Parametric timed formalisms for specification and monitoring

Akshay Mambakam

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : VERIMAG

Formalismes temporisés paramétriques pour la spécification et la surveillance

Parametric timed formalisms for specification and monitoring

Présentée par :

Akshay MAMBAKAM

Direction de thèse :

Thao DANG Directeur de recherche, CNRS Délégation Alpes	Directrice de thèse
Nicolas BASSET Maitre de conférences, Université Grenoble Alpes	Co-encadrant de thèse

Rapporteurs :

ETIENNE ANDRE
Professeur des Universités, UNIVERSITE SORBONNE PARIS NORD
EZIO BARTOCCI
Professeur, Technische Universität Wien

Thèse soutenue publiquement le **7 juillet 2023**, devant le jury composé de :

THAO DANG Directeur de recherche, CNRS DELEGATION ALPES	Directrice de thèse
ETIENNE ANDRE Professeur des Universités, UNIVERSITE SORBONNE PARIS NORD	Rapporteur
EZIO BARTOCCI Professeur, Technische Universität Wien	Rapporteur
DEJAN NICKOVIC Senior scientist, Austrian Institute of Technology	Examinateur
SADDEK BENSALEM Professeur des Universités, UNIVERSITE GRENOBLE ALPES	Président
NICOLA PAOLETTI Associate professor, King's College London	Examinateur
PATRICIA BOUYER-DECITRE Directeur de recherche, CNRS DELEGATION ILE-DE-FRANCE SUD	Examinatrice

Invités :

EUGENE ASARIN
Professeur des Universités, UNIVERSITE PARIS CITE
NICOLAS BASSET
Maitre de conférences, UNIVERSITE GRENOBLE ALPES

ACKNOWLEDGMENTS

My time at Verimag laboratory as a doctoral student has been stimulating not only as an intellectual exercise but also as a motivation to find practical solutions for real-world problems. The challenges I faced came coupled with a highly encouraging international environment that made them easy.

I would first like to thank my supervisors, Thao Dang and Nicolas Basset, who patiently guided me especially during the difficult parts of my studies. I would also like to thank Eugene Asarin for his excellent guidance which he provided while facing the constraints of virtual interaction.

I would also like to thank Abdelhakim Baouya, Nikolaos Kekatos, and José Ignacio Requeno Jarabo for gently introducing me to the research field and helping me grow into a contributing member. A big thanks also to my colleagues and friends Aina, Asfand, Eleonora, Hadi, Marco, and Thomas as well as all other students and staff of Verimag I could not name here.

Finally I thank my parents who blessed me with constant support and encouragement throughout my life.

ABSTRACT

Cyber-physical systems (CPS) consist of computer systems which control physical processes. Examples of this include medical devices, autonomous cars, and robots. Due to the complexity and heterogeneity of their components, it is not always possible to derive mathematical models for such systems and data-driven design approaches are thus a crucial alternative. This thesis is built upon extensions of two specification formalisms, Signal Temporal Logic (STL) and Timed Regular Expressions (TRE), which are expressive to describe temporal aspects of CPS behaviours and additionally suitable for monitoring and validation purposes.

STL is a temporal logic that is adapted to specify properties of real-valued signals. We extend the logic with operators for counting events. These operators, for example, can be used to count Electrocardiogram (ECG) pulses. Parametric specifications (like parametric STL) over signals when assigned parameters can be viewed as producing predictors that indicate where a temporal pattern occurs. We term as Increasing Parametric Pattern Predictors (IPPP) specifications that become easy to satisfy as the parameter values increase. Given data labelled by experts, we propose a method to learn predictors by finding parameters of IPPP to match the labelled data as accurately as possible. We introduce a measure called epsilon-count to quantify prediction accuracy. For IPPP, decreasing both false positives and false negatives involves optimising opposing targets. We devise an algorithm that identifies parameters while respecting these opposing constraints. Pareto domination and binary search are the two main ideas behind efficient implementation of this algorithm.

TRE extend regular expressions with timing constraints defined over duration. Monitoring using TRE specifications is termed as Timed Pattern Matching (TPM). This involves finding in the input behaviour, all time intervals where the specification is matched. Parametric Timed Pattern Matching (PTPM) can be defined over parametric versions of TRE (PTRE) in which we compute along with time intervals also the parameter values that allow the match. The semantics of PTRE slightly differ for real-valued signals and sequences of time-stamped events. We show that for these two types of semantics we get two kinds of polytopes which we term parametric zones and parametric intervals. Operations over parametric zones can be optimised by exploiting their separation in time similar to the plane-sweep algorithms used for TPM. Operations over parametric intervals can also benefit greatly from sorting and subsequent binary-search procedures. We also show how parametric identification can be performed on top

of PTPM to compute all parameter values that makes the specification matches patterns labelled by experts.

For both parametric temporal formulae and expressions, along with synthetic examples for reader's understanding, we also apply our methods to analysis of ECGs and marine traffic data.

RÉSUMÉ

Les systèmes cyber-physiques (Cyber Physical Systems, CPS) consistent en des systèmes informatiques qui contrôlent les processus physiques. Les dispositifs médicaux, les voitures autonomes et les robots en sont des exemples. En raison de la complexité et de l'hétérogénéité de leurs composants, il n'est pas toujours possible de dériver des modèles mathématiques pour de tels systèmes et les approches de conception basées sur les données sont donc une alternative cruciale. Cette thèse est construite sur des extensions de deux formalismes de spécification, la logique temporelle du signal (Signal Temporal Logic STL) et les expressions régulières temporisées (Timed Regular Expressions TRE), qui sont expressifs pour décrire les aspects temporels des comportements CPS et en outre adaptés à des fins de surveillance et de validation.

STL est une logique temporelle adaptée pour spécifier les propriétés des signaux à valeurs réelles. Nous étendons la logique avec des opérateurs pour compter les événements. Ils peuvent être utilisés, par exemple, pour compter les impulsions d'électrocardiogramme (ECG). Les spécifications paramétriques (telles qu'écrites en STL paramétrique, PSTL) sur les signaux, lorsque des paramètres sont fixés, peuvent être considérées comme des prédicteurs qui indiquent à quels endroit d'un signal un motif temporel se produit. Nous appelons prédicteurs de modèles paramétriques croissants (Increasing Parametric Pattern Predictors IPPP) des spécifications paramétriques qui deviennent de plus en plus faciles à satisfaire à mesure que les valeurs des paramètres augmentent. Nous proposons une méthode pour apprendre des prédicteurs en trouvant des paramètres d'IPPP qui produisent un prédicteur presque aussi précis que le signal d'étiquetage donné par les experts, utilisé pour l'entrainement. Nous introduisons une mesure appelée epsilon-count pour quantifier la précision de la prédiction. Pour les IPPP, la diminution des faux positifs et des faux négatifs implique l'optimisation d'objectifs opposées. Nous concevons un algorithme qui identifie des paramètres tout en respectant ces contraintes opposées. La domination de Pareto et la recherche dichotomique sont les deux idées principales derrière la mise en œuvre efficace de cet algorithme.

Les TRE étendent les expressions régulières avec des contraintes sur la durée. La surveillance à l'aide des spécifications TRE est appelée correspondance par motif temporisé (Timed Pattern Matching TPM). Cela implique de trouver tous les intervalles de temps durant lesquels un signal donné se comporte selon la spécification donnée. La correspondance paramétrique temporisée (Parametric Timed Pattern Matching PTPM) peut être définie sur des versions paramé-

triques de TRE (PTRE) dans lesquelles nous calculons, en plus des intervalles de temps, les valeurs de paramètre qui permettent la correspondance. La sémantique de PTRE diffère légèrement pour les signaux à valeurs réelles et les séquences d'événements temporisés. Nous montrons que pour ces deux types de sémantique, nous obtenons deux types de polytopes que nous appelons zones paramétriques et intervalles paramétriques. Les opérations sur les zones paramétriques peuvent être optimisées en exploitant la dimension temporelle avec des algorithmes de balayage de plan utilisés pour TPM. Les opérations sur des intervalles paramétriques peuvent également bénéficier du tri et des procédures de recherche dichotomique ultérieures. Nous montrons également comment l'identification paramétrique peut être effectuée au-dessus de PTPM pour calculer toutes les valeurs de paramètres qui font que la spécification permet de retrouver exactement les motifs annotés par des experts.

Pour les deux types de formalismes temporisés (PSTL et PTRE) nous proposons des exemples synthétiques pour la compréhension du lecteur. Nous appliquons également nos méthodes à l'analyse des ECG et des données de trafic maritime.

PUBLICATIONS

[1] Nicolas Basset, Thao Dang, Akshay Mambakam, and José Ignacio Requeno Jarabo. "Learning Specifications for Labelled Patterns." In: *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*. Ed. by Nathalie Bertrand and Nils Jansen. Vol. 12288. Lecture Notes in Computer Science. Springer, 2020, pp. 76–93. DOI: 10.1007/978-3-030-57628-8_5. URL: https://doi.org/10.1007/978-3-030-57628-8_5.

[2] Akshay Mambakam, Eugene Asarin, Nicolas Basset, and Thao Dang. "Pattern Matching and Parameter Identification for Parametric Timed Regular Expressions." In: *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2023, San Antonio, TX, USA, May 9-12, 2023*. ACM, 2023, 14:1–14:13. DOI: 10.1145/3575870.3587115. URL: https://doi.org/10.1145/3575870.3587115.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

TRE Timed Regular Expressions
SRE Signal Regular Expressions
QRE Quantitative Regular Expressions
STL Signal Temporal Logic
TA Timed Automata
PSTL Parametric Signal Temporal Logic
PTA Parametric Timed Automata
PTDA Parametric Timed Data Automata

ML Machine Learning
DFA Deterministic Finite Automata
LTL Linear-time Temporal Logic
MITL Metric Interval Temporal Logic
MCL Metric Compass Logic
E-TRE Event-Bounded Timed Regular Expressions
PSRE Parametric Signal Regular Expressions
PTRE Parametric Timed Regular Expressions
PE-TRE Parametric Event-Bounded Timed Regular Expressions
AI Artificial Intelligence
DNN Deep Neural Networks
TPM Timed Pattern Matching
PTPM Parametric Timed Pattern Matching
CSV Comma-Separated Values

INTRODUCTION

Les systèmes cyber-physiques (Cyber Physical Systems, CPS) impliquent une interaction entre les dispositifs informatiques et l'environnement physique et sont devenus largement utilisés dans le monde moderne. Dans de nombreux cas, ces systèmes doivent être utilisés dans des scénarios de sécurité critiques. Il est donc nécessaire de vérifier que ces systèmes se comportent correctement dans tous les scénarios possibles, ce que l'on appelle la vérification formelle. Lorsque, en raison de la grande taille et de la complexité élevée des systèmes réels, la vérification n'est pas possible, nous devons nous rabattre sur des formes légères de vérification telles que la vérification du temps d'exécution. Les formalismes de modélisation temporelle tels que les automates temporisés (Timed Automata, TA), la logique temporelle des signaux (Signal Temporal Logic, STL) et les expressions régulières temporisées (Timed Regular Expressions, TRE) se sont révélés appropriés pour exprimer les comportements temporels à un certain niveau d'abstraction. Il existe des algorithmes efficaces qui détectent si les spécifications exprimées à l'aide de formalismes temporels sont satisfaites/violées par une trace observée obtenue par simulation ou par observation réelle. En particulier, il existe des algorithmes de surveillance pour les formules STL dont la complexité temporelle est, dans le pire des cas, linéaire en fonction de la taille de la trace surveillée. Cela a conduit à une utilisation assez large de STL dans la recherche et l'industrie. Pour une spécification STL donnée et une trace d'entrée, nous pouvons obtenir ce que l'on appelle un signal de satisfaction. Il s'agit d'un signal booléen qui est vrai aux points temporels où la spécification est satisfaite et faux dans le cas contraire. La surveillance des TA et TRE n'est pas basée sur le concept de signal de satisfaction. Il s'agit de trouver l'ensemble des intervalles de temps où la race d'entrée satisfait la spécification du TA ou de la TRE.

Le paradigme de l'apprentissage supervisé implique une classe d'objets mathématiques représentant la classe d'hypothèses et des données contenant des observations et un étiquetage par des experts concernant ces observations. À partir des données, nous obtenons l'hypothèse la plus apte à exprimer la relation entre les observations et l'étiquetage. Ce paradigme peut être appliqué à l'apprentissage de formules STL à partir de traces obtenues à partir de systèmes sous observation. La classe d'hypothèses la plus générale contiendrait toutes les formules STL possibles sur les variables des observations. Une classe d'hypothèses plus simple serait une formule STL paramétrique. Nous connaissons la structure de la formule, mais seuls les pa-

ramètres sont inconnus. Le processus consistant à trouver les valeurs des paramètres pour lesquelles la formule STL produite est satisfaite pour toutes les observations est appelé identification paramétrique.

Notre première contribution concerne la logique temporelle des signaux paramétriques (Parametric Signal Temporal Logic, PSTL). Au lieu d'une identification paramétrique stricte, nous calculons les valeurs des paramètres pour lesquels la formule correspond approximativement aux observations dans les limites définies par l'utilisateur sur les taux d'erreur faux positifs et faux négatifs. À cette fin, nous introduisons une mesure, appelée ϵ-count, du nombre de fois qu'un signal booléen devient vrai. Les prédicteurs de motifs paramétriques (Parametric Pattern Predictor, PPP) peuvent être définis comme des spécifications qui, pour une valeur donnée des paramètres, prennent une observation et produisent un signal booléen qui est vrai lorsqu'un motif est prédit et faux ailleurs. Les prédicteurs de motifs paramétriques croissants (Increasing Parametric Pattern Predictors, IPPP) sont la catégorie dans laquelle l'augmentation des valeurs des paramètres accroît l'ensemble des points temporels où le motif est prédit. Nous montrons comment la recherche de prédicteurs de motifs est liée au problème de l'exploration des ensembles optimaux de Pareto à l'aide de requêtes. Nous proposons un algorithme qui calcule approximativement l'ensemble d'intersection contenu entre deux ensembles Pareto optimaux de polarités opposées. L'idée cruciale qui sous-tend l'algorithme est la recherche dichotomique adaptée aux intervalles continus et aux dimensions multiples. L'algorithme prend en entrée une boîte à n dimensions indiquant l'ensemble des valeurs prises par les paramètres. Le résultat principal est un ensemble de boîtes qui se rapprochent de l'ensemble d'intersection. L'approche de l'algorithme peut être considérée de manière générale comme "diviser pour régner". Étant donné une boîte, nous effectuons une recherche dichotomique sur la diagonale. En fonction du résultat de la recherche, la boîte est divisée en trois types de sous-boîtes. Le premier type est constitué des boîtes qui se trouvent dans l'ensemble d'intersection. Le deuxième type est celui des boîtes dont on sait définitivement qu'elles ne sont pas contenues dans l'intersection. Le troisième type est celui des boîtes indéterminées, c'est-à-dire que nous ne savons pas si elles peuvent faire partie de l'intersection. Nous utilisons la récurrence et répétons le processus dans ces cases. Le processus s'arrête lorsque l'hyper-volume des ensembles de type indéterminé passe en dessous d'une limite définie par l'utilisateur.

La deuxième contribution concerne les expressions régulières temporisés paramétriques (Parametric Timed Regular Expressions, PTRE). Pour trois types différents de PTRE, nous résolvons le problème de la correspondance paramétrique temporisée (Parametric Timed Pattern Matching, PTPM). Il s'agit de trouver l'ensemble des valeurs des paramètres ainsi que l'instant de début et l'instant de fin des inter-

valles pour lesquels il existe une correspondance du sous-signal par rapport à la TRE. Il existe deux grands types de PTRE. Le premier, appelé basé sur l'état, traite des signaux à valeur booléenne. L'autre type, appelé basé sur les événements, traite de la liste des événements survenus à des dates observées. Pour le type basé sur l'état, nous utilisons des techniques de balayage de plan précédemment appliquées à la TRE, pour réaliser un PTPM efficace en utilisant des polytopes. L'idée est de lier les polytopes en rectangles sur les dimensions temporelles, puis d'utiliser le tri pour éviter d'effectuer des opérations (par exemple, l'intersection) dont le résultat est connu pour être un polytope vide. Pour le type basé sur les événements, nous montrons que le PTPM peut être réalisé à l'aide d'algorithmes qui utilisent une structure de données spéciale contenant des intervalles de temps. Les algorithmes utilisent efficacement le tri et la recherche dichotomique. Nous adaptons également le PTPM à l'identification paramétrique de la PTRE. Pour ce faire, l'étiquetage de l'expert est donné sous la forme d'un ensemble fini d'intervalles de temps où des motifs intéressants se produisent. Nous trouvons l'ensemble des paramètres pour lesquels nous avons une correspondance à chacun des intervalles de temps donnés. Pour optimiser l'identification paramétrique, nous définissons la notion de *région décisive*. Pour un intervalle de temps d'étiquetage donné, nous pouvons définir un triangle à angle droit sur les deux dimensions temporisées. Tous les intervalles situés en dehors de ce triangle peuvent être ignorés lors de l'identification paramétrique de l'intervalle d'étiquetage. Cette technique améliore considérablement la complexité temporelle.

La troisième contribution concerne les cas non paramétriques de TRE et STL. Nous introduisons des opérateurs de minimum, de maximum et de différence dans TRE et fournissons des théorèmes et des algorithmes en temps linéaire pour leur mise en correspondance temporelle. Nous introduisons également des opérateurs de comptage dans STL qui comptent le nombre d'événements dans une fenêtre glissante ou jusqu'à ce qu'un autre événement se produise. Les algorithmes que nous avons conçus pour ces opérateurs sont efficaces et s'exécutent en temps linéaire.

1 INTRODUCTION

Cyber-Physical Systems (CPS) involve interaction between computing devices and physical environment and have become widely used in the modern world. In many cases these systems need to be applied in safety-critical scenarios. This necessitates that such systems need to be verified to behave correctly under all possible scenarios, termed as formal verification. When due to the large size and high complexity of real-life systems verification is not possible, we have to fallback on light-weight forms of verification such as runtime verification. Timed modelling formalisms such as Timed Automata (TA), Signal Temporal Logic (STL) and Timed Regular Expressions (TRE) have been shown to be suitable to express behaviours at the timed level of abstraction. Efficient algorithms exist that detect whether specification expressed using timed formalisms are satisfied/violated by an observed trace obtained from either simulation or actual observation. Especially, monitoring algorithms for STL formulae exist that have worst-case linear time complexity in the size of the monitored trace. This led to STL being used widely both in academia and industry for runtime verification. For a given STL specification and an input trace, we can obtain what is called a satisfaction signal. This is a Boolean signal which is true at time points where specification is satisfied and false otherwise. Monitoring TA and TRE is not based on the concept of satisfaction signal. But it involves Timed Pattern Matching (TPM) which involves finding the set of all time intervals where the observation satisfies the TA or TRE.

The paradigm of supervised learning involves a class of mathematical objects representing the hypothesis class and data containing observations and labelling by experts regarding these observations. From the data we obtain the hypothesis that is most suitable in expressing the relation between the observations and the labelling. This paradigm can be applied to "learning" STL formulae from traces obtained from systems under observation. The most general hypothesis class would contain all possible STL formulae over the variables in the observations. A simpler hypothesis class would be a parametric STL formula. We know the structure of the formula but only the parameters are unknown. The process of finding the parameter values for which the produced STL formula is satisfied over all the observations is called parametric identification.

Our first contribution involves Parametric Signal Temporal Logic (PSTL). Instead of strict parametric identification, we compute the parameter values for which the formula approximately matches the

observations within the user defined bounds on the false positive and false negative error rates. For this purpose, we introduce a measure called ϵ-count of the number of times a Boolean signal becomes true. Parametric Pattern Predictors (PPP) can be defined as specifications, for a given value of parameters, take an observation and produce a Boolean signal which is true where a pattern is predicted an false elsewhere. Increasing Parametric Pattern Predictors (IPPP) are the class where increasing the parameter values augments the set of time points where the pattern is predicted. We show how finding pattern predictors is linked to the problem of exploring Pareto optimal sets using queries. And, we propose an algorithm that approximately computes the intersection set contained between two Pareto optimal sets of opposite polarities. The crucial idea behind the algorithm is binary search adapted to continuous intervals and multiple dimensions. The algorithm takes as input an n-dimensional box indicating the set of values taken by the parameters. The main output is a set of boxes that approximate the intersection set. The approach of the algorithm can be broadly considered as divide and conquer. Given a box, we perform binary search on the diagonal. Based on the result of the search, the box is divided into three kinds of sub-boxes. The first kind are the boxes that are in the intersection set. The second kind are those that are known definitively to be not contained in the intersection set. The remaining third type are those which are undetermined i.e. we do not know whether they might have some part of the intersection set. We use recurrence and repeat the process in these boxes. The process stops when the hyper-volume of the sets of type undetermined goes below a user-defined bound.

The second contribution involves Parametric Timed Regular Expressions (PTRE). For three different flavours of PTRE we solve the problem of Parametric Timed Pattern Matching (PTPM). This involves finding the set of parameter values and the start time and end time of intervals for which there exists a match of the sub-signal with respect to the PTRE. There exist two broad types of PTRE. One type, called state-based, deals with Boolean valued signals. The other type, called event-based, deals with list of events occurring at observed time-stamps. For the state-based type, we utilize plane-sweep techniques previously applied to TRE, to do efficient PTPM using polytopes. The idea is to bound the polytopes into rectangles on the timed dimensions and then use sorting to skip performing operations (e.g. intersection) whose result is known to be an empty polytope. For the event-based type, we show that PTPM can be performed using algorithms that use a special data-structure containing time intervals. The algorithms make efficient use of sorting and binary search. We also adapt PTPM to do parametric identification for PTRE. For this, the labelling of the expert is given as a finite set of time intervals where interesting patterns occur. We find the set of parameters for which

we have a match at each of the given time intervals. To optimize parametric identification we define the notion of *decisive region*. For a given labelling time interval we can define a right-angled triangle on the two timed dimensions. All intervals outside this triangle can be ignored when doing parametric identification for the labelling interval. This technique improves the time complexity greatly.

The third contribution is regarding the non-parametric cases of TRE and STL. We introduce minimum, maximum and difference operators in TRE and provide theorems and linear-time algorithms for their TPM. We also introduce counting operators in STL that count the number of events in a sliding window or until some other event occurs. The algorithms we devised for these operators are efficient and run in linear time.

1.1 RELATED WORK

We split the related work into three main axes. The first is monitoring using timed formalisms (mainly TRE and STL) and their extensions and modifications. The second is analysis of timed formalisms augmented with parameters (mainly Parametric Timed Automata (PTA) and PSTL). The third is supervised learning of formal specifications (mainly TA and STL) from data. In addition to this, we give a brief description of Pareto optimality and list works that were precursors to our work described in Chapter 4. Here, in the categories under whose scope our contributions fall, we will refer to the corresponding chapters and give a brief description.

1.1.1 *Monitoring Using Timed Formalisms and Their Extensions*

The most well-known formalisms that specify system behaviours on the timed level, are TA [4], TRE [8, 9] and Metric Interval Temporal Logic (MITL) [57]. Out of these STL and its extensions have been shown to allow efficient linear time algorithms for monitoring signals and widely known. There are two paradigms applicable to monitoring using timed formalisms. The first applies to STL and deals with satisfaction/violation of formulae over a signal. The second is the paradigm of pattern matching which applies to TA and TRE. This involves finding all the time intervals where the input behaviour matches a given expression or automaton.

1.1.1.1 *Monitoring as Satisfaction of Formulae*

One of the first works on monitoring STL formulae is [56]. It discusses how a satisfaction signal indicating the Boolean satisfaction of a given formula over a signal can be inductively computed. In [34] it is shown how a real-valued measure called robustness can be computed for a

formula with respect to a signal. The robustness measure is intuitive in the sense that a positive robustness value implies satisfaction and a negative robustness implies violation. In [28], a signal-value freezing operators was introduced to extend STL in order to capture various dynamic aspects of oscillations. The disadvantage of adding the freezing operators is the polynomial time complexity with respect to size of the signal. STL has been extended in [14] with minimum and maximum operators to express some of the properties that motivated the introduction of the freezing operator while preserving linear-time complexity. A theoretical study of introduction of counting operators with analysis of expressive power and decidability properties is studied in [48]. But in this manuscript we are mainly concerned with monitoring. We will be describing in Chapter 6, how operators for counting events can be added in STL and efficiently monitored.

1.1.1.2 *Monitoring as Pattern Matching*

TPM is monitoring defined as follows: Given a behaviour and a specification (a pattern), detect all the time intervals when the pattern occurs in the behaviour. The set of these time intervals is called a match-set. TPM using TRE was first introduced in [72]. It presents an offline approach for inductive construction of match-set which is a finite union of zones (a special type of convex polygon). In [73], the concept of derivatives is extended to TRE in order to come up with an online timed pattern matching algorithm. In [18], an online approach for TPM of specifications with a state-based view (dealing with Boolean signals) of TA is proposed. First, a TRE specification is converted into a Timed Automaton. Then, a zone-based reachability computation is performed to retrieve all matching segments. In [17], Signal Regular Expressions (SRE) are introduced to deal with real-valued signals. In this work it has been shown how to compute real-valued robustness measure using induction over structure for SRE. Recall that such a real-valued robustness measure for STL has been introduced in [34]. In [74], TPM is adapted to Metric Compass Logic (MCL) specifications. TRE are extended and modified in [37] to define Event-Bounded Timed Regular Expressions (E-TRE). The special property of E-TRE is that their match-sets will only consist of finite number of intervals. Various measures such as integral, duration etc. can then be computed over these intervals. In Chapter 6 we will be discussing how TPM can be handled if we introduce operators that deal with computing extrema within time intervals. And in Chapter 3, we will show how to extend TPM to what is called Parametric Timed Pattern Matching (PTPM) for parametric versions of TRE. TPM can also be performed using TA in a manner similar to TRE, the only difference being the specifications are in form of TA. In [81], a classic Boyer-Moore string matching algorithm has been used as the basis for introduction of an optimized algorithm for TPM using TA. In [85], efficient online approach for TPM

based on automata-based skipping has been proposed. A detailed study on the complexity of TPM can be found in [11]. In [63], shape expressions were introduced which have different shapes (e.g. linear, sinusoidal, exponential etc.) with parameters as atomic predicates along with constraints over the parameters. These atoms can then be combined using the usual regular expression operators. One special feature is that they allow noisy semantics by combining matching semantics with statistical regression. A comprehensive guide to specification and detection of timed patterns using shape expressions can be found in [64]. Quantitative Regular Expressions (QRE) is a specification language suitable for capturing complex numerical queries over data streams. They have been used in various works about monitoring of the human heart [1–3].

1.1.2 *Parametric Analysis of Timed Formalisms*

Introduction of parameters into STL was first studied in [10] and given the name Parametric Signal Temporal Logic (PSTL). The focus was not on model checking but to perform parametric identification. This involves computing the set of parameters values for which the formula is satisfied over a signal. PTA have been introduced in [5], and undecidability of empty language and other model-checking problems has been established. Therefore, monitoring data over specifications expressed as PTA being easier lead to many recent studies. We first discuss parametric analysis of PSTL and then discuss Parametric Timed Pattern Matching (PTPM) for PTA. In Chapter 3, we will show how PTPM can be performed independently using Parametric Timed Regular Expressions (PTRE). Towards the end of Chapter 3, We will show how to adapt parametric identification to the case of PTRE to find parameter values that result in matches at given time intervals.

1.1.2.1 *Parametric Analysis of Parametric Signal Temporal Logic*

In [10, 16] the aim was to compute the validity domain of a parametric formula, i.e., the set of parameter valuations that makes the formula true on a (or a set of) signal. Parameters in PSTL can be used to express constraints both on values and time bounds. They are called space and timing parameters respectively in [16]. In [10] two different methods for computing validity domains for PSTL formulae are presented. The first method demonstrates how exact validity domains can be computed using quantifier elimination, in principle. Though complete and exact, the main drawback of this approach is the exponential worst case complexity in nested depth of formulae. The second method computes approximations of monotonic validity domains using query functions. This method forms the foundation of our contributions regarding monotonic validity domains. In Chapter 5 we will show how query functions can be used in combination with al-

gorithms described in Chapter 4 to find specifications that classify or characterize Electrocardiogram signals. Another method which computes validity domains recursively is proposed in [16]. Works which utilize PSTL for the tasks of clustering and classification are as follows. They concentrate on extracting features and computing a single solution rather than complete validity domains. In [76], template PSTL formulae are used to extract features. These features are then used in an unsupervised learning context to cluster traces. In [77], Hausdorff distance based on monotonic validity domains boundaries [55] is used as a distance metric for traces. Clustering was used to generate labelling and then construct specifications from monotonic PSTL templates. In [41], parametric identification is posed as an optimization problem. To facilitate solving this problem a tightness measure is introduced into the quantitative semantics of STL. The predicates and temporal operators are made smooth to render them differentiable. Now, efficient gradient-based optimization algorithms can be used to solve for parameter values. A recent work [51] proposes using computation graphs from the field of Machine Learning (ML). The inherent backpropagation mechanism of the computation graphs is used to learn values of parameters that maximize robustness on input signals. This approach allows integration with other ML problems. For example, in reward or loss functions. Handling of timing parameters is left for future work. In [43], the problem is slightly modified to find parameter values that make a monotonic formula satisfied over all behaviours of a dynamic model. Repeated falsification is performed over the model and the algorithm terminates when parameter values are found that produce a formula that does not have a counterexample. A detailed account of all approaches for mining STL can be found in a recent survey paper [23].

1.1.2.2 *Parametric Analysis of Parametric Timed Automata*

In [6], offline Parametric Timed Pattern Matching (PTPM) with timed words is performed using specifications expressed as PTA. PTPM involves finding the parameter values and the start and end values of intervals for which the specification matches the input data. In [82], a more efficient online method based on automata-based skipping is proposed. In [83], more expressive Parametric Timed Data Automata (PTDA) specifications with both timing and data parameters are analyzed using a symbolic monitoring algorithm. It needs to be mentioned that the three aforementioned works involve an event-based view of TA. Event-based view means that the TA deal with list of events along with their corresponding time stamps (not signals). In [84], we find a comprehensive study of PTPM using PTA.

1.1.3 *Learning Formal Specifications*

To set the context, we first list a few works that learn logical formulae and regular expressions. The problem of learning Linear-time Temporal Logic (LTL) formulae without any requirements of a priori information in the form of formula templates has been recently explored in [61]. In [36], we find algorithms to learn regular expressions from positive data (strings). In [31] and [50] TRE and LTL specifications respectively are mined from system traces using formula templates and event binding. Note that these works do not involve any parameters. They learn the structure of the formula or the expression that matches the training data. This is in contrast to learning Deep Neural Networks (DNN) which usually involves finding parameter values while the structure of the network is already decided. We have already discussed parametric analysis of timed formalisms. So, we now concentrate solely on works that involve learning both the structure as well as parameters of the formula or the automaton.

1.1.3.1 *Learning Full Signal Temporal Logic*

A data-driven approach for statistical learning of temporal logic properties is introduced in [20]. A sub-class of STL called reactive STL is investigated in [45]. The formulae in this sub-class are enumerated by defining a partial order and simulated annealing is used for parameter estimation. Another sub-class named inference PSTL is proposed in [46] for learning formulae that detect anomalies. They propose algorithms for both online and offline learning. Both the aforementioned works use directed acyclic graphs to organize formulae. A grid-based approach to learn STL formulae is proposed in [75]. Signals are analyzed using a grid over two-dimensional plots of values changing respect to time. They are clustered based on the grids they share and generate STL formulae based on these clusters. A decision tree approach combined with a restricted set of PSTL primitives using impurity measures to sift through them is proposed in [27]. In [26], a pruning algorithm is proposed to reduce size of the constructed trees. A cost is defined that is based on mis-classification rate and the width of trees. Progressively, smaller trees are constructed by adjusting the weights of mis-classification rate and the tree width. Out of the produced trees we can choose those performing the best on a validation set with respect to classification error. In [25], the focus is on online learning where the data arrives in stages. The mechanism of updating the STL specifications needs to be adapted to progressive arrival of data. This requires updating decision trees if the new STL formula is strongly estimated to hold well based on probabilistic criteria. An unsupervised learning approach using clusters has been proposed in [24]. A tree is produced where each node has a corresponding STL formula. The leafs have actual clusters of signals corresponding to

them. Distinguishing between signals is achieved using distance measures defined over them. In [59], monotonic PSTL formulae are enumerated using formula signatures. Computation of validity domain boundaries [55] is combined with checks for misclassification rate for parameter estimation. The resulting algorithm is used to search for an STL formula to classify traces. Another enumeration based method for classifying traces using robustness value based decision trees is proposed in [58]. Grid sampling is used to estimate timing parameters. In [42], parameter estimation for PSTL is formulated as multi-objective optimization with respect to robustness. For inferring the structure of STL formulae in the absence of templates, they propose an incremental construction approach. Evolutionary Algorithms are inspired by natural evolution of organisms. In [62], struture of STL formula is built using an evolutionary algorithm and parameter values are estimated using optimization. Another genetic algorithm approach that learn only past fragment of STL is presented in [12]. A comprehensive survey on learning STL can be found in [23].

1.1.3.2 *Learning Timed Automata*

One of the first works on learning real-time automata (a class of one-clock timed automata) is [78]. In [79], one can find theoretical results and complexity analysis for learning various classes of timed automata. An algorithm called real-time identification (RTI) has been proposed in [80] for efficiently identifying deterministic real-time automata from labelled data. It is based on evidence-driven state-merging (EDSM) algorithm for identification of DFA. A method termed timed k-tail algorithm has been proposed in [65] for learning TA. It extends the k-tail automaton learning technique with the ability to generate clocks, resets and guards. In [67], the problem of synthesizing a timed automata from timed scenarios (not timed words) has been considered. A timed scenario is a sequence of event names along with a set of constraints between the times at which the named events occur. An automaton is built which is not only correct with respect to the timed scenarios but also with optimal number of clocks. In [70], a very different genetic programming approach for learning timed automata is explored. The system under learning is assumed to be deterministic. The approach is search-based and selection is based on fitness and altering is done using mutation and crossover. An approach using SMT solving to learn automata that are consistent with the observations given as a set of timed traces is proposed in [71]. Additionally, a set of restrictions can be imposed on the learnt models to avoid over generalization.

We now describe approaches for active learning which assumes interaction using queries. In [7], the L^* algorithm was introduced which learns an unknown regular set using queries for membership and equivalence. In [39], this idea was extended for the timed case

for a sub-class of TA called deterministic event-recording automata. In [52], an efficient polynomial time algorithm named TL* was proposed for learning event-recording automata. We first learns a DFA that accepts the untimed version of the timed language. Then passively refine the DFA by adding time constraints. It learns from queries involving guarded words which are lists of pairs of symbols and clock guards rather than timed words. In [53], the TL* algorithm was used to learn event-recording automata that can be used in compositional techniques such as assume-guarantees reasoning. In [40], the authors discuss active learning extended to reset-free event-recording automata, where some transitions may reset no clocks. Their algorithm and datastructures rely on the notion of invalidity to allow detection and pruning of reset hypotheses that contradict observations. This lead to an efficient active-learning procedure. In [69], the problem of learning deterministic one-clock automata is considered in the framework of Probably Approximate Correct (PAC) learning. The authors use PAC-style equivalence queries to replace exact equivalence queries. In addition, the idea of comparators is integrated into the framework in order to reduce the number of equivalence queries. In [68], algorithms for model checking and controller synthesis of TA have been presented. A timed automaton model is viewed as a paralled composition of a large finite automaton and a relatively small timed automaton. Active automata learning algorithms are used to learn finite automata approximations of the timed component. This reduces the problem to finite-state model checking or finite-state controller synthesis.

1.1.3.3 *Mining Expressions*

Here, we list works that deal with mining specifications that are expressions. A tool named QMine for mining QRE specifications has been presented in [54]. Shape expressions introduced in [63] can also be used for mining specifications. We recall that shape expressions are regular expressions with different kinds of shapes as atomic predicates. In [21], we find a method for mining shape expressions from positive examples. The method combines algorithms for linear regression, clustering and learning DFA. After the traces are abstracted as a sequence of events representing different classes of linear segments the DFA learning algorithm is used to learn the regular expression pattern. in [22], a tool named ShapeIt is introduced for mining specifications from behaviours of cyber-physical systems. The focus is limited to linear shape expressions that have only parameterized lines as atomic symbols. The architecture of the tool is described along with demonstration of its capabilities over several case studies.

1.1.4 *Pareto Fronts*

Algorithms are used to automatically analyze models of complex systems with the help of computers. This analysis frequently involves optimizing the system with respect to a given criterion and the allowed values of parameters. From now on we will only consider minimization for simplicity of presentation (maximization can be converted to minimization by negation). Multi-criteria optimization arises when we need to optimize with respect to multiple criteria. The set of solutions to such an optimization problem would contain a subset called a Pareto optimal set. A Pareto optimal set consists of all solutions that cannot be improved with respect to one criterion without worsening with respect to some other criterion. A vector is said to Pareto dominate another vector if it is better than the latter in all dimensions. Multi-criteria optimization methods can be broadly divided into two classes. One class uses mathematical optimization techniques to explore the Pareto front. The other class uses evolutionary algorithms which are inspired by ideas from Darwin's theory of natural evolution. In evolutionary algorithms, populations evolve over generations and only non-dominated solutions remain or survive. Concepts, methods and examples of multi-criteria optimization in engineering applications have been explained in detail in [30]. An alternative way of solving the problem is to do it through use of constraint solving. This is particularly useful when the problem cannot be described explicitly using mathematical equations. For example, when we are analyzing the performance of programs and hardware. This alternative approach has been implemented in [49] to devise an algorithm for computing approximate Pareto fronts. A knee tree data structure is used to represent the state of the algorithm with the goal of minimizing the number of queries. In [10], the idea of Pareto domination was applied to parametric identification of PSTL.

PRÉLIMINAIRES

Dans ce chapitre, nous présentons le contexte technique et une brève description des concepts qui constituent la base de notre travail. Nous commençons par les expressions régulières et les automates finis déterministes et décrivons comment ils peuvent être utilisés pour décrire des motifs dans un texte. Ensuite, nous expliquons comment la logique temporelle linéaire peut être utilisée pour décrire les changements qui se produisent dans le temps. Par exemple, en utilisant cette logique, nous pouvons affirmer qu'une certaine exigence de sécurité est toujours respectée ou qu'une bonne chose finira par se produire. Nous définissons ensuite formellement les signaux à valeur réelle et les mots temporisés. Les signaux peuvent être utilisés pour représenter les changements d'une valeur dans le temps. Les mots temporisés sont utilisés pour exprimer des séquences d'événements avec des horodatages. Nous suivons la théorie de la logique temporelle et montrons comment elle aboutit à la logique temporelle des signaux, qui est un formalisme populaire pour spécifier les propriétés des signaux. Ensuite, nous examinons comment les expressions régulières temporisées peuvent étendre les expressions régulières de diverses manières pour décrire les aspects temporels des signaux, des mots temporisés et de leur combinaison. Nous terminons par une brève description d'un algorithme important permettant de calculer approximativement les fronts de Pareto à partir de requêtes.

2

PRELIMINARIES

In this chapter we first have a brief discussion on strings, regular expressions and finite automata. Then, we talk about the syntax and semantics of timed specifications. We close the chapter with a description of an algorithm that approximately computes Pareto fronts from queries.

2.1 STRINGS AND LANGUAGES

In this section we present definitions related to automata theory and regular languages, such as strings, languages and operations over them.

Alphabet and Strings. An alphabet is a set of symbols. We generally denote it by Σ. A finite sequence of symbols from the alphabet is called a string. A simple example of an alphabet is the roman alphabet $\Sigma = \{a, b, c, ..., x, y, z\}$ with twenty-six symbols. Using a finite alphabet one can construct an infinite number of strings. Some examples of strings over roman alphabet are *(s,t,r)*, *(g,o,o,d)* or *(d,e,c,e,n,t)*. The empty string is denoted using the symbol ϵ. Given a string s, its length $|s|$ is defined as the number of symbols it contains. By default, the empty string ϵ is defined to have a length of zero. The set of all finite words over an alphabet Σ is denoted by Σ^*. The set of all non-empty finite words over Σ is denoted by Σ^+. Given two strings $s = (a_1, ..., a_m)$ and $t = (b_1, ..., b_n)$, their concatenation $s \cdot t$ is defined as $s \cdot t = (a_1, ..., a_m, b_1, ..., b_n)$. The empty word ϵ is the identity element with respect to concatenation operation. This means that given a string $s \in \Sigma^*$, it holds that $s \cdot \epsilon = \epsilon \cdot s = s$. Concatenation operation also respects associativity. Given any strings $s, t, r \in \Sigma^*$, the condition $(s \cdot t) \cdot r = s \cdot (t \cdot r)$ always holds.

Languages. A language L over an alphabet Σ is a subset of Σ^*. All the usual set theoretic operations such as intersection $L_1 \cap L_2$, union $L_1 \cup L_2$ and complementation $\overline{L}$ have the usual semantics.

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

Concatenation operation can be naturally extended to languages and is defined as follows.

$$L_1 \cdot L_2 = \{s_1 \cdot s_2 \mid s_1 \in L_1 \text{ and } s_2 \in L_2\}$$

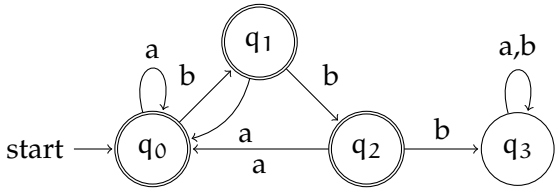


Figure 1: An example DFA over $\Sigma = \{a, b\}$

The exponent notation can also be extended to languages. The n-th power of a language L is defined inductively as $L^n = L^{(n-1)} \cdot L$ if n is a positive integer and the base case is $L^0 = \{\epsilon\}$. The unary operations Kleene Star (L^*) and Kleene Plus (L^+) over a language L are defined as follows:

$$L^* = \bigcup_{n \geqslant 0} L^n \text{ and } L^+ = \bigcup_{n \geqslant 1} L^n$$

2.2 REGULAR EXPRESSIONS AND AUTOMATA

For a given alphabet Σ, regular expressions define a sub-class of languages over it. This sub-class is called the class of regular languages. The syntax of regular expressions over Σ is defined using the following rules:

$$\varphi := \emptyset \mid \epsilon \mid a \mid \varphi_1 \cap \varphi_2 \mid \varphi_1 \cup \varphi_2 \mid \varphi^*$$

Here, $a \in \Sigma$. The semantics of regular expressions is defined in terms of regular languages that correspond to these expressions. It is defined inductively as follows:

$$\begin{aligned} [\![\emptyset]\!] &= \emptyset & [\![\varphi_1 \cup \varphi_2]\!] &= [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\ [\![\epsilon]\!] &= \{\epsilon\} & [\![\varphi_1 \cap \varphi_2]\!] &= [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\ [\![a]\!] &= \{a\} & [\![\varphi^*]\!] &= [\![\varphi]\!]^* \end{aligned}$$

Now, we describe Deterministic Finite Automata (DFA). A DFA $\mathcal{D}$ is defined as a 5-tuple $\mathcal{D} := (\Sigma, Q, \delta, q_s, F)$. Here, Σ is a finite alphabet, Q is a finite set of states, $q_s \in Q$ is called the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of final states. As an example, see Figure 1, which shows an automaton $(\{a, b\}, \{q_0, q_1, q_2, q_3\}, \delta, q_0, \{q_0, q_1, q_2\})$. Starting in the initial state, DFA read symbols in the input string one by one and make transitions from one state to another following the transition function δ. If the automaton finishes at a final state after reading the input string, then the string is said to be accepted by it. The set of strings accepted by an automaton D is the language $L \in \Sigma^*$ it recognizes, denoted as $[\![\mathcal{D}]\!]$. A language is called DFA-recognizable if there exists a DFA that recognizes it.

From the well known Kleene's theorem we know that regular expressions have the same expressive power as DFA. What this means

is that the the class of regular languages is the same as the class of DFA-recognizable languages.

2.3 LINEAR-TIME TEMPORAL LOGIC

In [66], LTL was proposed for describing temporal properties of infinite duration. On top of propositional logic, temporal modalities such as next (X), eventually (F), always (G) and until (U) are introduced. We discuss only the future fragment of LTL. The meaning of these modalities over a logical formula φ in natural language is as follows:

$X\varphi$	φ holds at the next time point.
$F\varphi$	φ holds at some time point in future.
$G\varphi$	φ holds at all time points in future.
$\varphi_1 U \varphi_2$	φ_2 holds at some future time point and φ_1 holds for all future time points until φ_2 holds.

We now describe the formal syntax and semantics of the future fragment of LTL. Let $N = [0, n)$ be an interval of integer time line. Here, n can be either a positive integer or infinity (∞). We are given a set of propositions $P = \{p_1, ..., p_m\}$. A discrete-time behaviour $w : N \to \mathcal{B}^m$ is a function that assigns a Boolean value {FALSE,TRUE} for each proposition $p \in P$ at every time step. The length of the behaviour w is defined as $|w| = n$. The syntax of the future fragment of LTL over P is as follows:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 U \varphi_2$$

The satisfaction relation $\vdash$ of a formula φ with respect to a discrete behaviour w (either finite or infinite) is inductively defined as follows:

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

The next (X), eventually (F) and always (G) operators can be derived using the following equivalences:

$$\begin{aligned}
X\varphi &= \text{FALSE } U\ \varphi \\
F\varphi &= \text{TRUE } U\ \varphi \\
G\varphi &= \neg F \neg \varphi
\end{aligned}$$

2.4 SIGNALS, TIMED WORDS AND TIME-EVENT SEQUENCES

In this section we talk about different kinds of observations over dense-time. These include signals, timed words and time-event sequences. Out of these three, timed words and time-event sequences can be considered equivalent.

2.4.1 *Signals*

Let us assume that the behaviour of a system under observation is captured by a set of m variables $V = \{x_1, x_2, ..., x_m\}$. The domain of valuation of V is denoted by $\mathbb{D} = \mathbb{D}_1 \times \mathbb{D}_2 \times ... \times \mathbb{D}_m$. The set of real numbers is denoted by $\mathbb{R}$. The Boolean domain is represented as $\mathbb{B} = \{\text{FALSE}, \text{TRUE}\}$. The time domain $\mathbb{T}$ is an interval of $\mathbb{R}_{\geqslant 0}$ which is the set of non-negative reals. A signal (or behaviour) w is a function mapping $\mathbb{T}$ to $\mathbb{D}$ that captures the evolution over time of the system under observation. If $\mathbb{T} = [s, s')$, the length of the signal is defined as $|w| = s' - s$ and usually we take $s = 0$. For $i \in [1, ..., m]$, we use w_{x_i} to denote the projection of signal w over the variable x_i. Signals with $\mathbb{D} = \mathbb{B}^m$ are called Boolean signals and those with $\mathbb{D} = \mathbb{R}^m$ are real-valued signals. The value of a signal at time t is denoted as $w[t]$. The segment of the signal over the interval $[t, t']$ is denoted as $w[t, t']$.

Definition 2.4.1 (Signal Distance)**.** *The (uniform norm) distance between signals u and v over the same set of variables V, denoted $d(u, v)$, is defined by $d(u, v) = \sup_{t \in \mathbb{T}} \max_{x \in V} |u_x[t] - v_x[t]|$ if u and v have the same temporal domain $\mathbb{T}$, otherwise $d(u, v) = +\infty$.*

We now give a formal definition of signals and introduce the notion of finite variability applicable to them. Let us consider a set of real variables $X = \{x_1, x_2, ..., x_l\}$ and a set of propositional variables $P = \{p_1, p_2, ..., p_n\}$.

Definition 2.4.2 (Signals)**.** *A signal is a function $w : \mathbb{T} \to \mathbb{R}^l \times \mathbb{B}^n$. At each time point in $\mathbb{T}$ the signal assigns real values to variables $x \in X$ and Boolean values to variables $p \in P$. They are called real-valued and Boolean if they are of the form $x : \mathbb{T} \to \mathbb{R}^l$ and $x : \mathbb{T} \to \mathbb{B}^n$ respectively.*

The value of x at time t is denoted by $x[t]$ and its i^{th} coordinate by $x_i[t]$. Abusing the notation, we often use the variable names $x_1, \ldots, x_l, p_1, \ldots, p_n$ both for the values of the signal and in the expression.

Definition 2.4.3 (Finite Variability)**.** *Let $w : \mathbb{T} \to \mathbb{R}^l \times \mathbb{B}^n$ be a signal. Let us also assume that we are given a set of atomic predicates Π of the form $(x \geqslant c)$ or $(x \leqslant c)$ where $x \in X$ and $c \in \mathbb{R}$. Each of these predicates in Π produces a Boolean signal $t \mapsto x(t) \geqslant c$. The signal w is said to be of finite variability if for every propositional variable/atomic predicate in $P \cup \Pi$ the corresponding Boolean signal has a finite set of discontinuities.*

From here on, the Boolean signals we talk about are assumed to be of finite variability. Also, the real-valued signals considered are assumed to be piecewise affine so that the Boolean signals they produce using atomic predicates (like $x \geqslant c$) are also of finite variability. The property of finite variability of signals is required for the validity of our theorems and algorithms.

2.4.2 *Timed Words and Time-Event Sequences*

In this subsection, we provide the definitions of timed words and time-event sequences. We also give examples for these and show how they are equivalent.

Definition 2.4.4 (Timed Words)**.** *A timed word of length l over an alphabet Σ is a sequence $\omega = t_1 a_1 ... t_l a_l$ with $a_i \in \Sigma, t_i \in \mathbb{R}_{\geqslant 0}$ and $0 \leqslant t_1 \leqslant ... \leqslant t_l$. Here, t_i represents the date at which the event a_i occurs.*

Definition 2.4.5 (Time-Event Sequences)**.** *Time-event sequences consist of sequences of passage of time and symbols from the event alphabet Σ.*

The set of all time-event sequences over an event alphabet Σ is denoted as $\mathcal{TE}(\Sigma)$. When the alphabet Σ is clear from the context we simply use $\mathcal{TE}$. Consider the following time-event sequence over $\Sigma = \{a, b, c\}$:

$$0.5 \cdot ba \cdot 6.2 \cdot baca \cdot 3.4 \cdot a$$

The equivalent timed word representation of this time-event sequence is as follows:

$$(0.5, b), (0.5, a), (6.7, b), (6.7, a), (6.7, c), (6.7, a), (10.1, a)$$

We now define the length of a time-event sequence:

Definition 2.4.6 (Length)**.** *The length $\lambda : \mathcal{TE} \to \mathbb{R}_{\geqslant 0}$ is the projection on $\mathbb{R}_{\geqslant 0}$ obtained by mapping elements of Σ^* to 0.*

The length of the aforementioned example of time-event sequence is calculated as follows:

$$\lambda(0.5 \cdot ba \cdot 6.2 \cdot baca \cdot 3.4 \cdot a) = 10.1$$

2.5 METRIC INTERVAL TEMPORAL LOGIC

MITL is a logic well suited for describing timed behaviours represented as Boolean signals. As defined in the previous section, $\mathbb{T}$ represents the time domain. We are given a set of propositions $P = \{p_1, p_2, ..., p_m\}$. A Boolean signal $w : \mathbb{T} \to \mathbb{B}^m$ assigns a Boolean value

for each proposition $p \in P$ at each time point in $\mathbb{T}$. The syntax of MITL can be expressed using the grammar given below:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathsf{U}_I \varphi_2$$

Here, $p \in P$ and I is an interval of time values. The satisfaction relation $\vdash$ of an MITL formula φ over Boolean signal w is inductively defined as follows:

$$\begin{aligned}
(w,t) \vdash p &\leftrightarrow w_p[t] = \textsc{true} \\
(w,t) \vdash \neg\varphi &\leftrightarrow (w,t) \nvdash \varphi \\
(w,t) \vdash \varphi_1 \wedge \varphi_2 &\leftrightarrow (w,t) \vdash \varphi_1 \text{ and } (w,t) \vdash \varphi_2 \\
(w,t) \vdash \varphi_1 \mathsf{U} \varphi_2 &\leftrightarrow \exists t' \in \mathbb{T}.(w,t') \vdash \varphi_2 \text{ and} \\
&\quad \forall t'' \in (t,t').(w,t'') \vdash \varphi_1 \text{ and } t' - t \in I
\end{aligned}$$

The operations eventually (F) and always (G) can be derived using the following equivalences:

$$\begin{aligned}
\mathsf{F}_I \varphi &= \textsc{true}\ \mathsf{U}_I\ \varphi \\
\mathsf{G}_I \varphi &= \neg \mathsf{F}_I \neg \varphi
\end{aligned}$$

2.6 SIGNAL TEMPORAL LOGIC

Signal Temporal Logic (STL) allows reasoning over dense-time real-valued signals with $\mathbb{D} = \mathbb{R}^m$. STL can be connected to MITL using a finite set of predicates (Booleanizers) $M = \{\mu_1, \mu_2, ..., \mu_k\}$. For each $j \in [1, .., k]$, the predicate μ_j is of the form $\mu_j \equiv f_j(x_1, x_2, ..., x_m) \geqslant 0$ where f_j is a real-valued function. The syntax and semantics of STL is identical to that of MITL except for the set of predicates in M. The satisfaction relation $\vdash$ for each $\mu_j \in M$ can be defined as follows:

$$(w,t) \vdash \mu_j \leftrightarrow f_j(x_1[t], ..., x_m[t]) \geqslant 0$$

The kind of predicates that are usually used are threshold predicates like the ones given below:

$$\mu_1 \equiv (x_1 \geqslant 5) \text{ and } \mu_2 \equiv (x_1 \leqslant 3)$$

The satisfaction relation $\vdash$ of MITL and STL defined previously is Boolean semantics and is of a qualitative kind. It just tells us whether the formula is satisfied or violated. It does not give us a quantitative measure of the extent to which we have a satisfaction or violation. The *satisfaction signal* $\chi(\varphi, w, .)$ is defined as follows:

$$\forall t \in \mathbb{T},\ \chi(\varphi, w, t) = \begin{cases} \textsc{true}, & \text{if } (w,t) \vdash \varphi \\ \textsc{false}, & \text{otherwise} \end{cases}$$

Quantitative Semantics. Given an STL formula φ, and a signal $w : \mathbb{T} \to \mathbb{R}^m$, the quantitative semantics $\rho(\varphi, w, t)$ is defined inductively as follows:

$$\begin{aligned}
\rho(\mu, w, t) &= f(x_1[t], ..., x_m[t]) \text{ for } \mu \equiv f(x_1, ..., x_m) \geqslant 0 \\
\rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\
\rho(\varphi_1 \wedge \varphi_2, w, t) &= \min\{\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)\} \\
\rho(\varphi_1 \mathcal{U}_I \varphi_2, w, t) &= \sup_{t \in t'+I} \min\{\rho(\varphi_2, w, t'), \inf_{t'' \in [t, t']} \rho(\varphi_1, w, t'')\}
\end{aligned}$$

Here, $t' + I = \{t' + t''' \mid t''' \in I\}$.

The above quantitative metric is called robustness estimate in [33]. Two properties of robustness estimate are given in the aforementioned paper to justify its introduction. The first says that whenever $\rho(\varphi, w, t)$ is non-zero the sign indicated the satisfaction status.

Theorem 2.6.1 (Soundness). *Let φ be an STL formula, w a signal, and t indicates time.*

$$\begin{aligned}
\rho(\varphi, w, t) > 0 &\implies (w, t) \vdash \varphi \\
\rho(\varphi, w, t) < 0 &\implies (w, t) \nvdash \varphi
\end{aligned}$$

The second states that if a signal w satisfies an STL formula φ at time t, any other signal w' whose point-wise distance from w is smaller than $\rho(\varphi, w, t)$ also satisfies φ at time t.

Theorem 2.6.2 (Correctness). *Let φ be an STL formula, w and w are signals defined over the same time domain $\mathbb{T}$, and $t \in \mathbb{T}$.*

$$(w, t) \vdash \varphi \text{ and } \|w - w'\|_\infty < \rho(\varphi, w, t) \implies (w', t) \vdash \varphi$$

Similar to the satisfaction signal, one can define a *robustness signal* over the time domain $\mathbb{T}$. Given a signal w and an STL formula φ, the robustness signal of φ with respect to w, is the signal $\rho(\varphi, w, .)$.

2.7 EXTENDED SIGNAL TEMPORAL LOGIC

Here, we explain how we can extend STL with Min and Max operators as introduced in [14]. We first present the motivation for introducing these operators using examples. Then, we show how the use of these operators extends STL.

Stabilization example: In this example, we consider the property of stabilization towards an unknown value. Consider the natural language phrase "Starting from now, the signal x will stay within 0.05 units of some value for at least 200 time units". In order to check this property we can do the following. First compute the maximum value attained by x in the interval $[0, 200]$. Then, compute the minimum value attained by x in the same interval. Finally, compute the difference between these extremal values and check if it is less than or equal to $0.1 = 2 * 0.05$.

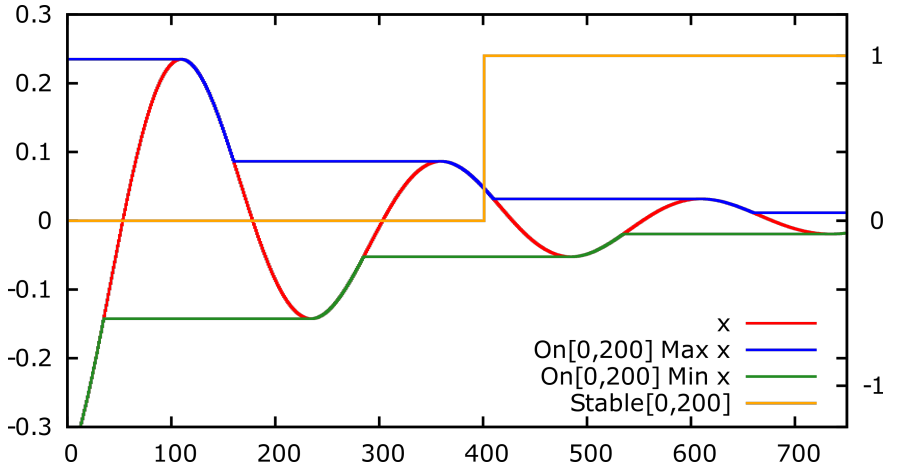


Figure 2: Damped oscillation $x(t)$ and its maximum and minimum over the window $[t, t + 200]$.

For clarity of presentation, we use two separate operators to describe the aforementioned property. The first is the operator $On_{[a,b]}$ which expresses the time window specified using a and b. Operators Min and Max are used to denote the computation of minimum and maximum respectively over a given time window. The property can be expressed using formal syntax as follows:

$$On_{[0,200]} Max\, x - On_{[0,200]} Min\, x \leqslant 0.1$$

Figure 2 (from [14]) illustrates the checking of the property over a signal x varying over time and shown in red. It performs damped oscillations with period of 250 time units. The values of maximum and minimum computed over a sliding window of $[t, t + 200]$ are shown in blue and green respectively. Finally, we see in orange the satisfaction (Boolean) signal for the whole property.

Local maximum example: Let us consider the property "The value of a signal x at the current point is the maximum or the minimum when considered over an interval encoding its neighbourhood". A concrete example is as follows:

$$x \geqslant On_{[0,85]} Max\, x$$

In natural language, this means that, "The current value of x is a local maximum when considered over the interval [0,85] with respect to the current instant of time."

Figure 3 (from [14]) illustrates this property over a signal x (shown in red) oscillating with a period of 250 time units. In blue we see the maximum of x computed over the sliding window $[t, t + 85]$. In orange we have the satisfaction signal for the whole property. Basically, this property is true at time points where there exists a local maximum and false otherwise.

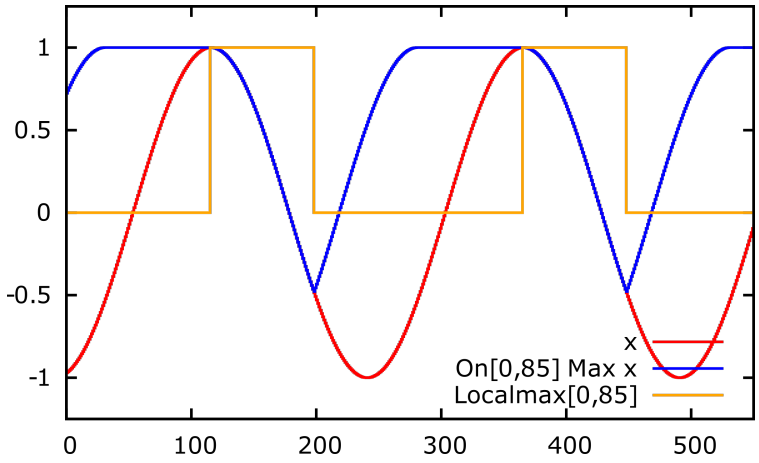


Figure 3: Sine wave $x(t)$, its maximum over the window $[t, t+85]$, and whether $x(t)$ is a local maximum on the interval $[t, t+85]$.

Another stabilization example (using Until): Let us say, we do not want to explicitly state the interval over which we want expresses stabilization. But we want the stabilization to happen before some event occurs (some signal q becomes true). We can express this by tweaking and defining a quantitative version of the usual Until operator. Consider the following property:

$$(Max\, x\ U\ q) - (Min\, x\ U\ q) \leqslant 0.1$$

This means, "Signal x is stabilizes within 0.05 units of an unknown value until/before q becomes true." And, it can be checked by finding the time point closest to the current one where q becomes true. Then, we can compute the maximum and minimum over the time interval starting from the current instant and ending when q turns true.

Let us now move on to describe the relevant parts of the formal syntax of extended STL from [14].

$$\begin{aligned} \varphi &:= \quad c \mid \mathbf{x} \mid f(\varphi_1 \cdots \varphi_n) \mid On_{[a,b]}\psi \mid \psi\ U^{d}_{[a,b]}\varphi \\ \psi &:= \quad Min\, \varphi \mid Max\, \varphi \end{aligned}$$

Here, c is a real-valued constant; x stands for an input signal x; f stands for a real-valued function (e.g. absolute value, sum, difference etc.);

The corresponding semantics of the above is real-valued i.e. we process real-valued signals (like x) and produce real-valued signals corresponding to the properties. We first describe the semantics of the fragment of the syntax excluding the new Until operator which is dealt with separately later. The semantics of the formulae (except U) of type φ is as given below:

$$[\![x]\!](t) = x(t) \qquad [\![c]\!](t) = c \qquad [\![On_{[a,b]}\psi]\!](t) = [\![\psi]\!]([t+a, t+b])$$
$$[\![f(\varphi_1 \ldots \varphi_n)]\!](t) = f([\![\varphi_1]\!](t) \ldots [\![\varphi_n]\!](t)$$

The semantics of the formulae of type ψ is defined over time intervals and is as follows:

$$[\![\mathit{Min}\ \varphi]\!][t_1, t_2] = \mathtt{min}_{[t_1,t_2]}[\![\varphi]\!] \qquad [\![\mathit{Max}\ \varphi]\!][t_1, t_2] = \mathtt{max}_{[t_1,t_2]}[\![\varphi]\!]$$

The Boolean operators ($\wedge$, $\vee$ and $\neg$), eventually operator (F) and always operator (G) of STL can be re-described as follows:

$$\begin{aligned} \varphi_1 \wedge \varphi_2 &\equiv \mathtt{min}\{\varphi_1, \varphi_2\} & \varphi_1 \vee \varphi_2 &\equiv \mathtt{max}\{\varphi_1, \varphi_2\} \\ \neg\varphi &\equiv 1 - \varphi & \mathsf{F}_{[t_1,t_2]}\varphi &\equiv \mathit{On}_{[t_1,t_2]}\mathit{Max}\ \varphi \\ \mathsf{G}_{[t_1,t_2]}\varphi &\equiv \mathit{On}_{[t_1,t_2]}\mathit{Min}\ \varphi \end{aligned}$$

Let us finally move to the new Until operator and show how to derive the STL Until using it. The new Until $\psi\ \mathsf{U}^{d}_{[a,b]}\varphi$ is interpreted as follows. For the current time instant t, we associate a time interval ending when ϕ becomes non-zero if it indeed exists. Otherwise, we output the default value d. For the first case where the interval exists, we compute ψ over it and output it. The Until of STL can be re-described as follows:

$$\varphi_1\ \mathsf{U}^{\mathsf{STL}}_{[a,b]}\ \varphi_2\ =\ (\mathit{Min}\ \varphi_1)\ \mathsf{U}^{0}_{[a,b]}\ \varphi_2$$

2.8 TIMED REGULAR EXPRESSIONS

Timed Regular Expressions (TRE) are timed extensions of regular expressions that incorporate duration into the language. Unlike STL, for TRE we talk about pattern matching rather than satisfaction. Timed pattern matching using TRE means finding all segments of a given observation that match the given TRE. In this section, we talk about three types of TRE. The first type, TRE (state-based), is suited for detecting patterns in Boolean signals. The second type, Signal Regular Expressions (SRE), is a slight modification of the first type that can detect patterns in general real-valued signals. The third type, TRE (event-based), is tailored for timed words rather than signals.

2.8.1 *Timed Regular Expressions (State-Based)*

We are given a set of propositions $P = \{p_1, p_2, ..., p_m\}$. A Boolean signal $w : \mathbb{T} \to \mathbb{B}^m$ that assigns a Boolean value for each proposition $p \in P$ at each time point in $\mathbb{T}$. We now define the syntax and semantics of TRE (state-based).

Definition 2.8.1 (Syntax)**.** *The syntax of TRE (state-based) is given by the grammar below:*

$$\varphi := \epsilon \mid p \mid \overline{p} \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi^* \mid \langle\varphi\rangle_I$$

Here, $p \in P$ *and* I *is an interval inside* $\mathbb{R}_{\geqslant 0}$.

Exponential notation is defined over concatenation as $\varphi^0 = \epsilon$ and $\varphi^{k+1} = \varphi^k \cdot \varphi$. For MITL the semantics is defined in terms of a satisfaction relation. It is of the form $(w, t) \vdash \varphi$. This indicates that the formula φ is satisfied by the signal w starting from the time point t. For TRE, instead of satisfaction, we define matching over an interval defined by two time points t, t' with $t \leqslant t'$. In this context $(w, t, t') \models \varphi$ means that $w[t, t']$ matches the semantics of expression φ. Now, we formally define matching semantics of TRE (state-based).

Definition 2.8.2 (Semantics). *The matching relation $\models$ of a timed regular expression φ by a signal w, relative to start time t and end time $t' \geqslant t$ is defined as follows:*

$$
\begin{aligned}
(w, t, t') &\models \epsilon && \leftrightarrow && t = t' \\
(w, t, t') &\models p && \leftrightarrow && t < t' \textit{ and } \forall t''.t < t'' < t' \rightarrow w_p[t''] = 1 \\
(w, t, t') &\models \overline{p} && \leftrightarrow && t < t' \textit{ and } \forall t''.t < t'' < t' \rightarrow w_p[t''] = 0 \\
(w, t, t') &\models \varphi_1 \cdot \varphi_2 && \leftrightarrow && \exists t''.(w, t, t'') \models \varphi_1 \textit{ and } (w, t'', t') \models \varphi_2 \\
(w, t, t') &\models \varphi_1 \vee \varphi_2 && \leftrightarrow && (w, t, t') \models \varphi_1 \textit{ or } (w, t, t') \models \varphi_2 \\
(w, t, t') &\models \varphi_1 \wedge \varphi_2 && \leftrightarrow && (w, t, t') \models \varphi_1 \textit{ and } (w, t, t') \models \varphi_2 \\
(w, t, t') &\models \varphi^* && \leftrightarrow && \exists k \geqslant 0.(w, t, t') \models \varphi^k \\
(w, t, t') &\models \langle \varphi \rangle_I && \leftrightarrow && t' - t \in I \textit{ and } (w, t, t') \models \varphi
\end{aligned}
$$

The set of segments of w that match a given expression φ is defined as a match-set.

Definition 2.8.3 (Match-Set). *For any signal w and expression φ, we let*

$$\mathcal{M}(\varphi, w) := \{(t, t') \in T \times T : (w, t, t') \models \varphi\}$$

2.8.2 *Signal Regular Expressions*

Signal Regular Expressions (SRE) are a modification of TRE that are tailored to deal with real-valued signals rather than Boolean signals. The building blocks of SRE are atomic constraints such as $x \geqslant 5$. The rest of the features are the same as TRE. We consider a signal w which is over variables $V = \{x_1, x_2, ..., x_m\}$. We now formally define the syntax and semantics of SRE.

Definition 2.8.4 (Syntax). *The syntax of SRE is defined using the grammar below:*

$$\varphi := \emptyset \mid \epsilon \mid x \geqslant c \mid x \leqslant c \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \cdot \varphi_2 \mid \varphi^* \mid \langle \varphi \rangle_I$$

Here, $x \in V$,$c \in \mathbb{R}$, and I is an interval of $\mathbb{R}_{\geqslant 0}$ with integer bounds.

The qualitative semantics of an expression φ with respect to a signal w denoted by symbol μ is a Boolean valued function that indicates whether w matches φ.

Definition 2.8.5 (Qualitative Semantics). *The semantics* $\mu(\varphi, w)$ *of an expression* φ *with respect to a signal* w *is defined as follows:*

$$
\begin{aligned}
\mu(\emptyset, w) = 1 &\leftrightarrow \bot \\
\mu(\epsilon, w) = 1 &\leftrightarrow |w| = 0 \\
\mu(x \geqslant c, w) = 1 &\leftrightarrow |w| > 0 \text{ and } \forall t \in [0, |w|), w_x[t] \geqslant c \\
\mu(x \leqslant c, w) = 1 &\leftrightarrow |w| > 0 \text{ and } \forall t \in [0, |w|), w_x[t] \leqslant c \\
\mu(\varphi_1 \vee \varphi_2, w) = 1 &\leftrightarrow \mu(\varphi_1, w) \text{ or } \mu(\varphi_2, w) \\
\mu(\varphi_1 \wedge \varphi_2, w) = 1 &\leftrightarrow \mu(\varphi_1, w) \text{ and } \mu(\varphi_2, w) \\
\mu(\varphi_1 \cdot \varphi_2, w) = 1 &\leftrightarrow \exists uv = w, \mu(\varphi_1, u) = 1 \text{ and } \mu(\varphi_2, w) = 1 \\
\mu(\varphi^*, w) = 1 &\leftrightarrow \exists k \geqslant 0, \mu(\varphi^k, w) = 1 \\
\mu(\langle\varphi\rangle_I, w) &\leftrightarrow \mu(\varphi, w) = 1 \text{ and } |w| \in I
\end{aligned}
$$

Here, uv *stands for concatenation of signals* u *and* v*.*

Like we did for regular expressions, we can define a language for an SRE φ denoted as $L(\varphi) = \{w \mid \mu(\varphi, w) = 1\}$. Similar to STL one can introduce a quantitative semantics for SRE. Given an expression φ and a signal w, one can define over them, a robustness function ρ that returns a real value ($\mathbb{R} \cup \{+\infty, -\infty\}$) which indicates how robustly w matches φ. We now formally define the quantitative semantics.

Definition 2.8.6 (Quantitative Semantics). *The robustness* $\rho(\varphi, w)$ *of an expression* φ *with respect to a signal* w *is defined inductively as follows:*

$$
\begin{aligned}
\rho(\emptyset, w) &= -\infty \\
\rho(\epsilon, w) &= \begin{cases} +\infty & \text{if } |w| = 0 \\ -\infty & \text{otherwise} \end{cases} \\
\rho(x \geqslant c, w) &= \begin{cases} \inf\limits_{t \in [0, |w|)} w_x[t] - c & \text{if } |w| > 0 \\ -\infty & \text{otherwise} \end{cases} \\
\rho(x \leqslant c, w) &= \begin{cases} \inf\limits_{t \in [0, |w|)} c - w_x[t] & \text{if } |w| > 0 \\ -\infty, & \text{otherwise} \end{cases} \\
\rho(\varphi_1 \vee \varphi_2, w) &= \max\{\rho(\varphi_1, w), \rho(\varphi_2, w)\} \\
\rho(\varphi_1 \wedge \varphi_2, w) &= \min\{\rho(\varphi_1, w), \rho(\varphi_2, w)\} \\
\rho(\varphi_1 \cdot \varphi_2, w) &= \sup_{uv = w} \min\{\rho(\varphi_1, w), \rho(\varphi_2, w)\} \\
\rho(\varphi^*, w) &= \sup_{k \geqslant 0} \rho(\varphi^k, w) \\
\rho(\langle\varphi\rangle_I) &= \begin{cases} \rho(\varphi, w) & \text{if } |w| \in I \\ -\infty & \text{otherwise} \end{cases}
\end{aligned}
$$

Here, $\sup \emptyset = \inf \mathbb{R} = -\infty$ *and* $\inf \emptyset = \sup \mathbb{R} = +\infty$.

Similar to STL we state two important properties of robustness as theorems. Recall the Definition 2.4.1 of distance d between signals.

Theorem 2.8.7 (Soundness). *Let φ be an SRE and w is a signal. If $\rho(\varphi, w) > 0$ then $w \in L(\varphi)$. Similarly if $\rho(\varphi, w) < 0$ then $w \notin L(\varphi)$.*

Theorem 2.8.8 (Correctness). *Let φ be an SRE, and u,v two signals. If $v \in L(\varphi)$ and $d(u, v) < \rho(\varphi, v)$ then $u \in L(\varphi)$. Similarly if $v \notin L(\varphi)$ and $d(u, v) < -\rho(\varphi, v)$ then $u \notin L(\varphi)$.*

2.8.3 *Timed Regular Expressions (Event-Based)*

Timed Regular Expressions (Event-Based) deal with timed words rather than signals. We define the syntax and two kinds of semantics of TRE (event-based).

Definition 2.8.9 (Syntax). *The syntax of TRE (event-based) over an event alphabet Σ is given by the following grammar:*

$$\underline{a} \mid \epsilon \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi^* \mid \langle \varphi \rangle_I$$

Here, $a \in \Sigma$ and I is an interval inside $\mathbb{R}_{\geqslant 0}$.

We first define the time-event semantics and then the matching semantics. The exponent notation has the usual meaning over concatenation.

Definition 2.8.10 (Time-Event Semantics). *For TRE (event-based) the semantics based on time-event sequences, $[\![\ ]\!]$, is defined as follows:*

$$\begin{aligned}
[\![\underline{a}]\!] &= \{r \cdot a : r \in \mathbb{R}_{\geqslant 0}\} \\
[\![\epsilon]\!] &= \{\epsilon\} \\
[\![\varphi_1 \cdot \varphi_2]\!] &= [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \\
[\![\varphi_1 \vee \varphi_2]\!] &= [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
[\![\varphi_1 \wedge \varphi_2]\!] &= [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\
[\![\varphi^*]\!] &= \bigcup_{i=0}^{\infty} ([\![\varphi^i]\!]) \\
[\![\langle \varphi \rangle_I]\!] &= [\![\varphi]\!] \cap \{u : \lambda(u) \in I\}
\end{aligned}$$

Definition 2.8.11 (Matching Semantics). *The matching relation $\models$ of a TRE (event-based) φ for a timed word $w = t_1 a_1 \ldots t_n a_n$, relative to start time t and end time t' is defined inductively as follows ($t_0 = 0$ by default):*

$$\begin{aligned}
(w, t, t') \models \underline{a} &\leftrightarrow \exists i \in [1 \ldots n] \cdot a = a_i \wedge t = t_{i-1} \wedge t' = t_i \\
(w, t, t') \models \epsilon &\leftrightarrow \exists i \in [0 \ldots n] \cdot t = t_i \wedge t' = t_i \\
(w, t, t') \models \varphi_1 \cdot \varphi_2 &\leftrightarrow \exists t''.(w, t, t'') \models \varphi_1 \text{ and } (w, t'', t') \models \varphi_2 \\
(w, t, t') \models \varphi_1 \vee \varphi_2 &\leftrightarrow (w, t, t') \models \varphi_1 \text{ or } (w, t, t') \models \varphi_2 \\
(w, t, t') \models \varphi_1 \wedge \varphi_2 &\leftrightarrow (w, t, t') \models \varphi_1 \text{ and } (w, t, t') \models \varphi_2 \\
(w, t, t') \models \varphi^* &\leftrightarrow \exists k \geqslant 0.(w, t, t') \models \varphi^k \\
(w, t, t') \models \langle \varphi \rangle_I &\leftrightarrow t' - t \in I \text{ and } (w, t, t') \models \varphi
\end{aligned}$$

The Match-set for TRE (event-based) is defined in exactly the same way as TRE (state-based).

Definition 2.8.12 (Match-Set). *For any timed word ω and expression φ, we let*

$$\mathcal{M}(\varphi, \omega) := \{(t, t') \in T \times T : (\omega, t, t') \models \varphi\}$$

2.8.4 *Conditional TRE and Introduction of Event-Bounded TRE*

Here, we introduce an extension to TRE called conditional TRE. Then we impose a restriction to obtain what is called E-TRE. The conditional TRE is the same as TRE (state-based) except for the addition of precondition operator (denoted as ?) and postcondition operator (denoted as !). Consider two TRE φ_1 and φ_2, the matching semantics of the two aforementioned operators are given as follows:

$$\begin{aligned}(w, t, t') \models \varphi_1?\varphi_2 \quad &\leftrightarrow \quad (w, t, t') \models \varphi_2 \text{ and } \exists t'' \leqslant t, (w, t'', t) \models \varphi_1 \\ (w, t, t') \models \varphi_1!\varphi_2 \quad &\leftrightarrow \quad (w, t, t') \models \varphi_1 \text{ and } \exists t'' \geqslant t', (w, t', t'') \models \varphi_2\end{aligned}$$

Given a propositional term p, using the precondition and postcondition operators, one can define the syntactic sugar of rising edge $\uparrow p := \neg p?\epsilon!p$ and falling edge $\downarrow p :=\uparrow \neg p$. Using the rising edge and falling edge one can define E-TRE which are of form $\uparrow p$, $\psi_1 \cdot \varphi \cdot \psi_2$, $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \varphi$ with p a proposition, φ a TRE, and ψ_1,ψ_2 which are E-TRE. The match-set of E-TRE has the property of being finite.

2.9 APPROXIMATING MONOTONIC PARTITIONS USING QUERIES

In this section we explain an algorithm (Algorithm 1) from [55] which approximately computes an upward-closed set $\overline{X}$ and its complement $\underline{X}$ which would naturally be downward-closed. We assume the existence of the set $X = \overline{X} \cup \underline{X}$ over which a partially ordered relation $\geqslant$ is defined. In the terminology of multi-criteria optimization the boundary between $\overline{X}$ and $\underline{X}$ is called a Pareto front. Thus we call Algorithm 1 as Pareto front algorithm. For simplicity, we also assume that $X = [0, 1]^n$ where n is the number of dimensions over which the sets are defined. An upward closed set is characterized by the following property it satisfies defined in terms of X and the partially ordered relation $\geqslant$.

$$\forall x, x' \in X \; (x \in \overline{X} \wedge x' \geqslant x) \implies x' \in \overline{X}$$

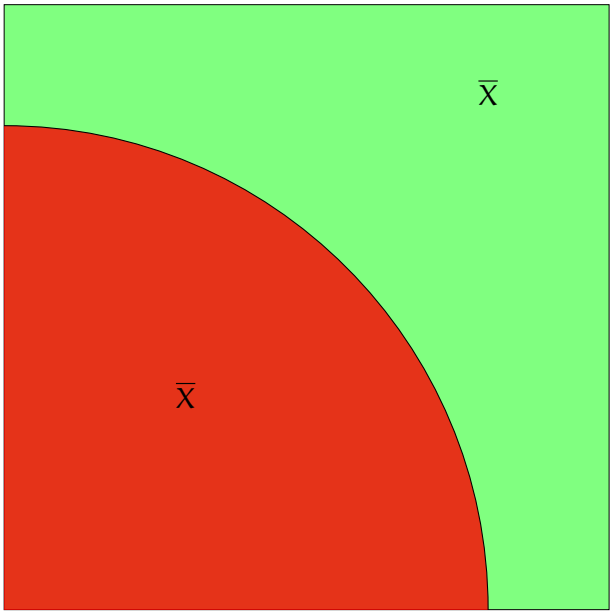
We consider the situation where $\overline{X}$ is not known but we have a Boolean-valued query function $\rho(x)$ which returns for a given $x \in X$ whether $x \in \overline{X}$. The function ρ is monotonically increasing over X because $\overline{X}$ is upward-closed and for all x and x' in X if $x' \geqslant x$ then $\rho(x')$ is greater than or equal to $\rho(x)$. We call $M = (\underline{X}, \overline{X})$ a monotone

bi-partition of X because ρ is monotonic. Algorithm 1 is approximate and takes as inputs the function ρ, two parameters δ and ϵ and the unit box X. Though approximate, it becomes more and more accurate as we decrease the values of ϵ and δ. The output is an approximation $M' = (\underline{Y}, \overline{Y})$ of M and the set L which is the undecided region with volume bounded by δ. An illustration of a two-dimensional example of the partition M and its approximation M' using boxes is shown in Figure 4. We now explain the details of the algorithm.

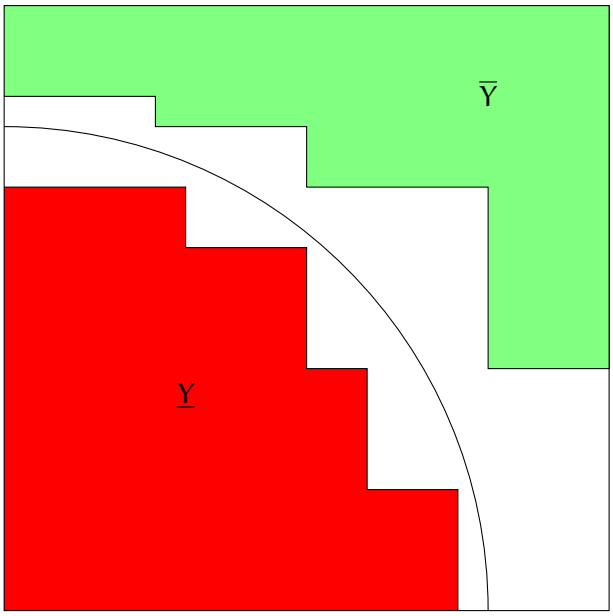
Algorithm 1 Pareto front algorithm

Input: A box $X = \lfloor \mathbf{0}, \mathbf{1} \rceil$; ρ which is a monotonically increasing Boolean-valued functions; error bounds δ and ε.
Output: An approximation $M' = (\underline{Y}, \overline{Y})$ of the partition. A set L representing the undecided region such that $|L| \leqslant \delta$. All sets are represented by unions of boxes.
$L = \{X\}$; $(\underline{Y}, \overline{Y}) = (\emptyset, \emptyset)$ ▷ initialization
repeat
  pop first $\lfloor \underline{x}, \overline{x} \rceil \in L$ ▷ take the largest box
  $\langle \underline{y}, \overline{y} \rangle = \texttt{boundary}(\rho, \langle \underline{x}, \overline{x} \rangle, \epsilon)$
  $\underline{Y} = \underline{Y} \cup \lfloor \underline{x}, \underline{y} \rceil$
  $\overline{Y} = \overline{Y} \cup \lfloor \overline{y}, \overline{x} \rceil$
  $L = L \cup I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$
until $\texttt{Vol}(L) \leqslant \delta$

Algorithm 1 represents all sets using a union of boxes. The two main ideas behind the algorithm are binary search and exploiting the idea of monotonicity to split a box and use recursion. The procedure boundary (Algorithm 2) is mainly a binary search idea applied to finding where (shown as z in Figure 5) on a given line $\langle \underline{x}, \overline{x} \rangle$ the query function ρ changes from 0 to 1 (or from false to true). We start with the diagonal $\langle \underline{x}, \overline{x} \rangle$ of a box $\lfloor \underline{x}, \overline{x} \rceil$. Initially, we do not know where the boundary is on the line and our best estimate of the line segment containing the boundary is $\langle \overline{y}, \underline{y} \rangle = \langle \underline{x}, \overline{x} \rangle$ i.e. the full diagonal. We find the mid-point of the uncertain segment and query the value of ρ there. Depending on the result of the query we halve the uncertain segment $\langle \overline{y}, \underline{y} \rangle$ and repeat this process. We terminate when the length of this uncertain segment goes below the given parameter ϵ. This idea has been illustrated in Figure 5. The next step is to split the box $\lfloor \underline{x}, \overline{x} \rceil$ into three regions using the result of the search. We illustrate this in Figure 6. From the monotonicity of ρ, we know that the green box $G = \lfloor \overline{y}, \overline{x} \rceil$ will be contained in the upward-closed set $\overline{X}$. Similarly, the red box $R = \lfloor \underline{x}, \underline{y} \rceil$ will be contained in the downward-closed set $\underline{X}$. We are left with the incomparable boxes represented using $I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$. In two-dimensional case as shown in Figure 6, this set consists of two boxes (B_1 and B_2). A formal definition and details on how to represent $I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ using overlapping sub-boxes is given



(a) Exact Pareto front accessible through the function $\rho = (x \in \overline{X})$



(b) Example output of Pareto front approximation algorithm on ρ

Figure 4: Illustration of input and output of Pareto front algorithm (from [55])

later in Sub-section 4.2.1 of Chapter 4. We add these boxes into the list L of incomparable boxes which represents the undecided region. Algorithm 1 contains repeated application of these search and split operations. It maintains a list of incomparable boxes L which initially contains just the one box X. It repeatedly selects the largest box from L, performs search and split, adds the produced incomparable boxes to L and terminates when the volume of L goes below the parameter δ. It outputs the approximation $M' = (\underline{Y}, \overline{Y})$ of the actual partition $M = (\underline{X}, \overline{X})$ which becomes more and more accurate as we decrease the value of the parameter δ.

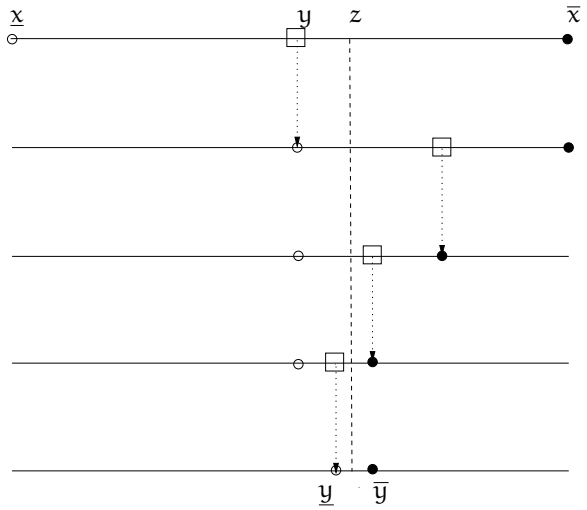


Figure 5: Binary search and the successive reduction of the uncertainty interval (from [55]).

Algorithm 2 One dimensional binary search: $\mathtt{boundary}(\langle \underline{x}, \overline{x} \rangle, \rho, \varepsilon)$

Input: A line segment $\ell = \langle \underline{x}, \overline{x} \rangle$, ρ a monotonically increasing Boolean-valued function and an error bound $\varepsilon \geqslant 0$.
Output: A line segment $\langle \overline{y}, \underline{y} \rangle$ containing the Pareto boundary of ρ such that $\overline{y} - \underline{y} \leqslant \varepsilon$.
$\langle \underline{y}, \overline{y} \rangle = \langle \underline{x}, \overline{x} \rangle$
while $\overline{y} - \underline{y} \geqslant \varepsilon$ **do**
    $y = (\underline{y} + \overline{y})/2$
    if $\mathtt{member}(y)$ **then**
        $\langle \underline{y}, \overline{y} \rangle = \langle \underline{y}, y \rangle$ ▷ left sub-interval
    else
        $\langle \underline{y}, \overline{y} \rangle = \langle y, \overline{y} \rangle$ ▷ right sub-interval
return $\langle \underline{y}, \overline{y} \rangle$

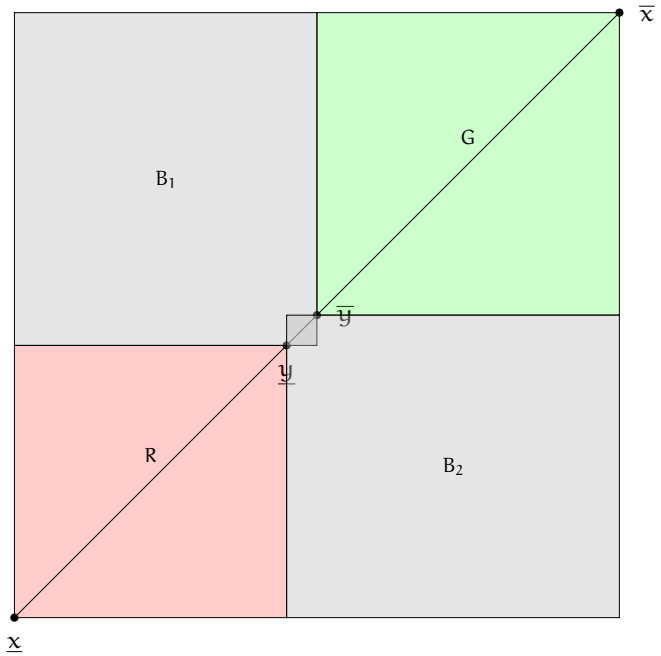


Figure 6: Recursion step of Pareto front algorithm (from [55])

EXPRESSIONS RÉGULIÈRES TEMPORISÉES PARAMÉTRIQUES

Parmi les formalismes temporels, l'identification paramétrique de la logique temporelle paramétrique du signal (PSTL) a déjà été bien étudiée. Il s'agit de calculer le domaine de validité qui est l'ensemble de toutes les valeurs de paramètres pour lesquelles la formule de logique temporelle du signal (STL) obtenue est satisfaite sur un signal observé. Il a été démontré que la représentation précise des domaines de validité nécessite l'utilisation d'ensembles semi-linéaires, qui s'écrivent sous la forme d'une combinaison booléenne d'inégalités linéaires. Il a été démontré que le calcul et la représentation exacts des domaines de validité pour la PSTL sont coûteux sur le plan informatique et parfois irréalisables sur le plan pratique. Cela nous a amenés à examiner si les versions paramétriques des expressions régulières temporisées (TRE) peuvent être analysées dans des délais raisonnables. Le problème de la correspondance paramétrique temporisée (PTPM) consiste à trouver l'ensemble des paramètres et des intervalles de temps où l'on trouve un motif temporel. Cet ensemble est appelé "ensemble de correspondance paramétrique". Cette méthode s'inspire fortement du problème de la recherche de chaînes de caractères correspondantes dans des documents textuels. Ce problème a été bien exploré pour les motifs exprimés à l'aide d'automates temporisés paramétriques (Parametric Timed Automata, PTA) qui traitent des séquences d'événements temporels. On dit que ces PTA ont une sémantique basée sur les événements. Mais, au lieu de cela, nous voulions traiter des signaux booléens et à valeurs réelles. Les TRE qui traitent ce type de signaux sont appelées expressions régulières de signaux paramétriques (Parametric Signal Regular Expressions, PSRE) et sont dites à sémantique basée sur l'état. Bien que notre motivation soit de traiter la sémantique basée sur l'état, nous discutons également du PTPM pour les TRE paramétriques avec une sémantique basée sur l'événement, appelées expressions régulières temporelles paramétriques (basées sur l'événement). Les expressions basées sur les événements traitent des mots ou des chaînes de caractères dans lesquels chaque caractère a un horodatage correspondant. Ces deux types de sémantique peuvent en fait être combinés pour définir ce que nous appelons les expressions régulières paramétriques basées sur les événements (Parametric Event-bounded Timed Regular Expressions, PETRE). Dans ces expressions, l'occurrence des événements est déduite lorsque les signaux booléens changent de valeur. Nous décrivons comment effectuer la PTPM pour ces types d'expressions également. Pour tous ces différents types de TRE paramétriques, nous montrons

que le PTPM peut être réalisé par induction en utilisant des types spéciaux de polytopes. Le PTPM de PSRE consiste à effectuer des opérations sur des listes de zones paramétriques (type spécial de polytopes). De même, le PTPM pour PTRE (basé sur les événements) et PE-TRE implique d'effectuer des opérations sur ce que nous appelons des intervalles paramétriques. Il s'agit d'intervalles de temps combinés à des contraintes sur des paramètres exprimés sous forme de polytopes. Les opérations sur les listes de zones paramétriques peuvent être effectuées efficacement à l'aide de ce que l'on appelle des techniques de balayage de plan. L'idée sous-jacente est d'ordonner les zones en fonction de leurs dimensions temporelles et d'éviter les opérations redondantes. De même, les opérations sur les intervalles paramétriques peuvent également être effectuées efficacement en triant et en recherchant en fonction des intervalles de temps. Enfin, nous formulons et résolvons le problème de l'identification paramétrique pour PSRE. Pour une identification paramétrique efficace, nous définissons le concept de "région décisive" et montrons son application. En termes simples, une région décisive caractérise l'ensemble des intervalles de temps qui peuvent éventuellement conduire à une correspondance. À la fin, nous montrons comment le PTPM est exécuté sur des exemples synthétiques simples pour faciliter la compréhension du lecteur. Parallèlement, nous donnons des exemples traitant de l'électrocardiogramme (ECG) et des données relatives au trafic maritime dans le monde réel.

3

PARAMETRIC TIMED REGULAR EXPRESSIONS

Out of the timed formalisms, one of the first works that deal the parametric version of STL i.e. PSTL is [10]. In this work, they deal with the validity domain, which is the set of all parameter values for which the obtained STL formula is satisfied over an observed signal. Accurately representing validity domains has been shown to require the use of semilinear sets, which are written as a Boolean combination of linear inequalities. The authors show that exactly computing and representing validity domains for PSTL is computationally expensive and sometimes practically intractable. This lead us to explore whether parametric versions of TRE can be analyzed within reasonable time complexity. The problem of PTPM involves finding all the set of parameters and time intervals where we find a pattern. This takes strong inspiration from the problem of finding matching strings inside text documents. This problem has been well explored for patterns expressed using Parametric Timed Automata (PTA) that deal with time-event sequences [84]. These PTA are said to be of event-based semantics. But, instead of that we wanted to deal with Boolean and real-valued signals. The TRE that deal with these kinds of signals are said to be of state-based semantics. Let us illustrate these parametric expressions by a simple example which concerns two Boolean signals p and q depicted on Fig. 7. Consider the following expression:

$$\phi_1 := (\langle p \rangle_{\theta_1} \cdot \neg p) \wedge (\neg q \cdot \langle q \rangle_{\theta_2}) \wedge (\texttt{true} \cdot \langle p \wedge q \rangle_{\theta_3} \cdot \texttt{true}) \qquad (1)$$

It represents the intersection of three different expressions. The first expression represents a signal p of duration θ_1. The second one represents a signal q of duration θ_2. Finally, the third expression represents the conjunction of p and q with a duration of θ_3. The period from t to t' matches the parametric expressions. We are interested in finding the *match-set*, that is the set of all the tuples $(t, t', \theta_1, \theta_2, \theta_3)$ corresponding to such matches. We first explored PTPM for this kind of TRE with parameters. We show that this can be computed and represented using polytopes with simple relations over timed variables. We also discuss how temporal ordering of these polytopes can be exploited to get efficient algorithms. Next, we describe types of parametric TRE that deal with time-event sequences and with mixed type of semantics. Naturally, we solve the PTPM problem for these. Finally, we formulate and solve the problem of parametric identification for TRE with parameters and those which deal with signals.

In this chapter we introduce three types of parametric TRE. We first talk about Parametric Signal Regular Expressions (PSRE). Then, we

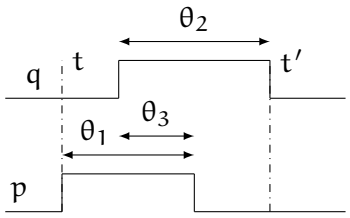


Figure 7: Matching a Parametric Timed Regular Expression ϕ_1

introduce PTRE (Event-Based). Later, we talk about Parametric Event-Bounded Timed Regular Expressions (PE-TRE). Finally, we formulate and solve the problem of parametric identification for PSRE. For efficient parametric identification we define the concept of "decisive region" and show its application. To explain in simple terms, a decisive region characterizes the set of time intervals that can possibly lead to a match. At the end, we show how PTPM is performed over simple synthetic examples for the reader's understanding. Along with this, we give examples dealing with real-world Electrocardiogram (ECG) and ship traffic data.

3.1 PARAMETRIC SIGNAL REGULAR EXPRESSIONS (PSRE)

We are given a signal $w : \mathbb{T} \to \mathbb{R}^n \times \mathbb{B}^m$ over a set of real variables $X = \{x_1, x_2, ..., x_n\}$ and a set of propositional variables $P = \{p_1, p_2, ..., p_m\}$. At each time point in $\mathbb{T}$ the signal assigns real values to variables $x \in X$ and Boolean values to variables $p \in P$. In PSRE, there are two types of parameters, *magnitude* parameters denoted by a vector $q = (q_1, ..., q_g)$ and *timing* parameters denoted by a vector $s = (s_1, ..., s_h)$. Their domains are polytopes $\mathcal{Q} \subseteq \mathbb{R}^g$ and $\mathcal{S} \subseteq \mathbb{R}^h$. A syntactic description of PSRE is as follows:

Definition 3.1.1 (Parametric Signal Regular Expressions)**.**

$$x \geqslant \lambda \mid x \leqslant \lambda \mid p \mid \langle \phi \rangle_I \mid \phi \cdot \psi \mid \phi^* \mid \phi \wedge \psi \mid \phi \vee \psi$$

where $x \in X$ *is a (signal) variable,* λ *is either a constant* c *or a magnitude parameter* q_i, $p \in P$ *a propositonal variable, and* I *stands for an interval* $[a, b]$ *where each of* a, b *is either a non-negative constant or a timing parameter* s_i.

Here x corresponds to a real-valued signal, and p a Boolean one. A parametric valuation $(u, v) \in \mathcal{Q} \times \mathcal{S}$ converts a PSRE formula φ into a SRE formula $\varphi_{u,v}$ obtained by substituting the values (u, v) in the parameters (q, s). We use the notations $\lambda_{u,v}$ and $I_{u,v}$ to denote the threshold and interval respectively obtained from such a substitution. The semantics of a PSRE φ with respect to a signal w is given in terms of a parametric match set.

Definition 3.1.2 (PSRE Semantics). *The satisfaction relation* $\models$ *of a* PSRE φ *by a signal* w *with respect to a start time* t*, a end time* t' *and a parametric valuation* (u, v) *is defined inductively as follows:*

$$(w, t, t', u, v) \models (x \leqslant \lambda) \leftrightarrow t < t' \wedge \forall t''.t < t'' < t' \rightarrow x[t''] \leqslant \lambda_{u,v}$$

$$(w, t, t', u, v) \models (x \geqslant \lambda) \leftrightarrow t < t' \wedge \forall t''.t < t'' < t' \rightarrow x[t''] \geqslant \lambda_{u,v}$$

$$(w, t, t', u, v) \models p \leftrightarrow t < t' \wedge \forall t''.t < t'' < t' \rightarrow p[t''] = 1$$

$$(w, t, t', u, v) \models \langle\varphi\rangle_I \leftrightarrow t' - t \in I_{u,v} \wedge (w, t, t', u, v) \models \varphi$$

$$(w, t, t', u, v) \models \varphi \cdot \psi \leftrightarrow \exists t''.(w, t, t'', u, v) \models \varphi$$

$$\wedge (w, t'', t', u, v) \models \psi$$

$$(w, t, t', u, v) \models \varphi \wedge \psi \leftrightarrow (w, t, t', u, v) \models \varphi \textit{ and } (w, t, t', u, v) \models \psi$$

$$(w, t, t', u, v) \models \varphi \vee \psi \leftrightarrow (w, t, t', u, v) \models \varphi \textit{ or } (w, t, t', u, v) \models \psi$$

$$(w, t, t', u, v) \models \varphi^* \leftrightarrow \exists k \geqslant 0.(w, t, t', u, v) \models \varphi^k$$

The set of parameter valuations and the segments of w that match a PSRE φ is expressed as follows.

Definition 3.1.3 (Parametric Match-Set). *For any signal* w *and* PSRE φ*, we let*

$$\mathcal{M}(\varphi, w) = \{(t, t', u, v) \in \mathbb{T} \times \mathbb{T} \times \mathcal{Q} \times \mathcal{S} : (w, t, t', u, v) \models \varphi\}$$

3.1.1 *Parametric Zones and Parametric Match-Sets*

Recall that we represent the vectors of magnitude and timing parameters with the symbols q and s respectively. A parametric zone is defined by six constraints of the following form: $\begin{pmatrix} t \prec c_1(q,s) \\ t' \prec c_2(q,s) \\ t' - t \prec c_3(q,s) \\ c_4(q,s) \prec t' - t \\ c_5(q,s) \prec t' \\ c_6(q,s) \prec t \end{pmatrix}$ along with a set of linear constraints $\mathbf{C_z}(q, s)$ over the parameter space, where $c_1, ..., c_6$ are piece-wise linear functions over (q, s). More precisely, c_1, c_2, c_3 can be expressed as minima of linear functions and c_4, c_5, c_6 as maxima of linear functions. The symbol $\prec$ is $\leqslant$ / $<$.

Theorem 3.1.4. *For signals that are Boolean or piece-wise linear, the parametric match-set of* PSRE *is a finite union of parametric zones.*

Proof. Let us consider the operators one by one and show that they preserve the form when applied over parametric zones.

It is proved in [72] that the parametric match set $\mathcal{M}(p, w)$ for Boolean signal p, is a set of triangular cylinders touching the diagonal $t \leqslant t'$.

The parametric match set for $(x \geqslant \lambda)$ is

$$\mathcal{M}(x \geqslant \lambda, w) = \{(t, t', \lambda) \mid \lambda \leqslant \inf_{t'' \in (t,t')} w_x[t'']\},$$

which is equivalent to the robustness support [17] for $(x \geqslant 0)$, that is

$$\mathcal{R}(x \geqslant 0, w) = \{(t, t', r) \mid r \leqslant \inf_{t'' \in (t,t')} w_x[t'']\}$$

Similarly $\mathcal{R}(x \leqslant 0, w)$ is equivalent to $\mathcal{M}(x \leqslant \lambda, w)$. In [17], it is shown that the robustness support of $x \leqslant 0$ and $x \geqslant 0$ for piece-wise linear and piece-wise constant signals can be represented as a finite union of polytopes (which are indeed parametric zones). Therefore, it follows that the parametric match-set for $x \geqslant \lambda$ and $x \leqslant \lambda$ is a finite union of parametric zones for these classes of signals.

Now, for the disjunction operator, we have

$$\mathcal{M}(\varphi \vee \psi, w) = \mathcal{M}(\varphi, w) \cup \mathcal{M}(\psi, w)$$

Note that a finite union of parametric zones is closed under the disjunction operator.

Next, the parametric match set for conjunction is

$$\mathcal{M}(\varphi \wedge \psi, w) = \mathcal{M}(\varphi, w) \cap \mathcal{M}(\psi, w)$$

Let us consider two parametric zones:

$$\mathbf{z}_1 = \begin{pmatrix} t \prec c_1 \\ t' \prec c_2 \\ t' - t \prec c_3 \\ c_4 \prec t' - t \\ c_5 \prec t' \\ c_6 \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_1}} \text{ and } \mathbf{z}_2 = \begin{pmatrix} t \prec c'_1 \\ t' \prec c'_2 \\ t' - t \prec c'_3 \\ c'_4 \prec t' - t \\ c'_5 \prec t' \\ c'_6 \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_2}}.$$

We have, $\mathbf{z}_1 \cap \mathbf{z}_2 = \begin{pmatrix} t \prec \min(c_1, c'_1) \\ t' \prec \min(c_2, c'_2) \\ t' - t \prec \min(c_3, c'_3) \\ \max(c_4, c'_4) \prec t' - t \\ \max(c_5, c'_5) \prec t' \\ \max(c_6, c'_6) \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_1}} \wedge \mathbf{C}_{\mathbf{z_2}}.$

Finite union of parametric zones is also closed under the conjunction operator.

For the concatenation operator, the parametric match set for concatenation is

$$\mathcal{M}(\varphi \cdot \psi, w) = \mathcal{M}(\varphi, w) \circ \mathcal{M}(\psi, w)$$

We have, $\mathbf{z}_1 \circ \mathbf{z}_2 = \begin{pmatrix} t \prec \min(c_1, c_1' - c_4, c_2 - c_4) \\ t' \prec \min(c_2', c_2 + c_3', c_1' + c_3') \\ t' - t \prec c_3 + c_3' \\ c_4 + c_4' \prec t' - t \\ \max(c_5', c_5 + c_4', c_6' + c_4') \prec t' \\ \max(c_6, c_6' - c_3, c_5 - c_3) \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_1}} \wedge \mathbf{C}_{\mathbf{z_2}} \wedge$
$(c_5 \prec c_1') \wedge (c_6' \prec c_2) \wedge (c_4 \prec c_3) \wedge (c_4' \prec c_3') \wedge (c_5 \prec c_2) \wedge (c_6' \prec c_1')$
The above equation for sequential composition is obtained by performing Fourier-Motzkin quantifier elimination. It follows that, a finite union of parametric zones is closed under concatenation operator.

For duration restriction, the parametric match set is
$\mathcal{M}(\langle\varphi\rangle_{[a,b]}, w) = \mathcal{M}(\varphi, w) \cap \{(t, t') : t' - t \in [a, b]\}$.

For a zone $\mathbf{z}_1 = \begin{pmatrix} t \prec c_1 \\ t' \prec c_2 \\ t' - t \prec c_3 \\ c_4 \prec t' - t \\ c_5 \prec t' \\ c_6 \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_1}}$,

We have:

$$\mathbf{z}_1 \wedge \{(t, t') : t' - t \in [a, b]\} = \begin{pmatrix} t \prec c_1 \\ t' \prec c_2 \\ t' - t \prec \min(c_3, b) \\ \max(c_4, a) \prec t' - t \\ c_5 \prec t' \\ c_6 \prec t \end{pmatrix} \wedge \mathbf{C}_{\mathbf{z_1}}$$

Again, we can see that a finite union of parametric zones is closed under duration restriction. □

3.1.2 *Kleene Star and Finite Number of Concatenations*

In this subsection we prove that the parametric match-set of Kleene star of PSRE can be computed by a finite number of concatenations. We start by recalling a couple of definitions followed by Lemma 3.1.5 from [72]. We define w_u as the set of Boolean signals obtained by introducing a magnitude parametric valuation u in a PSRE φ. An interval $[t, t']$ is said to be *unitary* with respect to a signal w if $t' - t < 1$ and w is constant throughout its interior (t, t'). Let $\sigma(w)$ be the least k such that w can be covered by k unitary intervals. A key property of $m = \sigma(w)$ is stated in the following lemma.

Lemma 3.1.5. *[72] For any* $n > 2m+1$ *if* $(w, t, t') \models \varphi^n$ *then* $(w, t, t') \models \varphi^{n-1}$.

In the general parametric case let $m' = 2 + 2 \max_{q \in Q} \sigma(w_q)$.

Lemma 3.1.6. *Kleene star can be bounded as follows:* $(w, t, t', u, v) \models \phi^*$ *if and only if* $(w, t, t', u, v) \models \phi^n$ *for some* $n \leqslant m'$.

Proof. It is easy to see that for piece-wise constant and piece-wise linear signals, m indeed has finite value. For the sake of contradiction, let us assume the following,

$$\exists (n > m'), t, t', u, v \cdot (w, t, t', u, v) \models \varphi^n \wedge (w, t, t', u, v) \not\models \varphi^{(n-1)}.$$

Simplifying this we get, $(w_{u_0}, t_0, t_0') \models \varphi^{n_0}_{u_0,v_0} \wedge (w_{u_0}, t_0, t_0') \not\models \varphi^{n_0-1}_{u_0,v_0}$ where n_0, t_0, t_0', u_0 and v_0 are constants with $n_0 > m'$. From the definition of m' we can deduce that $n_0 > 2\sigma(w_{u_0}) + 2$. This contradicts Lemma 3.1.5 for the case of signal w_{u_0} and expression φ_{u_0,v_0}. □

PSRE Example Semantics: Let us consider Expression (2) which can approximately match an electrocardiogram signal.

$$\phi_2 := \langle -0.55 \leqslant x \leqslant 0.29 \rangle_{\theta_1} \cdot \langle 0.29 \leqslant x \leqslant 2.0 \rangle_{\theta_2} \cdot \langle -0.6 \leqslant x \leqslant 0.29 \rangle_{\theta_3} \tag{2}$$

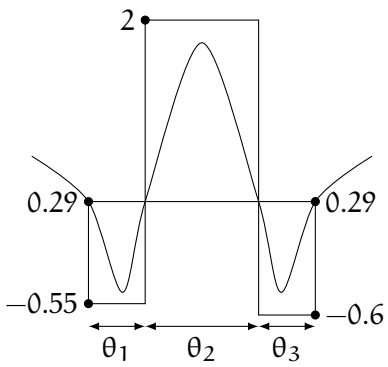


Figure 8: ECG matching expression ϕ_2

In Expression (2), the signal x stays in the interval $[-0.55, 0.29]$ for a duration of θ_1 time units. Then, its value increases and stays within $[0.29, 2.0]$ for θ_2 time units. Finally, it decreases and stays within $[-0.6, 0.29]$ for θ_3 time units. Figure 8 illustrates a matching signal.

3.2 PARAMETRIC TIMED REGULAR EXPRESSIONS WITH EVENT-BASED SEMANTICS

In this sub-section, we consider Parametric Timed Regular Expressions (PTRE) with event-based semantics that apply to timed words rather than to signals. The vector of timing parameters $(s_1, ..., s_h)$ is again denoted by s. Note that this version of PTRE does not have magnitude parameters.

Definition 3.2.1 (Parametric Timed Regular Expressions)**.**

$$\underline{a} \mid \epsilon \mid \langle \varphi \rangle_I \mid \varphi_1 \cdot \varphi_2 \mid \varphi_1 \cup \varphi_2 \mid \varphi_1 \cap \varphi_2 \mid \varphi^*$$

where $I = [\alpha, \beta]$ *and each of* α, β *is either a non-negative constant or a timing parameter* s_i*. For all* $a \in \Sigma$ *we define* $\underline{a}$ *which represents an arbitrary passage of time followed by event* a *where* Σ *is the event alphabet over which the PTRE is defined.*

The domain of parameters $\mathcal{S} \subseteq \mathbb{R}^h$ is expressed using a polytope. A parametric valuation $\nu \in \mathcal{S}$ converts a PTRE formula φ into a TRE formula φ_ν obtained by substituting the values ν in parameters s. We use I_ν to denote the interval obtained from such a substitution.

We first define the time-event semantics and then the matching semantics. The exponent notation has the usual meaning over concatenation.

Definition 3.2.2 (PTRE Time-Event Semantics)**.** *For PTRE (event-based) the semantics based on time-event sequences,* $[\![\ ]\!]$*, is defined as follows:*

$$\begin{aligned}
[\![\underline{a}]\!] &= \{r \cdot a : r \in \mathbb{R}_{\geqslant 0}\} \\
[\![\epsilon]\!] &= \{\epsilon\} \\
[\![\varphi_1 \cdot \varphi_2]\!] &= [\![\varphi_1]\!] \cdot [\![\varphi_2]\!] \\
[\![\varphi_1 \vee \varphi_2]\!] &= [\![\varphi_1]\!] \cup [\![\varphi_2]\!] \\
[\![\varphi_1 \wedge \varphi_2]\!] &= [\![\varphi_1]\!] \cap [\![\varphi_2]\!] \\
[\![\varphi^*]\!] &= \bigcup_{i=0}^{\infty} ([\![\varphi^i]\!]) \\
[\![\langle \varphi \rangle_I]\!] &= [\![\varphi]\!] \cap \{u : \lambda(u) \in I\}
\end{aligned}$$

Please note that the above PTRE time-event semantics is the same as the Definition 2.8.10 for the non-parametric case, except that in the parametric case the interval I is defined using parameters rather than constants.

Definition 3.2.3 (PTRE Event-Based Match-Set Semantics)**.** *The parametric match-set of a PTRE expression* φ *(with event-based semantics) for a timed word* $\omega = t_1 a_1 ... t_n a_n$ *is defined inductively as follows* ($t_0 = 0$ *by default):*

$$\mathcal{M}(\underline{a}, \omega) := \{(t, t', \nu) : \exists i \in [1...n] \cdot a = a_i \wedge t = t_{i-1} \wedge t' = t_i\}$$

$$\mathcal{M}(\epsilon, \omega) := \{(t, t, \nu) : \exists i \in [0...n] \cdot t = t_i\}$$

$$\mathcal{M}(\langle \varphi \rangle_I, \omega) := \{(t, t', \nu) : t' - t \in I_\nu \wedge (t, t', \nu) \in \mathcal{M}(\varphi, \omega)\}$$

$$\mathcal{M}(\varphi \cdot \psi, \omega) := \{(t, t', \nu) : \exists t''.(t, t'', \nu) \in \mathcal{M}(\varphi, \omega) \wedge (t'', t', \nu) \in \mathcal{M}(\psi, \omega)\}$$

$$\mathcal{M}(\varphi \wedge \psi, \omega) :=$$

$$\{(t, t', \nu) : (t, t', \nu) \in \mathcal{M}(\varphi, \omega) \wedge (t, t', \nu) \in \mathcal{M}(\psi, \omega)\}$$

$$\mathcal{M}(\varphi \vee \psi, \omega) :=$$

$$\{(t, t', \nu) : (t, t', \nu) \in \mathcal{M}(\varphi, \omega) \vee (t, t', \nu) \in \mathcal{M}(\psi, \omega)\}$$

$$\mathcal{M}(\varphi^*, \omega) := \{\exists k \geqslant 0.(t, t', \nu) \in \mathcal{M}(\varphi^k, \omega)\}$$

3.2.1 *Parametric Intervals and Parametric Match-Sets*

A parametric interval is defined by constraints of form $t = c_1$, $t' = c_2$ and $\mathbf{C_y}(r)$ which is a set of linear constraints over the parameter set r. Given constants c_1 and c_2, the constraints $t = c_1$ and $t' = c_2$ represent the beginning and ending of a time interval. A parametric interval is also a parametric zone but it is simpler. Theorem 3.2.4 describes the relation between parametric intervals and parametric match-sets of PTRE.

Theorem 3.2.4. *For a timed word the parametric match-set of a PTRE is a finite union of parametric intervals.*

Proof. Let $\mathbf{y}_1 := (t = d_1 \wedge t' = d_2 \wedge \mathbf{C_{y_1}})$ and $\mathbf{y}_2 := (t = e_1 \wedge t' = e_2 \wedge \mathbf{C_{y_2}})$ be two parametric intervals where d_1, d_2, e_1, e_2 are constants. We prove the theorem by considering the various cases in the definition of PTRE.

For $\underline{a}$ and ϵ, one can see that there exist a finite number of intervals in the match set.

If we apply duration restriction operation $\langle\varphi\rangle_{[\alpha,\beta]}$ on $\mathbf{y}_1$ the resulting parametric interval $\mathbf{y}$ can be written as follows,

$\mathbf{y} := (t = d_1 \wedge t' = d_2 \wedge \alpha \leqslant t' - t \leqslant \beta \wedge \mathbf{C_{y_1}})$

It can be further simplified as,

$\mathbf{y} := (t = d_1 \wedge t' = d_2 \wedge (\alpha \leqslant d_2 - d_1 \leqslant \beta \wedge \mathbf{C_{y_1}}))$

Finite unions of parametric intervals are closed under Boolean and concatenation operations. For concatenation operation $\varphi \cdot \psi$, let us consider $\mathbf{y} := \mathbf{y}_1 \circ \mathbf{y}_2$,

$\mathbf{y} := \exists t'' \cdot (t = d_1 \wedge t'' = d_2 \wedge \mathbf{C_{y_1}}) \wedge (t'' = e_1 \wedge t' = e_2 \wedge \mathbf{C_{y_2}})$

The resulting parametric interval $\mathbf{y}$ is non-empty only when $d_2 = e_1$. We can further simplify the equation for the resulting $\mathbf{y}$ as follows:

$\mathbf{y} := (t = d_1 \wedge t' = e_2 \wedge (\mathbf{C_{y_1}} \wedge \mathbf{C_{y_2}}))$

For intersection operation $\varphi \wedge \psi$, let us consider $\mathbf{y} := \mathbf{y}_1 \cap \mathbf{y}_2$, i.e.,

$\mathbf{y} := (t = d_1 \wedge t' = d_2 \wedge \mathbf{C_{y_1}}) \wedge (t = e_1 \wedge t' = e_2 \wedge \mathbf{C_{y_2}})$

The resulting parametric interval $\mathbf{y}$ is non-empty only when $d_1 = e_1$ and $d_2 = e_2$. For this case, we can further simplify the equation for $\mathbf{y}$ as follows: $\mathbf{y} := (t = d_1 \wedge t' = d_2 \wedge (\mathbf{C_{y_1}} \wedge \mathbf{C_{y_2}}))$

For union operation $\varphi \vee \psi$, we simply concatenate the two lists of parametric intervals that correspond to the expressions.

For the case of Kleene star the reasoning to show that the match set is finite union of parametric intervals is as follows. Let us consider a timed word $\omega = t_1 a_1 ... t_n a_n$. From the semantics in Definition 3.2.3 it follows that for any parametric interval $y_1 := (t = d_1 \wedge t' = d_2 \wedge$

$\mathbf{C_{y_1}}$) the start value $t = d_1$ and the end value $t = d_2$ both take values only from $\{t_1, ..., t_n\}$. They cannot take values in the dense space in between these discrete points. From this it follows that the number of possible values for (t, t') in the parametric match-set is at most quadratic in n. The exception is when one of the parametric intervals involved in sequential composition is of time length zero. Let us assume that in the sequential composition, the first parametric interval is $y_1 := (t = d_1 \wedge t' = d_2 \wedge \mathbf{C_{y_1}})$ and the second one $y_2 := (t = d_2 \wedge t' = d_2 \wedge \mathbf{C_{y_2}})$ is of time length zero. It follows that $\mathbf{y} := \mathbf{y}_1 \circ \mathbf{y}_2 := (t = d_1 \wedge t' = d_2 \wedge (\mathbf{C_{y_1}} \wedge \mathbf{C_{y_2}}))$. One can notice that $\mathbf{y} \subseteq \mathbf{y}_1$. So, $\mathbf{y}$ can be safely ignored when computing the transitive closure. Similar reasoning also applies when $\mathbf{y}_1$ is of time length zero. Therefore, the parametric match-set for Kleene star contains a finite number of parametric intervals. □

PTRE Example Time-Event Semantics: Let us consider the following PTRE which has both concatenation and intersection operators:

$$\phi_3 := (\langle \underline{a} \cdot \underline{b} \rangle_{\theta_1} \cdot \underline{c}) \wedge (\underline{a} \cdot \langle \underline{b} \cdot \underline{c} \rangle_{\theta_2}) \tag{3}$$

Recall that $\underline{a}$ represents an arbitrary passage of time followed by the event a. This can be represented as $r \cdot a$, where r represents passage of time and a is an event. Therefore we can write the semantics of $\underline{a}$ as, $[\![\underline{a}]\!] := \{r \cdot a : r \in \mathbb{R}_+\}$.

The time-event semantics of Expression (3) containing parameters θ_1 and θ_2 can be deduced as follows:

$$\{r_1 \cdot a \cdot r_2 \cdot b \cdot r_3 \cdot c : (r_1 + r_2 = \theta_1) \wedge (r_2 + r_3 = \theta_2)\}.$$

3.3 EVENT-BOUNDED TIMED REGULAR EXPRESSIONS WITH PARAMETERS

In Sub-section 2.8.4, we already presented an extension of TRE called E-TRE. Let us now examine the parametric version of E-TRE called Parametric Event-Bounded Timed Regular Expressions (PE-TRE). We are given a signal w defined exactly the same as in 3.1. The signal w assigns to each propositional variable $p \in P$ a Boolean value at each time point t in the time domain $\mathbb{T}$. So, for each $p \in P$ we have a corresponding Boolean signal. A PE-TRE is of form $\uparrow p$, $\psi_1 \cdot \varphi \cdot \psi_2$, $\psi_1 \vee \psi_2$, or $\psi_1 \wedge \varphi$ where $p \in P$, $\uparrow p$ stands for a rising edge of p, ψ_1 and ψ_2 are PE-TRE and φ stands for a PSRE. Note that falling edge can be defined as $\downarrow p :=\uparrow \neg p$.

Theorem 3.3.1. *For Boolean, piece-wise linear and piece-wise constant signals the parametric match-set of PE-TRE is always a finite union of parametric intervals.*

Proof. Let $\mathbf{y}_1 = (t = d_1 \wedge t' = d_2 \wedge \mathbf{C_{y_1}})$ and $\mathbf{y}_2 = (t = e_1 \wedge t' = e_2 \wedge \mathbf{C_{y_2}})$ be two parametric intervals where d_1, d_2, e_1, e_2 are constants.

Let $\mathbf{z} = \begin{pmatrix} t \prec c_1 \\ t' \prec c_2 \\ t' - t \prec c_3 \\ c_4 \prec t' - t \\ c_5 \prec t' \\ c_6 \prec t \end{pmatrix} \wedge \mathbf{C_z}$ be a parametric zone. We prove the theorem by considering the various cases in the definition of PE-TRE.

For $\uparrow p$, given a Boolean signal p, there exists a finite number of time points with a rising edge. One can see that these time points can be represented as parametric intervals.

For $\psi_1 \cdot \varphi \cdot \psi_2$, we iterate and perform sequential composition operation over $\mathbf{y}_1 \in \mathcal{M}(\psi_1, w)$, $\mathbf{z} \in \mathcal{M}(\varphi, w)$ and $\mathbf{y}_2 \in \mathcal{M}(\psi_2, w)$, where w is a signal, $\mathbf{y}_1$,$\mathbf{y}_2$ are parametric intervals and $\mathbf{z}$ is a parametric zone. The sequential composition $\mathbf{y}_1 \circ \mathbf{z} \circ \mathbf{y}_2$ can be written as:

$$\exists s, s' \cdot t = d_1 \wedge s = d_2 \wedge \mathbf{C_{y_1}} \wedge \begin{pmatrix} s \prec c_1 \\ s' \prec c_2 \\ s' - s \prec c_3 \\ c_4 \prec s' - s \\ c_5 \prec s' \\ c_6 \prec s \end{pmatrix} \wedge \mathbf{C_z} \wedge s' = e_1 \wedge t' = e_2 \wedge \mathbf{C_{y_2}}$$

One can see that, after performing quantifier elimination over s and s' the result will be a parametric interval.

For $\psi_1 \vee \psi_2$, we simply concatenate the two lists of parametric intervals that correspond to ψ_1 and ψ_2. And for $\psi_1 \wedge \varphi$, we need to perform intersection of a parametric interval with a parametric zone. One can easily see that the result of this intersection is also a parametric interval. □

PE-TRE Example Semantics: Expression (4) denotes the pattern of a brake control signal b for a vehicle under heavy braking situation. It is a parameterized version of the brake control signal pattern of the anti-lock brake system example given in [37]. It starts with a rise edge of b followed by a braking period of duration less than θ_1. It continues with one or more pulses with duration less than θ_2. It ends with a falling edge of b. In Figure 9, we can see an illustration of the braking pattern b.

$$\phi_4 := \uparrow b \cdot \langle b \rangle_{[0,\theta_1]} \cdot \langle \neg b \cdot b \rangle^{+}_{[0,\theta_2]} \cdot \downarrow b \tag{4}$$

3.4 PARAMETRIC MATCH-SET COMPUTATION

In this section, we discuss how to represent and compute parametric match-sets. First in Subsections 3.4.1 and 3.4.2, we describe how to

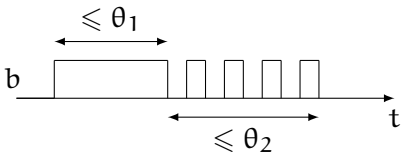


Figure 9: Braking Pattern

perform the *intersection* and *concatenation* operations operations. In Subsection 3.4.3, we discuss the transitive closure operation. Once we know how to perform intersection, concatenation and transitive closure operations, we can inductively compute the parametric match set of any expression.

3.4.1 *Binary Operations of PSRE and PE-TRE*

We remark that some sub-expressions of PE-TRE and all PSRE expressions have parametric match-sets that are unions of parametric zones. Therefore, for both PE-TRE and PSRE the parametric match-sets for sub-expressions are represented as unions of polytopes. We inductively perform binary zone operations (*intersection* and *concatenation*) over unions of polytopes. The resulting final match-set for the whole expression will also be a union of polytopes. A naive implementation of intersection and concatenation operations would involve $\mathcal{O}(n^2)$ of polytope intersections and sequential compositions respectively. However, if we exploit the inherent temporal ordering between parametric zones, this can be avoided. Indeed, using the plane-sweep idea, we need only to consider pairs of polytopes which potentially overlap. Note that this idea was also used in [72].

We first define the functions $\pi_1^+, \pi_1^-, \pi_2^+, \pi_2^-$ over parametric zones as below:

$$\pi_1^+(\mathbf{z}) = \{\max(t) : (t, t', u, v) \in \mathbf{z}\}$$
$$\pi_1^-(\mathbf{z}) = \{\min(t) : (t, t', u, v) \in \mathbf{z}\}$$
$$\pi_2^+(\mathbf{z}) = \{\max(t') : (t, t', u, v) \in \mathbf{z}\}$$
$$\pi_2^-(\mathbf{z}) = \{\min(t') : (t, t', u, v) \in \mathbf{z}\}$$

They can be computed using convex optimization over parametric zones (polytopes). With the values of these functions we can bound each zone inside a rectangular cylinder over the timed dimensions t, t'. For intersection, Algorithm 3, we sort the lists Z and Z' according to π_1^-. We keep two active lists Y and Y' that contain the candidates for intersection. We move elements one by one to the active lists and remove them when we deduce that they will not participate in further non-empty intersections. We can remove a $z \in Y$ if $\pi_1^+(z) < \pi_1^-(z')$ for every $z' \in Y'$ and vice versa. For concatenation, Algorithm 4, we sort Z by π_2^- and Z' by π_1^-. We then compute all pairs $z \circ z'$ such that

$[\pi_2^-(z), \pi_2^+(z)] \cap [\pi_1^-(z'), \pi_1^+(z')] \neq \emptyset$. This means that we discard the combinations where the end interval for $z \in Z$ does not intersect the begin interval for $z' \in Z'$. Note that first($\mathbf{Z}$) denotes the first element of the ordered list $\mathbf{Z}$.

Algorithm 3 INTERSECT(Z, Z')
assume Z, Z' *sorted by* π_1^-

1: **for each** $z \in Z$ **do**
2: Compute $\pi_1^+, \pi_1^-, \pi_2^+, \pi_2^-$
3: **for each** $z' \in Z'$ **do**
4: Compute $\pi_1^+, \pi_1^-, \pi_2^+, \pi_2^-$
5: $Y := Y' := Z'' := \emptyset$
6: **while** $Z \neq \emptyset \vee Z' \neq \emptyset$ **do**
7: $z := \text{first}(Z)$; $l := \pi_1^-(z)$
8: $z' := \text{first}(Z')$; $l' := \pi_1^-(z')$
9: **if** $l < l'$ **then**
10: Move z from Z to Y
11: $Y' = \{z' \in Y' : \pi_1^+(z') \geqslant l\}$
12: **for each** $z' \in Y'$ **do**
13: $z'' := z \cap z'$ ▷ Polyhedra intersection
14: $Z'' := Z'' \cup z''$ ▷ Check emptiness before adding
15: **else**
16: Move z' from Z' to Y'
17: $Y = \{z \in Y : \pi_1^+(z) \geqslant l'\}$
18: **for each** $z \in Y$ **do**
19: $z'' := z \cap z'$
20: $Z'' := Z'' \cup z''$ ▷ Check emptiness before adding
21: **return** Z''

3.4.2 *Binary Operations in PTRE (with Event-Based Semantics)*

All sub-expressions of PTRE (Event-Based) have parametric match sets that are union of parametric intervals. Therefore, we discuss intersection and concatenation operations over unions of parametric intervals. Sorting and binary search algorithms can be used to make these operations efficient. Given a parametric interval $\mathbf{y} := (t = d_1 \wedge t' = d_2 \wedge \mathbf{C_y})$ we define the functions π^- and π^+ as $\pi^-(\mathbf{y}) = d_1$ and $\pi^+(\mathbf{y}) = d_2$ respectively.

For intersection, in Algorithm 5, we sort the second list R' using lexicographical ordering over (π^-, π^+). Then, for each parametric interval r in R we perform a binary search on R' using $(\pi^-(r), \pi^+(r))$. After this search, using a simple iteration, we find the set of parametric intervals Y that might have a non-empty intersection with r. For concatenation, in Algorithm 6, we sort the second list R' using π^-.

Algorithm 4 CONCAT(Z, Z')
assume Z sorted by π_2^-, Z' sorted by π_1^-

1: **for each** $z \in Z$ **do**
2: Compute $\pi_1^+, \pi_1^-, \pi_2^+, \pi_2^-$
3: **for each** $z' \in Z'$ **do**
4: Compute $\pi_1^+, \pi_1^-, \pi_2^+, \pi_2^-$
5: $Y := Y' := Z'' := \emptyset$
6: **while** $Z \neq \emptyset \vee Z' \neq \emptyset$ **do**
7: $z := \text{first}(Z)$; $l := \pi_2^-(z)$
8: $z' := \text{first}(Z')$; $l' := \pi_1^-(z')$
9: **if** $l < l'$ **then**
10: Move z from Z to Y
11: $Y' = \{z' \in Y' : \pi_1^+(z') \geqslant l\}$
12: **for each** $z' \in Y'$ **do**
13: $z'' := z \circ z'$ ▷ Polyhedra renaming, intersection and quantifier elimination
14: $Z'' := Z'' \cup z''$ ▷ Check emptiness before adding
15: **else**
16: Move z' from Z' to Y'
17: $Y = \{z \in Y : \pi_2^+(z) \geqslant l'\}$
18: **for each** $z \in Y$ **do**
19: $z'' := z \circ z'$
20: $Z'' := Z'' \cup z''$ ▷ Check emptiness before adding
21: **return** Z''

Then, for each parametric interval r in R we perform binary search on R' using $\pi^+(r)$. Thus, we find the set Y' of parametric intervals that start where r ends. The use of sorting combined with binary search makes the Algorithms 5,6 efficient. Hence a lot of redundant polytopic operations can be avoided. It is also important to note that the number of variables involved in the polytopes is also reduced by two (beginning and ending of intervals can be stored as constants).

Algorithm 5 INTERVAL_INTERSECTION(R, R')
assume R' sorted by (π^-, π^+) lexicographically

```
1: R'' := ∅
2: for each r ∈ R do
3:     (l, l') := (π⁻(r), π⁺(r))
4:     Y = {z ∈ R' : (π⁻(z) = l) ∧ (π⁺(z) = l')} ▷ using binary search
5:     for each z ∈ Y do
6:         z'' := r ∩ z'
7:         R'' := R'' ∪ z''
8: return R''
```

Algorithm 6 INTERVAL_CONCAT(R, R')
assume R' sorted by π^-

```
1: R'' := ∅
2: for each r ∈ R do
3:     l' := π⁺(r)
4:     Y' = {z ∈ R' : π⁻(z) = l'}          ▷ using binary search
5:     for each z ∈ Y' do
6:         z'' := r ∘ z'
7:         R'' := R'' ∪ z''
8: return R''
```

3.4.3 *Transitive Closure for Kleene Star*

The transitive closure over both a union of polytopes and a union of parametric intervals can be computed in a straightforward manner. We can directly use the squaring based algorithm given in [72]. We just have to choose the right concatenation operation that applies from Subsections 3.4.1 and 3.4.2.

3.5 PARAMETRIC IDENTIFICATION FOR PSRE

In this section we deal with a PSRE parametric identification problem stated as follows: given labelled signals, find parameter values in a PSRE that produce the matches corresponding to the labels.

A computational problem arises when the expression contains atomic predicates of form $x \leqslant q$ or $x \geqslant q$ with the magnitude parameter q. For a given signal w, the parametric match-sets $\mathcal{M}(x \leqslant q, w)$ and $\mathcal{M}(x \geqslant q, w)$ contain a number of parametric zones approximately equal to the number of samples in w, which can be large in practice. Unlike STL that involves absolute time, TRE is more specific to relative time and we can thus exploit this specificity to decompose the signal horizon into non-overlapping time intervals and then perform the computation separately on each interval. To explain this idea, we first need the concept of *decisive region*.

3.5.1 *Decisive Regions*

Let us assume we are given a PSRE ψ over magnitude parameters q and timing parameters s. The expression ψ is defined over a signal w. We use vectors u and v to represent the parametric valuations for magnitude and timing parameters respectively. Given a time interval $I = [a, b]$, the decisive region $\mathcal{D}(I)$ corresponding to the interval I is defined as follows:

$$\mathcal{D}(I) := \{(t, t', u, v) : (a \leqslant t) \wedge (t' \leqslant b) \wedge (t \leqslant t')\}$$

The aim is to compute the intersection of the parametric match-set for ψ and w with the decisive region, that is $\mathcal{M}(\psi, w) \cap \mathcal{D}(I)$.

As an illustration we show in Figure 10 interval I_1 with the corresponding decisive region $\mathcal{D}(I_1)$ in green. Consider Interval I_{out} which is outside $\mathcal{D}(I_1)$. It goes to the left by left concatenation and upwards for right concatenation, to points I_{lc} and I_{rc} respectively. It becomes clear that the intervals outside the decisive region can never come inside it by application of concatenation operation. Therefore, we can safely ignore the points outside the decisive region.

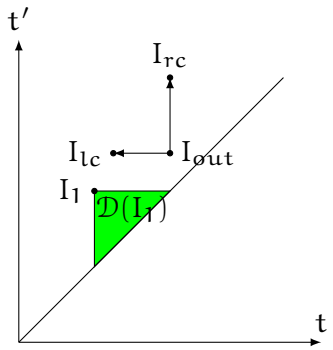


Figure 10: Points outside will be pushed further outside with concatenation

Now, we show some equivalences that will allow us to compute the above intersection efficiently in an inductive manner. Let us consider the PSRE φ_1,φ_2. It is easy to see that

$$\mathcal{M}(\varphi_1 \vee \varphi_2, w) \cap \mathcal{D}(I) = (\mathcal{M}(\varphi_1, w) \cap \mathcal{D}(I)) \cup (\mathcal{M}(\varphi_2, w) \cap \mathcal{D}(I))$$

$$\mathcal{M}(\varphi_1 \wedge \varphi_2, w) \cap \mathcal{D}(I) = (\mathcal{M}(\varphi_1, w) \cap \mathcal{D}(I)) \cap (\mathcal{M}(\varphi_2, w) \cap \mathcal{D}(I))$$

$$\mathcal{M}(\langle \varphi_1 \rangle_I, w) \cap \mathcal{D}(I) = (\mathcal{M}(\varphi_1, w) \wedge \mathcal{D}(I)) \cap \{(t, t', u, v) : t' - t \in I_{u,v}\}$$

Now for the final case of $\varphi_1 \cdot \varphi_2$, we need to prove the following equality:

$$\mathcal{M}(\varphi_1 \cdot \varphi_2, w) \cap \mathcal{D}(I) = (\mathcal{M}(\varphi_1, w) \cap \mathcal{D}(I)) \circ (\mathcal{M}(\varphi_2, w) \cap \mathcal{D}(I)) \cap \mathcal{D}(I)$$

The following inclusion is straightforward:

$$(\mathcal{M}(\varphi_1, w) \cap \mathcal{D}(I)) \circ (\mathcal{M}(\varphi_2, w) \cap \mathcal{D}(I)) \cap \mathcal{D}(I) \subseteq \mathcal{M}(\varphi_1 \cdot \varphi_2, w) \cap \mathcal{D}(I)$$

Let $(t_0, t_0', u, v) \in \mathcal{M}(\varphi_1 \cdot \varphi_2, w) \cap \mathcal{D}(I)$. This implies there exists t_0'' such that $(t_0, t_0'', u, v) \in \mathcal{M}(\varphi_1)$ and $(t_0'', t_0', u, v) \in \mathcal{M}(\varphi_2)$. Since $(t_0, t_0', u, v) \in \mathcal{D}(I)$ it implies that $(t_0, t_0'', u, v) \in \mathcal{D}(I)$ and $(t_0'', t_0', u, v) \in \mathcal{D}(I)$.

Therefore, the following inclusion is also true:

$$\mathcal{M}(\varphi_1 \cdot \varphi_2, w) \cap \mathcal{D}(I) \subseteq (\mathcal{M}(\varphi_1, w) \cap \mathcal{D}(I)) \circ (\mathcal{M}(\varphi_2, w) \cap \mathcal{D}(I)) \cap \mathcal{D}(I)$$

which establishes the equality we need to prove. □

Now, we consider how to efficiently compute $\mathcal{M}(\psi, w) \cap \mathcal{D}(I)$. Note that the intersection with the decisive region trickles right down to the leaves of the parse tree till it reaches the atomic predicates. The bottle neck is when ψ contains atomic predicates of form $x \leqslant p$ or $x \geqslant p$ with the magnitude parameter p. The parametric match set $\mathcal{M}(x \leqslant p, w)$ contains a number of parametric zones equal to the number of samples in w. Using the four aforementioned equivalences, to compute $\mathcal{M}(\psi, w) \cap \mathcal{D}(I)$ we only need to handle $\mathcal{M}(x \leqslant p, w) \cap \mathcal{D}(I)$. We can now see the interest of the concept of decisive region since the number of parametric zones to consider is reduced to the number of samples in the part of the signal between time units a and b i.e. in $w[a, b]$. This is particularly useful when the signal w is very large.

3.5.2 *Parametric Identification using Decisive Regions*

We consider the following scenario: signals with labelled intervals are provided to us by some expert, (e.g. temperature records and heat waves, road traffic records and dangerous overtakings, etc.). Our aim is to automatise the labelling of the signal (e.g. finding heat waves in unlabelled temperature curves) by pattern matching. To do so, we try to find (aka. identify) parameter valuations for a given parametric formula so that matches of the formula over the signals corresponds to the labelled intervals of the expert.

We are given a PSRE ψ, a signal w and a list $\mathcal{I}$ of n intervals $I_1 = [a_1, b_1], \ldots, I_n = [a_n, b_n]$. We would like to compute the set of param-

eters that produce a match at each of these n intervals. More formally, we want to compute the set $\mathcal{P}(\psi, w, \mathcal{I})$ which is defined as follows:

$$\{(u, v) : \bigwedge_{1 \leqslant k \leqslant n} (\exists t, t' \, (t, t', u, v) \in \mathcal{M}(\psi, w) \wedge t = a_k \wedge t' = b_k)\}$$

We call $\mathcal{P}(\psi, w, \mathcal{I})$ the solution set of ψ, w and $\mathcal{I}$. Note that $(t, t', u, v) \in \mathcal{M}(\psi, w) \wedge t = a_k \wedge t' = b_k$ is equivalent to $(t, t', u, v) \in (\mathcal{M}(\psi, w) \cap \mathcal{D}(I_k)) \wedge t = a_k \wedge t' = b_k$. Therefore, the previously described concept of decisive region can be applied for parametric identification.

3.5.3 *Combining Booleanization and Parametric Matching*

While the concept of decisive region can help reducing the number of parametric zones to be handled at a time, this number can still be large, often due to the magnitude parameters. In order to further reduce time complexity, it is of interest to separate magnitude parameters and perform matching expressions containing only timing parameters. This can be done by Booleanizing the signals where the Booleanization involves magnitude variables. Such Booleanization can be done using various operations over signals. For example, one can use *extended STL* [14] which is particularly appropriate for this purpose since this formalism allows expressing quantitatively shapes of signals. Indeed, combining with this specification language we can avoid the use of magnitude parameters in several cases. For instance to detect a peak with unknown height one could be tempted to use a magnitude parameter θ and write $x \geqslant \theta$. With extended STL it suffices to detect the maximum over a window as discussed earlier in Section 2.7. Another example is the stabilisation of a signal around an unknown value θ within a tolerance ϵ, that is, $\theta - \epsilon \leqslant x \leqslant \theta + \epsilon$. We can avoid the use of the magnitude parameter θ by saying that the difference between the maximum and minimum on a window I is within the tolerance ϵ, that is, $\max_I x - \min_I x \leqslant \epsilon$.

3.6 EXPERIMENTS

We implemented the algorithms discussed in Section 3.4 in C++, into a tool named `parameTRE`. We represent parametric zones as polytopes. Parametric intervals are represented as a specialized C++ class with the beginning and end of the interval stored as member variables. Constraints on parameters are stored as a member polytope. For PSRE and PE-TRE, we maintain a parametric match set as a union of polytopes. For event-based PTRE, we maintain it as a union of parametric intervals. To handle polytopes, the tool uses the Parma Polyhedra Library (PPL) [13]. We evaluate the performance of the tool using various kinds of expressions, signals and timed words. All

experiments have been performed on a laptop with a Core i7-8665U and 16GB RAM. We first show experimental results for synthetic examples to evaluate the correctness of the implementation and the performance of the proposed algorithms. Then, we show how we can match or detect interesting behaviours in real-life scenarios.

3.6.1 *Synthetic Examples*

We first describe the data (signals and timed words) and expressions we are dealing with and then give the experimental results.

3.6.1.1 *Signals and Timed Words*

We use four different types of signals and a single timed word. The signal $wsynth_1$ for ϕ_1 (Expr (1)) consists of signal p which is a square wave of period 14 and q which is p shifted by 3 time units. Both p and q are repeated 1000 times. The signal $wbrake$ for ϕ_4 (Expr (4)) consists of the signal b which represents the braking pattern. The signal $wsynth_2$ for ϕ_5 (Expr (5)) consists of signal p_0 which is a square wave of period 200 time units. We get p_1 and p_2 by shifting p_0 by 35 and 45 time units respectively. All of p_0, p_1 and p_2 are repeated 1000 times. The timed word $word_1$ for ϕ_6 (Expr (6)) consists of alternating occurrences of b and a with a total of around 1000 events.

3.6.1.2 *Expressions*

The expressions ϕ_1 and ϕ_4 have already been discussed in the introduction at the beginning of this chapter and in Section 3.3 respectively. The expressions ϕ_5 and ϕ_6 given below are simple examples of Kleene closure operation for state-based and event-based paradigms respectively.

$$\phi_5 := p_0 \cdot (\langle p_1 \rangle_{\theta_1} \cdot \langle p_2 \rangle_{\theta_2})^+ \tag{5}$$

$$\phi_6 := \langle \underline{a} \cdot \langle \underline{b} \cdot \underline{a} \rangle^*_{[\theta_2, \theta_3]} \cdot \underline{b} \rangle_{\theta_1} \tag{6}$$

The experimental results are summarized in Table 1. The column “Data" gives the type of the signal or timed word. The column “Size" expresses the number of samples for signals and number of events for timed words. The column “Matches" expresses the number of parametric zones in a match-set. The “Time" column gives the total time taken to find matches in the given data. In Tables 1 and 2, we mentioned the number of parametric zones/intervals in the column “Matches” since it can serve as an empirical measure of geometric complexity of match sets.

Expression	Data	Size	Matches	Time
ϕ_1 (Expr 1)	$wsynth_1$	4000	1000	90s
ϕ_4 (Expr 4)	$wbrake$	16	9	1.3s
ϕ_5 (Expr 5)	$wsynth_2$	6000	8000	87s
ϕ_6 (Expr 6)	$word_1$	1002	125251	77s

Table 1: Experiments With Synthetic Data

A parametric zone in the parametric match set for ϕ_1 and $wsynth_1$ is:

$$\begin{aligned}\mathbf{z}_I := (t' - \theta_2 = 3) \wedge (t + \theta_1 = 7) \wedge (\theta_3 \leqslant 4) \wedge (t' \leqslant 10) \wedge \\ (t \leqslant 3) \wedge (\theta_3 \geqslant 1) \wedge (t \geqslant 0) \wedge (t' \geqslant 7)\end{aligned}$$

We can project $\mathbf{z}_I$ on three dimensions out of which two are related to time and one is a parameter. For example, we denote $\mathbf{z}_I$ projected on dimensions (t, t', θ_1) as $\mathbf{z}_I(t, t', \theta_1)$. Consider the following projections:

$$\begin{aligned}\mathbf{z}_I(t, t', \theta_1) &:= (0 \leqslant t \leqslant 3) \wedge (7 \leqslant t' \leqslant 10) \wedge (t + \theta_1 = 7) \\ \mathbf{z}_I(t, t', \theta_2) &:= (0 \leqslant t \leqslant 3) \wedge (7 \leqslant t' \leqslant 10) \wedge (t' - \theta_2 = 3) \\ \mathbf{z}_I(t, t', \theta_3) &:= (0 \leqslant t \leqslant 3) \wedge (7 \leqslant t' \leqslant 10) \wedge (1 \leqslant \theta_3 \leqslant 4)\end{aligned}$$

The above projections are illustrated in Figure 11. Projecting $\mathbf{z}_I$ on the parameter space $(\theta_1, \theta_2, \theta_3)$ gives us the following and is illustrated in Figure 12.

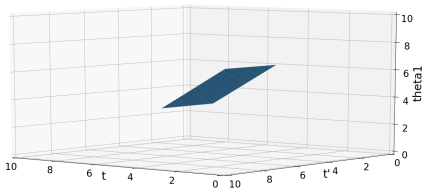
$$\mathbf{z}_I(\theta_1, \theta_2, \theta_3) := (4 \leqslant \theta_1 \leqslant 7) \wedge (4 \leqslant \theta_2 \leqslant 7) \wedge (1 \leqslant \theta_3 \leqslant 4)$$

Another parametric zone which comes from the parametric match set for ϕ_5 and $wsynth_2$:

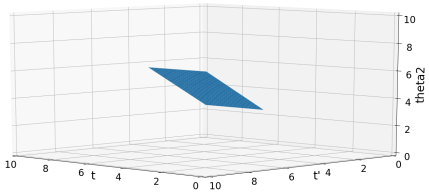
$$\begin{aligned}\mathbf{z}_K := (\theta_1 \geqslant 1) \wedge (\theta_2 \geqslant 1) \wedge (\theta_2 \leqslant 100) \wedge \\ (t \geqslant 20) \wedge (t' \leqslant 200) \wedge (t' - t - \theta_1 - \theta_2 \geqslant 0) \wedge (t' - \theta_2 \leqslant 135) \wedge \\ (t' - \theta_1 - \theta_2 \leqslant 120) \wedge (t' - \theta_2 \geqslant 45) \wedge (t' - \theta_1 - \theta_2 \geqslant 35)\end{aligned}$$

Similar to $\mathbf{z}_I$, we can project $\mathbf{z}_K$ on three dimensions (two dimensions in time with one parameter) and get $\mathbf{z}_K(t, t', \theta_1)$ and $\mathbf{z}_K(t, t', \theta_2)$. They are illustrated in Figure 13 and are as follows:

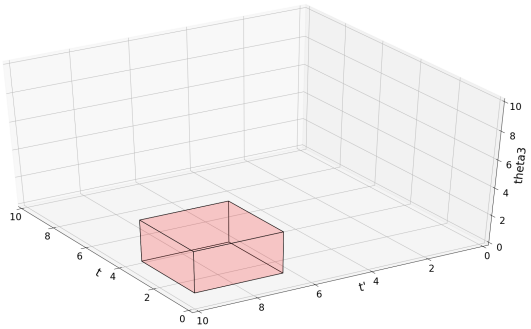
$$\begin{aligned}\mathbf{z}_K(t, t', \theta_1) := (20 \leqslant t \leqslant 120) \wedge (46 \leqslant t' \leqslant 200) \wedge \\ (1 \leqslant \theta_1 \leqslant 100) \wedge (t' - \theta_1 \geqslant 36) \wedge (t' - t - \theta_1 \geqslant 1) \wedge (t + \theta_1 \leqslant 135) \\ \mathbf{z}_K(t, t', \theta_2) := (20 \leqslant t \leqslant 120) \wedge (t' \leqslant 200) \wedge \\ (1 \leqslant \theta_2 \leqslant 100) \wedge (45 \leqslant t' - \theta_2 \leqslant 135) \wedge (t' - t \geqslant 1 + \theta_2)\end{aligned}$$



(a) Projection of $\mathbf{z}_I$ on dimensions (t, t', θ_1).



(b) Projection of $\mathbf{z}_I$ on dimensions (t, t', θ_2).



(c) Projection of $\mathbf{z}_I$ on dimensions (t, t', θ_3).

Figure 11: Projections of $\mathbf{z}_I$ on the two time dimensions t, t' and a parameter dimension (either θ_1 or θ_2 or θ_3).

3.6.2 *Real-Life Scenarios*

We now describe how we can use PSRE to find matches for real-life behaviours of interest. Electrocardiogram is a simple test that can be used to check the heart's rhythm and electrical activity. The signals $wecg_{205}$, $wecg_{221}$ and $wecg_{123}$ correspond to the Electrocardiograms (ECG) 205, 221 and 123 respectively taken from the MIT-BIH Arrhythmia Database [38, 60]. First, we consider the problems of matching and parametric identification for ECG pulses. Then, we show how we can utilize Booleanization of signals to aid with PSRE matching. After that, we describe how we can detect marine traffic rule violations involving ships crossing each other. Finally, we

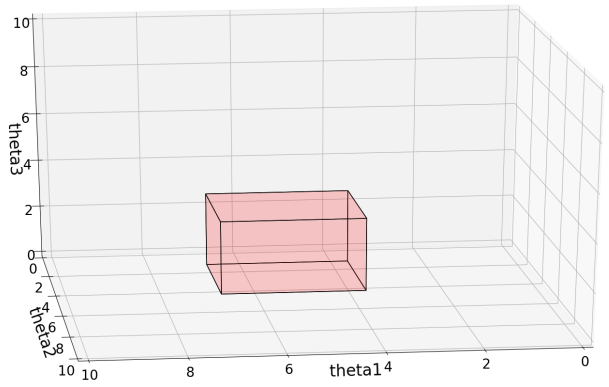
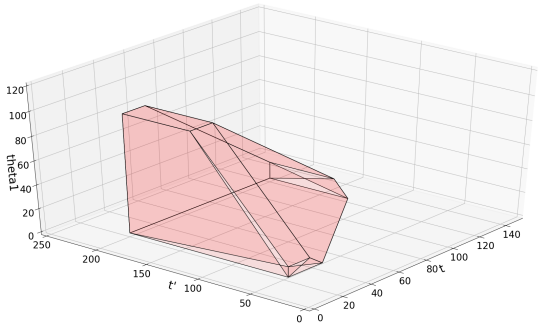
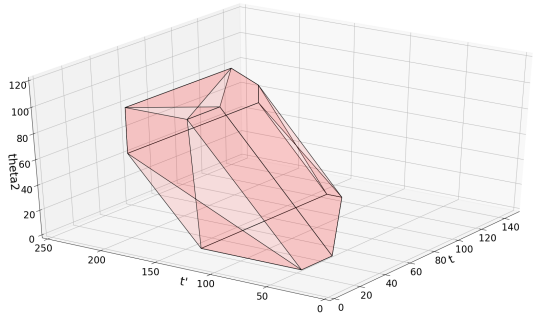


Figure 12: Projection of $\mathbf{z}_I$ on the parameter dimensions $(\theta_1, \theta_2, \theta_3)$.



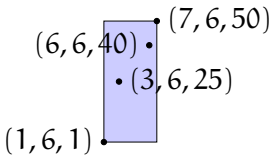
(a) Projection of $\mathbf{z}_K$ on dimensions (t, t', θ_1).



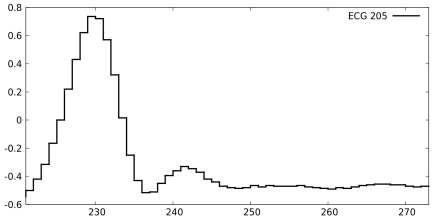
(b) Projection of $\mathbf{z}_K$ on dimensions (t, t', θ_2).

Figure 13: Projection of $\mathbf{z}_K$ on the time dimensions t, t' and a parameter dimension (either θ_1 or θ_2).

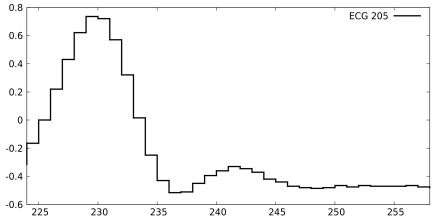
show how show how to detect aberrant heart rhythms (ventricular bigeminy and trigeminy to be precise) from the expert's labelling of pulses in ECG 106 from the MIT-BIH Arrhythmia Database [38, 60].



(a) Parameter set (rectangle) from matching of ϕ_2 and $wecg_{205}$



(b) $(t, t') = (221, 273)$ for $(\theta_1, \theta_2, \theta_3) = (6, 6, 40)$



(c) $(t, t') = (224, 258)$ for $(\theta_1, \theta_2, \theta_3) = (3, 6, 25)$

Figure 14: Matching ECG-205 signal ($wecg_{205}$) with ϕ_2

3.6.2.1 *Matching ECG Pulses Using Expression With Only Timing Parameters*

Consider ϕ_2 (Expr (2)) that has only timing parameters. The ECG signals are denoted by the symbol x in ϕ_2. The experimental results for matching of ϕ_2 for the three ECG signals are in Table 2. The "Error" column in Table 2 gives an estimate of the error involved when detecting ECG pulses using ϕ_2 when compared to an expert doctor.

Expression	Data	Size	Matches	Time	Error
ϕ_2 (Expr 2)	$wecg_{205}$	650000	2646	9s	2.83%
ϕ_2 (Expr 2)	$wecg_{221}$	650000	2638	9s	23%
ϕ_2 (Expr 2)	$wecg_{123}$	650000	1518	9s	0.2%

Table 2: ECG Matching Experiments

A parametric zone in the parametric match set for ϕ_2 and $wecg_{205}$ is given below:

$$(t + \theta_1 = 227) \wedge (\theta_2 = 6) \wedge (t' - \theta_3 = 233) \wedge (220 \leqslant t \leqslant 226) \wedge (234 \leqslant t' \leqslant 283)$$

Projecting it on to the parameter space gives the following rectangle:

$$(1 \leqslant \theta_1 \leqslant 7) \wedge (\theta_2 = 6) \wedge (1 \leqslant \theta_3 \leqslant 50)$$

Figure 14 shows an illustration for ϕ_2 and $wecg_{205}$. We take two points from the aforementioned rectangle and plot the corresponding matches in the signal.

3.6.2.2 *Pulse Train Detection for ECG-221*

In ECG-221 there are two kinds of pulses. One is a normal pulse and the other is an abnormal pulse with premature ventricular contraction. We observe an abnormal pulse once every few normal pulses. The following expression will match trains of normal pulses without being interrupted by abnormal pulses.

$$\phi_7 := \langle (p_\lambda \cdot \langle \mathtt{true} \rangle_{[20\theta_1, 35\theta_1]})^+ \rangle_{[\theta_2, \theta_2+100]} \tag{7}$$

Here, $\theta_1 \in [10, 11]$ and $\theta_2 \in [400, 650000]$. The parameter θ_1 captures the permitted time separation between two normal pulses. And θ_2 expresses the bounds on the length of the train of pulses. The propositional variable p_λ indicates the Boolean labelling signal given by the expert. It is true at the peak of normal pulses for one time unit and false otherwise. Matching the expression ϕ_7 using the aforementioned Boolean signal gives 5173 parametric zones in 91 seconds. In Figure 15, we show the first six pulses of ECG-221. Out of these, we have four normal pulses and two abnormal pulses. The first pulse train of the two normal pulses starts at 220 time units. And the corresponding parametric zone is given below:

$$\begin{aligned}(t \leqslant 221) \wedge (t' - 35\theta_1 \leqslant 443) \wedge (\theta_1 \leqslant 11) \wedge \\ (t' - t - \theta_2 \leqslant 100) \wedge (\theta_2 \geqslant 400) \wedge (\theta_1 \geqslant 10) \wedge \\ (t' - 20\theta_1 \geqslant 442) \wedge (t \geqslant 220) \wedge (t' - t - \theta_2 \geqslant 0)\end{aligned}$$

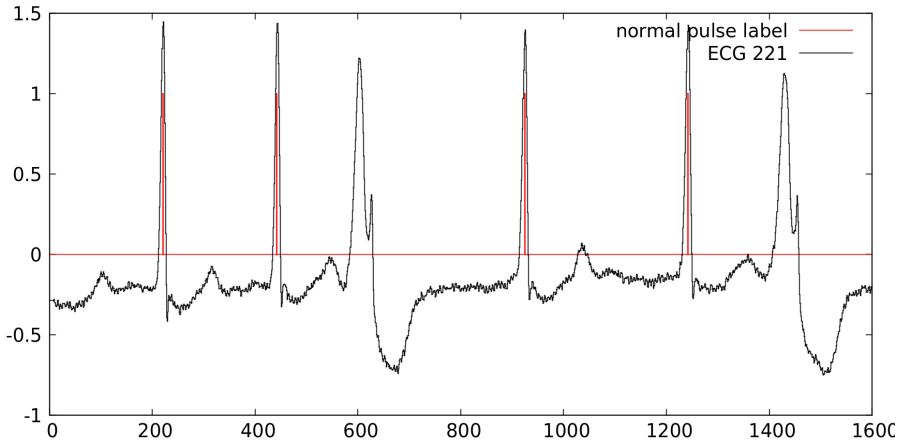


Figure 15: The first six pulses in ECG-221

3.6.2.3 *Parametric Identification for Labelled ECG-205 (Mini)*

Let us consider the sub-signal $wecg_{mini205}$ of $wecg_{205}$ containing the first ten pulses. $wecg_{mini205}$ has the size of 2500 time units. For each of these ten pulses we are given ten intervals of length 80 time units. These intervals are denoted by $\mathcal{I}_{205}$. The expressions ϕ_8,

ϕ_9 and ϕ_{10} (Expressions 8,9,10) have 4, 5 and 6 parameters respectively some of them being magnitude parameters. We would like to compute the solution sets for these three expressions with respect to $wecg_{mini205}$ and $\mathcal{I}_{205}$. The solution sets for all expressions contain exactly 75 polytopes. The computation times for ϕ_8, ϕ_9 and ϕ_{10} are 478s, 888s and 1881s respectively and increase exponentially with the number of parameters. Figure 16 shows the solution set $\mathcal{P}(\phi_7, wecg_{mini205}, \mathcal{I}_{205})$ projected on the parameters q_1, q_2 and q_3. The original ECG-205 contains around 2000 pulses and it becomes intractable to do parametric identification for all these 2000 pulses. Therefore, we next show how to reduce the time complexity by using Booleanization of signals as a pre-processing step, as proposed in Subsection 3.5.3.

$$\phi_8 := \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_2]} \cdot \langle q_1 \leqslant x \leqslant q_0 \rangle_{[0,q_3]} \cdot \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_2]} \tag{8}$$

$$\phi_9 := \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_3]} \cdot \langle q_1 \leqslant x \leqslant q_2 \rangle_{[0,q_4]} \cdot \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_3]} \tag{9}$$

$$\phi_{10} := \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_3]} \cdot \langle q_1 \leqslant x \leqslant q_2 \rangle_{[0,q_4]} \cdot \langle -q_0 \leqslant x \leqslant q_1 \rangle_{[0,q_5]} \tag{10}$$

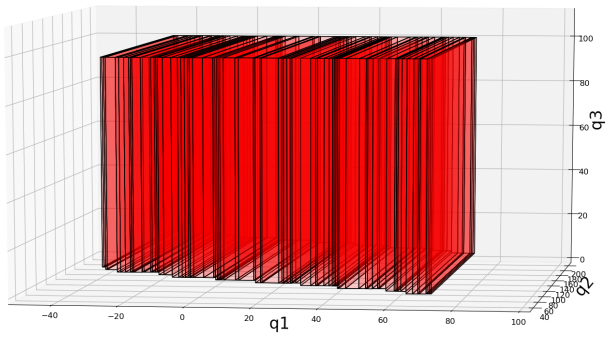


Figure 16: Solution set of ϕ_8 (Expression 8) and $wecg_{mini205}$ for labelling $\mathcal{I}_{205}$: Projected on q_1, q_2 and q_3.

3.6.2.4 *Booleanization and Matching for ECGs*

In Figure 8, the ECG pulse is shown as having a maximum between two minima. We can Booleanize an ECG signal x to get two Boolean

signals b_{max}, b_{min} for the approximate maximum and minimum respectively using the operators defined in [14] as below:

$$(b_{max} := \max_{[-150,150]} x - x \leqslant 0.05)$$
$$(b_{min} := x - \min_{[-10,10]} x \leqslant 0.05)$$

The Boolean signals b_{max} and b_{min} are true at time points where there are approximate maxima and minima respectively. More precisely, b_{max} is true at time points where the current value is within 0.05 of the maximum value in the time interval $[-150, 150]$ around it. Similarly, b_{min} is true at time points where the current value is within 0.05 of the minimum value in the time interval $[-10, 10]$ around it. Now we have an abstraction of an ECG in the form of ϕ_{11} (see Expr (11)).

$$\phi_{11} := b_{min} \cdot \langle \mathtt{true} \rangle_{[0,\theta_1]} \cdot b_{max} \cdot \langle \mathtt{true} \rangle_{[0,\theta_2]} \cdot b_{min} \tag{11}$$

We can now do matching using ϕ_{11} (Expr (11)) in which the time delay between the first minimum and the maximum is bounded by the parameter $\theta_1 \in [0, 20]$. And the time delay between the maximum and the second minimum is bounded by $\theta_2 \in [0, 20]$. Note that we have multiple matches per pulse, and the number of computed matches of ϕ_{11} with respect to ECGs 205, 221 and 123 are 8726, 8144 and 1971 respectively computed under 33s, 51s and 29s respectively.

3.6.2.5 *Detecting Marine Traffic Rule Violations on Florida Coast*

Here, we deal with finding marine traffic rule violations when two ships are crossing each other. We use scenarios from Marine Cadastre dataset[1] preprocessed[2] by [47]. From the preprocessed data we generate Boolean signals denoted by propositional variables p_1 and p_2. The variable p_1 is true at time points when the ego ship detects a crossing with another ship. The variable p_2 is true at time points where a maneuver appropriate for crossing scenario is executed.

$$\begin{aligned}\phi_{12} := \overline{p_1} \cdot (\langle \mathtt{true} \rangle_{dt} \cdot \langle p_1 \rangle_{\theta_r - dt} \cdot \mathtt{true} \wedge \langle \overline{p_2} \rangle_{\theta_r + \theta_m} \cdot \mathtt{true}) \vee \\ \overline{p_1} \cdot (\langle \mathtt{true} \rangle_{dt} \cdot \langle p_1 \rangle_{\theta_r - dt} \cdot \mathtt{true} \wedge \langle \mathtt{true} \rangle_{\theta_r} \cdot \langle p_1 \rangle_{2\theta_m} \cdot \mathtt{true})\end{aligned} \tag{12}$$

We formalise traffic rule violation for crossing ships as matching of ϕ_{12} (Expr (12)) where the constant $dt = 1$ is the time step, parameter $\theta_r \in [6, 20]$ is the reaction time and $\theta_m \in [6, 20]$ is the maneuver time. The expression captures two kinds of rule violations. The first being that a crossing detection is maintained until θ_r time and no maneuver is observed for the duration of the sum of reaction time and maneuver time. The second being that a crossing is detected like before but

1 https://marinecadastre.gov/ais/
2 doi.org/10.24433/CO.8258454.v2

the crossing situation is maintained for the duration of twice the maneuver time. The formulation has been heavily inspired by the STL formula for traffic rule R_3 (related to crossing ships) in Table 1 of [47]. We introduce parameters θ_r and θ_m as they can serve to estimate the reaction and maneuver time while matching. We did matching for 370 signals obtained from scenarios on the Florida coast. Each signal contains two components corresponding to crossing and maneuvering represented by p_1 and p_2 respectively. Each signal contains between 100 to 500 time steps. Out of the 370 signals, we detected a violation/-matching in one signal and none in the others. The detected violation involves a crossing situation of duration 5 time units followed by no maneuver for more than 12 time units. This violation is found in the scenario with the tag USOCEAN_Florida-20190103_7_T-1 and occurs at around 98 time units from the start of the signal. The matching time combined for all the 370 signals took around 1.26 seconds.

3.6.2.6 *Detecting Ventricular Bigeminy and Trigeminy in ECG-106*

The experiments discussed here from now on have been inspired by the case studies from [19]. Unlike how we did previously in this section, here we start with and build on top of the labelling of ECG-106 given by expert (doctor) from [38, 60]. The original real-valued signal for ECG-106 is denoted as $wecg_{106}$. The labelling which we process as a timed word (ω_{106}) consists of two types of pulses in the ECG. The first type is the normal pulse denoted by event N and the second type is a premature ventricular contraction denoted by event V. Here, we consider two types of aberrant behaviours called ventricular bigeminy and trigeminy. A ventricular bigeminy consists of an alternating sequence of normal (N) and abnormal (V) pulses. A ventricular trigeminy consists of a repeating sequence of a premature ventricular contraction (V) followed by two normal pulses. We use two event-based PTRE to detect these. The PTRE ϕ_{13} (Expr 13) is matched when we have two or more repetitions of $\underline{V} \cdot \underline{N}$ and therefore captures ventricular bigeminy. Similarly, the PTRE ϕ_{14} (Expr 14) is matched when we have two or more repetitions of $\underline{V} \cdot \underline{N} \cdot \underline{N}$ and therefore captures ventricular trigeminy.

$$\phi_{13} := \langle \langle \underline{V} \cdot \underline{N} \rangle_{\theta_1} \cdot (\langle \underline{V} \cdot \underline{N} \rangle_{[\theta_2,\theta_3]})^+ \cdot \underline{V} \rangle_{\theta_4} \tag{13}$$

$$\phi_{14} := \langle \langle \underline{V} \cdot \underline{N} \cdot \underline{N} \rangle_{\theta_1} \cdot (\langle \underline{V} \cdot \underline{N} \cdot \underline{N} \rangle_{[\theta_2,\theta_3]})^+ \cdot \underline{V} \rangle_{\theta_4} \tag{14}$$

The labelling timed word ω_{106} is of size 2027 of which 1507 are events of type N (normal pulse) and 520 are of type V (premature ventricular contraction). In both PTRE (ϕ_{13} and ϕ_{14}) there are four parameters $\theta_1, \theta_2, \theta_3, \theta_4 \in [0, 650000]$ whose values are bounded from above by the duration (650000) of ω_{106}. Their meaning and purpose are also the same for both the expressions. The first parameter θ_1 captures the exact duration of the first occurrence of the repeating sequence. The

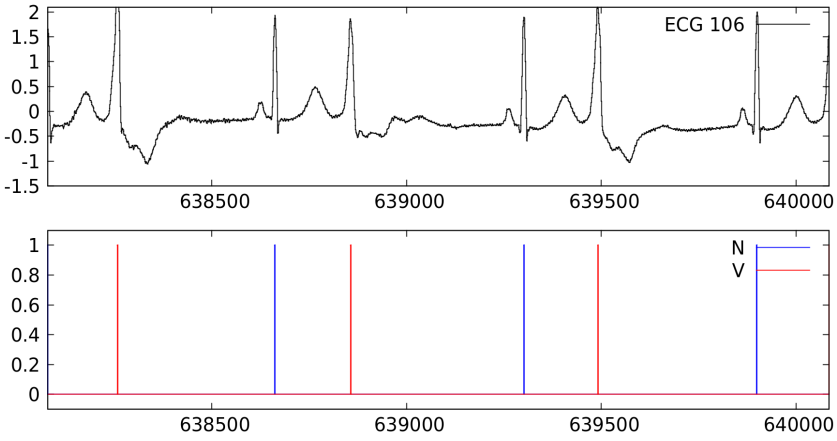


Figure 17: Ventricular Bigeminy occurring in time interval $[638078, 640085]$ shown for $wecg_{106}$ and ω_{106}.

second and third parameters (θ_2 and θ_3) capture the lower and upper bounds on the duration of the second and later occurrences. The final parameter θ_4 captures the whole duration of the matching pattern.

The parametric match-set of ϕ_{13} and ω_{106} for detecting ventricular bigeminy consists of 4665 parametric intervals. One of these is given below:

$$(t = 638078) \wedge (t' = 640085) \wedge (\theta_1 = 583) \wedge (0 \leqslant \theta_2 \leqslant 598) \wedge (640 \leqslant \theta_3 \leqslant 650000) \wedge (\theta_4 = 2007)$$

The corresponding matching segments $wecg_{106}[638078, 640085]$ and $\omega_{106}[638078, 640085]$ of the signal and labelling respectively for the above parametric interval are shown in Figure 17.

The parametric match-set of ϕ_{14} and ω_{106} for detecting ventricular trigeminy consists of 7 parametric intervals. One of these is given below:

$$(t = 613516) \wedge (t' = 616750) \wedge (\theta_1 = 995) \wedge (0 \leqslant \theta_2 \leqslant 1016) \wedge (1037 \leqslant \theta_3 \leqslant 650000) \wedge (\theta_4 = 3234)$$

The corresponding matching segments $wecg_{106}[613516, 616750]$ and $\omega_{106}[613516, 616750]$ of the signal and labelling respectively for the above parametric interval are shown in Figure 18. Another parametric interval in the match-set is given below:

$$(t = 392911) \wedge (t' = 395075) \wedge (\theta_1 = 991) \wedge (0 \leqslant \theta_2 \leqslant 966) \wedge (966 \leqslant \theta_3 \leqslant 650000) \wedge (\theta_4 = 2164)$$

The corresponding matching segments $wecg_{106}[392911, 395075]$ and $\omega_{106}[392911, 395075]$ of the signal and labelling respectively for the above parametric interval are shown in Figure 19.

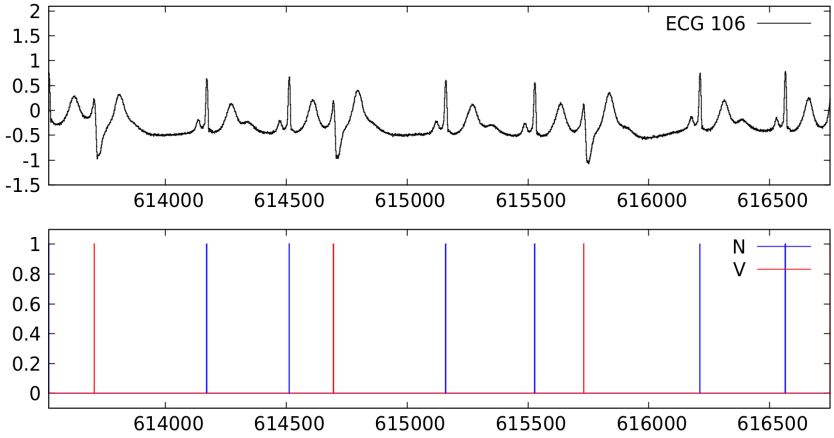


Figure 18: Ventricular Trigeminy occurring in time interval $[613516, 616750]$ shown for $wecg_{106}$ and ω_{106}.

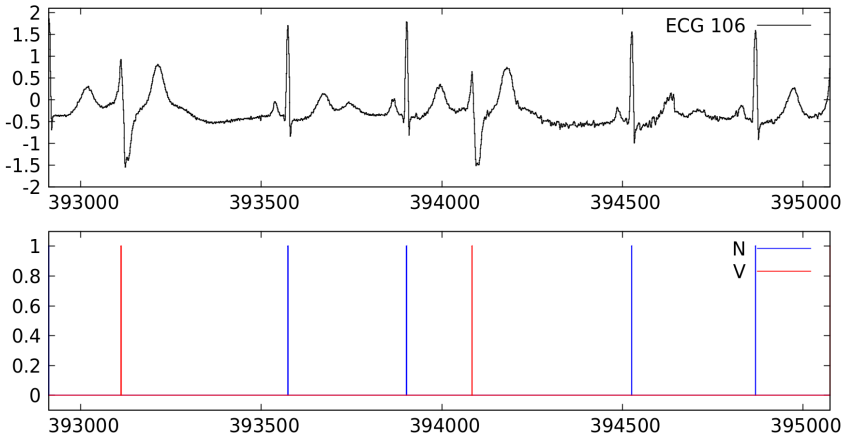


Figure 19: Ventricular Trigeminy occurring in time interval $[392911, 395075]$ shown for $wecg_{106}$ and ω_{106}.

FRONTS DE PARETO ET IDENTIFICATION PARAMÉTRIQUE

La conception de systèmes complexes implique généralement d'explorer les valeurs de certains paramètres qui sont optimales au regard de critères multiples. Dans de nombreux cas, nous ne disposons pas d'une solution optimale unique, mais de plusieurs solutions efficaces. Pour chacune de ces solutions, il n'est pas possible d'améliorer un paramètre sans détériorer un autre paramètre. L'ensemble de ces solutions efficaces est appelé front de Pareto. Une autre solution consiste à formuler le problème sous la forme de contraintes sur les différents coûts. Une approche "boîte noire" consiste à attribuer des valeurs aux paramètres et à utiliser un solveur de contraintes pour vérifier la faisabilité par rapport aux contraintes. Considérons le scénario de la détection de l'hypertension artérielle à l'aide d'une simple formule $P > P_0$. La variable P représente la pression mesurée et P_0 est le seuil inconnu qui implique une pression artérielle élevée. Nous devons déterminer la valeur seuil P_0 à partir des données annotées fournies par un expert concernant la valeur de P considérée comme une pression artérielle élevée. Si nous fixons la valeur de $P_0 = 0$, nous n'aurons aucun faux négatif, c'est-à-dire que nous ne manquerons aucun cas d'hypertension. Inversement, si nous attribuons $P_0 = \infty$, nous n'aurons aucun faux positif, c'est-à-dire que nous ne classerons jamais à tort un cas comme un cas d'hypertension artérielle. Nous pouvons constater que l'augmentation de la valeur seuil réduit les faux positifs et que sa diminution réduit les faux négatifs. Nous devons trouver un équilibre et déterminer la bonne valeur du seuil qui limite à la fois les faux positifs et les faux négatifs.

Nous allons maintenant donner un autre exemple qui concerne deux paramètres. Considérons la situation suivante. Un café nouvellement ouvert cherche des moyens de gagner des clients et de développer son activité. Il envisage deux options en matière de publicité. L'une consiste à essayer les annonces traditionnelles dans les journaux et l'autre à publier des annonces sur Internet. Nous désignons le nombre d'annonces dans les journaux et le nombre d'annonces sur Internet par p_1 et p_2 respectivement. Supposons que le nombre de clients gagnés soit proportionnel au nombre d'annonces et soit donné par l'expression $p_1 + p_2$. Le magasin souhaite gagner au moins trois clients. Cela signifie que la contrainte $p_1 + p_2 \geqslant 3$ doit être satisfaite. Si nous voulons y parvenir avec le moins de publicités possible, nous trouverons la solution sur la ligne $p_1 + p_2 = 3$. Dans l'ensemble des solutions, nous ne pouvons pas réduire le nombre d'un type de publicité sans augmenter le nombre de l'autre type, tout en respec-

tant la contrainte. Cette situation est appelée Pareto-optimale. La région exprimée à l'aide de la contrainte $p_1 + p_2 > 3$ est dite dominée par Pareto car nous pouvons réduire le nombre des deux types d'annonces et améliorer la situation. Nous appelons ce type de région "Upset". Examinons à présent les coûts liés à la diffusion des annonces. Supposons que les coûts d'une annonce dans un journal et d'une annonce sur Internet s'élèvent respectivement à 4 et 5 en unités monétaires. Le magasin ne peut pas dépenser plus de 20 en unités au total, d'où la contrainte $4p_1 + 5p_2 \leqslant 20$. Comme précédemment, les solutions Pareto-optimales seront sur la ligne $4p_1 + 5p_2 = 20$. La région exprimée à l'aide de la contrainte $4p_1 + 5p_2 < 20$ contient les solutions dominées par Pareto et est ce que nous appelons un Downset. L'ensemble des solutions réalisables respecte les contraintes de coût et d'exigence du client. Cet ensemble est compris entre deux ensembles Pareto-optimaux d'objectifs opposés. Dans l'exemple, nous avons besoin de plus de publicités pour avoir plus de clients, mais nous voulons aussi dépenser moins d'argent, ce qui nous oblige à réduire le nombre de publicités. Les solutions réalisables seront un équilibre entre ces deux contraintes. Dans cet exemple, nous connaissons les contraintes exactes et pouvons calculer directement l'ensemble des solutions réalisables. Si l'on nous donne des fonctions d'interrogation/oracle sans connaître les contraintes, nous aurons besoin d'un nouvel algorithme. Nous concevons et expliquons cet algorithme. Nous présentons d'abord quelques notions mathématiques utiles, puis nous décrivons l'algorithme à l'aide d'illustrations.

4

PARETO FRONTS AND PARAMETRIC IDENTIFICATION

Designing complex systems generally involves exploring the values of certain parameters that are optimal with respect to multiple criteria. In many cases we do not have a single optimal solution but various efficient solutions. For each of these solutions we cannot improve over one parameter without worsening with respective another parameter. The set of these efficient solutions is called a Pareto front. An alternative way is to formulate the problem as a constraints over the various costs. A black-box approach will be to assign values to parameters and use a constraint solver to check the feasibility with respect to the constraints. Let us consider the scenario of detecting high blood pressure using a simple formula $P > P_0$. The variable P stands for the measured pressure and P_0 is the unknown threshold crossing which implies high blood pressure. We need to determine the threshold value P_0 from annotated data given by an expert regarding what value of P is considered high blood pressure. If we set the value of $P_0 = 0$, then we would have no false negatives i.e. we will not miss any cases of high blood pressure. Conversely, if we assign $P_0 = \infty$, then we would have no false positives i.e. we will never wrongly classify a case as one of high pressure. We can see that increasing the threshold value reduces false positives and decreasing it reduces false negatives. We need to find a balance and find out the right value of threshold that bounds both false positives and false negatives.

We will now give another example which deals with two parameters. Consider the following situation. A newly opened coffee shop is looking for ways to gain customers and grow its business. It is considering two options for advertising. One is trying the traditional newspaper ads and the other is posting internet ads. We denote the number of newspaper ads and internet ads using p_1 and p_2 respectively. Let us assume that the number of customers gained is proportional to the number of ads and is given by the expression $p_1 + p_2$. The shop wants to gain at least three customers. Which means that the constraint $p_1 + p_2 \geqslant 3$ should be satisfied. An illustration is given in Figure 20. If we want to do this with the least number of ads we will find the solution on the line $p_1 + p_2 = 3$. In the solution set we cannot reduce the number of one type of ad without increasing the number of the other type and still satisfy the constraint. This situation is called Pareto-optimal. The green region in Figure 20 is said to be Pareto-dominated because we can reduce both number of

both types of ads and improve. We call this type of region as Upset later on in this chapter. Now, let us consider the costs involved in putting up ads. Assume that the costs of a single newspaper ad and a single internet ad are 4 and 5 currency units respectively. The shop cannot spend more than 20 units in total and therefore the constraint $4p_1 + 5p_2 \leqslant 20$. As before, the Pareto-optimal solutions will be on the line $4p_1 + 5p_2 = 20$. This is illustrated in Figure 21. The green region contains the Pareto-dominated solutions and is what we call a Downset later. The set of all feasible solutions respecting the constraints on cost and customer requirement is shown in Figure 22. This set is contained in between two Pareto-optimal sets of opposing targets. In the example, we need to have more ads to have more customers but we also want to spend less money which requires us to reduce the number of ads. The feasible solutions will be a balance between these two constraints. This situation is illustrated in Figure 22. In this example, we know the exact constraints and can directly compute the feasible solution set. If we are instead given query/oracle functions without knowing the constraints we would need a new algorithm. We devise and explain this algorithm in the current chapter. We first present some useful mathematical notions and then describe the algorithm with illustrations.

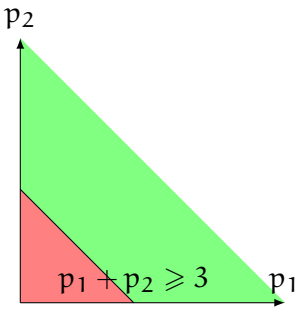


Figure 20: Pareto optimality for number of customers gained (Advertisement).

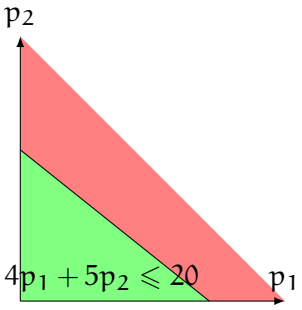


Figure 21: Pareto optimality for expenditure (Advertisement).

Partial order on $\mathbb{R}^n$, Upset, Downset and Pareto Front. Given two vectors $p, q \in \mathbb{R}^n$, vector p is lower than q, denoted by $p \leqslant q$, if

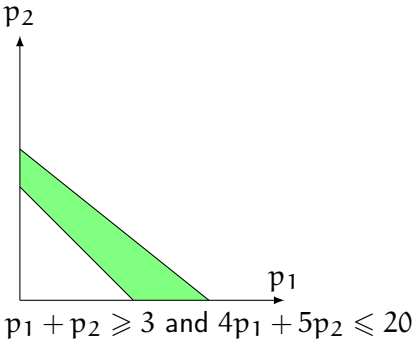


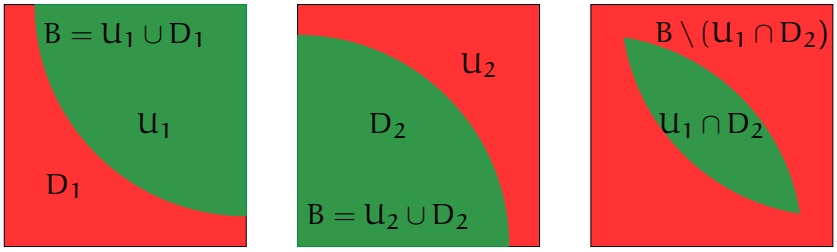
Figure 22: Pareto solution set (Advertisement).

$\forall i, p_i \leqslant q_i$. A set $\overline{X}$ is an *upset* if for all $p, q \in \mathbb{R}^n$ such that $p \leqslant q$ if $p \in \overline{X}$ then $q \in \overline{X}$. A set $\underline{X}$ is a *downset* if for all $p, q \in \mathbb{R}^n$ such that $q \leqslant p$ if $p \in \underline{X}$ then $q \in \underline{X}$. The boundary consisting of all the minimal elements of an upset (or all the maximal elements of a downset) is called a *Pareto front* in the field of multi-criteria optimization. The *box* between two vectors $\underline{x}$ and $\overline{x}$ with $\underline{x} \leqslant \overline{x}$ is $\lfloor \underline{x}, \overline{x} \rceil = \{y \mid \underline{x} \leqslant y \leqslant \overline{x}\}$.

Consider the box B from Figure 23. The first Pareto front P_1 separates the upset U_1 and the downset D_1 (Figure 23a), while the second Pareto front P_2 separates the upset U_2 and the downset D_2 (Figure 23b). The algorithms rely on the existence of two oracle functions defined over the box B. Given a query point x contained in B, the first oracle function $\rho_+(x)$ indicates whether x belongs to U_1 or not (i.e., $x \in U_1$). The second oracle function $\rho_-(x)$ works similarly and returns the membership of x to D_2 (i.e., $x \in D_2$).

Assuming the existence of a Pareto front, the Pareto front algorithm takes an oracle and a box to approximately compute the upset and downset. In particular, U_1 and D_1 are approximated using oracle ρ_+ and box B. Similarly, U_2 and D_2 are computed using oracle ρ_- and box B. Please note that the algorithm requires that the oracles are monotonic. In these examples, ρ_+ is increasing: for $p, q \in B$ with $p \leqslant q$, it applies that $\rho_+(q) \geqslant \rho_+(p)$. Similarly, the oracle ρ_- is decreasing: for $p, q \in B$ with $p \leqslant q$, it applies that $\rho_-(q) \leqslant \rho_-(p)$.

Conversely, the Pareto intersection algorithm, which is the main focus of this chapter, computes the intersection set contained between two Pareto fronts. This method applies when we are dealing with one increasing oracle and another decreasing oracle: it involves the intersection of the upset of the increasing oracle and the downset of the decreasing oracle. For example, it can approximately compute $U_1 \cap D_2$ (in Figure 23c) using oracles ρ_+, ρ_- and the box B. It computes approximately but directly, the intersection of U_1 and D_2. A less efficient way is to separately compute U_1 and D_2 and then intersect them.



(a) Showing D_1 and U_1. (b) Showing D_2 and U_2. (c) Showing $U_1 \cap D_2$.

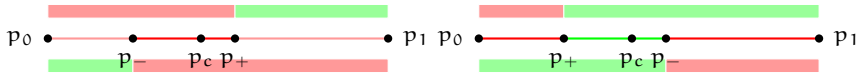
Figure 23: Illustration for Pareto algorithms

4.1 BOUNDARY AND INTERSECTION SEARCH IN 1-DIMENSION

The procedure boundary (Algorithm 2) finds the Pareto boundary of a monotonically increasing function on a given line using the classical idea of binary search. It is discussed in detail in Section 2.9 of Chapter 2. In Figure 5, we see the line $\langle \underline{x}, \overline{x} \rangle$ intersecting the Pareto front at the point z. Algorithm 2 makes queries using monotonically increasing Boolean-valued function ρ. It uses the idea of binary search to refine and find a line-segment $\langle \underline{y}, \overline{y} \rangle$ of length less than user defined ε containing z.

The procedure intersect (Algorithm 7) finds the intersection of two monotonic Boolean functions ρ_+ (increasing) and ρ_- (decreasing) on a line $\langle \underline{x}, \overline{x} \rangle$. Before starting intersection search on a line, by simple queries on the endpoints we can sometimes altogether discard (o_c=*discard*) or fully accept (o_c=*accept*) the bounding hyper-rectangle. This happens when the hyper-rectangle is wholly contained in a negative or a positive intersection. When this is not the case, we query for the values of ρ_+ and ρ_- at the midpoint. If a point in the intersection is found we return with the result on whether it is positive (o_c=*splitpos*) or negative (o_c=*splitneg*). Otherwise, we continue the search recursively by discarding the half segment not containing an intersection. This is possible because ρ_+ and ρ_- are monotonically increasing and decreasing respectively. In this way we end up either finding an intersection or returning a line segment of length equal to an error bound ε containing the intersection (o_c=*notfound*).On a line (p_0, p_1), we can have three outcomes of the search. The first two outcomes are when a point p_c in the positive intersection (Fig. 24b) or the negative intersection (Fig. 24a) is found. For these cases, we can divide the line into two segments (p_0, p_c) and (p_c, p_1). On these segments we can apply the classical binary search to find the Pareto

fronts corresponding to the monotonically decreasing and monotonically increasing functions. We call this operation an *expansion*. In Fig. 24a,24b, the points p_+, p_- represent the points where the Pareto fronts for the monotonically increasing and decreasing functions respectively intersect with the line (p_0, p_1). The third and last case is when no intersection has been found.



(a) Negative intersection. (b) Positive intersection.

Figure 24: Intersection on a line.

Algorithm 7 1D intersection search: $\texttt{intersect}(\langle \underline{x}, \overline{x} \rangle, \rho_+, \rho_-, \varepsilon)$

Input: $\ell = \langle \underline{x}, \overline{x} \rangle$ is a line segment, ρ_+ and ρ_- are monotonically increasing and decreasing Boolean-valued functions; and $\varepsilon \geqslant 0$ is an error bound.
Output: A line segment $\langle \underline{y}, \overline{y} \rangle$, an enum variable o_c which can take values {*notfound, accept, discard, splitpos, splitneg*}.
$\langle \underline{y}, \overline{y} \rangle = \langle \underline{x}, \overline{x} \rangle$, o_c = *notfound*
if $\rho_+(\underline{y})$ **and** $\rho_-(\overline{y})$ **then**
    return $\langle \underline{y}, \overline{y} \rangle$, o_c = *accept*
if (**not** $\rho_+(\overline{y})$) **or** (**not** $\rho_-(\underline{y})$) **then**
    return $\langle \underline{y}, \overline{y} \rangle$, o_c = *discard*
while $\overline{y} - \underline{y} \geqslant \varepsilon$ **do**
    $y = (\underline{y} + \overline{y})/2$
    if $\rho_+(y)$ **and** $\rho_-(y)$ **then**
        return $\langle y, y \rangle$, *splitpos* ▷ positive intersection
    else if (**not** $\rho_+(y)$) **and** (**not** $\rho_-(y)$) **then**
        return $\langle y, y \rangle$, *splitneg* ▷ negative intersection
    else if $\rho_+(y)$ **and** (**not** $\rho_-(y)$) **then**
        $\langle \underline{y}, \overline{y} \rangle = \langle \underline{y}, y \rangle$ ▷ left sub-interval
    else if (**not** $\rho_+(y)$) **and** $\rho_-(y)$ **then**
        $\langle \underline{y}, \overline{y} \rangle = \langle y, \overline{y} \rangle$ ▷ right sub-interval
return $\langle \underline{y}, \overline{y} \rangle$, o_c

4.2 EXTENDING TO MULTI-DIMENSIONAL CASE

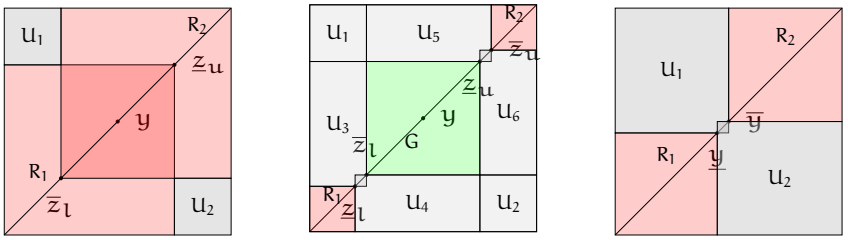
Algorithm 8 uses the result of the binary intersection search on the diagonal of a box to deduce which regions (inside the box) do or do not contain a solution and which are undecided. Then, it decomposes the undecided region into sub-boxes and recursively processes the resulting sub-boxes (see Fig. 25). There are three cases:

- *No intersection* has been found (see Fig. 25c). As a result of monotonicity, we know that the sub-boxes R_1 and R_2 do not contain a solution, and proceed with the remaining region which is decomposed into two overlapping sub-boxes U_1 and U_2. This decomposition is formulated in Section 4.2.1.

- A *negative intersection* has been found (see Fig. 25a). We can identify a line segment on the diagonal where a solution can not exist and deduce that the regions R_1 and R_2 do not contain a solution. The decomposition of the undecided region leads to two sub-boxes U_1 and U_2 (see Section 4.2.2).

- A *positive intersection* has been found (see Fig. 25b). We obtain the sub-boxes U_1 and U_2 as in the previous cases but use the procedure in Section 4.2.1 twice to obtain overlapping sub-boxes U_3, U_4, U_5 and U_6.

The decompositions into non-overlapping and overlapping sub-boxes are denoted by I_{nov} and I_{ov} in Algorithm 8 and explained in detail in Sub-sections 4.2.1 and 4.2.2.



(a) Negative intersection (b) Positive intersection (c) No intersection found

Figure 25: Illustration of sub-boxes.

Before continuing, we need a formal definition of sub-boxes resulting from subdivision based on points $\underline{y} < \overline{y}$ on the diagonal of a box $\lfloor \underline{x}, \overline{x} \rceil$. These sub-boxes are products of sub-intervals I_{α_i} where for each dimension, their bounds are taken among the following sequence of coordinates $\underline{x}_i < \underline{y}_i < \overline{y}_i < \overline{x}_i$.

Definition 4.2.1 (Sub-interval encoding). *A sub-interval $I_{\alpha_i} \subseteq [\underline{x}, \overline{x}]$ is encoded by its subscript $\alpha_i \in \{l, m, u, U, L, T\}$ which is a letter such that $\alpha_i = l$ for the lower interval $[\underline{x}_i, \underline{y}_i]$; $\alpha_i = U$ for its complement $[\underline{y}_i, \overline{x}_i]$; $\alpha_i = u$ for the upper interval $[\overline{y}_i, \overline{x}_i]$; $\alpha_i = L$ for its complement $[\underline{x}_i, \overline{y}_i]$; $\alpha_i = m$ for the middle interval $[\overline{y}_i, \overline{x}_i]$; and $\alpha_i = T$ for the whole interval $[\underline{x}_i, \overline{x}_i]$.*

Definition 4.2.2 (Sub-boxes). *Given $\alpha = (\alpha_1, \ldots, \alpha_n) \in \{l, m, u, U, L, T\}^n$ and four n-dimensional points $\underline{x} = (\underline{x}_1, \ldots, \underline{x}_n)$, $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$, $\underline{y} = (\underline{y}_1, \ldots, \underline{y}_n)$, $\overline{y} = (\overline{y}_1, \ldots, \overline{y}_n)$ such that $\underline{x} \leqslant \underline{y} \leqslant \overline{y} \leqslant \overline{x}$, the sub-box of $\lfloor \underline{x}, \overline{x} \rceil$ induced by α and $\lfloor \underline{y}, \overline{y} \rceil$ is $B_\alpha = \prod_{i=1}^{n} I_{\alpha_i}$ with I_{α_i} defined in Definition 4.2.1.*

Algorithm 8 Pareto front intersection algorithm

Input: A box $X = \lfloor \mathbf{0}, \mathbf{1} \rceil$; ρ_+ and ρ_- are respectively monotonically increasing and decreasing Boolean-valued functions; error bounds δ and ε.
Output: A set of boxes S containing the positive intersection of ρ_+ and ρ_- and a set L representing the undecided region such that $|L| \leqslant \delta$. All sets are represented by unions of boxes.
$L = \{X\}$; $S = \emptyset$ ▷ initialization
repeat
  pop first $\lfloor \underline{x}, \overline{x} \rceil \in L$ ▷ take the largest box
  $\{\langle \underline{y}, \overline{y} \rangle, o_c\} = \mathtt{intersect}(\langle \underline{x}, \overline{x} \rangle, \rho_+, \rho_-, \varepsilon)$ ▷ intersect search on the diagonal
  if o_c == *splitpos* **then** ▷ found a positive intersection.
    $\langle \underline{z}_l, \overline{z}_l \rangle = \mathtt{boundary}(\langle \underline{x}, \underline{y} \rangle, \rho_+, \varepsilon)$
    $\langle \underline{z}_u, \overline{z}_u \rangle = \mathtt{boundary}(\langle \overline{y}, \overline{x} \rangle, \neg\rho_-, \varepsilon)$
    $S = S \cup \lfloor \overline{z}_l, \underline{z}_u \rceil$
    $L = L \cup I_{nov}(\underline{x}, \overline{x}, \overline{z}_l, \underline{z}_u) \cup I_{ov}(\underline{x}, \underline{z}_u, \underline{z}_l, \overline{z}_l) \cup I_{ov}(\overline{z}_l, \overline{x}, \underline{z}_u, \overline{z}_u)$ ▷ see Fig. 25b
  else if o_c == *splitneg* **then** ▷ found a negative intersection.
    $\langle \underline{z}_l, \overline{z}_l \rangle = \mathtt{boundary}(\langle \underline{x}, \underline{y} \rangle, \neg\rho_-, \varepsilon)$
    $\langle \underline{z}_u, \overline{z}_u \rangle = \mathtt{boundary}(\langle \overline{y}, \overline{x} \rangle, \rho_+, \varepsilon)$
    $L = L \cup I_{nov}(\underline{x}, \overline{x}, \underline{z}_l, \underline{z}_u)$ ▷ see Fig. 25a
  else if o_c == *accept* **then**
    $S = S \cup \lfloor \underline{x}, \overline{x} \rceil$
  else if $o_c \neq$ *discard* **then** ▷ no intersection found
    $L = L \cup I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ ▷ see Fig 25c.
until $\mathtt{Vol}(L) \leqslant \delta$
return S, L

4.2.1 *Decomposition Into Overlapping Sub-boxes*

This decomposition is useful when we have to remove from a box $\lfloor \underline{x}, \overline{x} \rceil$ the downward closure of $\underline{y}$ (*i.e.* $B_{(l,\ldots,l)} = \lfloor \underline{x}, \overline{y} \rceil$) and the upward-closure of $\overline{y}$ (*i.e.* $B_{(u,\ldots,u)} = \lfloor \underline{y}, \overline{x} \rceil$). The resulting sub-boxes can overlap but this decomposition is only used with points that are ε-close. At least in one dimension i the overlap is restricted to the middle interval $[\underline{y}_i, \overline{y}_i]$ whose length is at most ε leading to a negligible volume when ε is small compared to the length of $[\underline{x}_i, \overline{y}_i]$ and $[\underline{y}_i, \overline{x}_i]$.

Definition 4.2.3 (Overlapping sub-boxes)**.** *Let $\underline{x}, \underline{y}, \overline{y}, \overline{x}$ be four n-dimensional points with $\underline{x} < \underline{y} < \overline{y} < \overline{x}$. We define*

$$I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = \{B_\alpha \mid \alpha \in \mathbb{D}_n\}$$

where $(\mathbb{D}_n)_{n\in\mathbb{N}}$ is a sequence of set of words defined inductively as follows:

$$\mathbb{D}_2 = \{\mathrm{LU}, \mathrm{UL}\}, \mathbb{D}_3 = \{\mathrm{LUT}, \mathrm{TLU}, \mathrm{UTL}\}$$

, and for $n \geqslant 4$

$$\mathbb{D}_{n+1} = \mathrm{T}\mathbb{D}_n \cup \mathrm{L}\mathrm{U}^n \cup \mathrm{U}\mathrm{L}^n.$$

As an example $\mathbb{D}_4 = \{\mathrm{TLUT}, \mathrm{TTLU}, \mathrm{TUTL}, \mathrm{LUUU}, \mathrm{ULLL}\}$

Proposition 4.2.4. *The set* $I_{ov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ *contains* $(2n-3)$ *boxes whose union is the complement in* $\lfloor \underline{x}, \overline{x} \rceil$ *of* $B_{(l,\ldots,l)} \cup B_{(u,\ldots,u)}$.

4.2.2 *Decomposition Into Non-overlapping Sub-boxes*

In case a negative intersection has been found the set of points in the upward-closure of $\underline{y}$ should be removed. This is the sub-box $B_{(U,\ldots,U)} = \lfloor \underline{y}, \overline{x} \rceil$. The same reasoning holds for the downward-closure of $\overline{y}$ which is $B_{(L,\ldots,L)} = \lfloor \underline{x}, \overline{y} \rceil$.

Definition 4.2.5 (Non-overlapping sub-boxes)**.** *We define* $\mathbb{A}_n$, $\mathbb{C}_n$, $\mathbb{E}_n$ *recursively as follows*

$$\mathbb{E}_0 = \mathbb{A}_0 = \mathbb{C}_0 = \emptyset \text{ and for } n \geqslant 1$$

$$\mathbb{E}_{n+1} = m\mathbb{E}_n \cup u\mathbb{A}_n \cup l\mathbb{C}_n, \quad \mathbb{A}_{n+1} = l\mathrm{T}^n \cup U\mathbb{A}_n,$$

$$\mathbb{C}_{n+1} = u\mathrm{T}^n \cup L\mathbb{C}_n.$$

Let $\underline{x}, \underline{y}, \overline{y}, \overline{x}$ *be* 4 n*-dimensional points with* $\underline{x} < \underline{y} < \overline{y} < \overline{x}$*. We define*

$$I_{nov}(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = \{B_\alpha \mid \alpha \in \mathbb{E}_n\}.$$

Proposition 4.2.6. *The set* $I_{nov}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ *contains* $(n^2 - n)$ *boxes whose union is the complement in* $\lfloor \underline{x}, \overline{x} \rceil$ *of* $B_{(U,\ldots,U)} \cup B_{(L,\ldots,L)}$.

As an illustration we give $\mathbb{A}_n, \mathbb{C}_n, \mathbb{E}_n$ for the dimensions one, two and three:

n	1	2	3
$\mathbb{A}_n$	l	$l\mathrm{T} \cup U l$	$l\mathrm{TT} \cup U l\mathrm{T} \cup \mathrm{UU} l$
$\mathbb{C}_n$	u	$u\mathrm{T} \cup L u$	$u\mathrm{TT} \cup L u\mathrm{T} \cup \mathrm{LL} u$
$\mathbb{E}_n$	$\emptyset$	$lu \cup ul$	$mlu \cup mul \cup ul\mathrm{T} \cup uUl \cup lu\mathrm{T} \cup l\mathrm{L}u$

4.3 TOY EXAMPLE OF 3D-INTERSECTION

Let us consider a toy example of intersection inside a 3-dimensional unit box $[0,1]^3$. The three dimensions are named p_1,p_2 and p_3. The increasing oracle is defined as:

$$\rho_+ := \left(\frac{\lfloor 10 p_1 \rfloor}{10} + p_2 > 1.0 \right)$$

Here, we use the floor function to produce a set bounded from below by a staircase function. The decreasing oracle which is defined as:

$$\rho_- := (p_1 + p_2 < 1.1)$$

The set corresponding to the decreasing oracle is bounded from above by a simple linear function. Note that the oracles do not utilize the third dimension p_3. This means that the intersection set will consist of nine pillars in which the height dimension is p_3. We see the set in 3D in Figure 26. The computed intersection set is approximate with an unexplored volume of $V_\delta = 1\%$. The three projections of this set onto 2D are shown in Figure 27. In Figure 27a, we see that the projection on (p_1, p_2) consists of right angled triangles as expected. In Figures 27b,27c we see nine approximate rectangles.

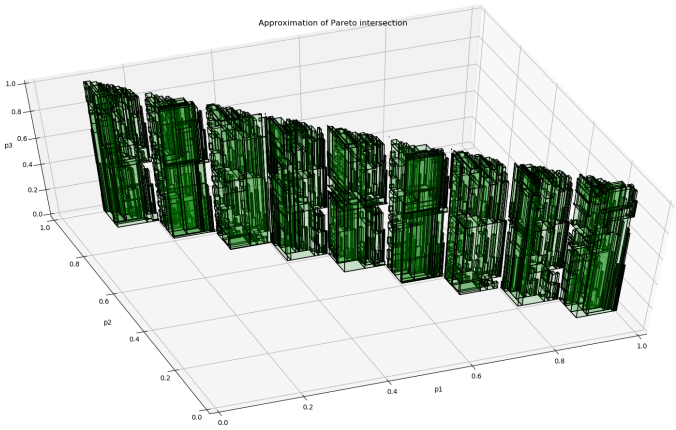
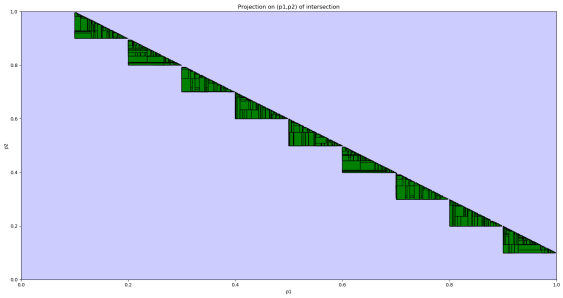
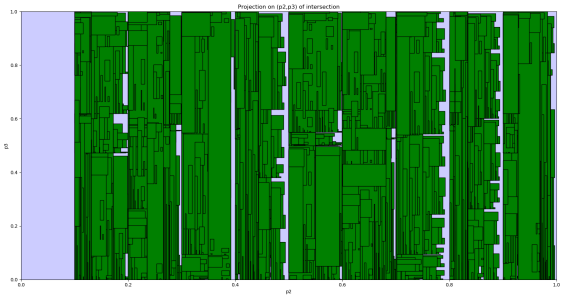


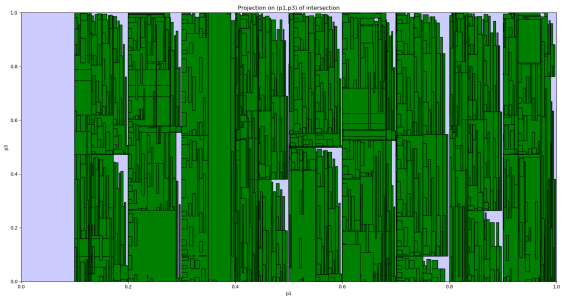
Figure 26: Toy example: 3D intersection.



(a) Toy example: Intersection projected on (p_1, p_2).



(b) Toy example: Intersection projected on (p_2, p_3).



(c) Toy example: Intersection projected on (p_1, p_3).

Figure 27: Toy example: Projections in 2D of 3D intersection.

APPRENTISSAGE D'UN PRÉDICTEUR DE MOTIFS À PARTIR DE SIGNAUX ÉTIQUETÉS

Dans le domaine de l'intelligence artificielle (Artificial Intelligence, AI), l'apprentissage supervisé consiste à trouver l'hypothèse qui correspond le mieux aux observations et à l'annotation donnée par un expert. Ces dernières années, des progrès considérables ont été réalisés dans l'apprentissage des réseaux neuronaux profonds (Deep Neural Networks, DNN), qui peuvent effectuer une myriade de tâches telles que reconnaître des objets dans des images, jouer à des jeux (par exemple aux échecs) et conduire des voitures. L'apprentissage des réseaux neuronaux profonds implique généralement une classe d'hypothèses représentée à l'aide d'une fonction paramétrée contenant l'application alternée de fonctions linéaires et non linéaires, parfois avec une profondeur d'une centaine de couches. Le processus d'apprentissage consiste à trouver les valeurs des paramètres qui permettent à la fonction de correspondre au mieux aux observations en fonction de certains critères d'optimisation. Nous reprenons le concept d'apprentissage supervisé et l'appliquons à des spécifications sur des séries de données temporelles. Au lieu de trouver un vecteur unique de valeurs de paramètres, nous nous attaquons au problème de la recherche approximative de l'ensemble des valeurs de paramètres qui rendent la spécification applicable aux séries temporelles observées dans les limites d'erreurs données par l'utilisateur (par exemple, les faux positifs et les faux négatifs). En plus d'être explicables, les spécifications que nous apprenons sont facilement lisibles et compréhensibles par les humains. Cela contraste avec les réseaux neuronaux qui sont généralement considérés comme des boîtes noires. Il est généralement difficile d'expliquer pourquoi et comment un réseau neuronal fait une prédiction.

Pour fournir une approche générique qui ne soit pas liée à un formalisme de spécification particulier, nous introduisons la notion de prédicteurs de motifs paramétriques (Parametric Pattern Predictor, PPP). Un PPP est un opérateur paramétrique qui transforme un signal non étiqueté en un signal booléen d'étiquetage qui est vrai aux points temporels où le modèle est prédit. Nous nous intéressons aux PPP croissants : lorsque les valeurs des paramètres augmentent pour un signal donné, l'ensemble des points temporels où la formule est vraie s'élargit. Nous introduisons la mesure epsilon-count pour quantifier dans quelle mesure une formule est vraie. En combinant cette mesure avec le PPP, nous développons un cadre pour l'apprentissage des spécifications. Nous appliquons ce cadre à des scénarios réels

afin de caractériser et de classifier les signaux d'électrocardiogramme (ECG).

5 LEARNING PATTERN PREDICTOR FROM LABELS

In the field of Artificial Intelligence (AI) supervised learning involves finding the hypothesis that best matches the observations and the annotation given by an expert. In recent years there has been tremendous progress in learning DNN that can perform a myriad of tasks like recognizing objects in images, playing games (e.g. chess) and driving cars. Learning DNN generally involves a hypothesis class represented using a parameterized function containing alternating application of linear and non-linear functions sometimes with a depth of hundred of layers. The learning process involves finding the parameter values that make the function best match the observations with respect to some optimization criteria. We lift the concept of supervised learning and apply it to specifications over time-series data. Instead of finding a single vector of parameter values, we tackle the problem of finding approximately the set of all parameter values that make the specification applicable to the observed time-series within user-given bounds on errors (e.g. false positives and false negatives).

To provide a generic approach which is not tied to a specific specification formalism, we introduce the notion of parametric pattern predictors (PPP). A PPP defined using a parameter vector p is a parametric operator Ψ_p that transforms unlabelled signal s to a labelling Boolean signal $\Psi_p(s)$ that is true on time points where the pattern is predicted. We focus our attention on PPPs that are increasing: when the value of p increases for any given signal s, the set of time points where $\Psi_p(s)$ is true expands.

5.1 SPECIFICATION LEARNING FRAMEWORK

In our framework, we allow the learned specification Ψ_p to produce some false positives and false negatives on parts of the training signals, i.e., there are time points where Ψ_p predicts a pattern while there is none, or misses it, respectively. We are interested in computing several sets of parameter valuations (called *solution sets*) which ensure that the "quantities" of false positive and/or false negatives are lower than given bounds. To define such quantities, we can use neither counts of time points or of intervals nor the Lebesgue measure since, as we will see later, these measures are not suitable. Instead we adapt the notion of ϵ-separated set from information theory [44] to propose a new measure, called ϵ-count, with suitable properties (Prop. 5.1.3). The notion of *support* which is used in the definition of

ϵ-count is defined as follows. The *support* of a signal w denoted by $\mathtt{supp}(w)$ is the smallest closed set that contains the set $\{t \mid w(t) \neq 0\}$.

5.1.1 *Parametric Pattern Predictor*

The labels of patterns in our problem are modelled using Boolean signals that we call *labelling signals*. A labelling signal λ_s for a signal s being 1 or 0 at a time point indicates respectively the occurrence or absence of a pattern in s at this time point. Particular cases of labelling signals are those whose support is a list of time points where patterns occur. In these cases, a pattern is a discrete event, and several labelling signals can be merged together to form what is called a timed word in timed automata theory [4]. We prefer using Boolean signals in continuous time for two main reasons. We want to allow patterns to have duration, that is their occurrence lasts continuously throughout a time interval (composed thus of uncountable number of points). They can be considered both as input or output signals for monitoring tools for temporal properties in dense time, such as StlEval [14] which we will use for our experiments.

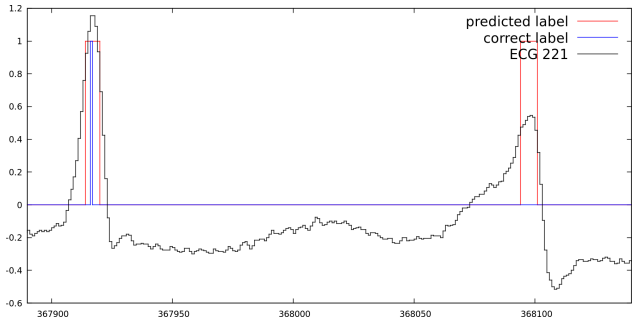


Figure 28: Showing the single false positive of $\Psi^{ch}_{(8.20,0.64,-0.44)}$ for ECG 221

Example 1. *Consider electro-cardiograms (ECG) from the MIT-BIH Arrhythmia Database of Physionet [38, 60]. They are provided with annotations of time-stamps where normal or abnormal peaks occur. The annotations for the normal peaks can be modelled into a labelling signal that is* 1 *when a normal peak occurs and* 0 *everywhere else. A portion of ECG 221 is depicted as in Fig. 28 where the blue labelling signal comes from the database.*

Our aim is to develop a pattern predictor, a tool that generates a labelling signal for a given signal. For ECG signals, it is used to annotate them with normal peaks, such as in Fig. 28 the red signal is predicted by our tool.

Definition 5.1.1 ((Increasing) Parametric Pattern Predictor (IPPP))**.** *A* parametric pattern predictor *(PPP) is a function that maps a vector* p *of reals to an operator* Ψ_p *that maps real-valued signals to Boolean-valued signals.* Ψ *is said* increasing *if for all* $p \leqslant p'$*, for all signal* s*,* $\forall t \in [0, l)$ *with* l *the length of* s*,* $\Psi_p(s)(t) \leqslant \Psi_{p'}(s)(t)$.

Example 2. *Formula (15) specified in the extended STL[1] [14] gives a simple and rough characterization of a normal ECG peak.* $\Psi^{ch}_{(p_1,p_2,p_3)}(s)(t) = 1$ *if the maximum of* s *on* $[t-p_1, t+p_1]$ *is above* $-p_3$*, and its variation is within the bound* p_2 *on* $[t-c, t-p_1]$ *and on* $[t+p_1, t+c]$*. The parameter domains are* $p_1 \in [0, 70]$*,* $p_2 \in [0, 1]$ *and* $p_3 \in [-1, 0]$*. Here,* $c = 70$ *is a constant representing an upper limit on* p_1*. Note that if one increases* (p_1, p_2, p_3)*, the property is easier to achieve.*

$$\begin{aligned}\Psi^{ch}_{(p_1,p_2,p_3)} := ((\mathrm{Max}_{[-c,-p_1]}\, s - \mathrm{Min}_{[-c,-p_1]}\, s) \leqslant p_2) \wedge \\ ((\mathrm{Max}_{[-p_1,p_1]}\, s) \geqslant -p_3) \wedge ((\mathrm{Max}_{[p_1,c]}\, s - \mathrm{Min}_{[p_1,c]}\, s) \leqslant p_2) \quad (15)\end{aligned}$$

We remark again that although our work uses the extended STL [14], our framework can be applied to other specification formalisms. Indeed, many matching problems can be cast into an IPPP, for instance matching as closely as possible a signal for the longest time possible. More formally, we can define an IPPP Ψ^{π} such that $\Psi^{\pi}_{(p_1,p_2)}(s)$ is 1 at time t if the signal s restricted on the interval $[t, t+T-p_1]$ is point-wise p_2-close to a given signal π (representing a shape of interest), that is, $\forall t' \in [0, T-p_1], |s(t+t') - \pi(t')| \leqslant p_2$. The idea of matching such predefined shapes is inspired by the work on shape expression [63].

5.1.2 *Quantifying Mismatches via ϵ-count*

A labelled signal (s, λ_s) is a pair of signal s and labelling signal λ_s. We aim at learning parameters p for an IPPP Ψ_p so that for every given labelled signal (s, λ_s), the labelling signals $\Psi_p(s)$ and λ_s should match together as much as possible. We measure two kind of mismatches by measuring "how often" the two following signals are true. The *false positive signal* $\neg\lambda_s \wedge \Psi_p(s)$ indicates when the predictor predicts an occurrence when there is none. The *false negative signal* $\lambda_s \wedge \neg\Psi_p(s)$ indicates when the predictor misses an actual occurrence.

The phrase "how often" may make one think of counting events like occurrences of a peak. However we cannot count the points where a Boolean formula is true since they are in general uncountable. Counting the intervals where a Boolean signal is true is also problematic since it is not always increasing with the support of the signal. For example, a Boolean signal defined as $b(t) := s(t) < p$ has support that increases with p, but such interval counting is not monotonically increasing with p. This is explained using the Figure 29. The signal $s(t)$ is shown in black and the Boolean signals obtained by assigning $p = 2, 3, 6$ in $b(t) := s(t) < p$ are shown in different colours. The number of intervals where the Boolean signal is true is two for $s(t) < 3$ but only one for $s(t) < 2$ and $s(t) < 6$. This shows that

1 Here and in the rest of the manuscript we slightly simplified the syntax of [14] by replacing $(\mathrm{On}_{[a,b]}\, \mathrm{Max}\, s)$ by $(\mathrm{Max}_{[a,b]}\, s)$ whose value in t is $\max_{t' \in [t+a, t+b]} s(t')$.

counting the number of intervals is not monotonic. Also there can be infinitely many intervals. Last but not least, the most standard measure of subsets of $\mathbb{R}$ is the Lebesgue's measure. This is not convenient for our purpose because a signal whose support is the disjoint union of many intervals of almost-null measure which are quite far apart will entails a small measure while for such a signal we want instead a big "count" because it can represent the number of mismatches. In this work we introduce the notion of ϵ-count, inspired by the notions of ϵ-separated sets and ϵ-capacity proposed in [44]. When applied to Boolean signals, this count is monotonic and also gives a discrete number.

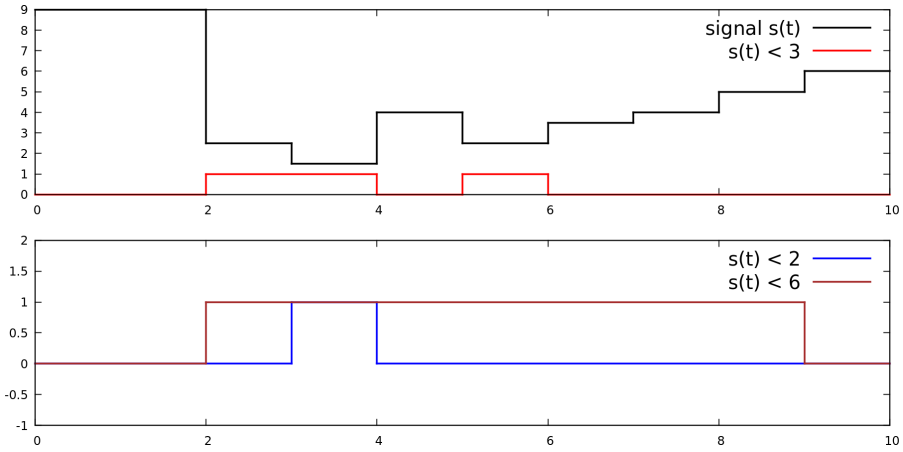


Figure 29: Non-monotonicity of interval count.

Definition 5.1.2 (ϵ-separated set and ϵ-count)**.** *Given a boolean signal w, a set S of reals is ϵ-separated w.r.t. w if $S \subseteq \mathit{supp}(w)$ and for every $t, t' \in S$ with $t \neq t'$, it holds that $|t - t'| \geqslant \epsilon$. The ϵ-count of a signal w is $c_\epsilon(w) = \max\{|S| \mid S \text{ is } \epsilon\text{-separated w.r.t. } w\}$.*

Proposition 5.1.3. *The ϵ-count of a signal w is determined in a greedy manner with the following recursive equations: $c_\epsilon(\mathbf{0}) = 0$ and $c_\epsilon(w) = 1 + c_\epsilon(w')$ where $w'(t) = 0$ if $t < \epsilon + \min(\mathit{supp}(w))$ and $w'(t) = w(t)$ otherwise.*

Proof. Intuitively, the support of the signal w' is obtained from that of the signal w by removing an interval of length not greater than ϵ, meaning that in the difference between the two supports we cannot put two points that are ϵ-separated. More formally, let S be a maximal ϵ-separated set w.r.t. w and $t_{\min}$ be the minimum of S, we first show that the set $S' = S \setminus \{t_{\min}\}$ is ϵ-separated w.r.t. w'. Every point t of this set satisfies $|t - t_{\min}| \geqslant \epsilon$ and thus $t > t_{\min} + \epsilon$. This means that $S' \subseteq \mathsf{supp}(w')$. Hence $|S'| \leqslant c_\epsilon(w')$. Since $|S'| = |S| - 1$ and S is maximal we have that $c_\epsilon(w) - 1 \leqslant c_\epsilon(w')$. It remains to show the inequality $c_\epsilon(w) - 1 \geqslant c_\epsilon(w')$ to establish the equality. Let now S' be a maximal ϵ-separated w.r.t. w' and let $S = S' \cup \{\min(\mathsf{supp}(w))\}$.

Then S is ϵ-separated w.r.t. w, from which we know that $|S| \leqslant c_\epsilon(w)$. Since $|S| = |S'| + 1 = c_\epsilon(w') + 1$, we obtain the suited inequality. □

Proposition 5.1.4. *The following proposition states useful properties of the ϵ-count. The first two statements are not hard to see. We prove only the third statement.*

1. *The ϵ-count is null iff it is applied to the constant signal* $\mathbf{0}$.
2. *The ϵ-count is increasing: if $w \leqslant w'$ then $c_\epsilon(w) \leqslant c_\epsilon(w')$.*
3. *The ϵ-count satisfies a triangular inequality: $c_\epsilon(w \vee w') \leqslant c_\epsilon(w) + c_\epsilon(w')$.*

Proof. Let S'' be a maximal ϵ-separated set w.r.t. $w \vee w'$ so that $|S''| = c_\epsilon(w \vee w')$. Let $S = S'' \cap \mathtt{supp}(w)$. The set S is hence ϵ-separated w.r.t. w and we thus have $|S| \leqslant c_\epsilon(w)$. Let $S' = S'' \setminus S$ so that in particular $|S''| = |S| + |S'|$. S' is included in $\mathtt{supp}(w \vee w') \setminus \mathtt{supp}(w)$ so it is included in $\mathtt{supp}(w')$. The set S' is hence ϵ-separated w.r.t. w' and we thus have $|S'| \leqslant c_\epsilon(w')$. We can conclude $c_\epsilon(w \vee w') = |S''| = |S| + |S'| \leqslant c_\epsilon(w) + c_\epsilon(w')$. □

We explain the computation of ϵ-count using an analogy illustrated in Figure 30. We consider a Boolean signal as a one-dimensional landscape. Where the signal is true is considered as land and where it is false is considered water. We have a one legged frog that starts from the beginning of the signal and traverses to reach the end of the signal. When on land it jumps by exactly ϵ and swims when in water. The number of footsteps left by the frog going from the beginning to the end is equal to the ϵ-count of the signal.

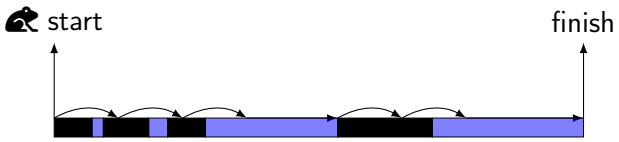


Figure 30: ϵ-count analogy illustration

5.1.3 *Parametric Identification Problems*

Given bounds $\mathtt{f}_+$, $\mathtt{f}_-$ on the allowed ϵ-count of false-positives and false-negatives, we are interested in the following three sets:

$$\mathtt{Dom}{+}(\Psi, \mathcal{S}, \mathtt{f}_+) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\Psi_p(s) \wedge \neg\lambda_s) \leqslant \mathtt{f}_+\}, \tag{16}$$

$$\mathtt{Dom}{-}(\Psi, \mathcal{S}, \mathtt{f}_-) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\neg\Psi_p(s) \wedge \lambda_s) \leqslant \mathtt{f}_-\}, \tag{17}$$

$$\mathtt{DomInter}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-) = \mathtt{Dom}{+}(\Psi, \mathcal{S}, \mathtt{f}_+) \cap \mathtt{Dom}{-}(\Psi, \mathcal{S}, \mathtt{f}_-). \tag{18}$$

For convenience, we call them respectively the *positive*, *negative* and *intersection* solution sets. It is also of great interest to compute the

set of couples $(\mathtt{f}_+, \mathtt{f}_-)$, called *set of feasible error bounds*, for which a solution exists:

$$\mathcal{P}(\Psi, \mathcal{S}) = \{(\mathtt{f}_+, \mathtt{f}_-) \mid \mathtt{DomInter}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-) \neq \emptyset\}. \tag{19}$$

In addition, we are interested in a relaxed version of the identification problem for false positive bounding, by tolerating a difference of σ time units in matching the labels. This can be done by replacing λ_s with the signal[2] $F_{[-\sigma,\sigma]}\,\lambda_s$ in (16). More concretely, the solution set of the corresponding σ-relaxed problem is:

$$\mathtt{Dom}{+}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+) = \{p \mid \forall (s, \lambda_s) \in \mathcal{S}, c_\epsilon(\Psi_p(s) \wedge \neg F_{[-\sigma,\sigma]}\,\lambda_s) \leqslant \mathtt{f}_+\}.$$

Hence, the corresponding relaxed version of the intersection solution set (18) is

$$\mathtt{DomInter}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-) = \mathtt{Dom}{+}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+) \cap \mathtt{Dom}{-}(\Psi, \mathcal{S}, \mathtt{f}_-). \tag{20}$$

And, the relaxed version of $\mathcal{P}(\Psi, \mathcal{S})$ in (19) is

$$\mathcal{P}^{\sigma}(\Psi, \mathcal{S}) = \{(\mathtt{f}_+, \mathtt{f}_-) \mid \mathtt{DomInter}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-) \neq \emptyset\}. \tag{21}$$

Note that $\mathtt{Dom}{+}(\Psi, \mathcal{S}, \mathtt{f}_+)$ is a downset and $\mathtt{Dom}{-}(\Psi, \mathcal{S}, \mathtt{f}_-)$ is an upset (see the beginning of Section 5.1) because Ψ is increasing. Sets of this kind can be learned from membership queries as proposed in [55]. The set $\mathtt{DomInter}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-)$ is the intersection of an upset and a downset and so is $\mathtt{DomInter}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-)$. The sets $\mathcal{P}(\Psi, \mathcal{S})$ and $\mathcal{P}^{\sigma}(\Psi, \mathcal{S})$ are upsets and their minimal elements form Pareto fronts. We compute them via membership-queries for couples $(\mathtt{f}_+, \mathtt{f}_-)$. This is done via non-emptiness checking of $\mathtt{DomInter}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-)$ and $\mathtt{DomInter}^{\sigma}(\Psi, \mathcal{S}, \mathtt{f}_+, \mathtt{f}_-)$ respectively. This non-emptiness checking is an easier problem than computing the whole set.

5.2 EXPERIMENTS

We have implemented the algorithms proposed in this work and incorporated them as additions to ParetoLib [15] (a Python library for Pareto-Front learning) and the tool StlEval [14]. Implementations in the ParetoLib library only deal with upsets (or downsets). To satisfy this, one can easily replace some of the parameters with their opposite.

We now present some experimental results obtained by using our supervised learning framework to analyse labelled electrocardiogram (ECG). ECG signals capture information about electrical activity of the heart and can help detect anomalies in its functioning. We characterize several features (e.g. peaks and ditches) as parametric specification, and provide the intersection solution set for the involved parameters with the best possible trade-off between false positives and false negatives.

2 where $(F_{[-\sigma,\sigma]}\,\lambda_s)(t) = 1$ iff $\exists t' \in [t-\delta, t+\delta], s(t') = 1$.

5.2.1 *Learning STL Specifications for Labelled ECGs*

We present our results on three ECGs (100, 123, 221) each containing between 1500 to 2500 labelled pulses taken from the MIT-BIH Arrhythmia Database of Physionet [38, 60].

We use the formula (15) from Section 5.1 to characterize a parameter predictor Ψ^{ch} for a normal heart pulse. Given a set $\mathcal{S}$ of labelled ECG signals, the upper bounds f_- and f_+ on the numbers of false negatives and positives respectively and a matching tolerance value σ, we use the intersection algorithm to find the relaxed intersection solution set $\texttt{DomInter}^{\sigma}(\Psi^{ch}, \mathcal{S}, f_+, f_-)$ as defined in (20) (that is the set of parameter values p such that the predictor Ψ^{ch}_p can predict normal heart pulses on the labelled signal set $\mathcal{S}$ with the numbers of false positives and false negatives bounded by f_+ and f_-).

5.2.1.1 *Trade off between false negatives and false positives*

Using a modification of Algorithm 8, we query about emptiness of $\texttt{DomInter}^{\sigma}(\Psi^{ch}, \mathcal{S}, f_+, f_-)$ and compute the set $\mathcal{P}^{\sigma}(\Psi^{ch}, \mathcal{S})$ of feasible error bounds as defined in (21). We recall that Algorithm 8 explores the parameter space up to a given bound δ on volume. In our experiments we search until we have reduced the volume of the undecided region to V_δ percentage of the total volume of the parameter space. The Pareto front that we obtain asymptotically becomes exact as the value of V_δ tends to zero. In Fig. 32a and Fig. 32b, we show the Pareto front approximations for ECG-221 and ECG-123 respectively with $V_\delta = 0.1\%$. In Fig. 33a and Fig. 33b, we show the approximate Pareto fronts for ECG-100 with $V_\delta = 0.1\%$ and $V_\delta = 1\%$ respectively. In Fig. 33c these two Pareto fronts are shown superimposed; the front separating the brown and the green regions corresponds to $V_\delta = 1\%$. The Pareto front separating the red and the brown regions corresponds to $V_\delta = 0.1\%$ and is more accurate owing to better exploration.

One can see that, for ECG-221, the predictor (15) can match the labelling with no false negatives and only a single false positive (shown in Fig. 28). For ECG-123, it can match with a single false negative and no false positives. Looking at Fig. 33, it is pertinent to ask why the predictor (15) cannot match the labelling with better accuracy for ECG-100. Actually, our formula only takes the shape of the heart pulses into account but not their time period. Some heart pulses in ECG-100 are not labelled as normal because they violate the natural rhythm of the heart and arrive too soon or too late. It is thus not possible to distinguish them by considering only the shape. See the first pulse after 2000 time units in Figure 31. It is called an Atrial Premature Beat which should not be considered a normal peak. The new predictor $\Psi^{ch_b}_{(p_1,p_2,p_3,p_4)}$ (22) enriches the old predictor (15) with explicit information about the distance between peaks, which improves

the accuracy of our peak detector and reduces the number of errors to 3 false positives and 1 false negative. *peak* is an alias for the old predictor (15).

$$\Psi^{ch_b}_{(p_1,p_2,p_3,p_4)} := peak(p_1,p_2,p_3) \wedge F_{[0,p_4]}\, peak(p_1,p_2,p_3) \quad (22)$$

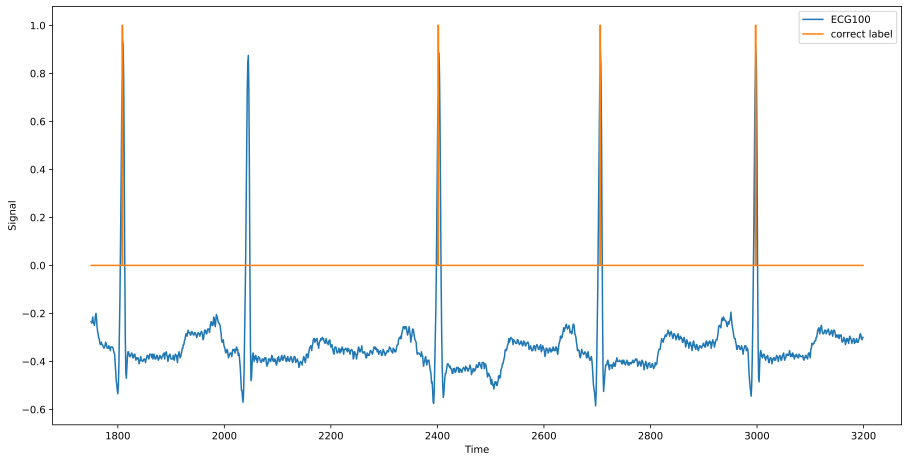
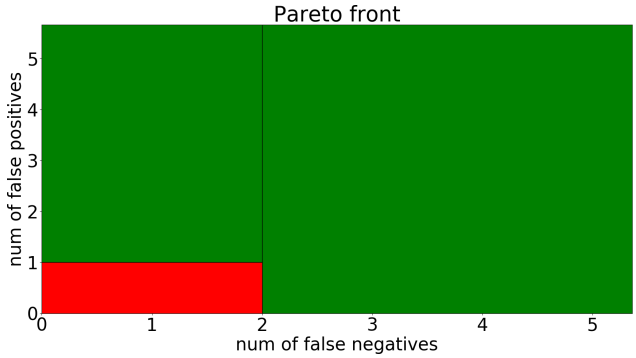
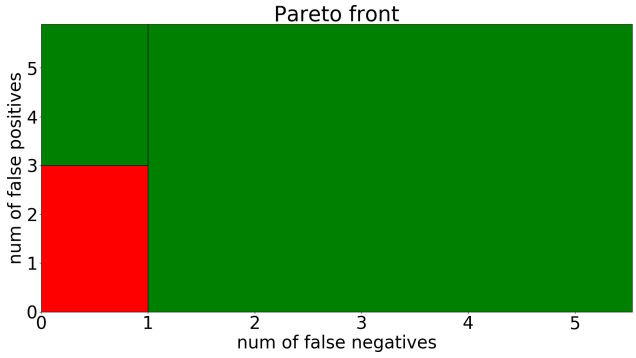


Figure 31: Excerpt from ECG 100.

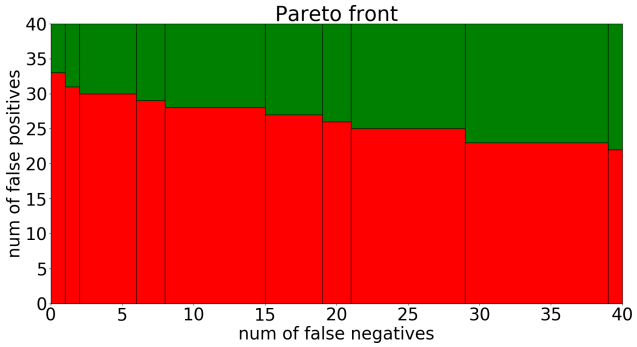


(a) ECG 221, $V_\delta = 0.1\%$, $t \approx 252s$

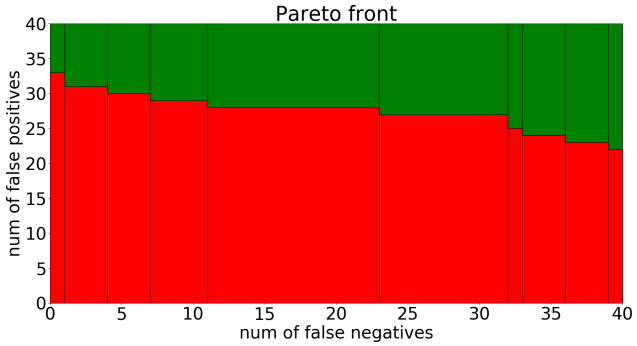


(b) ECG 123, $V_\delta = 0.1\%$, $t \approx 303s$

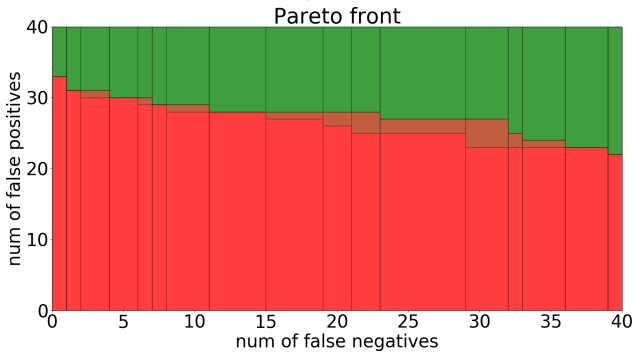
Figure 32: Pareto fronts for ECGs 221 and 123



(a) ECG 100, $V_\delta = 0.1\%$, $t \approx 11919s$



(b) ECG 100, $V_\delta = 1\%$, $t \approx 7903s$



(c) ECG 100, $V_\delta = 1\%$ vs $V_\delta = 0.1\%$

Figure 33: ECG 100: Decreasing V_δ gives a more accurate Pareto front

5.2.1.2 *3D intersections*

Once we have computed $\mathcal{P}^\sigma(\Psi^{ch}, \mathcal{S})$ with adequate accuracy, we can use Algorithm 8 to further explore the parameter space for different values of V_δ, f_- and f_+. Some exploration results are depicted in Fig. 34 and the associated computation time in Table 3.

5.2.2 *Classification of ECGs*

We now demonstrate the application of our approach to binary classification of signals, using the ECGFiveDays dataset from the UCR Time Series Classification Archive [32]. More concretely, we consider two classes of ECGs taken 5 days apart from the same person and

Table 3: Computation time for 3D intersection solution sets.

ECG n^0	f_-	f_+	$V_\delta = 1\%$	$V_\delta = 0.1\%$	$V_\delta = 0.01\%$	τ^1
221	0	1	62s	262s	1279s	19s
123	1	0	103s	592s	3189s	36s
100	0	33	758s	5273s	18670s	12s

[1] τ represents the time taken to find the first point in the solution set.

want to find a classifier that can correctly predict given an ECG, on which day it was observed ($\texttt{day}_1$ or $\texttt{day}_5$). In Figure 35, we first show an ECG from $\texttt{day}_1$ followed by one from $\texttt{day}_5$. In [58], an enumerative method over STL is applied to solve this problem in the absence of a priori information. By defining expressive and meaningful features, we show how more informative formulae can be obtained with less training data (23 traces as compared to 300 in [58]). The features are based on the well known theoretical modelling of ECG signal as P and T waves combined with a QRS complex (see e.g. [29]). For each ECG in the dataset, we observe two prominent peaks with a ditch in between. We define STL formulae to quantify some features of the signal around the ditch and the peaks as shown in Table 4. The range of feature values (D_1,D_5) for $\texttt{day}_1$ and $\texttt{day}_5$ observed in the ECGs are shown in Table 5. They can be computed using a slight modification of classical binary search.

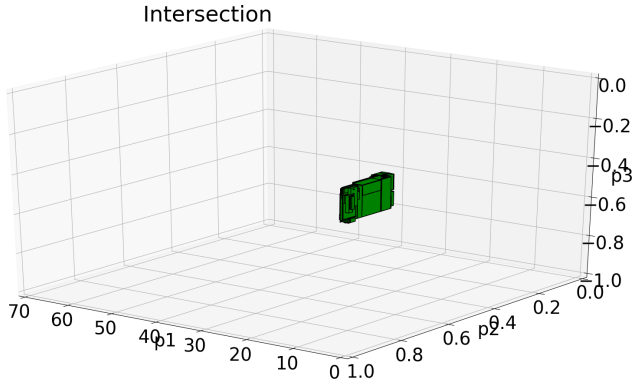
5.2.2.1 *Enumeration of formulae and finding a classifier.*

We rank the features using the measure[3] $m = \frac{|D_1 \bigtriangleup D_5|}{|D_1 \cup D_5|}$. We then enumerate all the distinct pairs (φ_i, φ_j) of features using lexicographical ordering over rank. For each pair we use the intersection algorithm to learn the parameters which make the disjunction formula $(\Psi := \varphi_i \vee \varphi_j)$ classify correctly the ECGs given in the training data. Let S_1 and S_2 be the labelled signals corresponding to the classes $\texttt{day}_1$ and $\texttt{day}_5$ of ECGs respectively. We compute the intersection of $\texttt{Dom}-(\Psi, S_1, f_-)$ and $\texttt{Dom}+(\Psi, S_2, f_+)$.

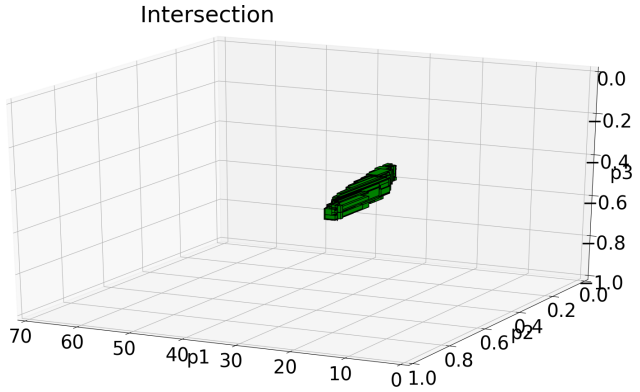
5.2.2.2 *Results for ECG5days classify.*

Table 6 summarizes our results for five different training and testing configurations. For the first configuration, we use the original 23 training traces and 861 testing traces from [35] without any changes. For the other configurations we split the training set of size 861 and use one portion for training and the other for testing. The number of traces used for training and testing are mentioned within brackets in

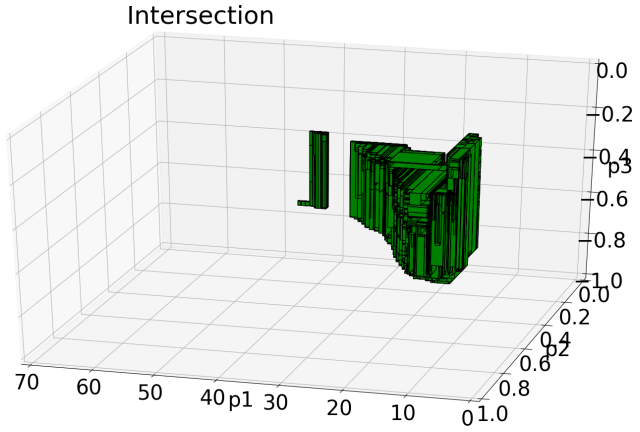
3 $\bigtriangleup$ is the symmetric difference of two sets.



(a) ECG 221,
$V_\delta = 0.01\%$, $f_- = 0$, $f_+ = 1$



(b) ECG 123,
$V_\delta = 0.01\%$, $f_- = 1$, $f_+ = 0$



(c) ECG 100, $V_\delta = 0.01\%$, $f_- = 0$, $f_+ = 33$

Figure 34: Case study 1: intersection solution sets in 3D

the first column of Table 6. For example, in the second row we indicate that for configuration 2 we use 100 traces for training and the remaining 761 for testing. Note that for configuration 4 we use all the 861 traces for training and have no traces for testing. For this configuration, the reason we did not find a solution could be because we required 100% training accuracy. For configuration 5, when we keep the same data split but allow $f_-, f_+ = 2$, we succeed in finding a for-

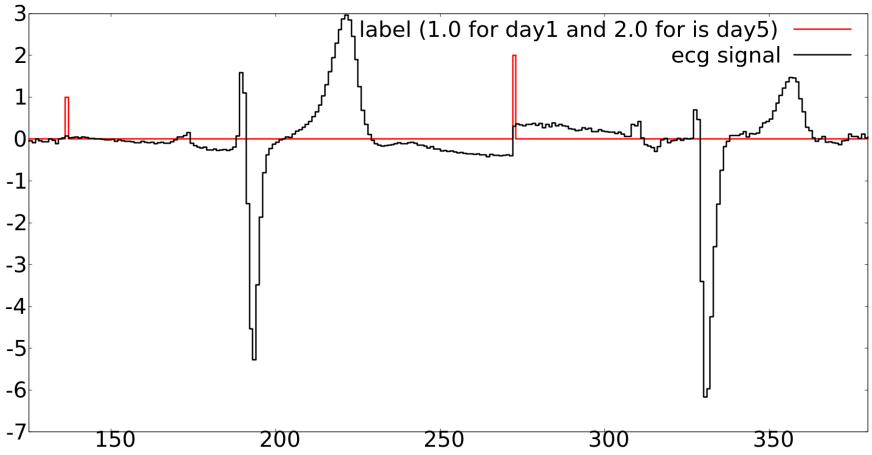


Figure 35: ECGs signals from $\texttt{day}_1$ and $\texttt{day}_5$

Table 4: Features and formulae.

Feature	Formula
Def. of `peak`	$(s \geqslant (\text{Max}_{[-10,10]}\ s)) \wedge s \geqslant 1$
Def. of `ditch`	$(s \leqslant (\text{Min}_{[-10,10]}\ s)) \wedge s \leqslant -1$
Depth of the ditch[1]	$(\text{Min}_{[0,c_1]}\ s) \leqslant p\ \{\text{or} \geqslant p\}$
Location of the ditch	$F_{[\theta_1,\theta_2]}\ \texttt{ditch}$
Height of peak 1[2]	$(\text{Max}\ s\ \text{U}\ \texttt{ditch}) \leqslant p\ \{\text{or} \geqslant p\}$
Location of peak 1	$F_{[\theta_1,\theta_2]}\ \texttt{peak}$
Height of peak 2[3]	$\texttt{ditch} \wedge ((\text{Max}_{[0,c_2]}\ s) \leqslant p)\ \{\text{or} \geqslant p\}$
Location of peak 2	$\texttt{ditch} \wedge F_{[\theta_1,\theta_2]}\ \texttt{peak}$

[1] $c_1 = 136$ time units. Constant related to the duration of a sample signal.

[2] We recall from [14] that $(\text{Max}\ s\ \text{U}\ \texttt{ditch})$ takes at time t the value $\max_{t' \in [t,t'']} s(t')$ where t'' is the first time from t where `ditch` is true.

[3] $c_2 = 60$ time units. Also applies for Formula (23).

Table 5: Features and Feature Range.

Feature	D_1 for $\texttt{day}_1$	D_5 for $\texttt{day}_5$
Depth of the ditch	(-6.12, -4.767)	(-6.51, -5.71)
Location of the ditch	(51.00, 58.99)	(51.00, 59.99)
Height of peak 1	(1.01, 5.42)	(0.77, 3.81)
Location of peak 1	(48.00, 56.99)	(0.00, 55.99)
Height of peak 2	(1.25, 3.296)	(1.43, 2.58)
Location of peak 2	(25.00, 30.99)	(23.00, 26.99)

mula. The 0/0 in the column for testing error is because the test set is empty. Note that, for a PSTL formula each parameter valuation in the solution set produces a classifier. Two such classifiers we found, $\Psi^{cl_1}_{(28.3,11.0,4.0)}$ and $\Psi^{cl_2}_{(27.5,1.0,-1.3)}$ have error values 2/861 and 17/861 respectively on the original testing set.

$$\Psi^{cl_1}_{(p_1,p_2,p_3)} := (\texttt{ditch} \wedge F_{[p_1,c_2-p_2]}\,\texttt{peak}) \vee (\ (\text{Max}\ \ s\ \mathcal{U}\ \texttt{ditch}) \geqslant p_3)$$
$$\Psi^{cl_2}_{(p_1,p_2,p_3)} := (\texttt{ditch} \wedge F_{[p_1,c_2-p_2]}\,\texttt{peak}) \vee (\texttt{ditch} \wedge (\text{Max}_{[0,c_2]}\ \ s) \leqslant -p_3) \tag{23}$$

Table 6: Results for learning (case study 2). See Formula (23) for Ψ^{cl_1}, Ψ^{cl_2}

Configuration	time (s)	Testing error		Training error	
	$\delta = 5.10^{-3}$	Ψ^{cl_1}	Ψ^{cl_2}	Ψ^{cl_1}	Ψ^{cl_2}
Confg. 1 (23, 861)	787	2/861	17/861	0/23	0/23
Confg. 2 (100, 761)	10	2/761	17/761	0/100	0/100
Confg. 3 (300, 561)	5	2/561	NA[1]	0/300	NA
Confg. 4 (861, 0)	153	NA	NA	NA	NA
Confg. 5 (861, 0)	12	0/0	NA	2/861	NA

[1] NA: Not Applicable. Parameter search is unsuccessful.

Ici, dans la première partie, nous essayons d'enrichir TRE avec des prédicats qui comparent le maximum ou le minimum d'un signal sur intervalle de temps à une valeur constante donnée. Nous découvrons et prouvons ensuite que pour la tâche de filtrage temporel, l'introduction de ces prédicats n'améliore pas le pouvoir d'expression. Plus précisément, la sémantique de ces prédicats peut être simulée par des expressions équivalentes formées en utilisant l'ancienne syntaxe. En outre, nous introduisons un nouveau prédicat qui calcule la différence extrême (c'est-à-dire la différence entre le maximum et le minimum) dans des intervalles de temps et la compare à une constante. Nous montrons comment le monitoring de ce prédicat peut être effectué efficacement en exploitant la monotonicité. Dans la deuxième partie, nous nous concentrons sur l'extension de STL et introduisons des opérateurs qui comptent les événements. Les événements sont considérés comme se produisant lorsqu'un signal booléen change de valeur. Nous fournissons des algorithmes de monitoring pour ces opérateurs. Enfin, nous montrons comment ces opérateurs peuvent être appliqués au comptage de divers types d'événements impliquant des impulsions dans les signaux d'électrocardiogramme (ECG).

6

EXTENDING STL AND TRE

In this chapter we first consider how to extend TRE with operators to compute minimum, maximum or difference of extrema. Then we discuss how to introduce operators that count events into STL. Finally, we show how the counting operators can be applied to count various kinds of events involving pulses in Electrocardiogram (ECG) signals.

6.1 EXTENDING TRE TO COMPUTE EXTREMA

After looking at the extension of STL done in [14], it is pertinent to ask whether we can extend TRE (state-based) with similar minimum and maximum operators. We can introduce atomic predicates of form

$$\phi := (\max x \geqslant c) \mid (\max x \leqslant c) \mid (\min x \geqslant c) \mid (\min x \leqslant c)$$

For real-valued signal w and constant c, the semantics (similar to TRE (state-based)) can be defined as follows:

$$\begin{aligned}
(w, t, t') \models (\max x \geqslant c) &\quad\leftrightarrow\quad t \leqslant t' \wedge (\max_{[t,t']} w_x) \geqslant c \\
(w, t, t') \models (\max x \leqslant c) &\quad\leftrightarrow\quad t \leqslant t' \wedge (\max_{[t,t']} w_x) \leqslant c \\
(w, t, t') \models (\min x \geqslant c) &\quad\leftrightarrow\quad t \leqslant t' \wedge (\min_{[t,t']} w_x) \geqslant c \\
(w, t, t') \models (\min x \leqslant c) &\quad\leftrightarrow\quad t \leqslant t' \wedge (\min_{[t,t']} w_x) \leqslant c
\end{aligned}$$

We now give a theorem containing equivalences regarding the minimum and maximum operators.

Theorem 6.1.1. *The following equivalences hold with respect to semantics of timed pattern matching.*

$$\begin{aligned}
&(\max x \leqslant c) \equiv (x \leqslant c) \\
&(\min x \geqslant c) \equiv (x \geqslant c) \\
&(\max x \geqslant c) \equiv \mathit{true} \cdot (x \geqslant c) \cdot \mathit{true} \\
&(\min x \leqslant c) \equiv \mathit{true} \cdot (x \leqslant c) \cdot \mathit{true}
\end{aligned}$$

Proof. The first two equivalences hold because of the following tautologies arising from the definition of maximum and minimum.

$$\begin{aligned}
(\max_{[t,t']} w_x) \leqslant c &\quad\leftrightarrow\quad \forall t''.(t \leqslant t'' \leqslant t' \rightarrow (w_x[t''] \leqslant c)) \\
(\min_{[t,t']} w_x) \geqslant c &\quad\leftrightarrow\quad \forall t''.(t \leqslant t'' \leqslant t' \rightarrow (w_x[t''] \geqslant c))
\end{aligned}$$

The third equivalence can be proved by showing an implication both ways. We know from the definition of maximum the following implication:

$$(\max_{[t,t']} w_x) \geqslant c \implies \exists t''.(t \leqslant t'' \leqslant t' \wedge (w_x[t''] \geqslant c))$$

From this it follows that,

$$(w, t'', t'') \models (x \geqslant c)$$

and since $t \leqslant t'' \leqslant t'$ it follows that,

$$(w, t, t') \models \texttt{true} \cdot (x \geqslant c) \cdot \texttt{true}$$

The implication for the other direction can be proved as follows. Let us assume that for some t,t′,

$$(w, t, t') \models \texttt{true} \cdot (x \geqslant c) \cdot \texttt{true}$$

This implies there exist t'' and t''' with $t \leqslant t'' \leqslant t''' \leqslant t'$ such that,

$$(w, t'', t''') \models (x \geqslant c)$$

Finally, from the definition of maximum it follows that,

$$(\max_{[t,t']} w_x) \geqslant c$$

Hence, the third equivalence is proved. The fourth equivalence also can be proved in a similar manner as the third. □

From Theorem 6.1.1, we realize that under match-set semantics the minimum and maximum operator can be expressed using existing elements of TRE (state-based). This result inspires us to search further for additional operators that can be efficiently handled. The simplest one is the difference operator which enriches the syntax as follows:

$$\phi := (\text{diff}\, x \leqslant c) \mid (\text{diff}\, x \geqslant c)$$

whose match set semantics is defined below:

$$(w, t, t') \models (\text{diff}\, x \geqslant c) \quad \leftrightarrow \quad t \leqslant t' \wedge (\max_{[t,t']} w_x) - (\min_{[t,t']} w_x) \geqslant c$$
$$(w, t, t') \models (\text{diff}\, x \leqslant c) \quad \leftrightarrow \quad t \leqslant t' \wedge (\max_{[t,t']} w_x) - (\min_{[t,t']} w_x) \leqslant c$$

Let us assume that we have four timing variables $t \leqslant t'' \leqslant t''' \leqslant t'$. We can exploit the monotonicity property arising from the inclusion $[t'', t'''] \subseteq [t, t']$ to deduce that for $(\text{diff}\, x \leqslant c)$:

$$(w, t, t') \models (\text{diff}\, x \leqslant c) \quad \rightarrow \quad (w, t'', t''') \models (\text{diff}\, x \leqslant c)$$

Because,

$$\begin{aligned}(\max_{[t,t']} w_x) &\geqslant (\max_{[t'',t''']} w_x)\\ (\min_{[t,t']} w_x) &\leqslant (\min_{[t'',t''']} w_x)\end{aligned}$$

Which implies,

$$(\max_{[t,t']} w_x) - (\min_{[t,t']} w_x) \leqslant c \quad \rightarrow \quad (\max_{[t'',t''']} w_x) - (\min_{[t'',t''']} w_x) \leqslant c$$

Similarly, we can deduce that for $(\text{diff}\, x \geqslant c)$:

$$(w, t'', t''') \models (\text{diff}\, x \geqslant c) \quad \rightarrow \quad (w, t, t') \models (\text{diff}\, x \geqslant c)$$

These monotonicity results can be exploited to efficiently represent and compute the following match sets

$$\mathcal{M}(\text{diff}\, x \leqslant c, w) \text{ and } \mathcal{M}(\text{diff}\, x \geqslant c, w)$$

assuming that w_x is piece-wise constant signal. The match set will contain a number of two-dimensional zones equal to the number of segments in w_x and the computation has linear time complexity. In Figure 36, we illustrate an example signal x at the top and the match set for $(\text{diff}\, x \leqslant 2)$ below it. The match set contains four zones shown in four different colours. This cannot be reduced without using approximation.

6.2 EXTENDING STL WITH COUNTING OPERATORS

In this section we discuss how STL can be extended with counting operators. This has been inspired by the min/max operators introduced in [14]. Introduction of counting modalities in Metric Temporal Logic (MTL) has already been studied in [48] from the perspective of expressive power and decidability properties. But we are concerned with monitoring and ours is a naturally quantitative semantics. For example, we can easily combine counted values with arithmetic operators like addition, ratio, and weighted sum. If required comparison or other types of operators with Boolean semantics can then be applied. The syntax of the newly added elements is as follows:

$$\begin{aligned}\varphi &:= On_{[a,b]}\psi \mid \psi \,U\, \varphi\\ \psi &:= c^+\varphi \mid c^-\varphi\end{aligned}$$

We now describe the semantics. In the above definitions c^+ and c^- stand for counting rising and falling edges respectively. A rising edge is defined to occur when a Boolean signal changes from false to true. Similarly, a falling edge is defined to occur when a Boolean function changes from true to false. The formula $On_{[a,b]}c^+\varphi$ takes

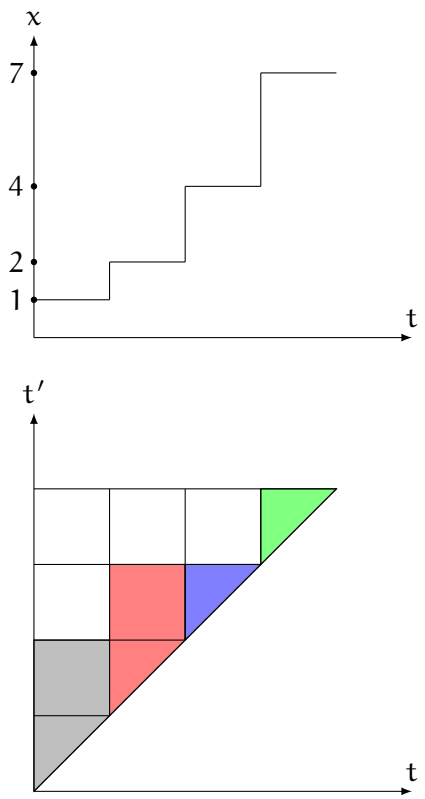


Figure 36: Illustration of Difference in Extrema: Showing Signal x and Match Set for $(\text{diff}\, x \leqslant 2)$.

at time t the value of the number of rising edges of φ in the interval $[t + a, t + b]$. Similarly, the semantics of $On_{[a,b]}c^{-}\varphi$ can also be understood. The formulae $On_{[a,b]}c^{+}\varphi$ and $On_{[a,b]}c^{-}\varphi$ can be shortened as $c^{+}_{[a,b]}\varphi$ and $c^{-}_{[a,b]}\varphi$ respectively. The formula $c^{+}\varphi_1\ \mathcal{U}\ \varphi_2$ at time t takes the value of the number of rising edges in φ_1 until the time φ_2 becomes true. If after time t, the evaluation of φ_2 never becomes true, then the default value of the formula at time t will be zero. The semantics of $c^{-}\varphi_1\ \mathcal{U}\ \varphi_2$ is similarly defined but counts falling edges.

6.2.1 *Monitoring Algorithms*

In this subsection we describe our algorithms for counting operators. The signals we are dealing with are assumed to be piece-wise constant. We first describe how counting edges over intervals is performed. Later we present the algorithm for counting operator combined with the until operator.

We describe the procedure for counting edges over intervals using the illustration given in Figure 37. Consider the satisfaction signal for the formula φ at the bottom of the figure. It is a Boolean signal with a length/duration of 8 time units. We describe the procedure

to compute the signal for $c^{+}_{[0,4]}\varphi$. We first evaluate the formula at time $t = 0$. We can see that in the interval $[0, 4]$ there are two rising edges. Therefore, the value of the formula at $t = 0$ is 2. The idea is to slide the interval $[t + 0, t + 4]$ by increasing the value of t till it becomes equal to the length of the signal for φ which is 8 time units. Note that the valuation of $c^{+}_{[0,4]}\varphi$ will only change finite number of times in the time interval $[0, 8]$. This holds assuming that the signal we are operating on is piece-wise constant and changes only finite number of time. We can see that shifting the interval $[0, 4]$ by 0.2 time units to the right makes the lower bound of the interval hits the first rising edge of the signal for φ and reduces the count by one. Right shifting by any amount less than this will not lead to any change in the evaluation of the count. We repeatedly compute the least shift that changes the count and keep updating the count. In Figure 37 we can see that the count over rising edges reduces to 1 when we shift by 0.2 time units. And shifting by 0.4 more (i.e. 0.6) we reduce the count to zero. When the absolute shift is 2 units the upper bound of the interval hits a rising edge at time $t = 6$ which increase and makes the count equal to 1. The count remains constant with shifting till the absolute shift becomes 6 and the lower bound of the interval crosses the rising edge at $t = 6$. At this point of crossing the count becomes 0 and remains the same till the end of the signal. We can compute the signal for $c^{-}_{[0,4]}\varphi$ in a similar fashion as shown in Figure 37. We can see how shifting affects the count with the help of the vertical dashed lines in Figure 37.

We now explain the algorithm for counting combined with until operator. The main function is `untilCount` (Algorithm 9). This is a modified version of the until function from [14]. Let us consider our algorithm for $(c^{+}\varphi_1 \, U \, \varphi_2)$ or $(c^{-}\varphi_1 \, U \, \varphi_2)$. The piece-wise constant input signals are denoted by u_1 and u_2 representing the output signals of φ_1 and φ_2 respectively. The signal u_1 is represented using a list of intervals $J^1_0, \ldots, J^1_{m-1}$. Note that the end of one interval in the list will be the beginning of next interval in the list. And also, the intervals are disjoint and their union will be the interval $[0, d]$ where d is the duration of the signal. For each of these intervals we have the corresponding value denoted by $v^1_0, \ldots, v^1_{m-1}$. The signal u_2 is also represented in a similar way. The input f to `untilCount` is a function for updating the value of the count used in the sub-function `untilAdd` (Algorithm 10). See below for the definition of this function for the two types of counting edges.

function: $f(v', v_1, w)$	**Purpose**
$c^{+}(v', v_1, w) = v' + (w > v_1)?1 : 0$	For counting rising edges
$c^{-}(v', v_1, w) = v' + (w < v_1)?1 : 0$	For counting falling edges

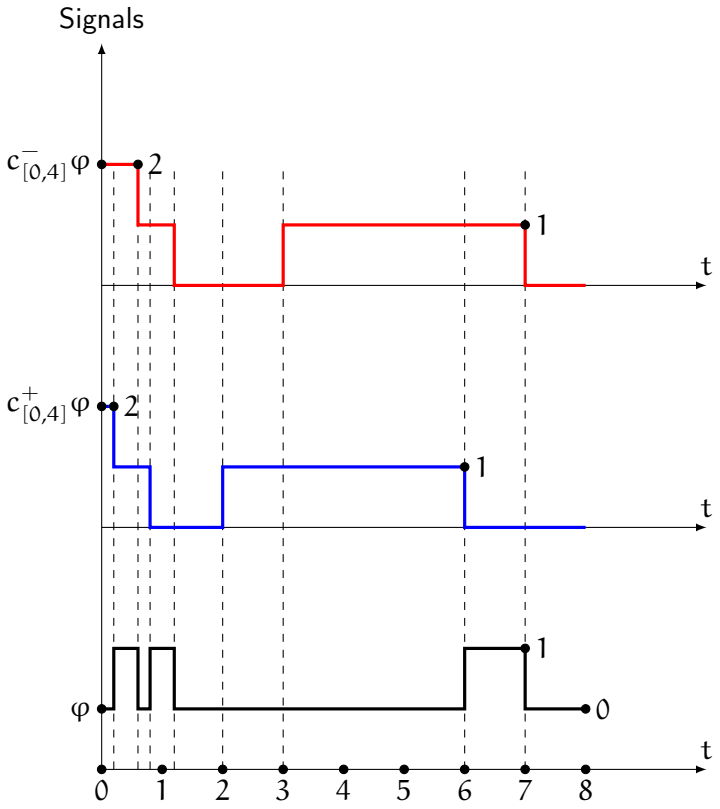


Figure 37: Illustration of counting over a window

The main idea behind untilCount is a backward scan of the two input signals. We keep track of the intervals (J) over which both the input signals remain constant. Each such interval is sent to the sub-function untilAdd. Inside this sub-function we use the function f to update the running count (v'). The function f updates the running count by comparing v_1 and w. The variables v_1 and w stand for the current value under consideration and the old value respectively. After this, the function prepend in untilAdd is a simple list operation which adds an element at the beginning of the list. Using prepend the sub-function untilAdd constructs the output signal u_r.

We show how the results of the operations $(c^+\varphi_1 \ U \ \varphi_2)$ and $(c^-\varphi_1 \ U \ \varphi_2)$ look like for a given pair of input signals for φ_1 and φ_2 in Figure 38. We can see in Figure 38 that the value of $(c^+\varphi_1 \ U \ \varphi_2)$ is 2 at time $t = 0$. This is because we have two rising edges in the signal for φ_1 until φ_2 becomes true. The value decreases by one unit twice at $t = 0.5$ and $t = 1.5$ and becomes 0 as we cross the two rising edges. Similarly, it takes the value 1 at $t = 5$ and decreases to 0 at $t = 6$ after crossing a rising edge. The signal resulting from operation $(c^-\varphi_1 \ U \ \varphi_2)$ is also shown at the top in Figure 38.

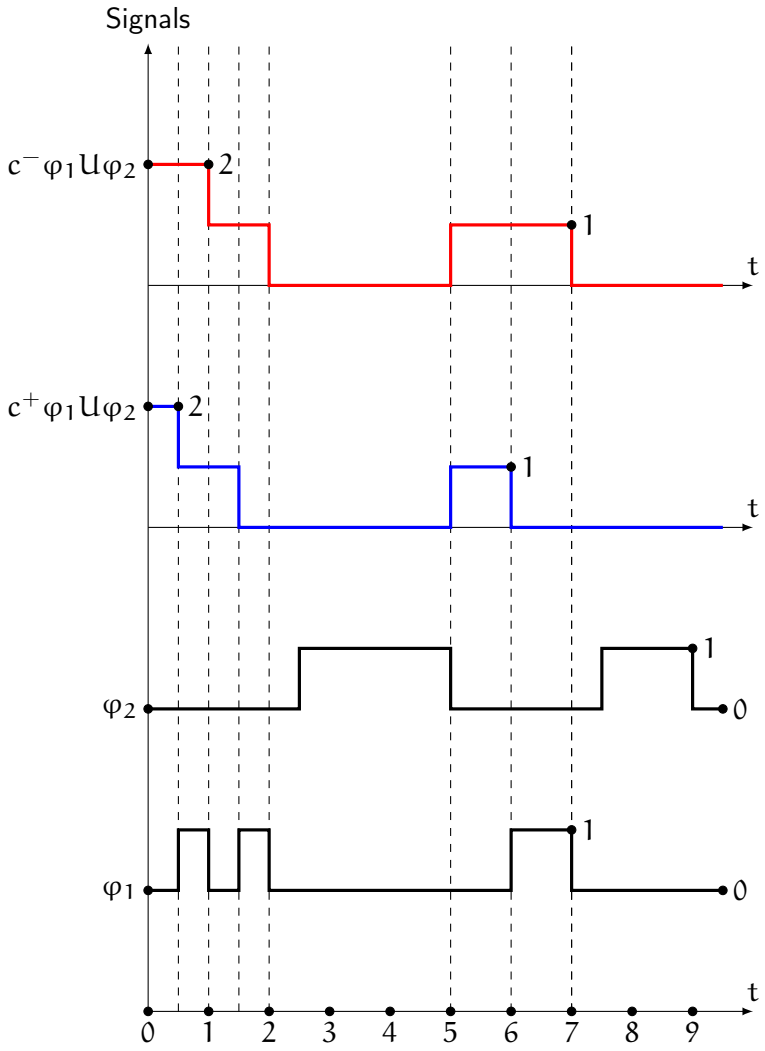


Figure 38: Illustration of counting until

6.2.2 *Real-World Example over Electrocardiogram signal*

In addition to the synthetic examples given in the previous sub-section we give here a real-world example. We show how counting operation can be used over Electrocardiogram (ECG) number 221 from [38]. In Figure 39, we show the first six pulses of ECG 221 signal in black. The normal ECG pulses are labelled using the Boolean signal in blue. Similarly, the abnormal pulse with premature ventricular contraction is labelled using the Boolean signal in red. The Boolean signals used for labelling are true for a duration of unit time at the peak of the pulse and false elsewhere. We can see in Figure 39 that the first two pulses and the fourth and fifth pulses are normal. And the third and sixth pulses are abnormal. We use N to denote the labelling signal for normal pulses and V to denote the one for abnormal pulses. We can use the formulae $(c^{+}_{[0,\infty)}N)$ and $(c^{+}_{[0,\infty)}V)$ to count the number of normal and abnormal pulses respectively. The two formulae evaluated at

Algorithm 9 $\mathtt{untilCount}(u_1, u_2, f, d, w)$

let $u_1 = \langle J_0^1 \mapsto v_0^1, \ldots, J_{m-1}^1 \mapsto v_{m-1}^1 \rangle$
let $u_2 = \langle J_0^2 \mapsto v_0^2, \ldots, J_{k-1}^2 \mapsto v_{k-1}^2 \rangle$
$i \leftarrow m-1, j \leftarrow k-1$
$(u_r, s, v') \leftarrow (\langle\rangle, 0, d)$
while $i \geqslant 0 \wedge j \geqslant 0$ **do**
    $J \leftarrow J_i^1 \cap J_j^2$
    $(u_r, s, v') \leftarrow \mathtt{untilAdd}(f, u_r, s, v', J, v_i^1, v_j^2, w)$
    $w \leftarrow v_i^1$
    if $\exists t_1 \in J_i^1.\ \forall t_2 \in J_j^2.\ t_1 > t_2$ **then**
        $j \leftarrow j-1$
    else if $\exists t_2 \in J_j^2.\ \forall t_1 \in J_i^1.\ t_2 > t_1$ **then**
        $i \leftarrow i-1$
    else
        $i \leftarrow i-1, j \leftarrow j-1$
return u_r

Algorithm 10 $\mathtt{untilAdd}(f, u_r, s, v', J, v_1, v_2, w)$

if $v_2 \neq 0$ **then**
    $v' \leftarrow 0$
    $s \leftarrow 1$
else if $s \neq 0$ **then**
    $v' \leftarrow f(v', v_1, w)$
$\mathtt{prepend}(u_r, J \mapsto v')$
return (u_r, s, v')

the start of the signal will return the counts 4 and 2 respectively. Let us now consider the formula $(c^+N)\ U\ V$ which can be used to count the number of normal pulses we have until we see an abnormal one. This signal corresponding to this formula is shown in Figure 40 in brown superimposed over the ECG signal. The signal forms two stair cases descending from two to one. This example shows how we can count how many interesting events occur between another signal that flags another occurrence.

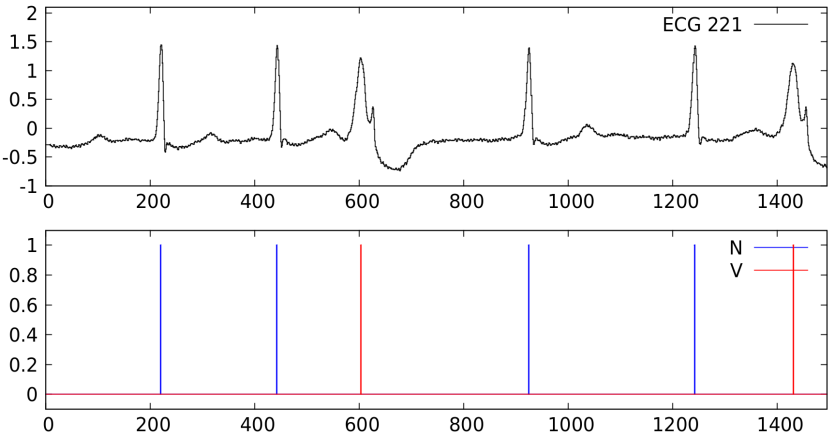


Figure 39: Labelling of the first few pulses of ECG 221

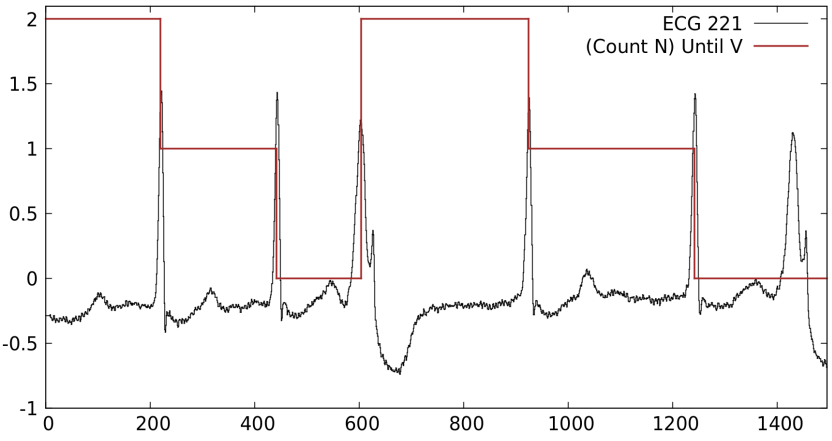


Figure 40: Counting number of normal pulses until abnormal pulses $((c^{+}N)\ \mathcal{U}\ V)$ superimposed over the trace of ECG 221

GUIDE D'UTILISATION DES OUTILS

Nous expliquons ici l'utilisation et les caractéristiques pertinentes de nos outils. Les outils prennent en entrée des signaux à valeur réelle ou des mots temporisés. Nous expliquons comment ces entrées sont fournies à l'aide d'un format de valeurs séparées par des virgules (Comma-Separated Values, CSV). L'outil parameTRE exécute le PTPM selon trois modes différents en fonction de l'application. Le premier mode concerne les signaux à valeur réelle. Le second traite des mots temporisés et le troisième des signaux booléens. Nous expliquons l'ensemble de la syntaxe des expressions prises en charge par l'outil. Ensuite, nous montrons comment l'identification paramétrique peut être effectuée à l'aide de l'outil. Nous montrons également comment les opérations sur les zones paramétriques peuvent être effectuées à l'aide d'opérations sur les polytopes. Ensuite, nous expliquons l'utilisation de l'outil StlEval et décrivons la syntaxe utilisée pour exprimer les opérateurs de comptage que nous avons introduits. Enfin, nous décrivons l'utilisation des fonctionnalités que nous avons ajoutées à la bibliothèque ParetoLib. Nous montrons d'abord l'application de notre algorithme d'intersection de Pareto au cas tridimensionnel. Nous montrons ensuite comment l'utilisateur peut définir ses propres Oracles personnalisés en héritant de la classe générique Oracle. Nous terminons en expliquant comment notre algorithme d'intersection de Pareto peut être importé et appliqué au cas général à n dimensions en important la fonction correspondante à l'aide de la bibliothèque.

7 USAGE GUIDE FOR TOOLS

Before explaining the usage of the tools we first explain the input format and interpretation of real-valued signals and timed words. Next, we explain the usage of `parameTRE`[1], StlEval[2] and ParetoLib [15] in order.

7.1 INPUTTING REAL-VALUED SIGNALS AND TIMED WORDS

For real-valued signals the format is Comma-Separated Values (CSV). In the file, the first column specifies the time stamp. The other columns state the value of different signals at those time stamps. These signals are named x0, x1 , x2, x3 and so on from left to right in a sequence. In the Listing 1, we show an example input with three real-valued signals. The first signal (shown in second column) is accessed by identifier x0 and it takes values $(1, 2, 4, 7, 7)$ in order. The other two signals (shown in third and fourth columns) are accessed by identifiers x1 and x2 respectively. Note that the tool only supports interpreting real-valued signals as piece-wise constant interpolations. See Figure 41 for a pictorial illustration of the signal in Listing 1.

Listing 1: Example Input: Real-valued signals

```
0,1,9,0.5
1,2,0.5,0.5
2,4,0.5,8
3,7,6,3
4,7,6,3
```

For timed words also the format is CSV with the file having exactly two columns. The first column lists the time stamp and the second column lists the name of the event that occurs at this time stamp. The event names can only take values of the form y0, y1, y2, y3 and so on. This is to simplify the implementation. In the Listing 2 we show an example input with three different type of events. In Figure 42 we show an illustration for the timed word.

Listing 2: Example Input: Timed words

```
1,y0
5,y1
7,y0
9,y2
```

1 https://gricad-gitlab.univ-grenoble-alpes.fr/mambakaa/hscc_ptre
2 https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/StlEval

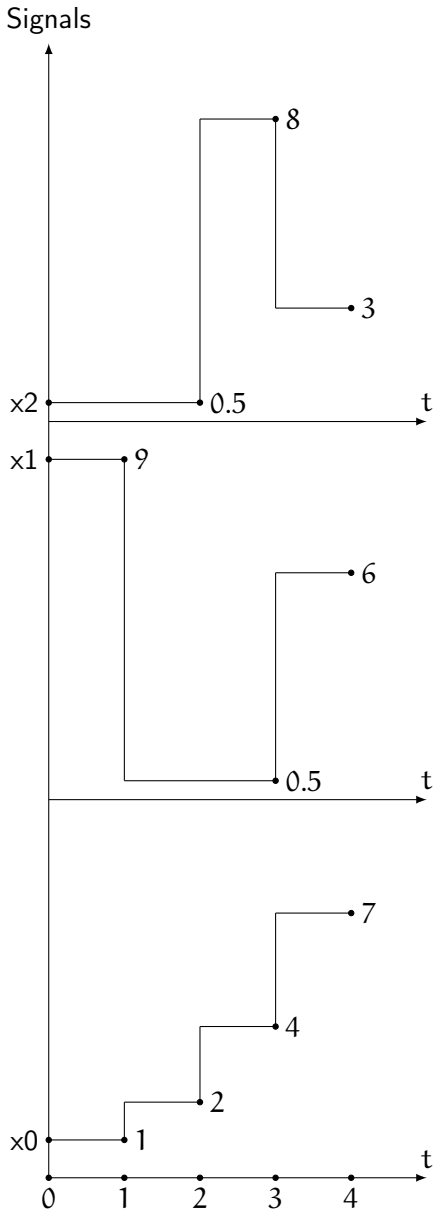


Figure 41: Illustration for inputting real-valued signals

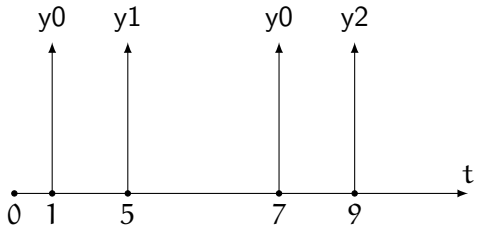


Figure 42: Illustration for inputting timed words

7.2 THE PARAMETRE TOOL

The `parameTRE` tool is an amalgamated implementation of parsing and algorithmic kernels for all three PSRE, PE-TRE and PTRE (event-

based) combined in a monolithic structure. The tool is only a prototype and in our opinion could be split into different tools in order to have a more robust, concise and efficient implementation.

7.2.1 *Tool Description*

The tool incorporates offline algorithms for Parametric Timed Pattern Matching (PTPM) and can also perform parametric identification based on PTPM. It is to be noted that PTPM can be done for real-valued signals, Boolean signals and time-event sequences (expressed as timed words). But parametric identification has only been implemented for PSRE.

For PTPM, the tool takes as input the file containing the expression, the file containing the data (signals or timed words). It also takes the number of parameters and the mode of operation. There are three modes of operation. The first mode deals with PSRE and PE-TRE and because they are interpreted over real-valued signals the input is a single CSV file containing the values of the signals for each time segment. The second mode deals with PTRE (event-based) and because they are interpreted over timed-words the input is a single CSV file with two columns. The first column contains the time-stamp and the second specifies which event occurs there. The third mode deals with PSRE and the input is a list of files each containing a Boolean signal. This mode is not strictly needed but has been introduced for ease of use when dealing with signals pre-processed into Boolean signals. We show below example commands for the first, second and third mode in order.

```
./ptre insre.ptre indata.csv 3 0
./ptre intre.ptre indata.csv 4 1
./ptre inboolsre.ptre indata.txt 5 2
```

We see that the first argument is the input expression file name. The second is the input data file name. The third is the number of parameters and the fourth is the mode. Let us now go into the details. We will first describe the syntax of the input expression. Then, we explain the different kinds of formats of the input data.

We split the syntax into three tables (Tables 7,8,9). Let us first consider Table 7. There are three different identifiers for signals/events. The first one (x) names real-valued signals and pertains to the first mode. The second one (y) names events (passage of time followed by an event) and is for the second mode. The third (z) names Boolean signals and is used in the third mode. The identifier type p is for parameters. Finally, the identifiers "int" and "eps" are for integers and empty strings respectively. The symbol E used in Tables 8,9 stands for a linear expression whose syntax is as given below:

$$E := E + E \mid E - E \mid \mathtt{int} * E \mid \mathtt{int}$$

In Table 8, we show the syntax for all regular expression operators along with the duration restriction operators. In Table 9, we have the syntax of three unary operators over Boolean signals which are rising edge (recall $\uparrow$), falling edge (recall $\downarrow$) and negation. Note that $\langle x \rangle_b$ is syntactically the same as the identifier x for real-valued signals but is only valid for Boolean signals. Similarly, $\langle E \rangle_c$ is syntactically the same as the identifier E but is only valid for constant expressions. The last five rows are predicates defined over real-valued signals. In these we have simple threshold operators and also thresholds applied on top of maximum (MAX), minimum (MIN) and extreme difference (DIFF) operators. The "int" in these five rows is a syntactic hack to allow linear expressions with rational coefficients. Here are some examples on how to use this hack.

Use	$10\ x \leqslant 2p_0 + 5p_1$	**for**	$x \leqslant 0.2p_0 + 0.5p_1$
Use	$2 \leqslant 4\ x \leqslant 1$	**for**	$0.5 \leqslant x \leqslant 0.25$
Use	10 MAX $x \geqslant 1$	**for**	MAX $x \geqslant 0.1$
Use	100 MIN $x \leqslant 21$	**for**	MIN $x \leqslant 0.21$
Use	100 DIFF $x \leqslant 25$	**for**	DIFF $x \leqslant 0.25$

Identifier type	Syntax
x	$x[0-9]^+$
y	$y[0-9]^+$
z	$z[0-9]^+$
p	$p[0-9]^+$
int	$[0-9]^+$
eps	EPS

Table 7: `parameTRE` syntax: Identifiers

Operator type	Syntax
Concatenation	PTRE % PTRE
Conjunction	PTRE & PTRE
Disjunction	PTRE | PTRE
Kleene Plus	PTRE ^+
Duration Restriction	PTRE [E E]

Table 8: `parameTRE` syntax: Regular Expression and Temporal Operators

We now explain the three different input formats corresponding to the three modes. The input format for the first mode is CSV pertaining to real-valued signals as discussed in Section 7.1. For the second mode the input is timed words represented in the CSV format as previously described in Section 7.1. For the third and final mode we

Unary operators and predicates	Syntax
Rising edge	+@ $\langle x \rangle_b$
Falling edge	-@ $\langle x \rangle_b$
Negation	! $\langle x \rangle_b$
Threshold predicates	int x $(\leqslant / \geqslant)$ E
Combined threshold predicate	$\langle E \rangle_c \leqslant$ int x $\leqslant \langle E \rangle_c$
Threshold predicates with max operator	int MAX x $(\leqslant / \geqslant) \langle E \rangle_c$
Threshold predicates with min operator	int MIN x $(\leqslant / \geqslant) \langle E \rangle_c$
Threshold predicates with diff operator	int DIFF x $(\leqslant / \geqslant) \langle E \rangle_c$

Table 9: parameTRE syntax: Unary operators and predicates

have text format for input file. In each line we specify the name of a CSV file. In Listing 3 we show the input file containing the names of three Boolean signals sig1, sig2 and sig3. In Listings 4,5,6 we show the corresponding CSV files respectively. Figure 43 shows an illustration of these Boolean signals accessed using identifiers z0, z1 and z2 respectively. For all the three modes of PTPM, the output is the set of parametric zones in the parametric match set.

Listing 3: parameTRE input: List of Boolean signals (mode three)

```
sig1.csv
sig2.csv
sig3.csv
```

Listing 4: parameTRE input: Boolean signal sig1.csv (z0)

```
0,0
1,1
3,0
4,0
```

Listing 5: parameTRE input: Boolean signal sig2.csv (z1)

```
0,0
2,1
4,1
```

Listing 6: parameTRE input: Boolean signal sig3.csv (z2)

```
0,1
1,0
4,0
```

Now we describe how the tool handles parametric identification. As stated earlier this feature of the tool currently only works for PSRE. Consider the following command:

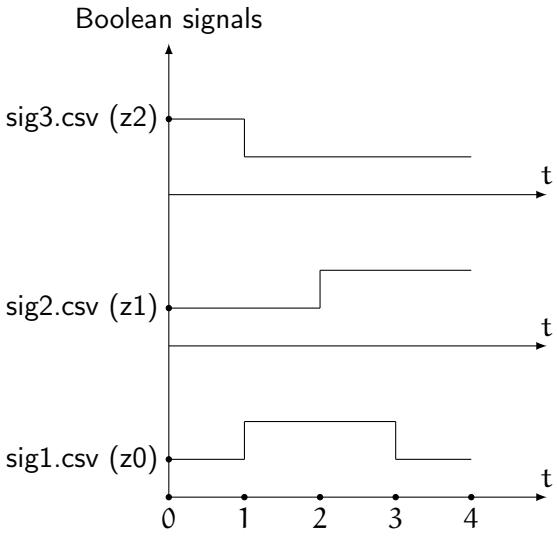


Figure 43: parameTRE: Illustration of Boolean signals (mode three)

```
./ptre paramecg.ptre ecg.csv 4 0 ecg.label
```

The first four command line arguments are the same as for the mode one of PTPM. The first argument is the name of the file containing the PSRE specification. The second is a CSV file specifying the real-valued signals of our data. The third argument specifies the number of parameters and the fourth specifies the mode which is always zero as we only deal with real-valued signals. The final argument is a two-columned CSV file specifying the start and the end of the time intervals where pattern is labelled to occur. See the following (Listing 7) which shows the content of an example file for labelling. It lists ten disjoint intervals where the pattern is stated to occur. The output of the tool for this feature will be list of polytopes specifying the parameter values over the parameter space for which the expression has a match for each of the labelling intervals.

Listing 7: parameTRE: Example labelling (ecg.label) for parametric identification

```
189,269
426,506
666,746
904,984
1148,1228
1390,1470
1631,1711
1862,1942
2105,2185
2353,2433
```

7.2.2 *Parametric Zone Operations*

The two logical/timed operations on parametric zones are intersection and sequential composition. We show examples for these operations along with illustrations. We first illustrate an example of intersection operation.

$$\text{Consider the parametric zone } \mathbf{z}_1 = \begin{pmatrix} t \leqslant 3 \\ t' \leqslant 6 \\ t' - t \leqslant 2\theta \\ 0 \leqslant t' - t \\ 4 \leqslant t' \\ 1 \leqslant t \end{pmatrix} \wedge (\theta \leqslant 10),$$

$$\text{parametric zone } \mathbf{z}_2 = \begin{pmatrix} t \leqslant 4 \\ t' \leqslant 7 \\ t' - t \leqslant 5 \\ \theta \leqslant t' - t \\ 5 \leqslant t' \\ 2 \leqslant t \end{pmatrix} \wedge (0 \leqslant \theta),$$

$$\text{and parametric zone } \mathbf{z}_3 = \mathbf{z}_1 \cap \mathbf{z}_2 = \begin{pmatrix} t \leqslant 3 \\ t' \leqslant 6 \\ t' - t \leqslant 2\theta \\ \theta \leqslant t' - t \\ 5 \leqslant t' \\ 2 \leqslant t \end{pmatrix}.$$

The parametric zone $\mathbf{z}_3$ is the intersection of $\mathbf{z}_1$ and $\mathbf{z}_2$. Parametric zones $\mathbf{z}_1$, $\mathbf{z}_2$ and $\mathbf{z}_3$ are show separately in Figures 44, 45 and 46 respectively. In Figure 47 we show all of them together in a single illustration.

Next, we give an example of sequential composition operation with illustrations.

$$\text{Consider parametric zone } \mathbf{z}_4 = \begin{pmatrix} t \leqslant 2 \\ t' \leqslant 6 \\ t' - t \leqslant 2\theta \\ 0 \leqslant t' - t \\ 3 \leqslant t' \\ 0 \leqslant t \end{pmatrix} \wedge (0 \leqslant \theta \leqslant 10)$$

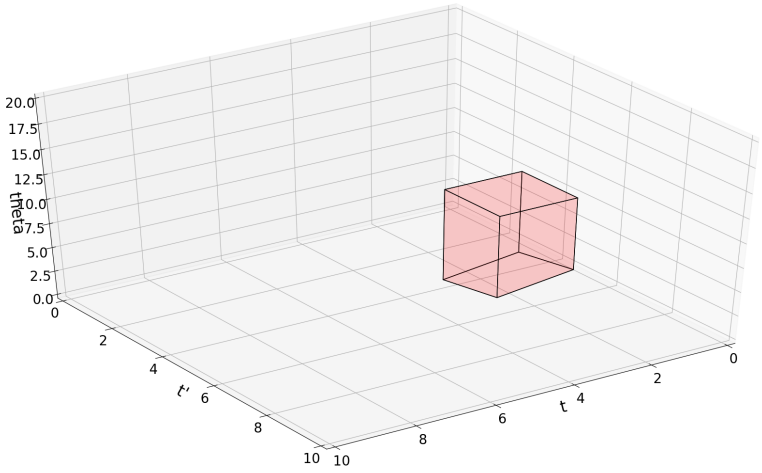


Figure 44: Illustration of parametric zone $\mathbf{z}_1$.

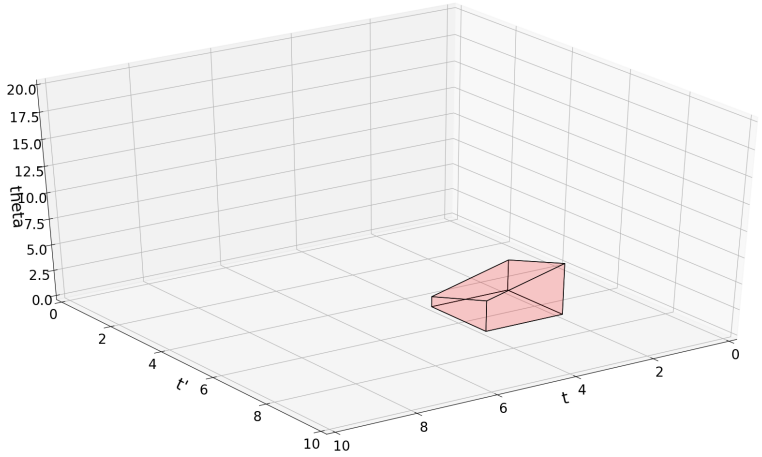


Figure 45: Illustration of parametric zone $\mathbf{z}_2$.

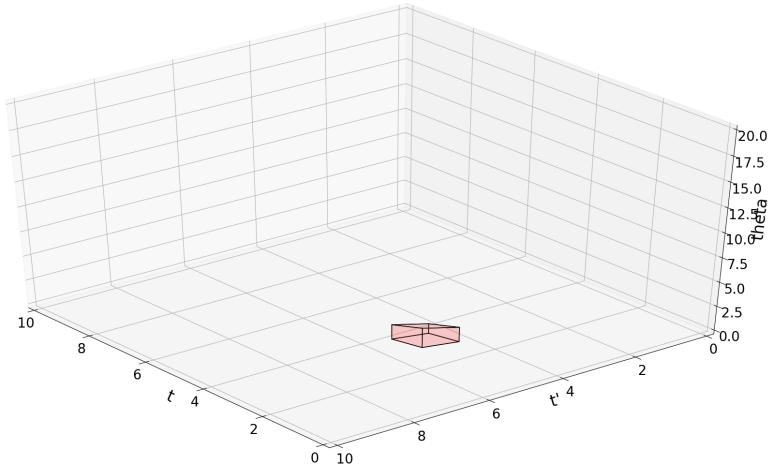


Figure 46: Illustration of the parametric zone $\mathbf{z}_3$ which is intersection of $\mathbf{z}_1$ and $\mathbf{z}_2$.

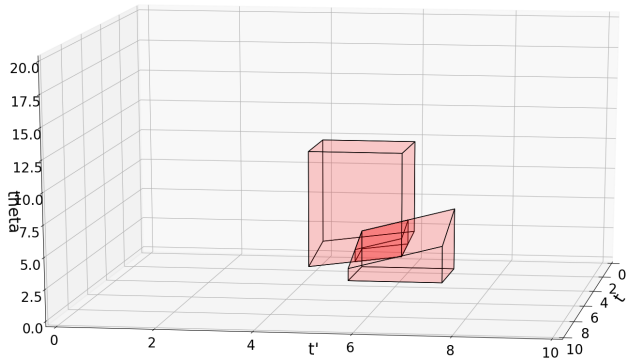


Figure 47: Illustration of the parametric zones $\mathbf{z}_1$,$\mathbf{z}_2$ and $\mathbf{z}_3$ together.

and parametric zone $\mathbf{z}_5 = \begin{pmatrix} t \leqslant 6 \\ t' \leqslant 9 \\ t' - t \leqslant 3\theta \\ \theta \leqslant t' - t \\ 8 \leqslant t' \\ 5 \leqslant t \end{pmatrix} \wedge (0 \leqslant \theta \leqslant 10)$.

Sequential composition of $\mathbf{z}_4$, $\mathbf{z}_5$ is $\mathbf{z}_6 = \mathbf{z}_4 \circ \mathbf{z}_5 = \begin{pmatrix} t \leqslant 2 \\ t' \leqslant 9 \\ 8 \leqslant t' \\ 5 + \theta \leqslant t' \\ 0 \leqslant t \\ 5 - 2\theta \leqslant t \end{pmatrix}$

We can also write $\mathbf{z}_6$ as $\begin{pmatrix} t \leqslant 2 \\ t' \leqslant 9 \\ \max(8, 5 + \theta) \leqslant t' \\ \max(0, 5 - 2\theta) \leqslant t \end{pmatrix}$

The equation for $\mathbf{z}_6$ does not show constraints over the duration of the interval $t' - t$ because they are redundant. Parametric zones $\mathbf{z}_4$, $\mathbf{z}_5$ and $\mathbf{z}_6$ are shown in Figures 48, 49 and 50.

Sequential composition can be implemented using swapping dimensions, intersection and projection operations over polytopes. We assume that start and end of time intervals is denoted by t and t' respectively. The variable t'' is used to stand for the time dimension that is eliminated. For polytope P, swapping dimension d_1 with dimension d_2 is denoted as $P[d_1 \leftrightarrow d_2]$. Projection of P removing dimension d_1 is denoted as $\sigma_{d_1}(P)$. See below for a redefinition of sequential composition in terms of operations over polytopes:

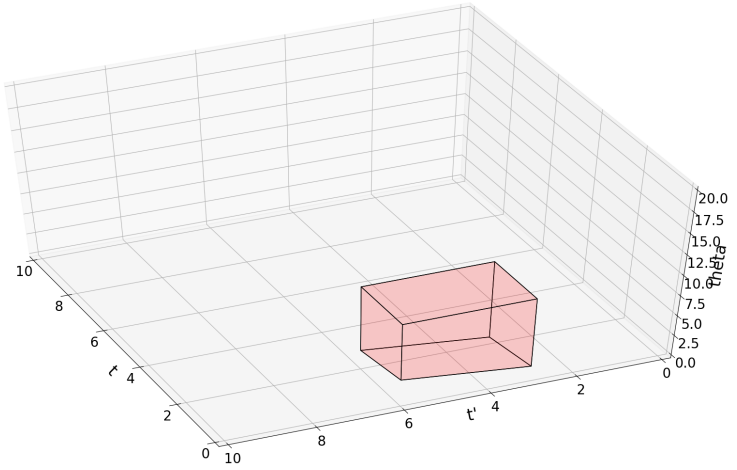


Figure 48: Illustration of parametric zone $\mathbf{z}_4$.

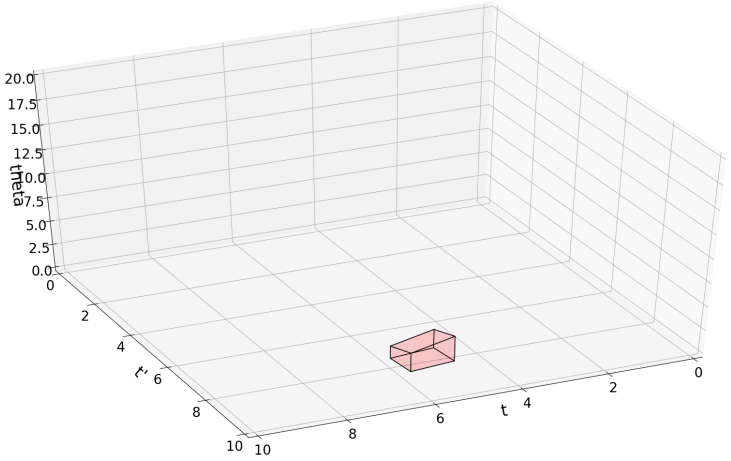


Figure 49: Illustration of parametric zone $\mathbf{z}_5$.

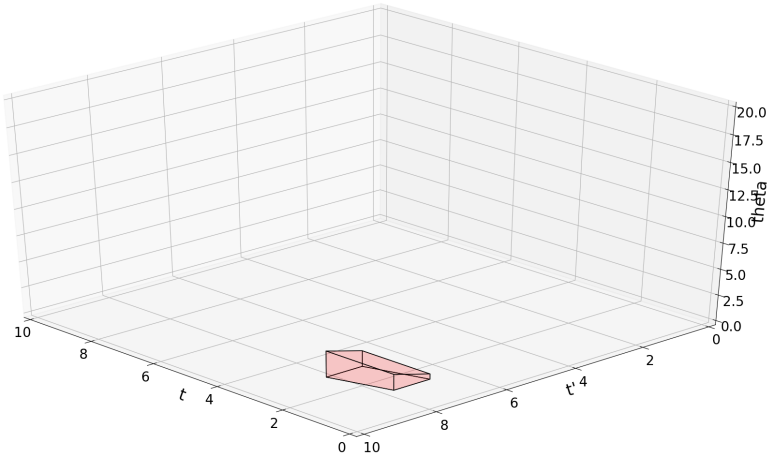


Figure 50: Illustration of parametric zone $\mathbf{z}_6$ which is the sequential composition of $\mathbf{z}_4$ and $\mathbf{z}_5$.

$$P_1 \circ P_2 = \sigma_{t''}(P_1[t' \leftrightarrow t''] \cap P_2[t \leftrightarrow t''])$$

7.3 STLEVAL TOOL

In this section we demonstrate the usage of the tool StlEval. We do not explain all the features of the tool but only those which are relevant here. We assume the input signal is given in a CSV format. It is interpreted using a piece-wise constant interpolation as discussed in Section 7.1. We show below the relevant parts of the syntax of formulae supported by the tool.

Type	Syntax
identifier for signals	$x[0-9]^+$
num	float | inf | -inf
Interval	(num num)
Arithmetic	φ $(+\mid -\mid *)$ φ
Comparison	φ $(> \mid >= \mid < \mid <=)$ φ
Logical operators	φ and φ
	φ and φ
	not φ
STL operators	(F Interval φ)
	(G Interval φ)
	(StlUntil Interval φ φ)
Aggregate	(Min φ)
	(Max φ)
	(Count φ)
Operators beyond STL	(On Interval Aggregate)
	(Until Interval num Aggregate φ)

We now give the command for running the tool and explain it.

```
./stle stlin.csv -ff testCountUntil.stl  -os 1 -osf c
```

In the above command the argument stlin.csv is the input in the CSV format. The option `-ff testCountUntil.stl` indicates the file containing the formula to be monitored. And, the option `-os 1` requires that the tool output the whole satisfaction signal rather than the evaluation at time $t = 0$. Finally, the option `-osf c` tells the tool to generate the output in a CSV format. Below is the tool's syntax for two examples of usage of new counting operators not in STL.

```
(On (0 inf) (Count x0))
(Until (0 inf) 0 (Count x2) x3)
```

These are equivalent to $c^+_{[0,\infty)}x_0$ and $(c^+x_2)\ \mathcal{U} x_3$ respectively.

7.4 PARETO INTERSECTION IN PARETOLIB

Here, we explain how to use our Algorithm 8 by importing ParetoLib library in Python. The following script corresponds to the toy example in Section 4.3.

```
from ParetoLib.Oracle.Oracle import Oracle
from ParetoLib.Search.Search import SearchIntersection3D
...
...
if __name__ == "__main__":
    (min_x, min_y, min_z) = (0.0, 0.0, 0.0)
    (max_x, max_y, max_z) = (1.0, 1.0, 1.0)
    orac1 = OracleStep(np=10)
    orac2 = OracleLine(li=1.1)
    rs = SearchIntersection3D(ora1=orac1,
        ora2=orac2,
        min_cornerx=min_x,
        min_cornery=min_y,
        min_cornerz=min_z,
        max_cornerx=max_x,
        max_cornery=max_y,
        max_cornerz=max_z,
        epsilon=0.0001,   # EPS
        delta=0.01,       # DELTA
        max_step=10000,   # STEPS
        blocking=False,
        sleep=0,
        opt_level=2,
        parallel=False,
        logging=False,
        simplify=False)
```

In the above code we run the Algorithm 8 for the case of three dimensions by importing the function `SearchIntersection3D`. We need to pass the oracles (`ora1` and `ora2`) as arguments. We specify the lower bounds on the parameter values by assigning values to `min_cornerx`, `min_cornery`, and, `min_cornerz`. We assign the upper bounds similarly. After this we assign the values of epsilon and delta. We control the two modes of the algorithm using the argument `opt_level`. Assigning the value two to this argument will make the algorithm compute the whole intersection region till the value of the unexplored region goes below delta. If we assign it a value of zero the function will return immediately after it finds a point in the intersection. We now show how the user can create their own oracle. Two oracles named `OracleStep` and `OracleLine` are defined below.

```
from ParetoLib.Oracle.Oracle import Oracle
class OracleStep(Oracle):
  def __init__(self, np=7):
    Oracle.__init__(self)
    self.np = np
```

```
  def member(self, np):
    disc = math.floor(p[0] * self.np)
    disc = disc/self.np
    return disc + p[1] > 1.0
  def dim(self):
    return 3
class OracleLine(Oracle):
  def __init__(self, li=0):
    Oracle.__init__(self)
    self.li = li
  def member(self, li):
    return p[0] + p[1] < self.li
  def dim(self):
    return 3
```

Both these oracles inherit from the generic Oracle class and override the method `member`. They also override the method `dim` to indicate the number of dimensions (i.e. the number of parameters) the oracle deals with. The `member` methods of `OracleStep` and `OracleLine` correspond to the functions ρ_+ and ρ_- respectively from Section 4.3.

For the general case which applies to any number of parameters, the function `SearchIntersectionND_2` is to be used. For this function, the bounds on the parameter values are given as list of intervals (`list_intervals`). The argument `list_constraints` was provided in order to define linear constraints over parameters. But it should be assigned an empty set since it is an incompletely implemented feature whose correctness is not guaranteed. Example code showing the usage of the `SearchIntersectionND_2` function is shown below.

```
from ParetoLib.Search.Search import SearchIntersectionND_2
...
...
list_intervals = [(0, 30), (0, 20), (-10, 10), (-10, 10)]
list_constraints = []
rs = SearchIntersectionND_2(orac1,
    orac2,
    list_intervals,
    list_contraints,
    epsilon=EPS,
    delta=delta,
    max_step=STEPS,
    blocking=False,
    sleep=0,
    opt_level=0,
    parallel=False,
    logging=False,
    simplify=False)
```

CONCLUSIONS

Dans cette thèse, nous avons étendu les formalismes temporels (STL et TRE) avec de nouveaux opérateurs et développé des méthodes pour leur analyse paramétrique. Nous avons intégré les structures de données et les algorithmes que nous avons développés dans des outils existants tels que StlEval [3] et Paretolib [4] ainsi que dans un nouvel outil appelé parameTRE [5]. Nous nous sommes inspirés des opérateurs min/max introduits dans STL et de la théorie du Timed Pattern Matching développée pour TRE. Avant de conclure en discutant les principaux aspects de la thèse, nous rappelons quelques contributions mineures. Nous avons ajouté à STL des opérateurs pour les événements et nous avons montré leur utilisation sur des exemples appropriés. Nous avons également discuté de la manière dont les prédicats atomiques impliquant des extrema sur des intervalles peuvent être introduits dans TRE et de la manière de les analyser. Dans le cadre de travaux futurs, de nouveaux opérateurs d'intégration et de différenciation peuvent être ajoutés à STL et TRE. Nous résumons maintenant les concepts clés et les contributions de cette thèse, ainsi que les discussions sur les orientations possibles des travaux futurs.

Spécifications monotones et domination de Pareto. Nous avons introduit un cadre dans lequel une spécification est considérée comme la prédiction d'un modèle dans un signal. La prédiction, sous la forme d'un signal booléen, indique à chaque instant l'occurrence ou la non-occurrence d'un comportement. Nous avons introduit une mesure appelée epsilon-count pour les signaux booléens, qui est un comptage monotone proportionnel à la durée pendant laquelle un signal booléen est vrai. Nous avons démontré comment cette mesure, combinée à des spécifications paramétriques monotones, peut être utilisée pour identifier les paramètres qui génèrent des prédicteurs précis. La précision est définie en termes de faux positifs et de faux négatifs par rapport aux prédictions données par des experts. Nous avons présenté un algorithme qui recherche un prédicteur en divisant intelligemment l'espace des paramètres à l'aide du concept de domination de Pareto. Nous avons illustré ces idées en apprenant des spécifications qui caractérisent ou classifient avec précision divers signaux d'électrocardiogramme (ECG).

Dans le cadre de travaux futurs, on peut étudier comment nos méthodes exploitant la monotonie peuvent être intégrées dans des outils qui calculent des spécifications STL générales. La performance

3 https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/StlEval

4 https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/multidimensional_search

5 https://gricad-gitlab.univ-grenoble-alpes.fr/mambakaa/hscc_ptre

optimale sur les données d'apprentissage ne se généralise pas toujours bien. On peut développer des méthodes qui utilisent les ensembles de solutions pour apprendre des spécifications STL robustes qui donnent de bons résultats dans le cas général. Nous expliquons un autre problème mathématique non résolu à l'aide de la formule d'exemple, $F_{[\tau_1,\tau_2]}\varphi$. La formule est monotone par rapport aux paramètres τ_1 et τ_2, mais τ_1 et τ_2 sont liés par une contrainte implicite, $\tau_1 \leqslant \tau_2$. Cela complique la représentation des ensembles de solutions à l'aide de boîtes et nécessite d'examiner l'utilisation d'autres structures de données appropriées. On peut également étudier comment des mesures telles que l'epsilon-count peuvent être intégrées dans la machinerie de TPM en utilisant des automates temporisés et permettre un contrôle approximatif tout en limitant les faux positifs et les faux négatifs comme nous l'avons fait pour STL. Enfin, les solutions à l'identification paramétrique des spécifications peuvent être considérées comme des valeurs de caractéristiques (en anglais "feature"). On peut définir des spécifications qui capturent diverses caractéristiques d'un comportement. Par exemple, les caractéristiques peuvent être la durée des événements, le délai entre deux événements ou les caractéristiques relatives à l'ampleur, comme la hauteur d'un pic. Un algorithme peut être développé pour utiliser les spécifications des caractéristiques afin de déduire une formule permettant de caractériser ou de classer les comportements.

Expressions régulières temporisées paramétriques. Dans cette thèse, nous avons présenté la manière d'effectuer la correspondance paramétrique temporisée (Parametric Timed Pattern Matching PTPM) lorsque les spécifications sont données sous la forme de TRE paramétriques. Pour les versions paramétriques de trois types différents de TRE, nous avons proposé des algorithmes qui effectuent le PTPM en utilisant des types spéciaux de polytopes que nous avons appelés zones paramétriques et intervalles paramétriques. Pour les expressions dont la sémantique est basée sur l'état et qui traitent de signaux à valeurs réelles, nous avons montré que l'appariement implique des opérations sur des zones paramétriques. Nous avons également décrit comment les techniques de balayage de plan peuvent être appliquées pour optimiser les opérations sur les listes de zones paramétriques. Nous avons montré que pour les expressions à sémantique événementielle traitant de mots temporisés, la mise en correspondance implique des intervalles paramétriques. Nous avons expliqué comment le tri et la recherche dichotomique traditionnels peuvent être utilisés pour optimiser les opérations sur les listes d'intervalles paramétriques. En nous appuyant sur le PTPM, nous avons présenté comment l'identification paramétrique peut être effectuée pour apprendre des spécifications à partir d'exemples positifs. Nous avons également introduit le concept de "région décisive" qui caractérise l'ensemble des intervalles de temps qui peuvent conduire à un in-

tervalle de correspondance donné. Ce concept est essentiel pour une identification paramétrique efficace pour TRE paramétrique. En plus des exemples synthétiques, nous avons démontré nos méthodes sur des exemples réels impliquant des signaux ECG et des données de trafic maritime avec des scénarios de croisement de navires.

Dans le cadre de travaux futurs, des structures spécifiques dans les sous-expressions de TRE paramétriques peuvent être exploitées à des fins d'efficacité. Par exemple, les expressions qui ne contiennent aucun paramètre temporel peuvent être entièrement traitées à l'aide de zones bidimensionnelles (non paramétriques) combinées à des boîtes. Un objectif ambitieux consiste à concevoir une grammaire pour le TRE qui combine de manière transparente les aspects sémantiques basés sur les événements et sur les états afin de décrire à la fois les séquences d'événements et les signaux se produisant ensemble dans les données. Il est pertinent de se demander quels types de comportements nécessitent l'utilisation de zones et ne peuvent pas être traités à l'aide d'intervalles de temps. Une utilisation évidente des zones sera de les appliquer lorsque l'occurrence précise d'un événement n'est pas connue. Par exemple, lorsque l'on connaît l'intervalle de temps dans lequel l'événement est garanti d'être trouvé, mais pas l'instant précis. On peut examiner comment divers comportements temporels courants peuvent être exprimés à l'aide de TRE et étudier ceux qui nécessitent une sémantique basée sur l'état et ceux qui nécessitent une sémantique basée sur l'événement. Une étude bibliographique peut être menée pour trouver des travaux qui utilisent STL pour spécifier des modèles temporels et l'on peut essayer de trouver des TRE qui expriment les mêmes modèles. Une tentative formelle de traduction de STL à TRE et vice-versa élargira notre compréhension des relations temporelles. On peut également analyser comment combiner les théories de PTPM pour les TRE paramétriques et les automates temporisés paramétriques (PTA). Cela nous aidera à comprendre quelles tâches sont plus faciles et intuitives pour les PTA et lesquelles sont plus appropriées pour les TRE paramétriques. Enfin, la monotonie peut également être observée dans le PTPM. On peut examiner comment cela peut être utilisé pour intégrer des méthodes qui apprennent TRE dans ParetoLib.

8

CONCLUSIONS

In this thesis, we extended timed formalisms (Signal Temporal Logic (STL) and Timed Regular Expressions (TRE)) with new operators and also developed methods for their parametric analysis. We integrated the data structures and algorithms we developed into existing tools like StlEval [1] and Paretolib [15] as well as a new tool named `parameTRE`[2]. We have been inspired by the min/max operators introduced into STL [14] and the theory of Timed Pattern Matching (TPM) developed for TRE [72]. Before concluding by discussing the main aspects of the thesis we recall some minor contributions. We augmented STL with operators for events and showed their use over suitable examples. We also discussed how atomic predicates involving extrema over intervals can be introduced into TRE and how to analyze them. As future work, new operators with integration, differentiation can be added to STL and TRE. We now summarize the key concepts and contributions of this thesis along with discussions on possible directions future work can take.

Monotonic Specifications and Pareto Domination. We introduced a framework in which a specification is viewed as predicting a pattern in a signal. The prediction in the form of a Boolean signal indicates at each time point occurrence or non-occurrence of a behaviour. We introduced a measure called epsilon-count for Boolean signals which is a monotonic count proportional to the amount of time a Boolean signal is true. We demonstrated how this measure when combined with monotonic parametric specifications can be used for identifying parameters that generate accurate predictors. The accuracy is defined in terms of false positives and false negatives when compared to a predictions given by experts. We presented an algorithm that searches for a predictor by intelligently dividing the parameter space using the concept of Pareto domination. We demonstrated the ideas by learning specifications that accurately characterize or classify various Electrocardiogram (ECG) signals.

As future work, one can explore how our methods exploiting monotonicity can be integrated into tools that compute general STL specifications. Optimal performance over training data does not always generalize well. One can develop methods that use the solution sets to learn robust STL specifications that perform well on the general case. We explain another unsolved mathematical problem using the example formula, $F_{[\tau_1,\tau_2]}\,\varphi$. The formula is monotonic with respect to

1 https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/StlEval
2 https://gricad-gitlab.univ-grenoble-alpes.fr/mambakaa/hscc_ptre

the parameters τ_1 and τ_2, but τ_1 and τ_2 are related by an implicit constraint, $\tau_1 \leqslant \tau_2$. This complicates the representation of solution sets using boxes and requires examining the use of other suitable data structures. One can also explore how measures like epsilon-count can be integrated into the machinery of TPM using Timed Automata (TA) and allow approximate monitoring while bounding false positives and false negatives like we did for STL. Finally, solutions to parametric identification of specifications can be viewed as feature values. One can define specifications that capture various features of a behaviour. For example, the features can be durations of events, delays between two events or features relating to magnitude like the height of a peak. An algorithm can be developed that can make use of the feature specifications to deduce a formula for characterizing or classifying behaviours.

Parametric Timed Regular Expressions. In this thesis, we presented how to perform Parametric Timed Pattern Matching (PTPM) when specifications are given as parametric TRE. For parametric versions of three different kinds of TRE we proposed algorithms that perform PTPM using special kinds of polytopes which we termed parametric zones and parametric intervals. For the expressions with state-based semantics dealing with real-valued signals we showed that matching involves operations over parametric zones. We also described how plane-sweep techniques can be applied to optimize operations over lists of parametric zones. We showed that for expressions with event-based semantics dealing with timed words matching involves parametric intervals. We explained how traditional sorting and searching can be used to optimize operations over list of parametric intervals. Building on top of PTPM we presented how parametric identification can be performed for learning specifications from positive examples. We also introduced the concept of "decisive region" that characterizes the set of time intervals that can lead to a given matching interval. This concept is essential for efficient parametric identification for parametric TRE. Along with synthetic examples, we also demonstrated our methods over real-world examples involving ECG signals and marine traffic data with ship-crossing scenarios.

As future work, specific structures in sub-expressions of PTRE can be exploited for efficiency. For example, expressions that do not contain any timing parameters can be entirely processed using two-dimensional zones (non-parametric) combined with boxes. An ambitious future pursuit will be to devise a grammar for TRE that seamlessly combines event-based and state-based semantic aspects to describe both event sequences and signals occurring together in data. It is pertinent to ask matching what kinds of behaviours require use of zones and cannot be handled using time intervals. One obvious use of zones will be to apply them when the precise occurrence of an event is not known. For example, when one knows the time interval

in which the event is guaranteed to be found but not the precise time stamp. One can examine how various common timed behaviours can be expressed using TRE and explore which ones need state-based semantics and which require event-base semantics. A survey can done to find works that use STL to specify temporal patterns and one can try to come up with TRE that express the same patterns. A formal attempt to translate from STL to TRE and vice-versa will expand our understanding of timed relations. One can also analyze how to combine the theories of PTPM for parametric TRE and Parametric Timed Automata (PTA). This will help us in understanding which tasks are easier and intuitive for PTA and which are more suitable for parametric TRE. Finally, monotonicity can also be observed in the PTPM. One can examine how this can be used to integrate methods that learn TRE into ParetoLib.

BIBLIOGRAPHY

[1] Houssam Abbas, Rajeev Alur, Konstantinos Mamouras, Rahul Mangharam, and Alëna Rodionova. "Quantitative Regular Expressions for Monitoring Cardiac Arrhythmias." In: *3rd Workshop on Monitoring and Testing of Cyber-Physical Systems, MT@CPSWeek 2018, Porto, Portugal, April 10, 2018*. IEEE, 2018, pp. 1–2. DOI: 10.1109/MT-CPS.2018.00007. URL: https://doi.org/10.1109/MT-CPS.2018.00007.

[2] Houssam Abbas, Alëna Rodionova, Ezio Bartocci, Scott A. Smolka, and Radu Grosu. "Quantitative Regular Expressions for Arrhythmia Detection Algorithms." In: *Computational Methods in Systems Biology - 15th International Conference, CMSB 2017, Darmstadt, Germany, September 27-29, 2017, Proceedings*. Ed. by Jérôme Feret and Heinz Koeppl. Vol. 10545. Lecture Notes in Computer Science. Springer, 2017, pp. 23–39. DOI: 10.1007/978-3-319-67471-1_2. URL: https://doi.org/10.1007/978-3-319-67471-1_2.

[3] Houssam Abbas, Alena Rodionova, Konstantinos Mamouras, Ezio Bartocci, Scott A. Smolka, and Radu Grosu. "Quantitative Regular Expressions for Arrhythmia Detection." In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 16.5 (2019), pp. 1586–1597.

[4] Rajeev Alur and David L. Dill. "A Theory of Timed Automata." In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. ISSN: 0304-3975.

[5] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. "Parametric real-time reasoning." In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, CA, USA: ACM, 1993, pp. 592–601. DOI: 10.1145/167088.167242. URL: https://doi.org/10.1145/167088.167242.

[6] Étienne André, Ichiro Hasuo, and Masaki Waga. "Offline Timed Pattern Matching under Uncertainty." In: *23rd International Conference on Engineering of Complex Computer Systems, ICECCS 2018, Melbourne, Australia, December 12-14, 2018*. IEEE Computer Society, 2018, pp. 10–20.

[7] Dana Angluin. "Learning Regular Sets from Queries and Counterexamples." In: *Inf. Comput.* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6. URL: https://doi.org/10.1016/0890-5401(87)90052-6.

[8] Eugene Asarin, Paul Caspi, and Oded Maler. "A Kleene Theorem for Timed Automata." In: *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, June 29 - July 2, 1997*. Warsaw, Poland: IEEE Computer Society, 1997, pp. 160–171. DOI: 10.1109/LICS.1997.614944. URL: https://doi.org/10.1109/LICS.1997.614944.

[9] Eugene Asarin, Paul Caspi, and Oded Maler. "Timed regular expressions." In: *J. ACM* 49.2 (2002), pp. 172–206.

[10] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. "Parametric Identification of Temporal Properties." In: *RV*. Vol. 7186. Lecture Notes in Computer Science. Springer, 2011, pp. 147–160.

[11] Eugene Asarin, Thomas Ferrère, Dejan Nickovic, and Dogan Ulus. "On the Complexity of Timed Pattern Matching." In: *Formal Modeling and Analysis of Timed Systems - 19th International Conference, FORMATS 2021, Paris, France, August 24-26, 2021, Proceedings*. Ed. by Catalin Dima and Mahsa Shirmohammadi. Vol. 12860. Lecture Notes in Computer Science. Springer, 2021, pp. 15–31. DOI: 10.1007/978-3-030-85037-1_2. URL: https://doi.org/10.1007/978-3-030-85037-1_2.

[12] Sertaç Kagan Aydin and Ebru Aydin Gol. "Synthesis of Monitoring Rules with STL." In: *J. Circuits Syst. Comput.* 29.11 (2020), 2050177:1–2050177:26. DOI: 10.1142/S0218126620501777. URL: https://doi.org/10.1142/S0218126620501777.

[13] Roberto Bagnara, Patricia Hill, and Enea Zaffanella. "The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems." In: *Science of Computer Programming* 72 (June 2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001.

[14] Alexey Bakhirkin and Nicolas Basset. "Specification and Efficient Monitoring Beyond STL." In: *TACAS (2)*. Vol. 11428. Lecture Notes in Computer Science. Springer, 2019, pp. 79–97.

[15] Alexey Bakhirkin, Nicolas Basset, Oded Maler, and José Ignacio Requeno. "ParetoLib: A Python Library for Parameter Synthesis." In: *Proceedings of the 17th International Conference on Formal Modeling and Analysis of Timed Systems*. Vol. 11750. Theoretical Computer Science and General Issues. Cham: Springer, 2019, pp. 114–120. ISBN: 978-3-030-29662-9.

[16] Alexey Bakhirkin, Thomas Ferrére, and Oded Maler. "Efficient Parametric Identification for STL." In: *Proc. 21st Int. Conf. on Hybrid Systems: Computation and Control*. HSCC'18. New York, NY, USA: ACM, 2018, pp. 177–186.

[17] Alexey Bakhirkin, Thomas Ferrère, Oded Maler, and Dogan Ulus. "On the Quantitative Semantics of Regular Expressions over Real-Valued Signals." In: *FORMATS*. Vol. 10419. Lecture Notes in Computer Science. Springer, 2017, pp. 189–206.

[18] Alexey Bakhirkin, Thomas Ferrère, Dejan Nickovic, Oded Maler, and Eugene Asarin. "Online Timed Pattern Matching Using Automata." In: *FORMATS*. Vol. 11022. Lecture Notes in Computer Science. Springer, 2018, pp. 215–232.

[19] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. "Learning Temporal Logical Properties Discriminating ECG models of Cardiac Arrhytmias." In: *CoRR* abs/1312.7523 (2013). arXiv: 1312.7523. URL: http://arxiv.org/abs/1312.7523.

[20] Ezio Bartocci, Luca Bortolussi, and Guido Sanguinetti. "Data-Driven Statistical Learning of Temporal Logic Properties." In: *Formal Modeling and Analysis of Timed Systems - 12th International Conference, FORMATS 2014, Florence, Italy, September 8-10, 2014. Proceedings*. Ed. by Axel Legay and Marius Bozga. Vol. 8711. Lecture Notes in Computer Science. Springer, 2014, pp. 23–37. DOI: 10.1007/978-3-319-10512-3_3. URL: https://doi.org/10.1007/978-3-319-10512-3_3.

[21] Ezio Bartocci, Jyotirmoy Deshmukh, Felix Gigler, Cristinel Mateis, Dejan Nickovic, and Xin Qin. "Mining Shape Expressions From Positive Examples." In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.11 (2020), pp. 3809–3820. DOI: 10.1109/TCAD.2020.3012240. URL: https://doi.org/10.1109/TCAD.2020.3012240.

[22] Ezio Bartocci, Jyotirmoy Deshmukh, Cristinel Mateis, Eleonora Nesterini, Dejan Nickovic, and Xin Qin. "Mining Shape Expressions with ShapeIt." In: *Software Engineering and Formal Methods - 19th International Conference, SEFM 2021, Virtual Event, December 6-10, 2021, Proceedings*. Ed. by Radu Calinescu and Corina S. Pasareanu. Vol. 13085. Lecture Notes in Computer Science. Springer, 2021, pp. 110–117. DOI: 10.1007/978-3-030-92124-8_7. URL: https://doi.org/10.1007/978-3-030-92124-8_7.

[23] Ezio Bartocci, Cristinel Mateis, Eleonora Nesterini, and Dejan Nickovic. "Survey on mining signal temporal logic specifications." In: *Inf. Comput.* 289.Part (2022), p. 104957. DOI: 10.1016/j.ic.2022.104957. URL: https://doi.org/10.1016/j.ic.2022.104957.

[24] Giuseppe Bombara and Calin Belta. "Signal Clustering Using Temporal Logics." In: *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*. Ed. by Shuvendu K. Lahiri and Giles Reger. Vol. 10548. Lecture Notes in Computer Science. Springer, 2017, pp. 121–

137. DOI: 10.1007/978-3-319-67531-2_8. URL: https://doi.org/10.1007/978-3-319-67531-2_8.

[25] Giuseppe Bombara and Calin Belta. "Online Learning of Temporal Logic Formulae for Signal Classification." In: *16th European Control Conference, ECC 2018, Limassol, Cyprus, June 12-15, 2018*. IEEE, 2018, pp. 2057–2062. DOI: 10.23919/ECC.2018.8550271. URL: https://doi.org/10.23919/ECC.2018.8550271.

[26] Giuseppe Bombara and Calin Belta. "Offline and Online Learning of Signal Temporal Logic Formulae Using Decision Trees." In: *ACM Trans. Cyber Phys. Syst.* 5.3 (2021), 22:1–22:23. DOI: 10.1145/3433994. URL: https://doi.org/10.1145/3433994.

[27] Giuseppe Bombara, Cristian Ioan Vasile, Francisco Penedo, Hirotoshi Yasuoka, and Calin Belta. "A Decision Tree Approach to Data Classification using Signal Temporal Logic." In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. HSCC'16. New York, NY, USA: ACM, 2016, pp. 1–10.

[28] Lubos Brim, Petr Dluhos, David Safránek, and Tomas Vejpustek. "STL*: Extending signal temporal logic with signal-value freezing operator." In: *Inf. Comput.* 236 (2014), pp. 52–67. DOI: 10.1016/j.ic.2014.01.012. URL: https://doi.org/10.1016/j.ic.2014.01.012.

[29] Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. "A Simulink Hybrid Heart Model for Quantitative Verification of Cardiac Pacemakers." In: *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*. HSCC'13. New York, NY, USA: ACM, 2013, pp. 131–136.

[30] G. Chiandussi, M. Codegone, S. Ferrero, and F.E. Varesio. "Comparison of multi-objective optimization methodologies for engineering applications." In: *Computers & Mathematics with Applications* 63.5 (2012), pp. 912–942. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2011.11.057. URL: https://www.sciencedirect.com/science/article/pii/S0898122111010406.

[31] Greta Cutulenco, Yogi Joshi, Apurva Narayan, and Sebastian Fischmeister. "Mining timed regular expressions from system traces." In: *Proceedings of the 5th International Workshop on Software Mining, SoftwareMining@ASE 2016, Singapore, Singapore, September 3, 2016*. Ed. by Ming Li, Xiaoyin Wang, and Lucia. ACM, 2016, pp. 3–10. DOI: 10.1145/2975961.2975962. URL: https://doi.org/10.1145/2975961.2975962.

[32] Hoang Anh Dau et al. *The UCR Time Series Classification Archive*. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/. Oct. 2018.

[33] Alexandre Donzé, Thomas Ferrère, and Oded Maler. "Efficient Robust Monitoring for STL." In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 264–279.

[34] Alexandre Donzé and Oded Maler. "Robust Satisfaction of Temporal Logic over Real-Valued Signals." In: *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, September 8-10, 2010. Proceedings*. Ed. by Krishnendu Chatterjee and Thomas A. Henzinger. Vol. 6246. Lecture Notes in Computer Science. Klosterneuburg, Austria: Springer, 2010, pp. 92–106. DOI: 10.1007/978-3-642-15297-9_9. URL: https://doi.org/10.1007/978-3-642-15297-9_9.

[35] *ECGFiveDays data set*. http://www.timeseriesclassification.com/description.php?Dataset=ECGFiveDays.

[36] Henning Fernau. "Algorithms for learning regular expressions from positive data." In: *Information and Computation* 207.4 (2009), pp. 521–541. ISSN: 0890-5401. DOI: https://doi.org/10.1016/j.ic.2008.12.008. URL: https://www.sciencedirect.com/science/article/pii/S0890540109000169.

[37] Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus. "Measuring with Timed Patterns." In: *CAV (2)*. Vol. 9207. Lecture Notes in Computer Science. Springer, 2015, pp. 322–337.

[38] Ary L. Goldberger, Luis A.N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals." In: *Circulation* 101.23 (2000), e215–e220.

[39] Olga Grinchtein, Bengt Jonsson, and Martin Leucker. "Learning of event-recording automata." In: *Theor. Comput. Sci.* 411.47 (2010), pp. 4029–4054. DOI: 10.1016/j.tcs.2010.07.008. URL: https://doi.org/10.1016/j.tcs.2010.07.008.

[40] Léo Henry, Thierry Jéron, and Nicolas Markey. "Active Learning of Timed Automata with Unobservable Resets." In: *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*. Ed. by Nathalie Bertrand and Nils Jansen. Vol. 12288. Lecture Notes in Computer Science. Springer, 2020, pp. 144–160. DOI: 10.1007/978-3-030-57628-8_9. URL: https://doi.org/10.1007/978-3-030-57628-8_9.

[41] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. "TeLEx: Passive STL Learning Using Only Positive Examples." In: *Runtime Verification - 17th International Conference, RV 2017, Seattle, WA, USA, September 13-16, 2017, Proceedings*. Ed. by Shuvendu K. Lahiri and Giles Reger. Vol. 10548. Lecture Notes in Computer Science. Springer, 2017, pp. 208–224. DOI: 10.1007/978-3-319-67531-2_13. URL: https://doi.org/10.1007/978-3-319-67531-2_13.

[42] Susmit Jha, Ashish Tiwari, Sanjit A. Seshia, Tuhin Sahai, and Natarajan Shankar. "TeLEx: learning signal temporal logic from positive examples using tightness metric." In: *Formal Methods in System Design* 54.3 (2019), pp. 364–387.

[43] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. "Mining requirements from closed-loop control models." In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, pp. 43–52. DOI: 10.1145/2461328.2461337. URL: https://doi.org/10.1145/2461328.2461337.

[44] Andrey N. Kolmogorov and Vladimir M. Tikhomirov. "ε-entropy and ε-capacity of sets in function spaces." In: *Uspekhi Matematicheskikh Nauk* 14.2(86) (1959), p. 386.

[45] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. "Temporal logic inference for classification and prediction from data." In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. HSCC'14. New York, NY, USA: ACM, 2014, pp. 273–282.

[46] Zhaodan Kong, Austin Jones, and Calin Belta. "Temporal Logics for Learning and Detection of Anomalous Behavior." In: *IEEE Transactions on Automatic Control* 62.3 (2017), pp. 1210–1222.

[47] Hanna Krasowski and Matthias Althoff. "Temporal Logic Formalization of Marine Traffic Rules." In: *IEEE Intelligent Vehicles Symposium, IV 2021, Nagoya, Japan, July 11-17, 2021*. IEEE, 2021, pp. 186–192. DOI: 10.1109/IV48863.2021.9575685. URL: https://doi.org/10.1109/IV48863.2021.9575685.

[48] Shankara Narayanan Krishna, Khushraj Madnani, and Paritosh K. Pandya. "Metric Temporal Logic with Counting." In: *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*. Ed. by Bart Jacobs and Christof Löding. Vol. 9634. Lecture Notes in Computer Science. Springer, 2016, pp. 335–352. DOI: 10.1007/

978-3-662-49630-5_20. URL: https://doi.org/10.1007/978-3-662-49630-5_20.

[49] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. "Approximating the Pareto Front of Multi-criteria Optimization Problems." In: *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 6015. Theoretical Computer Science and General Issues. Berlin, Heidelberg: Springer, 2010, pp. 69–83. ISBN: 978-3-642-12002-2.

[50] Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. "General LTL Specification Mining (T)." In: *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering*. ASE'15. IEEE, 2015, pp. 81–92.

[51] Karen Leung, Nikos Aréchiga, and Marco Pavone. "Back-Propagation Through Signal Temporal Logic Specifications: Infusing Logical Structure into Gradient-Based Methods." In: *Algorithmic Foundations of Robotics XIV, Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2021, Oulu, Finland, June 21-23, 2021*. Ed. by Steven M. LaValle, Ming Lin, Timo Ojala, Dylan A. Shell, and Jingjin Yu. Vol. 17. Springer Proceedings in Advanced Robotics. Springer, 2021, pp. 432–449. DOI: 10.1007/978-3-030-66723-8_26. URL: https://doi.org/10.1007/978-3-030-66723-8_26.

[52] Shang-Wei Lin, Étienne André, Jin Song Dong, Jun Sun, and Yang Liu. "An Efficient Algorithm for Learning Event-Recording Automata." In: *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*. Ed. by Tevfik Bultan and Pao-Ann Hsiung. Vol. 6996. Lecture Notes in Computer Science. Springer, 2011, pp. 463–472. DOI: 10.1007/978-3-642-24372-1_35. URL: https://doi.org/10.1007/978-3-642-24372-1_35.

[53] Shang-Wei Lin, Étienne André, Yang Liu, Jun Sun, and Jin Song Dong. "Learning Assumptions for CompositionalVerification of Timed Systems." In: *IEEE Transactions on Software Engineering* 40.2 (2014), pp. 137–153. DOI: 10.1109/TSE.2013.57.

[54] Pradeep K. Mahato and Apurva Narayan. "QMine: A Framework for Mining Quantitative Regular Expressions from System Traces." In: *20th IEEE International Conference on Software Quality, Reliability and Security Companion, QRS Companion 2020, Macau, China, December 11-14, 2020*. IEEE, 2020, pp. 370–377. DOI: 10.1109/QRS-C51114.2020.00070. URL: https://doi.org/10.1109/QRS-C51114.2020.00070.

[55] Oded Maler. "Learning Monotone Partitions of Partially-Ordered Domains (Work in Progress)." Working paper or preprint. 2017. URL: https://hal.archives-ouvertes.fr/hal-01556243.

[56] Oded Maler and Dejan Nickovic. "Monitoring Temporal Properties of Continuous Signals." In: *FORMATS/FTRTFT*. Vol. 3253. Lecture Notes in Computer Science. Springer, 2004, pp. 152–166.

[57] Oded Maler, Dejan Nickovic, and Amir Pnueli. "From MITL to Timed Automata." In: *FORMATS*. Vol. 4202. Lecture Notes in Computer Science. Paris, France: Springer, 2006, pp. 274–289.

[58] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, and Aniruddh G. Puranic. "Mining Environment Assumptions for Cyber-Physical System Models." In: *Proceedings of the 11th ACM/IEEE International Conference on Cyber-Physical Systems (to appear)*. ICCPS'20. Sydney, Australia: IEEE, 2020.

[59] Sara Mohammadinejad, Jyotirmoy V. Deshmukh, Aniruddh Gopinath Puranic, Marcell Vazquez-Chanlatte, and Alexandre Donzé. "Interpretable classification of time-series data using efficient enumerative techniques." In: *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*. Ed. by Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh. ACM, 2020, 9:1–9:10. DOI: 10.1145/3365365.3382218. URL: https://doi.org/10.1145/3365365.3382218.

[60] George B Moody and Roger G Mark. "The impact of the MIT-BIH Arrhythmia Database." In: *IEEE Engineering in Medicine and Biology Magazine* 20.3 (2001), pp. 45–50.

[61] Daniel Neider and Ivan Gavran. "Learning Linear Temporal Properties." In: *Proceedings of the 18th International Conference on Formal Methods in Computer Aided Design*. FMCAD'11. Austin, TX, USA: ACM, 2018, pp. 1–10.

[62] Laura Nenzi, Simone Silvetti, Ezio Bartocci, and Luca Bortolussi. "A Robust Genetic Algorithm for Learning Temporal Specifications from Data." In: *Quantitative Evaluation of Systems - 15th International Conference, QEST 2018, Beijing, China, September 4-7, 2018, Proceedings*. Ed. by Annabelle McIver and András Horváth. Vol. 11024. Lecture Notes in Computer Science. Springer, 2018, pp. 323–338. DOI: 10.1007/978-3-319-99154-2_20. URL: https://doi.org/10.1007/978-3-319-99154-2_20.

[63] Dejan Nickovic, Xin Qin, Thomas Ferrère, Cristinel Mateis, and Jyotirmoy V. Deshmukh. "Shape Expressions for Specifying and Extracting Signal Features." In: *Proceedings of the 19th In-*

ternational Conference on Runtime Verification. Vol. 11757. Lecture Notes in Computer Science. Cham: Springer, 2019, pp. 292–309.

[64] Dejan Nickovic, Xin Qin, Thomas Ferrère, Cristinel Mateis, and Jyotirmoy Deshmukh. "Specifying and detecting temporal patterns with shape expressions." In: *Int. J. Softw. Tools Technol. Transf.* 23.4 (2021), pp. 565–577. DOI: 10.1007/s10009-021-00627-x. URL: https://doi.org/10.1007/s10009-021-00627-x.

[65] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. "Timed k-Tail: Automatic Inference of Timed Automata." In: *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*. IEEE Computer Society, 2017, pp. 401–411. DOI: 10.1109/ICST.2017.43. URL: https://doi.org/10.1109/ICST.2017.43.

[66] Amir Pnueli. "The temporal logic of programs." In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[67] Neda Saeedloei and Feliks Kluzniak. "Synthesizing Clock-Efficient Timed Automata." In: *Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings*. Ed. by Brijesh Dongol and Elena Troubitsyna. Vol. 12546. Lecture Notes in Computer Science. Springer, 2020, pp. 276–294. DOI: 10.1007/978-3-030-63461-2_15. URL: https://doi.org/10.1007/978-3-030-63461-2_15.

[68] Ocan Sankur. "Timed Automata Verification and Synthesis via Finite Automata Learning." In: *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II*. Ed. by Sriram Sankaranarayanan and Natasha Sharygina. Vol. 13994. Lecture Notes in Computer Science. Springer, 2023, pp. 329–349. DOI: 10.1007/978-3-031-30820-8_21. URL: https://doi.org/10.1007/978-3-031-30820-8_21.

[69] Wei Shen, Jie An, Bohua Zhan, Miaomiao Zhang, Bai Xue, and Naijun Zhan. "PAC Learning of Deterministic One-Clock Timed Automata." In: *Formal Methods and Software Engineering - 22nd International Conference on Formal Engineering Methods, ICFEM 2020, Singapore, Singapore, March 1-3, 2021, Proceedings*. Ed. by Shang-Wei Lin, Zhe Hou, and Brendan P. Mahony. Vol. 12531. Lecture Notes in Computer Science. Springer, 2020, pp. 129–146. DOI: 10.1007/978-3-030-63406-3_8. URL: https://doi.org/10.1007/978-3-030-63406-3_8.

[70] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. "Time to Learn - Learning Timed Automata from Tests." In: *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*. Ed. by Étienne André and Mariëlle Stoelinga. Vol. 11750. Lecture Notes in Computer Science. Springer, 2019, pp. 216–235. DOI: 10.1007/978-3-030-29662-9_13. URL: https://doi.org/10.1007/978-3-030-29662-9_13.

[71] Martin Tappler, Bernhard K. Aichernig, and Florian Lorber. "Timed Automata Learning via SMT Solving." In: *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*. Ed. by Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez. Vol. 13260. Lecture Notes in Computer Science. Springer, 2022, pp. 489–507. DOI: 10.1007/978-3-031-06773-0_26. URL: https://doi.org/10.1007/978-3-031-06773-0_26.

[72] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. "Timed Pattern Matching." In: *FORMATS*. Vol. 8711. Lecture Notes in Computer Science. Springer, 2014, pp. 222–236.

[73] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. "Online Timed Pattern Matching Using Derivatives." In: *TACAS*. Vol. 9636. Lecture Notes in Computer Science. Springer, 2016, pp. 736–751.

[74] Dogan Ulus and Oded Maler. "Specifying Timed Patterns using Temporal Logic." In: *HSCC*. Porto, Portugal: ACM, 2018, pp. 167–176.

[75] Prashant Vaidyanathan, Rachael Ivison, Giuseppe Bombara, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta. "Grid-based temporal logic inference." In: *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*. IEEE, 2017, pp. 5354–5359. DOI: 10.1109/CDC.2017.8264452. URL: https://doi.org/10.1109/CDC.2017.8264452.

[76] Marcell Vazquez-Chanlatte, Jyotirmoy V. Deshmukh, Xiaoqing Jin, and Sanjit A. Seshia. "Logical Clustering and Learning for Time-Series Data." In: *Proceedings of the 29th International Conference on Computer Aided Verification*. Vol. 10426. Theoretical Computer Science and General Issues. Cham: Springer, 2017, pp. 305–325. ISBN: 978-3-319-63387-9.

[77] Marcell Vazquez-Chanlatte, Shromona Ghosh, Jyotirmoy V. Deshmukh, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. "Time-Series Learning Using Monotonic Logical Properties." In: *Proceedings of the 18th International Conference on Run-*

time Verification. Vol. 11237. Lecture Notes in Computer Science. Cham: Springer, 2018, pp. 389–405. ISBN: 978-3-030-03769-7.

[78] S. Verwer, M. D. Weerdt, and C. Witteveen. "An algorithm for learning real-time automata." In: 2007.

[79] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. "The Efficiency of Identifying Timed Automata and the Power of Clocks." In: *Inf. Comput.* 209.3 (Mar. 2011), 606–625. ISSN: 0890-5401. DOI: 10.1016/j.ic.2010.11.023. URL: https://doi.org/10.1016/j.ic.2010.11.023.

[80] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. "Efficiently identifying deterministic real-time automata from labeled data." In: *Mach. Learn.* 86.3 (2012), pp. 295–333. DOI: 10.1007/s10994-011-5265-4. URL: https://doi.org/10.1007/s10994-011-5265-4.

[81] Masaki Waga, Takumi Akazaki, and Ichiro Hasuo. "A Boyer-Moore Type Algorithm for Timed Pattern Matching." In: *Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-26, 2016, Proceedings*. Ed. by Martin Fränzle and Nicolas Markey. Vol. 9884. Lecture Notes in Computer Science. Springer, 2016, pp. 121–139. DOI: 10.1007/978-3-319-44878-7_8. URL: https://doi.org/10.1007/978-3-319-44878-7_8.

[82] Masaki Waga and Étienne André. "Online Parametric Timed Pattern Matching with Automata-Based Skipping." In: *NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings*. Ed. by Julia M. Badger and Kristin Yvonne Rozier. Vol. 11460. Lecture Notes in Computer Science. Springer, 2019, pp. 371–389.

[83] Masaki Waga, Étienne André, and Ichiro Hasuo. "Symbolic Monitoring Against Specifications Parametric in Time and Data." In: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11561. Lecture Notes in Computer Science. Springer, 2019, pp. 520–539.

[84] Masaki Waga, Étienne André, and Ichiro Hasuo. "Parametric Timed Pattern Matching." In: *ACM Trans. Softw. Eng. Methodol.* (Feb. 2022). ISSN: 1049-331X. DOI: 10.1145/3517194.

[85] Masaki Waga, Ichiro Hasuo, and Kohei Suenaga. "Efficient Online Timed Pattern Matching by Automata-Based Skipping." In: *Formal Modeling and Analysis of Timed Systems - 15th International Conference, FORMATS 2017, Berlin, Germany, September 5-7, 2017, Proceedings*. Ed. by Alessandro Abate and Gilles Geeraerts. Vol. 10419. Lecture Notes in Computer Science. Springer,

2017, pp. 224–243. DOI: 10.1007/978-3-319-65765-3_13. URL: https://doi.org/10.1007/978-3-319-65765-3_13.