follows:*

$$
\begin{array}{lll}
(\mathcal{W},t,t') & \models p & \leftrightarrow \quad (t,t') \in V(p) \\
(\mathcal{W},t,t') & \models \overline{\varphi} & \leftrightarrow \quad (\mathcal{W},t,t') \not\models \varphi \\
(\mathcal{W},t,t') & \models \varphi_1 \cup \varphi_2 & \leftrightarrow \quad (\mathcal{W},t,t') \models \varphi_1 \text{ or } (\mathcal{W},t,t') \models \varphi_2 \\
(\mathcal{W},t,t') & \models \diamonddown_I\, \varphi & \leftrightarrow \quad \exists t < t'' < t'.\ t' - t'' \in I \text{ and } (\mathcal{W},t,t'') \models \varphi \\
(\mathcal{W},t,t') & \models \diamondup_I\, \varphi & \leftrightarrow \quad \exists t'' > t'.\ t'' - t' \in I \text{ and } (\mathcal{W},t,t'') \models \varphi \\
(\mathcal{W},t,t') & \models \diamondright_I\, \varphi & \leftrightarrow \quad \exists t < t'' < t'.\ t'' - t \in I \text{ and } (\mathcal{W},t'',t') \models \varphi \\
(\mathcal{W},t,t') & \models \diamondleft_I\, \varphi & \leftrightarrow \quad \exists t'' < t.\ t - t'' \in I \text{ and } (\mathcal{W},t'',t') \models \varphi \\
(\mathcal{W},t,t') & \models \diamondrightbar_I\, \varphi & \leftrightarrow \quad \exists t'' > t'.\ t'' - t' \in I \text{ and } (\mathcal{W},t',t'') \models \varphi \\
(\mathcal{W},t,t') & \models \diamondleftbar_I\, \varphi & \leftrightarrow \quad \exists t'' < t.\ t - t'' \in I \text{ and } (\mathcal{W},t'',t) \models \varphi
\end{array}
$$

Each MCL modality above fixes one dimension (either t or t') in their semantics, and a metric constraint restricts the range of quantification over the other (free) dimension. For example, the modality $\diamondright_{[a,b]}$ fixes the endpoint t' and quantifies over a restricted range $[t+a, t+b]$ as illustrated in Figure 14. Note that $\diamondright$ can be also seen two-dimensional analog of Prior's (and Pnueli's) eventuality (F) if you consider endpoints of periods to be fixed at the infinity and the reasoning is essentially performed over (begin) points.

Then, for a temporal structure $\mathcal{W} = (\Omega(T), V)$, we extend the valuation function V for arbitrary MCL formulas and define the algebraic semantics of MCL as follows.

Definition 4.5 (Algebraic Semantics). *For a temporal structure* $\mathcal{W} = (\Omega(T), V)$, *we extend the valuation function* V *for arbitrary* MCL *formulas as follows:*

$$
\begin{array}{rclrcl}
V(p) & = & V(p) \text{ for } p \in P & V(\varphi_1 \cup \varphi_2) & = & V(\varphi_1) \cup V(\varphi_2) \\
V(\overline{\varphi}) & = & \Omega(T) \backslash V(\varphi) & V(\varphi_1 \cap \varphi_2) & = & V(\varphi_1) \cap V(\varphi_2) \\
V(\Diamond_I\, \varphi) & = & \Diamond_I\, V(\varphi) & & &
\end{array}
$$

In Proposition 4.2, we show that algebraic and relational semantics of metric compass logic agree straightforwardly.

Proposition 4.2 (Semantic Agreement). *For every temporal structure* $\mathcal{W} = (\Omega(T), V)$ *and every* MCL *formula, the statement* $(\mathcal{W}, t, t') \vDash \varphi \leftrightarrow (t, t') \in V(\varphi)$ *holds.*

Proof. By induction on the structure. Cases for propositions and Boolean operations are straightforward. Then we only show the case $\diamondminus$ as cases for other compass operators are symmetric. We directly show

$$\begin{aligned}(\mathcal{W}, t, t') \vDash \diamondminus_I \varphi \leftrightarrow (t, t') \in V(\diamondminus_I \varphi) & \\ \leftrightarrow \exists t''.\ t < t'' < t',\ t'' - t \in I,& \\ \text{and } (t'', t') \in V(\varphi)& \\ \leftrightarrow \exists t''.\ t < t'' < t',\ t'' - t \in I,& \\ \text{and } (\mathcal{W}, t'', t') \vDash \varphi&\end{aligned}$$

□

4.4 OFFLINE MATCHING

Pattern matching is usually considered to be a computation for finding and reporting all satisfying segments, called matches, of an input sequence that satisfy a predefined pattern. In this section, we introduce timed pattern matching over timed behaviors where patterns are specified by timed regular expressions and metric compass logic. Then, we say the set of all satisfying segments is called the match set of the pattern over a timed behavior.

Assuming the input behavior w (thus valuations of propositions) is completely available before matching, computing the match set is equivalent to evaluating φ inductively in the temporal structure induced by w. In Algorithm 7, we present the offline evaluation algorithm $\text{EVAL}_{\mathcal{W}}(\varphi)$, suggested by the algebraic semantics of timed regular expressions and metric compass logic, for both formalisms in a unified manner. Since every timed regular expression and metric compass logic formula can be seen as a proposition in a temporal structure, it is straightforward to mix connectives/operators of both and have a richer pattern specification language for offline timed pattern matching purposes.

We illustrate of offline timed pattern matching for a timed regular expression $\varphi = p_1 \cdot \langle p_2 \cdot p_3 \rangle_{[2,4]} \cap \langle p_1 \cdot p_2 \rangle_{[3,5]} \cdot p_3$ and a timed behavior w over propositions p_1, p_2, and p_3 as given in Figure 15 at the top. The middle row of the figure illustrates match sets or valuations of subexpressions over the behavior w obtained in a bottom-up fashion according to the Algorithm 7. Finally, we illustrate the match set for the expression φ at the final row.

Algorithm 7 $\textsc{eval}_W(\varphi)$ with respect to $W = (\Omega(T), V)$

```
select ($\varphi$)
case $p$:
    $Z := V(p)$
case $\overline{\psi}$:
    $Z := \textsc{complement}(\textsc{eval}_W(\psi))$
case $\psi_1 \cup \psi_2$:
    $Z := \textsc{union}(\textsc{eval}_W(\psi_2),\ \textsc{eval}_W(\psi_2))$
case $\psi_1 \cap \psi_2$:
    $Z := \textsc{intersect}(\textsc{eval}_W(\psi_2),\ \textsc{eval}_W(\psi_2))$
case $\psi_1 \cdot \psi_2$:
    $Z := \textsc{concatenate}(\textsc{eval}_W(\psi_2),\ \textsc{eval}_W(\psi_2))$
case $\psi^+$:
    $Z := \textsc{closure}(\textsc{eval}_W(\psi))$
case $\langle\psi\rangle_I$:
    $Z := \textsc{restrict}(\textsc{eval}_W(\psi), I)$
case $\Diamond_I\,\psi$:
    $Z := \Diamond\text{-}\textsc{shift}(\textsc{eval}_W(\psi), I)$
end select
return $Z$
```

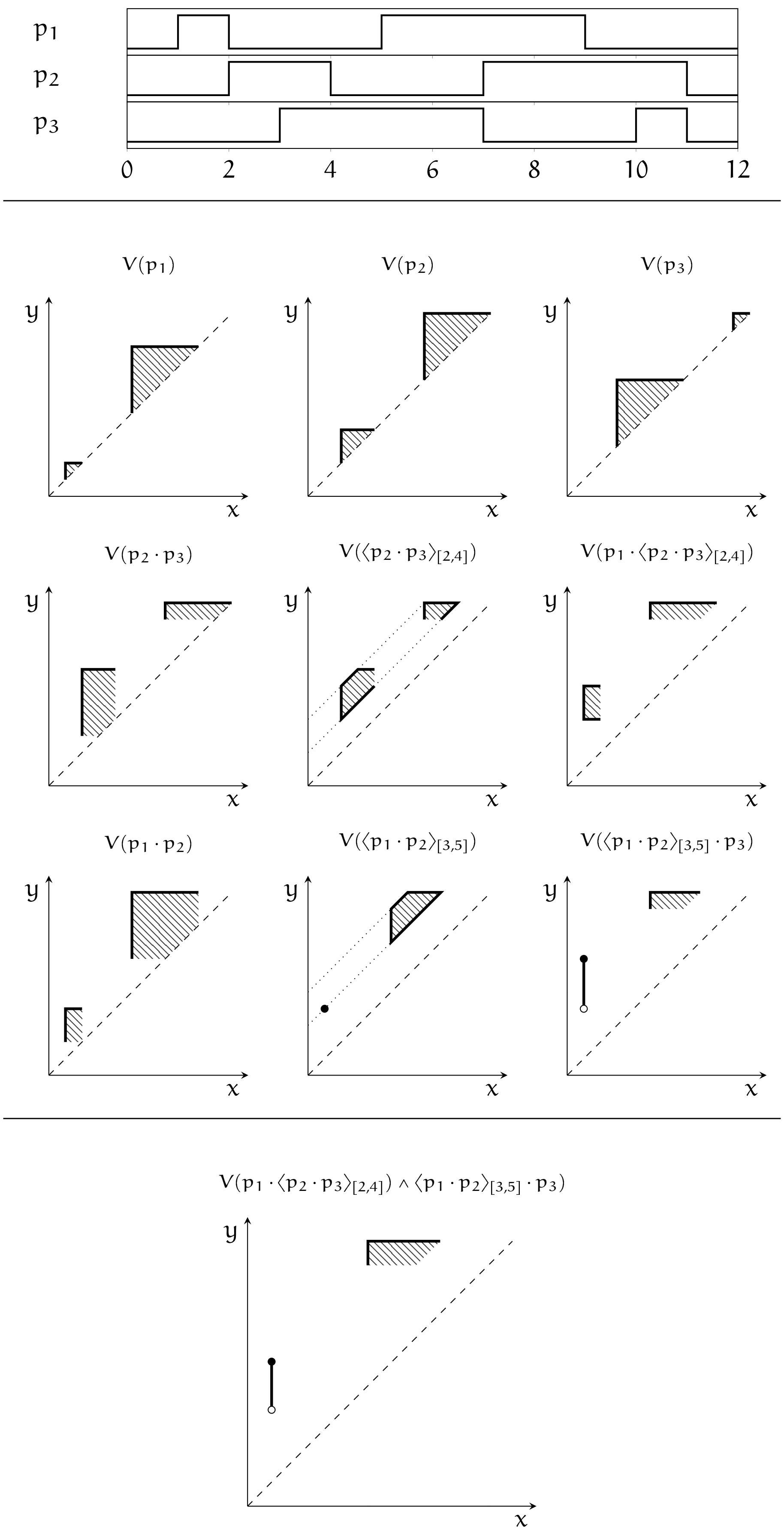
4.5 SOME EXTRA FEATURES

In this section, we present some additional features for timed pattern matching to enhance its usefulness in practice. Since a valuation function in our system can be any function that returns a timed relation over a temporal sequence, we indeed have a great deal of freedom for atomic expressions and their valuations. In the following, we describe several natural extensions in this direction.

Predicates over Real-Valued Behaviors. Atomic propositions can be easily extended towards predicates over real-valued behaviors in the usual sense. Although simple, it is a very useful extension in practice when reasoning about physical observations and hybrid systems in general. The simplest and most common type of predicates are threshold comparisons such as $p(x) : x < c$ and $p(x) : x > c$ where x is a real-valued variable and c is a constant [72]. Consequently we can apply timed pattern matching over real-valued signals via such a symbolic abstraction for the value domain, sometimes called quantization or categorization. For example, a timed pattern φ over a continuous-time temperature signal $temp$ that specifies an alternation between qualitative states denoted by predicates $(temp < 27)$ and $(temp \geqslant 27)$ can be expressed as follows:

$$\varphi = \left(\langle temp < 27\rangle_{[10,20]} \cdot \langle temp \geqslant 27\rangle_{[10,20]}\right)^+$$

For the rest of the thesis, we will use propositions and predicates interchangeably.



p_1
p_2
p_3
0
2
4
6
8
10
12
$V(p_1)$
$V(p_2)$
$V(p_3)$
$V(p_2 \cdot p_3)$
$V(\langle p_2 \cdot p_3 \rangle_{[2,4]})$
$V(p_1 \cdot \langle p_2 \cdot p_3 \rangle_{[2,4]})$
$V(p_1 \cdot p_2)$
$V(\langle p_1 \cdot p_2 \rangle_{[3,5]})$
$V(\langle p_1 \cdot p_2 \rangle_{[3,5]} \cdot p_3)$
$V(p_1 \cdot \langle p_2 \cdot p_3 \rangle_{[2,4]}) \wedge \langle p_1 \cdot p_2 \rangle_{[3,5]} \cdot p_3)$
x
y

Figure 15: Illustration of offline timed pattern matching.

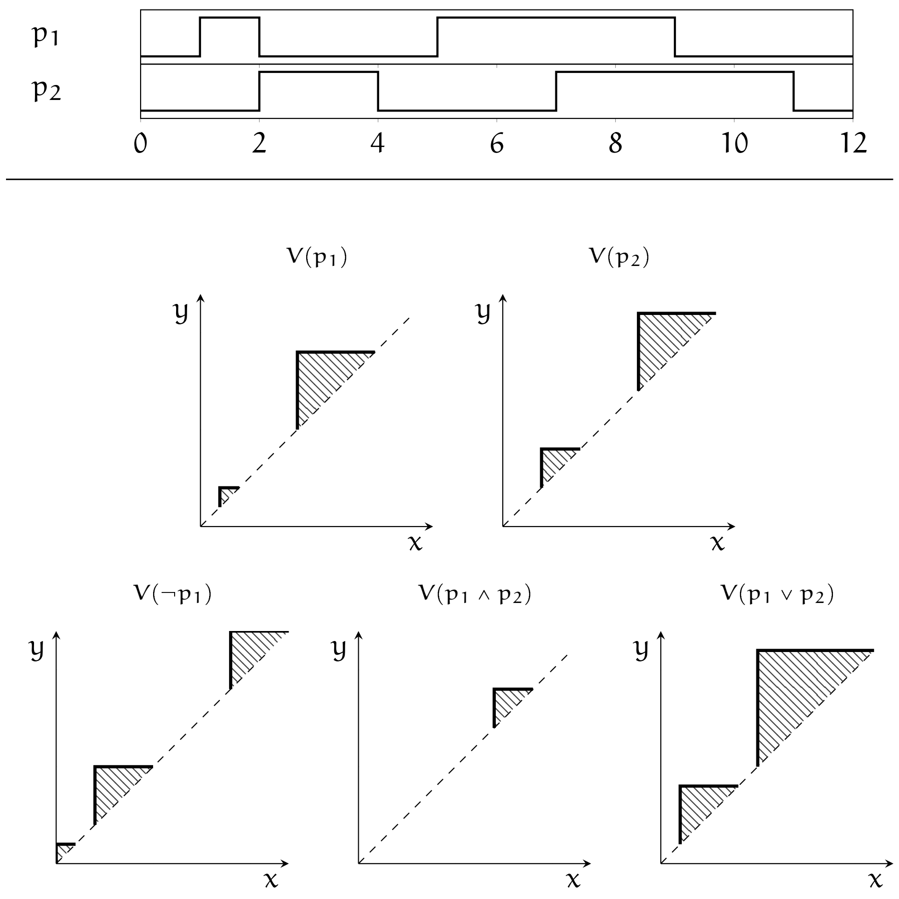


Figure 16: Valuations of Boolean expressions over atomic propositions.

Boolean Layer. Given a set $P = \{p_1, \dots, p_m\}$ of propositions, we extend atomic propositions towards Boolean expressions over propositions. The syntax of Boolean expressions over P, called the Boolean layer, are defined as

$$\phi := p \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

where $p \in P$ and $\neg$, $\wedge$ are negation and conjunction operators. We derive the disjunction $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$ as usual. Intuitively speaking, such a Boolean expression ϕ can be viewed as a new proposition in the system and its truth value, denoted by $\phi(a)$, for $a \in \Sigma$ at each time period are obtained by applying a substitution $\{p_1 \mapsto a(i), \dots, p_m \mapsto a(m)\}$. More precisely, Boolean operations on $V(p)$ for $p \in P$ are characterized via the valuation function V as follows. Provided that $V(\phi)$, $V(\phi_1)$, $V(\phi_2)$ satisfy the homogeneity property, we define

$$\begin{aligned} V(\neg\phi) &= \{(t,t') \in \Omega \mid \forall r, r'.\ t < r < r' < t' \rightarrow (r,r') \notin V(\phi)\} \\ V(\phi_1 \wedge \phi_2) &= V(\phi_1) \cap V(\phi_2) \end{aligned}$$

It is easily seen that $V(\neg\neg\phi) = V(\phi)$ and $V(\phi \wedge \neg\phi) = \varnothing$. In Figure 16, we illustrate valuations of such Boolean expressions over two example proposition p_1 and p_2.

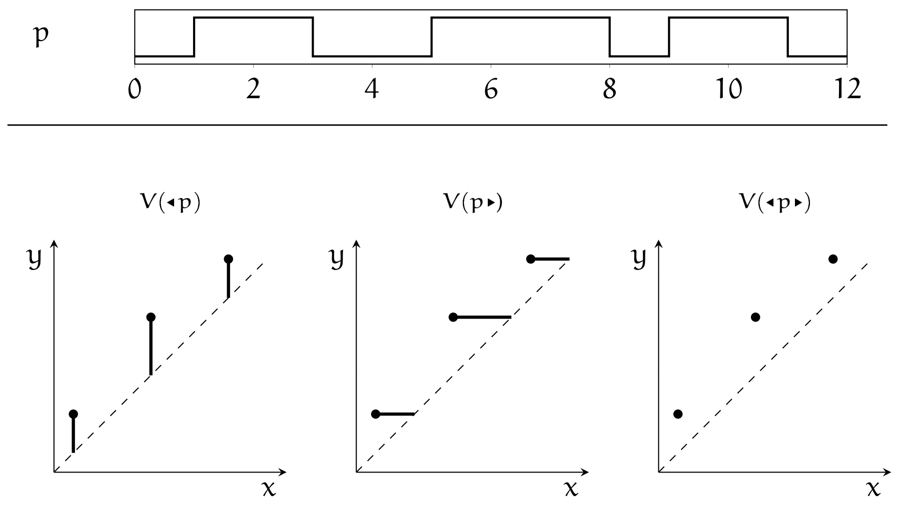


Figure 17: Valuations of anchored expressions.

Anchors. We introduce three anchor operations over atomic propositions, namely prefix rise (◂), postfix fall (▸), and outfix rise-fall(◂ ▸) operators. These operators are motivated by a practical need to fix begin [end, begin-end] points of matches to rise [fall, rise-fall] points of propositions in timed behaviors. Valuation of these operations on an atomic proposition ϕ is defined as

$$\begin{aligned}
V(\blacktriangleleft\phi) &= \{(t,t') \in V(\phi) \mid \forall t''.\ (t'',t) \notin V(\phi)\} \\
V(\phi\blacktriangleright) &= \{(t,t') \in V(\phi) \mid \forall t''.\ (t',t'') \notin V(\phi)\} \\
V(\blacktriangleleft\phi\blacktriangleright) &= V(\blacktriangleleft\phi) \cap V(\phi\blacktriangleright)
\end{aligned}$$

Note that anchor operations do not preserve the homogeneity property, therefore; anchored expressions can be viewed as nonhomogenous propositional atoms. In Figure 17, we illustrate valuations of anchored expressions over a proposition p.

Instantaneous Events. We show how to cast instantaneous events into our timed pattern matching system in a natural way. Let E be a set of instantaneous events. A timed event is usually considered to be a pair (t, e) such that $t \in T$ is a time point and e is an event $e \in E$. Without loss of generality, we define a timed event sequence $s = (t_1, e), (t_2, e), \dots, (t_n, e)$ over an event e to be a sequence such that $t_1 < t_2 < \cdots < t_n$. Although it seems that we need to introduce time points back into our system, there are several ways to express instantaneous events without time points. First, an instantaneous event can be considered to occur at an arbitrarily small period $(t - \varepsilon, t + \varepsilon)$ centered around the point t where for a positive ε. However, this solution may require an increase in time granularity and cause practical problems; therefore, we do not use arbitrary small time periods for events. Our solution is as follows. Intuitively speaking, we consider the valuation of a timed event e to be the set of all time periods such that e occurs during all these time periods. In other words, an event

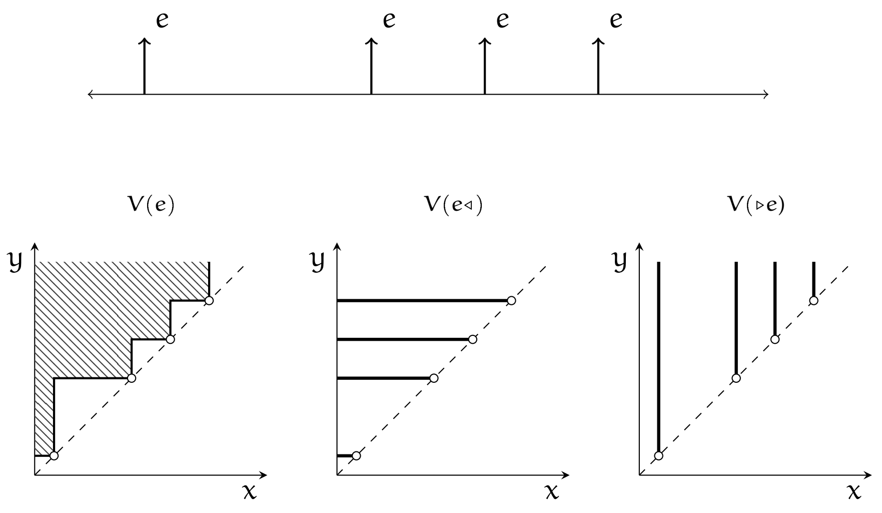


Figure 18: Valuations of instantaneous events.

(t, e) is matched by all time periods $\{(r, r') \mid \exists r, r'.\ r \leqslant t \leqslant r'\}$. Consequently the valuation of an event e with respect to a timed event sequence s is defined as

$$V(e) = \{(t, t') \in \Omega(T) \mid \exists r, r'.\ r \leqslant t \leqslant r' \text{ and } (s, t) \vdash e\}$$

where $(s, t) \vdash e$ holds if and only if the event e occurs at t in s. Besides we define two other event operators $(\triangleright e)$ and $(e\triangleright)$ that restrict valuations to only include time periods starting with the event e and time periods ending with the event e, respectively as follows.

$$\begin{aligned} V(\triangleright e) &= \{(t, r') \in \Omega(T) \mid \exists r'.\ (s, t) \leqslant r' \text{ and } t \vdash e\} \\ V(e\triangleleft) &= \{(r, t) \in \Omega(T) \mid \exists r.\ r \leqslant t \text{ and } (s, t) \vdash e\} \end{aligned}$$

In Figure 18, we illustrate valuations of event expressions over a timed sequence. Observe that a timed pattern φ such that an event e_1 is followed by another event e_2 can be written as $\varphi = e_1 \cdot e_2$ and matching the expression φ would return all time periods on which e_1 is followed by the event e_2.

5 GOING ONLINE FOR EXPRESSIONS

So the whole idea of science
is to introduce coordinates
that move sequentially in time.

— John Rhodes

Detecting patterns as soon as they occur in ongoing system behaviors has many important applications ranging from runtime verification to supervisory control and autonomous agents. It requires that matches are computed and reported in an incremental manner for each incoming segment of the input behavior. For that we first summarize our understanding on online computations formalized in the form of sequential functions [94] and the direct use of derivatives of regular expressions [26] to realize such functions for online pattern matching. Recall that derivatives are given as a set of rewriting rules for regular expressions with respect to a letter. Successive rewritings of a regular expression φ for each letter of a word w read from left to right can be easily viewed as an online procedure to check whether w is in the language of φ. Furthermore this process indeed yields a Boolean (acceptance) value for each prefix of the word w sequentially. This fact constitutes the starting point of this chapter.

We then introduce derivatives of timed regular expressions, in short, timed derivatives, by carefully adapting the concept to the timed setting. The major challenge in the online setting is to represent and manipulate all partial matches, that is to say, suffixes of the input behavior that satisfy prefixes of the pattern at the current step. To solve this problem, we first allow timed relations to appear as constants and coefficients in regular expressions similar to the numbers in arithmetic expressions. Then, our derivative operation essentially captures a regulated substitution mechanism that substitutes propositional variables in regular expressions with timed relations[1] with respect to the incoming segment. Partial matches would appear in derived expressions as (left) coefficients in derived expressions and full matches as constants. Consequently, the state of the online procedure is represented elegantly and transparently in the form of a single expression that contains propositional variables, timed relations, and operators.

Once we have developed such expressions and derivative rules, our online timed pattern matching algorithm becomes immediate and it is easy to show its correctness.

1 In the classical setting, the empty word is the only non-zero constant. From our point of view, we can say that letters are substituted by the empty word, thus deleted, by Brzozowski's derivative operation.

5.1 SEQUENTIAL FUNCTIONS

In this section, we briefly mention sequential functions and show how to realize some of them using derivatives. This realization would form the basis when extending timed pattern matching to an online setting. Let us now give the precise definition of sequential functions.

Let $f : X^+ \to Y$ be a mapping from sequences of elements of an input domain X to an output value $y \in Y$. Then a function $F : X^+ \to Y^+$ thats transforms an input sequence into an output sequence such that

$$F(x_1, \dots, x_n) = y_1, y_2, \dots, y_n = f(x_1), f(x_1 x_2), \dots, f(x_1, \dots, x_n)$$

is called a (causal) sequential function [94]. In other words, a sequential function maps every prefix of the input sequence to an output value in a sequential fashion. Note that a sequential function is a more semantic notion than transducers or sequential machines (e.g. Mealy [77] and Moore [78] machines) that implies an automaton structure. Recall that the membership test of a word $w \in \mathcal{L}$ for a language $\mathcal{L}$ is equivalent to the empty word check $\epsilon \in D_w(\mathcal{L})$ for the derived language of $\mathcal{L}$ with respect to w. Hence, to check whether abc is in the language of the expression $\varphi = a^* \cdot (b \cdot c)^*$, we compute $D_{abc}(\varphi) = D_c(D_b(D_a(\varphi))) = (b \cdot c)^*$ as follows:

$$a^* \cdot (b \cdot c)^* \underset{D_a}{\longrightarrow} a^* \cdot (b \cdot c)^* \underset{D_b}{\longrightarrow} c \cdot (b \cdot c)^* \underset{D_c}{\longrightarrow} (b \cdot c)^*,$$

and since $\nu((b \cdot c)^*) = \epsilon$, $abc \in [\![\varphi]\!]$. Notice that such successive rewriting of regular expressions and outputting true if the derived expression contains the empty word ϵ realizes a sequential function F such that f is the language membership function. The function F can be viewed as an online prefix matcher exactly like a DFA that outputs the acceptance of its current state.

Two-sided matching is more involved as we ask the membership of all factors of w, starting at arbitrary positions. Thus, having read j letters of w, the state of a matching algorithm should contain all the derivatives by $w[i..j]$, $i \leqslant j$. When letter $j+1$ is read, these derivatives are updated to become derivatives by $w[i..j+1]$, new matches are extracted and a new process for matches that start at $j+1$ is spawned. Table 2 illustrates the systematic application of derivatives to find segments of $w = abcbcd$ that match $\varphi = a^* \cdot (b \cdot c)^*$. The table is indexed by the start position (rows) and end position (columns) of the segments with respect to which we derive. Derivatives that contain ϵ correspond to matches and their time indexes constitute the match set $\{(1,1),(1,3),(1,5),(2,3),(2,5),(4,5)\}$. Importantly matches are obtained incrementally according to their end points. For example, we obtain $\{(1,1)\}$ at time 1, $\{(1,3),(2,3)\}$ at time 3, and $\{(1,3),(2,3),(4,5)\}$ at time 5. For times 2 and 4, we do not obtain any matches thus we have $\varnothing$ as the output. In short, we report a match immediately after it is matched.

In a discrete finite-state setting, there are finitely many such derivatives but this is not the case for timed systems. In timed setting,

Letters			a		b		c	
Positions			1		2		3	
1	φ	$\xrightarrow[D_a]{}$	$a^*(bc)^*$	$\xrightarrow[D_b]{}$	$c(bc)^*$	$\xrightarrow[D_c]{}$	$(bc)^*$	$\xrightarrow[D_b]{}$
2			φ	$\xrightarrow[D_b]{}$	$c(bc)^*$	$\xrightarrow[D_c]{}$	$(bc)^*$	$\xrightarrow[D_b]{}$
3					φ	$\xrightarrow[D_c]{}$	$\varnothing$	$\xrightarrow[D_b]{}$
4							φ	$\xrightarrow[D_b]{}$

Letters		b		c		d
Positions		4		5		6
1	$\xrightarrow[D_b]{}$	$c(bc)^*$	$\xrightarrow[D_c]{}$	$(bc)^*$	$\xrightarrow[D_d]{}$	$\varnothing$
2	$\xrightarrow[D_b]{}$	$c(bc)^*$	$\xrightarrow[D_c]{}$	$(bc)^*$	$\xrightarrow[D_d]{}$	$\varnothing$
3	$\xrightarrow[D_b]{}$	$\varnothing$	$\xrightarrow[D_c]{}$	$\varnothing$	$\xrightarrow[D_d]{}$	$\varnothing$
4	$\xrightarrow[D_b]{}$	$c(bc)^*$	$\xrightarrow[D_c]{}$	$(bc)^*$	$\xrightarrow[D_d]{}$	$\varnothing$
5		φ	$\xrightarrow[D_c]{}$	$\varnothing$	$\xrightarrow[D_d]{}$	$\varnothing$
6				φ	$\xrightarrow[D_d]{}$	$\varnothing$

Table 2: Pattern matching using derivatives for $w = abcbcd$ and $\varphi = a^* \cdot (b \cdot c)^*$. Entry (i,j) represents the derivative with respect to $w[i,j]$. Derivatives containing ϵ are shaded with gray. The state of an online matching algorithm after reading j symbols is represented in column j.

the analogue of the arrival of a new letter is the arrival of a segment (t_1, t_2, a) of the behavior w. When this occurs, the state of the matching should be updated to capture all derivatives by segments of the form (t, t_2) for $t < t_2$ and all matches ending in some $t < t_2$ should be returned. In the following, we show our symbolic technique for representing and manipulating such an uncountable number of derivatives together with their corresponding time segments.

5.2 TIMED DERIVATIVES

In the classical discrete setting, the derivative D_a is associated with a rewrite rule $a \to \epsilon$ and a word w is accepted if it can be transformed into the empty word ϵ by successive rewritings. For the purpose of timed pattern matching, we need a similar but duration-preserving view where reading a segment (t, t', a) corresponds to a rule $(t, t', a) \to (t, t')$. Matching a pattern that contains a single behavior $w(t, t')$ is then the rewriting (reduction) of $w(t, t')$ into a time period (t, t') from past to future (left to right). Since timed regular expressions denote sets of timed behaviors in general, online timed pattern matching using derivatives corresponds to the rewriting of timed regular expressions with respect to an input behavior. In this section, we define timed behavior languages for timed regular expressions extended with constants taken from the set $\mathcal{Z}$ of timed relations. Then we define derivatives of timed regular expressions and solve the problem of online timed pattern matching using the derivatives.

We first extend timed behaviors by allowing time periods as prefixes in compliance with the concatenation between timed behaviors. For example, a timed behavior w over an alphabet Σ having a time period prefix $(t_0, t_1) \cdot \cdots \cdot (t_{i-1}, t_i)$ can be written as

$$w = (t_0, t_1) \cdot \cdots \cdot (t_{i-1}, t_i) \cdot (t_i, t_{i+1}, a_{i+1}) \cdot \cdots \cdot (t_{n-1}, t_n, a_n)$$

where $a_{i+1}, \ldots, a_n \in \Sigma$. Throughout this chapter, we say a timed behavior is *pure* if it does not have a time period prefix and *reduced* if it only consist of time periods. In particular, we use initial Greek letters, α, β, and γ to denote reduced timed behaviors and hence the timed behavior w will be written as $w = \alpha \cdot v$ where α and v are reduced and pure timed behaviors, respectively. As usual, we use the symbol ϵ for the identity element of the concatenation satisfying that $w = \epsilon \cdot w = w \cdot \epsilon$. Since a sequence of time periods can be compacted under stuttering rules, we mostly, if not all, consider such prefixes to be single time periods. By slightly abusing the notation, we then write the set of all reduced timed behaviors as Ω and the set of all timed behaviors that a time period concatenated from left as $\Sigma_{\Omega}^{(+)}$. Subsets of $\Sigma_{\Omega}^{(+)}$ are called timed behavior languages as usual. We also define the star versions $\Omega^* = \Omega \cup \{\epsilon\}$ and $\Sigma_{\Omega}^{(*)} = \Sigma_{\Omega}^{(+)} \cup \{\epsilon\}$ for the sake of conciseness.

We now start by defining a reduction operation over timed behavior languages as follows.

Definition 5.1 (Left Reduction). *The left reduction δ of a timed behavior language $\mathcal{L}$ with respect to a timed behavior u on a time period β is given such that*

$$\delta_u(\mathcal{L}) = \{\ \alpha\beta v \mid \alpha u v \in \mathcal{L},\ \alpha \in \Omega(T),\ \textit{and}\ v \in \Sigma^{(*)}\}$$

We say that the pair (u, β) is the reduction rule $R(u)$ of u.

We use operation $\delta_u(\mathcal{L})$ in a similar way $D_u(\mathcal{L})$ is used in the classical setting but with one important difference. When $v = D_u(w)$ the length of the word is reduced, that is, $|v| = |w| - |u|$, while when $v = \delta_u(w)$ the domains (and hence durations) of v and w are the same. Consequently, unlike the classical case where membership of w in $\mathcal{L}$ amounts to $\epsilon \in D_w(\mathcal{L})$, here the membership is equivalent to $\gamma \in \delta_w(\mathcal{L})$ where γ is the time period on which the timed behavior w located. In other words, the reduced set contains the location of u if u in the language $\mathcal{L}$. For example, consider a timed behavior language $\mathcal{L} = \{w_1, w_2\}$ such that $w_1 = (0, 3, a) \cdot (3, 5, b)$ and $w_1 = (0, 2, a) \cdot (2, 5, b)$. We illustrate a left reduction operation $\delta_{u_3}(\delta_{u_2}(\delta_{u_1}(\mathcal{L}))) = \{w_1'''\}$ with respect to $u = u_1 u_2 u_3$ with $u_1 = (0, 1, a)$, $u_2 = (1, 3, a)$, and $u_3 = (3, 5, b)$ in Figure 19. Since

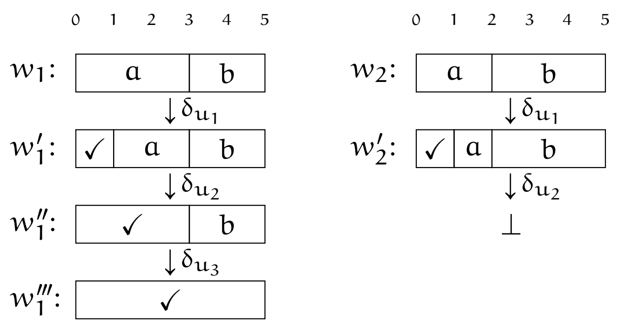


Figure 19: A left reduction example.

$w_1''' = (0, 5)$ is a reduced behavior, we have that $u \in \mathcal{L}$. Note that it is easy to verify that $\delta_{u_3}(\delta_{u_2}(\delta_{u_1}(\mathcal{L}))) = \delta_{u_1 \cdot u_2 \cdot u_3}(\mathcal{L})$.

So far we showed the reduction with respect a single behavior; however, a more interesting case would be the extension of the reduction operation for a set of timed behavior. We now introduce a derivative operation for timed behavior languages based on the left reduction operation. Since our goal is to solve the dense time matching problem, we have to operate on sets of timed behaviors and define derivatives more symbolically. Therefore, we define the derivative Δ_v to correspond to the left reduction with respect to all factors of v.

Definition 5.2 (Dense Derivation). *The dense derivation $\Delta_v(\mathcal{L})$ of a timed behavior language $\mathcal{L}$ with respect to a uniform timed behavior $v = (t, t', a) \in \Sigma^{(1)}$ is defined as follows:*

$$\Delta_v(\mathcal{L}) := \bigcup_{u \in \textsc{sub}(v)} \delta_u(\mathcal{L})$$

where $\textsc{sub}(v) = \{(r, r', a) \mid t \leqslant r < r' \leqslant t'\}$.

As mentioned previously, we consider reduced timed behaviors (time periods) to be the output of our matching procedure. Their existence in derived languages will be the witness of a match in the behavior; therefore we define the extraction (output) function as follows.

Definition 5.3 (Extraction). *The extraction* $\mathrm{xt}(\mathcal{L})$ *of a timed behavior language $\mathcal{L}$ is defined as follows:*

$$\mathrm{xt}(\mathcal{L}) := \{\ \alpha \mid \alpha \in \Omega(T) \textit{ and } \alpha \in \mathcal{L}\}$$

We now introduce timed regular $\mathcal{Z}$-expressions with constants taken form the set $\mathcal{Z}$ of timed relations. Below we first give the general syntax of timed regular $\mathcal{Z}$-expressions, then we present a syntactic class to precisely describe sets of timed behaviors (with time periods concatenated from left).

Definition 5.4 ($\mathcal{Z}$-Expression). *Timed regular $\mathcal{Z}$-expressions are defined over a set of propositions P by the following grammar:*

$$\varphi \ := \ Z \mid p \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \varphi^+ \mid \langle \varphi \rangle_I$$

where p is a propositional variable in P, Z is a timed relation, and I is an interval of duration values.

We note that, although we do not consider the identity ϵ in timed languages, we use $\varphi^* = \varphi^+ \cup \epsilon$ for the sake of conciseness. In this chapter, all such cases can be rewritten in the syntax above using identity and distributivity properties of the regular algebra.

Definition 5.5 ($\mathcal{Z}$-Language)**.** *The language of a timed regular $\mathcal{Z}$-expressions is a set of timed behaviors defined by the following rules.*

$$\begin{aligned}
[\![Z]\!] &= Z \\
[\![p]\!] &= \{(t, t', a) \mid t < t' \textit{ and } a(p) = 1\} \\
[\![\varphi \cdot \psi]\!] &= [\![\varphi]\!] \cdot [\![\psi]\!] \\
[\![\varphi \cup \psi]\!] &= [\![\varphi]\!] \cup [\![\psi]\!] \\
[\![\varphi \cap \psi]\!] &= [\![\varphi]\!] \cap [\![\psi]\!] \\
[\![\varphi^+]\!] &= \textstyle\bigcup_{i=1}^{\infty} [\![\varphi]\!]^i \\
[\![\langle\varphi\rangle_I]\!] &= \{w \mid w \in [\![\varphi]\!] \textit{ and } |w| \in I\}
\end{aligned}$$

The syntax in Definition 5.4 allows to define sets that contains timed behaviors with arbitrary interleavings of pure behaviors and time periods. Below we define three syntactic classes of expressions. The first class, called pure (or original) timed regular expressions, corresponds to the same syntax of expressions in Chapter 4 and pure expressions are Z-free. The second class is reduced timed regular expressions which is formed by timed relations only. Lastly, we have left-reduced timed regular expressions, obtained as compositions of reduced and pure expressions satisfying some conditions.

Definition 5.6 (Syntactic Classes)**.** *A timed regular expression φ belongs to the classes of* reduced, pure *or* left-reduced *timed regular expressions if functions* $r?$, $p?$ *or* $lr?$, *respectively, evaluate to* true *in the following table.*

Case	*Reduced* $r?(\varphi)$	*Pure* $p?(\varphi)$	*Left-reduced* $lr?(\varphi)$
Z	`true`	`false`	`true`
p	`false`	`true`	`true`
$\varphi_1 \cdot \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge p?(\varphi_2) \cup$ $r?(\varphi_1) \wedge lr?(\varphi_2)$
$\varphi_1 \cup \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge lr?(\varphi_2)$
$\varphi_1 \wedge \varphi_2$	$r?(\varphi_1) \wedge r?(\varphi_2)$	$p?(\varphi_1) \wedge p?(\varphi_2)$	$lr?(\varphi_1) \wedge lr?(\varphi_2)$
φ^+	$r?(\varphi)$	$p?(\varphi)$	$r?(\varphi) \cup p?(\varphi)$
$\langle\varphi\rangle_I$	$r?(\varphi)$	$p?(\varphi)$	$lr?(\varphi)$

Trivially any reduced expression ψ and any pure expression φ represent reduced and pure timed behavior languages such that $[\![\psi]\!] \subseteq \Omega$ and $[\![\varphi]\!] \subseteq \Sigma^{(+)}$. For left-reduced expressions we do not allow concatenation and star operations on arbitrary left-reduced expressions as in Definition 5.6. By doing that, we have the following result.

Proposition 5.1. *The language* $[\![\varphi]\!]$ *of a left-reduced timed regular expression* φ *is a timed behavior language such that* $[\![\varphi]\!] \subseteq \Sigma_{\Omega}^{(+)}$.

Proof. For the concatenation $\varphi_1 \cdot \varphi_2$, we have two possibilities: (1) $[\![\varphi_1]\!] \subset \Sigma_{\Omega}^{(+)}$ and $[\![\varphi_2]\!] \subset \Sigma^{(+)}$; (2) $[\![\varphi_1]\!] \subseteq \Omega$ and $[\![\varphi_2]\!] \subseteq \Sigma_{\Omega}^{(+)}$. For both possibilities, we have $[\![\varphi_1 \cdot \varphi_2]\!] = [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \subseteq \Sigma_{\Omega}^{(+)}$. Other cases are straightforward by following the definitions. □

We now introduce a syntactic derivative operation for left-reduced timed regular expressions. First, we have that the extraction xt can be computed syntactically for left-reduced timed regular expressions.

Theorem 5.2 (Extraction Computation)**.** *For a given left-reduced timed regular expression* φ*, applying the following rules recursively yields a timed relation* $Z = \mathrm{xt}([\![\varphi]\!])$.

$$\begin{array}{lcllcl}
\mathrm{xt}(Z) & = & Z & \mathrm{xt}(\psi_1 \cdot \psi_2) & = & \mathrm{xt}(\psi_1) \cdot \mathrm{xt}(\psi_2) \\
\mathrm{xt}(p) & = & \varnothing & \mathrm{xt}(\psi_1 \cup \psi_2) & = & \mathrm{xt}(\psi_1) \cup \mathrm{xt}(\psi_2) \\
\mathrm{xt}(\langle\psi\rangle_I) & = & \langle \mathrm{xt}(\psi)\rangle_I & \mathrm{xt}(\psi_1 \cap \psi_2) & = & \mathrm{xt}(\psi_1) \cap \mathrm{xt}(\psi_2) \\
\mathrm{xt}(\psi^+) & = & \mathrm{xt}(\psi)^+ & & &
\end{array}$$

Proof. We proceed by induction and only look at the case of concatenation, other cases are similar. For any expressions φ_1, φ_2 it holds

$$\begin{array}{rcl}
[\![\mathrm{xt}(\varphi_1 \cdot \varphi_2)]\!] & = & \{\alpha \mid \alpha \in \Omega \text{ and } \alpha \in [\![\varphi_1 \cdot \varphi_2]\!]\} \\
& = & \{\alpha_1\alpha_2 \mid \alpha_1, \alpha_2 \in \Omega,\ \alpha_1 \in [\![\varphi_1]\!] \text{ and } \alpha_2 \in [\![\varphi_2]\!]\} \\
& = & \{\alpha_1 \mid \alpha_1 \in \Omega \text{ and } \alpha_1 \in [\![\varphi_1]\!]\} \cdot \\
& & \quad \{\alpha_2 \mid \alpha_2 \in \Omega \text{ and } \alpha_2 \in [\![\varphi_2]\!]\} \\
& = & [\![\mathrm{xt}(\varphi_1)]\!] \cdot [\![\mathrm{xt}(\varphi_2)]\!]
\end{array}$$

□

We state our main result in this section concerning derivatives of (left-reduced) timed regular expressions.

Theorem 5.3 (Derivative Computation)**.** *Given a (left-reduced) timed regular expression* φ *and a uniform timed behavior* $v : (t, t', a)$*, applying the following rules yields an expression* ψ *such that* $[\![\psi]\!] = \Delta_v([\![\varphi]\!])$.

$$\begin{array}{rcl}
\Delta_v(Z) & = & \varnothing \\
\Delta_v(p) & = & \begin{cases} Z \cup Z \cdot p & \textit{if } a(p) = 1 \textit{ where} \\ & \quad Z = \{(r, r') \mid t \leqslant r < r' \leqslant t'\} \\ \varnothing & \textit{otherwise} \end{cases} \\
\Delta_v(\psi_1 \cdot \psi_2) & = & \Delta_v(\psi_1) \cdot \psi_2 \ \cup\ \mathrm{xt}\big(\psi_1 \cup \Delta_v(\psi_1)\big) \cdot \Delta_v(\psi_2) \\
\Delta_v(\psi_1 \cup \psi_2) & = & \Delta_v(\psi_1) \cup \Delta_v(\psi_2) \\
\Delta_v(\psi_1 \cap \psi_2) & = & \Delta_v(\psi_1) \cap \Delta_v(\psi_2) \\
\Delta_v(\langle\psi\rangle_I) & = & \langle \Delta_v(\psi)\rangle_I \\
\Delta_v(\psi^+) & = & \mathrm{xt}(\Delta_v(\psi))^* \cdot \Delta_v(\psi) \cdot \psi^*
\end{array}$$

Proof. By semantic definition, $\Delta_\nu(\varphi) = \{ \alpha\gamma w \mid \alpha u w \in [\![\varphi]\!] \text{ and } (u,\gamma) \in \textsc{rsub}(\nu)\}$ where $\textsc{rsub}(\nu) := \{ R(u) \mid u \in \textsc{sub}(\nu)\}$. We proceed by induction on the structure of φ. In the following we tend to use languages and expressions interchangeably, when in the interest of readability. Consider the cases:

- For $\varphi = Z$, $\alpha u w \notin [\![\varphi]\!]$ therefore $\Delta_\nu(\varphi) = \emptyset$.
- For $\varphi = p$: It needs that $\alpha = \epsilon$ and $u \in [\![p]\!]$. Then, $\alpha u w \in [\![p]\!]$ can be satisfied if either $w = \epsilon$ or $w \in [\![p]\!]$. By applying definitions, we get

$$\begin{aligned} \Delta_\nu(p) &= \{ \gamma \mid u \in [\![p]\!] \text{ and } (u,\gamma) \in \textsc{rsub}(\nu)\} \cup \\ &\quad\ \{ \gamma w \mid u \in [\![p]\!],\ w \in [\![p]\!] \text{ and } (u,\gamma) \in \textsc{rsub}(\nu)\} \\ &= Z \cup Z \cdot \{w \mid w \in [\![p]\!]\} \\ &= Z \cup Z \cdot p \end{aligned}$$

  where the expression Z is $\{(r,r') \mid t \leqslant r < r' \leqslant t'\}$. Hence, we have $\Delta_\nu(p) = Z \cup Z \cdot p$ if $u \in [\![p]\!]$, otherwise $\Delta_\nu(p) = \emptyset$. The condition $u \in [\![p]\!]$ can be easily checked by testing $a(p) = 1$.

- For $\varphi = \varphi_1 \cdot \varphi_2$: $\alpha u w \in [\![\varphi_1 \cdot \varphi_2]\!]$ should be satisfied. There are three possibilities to split $\alpha u w$ in dense time:
  - It can be split up into $\alpha u w_1 \in [\![\varphi_1]\!]$ and $w_2 \in [\![\varphi_2]\!]$.

$$\begin{aligned} \Delta_\nu(\varphi) &= \{\alpha\gamma w_1 w_2 \mid \alpha u w_1 \in [\![\varphi_1]\!],\ w_2 \in [\![\varphi_2]\!] \\ &\qquad \text{and } (u,\gamma) \in \textsc{rsub}(\nu)\} \\ &= \{\alpha\gamma w_1 \mid \alpha u w_1 \in [\![\varphi_1]\!] \\ &\qquad \text{and } (u,\gamma) \in \textsc{rsub}(\nu)\} \cdot \{w_2 \mid w_2 \in [\![\varphi_2]\!]\} \\ &= \Delta_\nu(\varphi_1) \cdot \varphi_2 \end{aligned}$$

  - It can be split up into $\alpha_1 \in [\![\varphi_1]\!]$ and $\alpha_2 u w \in [\![\varphi_2]\!]$.

$$\begin{aligned} \Delta_\nu(\varphi) &= \{\alpha_1\alpha_2\gamma w \mid \alpha_1 \in [\![\varphi_1]\!],\ \alpha_2 u w \in [\![\varphi_2]\!] \\ &\qquad \text{and } (u,\gamma) \in \textsc{rsub}(\nu)\} \\ &= \{\alpha_1 \mid \alpha_1 \in [\![\varphi_1]\!]\} \cdot \{\alpha_2\gamma w \mid \alpha_2 u w \in [\![\varphi_2]\!] \\ &\qquad \text{and } (u,\gamma) \in \textsc{rsub}(\nu)\} \\ &= xt(\varphi_1) \cdot \Delta_\nu(\varphi_2) \end{aligned}$$

  - It can be split up into $\alpha u_1 \in [\![\varphi_1]\!]$ and $u_2 w \in [\![\varphi_2]\!]$. For this case, it is required by definitions that φ_1 is a left-reduced expression and φ_2 is a pure expression. This is the most involved case requiring to split reducing signals.

$$\begin{aligned} \Delta_\nu(\varphi) &= \{\alpha\gamma_1\gamma_2 w \mid \alpha u_1 \in [\![\varphi_1]\!],\ u_2 w \in [\![\varphi_2]\!] \\ &\qquad \text{and } (u_1u_2,\gamma_1\gamma_2) \in \textsc{rsub}(\nu)\} \\ &= \{\alpha\gamma_1\gamma_2 w \mid \alpha u_1 \in [\![\varphi_1]\!],\ u_2 w \in [\![\varphi_2]\!],\ (u_1,\gamma_1) \in \textsc{rsub}(\nu), \\ &\qquad (u_2,\gamma_2) \in \textsc{rsub}(\nu) \text{ and } (u_1,\gamma_1) \text{ meets } (u_2,\gamma_2)\} \\ &= \{\alpha\gamma_1 \mid \alpha u_1 \in [\![\varphi_1]\!] \text{ and } (u_1,\gamma_1) \in \textsc{rsub}(\nu)\} \cdot \\ &\qquad \{\gamma_2 w \mid u_2 w \in [\![\varphi_2]\!] \text{ and } (u_2,\gamma_2) \in \textsc{rsub}(\nu)\} \\ &= xt(\Delta_\nu(\varphi_1)) \cdot \Delta_\nu(\varphi_2) \end{aligned}$$

Thus $\Delta_v(\varphi_1 \cdot \varphi_2)$ can be found by the disjunction of these three cases. Then, by rearranging the last two cases, we obtain the equality claimed in the theorem.

- For $\varphi = \psi^+$, we directly have

$$\begin{aligned}
\Delta_v(\psi^+) &= \Delta_v(\psi) \cup \Delta_v(\psi \cdot \psi^+) \\
&= \Delta_v(\psi) \cup \Delta_v(\psi) \cdot \psi^+ \cup \mathrm{xt}(\psi \cup \Delta_v(\psi)) \cdot \Delta_v(\psi^+) \\
&= \Delta_v(\psi) \cdot \psi^* \cup \mathrm{xt}(\Delta_v(\psi)) \cdot \Delta_v(\psi^+) \quad (\text{since } \mathrm{xt}(\psi) = \varnothing) \\
&= \mathrm{xt}(\Delta_v(\psi))^* \cdot \Delta_v(\psi) \cdot \psi^* \quad (\text{by Arden's lemma})
\end{aligned}$$

- Duration restriction and Boolean operations follow definitions straightforwardly.

□

Corollary 5.4. *The derivative $\Delta_v(\varphi)$ of a left-reduced timed regular expression φ with respect to a segment v of a timed behavior is a left-reduced timed regular expression.*

Proof. Theorem 5.3 shows that only finite number of regular operations is required to find the derivative and these equations satisfy requirements in Definition 5.6. □

Lemma 5.5. *The derivative $\Delta_w(\varphi)$ of a left-reduced timed regular expression φ with respect to a timed behavior $w = v_i \dots v_j$ with $j - i + 1$ segments is equivalent to the left reduction of φ with respect to the set of sub-behaviors of w beginning in $[t_{i-1}, t_i)$ and ending in $(t_{j-1}, t_j]$.*

$$\Delta_w(\varphi) = \bigcup_{u \in \mathrm{SUB}_i^j(w)} \delta_u([\![\varphi]\!])$$

where $\mathrm{SUB}_i^j(w) = \{w(t, t') \mid t \in [t_{i-1}, t_i), t' \in (t_{j-1}, t_j]$, *and* $t < t'\}$.

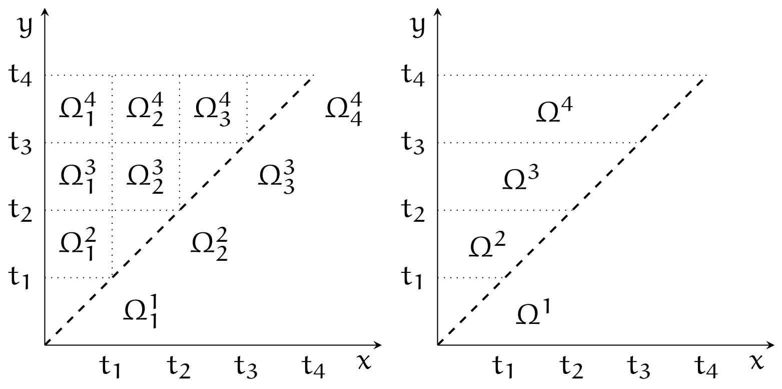
Proof. Using definitions we directly have

$$\begin{aligned}
\Delta_w(\varphi) &= \Delta_{v_j}(\Delta_{v_{j-1}}(\dots \Delta_{v_i}(\varphi)\dots)) \\
&= \delta_{\mathrm{SUB}(v_j)}(\delta_{\mathrm{SUB}(v_{j-1})}(\dots \delta_{\mathrm{SUB}(v_i)}([\![\varphi]\!])\dots)) \\
&= \delta_{\mathrm{SUB}(v_i)\cdot\mathrm{SUB}(v_{i+1})\cdot\dots\cdot\mathrm{SUB}(v_j)}([\![\varphi]\!]) \\
&= \delta_{\mathrm{SUB}_i^j(w)}([\![\varphi]\!])
\end{aligned}$$

□

5.3 ONLINE MATCHING USING DERIVATIVES

In this section, we solve the problem of online timed pattern matching for timed regular expressions by applying concepts and results introduced in previous sections. Our online matching procedure assumes the input signal w to be presented incrementally as follows. Let Σ be alphabet and $w = v_1 v_2 \dots v_n = (t_0, t_1, a_1), \dots, (t_{n-1}, t_n, a_n)$ be an n-variability timed behavior and we read a new segment v_k of w



y
t4
t3
t2
t1
t1 t2 t3 t4 x
Ω_1^4 Ω_2^4 Ω_3^4 Ω_4^4
Ω_1^3 Ω_2^3 Ω_3^3
Ω_1^2 Ω_2^2
Ω_1^1
Ω^4
Ω^3
Ω^2
Ω^1

Figure 20: Partitioning the set $\Omega(T)$ of all time periods.

incrementally for $k = 1, 2, \dots, n$. Before proceeding, let us allow to partition the set of all timed periods with respect to the behavior w with n segments for the sake of presentation as follows.

$$\Omega_i^j = \{(t, t') \in \Omega \mid t_{i-1} \leqslant t < t_i,\ t_{j-1} < t' \leqslant t_j \text{ and } t < t'\}$$

and we naturally define $\Omega^j = \bigcup_{i \leqslant j} \Omega_i^j$ where $1 \leqslant i, j \leqslant n$. In Figure 20 we visualize these sets on the xy-plane over a behavior of length 4. Note that equivalent timed behaviors with different representations (due to stuttering) would lead different partitions. Now recall that a valuation $V(\varphi)$ of an expression φ is a subset of Ω in a temporal structure $W = (\Omega, V)$. Therefore, we extend the notation to valuations such that $V_i^j(\varphi) = V(\varphi) \cap \Omega_i^j$ and $V^j(\varphi) = V(\varphi) \cap \Omega^j$. Notice that the valuation $V^j(\varphi)$ depends only on the prefix $v_1 \dots v_j$ of the behavior for a timed regular expression φ.

Then an online matching function Λ_φ for a timed regular expression φ is a sequential function that maps a timed behavior w to a sequence of timed relations such that the output $Z_k = V^k(\varphi)$ at a step k. In other words, the output Z_k contains exactly all matches whose endpoints in the interval $(t_{k-1}, t_k]$ with respect to a timed regular expression φ. Since $V^k(\varphi)$ does not depend on future segments of w, the function Λ_φ is causal. We formally state it by a definition.

Definition 5.7 (Matching Function). *Let $\Sigma = \mathbb{B}^{|P|}$ be alphabet over a set P of propositional variables. A matching function for a timed regular expression φ in a temporal structure $W = (\Omega, V)$ induced by a timed behavior $w = v_1, \dots, v_n$ of length n is a causal sequential function $\Lambda_\varphi : \Sigma^{(+)} \longrightarrow \mathcal{Z}^+$ such that*

$$\Lambda_\varphi(v_1 \dots v_n) = V^1(\varphi), \dots, V^n(\varphi) = Z_1, \dots, Z_n$$

Now we show how to realize matching functions for timed regular expressions using derivatives of timed regular expressions similar to the classical setting. For that, we first define the state of the online timed pattern matching at the step j as a left-reduced timed regular expression.

Definition 5.8 (The State of Online Matching). *Given a pure timed regular expression φ, the state ψ^j of the online timed pattern matching procedure after reading a prefix $v_1 \dots v_j$ of the input timed behavior is:*

$$\psi^j := \bigcup_{1 \leqslant i \leqslant j} \Delta_{v_i \dots v_j}(\varphi)$$

Then, starting with $\psi_0 = \varphi$, we update the state upon reading the next segment v_{j+1} by letting

$$\psi^{j+1} = \Delta_{v_{j+1}}(\psi^j) \cup \Delta_{v_{j+1}}(\varphi)$$

which is called the update equation. Now we show that the extraction of reduced timed behaviors from the $j-th$ state ψ^j provides the valuation $V^j(\varphi)$ at the step j.

Theorem 5.6. *Given a state ψ^j of an online matching procedure for the expression φ and a timed behavior w, the incremental valuation $V^j(\varphi)$ is found by the extraction of the state:*

$$V^j(\varphi) = \mathrm{xt}(\psi^j)$$

Proof. Following Definition 5.8 and Lemma 5.5 we know the state ψ^j represents a reduced language $L = \delta_{\text{SUB}^j(w)}(\varphi)$ of φ. Hence we can find the valuation $\mathcal{V}^j$ by extracting all reduced behaviors from the state ψ^j of online matching. □

Theorem 5.6 allows us to have a complete procedure for online timed pattern matching for given φ and an input signal $w = v_1 \dots v_n$ summarized below.

For $1 \leqslant j \leqslant n$ repeat:

1. Update the state of the matching ψ^j by deriving the previous state ψ^{j-1} with respect to v_j and adding a new derivation $\Delta_{v_j}(\varphi)$ to the state for matches starting in the segment j.

2. Extract ψ_j to get matches ending in the segment j.

Note that, since there are applications such that all the match set is not needed but only the matching of prefixes or the whole behavior, computing matches starting from segments other than the initial can be redundant. In this case, the algorithm can be easily modified to only output matches V_1^j starting in the first segment and ending in the segment j by simply letting the update equation $\psi^{j+1} = \Delta_{v_{j+1}}(\psi^j)$. Clearly this would be more efficient for those applications.

We now present an example run of online pattern matching for a timed regular expression $\varphi := \langle p \cdot q \rangle_{[4,7]}$ and input timed behavior $w := v_1 v_2 v_3 = (0, 3, (1, 0)), (3, 8, (1, 1)), (8, 10, (0, 1))$ over propositional variables p and q as illustrated in Figure 21 at the top. In Table 3, we depict the step-by-step computation of the valuation $V(\varphi)$ over the behavior w upon reading segments of w incrementally. For Step 1 the state ψ_1 is equal to the derivative of φ with respect to w_1

Letters			$(1,0)$		$(1,1)$	
Segments			1		2	
1	$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_1}]{}$	$\langle Z_1^1 \cdot q \rangle_I \cup \langle Z_1^1 \cdot p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_2}]{}$	$\langle Z_1^1 \cdot Z_2^2 \rangle_I \cup \langle Z_1^1 \cdot Z_2^2 \cdot q \rangle_I \cup \langle Z_1^1 \cdot Z_2^2 \cdot p \cdot q \rangle_I$	
2			$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_2}]{}$	$\langle Z_2^2 \rangle_I \cup \langle Z_2^2 \cdot q \rangle_I \cup \langle Z_2^2 \cdot p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_3}]{}$
3					$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{w_3}]{}$
Letters			$(0,1)$		$(0,0)$	
Segments			3		4	
1		$\xrightarrow[\Delta_{v_3}]{}$	$\langle Z_1^1 \cdot Z_2^2 \cdot Z_3^3 \rangle_I \cup \langle Z_1^1 \cdot Z_2^2 \cdot Z_3^3 \cdot q \rangle_I$	$\xrightarrow[\Delta_{v_4}]{}$	$\emptyset$	
2		$\xrightarrow[\Delta_{v_3}]{}$	$\langle Z_2^2 \cdot Z_3^3 \rangle_I \cup \langle Z_2^2 \cdot Z_3^3 \cdot q \rangle_I$	$\xrightarrow[\Delta_{v_4}]{}$	$\emptyset$	
3		$\xrightarrow[\Delta_{v_3}]{}$	$\emptyset$	$\xrightarrow[\Delta_{v_4}]{}$	$\emptyset$	
4		$\xrightarrow[\Delta_{v_3}]{}$	$\langle p \cdot q \rangle_I$	$\xrightarrow[\Delta_{v_4}]{}$	$\emptyset$	

Table 3: Timed pattern matching using derivatives for $w = v_1 v_2 v_3$ and $\varphi = \langle p \cdot q \rangle_I$. Entries (i,j) represent the derivative with respect to $v_i \dots v_j$. Reduced expressions, indicating matched segments, are shaded with gray. $I = [4,7]$.

such that $\psi_1 = \langle Z_1^1 \cdot q \rangle_{[4,7]} \cup \langle Z_1^1 \cdot p \cdot q \rangle_{[4,7]}$ where $Z_1^1 = \{(t,t') \mid 0 \leqslant t < 3,\ 0 < t \leqslant 3, \text{ and } t < t'\}$. The extraction $xt(\psi_1)$ returns the empty set therefore we do not have any matches ending in the first segment. Notice that time periods in Z_1^1 are partial matches that may be completed in the future. For Step 2 where $Z_2^2 = \{(t,t') \mid 4 \leqslant t < 7,\ 4 < t \leqslant 7, \text{ and } t < t'\}$ the extraction of the state ψ^2 is equal to $xt(\psi_2) = \langle Z_1^1 \cdot Z_2^2 \rangle_{[4,7]} \cup \langle Z_2^2 \rangle_{[4,7]} = \{(t,t') \mid 0 \leqslant t < 3,\ 4 < t \leqslant 8, \text{ and } 4 \leqslant t' - t \leqslant 7\} \cup \{(t,t') \mid 4 \leqslant t < 5,\ 7 < t \leqslant 8, \text{ and } 4 \leqslant t' - t \leqslant 5\}$. Similarly, for Step 3 where $Z_3^3 = \{(t,t') \mid 8 \leqslant t < 10,\ 8 < t \leqslant 10, \text{ and } t < t'\}$, the extraction of the state ψ_3 is equal to $xt(\psi_3) = \langle Z_1^1 \cdot Z_2^2 \cdot Z_3^3 \rangle_{[4,7]} \cup \langle Z_2^2 \cdot Z_3^3 \rangle_{[4,7]} = \{(t,t') \mid 1 \leqslant t < 3,\ 8 < t \leqslant 9, \text{ and } 5 \leqslant t' - t \leqslant 7\} \cup \{(t,t') \mid 4 \leqslant t < 6,\ 8 < t \leqslant 9, \text{ and } 4 \leqslant t' - t \leqslant 5\}$. For the final step, both propositions do not hold and the state ψ^4 is computed to be the empty set; therefore, we do not have any matches. In Figure 21 we illustrate corresponding segments (t,t') extracted in Steps 2 and 3 where solid regions show the actual outputs for the corresponding step.

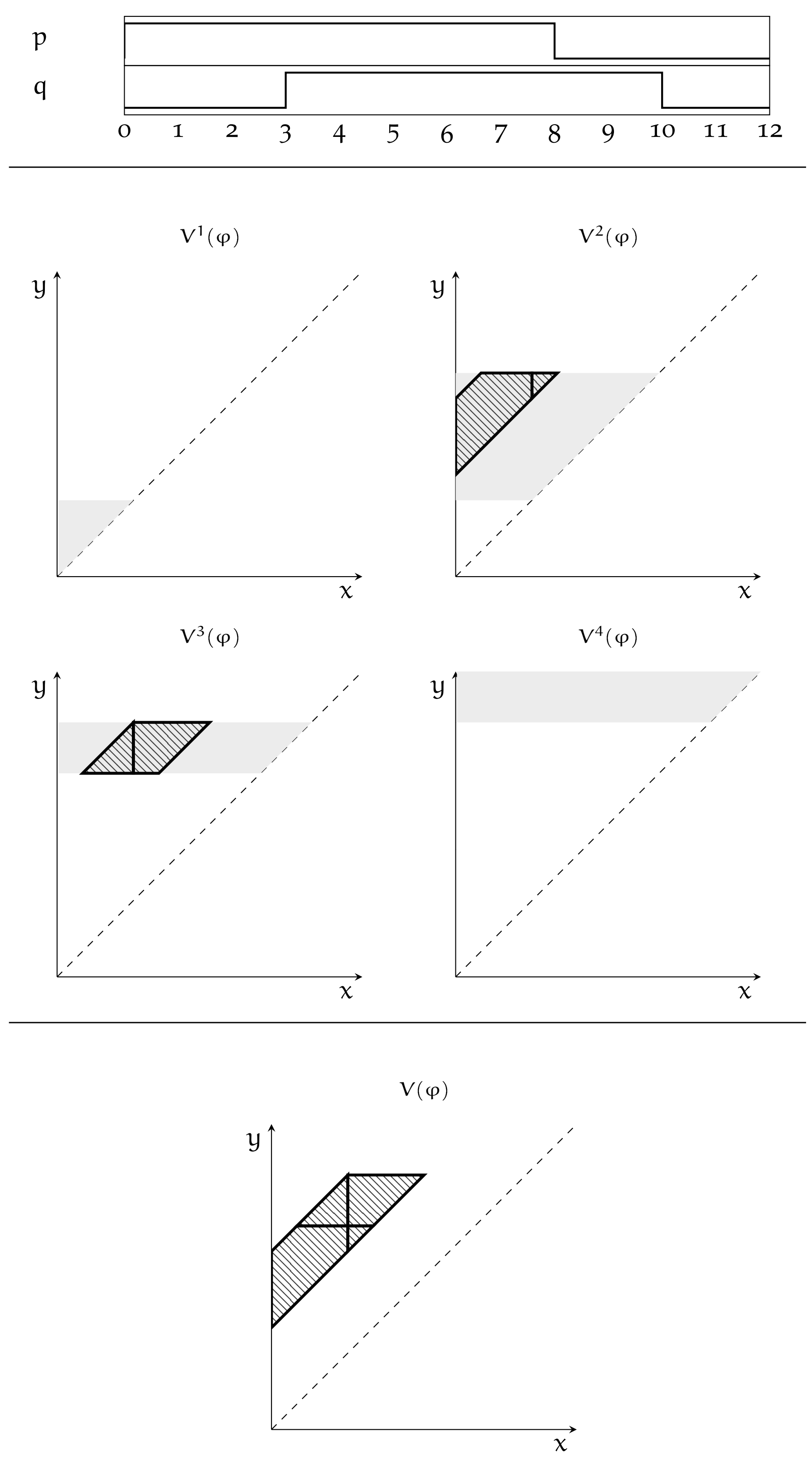


Figure 21: Illustration of online timed pattern matching presented in Table 3 with t and t′ denoting, respectively, the beginning and end of the match.

6 MONTRE & CASE STUDIES

The designer of a new kind of system
must participate fully in the implementation.

— Donald E. Knuth

In this chapter, we describe our timed pattern matching tool Montre and present three case studies in order to explore the usefulness of timed pattern matching in different application areas. We first explain the tool [108], which is intended to encapsulate timed pattern matching algorithms with a uniform easy-to-use interface. For pattern specification, we provide a rich set of operators in the syntax that can express many timed patterns. It is worth to note that some operations such as complementation and iteration are computationally more expensive than others. However, there are many cases that these operations are useful when used in a subtle manner and perform well due to the characteristics of the input behavior such as physical limitations or assumptions for the generator of the behavior. Indeed this is also true for all other operators; therefore, we focus more on typical cases than the worst-case scenarios. Then, we provide some performance results over the synthetic data generated according to these considerations before case studies.

The first case study comes from our foremost domain of interest, that is, runtime verification (monitoring) of real-time systems. We present a task to measure some quantitative properties of communication (bus) protocol employed by the automotive industry. To this end, timed regular expressions and metric compass logic are used to specify time windows over which (external) measurement operations are applied. Then we run timed pattern matching algorithms to find all such time windows and apply the corresponding measurement straightforwardly over them.

For the second case study, we present an example from the domain of sports analytics where we employ timed pattern matching to find all sprints of a soccer player. For that we preprocess the raw data of player positions tracked by other techniques, namely computer vision, to compute timed speed and acceleration behavior. We reduce noise, interpolate missing data points, differentiate, etc. using existing data analysis tools that implement algorithms for such tasks. Timed pattern matching comes after these preprocessing tasks and before post-processing tasks such as visualization. Therefore, the main objective of this case study is to emphasize the need for collaboration between timed pattern matching and other techniques to analyze sequential data and time series. For data scientists, timed pattern matching would be a complementary technique to analyze time series data that works in the pipeline with other techniques. For our part, it means that we delegate a plethora of necessary things to do in order to have a complete application. It illustrates our choices when designing our tool.

In the third case study, we explore opportunities to use timed patterns to encode knowledge about high-level temporal activities and extract information from temporal datasets using timed pattern matching. As virtually all high-level activities can be decomposed into more primitive activities, it is natural to specify timed patterns over an alphabet that consists of such primitive activities. We use the OPPORTUNITY activity recognition dataset, which provides a rich set of primitive human activities and interactions from an early morning scenario. First, we write more complex patterns using primitive activities from the data set and match these patterns over tracked behaviors. Second, we build a simple statistical model by matching and counting temporal occurrences detected by timed pattern matching. It is inspired by n-gram models, which are extensively used for natural language processing tasks, and the concurrent nature of time makes it different.

In all three case studies, we need to use a variety of other data analysis techniques. Since complete applications require many separate components from various domains of research, the collaboration between tools and techniques always needs to be maintained. We consider this fact in the design of our tool MONTRE and we believe timed pattern matching would be a valuable technique among all other temporal data analysis techniques.

6.1 TOOL DESCRIPTION

The tool MONTRE essentially incorporates online and offline timed pattern matching algorithms extended with some practical features such as anchors and the Boolean layer. It takes a timed behavior and a timed pattern specified using timed regular and metric compass operators as inputs, and produces a finite set of two dimensional zones representing the (possibly uncountable) set of segments that match the pattern. The syntax of all boolean, regular, and compass operations in MONTRE is given in Table 4. MONTRE provides a standard text-based interface for easy integration with other tasks such as data preparations and visualization as we consider them necessary but outside the scope of MONTRE. Figure 22 illustrates the work flow and extent of MONTRE, and we give details for each component in the following.

Implementation. MONTRE is a command line program[1] that uses structured text files for input/output specification. When invoked, MONTRE parses the timed pattern passed as an argument and starts to read the input file. According to flags set by the user, MONTRE would run in either online or offline mode. For online mode, it is possible that the input can be given interactively using the command line or directed from another process via a pipe. At its core, MON-

1 Available at http://github.com/doganulus/montre

Table 4: MONTRE timed pattern syntax.

Construct	Description
`p`	A propositional variable.
`!P`	Boolean NOT operation on `P`.
`P || Q`	Boolean OR operation on `P` and `Q`.
`P && Q`	Boolean AND operator on `P` and `Q`.
`P`	occurs on (t, t') if `P` holds from t to t' continuously.
`<:P`	occurs on a time period (t, t') if `P` occurs on (t, t') and there exists a rising edge for `P` at t.
`P:>`	occurs on a time period (t, t') if `P` occurs on (t, t') and there exists a falling edge for `P` at t'.
`<:P:>`	occurs on a time period (t, t') if `P` occurs on (t, t') and there exists a rising edge for `P` at t as well as a falling edge for P at t'.
`~E`	occurs on a time period (t, t') if `E` occurs on (t, t'') and `F` occurs on (t'', t') for $t \leqslant t'' \leqslant t'$.
`E|F`	occurs on a time period (t, t') if either `E` or `F` occurs on (t, t').
`E&F`	occurs on a time period (t, t') if `E` and `F` occur on (t, t') concurrently.
`E;F`	occurs on a time period (t, t') if `E` occurs on (t, t'') and `F` occurs on (t'', t') for $t \leqslant t'' \leqslant t'$.
`E*`	Zero-or-more repetition of `E`.
`E+`	One-or-more repetition of `E`.
`E%(m,n)`	occurs on a time period (t, t') if `E` occurs on (t, t') and the duration of the occurrence is in the specified range such that `m` $\leqslant t' - t \leqslant$ `n`.
`<X>%(m,n) E`	occurs on a time period (t, t') if `E` occurs on (t, t'') and `F` occurs on (t'', t') for $t \leqslant t'' \leqslant t'$.
`[X]%(m,n) E`	occurs on a time period (t, t') if `E` occurs on (t, t'') and `F` occurs on (t'', t') for $t \leqslant t'' \leqslant t'$.

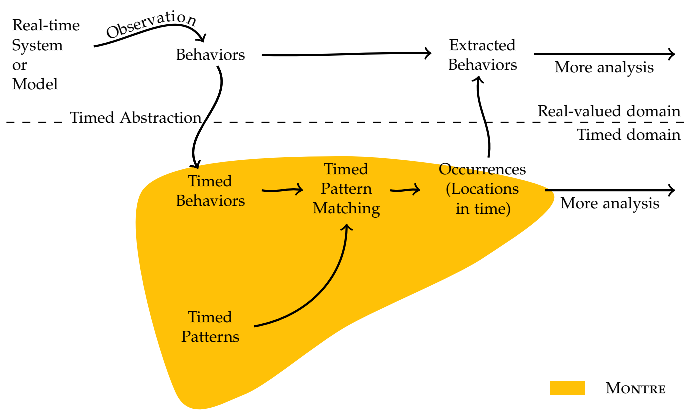


Figure 22: The work flow and extent of the monitoring tool MONTRE

TRE contains our efficient implementation of the algebra of timed relations — written in C++ and compiled as a shared library named `libmontre`. It is called dynamically by the top-level online and offline timed pattern matching algorithms. In the implementation, we use an integer-valued time model where all time values are represented by integers for efficiency and accuracy reasons. For the majority of applications, the integers give us sufficient precision and range; and a proper scaling can be found.

We implement timed pattern matching algorithms in PURELANG[2], a functional programming language based on term rewriting with a support for native code compilation and native calls to dynamic libraries. The offline algorithm [109] is a recursive computation over the syntax tree of the expression; therefore, the role of Pure's rewriting engine is minimal. Offline mode is invoked by the option `-b` or `-offline` and requires a valid pattern (in single quotes) and a file as arguments as follows.

```
montre -b '(p;q)%(3,4)' 'my_timed_beh.txt'
```

The input file should be structured such that each line contains a symbolic interval formed by a duration value and a string of propositions as below. For the empty symbol set the double-dash (`--`) can be used.

```
2 pq
2 p
1 pq
3 --
```

Listing 1: An input file for MONTRE

After execution the output file includes a set of zones where 6 zone inequalities are defined by 6 numbers ($b\ b'\ e\ e'\ d\ d'$) and a 6-bit vector denotes the strictness of the corresponding inequality, respectively. It is interpreted such that an inequality is strict if the corresponding bit value is `0`, and not strict otherwise. An example output file is given in Listing 2.

2 Available at http://purelang.bitbucket.io

```
(0 1 3 4 3 4) 100111
(5 6 8 9 3 4) 100111
```

Listing 2: An output file by the option `--output-type=zone`

Zones are the default output of montre that may be denoted explicitly by an option `-output-type=zone`. Besides, Montre provides an option `-output-type=end` to project zones over the end-axis. Then the output file contains a Boolean timed behavior that indicates the existence of the end of a match. It is useful when you would like to link timed regular expressions with point-based temporal logics similar to the suffix implication of psl and sva. An example output file for this option is given in Listing 3.

```
2 1
1 0
1 1
```

Listing 3: An output file by the option `--output-type=end`

For the online algorithm [110], built upon derivatives of regular expressions [88, 100], we extensively use the rewriting functionality when deriving an expression with respect to a newly observed segment. The online mode is invoked by the option `-i` or `-online`.

```
montre -i '(p;q)%(3,4)' 'my_timed_beh.txt'
```

For the online mode, the filename is optional and Montre will expect a duration value and a string of propositions from standard input if no file is provided. Otherwise it will read the file line by line in one pass.

The worst case complexity is polynomial in the size of input behavior and expression for the offline approach. For the online approach it is polynomial in the size of the behavior and exponential in the expression. In practice, however, we realistically assume patterns to be much shorter than behaviors and somewhat sparse in them. Then we expect a linear-time performance in the size of input behavior for both algorithms. Under these assumptions, Montre can process timed behaviors with a size of 1M segments in a few seconds (offline) and a few hundred seconds (online).

Synthetic Performance Tests. We perform our experiments on a 3.3GHz machine for a set of test patterns that are specified by timed regular regular expressions and metric compass logic. Input behaviors are generated automatically by repetitions of instances of corresponding patterns; thus the number of instances (zones) matched is linear to the size of the behavior. We expect (and have observed from case studies) that this number in practice is much smaller than in these synthetically generated examples. We depict performance results of the online procedure in comparison with the offline procedure in [109] in Table 5. We use the gnu `time -v` facility to measure

	Offline Algorithm		
	Input Size		
Test Patterns	100K	500K	1M
p	0.06/17	0.27/24	0.51/33
$p \cdot q$	0.08/21	0.42/46	0.74/77
$\langle p \cdot q \cdot \langle p \cdot q \cdot p \rangle_I \cdot q \cdot p \rangle_J$	0.23/28	1.09/77	2.14/140
$\langle p \cdot q \rangle_I \cdot r \ \cap \ p \cdot \langle q \cdot r \rangle_J$	0.13/23	0.50/51	1.00/86
$p \cdot (q \cdot r)^*$	0.11/20	0.49/37	0.96/60
$\overline{p}$	0.18/12	0.95/45	1.88/92
$\diamondright_I\, p$	0.07/16	0.29/65	0.66/163
$\boxright_I\, p$	0.49/23	1.98/100	3.92/163
$\diamondleft_I \diamondtop_J\, p$	0.08/20	0.32/37	0.96/60
$\diamondleft \diamondtop (\boxright p \cdot q)$	0.40/31	1.98/143	3.93/268
$\diamondleft \diamondtop (\boxright p \cdot q) \cap \diamondright_I\, q$	0.43/38	2.17/179	4.30/304

	Online Algorithm		
	Input Size		
Test Patterns	100K	500K	1M
p	6.74/14	29.16/14	57.87/14
$p \cdot q$	8.74/14	42.55/14	81.67/14
$\langle p \cdot q \cdot \langle p \cdot q \cdot p \rangle_I \cdot q \cdot p \rangle_J$	28.07/14	130.96/14	270.45/14
$\langle p \cdot q \rangle_I \cdot r \ \cap \ p \cdot \langle q \cdot r \rangle_J$	15.09/15	75.19/15	148.18/15
$p \cdot (q \cdot r)^+$	11.53/15	52.87/15	110.58/15

Table 5: Execution times/Memory usage (in seconds/megabytes)

execution times (user CPU time) and memory usage (maximum resident set size). For typical cases, experiments suggest a linear time performance with respect to the number of segments in the input for both algorithms. Although the online procedure runs slower than the offline procedure, it requires less memory and the memory usage, as expected, does not depend on the input size.

6.2 MEASURING A BUS PROTOCOL

Distributed Systems Interface (DSI3) is a flexible and powerful bus protocol developed for the automotive industry and, in particular, targeting airbag systems. The protocol is designed for powering and communicating with remotely located slave devices in a self-contained automotive network operating over a 2-wire twisted pair. The slave devices are typically acceleration sensors responding with

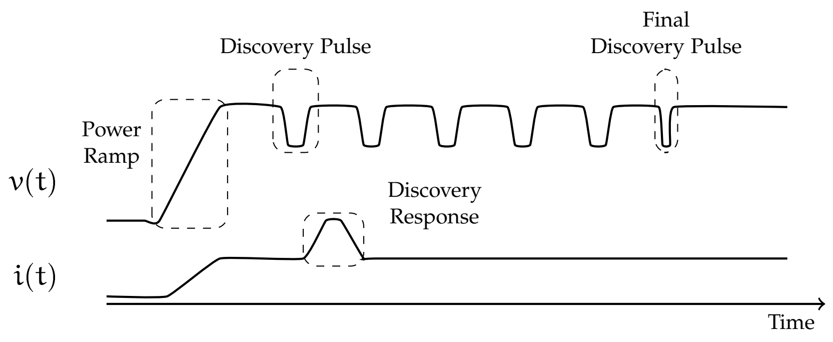


Figure 23: DSI3 discovery mode.

raw-g data or pressure sensors responding with relative pressure data. The controller interacts with the sensor devices over the wire via analog signals where the information is represented by time-varying voltage and current values. The correctness or performance of such networks is often evaluated by measuring some quantitative properties over the specific segments (measurement windows) of their (actual or simulated) behaviors. In this case study, we consider two modes of the DSI3 protocol (discovery and command & response modes) that requires complex interactions with timing and performance considerations [43]. We then define timed patterns for such behaviors and perform pattern matching to find measurement windows over which measurements are to be applied.

The discovery mode is the initialization phase initiated by the controller to automatically determine the position of devices and assign a unique physical address to them in the (serial daisy-chain) network. Typically this is done at the time of network formation after power-up. Therefore, following a voltage ramp, the controller transmit a predetermined number of (negative) voltage pulses over the wire. Devices respond by transmitting current pulses and determine their position in the network by sensing and counting such current pulses. In Figure 23, we illustrate the discovery mode in the DSI3 protocol and provides a high-level overview of its ordering and timing requirements such as the minimal time between the power turned on and the first discovery pulse, the maximal duration of the discovery mode, and the expected time between two consecutive discovery pulses.

The power delivery functionality of the command and response mode is illustrated in Figure 24. During this mode, the controller

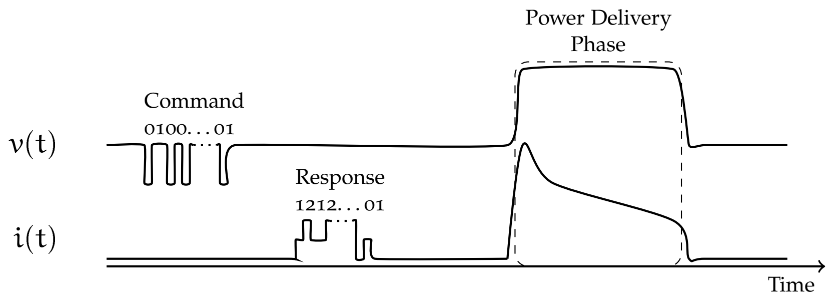


Figure 24: DSI3 power delivery functionality.

sends a command to the sensor as a series of pulses (a pulse train) on the voltage line, which transmits its response by another pulse train on the current line. For power-demanding applications, the command and response pairs are followed by a power pulse, which goes above V_{high}. Such power pulses enable the sensors to load their capacitors used for powering the internal operations.

In the following, we study two specific measures for the DSI3 protocol: (1) the time between consecutive discovery pulses; and (2) the amount of energy transmitted to the sensor through power pulses.

Time between consecutive discovery pulses. In order to measure the time between consecutive discovery pulses, we start by describing discovery pulses. For that, we first partition the continuous voltage axis using three predicates such that (1) `VHigh` is true when the voltage is above 7.8 Volts, (2) `VMiddle` between 7.8 and 4.92 Volts, and (3) `VLow` below 4.92 Volts as given in Table 6. We use these predicates to specify a timed pattern φ_{dp} for discovery pulses such that

$$\varphi_{dp} = \langle \blacktriangleleft \texttt{VMiddle} \cdot \texttt{VLow} \cdot \texttt{VMiddle} \blacktriangleright \rangle_{[12,20]} \cap \Diamond_{\geqslant 5} \texttt{VHigh} \cap \Diamond_{\geqslant 5} \texttt{VHigh}$$

where time units are in microseconds. Then, a discovery pulse is essentially expressed as a sequential composition of `VMiddle`, `VLow`, and `VMiddle` periods. We use begin and end anchor operators over the `VMiddle` predicates since a pulse is usually considered to be started/ended at the time when thresholds are crossed. Further, we specify the pulse width (duration) constraint according to the DSI3 standard and refine it by requiring `VHigh` holds at least for 5µs at preceding and trailing time periods.

The measurement window for this property starts at the beginning of a discovery pulse and ends at the beginning of the next one. It consists of a discovery pulse φ_{dp}, followed by a period of `VHigh`, and terminating when the voltage leaves v_h. This description is not sufficient to capture the property as we also need to ensure that this segment is indeed followed by another discovery pulse. Hence, we add an additional constraint $\Diamond \varphi_{dp}$ that specifies this requirement using MCL. The final pattern is formalized as follows:

$$\varphi_{distpp} = \varphi_{dp} \cdot \texttt{VHigh} \ \cap \ \Diamond \varphi_{dp}$$

For our experiment, we apply a scenario where the discovery mode is activated on/off 100 times in sequence. We generate signals from an abstract model using randomized timing parameters drawn from

Table 6: Voltage categories for discovery pulses.

Label	Voltage thresholds (V)
`VHigh`	> 7.8
`VMiddle`	4.92 - 7.8
`VLow`	< 4.92

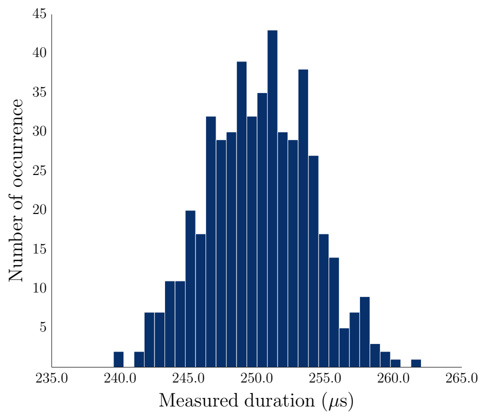


Figure 25: Distribution of the time between consecutive discovery pulses.

a normal distribution. Then we match the pattern $\varphi_{\mathtt{distpp}}$ and obtain a finite number of matches and consequently their duration. Notice that the deliberate use of anchors from both sides would guarantee the finiteness of number of matches. In Figure 25, we depict the measured duration values using a histogram. As expected, the distribution of the times between two discovery pulses follows a normal distribution according to the timing parameters used to generate it at the first place.

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 response 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 a power pulse rising to potential $V_{\mathtt{idle}}$ in which a large current can be drawn by the sensor. We first characterize the power pulse. It occurs when the voltage goes from below V_{pwr} to above V_{idle}, and then back below V_{pwr}. The three regions of interest are specified with the following predicates. Hence, the pattern φ_{pwr} specifying a power pulse is expressed as

$$\varphi_{pwr} = \langle \blacktriangleleft \mathtt{VPwr1090} \blacktriangleright \rangle_{[1,8]} \cdot \mathtt{VHigh} \cdot \langle \blacktriangleleft \mathtt{VPwr1090} \blacktriangleright \rangle_{[1,8]} \cap \Diamond \mathtt{VLow} \cap \Diamond \mathtt{VLow}$$

Given the voltage and current signals, $v(t)$ and $i(t)$ on the wire, the energy transfered to the device is given by the area under the signal $v \times i$ between the start and end of power pulse. We then measure energy by evaluating the integral $\int_t^{t'} v(t)i(t)dt$ for each power pulse

Table 7: Voltage categories for power pulses.

Label	Voltage thresholds (V)
VPwrHigh	> 6.5
VPwr1090	4.4 - 8.1
VPwrLow	< 6.5

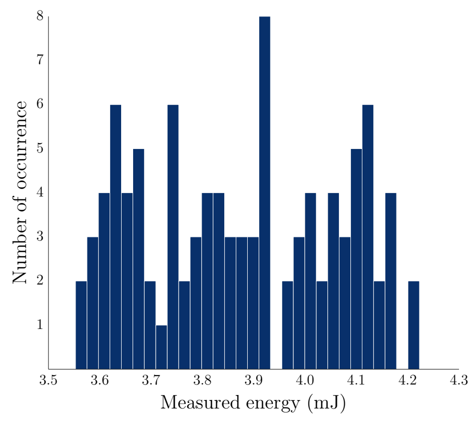


Figure 26: Distribution of the energy transmitted per power pulse.

identified at the time period (t, t'). In Figure 26, we depict measurement results using histograms. We note that the final integral operations are performed by standard numerical integration routines (in particular, we use the trapezoid method in this example over equally sampled voltage and current signals). We consider such operations to be the post-processing or analysis of the segments obtained by timed pattern matching.

6.3 DETECTING SPRINTS

We present a case study on a dataset obtained by tracking positions of players in a real soccer match. In this example, we find all sprints performed by a single player where a sprint is formally specified by a timed regular expression over speed and acceleration behaviors. The data are obtained by a computer vision algorithm with a frame rate of 10 Hz so we have an xy-coordinate for each player on the field at every 100 milliseconds. Therefore we use milliseconds as our base time unit for behaviors and expressions.

In order to specify a pattern for sprints, we need to address two issues in order: (1) how to categorize continuous speed and acceleration axes, and (2) which composition of these categories defines a sprinting effort best. Clearly, there are no universal answers for these questions so we rely on the study [39] in the following. First, we partition speed and acceleration axes into four categories (near-zero, low, medium, and high) as shown in Table 8. For example, a period of medium speed, denoted by `SpeedMedium`, means the speed value resides between 3.7 and 6 m/s during the period.

Often a sprint effort is characterized by any movement above a certain speed threshold for a limited time. This gives us our first sprint pattern such that a period of high speed between 1-10 seconds, formally written as follows:

$$\langle \blacktriangleleft \mathtt{SpeedHigh} \blacktriangleright \rangle_{[1000,10000]} \qquad \text{(P1)}$$

Above we use anchor operators from both sides on the proposition `SpeedHigh` to obtain only maximal periods that satisfy `SpeedHigh`; oth-

Table 8: Speed and acceleration categories for sprinting [39].

Label	Speed thresholds ($m \cdot s^{-1}$)
SpeedHigh	> 6.0
SpeedMedium	3.7 - 6
SpeedLow	2 - 3.7
SpeedNearZero	0 - 2
Label	**Acceleration thresholds ($m \cdot s^{-2}$)**
AccHigh	>1.60
AccMedium	1.17 - 1.60
AccLow	0.57 - 1.17
AccNearZero	-0.57 - 0.57

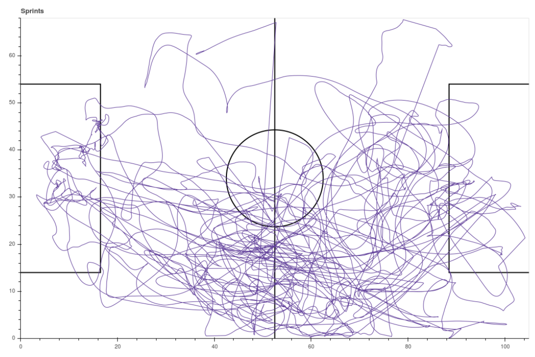
erwise, any sub-period satisfies the pattern as well. The duration restriction operator specifies that the duration is restricted to be in 1000 and 10000 milliseconds. Alternatively we may want to find other efforts starting with high acceleration but not reaching top speeds necessarily. This gives us our second sprint pattern such that a period of high acceleration followed by a period of medium or high speed between 1-10 seconds, formally written as follows:

$$\blacktriangleleft\texttt{AccHigh} \cdot \langle \blacktriangleleft(\texttt{SpeedMedium} \vee \texttt{SpeedHigh})\blacktriangleright \rangle_{[1000,10000]} \qquad \text{(P2)}$$

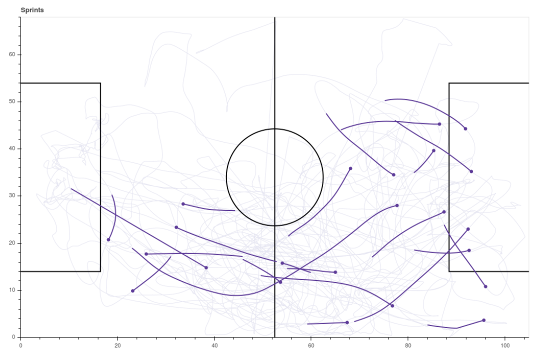
Notice that we do not use the end anchor on the predicate AccHigh. This allows a medium or high speed period to overlap with a high acceleration period as it is usually the case that they are concurrent. Writing an equivalent pattern using classical regular expressions over a product alphabet would be a very tedious task partly due to a requirement to handle such interleavings explicitly (and the lack of timing constraints). For TRES, all propositions are considered to be concurrent by definition, which results in concise and intuitive expressions. Finally, we give a third pattern to find rather short but intense sprints such that

$$<: (\texttt{AccMediumHigh} \vee \texttt{AccHigh}) \cdot \langle \blacktriangleleft\texttt{SpeedHigh}\blacktriangleright \rangle_{[1000,2000]} \qquad \text{(P3)}$$

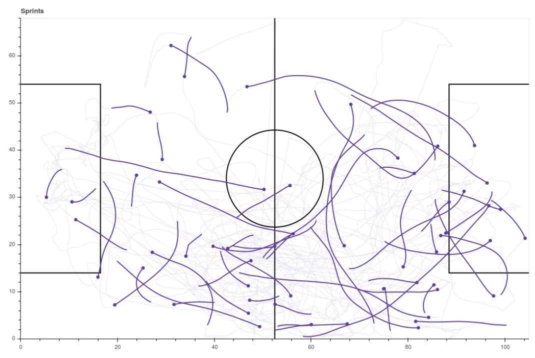
Then, we visualize all sprints detected by MONTRE for patterns P1-P3 in Figure 27 over the behavior of a single player during one half of the game (45 min.) containing 27K data points that reduces to timed behaviors of 5K segments after pre-processing. Note that we used Python to prepare data and visualize results.



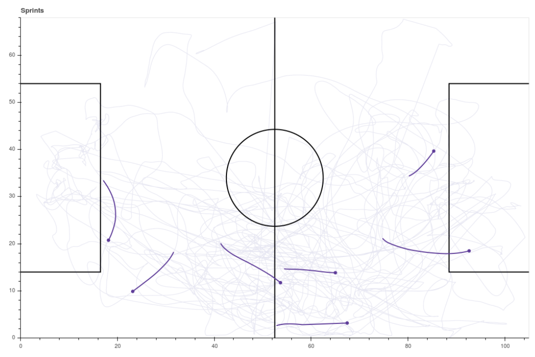
(a) Entire movement



(b) P1



(c) P2



(d) P3

Figure 27: The trajectory of a soccer player for 45 minutes on the field, and his sprinting periods detected by Montre for patterns P1-P3.

6.4 DETECTING DAILY ACTIVITIES

In this section, we present a case study that aims to specify and match high level activity patterns composed out of lower level activity primitives. To this end, we use a dataset that contains a rich set of activity primitives to be explained in more detail in the following. Our main goal is to explore the use of timed regular expressions and metric compass logic to encode (domain-specific) temporal knowledge and propose timed pattern matching as yet another method for extracting information from temporal datasets.

Creating new entities from meaningful combinations of basic entities is a fundamental task for any kind of reasoning. This allows to generalize situations, increase the level of abstraction, and reduce the complexity by forgetting unimportant details for higher levels. The process of *lexical analysis* in compilers [1] is perhaps the cleanest (and most successful) example we can give to illustrate these rather vague concepts. As for lexing, textual patterns (usually specified by regular expressions) are used to associate segments of the source code with meaningful labels (digit, integer, float, number etc.) called tokens. For example, a *digit* is a textual character from 0 to 9, an *integer* is a sequence of digits possibly preceded by a character of minus or plus, a *floating point number* is an integer followed by a dot character and a sequence of digits, a *number* is either an integer or a floating point number, and so on. The upper level (of parsing) then does not need to care about individual digits explicitly but only numbers when recognizing an arithmetic expression, which is a high level textual pattern that consists of numbers, variables, arithmetic operators, and parentheses.

Similar motivations and ideas can be found in the area of natural language processing (NLP) [58], in particular the topic of *part-of-speech tagging* and *named entity recognition*. For those, parts of text or speech are tagged by rule-based (or statistical) techniques with linguistic concepts such as verb, noun, or adverb. Besides some nouns may denote names of persons/organization/venues or some adverbs may be associated with certain emotions in some contexts, which are all indicates a increasing level of abstraction in reasoning. For all these applications, pattern matching (and regular expressions) over texts is considered a basic tool before more complex analyses [8, 80]. Since we have developed timed pattern matching in this thesis, we see a promising direction in exploring the applicability of these ideas for temporal datasets using timed pattern matching.

For this case study, we use the OPPORTUNITY activity recognition dataset[3], which is a dataset devised to benchmark human activity recognition algorithms [96]. We choose the topic of daily activities partly because we (and everyone else) have a knowledge on the topic and partly because the dataset provides a rich set of action and state labels on the time axis and clearly separates the levels of abstraction. Thus we can write meaningful timed patterns by hand using

3 At https://archive.ics.uci.edu/ml/datasets/opportunity+activity+recognition

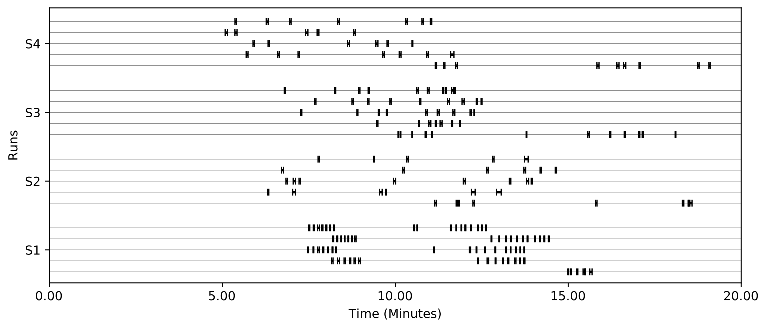


Figure 28: Sipping periods from the dataset.

our knowledge about daily activities, related verbs, and household objects.

The activity recognition environment and scenario is designed to generate many activity primitives in a realistic manner. The (numerical) data is obtained from sensors placed over objects as well as human subjects. In addition to sensory data, the dataset contains manually annotated tracks of primitive actions and states. These behaviors are uniformly sampled and the sampling rate is approximately 33KHz. Accordingly we use milliseconds as the base time unit. Subjects are told to follow the high-level scenario of ordinary morning activities such as getting up, having breakfast, and cleaning but they can freely interpret lower level activities such as holding cups or using different hands. This intends to provide complex, interleaved and hierarchical natural activities, with a particularly large number of primitive activities. Four subjects, called `S1`, `S2`, `S3`, and `S4`, repeat this scenario 5 times and we then have 20 runs of activity tracking where each run is completed in 20 minutes approximately. In the following figures such as Figure 28, we illustrate matching periods on parallel time axes from these total 20 runs for the corresponding patterns. In those figures, the gap between different subjects is larger than the vertical blank space between runs of the same subject.

In this case study, we do not use real-valued sensory data provided by the dataset but work directly over tracks of primitive actions and states. For example, the primitive action predicate `sip_left_hand` holds at a time point t if the corresponding subject is performing a sipping action with their left hand at t. Similarly we have the predicates `sip_right_hand` and several others in the dataset. For the rest, we abstract the distinction between left and right hands by simply defining `sip` pattern:

$$\texttt{sip} \quad = \quad \texttt{sip\_left\_hand} \vee \texttt{sip\_right\_hand}$$

Now let us look into the sipping periods depicted in Figure 28 of each subject observed during their (five) runs in order to have an idea about the nature of our data. At a first glance, we can see that the subject `S1` sips his/her beverage more regularly than other subjects. Our intention is to define higher level actions using a `drink` pattern telling that a drinking action is a repeated action of sipping

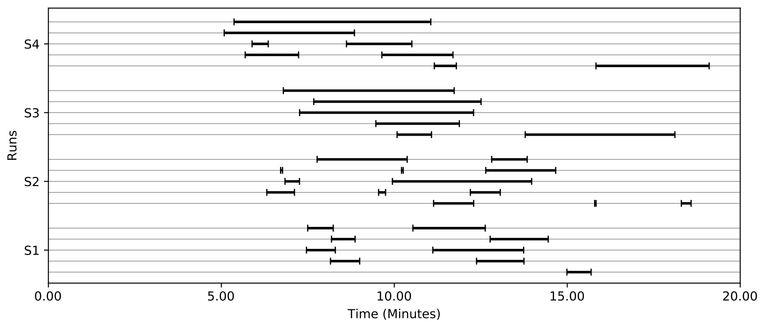


Figure 29: Drinking periods that match the pattern `drink`.

one or more times. We also require the duration between two sips (the duration of non-sipping) is two minutes at maximum. Therefore, such duration constraints are necessary since we do not want a sip in the morning and a sip in the evening to qualify for a drinking action from the morning till the evening. And finally we acknowledge that this particular duration bound comes from our personal experience empirically but such values may of course be subject to further statistical analyses in general. Then the resulting pattern is given as a timed regular expression as follows.

$$\texttt{drink} = \texttt{sip} \cdot (\langle \neg\texttt{sip} \rangle_{[0,2\,\text{min}]} \cdot \texttt{sip})^*$$

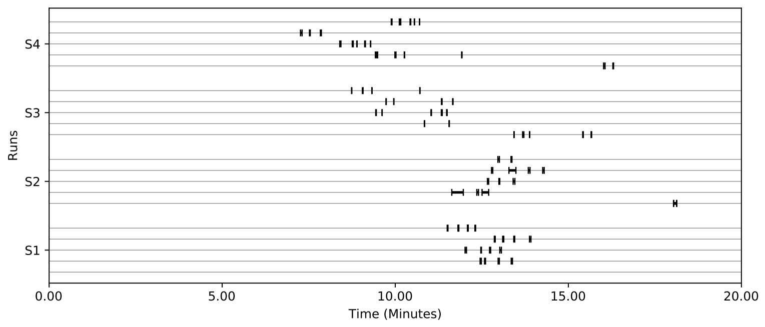
We depict drinking periods matched over the dataset according to the pattern `drink` in Figure 29.

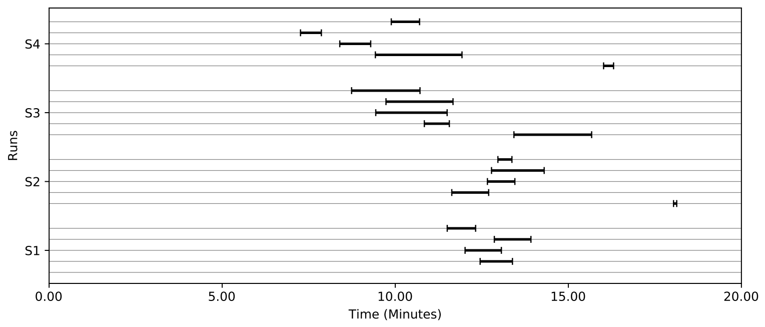
We continue our experiment by defining a similar pattern for eating action that consists of taking bites (from a sandwich). As the dataset provides us the predicate `sip_left_hand` and `sip_right_hand`, we define following patterns for eating action.

$$\begin{aligned} \texttt{bite} &= \texttt{bite\_left\_hand} \vee \texttt{bite\_right\_hand} \\ \texttt{eat} &= \texttt{bite} \cdot (\langle \neg\texttt{bite} \rangle_{[0,5\,\text{min}]} \cdot \texttt{bite})^* \end{aligned}$$

We depict biting and eating periods matched over the dataset according to the patterns `bite` and `eat` in Figure 30, respectively. Further we can measure the duration of drinking and eating periods similar to the case study in Section 6.2 and we depict the results in Figure 31.

Now we switch our attention to some basic probabilistic models called n-gram models, extensively used in natural language processing [58]. An n-gram is a sequence of n words and these models are used to assign a probability $P(w|h)$ for a word w given some history h of length $n-1$. In particular, we focus on the simplest case, bigram models, where $n = 2$ and a bigram model tells us the probability $P(w_i|w_{i-1})$ of a word w_i depending on the previous word w_{i-1}. The most basic method to estimate these probabilities is to count the number of occurrences in a textual dataset called corpus. More precisely, the probability $P(w_i|w_{i-1})$ can be estimated from a (large) dataset by dividing the number $\#(w_i, w_{i-1})$ of occurrences of the sequence



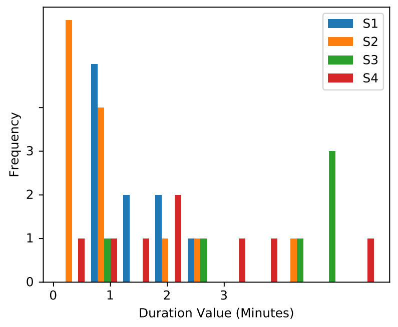


(b) Eating

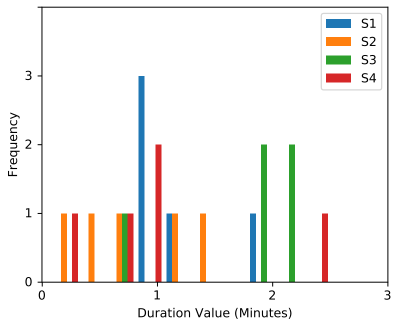
Figure 30: Biting and eating periods that match the pattern `bite` and `eat`, respectively.

(w_i, w_{i-1}) to the number $\#(w_{i-1})$ of occurrences of the word w_{i-1} in a corpus and this ratio is called relative frequency. Although simple, n-gram models (with some adjustments) are very effective for NLP tasks. Since regular expression matching is used to count the number of occurrences of words in NLP, we would like to experiment with the idea –counting patterns and estimating probabilities– over our temporal dataset using timed pattern matching in the following.

Now, as in bigrams, we would like to compute the probability $\mathcal{P}_d(q \mid p)$, that is the likelihood of the occurrence of q in d time units given the occurrence of p where $p, q \in P$ are atomic proposi-



(a) Drinking duration



(b) `Eating duration`

Figure 31: Distribution of matching periods.

tions. Notice that we here replace the concept of the next word with the concept of the occurrence of the proposition in d time units for this particular case. Then we estimate the probability $\mathcal{P}_d(q \mid p)$ from the relative frequency as follows.

$$\mathcal{P}_d(q \mid p) = \frac{\#(\blacktriangleleft p \cdot \langle \neg q \rangle_{[0,d]} \cdot q \blacktriangleright)}{\#(\blacktriangleleft p \blacktriangleright)}$$

where $\#(\varphi)$ is defined to be the total count of matches obtained from the whole dataset. We say that the timed pattern $\blacktriangleleft p \cdot \langle \neg q \rangle_{[0,d]} \cdot q \blacktriangleright$ in the numerator is the base pattern for our bigram-like model. We apply this model to six atomic propositions from OPPORTUNITY dataset. After abstracting left and right hand distinction similar to `sip` and `bite`, these propositions are `cut`, `spread`, `cheese`, `salami`, `knife_cheese`, and `knife_salami` that denote some actions of subjects as well as their interaction with some objects during their breakfast. Before applying timed pattern matching to estimate probabilities in our model, we normally expect that subjects spread their cheese with the cheese knife and cut their salami with the salami knife. We do not have any prior information whether or not subjects are instructed in this way. We tabulate our findings in Table 9 where $d = 30s$. In the table, we give the likelihood of the proposition in a column in 30 seconds is given with respect to the proposition in a row in the corresponding entry. For example, we observe that the probability $\mathcal{P}_{30}(\texttt{spread} \mid \texttt{cheese})$ is higher than $\mathcal{P}_{30}(\texttt{spread} \mid \texttt{salami})$. Similarly, we have $\mathcal{P}_{30}(\texttt{knife\_cheese} \mid \texttt{cheese}) > \mathcal{P}_{30}(\texttt{knife\_salami} \mid \texttt{cheese})$ and $\mathcal{P}_{30}(\texttt{knife\_salami} \mid \texttt{salami}) > \mathcal{P}_{30}(\texttt{knife\_cheese} \mid \texttt{salami})$. Indeed we see one cluster for `spread-cheese-knife_cheese`) and another for `cut-salami-knife_salami` as expected. For these clusters, the intra-cluster probabilities are consistently higher than the inter-cluster probabilities. Also notice that the sum of probabilities for a row sometimes exceeds 1 but it can be easily explained by the fact that there might be many "next" propositions in d time units for a proposition due to density and concurrency of time unlike sequential text. Another interesting observation is that the probability of repeating primitives are usually high, which suggests an iteration pattern as in the case of `sip` and `bite` can be considered as a template for primitive actions.

Table 9: Bigram-like probabilities for OPPORTUNITY dataset.

	cut	spread	cheese	salami	knife cheese	knife salami
cut	$\frac{6}{33} = 0.18$	$\frac{1}{33} = 0.03$	$\frac{5}{33} = 0.15$	$\frac{12}{33} = 0.36$	$\frac{9}{33} = 0.27$	$\frac{16}{33} = 0.48$
spread	$\frac{9}{31} = 0.29$	$\frac{12}{31} = 0.39$	$\frac{20}{31} = 0.65$	$\frac{13}{31} = 0.42$	$\frac{15}{31} = 0.48$	$\frac{13}{31} = 0.42$
cheese	$\frac{13}{228} = 0.06$	$\frac{25}{228} = 0.11$	$\frac{103}{228} = 0.45$	$\frac{48}{228} = 0.21$	$\frac{49}{228} = 0.21$	$\frac{28}{228} = 0.12$
salami	$\frac{23}{244} = 0.09$	$\frac{8}{244} = 0.03$	$\frac{36}{244} = 0.15$	$\frac{75}{244} = 0.31$	$\frac{25}{244} = 0.10$	$\frac{43}{244} = 0.18$
knife cheese	$\frac{18}{186} = 0.10$	$\frac{25}{186} = 0.13$	$\frac{46}{186} = 0.25$	$\frac{28}{186} = 0.15$	$\frac{84}{186} = 0.45$	$\frac{47}{186} = 0.25$
knife salami	$\frac{24}{162} = 0.15$	$\frac{4}{162} = 0.02$	$\frac{15}{162} = 0.09$	$\frac{31}{162} = 0.19$	$\frac{43}{162} = 0.27$	$\frac{68}{162} = 0.42$

To conclude, we are able to see meaningful statistical results using timed pattern matching as a counting tool over temporal datasets although the dataset is small compared to usual corpus sizes of n-gram applications. The structure of the dataset, in particular the high level scenario, helps us obtain these results. Otherwise, it would be much harder to observe something meaningful with a random activity dataset of a similar size.

7 CONCLUSIONS

Epigraf kullanmayın, çünkü yazının içindeki esrarı öldürür!

Never use epigraphs; they kill the mystery in the work!

— Black Book, Orhan Pamuk

In this thesis, we introduced timed pattern matching and developed its theory. We then used these results to implement a timed pattern matching tool and explored several application areas via case studies. Our main motivation has been the extension of the beautiful theory and useful applications of pattern matching, chiefly developed for text processing purposes in the past, toward the timed domain and the analysis of complex dynamical systems. For that, we have inspired by some classical works in automata, formal languages, and algebraic logic as well as our previous experiences in (runtime) verification and several practical issues we faced during the implementation and case studies. At the end, we can say concisely that a timed behavior is surely different than a text but this thesis shows that it is possible to handle these differences gracefully — at least in the context of pattern matching.

Some results of this thesis has been extended for timed automata patterns in [114, 115], for temporal logic patterns [13], for a measurement specification language in [43], and for quantitative domains [7, 14]. In the following, we discuss key specific points of the thesis and describe other possible future directions.

Timed Relations. In this thesis, we have defined finitely representable sets of time periods to be timed relations. This formulation is based on classical works of Boolean functions and the algebra of (binary) relations. We think that our formulation is important at least for three reasons. First, it provides a well-behaving and well-studied structure for time-related data. Second, it allows us to directly transfer some previous results such as canonical normal forms of Boolean expressions into our framework. Third, it leads to a separation (of concerns) between time-related data and higher-level formalisms (regular expressions and temporal logic). All in all, we believe the algebra of timed relations is a conceptual advance for timed formalisms despite its technical simplicity.

In this thesis, we represented timed relations as finite unions of zones. Accumulated knowledge on zones in timed systems research allowed us to get some results and a prototype implementation quickly. Alternatively, some decision diagram based solutions can be considered in the future since we know that Binary Decision Diagrams (BDDs) [25] usually provide more efficient representations of Boolean functions. In this case, one should expect that some operations might be easier (e.g. complementation) and some operations harder (e.g. compass operations) than the zone-based solution. We note that this direction explored by some works such as Numerical Decision Diagrams (NDDs) [12] and Clock Difference Diagrams (CDDs) [66] in the context of timed automata. We think that these proposals can be a starting point that can be improved by exploiting two-dimensional nature of timed relations.

Timed Pattern Specification. We have considered timed regular expressions and metric compass logic as timed pattern specification languages. These formalisms can express many compositions of timed patterns in an intuitive and elegant way thanks to their respective Boolean, regular, and temporal operators. Besides these operations, we have a great deal of freedom for atomic expressions when specifying timed pattern and we studied a few basic classes of atomic expressions in Section 4.5. This can be extended with more elaborate functions such as shape detectors that label specific shapes (pulses, spikes, decay curves, etc.) over real-valued behaviors and can be used as atomic propositions. For example, such an extension is perhaps more efficient than specifying a shape by a regular expression using several simple (threshold) predicates as we did in the case study in 6.2. Then, regular expressions and temporal logic would specify temporal compositions of primitive shapes. We also demonstrated in the case study in 6.4 that we can match composite (human) activities where we considered atomic expressions to be primitive activity detectors.

Next, although we believe that timed regular expressions and metric compass logic are adequate for a majority of pattern specification tasks, we mention some possible generalizations beyond these formalisms. Since context-free grammars are predominantly used to specify textual patterns for parsing purposes, a natural question would be the applicability and feasibility of a timed extension thereof. However, we argued in this thesis that intersection is a crucial operation for timed patterns. Therefore, we think that the generalization of context-free grammars under intersection (resp. Boolean operations) known as conjunctive (resp. Boolean) grammars [86] should be considered in this case. There are still many open questions regarding conjunctive and Boolean grammars but we consider it to be an elegant theory that can be useful in practice.

Timed Derivatives. We introduced derivatives of timed regular expressions and employed them for online timed pattern matching in Chapter 5. Derivatives are a very elegant algebraic tool in the theory of formal languages. Using them, it is very easy to achieve a sequential (also known as, online, streaming, or real-time) model of computation where we process a sequence and compute the output in an incremental manner. Therefore, we adapted Brzozowski's derivatives of regular expressions in a timed setting by considering elements from the set of timed relations to be constant values in regular expressions. A technical novelty in our timed derivatives is in handling dense sequences (over time), which do not naturally appear in similar proposals such as [63, 70] that extend derivatives with values (weights, multiplicities) over discrete sequences.

An interesting future work would be in studying and developing automata over timed relations and devising an automata-based algorithm for online timed pattern matching. An automata-based algorithm would be more congenial to the use of computers and can be implemented more efficiently than our rather naive rewriting algorithm based on derivatives. On the theory side, however, this is a topic that deserves a deeper and broader algebraic treatment outside the pattern matching theme of this thesis.

Montre and other implementations. The wide range of applications of pattern matching have been the main motivation of this thesis. We similarly believe that timed pattern matching would be a valuable tool in many application areas ranging from runtime verification and temporal data analytics to temporal data mining and supervisory control of robots. It is very hard, if not impossible, to reach all these applications in a satisfactory manner with a single implementation on a specific platform.

In this thesis, we implemented our algorithms as a generic command line program MONTRE that offers a basic but flexible timed pattern matching functionality. Given the goal of this thesis and available resources, we believe that this was the most rational choice. As demonstrated in case studies, we were not limited by any means and can use other tools that offer functions we needed in cooperation with MONTRE. However, we also acknowledge that a tighter integration might be desired for some data analysis frameworks. To this end, it would be useful to develop bindings and wrappers of MONTRE for programming languages such as Python.

Targeted implementations of timed pattern matching would be also important in order to reach certain communities. For example, during this thesis, timed pattern matching has been integrated in an analog and mixed signal circuit monitoring tool [84], which provides specific constructs and templates for that domain. Similarly, we think that specific implementations for robotics in robot operating systems and for temporal analytics in stream processing frameworks can be considered in the future.

BIBLIOGRAPHY

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd ed. Addison-Wesley, 2007 (cit. on p. 75).

[2] James F Allen. “Maintaining Knowledge about Temporal Intervals.” In: *Communications of the ACM* 26.11 (1983), pp. 832–843 (cit. on pp. 3, 15, 27, 41).

[3] James F Allen. “Towards a General Theory of Action and Time.” In: *Artificial Intelligence* 23.2 (1984), pp. 123–154 (cit. on p. 3).

[4] Rajeev Alur and David L Dill. “A Theory of Timed Automata.” In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235 (cit. on p. 4).

[5] Rajeev Alur, Dana Fisman, and Mukund Raghothaman. “Regular Programming for Quantitative Properties of Data Streams.” In: *Proceedings of the European Symposium on Programming (ESOP)*. 2016, pp. 15–40 (cit. on p. 18).

[6] Rajeev Alur and Thomas A Henzinger. “Logics and Models of Real Time: A Survey.” In: *Proceedings of the REX Workshop on Real-Time: Theory in Practice*. 1992, pp. 74–106 (cit. on p. 35).

[7] Rajeev Alur, Konstantinos Mamouras, and Dogan Ulus. “Derivatives of Quantitative Regular Expressions.” In: *Models, Algorithms, Logics and Tools*. 2017 (cit. on p. 81).

[8] Douglas E Appelt and David J. Israel. “Introduction to Information Extraction.” In: *AI Communications* 12.3 (1999), pp. 161–172 (cit. on p. 75).

[9] Dean N Arden. “Delayed-Logic and Finite-State Machines.” In: *Proceedings of the Symposium on Switching Circuit Theory and Logical Design (SWCT)*. 1961, pp. 133–151 (cit. on p. 12).

[10] Eugene Asarin, Paul Caspi, and Oded Maler. “A Kleene Theorem for Timed Automata.” In: *Proceedings of the Symposium on Logic in Computer Science (LICS)*. 1997, pp. 160–171 (cit. on pp. 4, 35, 38).

[11] Eugene Asarin, Paul Caspi, and Oded Maler. “Timed Regular Expressions.” In: *Journal of the ACM* 49.2 (2002), pp. 172–206 (cit. on pp. 4, 37, 38).

[12] Eugene Asarin, Marius Bozga, Alain Kerbrat, Oded Maler, Amir Pnueli, and Anne Rasse. “Data-Structures for the Verification of Timed Automata.” In: *Proceedings of the Workshop on Hybrid and Real-Time Systems (HART)*. 1997, pp. 346–360 (cit. on p. 82).

[13] Eugene Asarin, Oded Maler, Dejan Nickovic, and Dogan Ulus. "Combining the Temporal and Epistemic Dimensions for MTL Monitoring." In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2017 (cit. on p. 81).

[14] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. "On the Quantitative Semantics of Regular Expressions over Real-Valued Signals." In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2017 (cit. on p. 81).

[15] Howard Barringer, Ylies Falcone, Klaus Havelund, Giles Reger, and David Rydeheard. "Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors." In: *Proceedings of the Symposium on Formal Methods (FM)*. 2012, pp. 68–84 (cit. on p. 18).

[16] David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. "Monitoring Metric First-Order Temporal Properties." In: *Journal of the ACM* 62.2 (2015), p. 15 (cit. on p. 17).

[17] Johan van Benthem. *The Logic of Time. A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse*. Springer, 1991 (cit. on p. 3).

[18] Gérard Berry and Ravi Sethi. "From Regular Expressions to Deterministic Automata." In: *Theoretical Computer Science* 48.3 (1986), pp. 117–126 (cit. on p. 2).

[19] Patrick Blackburn, Johan van Benthem, and Frank Wolter. *Handbook of Modal Logic*. Elsevier, 2006 (cit. on pp. 19, 27).

[20] Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic*. Cambridge University Press, 2002 (cit. on pp. 19, 27).

[21] Archie Blake. "Canonical Expressions in Boolean Algebra." PhD thesis. University of Chicago, 1938 (cit. on pp. 19, 22).

[22] Marius Bozga, Susanne Graf, and Laurent Mounier. "IF-2.0: A Validation Environment for Component-based Real-Time Systems." In: *Proceedings of the Conference on Computer Aided Verification (CAV)*. 2002, pp. 343–348 (cit. on p. 21).

[23] Davide Bresolin, Dario Della Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. "Metric Propositional Neighborhood Logics: Expressiveness, Decidability, and Undecidability." In: *Proceedings of the European Conference on Artificial Intelligence (ECAI)*. Vol. 10. 2010, pp. 695–700 (cit. on p. 41).

[24] Frank Markham Brown. *Boolean Reasoning: The Logic of Boolean Equations*. Springer, 2012 (cit. on pp. 20, 22).

[25] Randal E Bryant. "Graph-based Algorithms for Boolean Function Manipulation." In: *IEEE Transactions on Computers* 100.8 (1986), pp. 677–691 (cit. on p. 81).

[26] Janusz A. Brzozowski. "Derivatives of Regular Expressions." In: *Journal of the ACM* 11.4 (1964), pp. 481–494 (cit. on pp. 2, 5, 7, 10, 12, 13, 49).

[27] Janusz A. Brzozowski and Ernst Leiss. “On Equations for Regular Languages, Finite Automata, and Sequential Networks.” In: *Theoretical Computer Science* 10.1 (1980), pp. 19–35 (cit. on pp. 1, 11).

[28] J. Richard Büchi. “Weak Second-Order Arithmetic and Finite Automata.” In: *Mathematical Logic Quarterly* 6.1-6 (1960), pp. 66–92 (cit. on p. 9).

[29] Eduard Cerny, Surrendra Dudani, John Havlicek, and Dmitry Korchemny. *SVA: The Power of Assertions in SystemVerilog*. Springer, 2014 (cit. on p. 2).

[30] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P Ravn. “A Calculus of Durations.” In: *Information Processing Letters* 40.5 (1991), pp. 269–276 (cit. on p. 16).

[31] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999 (cit. on p. 2).

[32] John Horton Conway. *Regular Algebra and Finite Machines*. Courier Corporation, 2012 (cit. on p. 9).

[33] Irving M Copi, Calvin C Elgot, and Jesse B Wright. “Realization of Events by Logical Nets.” In: *Journal of the ACM* 5.2 (1958), pp. 181–196 (cit. on p. 9).

[34] Yves Crama and Peter L Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, 2011 (cit. on p. 20).

[35] Maxime Crochemore and Wojciech Rytter. *Jewels of Stringology: Text Algorithms*. World Scientific, 2003 (cit. on p. 17).

[36] Dario Della Monica, Valentin Goranko, Angelo Montanari, and Guido Sciavicco. “Interval Temporal Logics: A Journey.” In: *Bulletin of the EATCS* 3.105 (2013) (cit. on p. 41).

[37] David L Dill. “Timing Assumptions and Verification of Finite-State Concurrent Systems.” In: *Proceedings of the Conference on Computer Aided Verification (CAV)*. 1989, pp. 197–212 (cit. on p. 21).

[38] Alexandre Donzé and Oded Maler. “Robust Satisfaction of Temporal Logic over Real-valued Signals.” In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2010, pp. 92–106 (cit. on p. 18).

[39] Dan B Dwyer and Tim J Gabbett. “Global Positioning System Data Analysis: Velocity Ranges and A New Definition of Sprinting for Field Sport Athletes.” In: *The Journal of Strength & Conditioning Research* 26.3 (2012), pp. 818–824 (cit. on pp. 72, 73).

[40] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Springer, 2007 (cit. on p. 2).

[41] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. “Reasoning with Temporal Logic on Truncated Paths.” In: *Proceedings of the Conference on Computer Aided Verification (CAV)*. 2003, pp. 27–39 (cit. on p. 13).

[42] Georgios E Fainekos and George J Pappas. “Robustness of Temporal Logic Specifications for Continuous-time Signals.” In: *Theoretical Computer Science* 410.42 (2009), pp. 4262–4291 (cit. on p. 18).

[43] Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus. “Measuring with Timed Patterns.” In: *Proceedings of the Conference on Computer Aided Verification (CAV)*. 2015 (cit. on pp. 69, 81).

[44] Bernd Finkbeiner and Lars Kuhtz. “Monitor Circuits for LTL with Bounded and Unbounded Future.” In: *Proceedings of the Workshop on Runtime Verification (RV)*. 2009, pp. 60–75 (cit. on p. 17).

[45] Bernd Finkbeiner and Henny Sipma. “Checking Finite Traces using Alternating Automata.” In: *Formal Methods in System Design* 24.2 (2004), pp. 101–127 (cit. on p. 17).

[46] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. “On the Temporal Analysis of Fairness.” In: *Proceedings of the Symposium on Principles of Programming Languages (POPL)*. 1980, pp. 163–173 (cit. on pp. 13, 41).

[47] Valentin Goranko and Antony Galton. “Temporal Logic.” In: *The Stanford Encyclopedia of Philosophy (Winter 2015 Edition)*. Ed. by Edward N. Zalta. 2015 (cit. on p. 7).

[48] Joseph Y Halpern and Yoav Shoham. “A Propositional Modal Logic of Time Intervals.” In: *Journal of the ACM* 38.4 (1991), pp. 935–962 (cit. on pp. 3, 15, 27, 41).

[49] Klaus Havelund. “Rule-based Runtime Verification Revisited.” In: *International Journal on Software Tools for Technology Transfer* 17.2 (2015), pp. 143–170 (cit. on p. 18).

[50] Klaus Havelund, Doron Peled, and Dogan Ulus. “First-Order Temporal Logic Monitoring with BDDs.” In: *Proceedings of the Conference on Formal Methods in Computer-Aided Design (FMCAD)*. 2017 (cit. on p. 18).

[51] Klaus Havelund and Giles Reger. “Runtime Verification Logics: A Language Design Perspective.” In: *Models, Algorithms, Logics and Tools*. Springer, 2017, pp. 310–338 (cit. on p. 2).

[52] Klaus Havelund and Grigore Roşu. “Monitoring Programs using Rewriting.” In: *Proceedings of the Conference on Automated Software Engineering (ASE)*. 2001, pp. 135–143 (cit. on p. 17).

[53] Klaus Havelund and Grigore Roşu. “Synthesizing Monitors for Safety Properties.” In: *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Vol. 2. 2002, pp. 342–356 (cit. on p. 17).

[54] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson, 2013 (cit. on pp. 7, 17).

[55] Lucian Ilie, Baozhen Shan, and Sheng Yu. "Fast Algorithms for Extended Regular Expression Matching and Searching." In: *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS)*. 2003, pp. 179–190 (cit. on p. 17).

[56] Edwin H Jacox and Hanan Samet. "Spatial Join Techniques." In: *ACM Transactions on Database Systems* 32.1 (2007), p. 7 (cit. on p. 29).

[57] Bjarni Jonnson and Alfred Tarski. "Boolean Algebras with Operators." In: *American Journal of Mathematics* 74.1 (1952), pp. 127–162 (cit. on pp. 19, 27).

[58] Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. MIT Press (cit. on pp. 75, 77).

[59] Hans Kamp. "Tense Logic and the Theory of Linear Order." PhD thesis. University of California, Los Angeles, 1968 (cit. on p. 13).

[60] Stephen Cole Kleene. "Representations of Events in Nerve Nets and Finite Automata." In: *Automata Studies: Annals of Mathematics Studies* 34 (1956), pp. 3–42 (cit. on pp. 1, 7, 9).

[61] James R Knight and Eugene W Myers. "Super-Pattern Matching." In: *Algorithmica* 13.1 (1995), pp. 211–243 (cit. on p. 17).

[62] Ron Koymans. "Specifying Real-Time Properties with Metric Temporal Logic." In: *Real-Time Systems* 2.4 (1990), pp. 255–299 (cit. on pp. 4, 41).

[63] Daniel Krob. "Differentiation of K-Rational Expressions." In: *International Journal of Algebra and Computation* 2.01 (1992), pp. 57–87 (cit. on p. 83).

[64] Orna Kupferman and Sharon Zuhovitzky. "An Improved Algorithm for the Membership Problem for Extended Regular Expressions." In: *Proceedings of the Symposium on Mathematical Foundations of Computer Science (MFCS)*. Springer, 2002, pp. 446–458 (cit. on p. 17).

[65] Leslie Lamport. "What Good is Temporal Logic?" In: *The IFIP Congress*. Vol. 83. 1983, pp. 657–668 (cit. on pp. 2, 37).

[66] Kim G Larsen, Carsten Weise, Wang Yi, and Justin Pearson. "Clock Difference Diagrams." In: *Nordic Journal of Computing* 6.3 (1999), pp. 271 –298 (cit. on p. 82).

[67] Ernst Leiss. *Language Equations*. Springer, 1999 (cit. on p. 11).

[68] Martin Leucker and Christian Schallhart. "A Brief Account of Runtime Verification." In: *The Journal of Logic and Algebraic Programming* 78.5 (2009), pp. 293–303 (cit. on p. 2).

[69] Orna Lichtenstein, Amir Pnueli, and Lenore Zuck. “The Glory of the Past.” In: *Proceedings of the Workshop on Logic of Programs*. 1985, pp. 196–218 (cit. on p. 13).

[70] Sylvain Lombardy and Jacques Sakarovitch. “Derivatives of Rational Expressions with Multiplicity.” In: *Theoretical Computer Science* 332.1 (2005), pp. 141–177 (cit. on p. 83).

[71] Oded Maler. “Some Thoughts on Runtime Verification.” In: *Proceedings of the Conference on Runtime Verification (RV)*. 2016, pp. 3–14 (cit. on p. 2).

[72] Oded Maler and Dejan Nickovic. “Monitoring Temporal Properties of Continuous Signals.” In: *Proceedings of the Conference on Formal Techniques, Modeling and Analysis of Timed and Fault-Tolerant Systems (FORMATS)*. 2004, pp. 152–166 (cit. on pp. 17, 27, 44).

[73] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer Science & Business Media, 1995 (cit. on p. 13).

[74] Nicolas Markey and Philippe Schnoebelen. “Model Checking A Path.” In: *Proceedings of the Conference on Concurrency Theory (CONCUR)*. 2003, pp. 251–265 (cit. on p. 17).

[75] Maarten Marx and Yde Venema. *Multi Dimensional Modal Logic*. Vol. 4. Springer Science & Business Media, 2012 (cit. on p. 27).

[76] R. McNaughton and H. Yamada. “Regular Expressions and State Graphs for Automata.” In: *IRE Transactions on Electronic Computers* EC-9.1 (1960), pp. 39–47 (cit. on p. 2).

[77] George H Mealy. “A Method for Synthesizing Sequential Circuits.” In: *Bell Labs Technical Journal* 34.5 (1955), pp. 1045–1079 (cit. on pp. 1, 50).

[78] Edward F. Moore. “Gedanken-Experiments on Sequential Machines.” In: *Automata Studies: Annals of Mathematics Studies* 34 (1956), pp. 129–153 (cit. on pp. 1, 50).

[79] Ben Moszkowski. “Executing Temporal Logic Programs.” In: *Proceedings of the Conference on Concurrency Theory (CONCUR)*. 1984, pp. 111–130 (cit. on p. 16).

[80] Ion Muslea et al. “Extraction Patterns for Information Extraction Tasks: A Survey.” In: *Proceedings of the AAAI Workshop on Machine Learning for Information Extraction*. 1999 (cit. on p. 75).

[81] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings: Practical On-line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, 2002 (cit. on p. 17).

[82] Anil Nerode. “Linear Automaton Transformations.” In: *Proceedings of the American Mathematical Society* 9.4 (1958), pp. 541–544 (cit. on p. 9).

[83] Dejan Nickovic. "Checking Timed and Hybrid Properties: Theory and Applications." PhD thesis. Université Joseph Fourier, 2008 (cit. on p. 27).

[84] Dejan Nickovic, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. "AMT2.0: Qualitative and Quantitative Trace Analysis with Extended Signal Temporal Logic." In: *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2018 (cit. on p. 83).

[85] Jürg Nievergelt and Franco P. Preparata. "Plane-Sweep Algorithms for Intersecting Geometric Figures." In: *Communications of the ACM* 25.10 (1982), pp. 739–747 (cit. on p. 29).

[86] Alexander Okhotin. "Conjunctive and Boolean grammars: The True General Case of the Context-Free Grammars." In: *Computer Science Review* 9 (2013), pp. 27–59 (cit. on p. 82).

[87] Joël Ouaknine and James Worrell. "Some Recent Results in Metric Temporal Logic." In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2008, pp. 1–13 (cit. on p. 41).

[88] Scott Owens, John H. Reppy, and Aaron Turon. "Regular Expression Derivatives Re-examined." In: *Journal of Functional Programming* 19.2 (2009), pp. 173–190 (cit. on p. 67).

[89] Amir Pnueli. "The Temporal Logic of Programs." In: *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*. 1977, pp. 46–57 (cit. on pp. 2, 7, 13).

[90] Amir Pnueli and Aleksandr Zaks. "On the Merits of Temporal Testers." In: *25 Years of Model Checking*. Springer, 2008, pp. 172–195 (cit. on p. 17).

[91] Arthur N. Prior. *Time and Modality*. Clarendon Press Oxford, 1957 (cit. on pp. 2, 7, 13).

[92] Arthur N Prior. *Past, Present and Future*. Clarendon Press Oxford, 1967 (cit. on pp. 2, 7, 13).

[93] Michael O. Rabin and Dana Scott. "Finite Automata and Their Decision Problems." In: *IBM Journal of Research and Development* 3.2 (1959), pp. 114–125 (cit. on pp. 1, 11).

[94] George N Raney. "Sequential Functions." In: *Journal of the ACM* 5.2 (1958), pp. 177–180 (cit. on pp. 49, 50).

[95] Anders P Ravn. "Design of Embedded Real-Time Computing Systems." PhD thesis. Technical University of Denmark, Lyngby, 1995 (cit. on p. 35).

[96] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, et al. "Collecting Complex Activity Datasets in Highly Rich Networked Sensor Environments." In: *Proceedings of the International Conference on Networked Sensing Systems (INSS)*. 2010, pp. 233–240 (cit. on p. 75).

[97] Roni Rosner and Amir Pnueli. "A Choppy Logic." In: *Proceedings of the Symposium on Logic in Computer Science (LICS)*. 1986 (cit. on p. 16).

[98] Grigore Roşu. "An Effective Algorithm for the Membership Problem for Extended Regular Expressions." In: *Proceedings of the Conference on Foundations of Software Science and Computational Structures (FOSSACS)*. Springer, 2007, pp. 332–345 (cit. on p. 17).

[99] Grigore Roşu and Klaus Havelund. "Rewriting Based Techniques for Runtime Verification." In: *Automated Software Engineering* 12.2 (2005), pp. 151–197 (cit. on p. 17).

[100] Grigore Roşu and Mahesh Viswanathan. "Testing Extended Regular Language Membership Incrementally by Rewriting." In: *Proceedings of the Conference on Rewriting Techniques and Applications (RTA)*. 2003, pp. 499–514 (cit. on pp. 17, 67).

[101] Grzegorz Rozenberg and Arto Salomaa. *Handbook of Formal Languages: Volume 1 Word, Language, Grammar*. Springer, 1997 (cit. on p. 7).

[102] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009 (cit. on p. 7).

[103] Ernst Schröder. *Algebra und Logik der Relative, der Vorlesungen über die Algebra der Logik Dritter Band*. 1895 (cit. on p. 19).

[104] Marshall H Stone. "The Theory of Representation for Boolean Algebras." In: *Transactions of the American Mathematical Society* 40.1 (1936), pp. 37–111 (cit. on p. 19).

[105] Alfred Tarski. "On the Calculus of Relations." In: *The Journal of Symbolic Logic* 6.03 (1941), pp. 73–89 (cit. on p. 19).

[106] Prasanna Thati and Grigore Roşu. "Monitoring Algorithms for Metric Temporal Logic Specifications." In: *Electronic Notes in Theoretical Computer Science* 113 (2005), pp. 145–162 (cit. on p. 17).

[107] Ken Thompson. "Regular Expression Search Algorithm." In: *Communications of the ACM* 11.6 (1968), pp. 419–422 (cit. on pp. 2, 17).

[108] Dogan Ulus. "Montre: A Tool for Monitoring Timed Regular Expressions." In: *Proceedings of the Conference on Computer Aided Verification (CAV)*. 2017 (cit. on p. 63).

[109] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. "Timed Pattern Matching." In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2014, pp. 222–236 (cit. on pp. 66, 67).

[110] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. "Online Timed Pattern Matching using Derivatives." In: *Proceedings of the Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2016, pp. 736–751 (cit. on p. 67).

[111] Moshe Y Vardi. “From Church and Prior to PSL.” In: *25 Years of Model Checking*. Springer, 2008, pp. 150–171 (cit. on p. 17).

[112] Yde Venema. “Expressiveness and Completeness of an Interval Tense Logic.” In: *Notre Dame Journal of Formal Logic* 31.4 (1990), pp. 529–547 (cit. on pp. 15, 27, 41).

[113] Yde Venema. “A Modal Logic for Chopping Intervals.” In: *Journal of Logic and Computation* 1.4 (1991), pp. 453–476 (cit. on pp. 4, 16).

[114] Masaki Waga, Takumi Akazaki, and Ichiro Hasuo. “A Boyer-Moore Type Algorithm for Timed Pattern Matching.” In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2016, pp. 121–139 (cit. on p. 81).

[115] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. “Efficient Online Timed Pattern Matching by Automata-Based Skipping.” In: *Proceedings of the Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*. 2017, pp. 224–243 (cit. on p. 81).

[116] Bruce W Watson and Richard E Watson. “A Boyer–Moore-style Algorithm for Regular Expression Pattern Matching.” In: *Science of Computer Programming* 48.2-3 (2003), pp. 99–117 (cit. on p. 17).

[117] Hiroaki Yamamoto. “A New Recognition Algorithm for Extended Regular Expressions.” In: *Proceedings of the International Symposium on Algorithms and Computation (ISAAC)*. Springer, 2001, pp. 257–267 (cit. on p. 17).

COLOPHON

The first edition of this manuscript was finalized in March 2018.

This manuscript was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*.

The mellow dragon curve at the back cover was generated by the following code that uses the `tikz` package `lindenmayersystems`.

```
\begin{tikzpicture}[rotate=65]
\draw[black, ultra thick]
  l-system
  [l-system={
    rule set={X->X+YF+,Y->-FX-Y},
    axiom=FX,
    angle=92,
    order=12,
    step=10pt
    }
  ];
\end{tikzpicture}
```