Università degli Studi di Udine

Corso di Laurea Magistrale in

Informatica

Multiple Formulas and Multiple Models Finite Interval-based Model Checking

---

Relatore:

Prof. Angelo Montanari

Laureando:

Ionel Eduard Stan

Correlatori:

Prof. Guido Sciavicco

Dr. Dario Della Monica

Anno Accademico 2018 – 2019

Abstract

Formal methods are a family of structured methodologies whose purpose is to mathematically establish the correctness of critical systems; safety and reliability are fundamental requirements in such a type of systems. The most representative family of formal methods is *Model Checking* (MC for short). MC is the problem of checking whether a given logical property, written is some specification language, is satisfied by a certain finite-state transition system (in this case, we call the problem the 1-to-1 MC problem). MC techniques have been developed since the late 80s, bearing in mind the well-known *point-based* temporal logics LTL and CTL. While these mathematical formalisms are a natural choice when reasoning over the states, that is, points, of a given computation, there are properties one may want to check that are inherently *interval-based*, that is, time periods. In order to deal with these properties, *interval* temporal logics, whose primitive ontological entities are time intervals, come in to play. The most prominent logical formalism when dealing with time intervals is *Halpern and Shoham's (multi-)modal logic of time intervals* (HS for short), which is built on top of *Allen's interval algebra*. The MC problem for interval temporal logics entered the research agenda only very recently. In the present work, we focus our attention on the *finite* MC problem, that is, the problem of checking an HS formula against a single finite path/history.

We systematically investigate the generalization of the *finite* MC problem to the multiple case, where a collection of m models and a family of n temporal properties are given as input. We first show how the m-to-n generalization

can be successfully applied in at least two cutting-edge application domains following a naive approach, that is, by simply calling $m \cdot n$ times the algorithm that solves the 1-to-1 MC problem. Once we convinced ourselves that such a problem is just one among a meaningful set of similar problems, by making use of some notions borrowed from probability theory, we devise a clever procedure, that gives us a better (expected) upper bound to the m-to-n MC problem.

Keywords: Interval temporal logics, Model checking.

Sommario

I *Metodi Formali* sono una famiglia di metodologie strutturate il cui scopo è stabilire in modo formale la correttezza di sistemi critici, nei quali la sicurezza e l'affidabilità sono requisiti fondamentali. La famiglia di metodi formali più rappresentativa è probabilmente quella degli algoritmi di *Model Checking* (MC per brevità). MC è il problema di stabilire se una data proprietà logica, scritta in un dato linguaggio di specifica, è soddisfatta da un certo sistema di transizione a stati finiti. In questo caso, parleremo di un problema di MC di tipo 1-a-1. Le tecniche di MC sono state sviluppate a partire dalla fine degli anni 80, utilizzando le note logiche temporali *puntuali* LTL e CTL come linguaggi di specifica. Tali formalismi sono una scelta naturale quando si vuol ragionare sugli stati, ossia i punti, di una data computazione, ma vi sono proprietà che potremmo voler verificare che sono inerentemente *intervallari*, ossia legate a periodi di tempo. In tal caso, le logiche temporali ad *intervalli*, dove le entità ontologiche primitive sono gli intervalli, diventano i candidati naturali. Quando ragioniamo sugli intervalli, il formalismo logico più rappresentativo è la *logica (multi-)modale degli intervalli temporale di Halpern and Shoham* (HS per brevità), basata sull'*algebra intervallare di Allen*. Il problema MC nel caso intervallare è entrato nell'agenda della ricerca solo molto di recente.

In questa tesi, focalizzeremo la nostra attenzione sul problema MC *finito*, vale a dire il problema di verificare una data formula di HS rispetto ad un singolo cammino/storia. Più precisamente, investighiamo sistematicamente la generalizzazione di tale problema al caso multiplo, dove una famiglia di

m modelli e una collezione di n proprietà temporali sono fornite in input (caso m-a-n). Mostriamo innanzitutto come tale generalizzazione possa essere utilizzata con successo in almeno due domani applicativi di particolare attualità attraverso un approccio naive, ossia chiamando $m \cdot n$ volte l'algoritmo che risolve il problema 1-a-1. A partire dalla constatazione che il problema considerato è uno di un ampio insieme di problemi simili da risolvere, sfruttando alcune nozioni mutuate dalla teoria delle probabilità, forniamo una procedura più sofisticata e dimostriamo che essa fornisce un miglior limite superiore alla complessità del problema MC finito intervallare m-a-n.

Parole chiave: Logiche temporali ad intervalli, Model checking.

Acknowledgements

I would like to express my entire gratitude, first of all, by thanking the people that have orchestrated this thesis: Angelo Montanari, Guido Sciavicco and Dario Della Monica. Thank you all for every little bit of teaching.

I thank my best friend, Damiano, for being the way he is. I also thank all my friends and my fellow students.

Thanks to my entire family and, most specially, to my parents, Nușa and Dorel, for everything!

Dedication

to my parents	English
a mis padres	Español
ai miei genitori	Italiano
părinților mei	Română

Contents

List of Figures

Chapter 1

Introduction

Model checking [CGP99] is the problem of establishing whether a given logical specification is satisfied by a certain structure. Model checking falls under the umbrella of *formal methods*; in particular, it is used within the *formal verification* context where the goal is to establish the correctness of hardware and software reactive systems with respect to some desired properties by exhaustively exploring all possible behaviours (of the system). The behaviour of a reactive system is captured by specifying the evolution of its state during an execution. Thus, it can be naturally represented as a (directed) labelled transition graph, where the vertices represent the states, which are labelled, the edges represent the transitions (from one state to another), and the paths model the computations of the system. For this reason, the model checking problem has been (historically) addressed in the context *point-based* temporal logics, such as, for instance, LTL, CTL , CTL*, and the like. Another reason is that the mathematical formalisms must predicate about points, that is, states, of a computation, and thus point-based temporal logics turn out to be a natural choice for the specification language.

In the recent literature, model checking with *interval-based* semantics have raised a significant interest in the scientific community (for a comprehensive account of the literature, see [Mol19]). The most prominent logical formalism

when dealing with time intervals is *Halpern and Shoham's (multi-)modal logic of time intervals* [HS91] (HS for short), which has its roots in the algebra of *Allen's relations* [All83]. It is well known that the satisfiability problem for HS is highly undecidable [HS91]. To overcome such a limitation, a lot of work has been done in order to find expressive enough and computationally appealing fragments of HS (see, for instance, [AMG+16, AMI+14, BMG+14, BMM+14, BMM+19, MVPS+19]). The model checking problem for HS has entered the research agenda only very recently. As shown in [Mol19], model checking HS, even under the homogeneity assumption, is computationally quite expensive. As in the case of the satisfiability problem, the picture is different if we restrict our attention to HS fragments (see again [Mol19] as well as [LM13, LM14, LM16]).

In this work, we focus on the *finite* model checking problem, that is, the problem of checking an HS formula against a single finite path/history, which turns out to be much easier than general model checking. In [Var05], Vardi observes that query evaluation and constraint checking in relational databases can be naturally viewed as a finite model checking problem. Time intervals are commonly used to represent temporal information in *temporal databases*. In particular, the cornerstone of temporal data support in SQL:2011 [Zem12] is the ability to define and associate *time periods*, that is, intervals, with records in a table, that allows one to compactly represent time periods over which data are valid. Moreover, period predicates are functionally similar (even though not identical) to Allen's interval operators [CT05]. Therefore, temporal queries, temporal constraints, and temporal rules can be naturally expressed as formulas of an interval temporal logic, such as HS, to be interpreted over temporal databases represented as finite interval models [MdFM+17]. Stated differently, the finite model checking problem for interval temporal logic can be used to evaluate interval formulas against temporal databases to solve problems belonging to the temporal databases and temporal data mining domain spectrum.

A common point that connects all the aforementioned applications, namely, temporal query evaluation, constraint checking and rule evaluation, is that the original data needs to be processed to obtain a temporal data set of interval-based timelines. A method that can be exploited to make such a transformation has been recently proposed in the literature [SSV19]. A graphical account of its

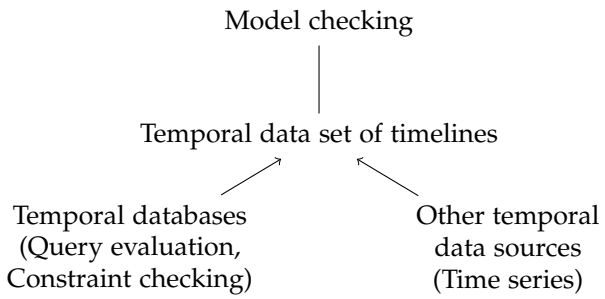


Figure 1.1: Data refinement towards model checking.

structure is given in Figure 1.1. Suppose to have a temporal database containing hospitalization historical data, where each section, such as cardiology and surgery, has its own stored records about patients. Each patient is uniquely identified by his/her *primary key*. In such scenarios, non-trivial temporal relations arise, and a proper treatment of such temporal data is required.

When evaluating a temporal database (by executing temporal queries or checking temporal constraints), the database can be represented/interpreted as a set of finite interval models and the model checking algorithm can be applied to check the (temporal) formula against each interval model. In this case, the procedure (directly or indirectly) returns a set of histories that meet the specification.

In the case of rule evaluation, available public data is generally represented using a *time series* format. A time series is a time-valued data. In order to apply an interval-based model checking, such raw data needs to be processed via *abstraction methods*, as shown in [SSV19]. In this case, some metrics for rating the goodness degree of the interval model with respect to the formula are needed, that is, it is necessary to establish to which extent each single history (respectively, the collection of histories) fulfills a certain temporal rule.

To summarize, there are some features of the considered application domains that make standard model checking procedures inappropriate:

- the problem is naturally interval-based, while classical model checking is point-based;
- models are finite, while, in the classical setting, they can be infinite (even

though finitely representable);

- in standard model checking, the input to the problem consists of a (temporal) formula and a model, while, in the considered cases, it may consist of n formulas to be checked against m models.

The thesis is organized as follows.

In Chapter 2, we start with a gentle introduction to the model checking problem and, then, we formally present the needed mathematical backgroud, i.e., interval temporal logics along with the model checking problem for such a class of logics, where interval-based models are interpreted as *timelines*. We conclude the chapter with an account of the interval-based model checking algorithm given in [MdFM+17]. We will use such an algorithm as the fundamental machinery throughout the thesis. In particular, the algorithm can be viewed as a 1-to-1 one, which takes as input a pair consisting of one interval-based model and one interval-based specification (formula).

In Chapter 3, we formalize the m-to-n generalization of the 1-to-1 finite model checking algorithm, which we call *Multiple Formulas and Multiple Models Finite Interval-based Model Checking* (MMMC, for short): m models against n formulas. Then, we provide a *naive* (algorithmic) solution to such a problem, that essentially calls $m \cdot n$ times the 1-to-1 algorithm, and we show how the presented methodology can be applied in at least two cutting-edge applications.

In Chapter 4, we analyse the time complexity of the naive solution given in the previous chapter. Then, making use of some (basic) notions borrowed from probability theory and exploiting a suitable supporting data structure that stores auxiliary information that might be used in the future (for further checks), we propose an alternative solution to the MMMC problem and we prove that it allows us to achieve a better (expected) execution time.

Finally, in Chapter 5, we conclude the thesis by summarizing its main contributions and outlining some possible future developments.

Chapter 2

Finite Interval Temporal Model Checking

In this chapter, we first gently introduce the interval-based model checking and, then, we give the needed mathematical formalisms used throughout this thesis.

2.1 A gentle introduction to the problem

Temporal data evaluation can be (roughly) defined as the problem of establishing how many histories, i.e., temporal instances of a temporal database, comply with a given temporal formula or constraint. There are at least three possible application domains:

i) *temporal query processing*, where the set of histories that satisfy a given condition must be returned,

ii) *temporal constraint checking*, where the set of histories that violate a given constraint must be identified, and

iii) *rule evaluation*, where one must determine how many histories, and which extent, comply a certain temporal rule.

Model checking is the problem of verifying whether or not a given formula of some logical language is satisfied by a certain model [CGP99]. One of its most successful application is the verification of a point-based temporal logic (e.g., Linear Temporal Logic) against some reactive system description. Query evaluation and constraint checking in relational databases can be naturally expressed as a model checking problem [Var05].

Time intervals are commonly used to express temporal information in temporal databases [CT05, Zem12]. Accordingly, temporal queries, constraints, and rules can be naturally formulated as formulas of an interval temporal logic to be evaluated over temporal data sets represented as finite interval models. The problem of evaluating a temporal data set can thus be reduced to the model checking problem for interval temporal logic formulas, making it possible to exploit techniques and tools from logic and formal methods to address and solve problems in temporal databases and data mining (see, for instance, the three aforementioned application domains).

2.2 Interval temporal logics

Interval temporal logics take as primitive ontological entities *time periods* (i.e., *intervals*), rather than points. The context of interval-based logics is still "young", meaning that it is a current, active research topic. For this reason, we do not have a proper representative of such class of logics, as we have for other logics (e.g., LTL, CTL and CTL* for point-based formalisms). However, Halpern and Shoham in [HS91] presented the *Halpern and Shoham's (multi)-modal logic of time intervals* (HS for short), among with its undecidable result over many classes of linearly ordered sets. The distinguishing elegance and expressive power of HS have attracted the attention of modal and temporal logic communities. In particular, since the undecidable result of full HS from the early nineties, the complete line between decidability and undecidability of the HS fragments has been draw [AMG+16, AMI+14, BMG+14, BMM+14, BMM+19]. Moreover, coarser fragments of HS have been also studied [MVPS+19]. Therefore, we take HS as representative of such class of logics.

Let $\mathbb{D} = \langle D, < \rangle$ be a linear order. A *strict* (respectively, *non-strict*) interval

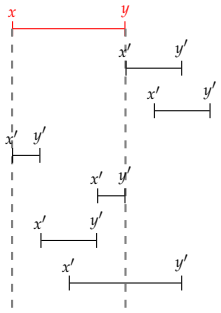
HS	Allen's relations	Graphical representation
$\langle A \rangle$	$[x,y]R_A[x',y'] \Leftrightarrow y = x'$	
$\langle L \rangle$	$[x,y]R_L[x',y'] \Leftrightarrow y < x'$	
$\langle B \rangle$	$[x,y]R_B[x',y'] \Leftrightarrow x = x', y' < y$	
$\langle E \rangle$	$[x,y]R_E[x',y'] \Leftrightarrow y = y', x < x'$	
$\langle D \rangle$	$[x,y]R_D[x',y'] \Leftrightarrow x < x', y' < y$	
$\langle O \rangle$	$[x,y]R_O[x',y'] \Leftrightarrow x < x' < y < y'$	

Figure 2.1: Allen's interval relations and HS modalities.

over $\mathbb{D}$ is an ordered pair $[x,y]$, where $x,y \in D$ and $x < y$ (respectively, $x \leq y$). We adopt the *strict semantics*, which admits strict intervals only. Such a choice conforms to the definition of interval given by Allen in [All83], but it differs from the one by Halpern and Shoham [HS91]. Even though most results can be easily rephrased in *non-strict semantics*, which also admits intervals of the form $[x,x]$ (i.e., *point intervals*), where $x \in D$, the strict one is definitely cleaner for at least two reasons. One the one hand, a number of representation paradoxes arise when the non-strict semantics is adopted, due to the presence of point intervals, as pointed out in [All83]. On the other hand, when point intervals are included there seems to be no intuitive semantics for interval relations that makes them both pairwise disjoint and jointly exhaustive.

We denote by $\mathbb{I}(\mathbb{D})$ the set of (strict) intervals over a linear order $\mathbb{D}$. If we exclude the identity relation, there are 12 different relations between two intervals in a linear order, often called *Allen's relations* [All83]: the six relations R_A (adjacent to), R_L (later than), R_B (begins), R_E (ends), R_D (during), and R_O (overlaps), depicted in Figure 2.1, and their inverses, i.e., $R_{\overline{X}} = (R_X)^{-1}$, for each $X \in \mathcal{A}$, where $\mathcal{A} = \{A, L, B, E, D, O\}$. We associate a universal modality $[X]$ and and existential one $\langle X \rangle$ with each Allen relation R_X. For each $X \in \mathcal{A}$, the *transposes* of the modalities $[X]$ and $\langle X \rangle$ are respectively the modalities $[\overline{X}]$ and $\langle \overline{X} \rangle$, corresponding to the inverse relation $R_{\overline{X}}$ of R_X, and vice versa.

Halpern and Shoham's HS can be viewed as a multi-modal logic whose formulas are built from a finite, non-empty set $\mathcal{AP}$ of *atomic propositions* (also referred to as *propositional letters*), the classical Boolean connectives, and a

modality for each Allen relation:

$$\varphi ::= p \mid \neg\varphi \mid \psi \vee \xi \mid \langle X \rangle \psi \mid \langle \overline{X} \rangle \psi,$$

where $p \in \mathcal{AP}$ and $X \in \mathcal{A}$. The other Boolean connectives and the logical constants, e.g., $\rightarrow$ and $\top$, as well as the universal modalities $[X]$, can be defined in the standard way:

$$\begin{array}{lcllcl} \top & \equiv & p \vee \neg p & \bot & \equiv & \neg\top \\ \psi \wedge \xi & \equiv & \neg(\neg\psi \vee \neg\xi) & \psi \rightarrow \xi & \equiv & \neg\psi \vee \xi \\ [X]\psi & \equiv & \neg\langle X \rangle \neg\psi & [\overline{X}]\psi & \equiv & \neg\langle \overline{X} \rangle \neg\psi \end{array}$$

Given a formula φ, it is always possible to rewrite φ into a semantically equivalent formula φ' where all the negations appear only in front of propositional letters; such transformation is called *negated normal form* and we will, hereafter, assume that each formula is in negated normal form.

As shown in [HS91], the modalities $\langle A \rangle, \langle B \rangle$, and $\langle E \rangle$, along with their transposes are sufficient to express all the other modalities through a formula of polynomial size. In particular, we summarize the procedure:

$$\begin{array}{lcllcl} \langle L \rangle \varphi & \equiv & \langle A \rangle \langle A \rangle \varphi & \langle \overline{L} \rangle \varphi & \equiv & \langle \overline{A} \rangle \langle \overline{A} \rangle \varphi \\ \langle D \rangle \varphi & \equiv & \langle B \rangle \langle E \rangle \varphi \equiv \langle E \rangle \langle B \rangle \varphi & \langle \overline{D} \rangle \varphi & \equiv & \langle \overline{B} \rangle \langle \overline{E} \rangle \varphi \equiv \langle \overline{E} \rangle \langle \overline{B} \rangle \varphi \\ \langle O \rangle \varphi & \equiv & \langle E \rangle \langle \overline{B} \rangle \varphi & \langle \overline{O} \rangle \varphi & \equiv & \langle B \rangle \langle \overline{E} \rangle \varphi \end{array}$$

Without loss of generality, hereafter we restrict ourselves to HS formulas over the set of modalities $\{\langle A \rangle, \langle \overline{A} \rangle, \langle B \rangle, \langle \overline{B} \rangle, \langle E \rangle, \langle \overline{E} \rangle\}$.

The semantics of HS formulas is given in terms of *interval models* $M = \langle \mathbb{D}, V \rangle$, where $\mathbb{D}$ is a linear order and $V : \mathcal{AP} \rightarrow 2^{\mathbb{I}(\mathbb{D})}$ is a *valuation function* which assigns to each atomic proposition $p \in \mathcal{AP}$ the set of intervals $V(p)$ on which p holds. In this work, we are interested in finite structures and thus we restrict our attention to linear orders over finite domains. Any finite linear order $\mathbb{D}$ of size d can be compactly represented by $[d] = \{i \in \mathbb{N} \mid i \leq d\}$. In the following, we will make use of such a representation.

The *truth* of a formula φ on a given interval $[x,y]$ in an interval model M is defined by structural induction on formulas as follows:

$$
\begin{array}{lcl}
M,[x,y] \Vdash p & \Leftrightarrow & [x,y] \in V(p), \text{ for } p \in \mathcal{AP}, \\
M,[x,y] \Vdash \neg\psi & \Leftrightarrow & M,[x,y] \nVdash \psi, \\
M,[x,y] \Vdash \psi \vee \xi & \Leftrightarrow & M,[x,y] \Vdash \psi \text{ or } M,[x,y] \Vdash \xi, \\
M,[x,y] \Vdash \langle X \rangle \psi & \Leftrightarrow & \text{exists } [z,t] \text{ such that } [x,y] R_X [z,t] \text{ and } M,[z,t] \Vdash \psi, \\
M,[x,y] \Vdash \langle \overline{X} \rangle \psi & \Leftrightarrow & \text{exists } [z,t] \text{ such that } [x,y] R_{\overline{X}} [z,t] \text{ and } M,[z,t] \Vdash \psi.
\end{array}
$$

The *modal depth* $md(\varphi)$ of a HS formula φ is recursively defined as:

$$
\begin{array}{lcll}
\varphi = p, & \Rightarrow & md(\varphi) = 0, & \text{where } p \in \mathcal{AP}, \\
\varphi = \neg\psi, & \Rightarrow & md(\varphi) = md(\psi), & \\
\varphi = \psi \vee \xi, & \Rightarrow & md(\varphi) = \max(md(\psi), md(\xi)), & \\
\varphi = \langle X \rangle \psi, & \Rightarrow & md(\varphi) = 1 + md(\psi), & \\
\varphi = \langle \overline{X} \rangle \psi, & \Rightarrow & md(\varphi) = 1 + md(\psi). &
\end{array}
$$

The set of sub-formulas of a given HS formula φ is denoted by $cl(\varphi)$ and is defined as the smallest set satisfying the following conditions:

$$
\begin{array}{lcl}
\varphi \in cl(\varphi), & & \\
\neg\psi \in cl(\varphi) & \Rightarrow & \psi \in cl(\varphi), \\
\langle X \rangle \psi \in cl(\varphi) & \Rightarrow & \psi \in cl(\varphi), \\
\langle \overline{X} \rangle \psi \in cl(\varphi) & \Rightarrow & \psi \in cl(\varphi), \\
\psi \vee \xi \in cl(\varphi) & \Rightarrow & \psi, \xi \in cl(\varphi).
\end{array}
$$

Note that, by definition, $|cl(\varphi)| \geq 1$, and we will intensively make use of such observation. Moreover, $cl(\varphi)$ is also called *closure* of φ.

In the following, we will make use of the *global modality* $[U]$, that allows one to assert a property φ on the entire model. HS is power enough to express such modality as follows:

$$[U]\varphi \stackrel{\text{def}}{\equiv} \varphi \wedge \bigwedge_{X \in \mathcal{A}} ([X]\varphi \wedge [\overline{X}]\varphi).$$

2.3 The model checking problem

The need for reliable hardware and software systems, where failure is unacceptable, is critical. Ensuring the correctness of such systems is a challenging task since many components of a system may interact with each other. *Model checking* is an automatic technique for verifying finite state concurrent systems [CGP99]. Given a Kripke structure M and a temporal logic formula specification φ, the *model checking problem* is the problem of establishing if $M \Vdash \varphi$. That is, it is a decision problem returning *yes*, if $M \Vdash \varphi$, and *no*, otherwise. A nice property of model checkers is that when $M \nVdash \varphi$ they produce a *counter trace* (i.e., a sequence of states) in the model M which represents a *counterexample* to the temporal specification φ. The generation of such traces is an important tool in the design and debugging of systems.

Now, let focus our attention on the HS model checking problem against a finite path (or history). Given a pair $\mathcal{I} = (M, \varphi)$, where M is a finite interval model and φ is a HS formula, the problem of model checking the HS formula φ against a finite path/history M (MC for short) consists in deciding whether $M, [0,1] \Vdash \varphi$. This is also called *initial satisfiability*, since the task is deciding if $M, [0,1] \Vdash \varphi$. In some scenarios, it might be convenient to ask if $M, [x,y] \Vdash \varphi$ for some interval $[x,y]$ (and we will refer to such scenarios in the following). The pair $\mathcal{I} = (M, \varphi)$ is called an *instance* of MC. With a little abuse of notation, for a given instance (M, φ) of MC, we write $(M, \varphi) \in$ MC to indicate that $M, [0,1] \Vdash \varphi$. In such a case, we say that MC applied to (M, φ), denoted by $\text{MC}(M, \varphi)$, returns true. We say that two instances $\mathcal{I}, \mathcal{I}'$ of MC are *equivalent*, denoted by $\mathcal{I} \equiv_{\text{MC}} \mathcal{I}'$ whenever $\text{MC}(\mathcal{I})$ returns true if and only if $\text{MC}(\mathcal{I}')$ does. Finally, we say that two HS formulas φ_1 and φ_2 are *equivalent*, denoted by $\varphi_1 \equiv \varphi_2$, if $(M, \varphi_1) \equiv_{\text{MC}} (M, \varphi_2)$, for every interval model M.

2.4 A timeline-based representation

In this section, we introduce the problem of how interval temporal models are represented. The public available data can be roughly classified as *atemporal* and *temporal data*. In the former case, the data does not have any inherently temporal dimension. In particular, since the data can be viewed as the content of a database, none of the attributes of the *records* (i.e., instances) of a table present any temporal information. In the latter case, the temporal dimension plays a key role when trying to reason about the data. In particular, the cornerstone of temporal data support in SQL:2011 is the ability to define and associate *time periods* (i.e., intervals) with the records of a table [Zem12]. Therefore, we can think at temporal databases when we talk about temporal data. Since we are interested in solving an interval temporal problem, hereafter, we consider only the case when time comes in to play.

The input of classic, point-based model checking consists of a temporal specification φ and a Kripke structure M generally represented by (a suitable encoding of) the set of its states, along with their labels, and the set of its transitions—for the model checking of very huge structures, other kinds of solution are used, e.g., on-the-fly or bounded model checking (see [BCCZ99, BHJ+06]). Classic model checking is infinite by nature: infinitely many, possibly infinite paths, which are finitely encoded in the input structure. On the contrary, in the interval-based setting, in order to model check a finite path/history, that is, a finite labelled structure, one can represent the model simply by specifying the number of points in the model and then listing, for each proposition letter, the set of sub-intervals over which it holds [MdFM+17] (see Figure 2.2).

In formal verification terminology, a Kripke structure is called *labelled transition system*, that is, a directed graph whose nodes are labelled. A point-based formalism suits perfectly with this interpretation of the Kripke structure since a transition from a state to another *implicitly* involves the passage of one time unit (i.e., from time unit t to time unit $t+1$). In the interval-based setting, the time span needs to be *explicitly* represented, that is, viewing the Kripke structure as a directed graph is not always suitable. To this end, we introduce the notion of *timeline* [BSS19, Mol19, SSV19]. A timeline $T = \langle \mathbb{D}, V \rangle$ is just the

n

$p_1 \quad : \quad [0,1], [2,5], [3,4]$

$p_2 \quad : \quad [0,1], [0,3]$

$\ldots$

Figure 2.2: A succint representation of a finite interval model for HS. The first line specifies the size of the history (i.e., number of points in the model); the next lines encode the valuation function V.

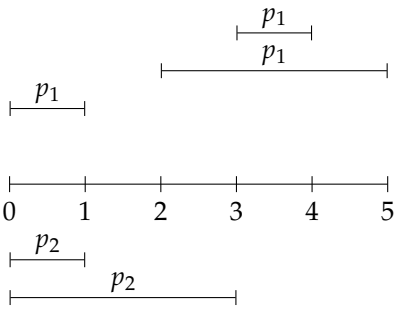


Figure 2.3: A timeline $T = \langle [d], V \rangle$, where $d = 5$ and V is defined over $\mathcal{AP} = \{p_1, p_2\}$ as in Figure 2.2.

interval-based reinterpretation of a Kripke structure $M = \langle \mathbb{D}, V \rangle$ (see Figure 2.3 for the timeline interpretation of the model from Figure 2.2). Hereafter, we will use timeline and finite interval model interchangeably.

The fundamental difference between the two frameworks is that, in the classical model checking problem, frame information, i.e., states and transitions, must be explicitly represented in the input, while, in finite interval models, frame information, i.e., intervals and their relations, is implicit in the size of the size of the temporal domain (this is because relations among intervals are induced by the underlying linear order). As a consequence, while the size of the representation of a Kripke structure is typically polynomial in the number of states and labels, the size of the representation of a finite model may be logarithmic in the number of intervals. As an example, consider the interval model $M = \langle [d], V \rangle$ over $\mathcal{AP} = \{p\}$, where $V(p) = \{[d-1, d]\}$. Its representation consists of the number d (which takes space $O(\log d)$) and the

mapping $p \mapsto \{[d-1, d]\}$ (which takes space $O(\log d)$ as well); we call such model a *sparse model* [MdFM+17].

Before we try to generalize the finite interval temporal model checking, we need to ask ourselves if interval models, as discussed in the preceeding paragraphs, are available. We claim that there are at least two possible sources for interval temporal models. On the one hand, as already observed, one way to represent the temporal information is via temporal databases, since sets of time instants describing the validity of a particular fact in the real world can be often described by an interval or a finite union of intervals [CT05]. At the first glance, one may think that temporal databases are easily obtainable, but this is not always the case since the vast majority of such databases are of private property. On the other hand, if we reason in terms of instances, many public (available) temporal data adopt a *time series* structure, where each variable (e.g., the patient's fever in a medical domain) is a serie of point observations indexed in time order, that is, time-valued data. On the contrary, such temporal data is at one's disposal more frequently than temporal databases. This is because, seeing the temporal data as a collection of time series removes the constraint that such data must live within a database. Thus, we have more freedom when dealing with time series.

A common key point that relate temporal databases and time series data together is that such data needs to be processed to obtain a temporal data set of interval-based models (i.e., timelines). In the recent literature several *temporal abstraction* methods of obtaining timelines from time series have been presented. Höppner in [Höp02] gives a survey on temporal abstraction methods. There exist several abstraction methods, which can be roughly separated into *adimensional*, that is, methods that do not consider the temporal dimension, and *dimensional*, which do consider the temporal dimension. Examples of the first category include: *(i) Equal Width Discretization* (EWD), *(ii) Equal Frequency Discretization* (EFD), and *(iii) k-means clustering*. Examples of dimensional methods include: *(iv) Symbolic Aggregate approXimation* (SAX) [LKWL07], which does not explicitly consider the temporal order of the values and *(v) Persist* [MU05], which maximizes the duration of the resulting time intervals and which explicitly considers the temporal order. Moreover, in [MS15], a classification-driven

discretization method, namely *Temporal Discretization for Classification* (TD4C), is presented. A comparative study on various *pattern languages* (i.e. approaches to represent the temporal interval patterns) is presented in [HP14], where the authors point out the strengths and weaknesses for each of them, based on four well-known problems in the literature: preservation of qualitative relationship, preservation of quantitative durations, concurrency and robustness; many of such languages are Allen-based, but none of them have a logical approach.

A more recent result presenting a general, domain-independent method for transforming time series into timelines can be found in [SSV19]. We briefly recall the methodology. Let $\overline{F}(t) = (f_1(t), \ldots, f_m(t))$ be a *multivariate time series*. Each $f_j(t)$ is referred to as *variable* (or *attribute*, to use the standard terminology of static data). A collection of multivariate time series has the form $\{\overline{F}_1(t), \ldots, \overline{F}_n(t)\}$. Although time series that describe a real-life data may have any codomain (typically, the reals), since a set of such multivariate time series is always finite and extensively described we can always assume that each $f_j(t)$ is a function of the type:

$$f_j : \mathbb{D} \to \mathbb{N},$$

where $\mathbb{D}$ is a finite temporal domain. We devised a method for converting $\overline{F}(t)$ into a timeline T in which every interval is suitably labelled to carry the same information (in abstract form). Time series are often abstracted in different ways with different aims; in some cases, an interval is labelled with a *state* of some variable $f_j(t)$, that is, with the average value of $f_j(t)$ in that intervals; in some other cases, a label represents the *trend* of some variable. In order to generalize and systematize such a labeling process, we introduce the concept of *z-th degree of timeline*, in analogy with the z-th degree of *discrete derivative*. As a matter of fact, states are simply averages of the values of some $f_j(t)$, while trends are averages of the values of $f_j^1(t)$. In general, therefore, one may be interested to abstract a time series at any degree of derivative, to obtain a timelines with different granularities. In the following, we use the symbol $\overline{F}^z(t)$ for $(f_1^z(t), \ldots, f_m^z(t))$.

Since at each degree of derivative the finite domain of the resulting function contains one less point, we denote by $\mathbb{D}^z$ the domains obtained from $\mathbb{D}$ at the

z-th degree of derivative. Fixed a degree z, the abstraction process consists of producing a timeline $T_{\overline{F}^z(t)}$ from $\overline{F}^z(t)$:

$$T_{\overline{F}^z(t)} = \langle \mathbb{D}^z, V \rangle,$$

and we have to specify the valuation function V. To this end, we first consider the mean (denoted by μ_j) and the standard deviation (denoted by σ_j) of the j-th component entire series (at the z-th derivative), and, for a specific interval $[x, y] \in \mathbb{I}(\mathbb{D})$, we define:

$$\mu_j^{xy} = \frac{\Sigma_{x \leq t \leq y} f_j^z(t)}{y - x},$$

that is, the mean of the values of $f_j^z(t)$ between x and y, and we use them to build a set of propositional letters to define the valuation function V. In classical solutions for temporal abstraction labels are often domain-dependent. In order to avoid the use of domain-related knowledge, we introduce two parameters, that is, $l > 1, l \in \mathbb{N}$ (*number of labels*, assumed to be odd) and $k \in [0, 1] \subset \mathbb{R}$ (*displacement*), and define the set of propositional letters:

$$\{L_p^j \mid 1 \leq p \leq l, 1 \leq j \leq m\},$$

and, finally, define $[x, y] \in V(L_p^j)$ iff

$$\begin{cases} \mu_j^{xy} < \mu_j - \lfloor \frac{l}{2} \rfloor k\sigma_j & \text{if } p = 1 \\ \mu_j - (\lceil \frac{l}{2} \rceil - p + 1)k\sigma_j \leq \mu_j^{xy} < \mu_j - (\lceil \frac{l}{2} \rceil - p)k\sigma_j & \text{if } 1 < p < \lceil \frac{l}{2} \rceil \\ \mu_j - k\sigma_j \leq \mu_j^{xy} \leq \mu_j + k\sigma_j & \text{if } p = \lceil \frac{l}{2} \rceil \\ \mu_j + (p - \lceil \frac{l}{2} \rceil)k\sigma_j < \mu_j^{xy} \leq \mu_j + (p - \lceil \frac{l}{2} \rceil + 1)k\sigma_j & \text{if } \lceil \frac{l}{2} \rceil < p < l \\ \mu_j^{xy} > \mu_j + \lfloor \frac{l}{2} \rfloor k\sigma_j & \text{if } p = l \end{cases}$$

So, for example, if $z = 0$, $l = 3$ and $k = 0.5$, then L_1 (respectively, L_2, L_3) can be read as *low* (respectively, *average*, *high*), and an interval $[x, y]$ is labelled with *low* if its mean value is less than the mean value of the entire series (on

Algorithm 1: Algorithm ***Abstract***.

input : A collection multivariate time series $\mathcal{F}$.
input : The degree z.
input : The number of labels l.
input : The displacement constant k.
output: A collection of multivariate timelines $\mathcal{T}$.

```
1 T ← ∅
2 for i = 1 to n do
3 |   T_i ← Abs(F̄_i, z, l, k)
4 |   T ← T ∪ {T_i}
5 end
6 return T
```

the same component) minus half of its standard deviation. As another example, if $z = 1$, $l = 3$, and $k = 0.25$, then an interval $[x, y]$ is labelled with *increasing* (corresponding to L_3 on the first derivative, that is, the series of the trends) if the mean value of the differences in $[x, y]$ exceeds the mean value of all differences plus one fourth of the standard deviations of all differences. In this way, we can temporally abstract any multivariate time series at any level of derivative, so that rules can be discovered that link the states, or the trends, or the accelerations, and so on, in a consistent, simple, and general way.

Given a time series $\overline{F}(t)$, we say that the *abstracted z-th degree timeline* $T_{\overline{F}^z(t)}$, with l labels and displacement k is:

$$T_{\overline{F}^z(t)} = \boldsymbol{Abs}(\overline{F}(t), z, l, k),$$

where ***Abs*** is a procedure that applies the above labelling strategy. Given a set of n time series $\mathcal{F} = \{\overline{F}_1(t), \ldots, \overline{F}_n(t)\}$, we convert it into a set of n timelines $\mathcal{T} = \{T_1, \ldots, T_n\}$ by simply applying the Algorithm ***Abstract*** depicted in Algorithm 1. We will silently assume that such abstraction method is used proving a collection of (finite) interval-based models.

Most available temporal data has the form of multivariate time series spread over *multiple instances*. Now, by way of example, suppose having one such type of temporal data set $\mathcal{F} = \{\overline{F}_1(t), \ldots, \overline{F}_n(t)\}$, where each $\overline{F}_i(t)$ is a multivariate time series carrying hospitalization information about a particular patient i

which has been monitored for some time period (e.g., one year observation period). In particular, suppose that each $\overline{F}_i(t) = (fever(t), head(t))$ is a (temporal) time series having two variables representing the following symptoms: *i)* fever evolution ($fever$), and *ii)* headache evolution ($head$). Calling algorithm ***Abstract*** on $\mathcal{F}$ (with a suitable set of parameters) produces a collection of timelines $\mathcal{T} = \{T_1, \ldots, T_n\}$ (recall Figure 1.1) on which interesting properties written in HS can be verified (i.e., model checked). To this end, we need an algorithmic interval-based model checking procedure to effectively solve such problem.

2.5 The model checking algorithm

A polynomial-time algorithm for solving the finite interval temporal model checking problem can be found in [MdFM+17]. Given an instance $\mathcal{I} = (M, \varphi)$, such algorithm is the generalization of the Clarke and Emerson model checking algorithm [CGP99] to the interval setting. In particular, such procedure labels each interval of M with the set of sub-formulas of φ which are true on it.

Recall the sparse model $M = \langle [d], V \rangle$ over $\mathcal{AP} = \{p\}$, where $V(p) = \{[d-1, d]\}$. Checking the formula $\langle A \rangle \langle A \rangle p$ would require to label all intervals $[x, d-1]$, with $1 \leq x \leq d-2$, whose number is linear in d, and thus exponential in the size of the representation of M. To circumvent this problem, in [MdFM+17] a preprocessing procedure that transforms the input instance $\mathcal{I} = (M, \varphi)$ into an equivalent one $\mathcal{I}' = (M', \varphi)$ (i.e., $\mathcal{I} \equiv_{MC} \mathcal{I}'$) is also presented. Such subroutine is at most quadratic in the size of the input $\mathcal{I}$, and thus polynomial. Given $\mathcal{I}' = (M', \varphi)$, for each sub-formula ψ of φ, let $L(\psi)$ denote the set of intervals of M' where ψ holds. Calling Algorithm ***FiniteMC*** on $\mathcal{I}' = (M', \varphi)$, depicted in Algorithm 2, solves the finite interval temporal model checking problem in time at most $O(|\mathcal{I}'|^7)$ [MdFM+17], where $|\mathcal{I}'|$ is the size of the (non-sparse) input $\mathcal{I}'$. In particular, in [MdFM+17], the **if** condition at line 22 of Algorithm 2 requires $[0,1] \in L(\varphi)$ (instead of $|L(\varphi)| \neq 0$), and, in such case, we say that $M, [0,1] \Vdash \varphi$, for $[0,1] \in \mathbb{I}([d])$. In general, we require that $M, [x,y] \Vdash \varphi$, for some interval $[x,y] \in \mathbb{I}([d])$, which is the case of Algorithm 2.

Hereafter, we will (reasonably) assume that the models are non-sparse, and

Algorithm 2: Algorithm $\boldsymbol{FiniteMC}$.

input : A non-sparse model $M = \langle [d], V \rangle$.
input : A HS formula φ.
output: `true` if $\exists [x,y] \in \mathbb{I}([d])$ such that $M, [x,y] \Vdash \varphi$; `false` otherwise.

foreach $\psi \in cl(\varphi)$ **do**
  if $\psi = p$, *with* $p \in \mathcal{AP}$ **then** $L(\psi) = V(p)$
  else $L(\psi) = \emptyset$
end
foreach $\psi \in cl(\varphi)$ *(ordered by increasing size)* **do**
  if $\psi = \neg\xi$ **then** $L(\psi) = \mathbb{I}([d]) \setminus L(\xi)$
  else if $\psi = \xi \vee \tau$ **then** $L(\psi) = L(\xi) \cup L(\tau)$
  else if $\psi = \langle A \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z < x$ **do** $L(\psi) = L(\psi) \cup \{[z,x]\}$
  else if $\psi = \langle B \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z > y$ **do** $L(\psi) = L(\psi) \cup \{[x,z]\}$
  else if $\psi = \langle E \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z < x$ **do** $L(\psi) = L(\psi) \cup \{[z,y]\}$
  else if $\psi = \langle \overline{A} \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z > y$ **do** $L(\psi) = L(\psi) \cup \{[y,z]\}$
  else if $\psi = \langle \overline{B} \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z < y$ **do** $L(\psi) = L(\psi) \cup \{[x,z]\}$
  else if $\psi = \langle \overline{E} \rangle \xi$ **then**
    for $[x,y] \in L(\xi)$ **and** $z > x$ **do** $L(\psi) = L(\psi) \cup \{[z,y]\}$
  end
end
if $|L(\varphi)| \neq 0$ **then return** `true`
else return `false`

thus avoiding any further discussion on this topic.

Chapter 3

A generalization of the Finite Interval Temporal Model Checking

In this chapter, we introduce the multiple formulas and multiple models finite interval-based model checking problem, which is the aim of this thesis, along with the naive (algorithmic) approach to solve it. Then, we present two applications, where the presented methodology gives a natural solution to both. In particular, the naive method can be used in both applications; nevertheless, in real world scenarios, where the presented applications are the primitive tool, the naive approach is not always adequate, and, therefore, we point out its weaknesses (that we improve in the next chapter).

3.1 Multiple formulas and multiple models finite interval-based model checking problem

Consider a collection of n HS formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ and a collection of m finite (interval) models $\mathcal{M} = \{M_1, \ldots, M_m\}$. We define the problem of *multiple formulas and multiple models finite interval-based model checking*, MMMC for short, the problem of checking, for each $1 \leq i \leq n$ and for each $1 \leq j \leq m$, if there

Algorithm 3: Algorithm $\boldsymbol{NaiveMMMC}$.

input : A collection of m timelines $\mathcal{M} = \{M_1, \ldots, M_m\}$.
input : A collection of n HS formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$.

```
1 foreach φ_i ∈ Φ do
2     foreach M_j ∈ M do
3         if FiniteMC(M_j, φ_i) then
4             Do something.                          ◁ O(1)
5         else
6             Do something else.                     ◁ O(1)
7         end
8     end
9 end
```

exists some interval $[x, y]$ such that $M_j, [x, y] \Vdash \varphi_i$. That is, the generalization is two folded:

i) a collection of formulas against a single formula, and

ii) a collection of models against a single model.

Note that such problem can be (naively) solved via $n \cdot m$ invocations of Algorithm 2 checking whether or not $M_j, [x, y] \Vdash \varphi_i$, for some interval $[x, y]$, and for each i and j. In Algorithm 3 we summarize this (naive) algorithm which we call $\boldsymbol{NaiveMMMC}$ algorithm.

3.2 Application 1: Temporal query evaluation

Query evaluation and constraint checking in relational databases can be expressed in terms of model checking problems [Var05]. Time intervals are usually used to represent temporal information in temporal databases. In particular, the cornerstone of SQL:2011 [Zem12] is the ability to define and associate *time periods* (i.e., intervals) with records in a table. Period predicates are functionally similar (but not identical) to the Allen's interval operators [CT05]. Moreover, temporal databases can be represented as finite interval models [MdFM+17].

Suppose to have a temporal database containing hospitalization historical data, where each table of the database has its own stored records about patients

(see Figure 3.1). As in the classical (non-temporal) setting, each patient is uniquely identified by his/her *primary key* (i.e., the underlined attribute in Subfigure 3.1a), but, as pointed out in [Zem12], in the temporal case, such primary keys must also encompass *non-overlapping* period information (i.e., the underlined attributes form the primary key in Subfigures 3.1b and 3.1c). A single patient may have various time periods hospitalization records within the same table. In particular, a patient is uniquely identified by its ID, and when referring to a patient we refer to its ID. It is crucial to observe the fact that the collection of patients can be interpreted as a collection of finite interval models (i.e., timelines). Generally speaking, temporal information about a patient is spread over various tables or a patient can have different (types of) entries in the same table. Thus, a patient can be interpreted as a multivariate timeline (see Figure 3.2), where the variability is dictated by the tables themselves. In such scenarios, non-trivial relations arise, and a proper treatment for such temporal data is required.

Since a collection of temporal instances (e.g., the temporal records of the patients in our example) can be easily interpreted as a collection of finite interval-based models, the MMMC approach fits perfectly within such scenarios, where the problem of temporal query evaluation may be formalized by means of a generalized finite interval-based model checking problem. For instance, when evaluating a temporal database (in terms of temporal queries or temporal constraint checking), the database might be expressed as a set of finite interval models and the model checking algorithm can be applied to each interval model against the temporal formula. In particular, given a temporal database, the collection of m temporal instances is translated into a collection of finite interval models $\mathcal{M} = \{M_1, \ldots, M_m\}$, and the collection of formulas Φ is compound of the temporal formulas to be evaluated $\{\varphi_1, \ldots, \varphi_n\}$ which are the HS-encoding of the n queries to be evaluated. Calling algorithm $\boldsymbol{NaiveMMMC}$, depicted in Algorithm 3, on input $\mathcal{M}$ and Φ solves the interval temporal query evaluation problem.

This naive algorithmic approach to solve such an interesting problem suffers in terms of computational complexity. For example, given Φ and $\mathcal{M}$, it is important to stress the fact that if $\varphi_1, \varphi_2 \in \Phi$ such that $\varphi_1 = \varphi_2$ (i.e., syntactically

Patient			
ID	Name	Age	Sex
1	Alice	24	Female
2	Bob	29	Male
3	Charlie	18	Male
...			

(a) Non-temporal table.

Fever (*fever*)		
ID	Start	End
1	1	5
2	0	4
3	2	4
3	5	7
...		

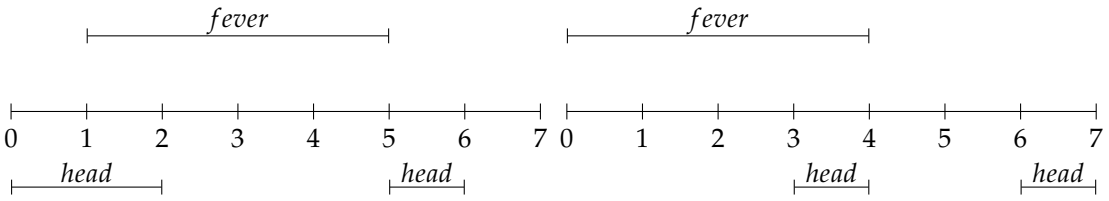
(b) Fever temporal table.

Headache (*head*)		
ID	Start	End
1	0	2
1	5	6
2	3	4
2	6	7
...		

(c) Headache temporal table.

Figure 3.1: An example of a temporal database containing a non-temporal table (at the top) along with temporal tables (at the bottom) representing the evolution of the symptoms.

are the same formula), then Algorithm $\boldsymbol{NaiveMMMC}$ model checks *again* the same formula against $\mathcal{M}$; nonetheless, φ_1 (or φ_2, depending on the order) has been already checked. The reader which is practical with databases can immediately observe that such situations arise very often. In particular, some queries which have been translated into HS-formulas may differ slightly, that is, may have almost identical structure. The intuition is that different formulas can share the same sub-structure and calling Algorithm $\boldsymbol{NaiveMMMC}$ on them requires passing through such sub-structure, verifying the same shared sub-formula, a number of times that is proportional to how many formulas (of Φ) share the same sub-structure. For example, consider calling Algorithm $\boldsymbol{NaiveMMMC}$ on $\mathcal{M}$ and $\Phi = \{\varphi_1, \varphi_2\}$, where $\varphi_2 = \varphi_1 \wedge [A]p$; that is, φ_2 is φ_1 conjuncted with $[A]p$, for some $p \in \mathcal{AP}$. In this case, the procedure takes time proportional to the entire set of formulas Φ. A better achievement



(a) Patient with ID 1. (b) Patient with ID 2.

Figure 3.2: Timeline interpretation of the database example from Figure 3.1 for patients with ID 1 and 2. Each patient is represented by a timeline.

can be accomplished if we are willing to store side information, where, in the aforementioned example, the overall execution time can be halved. This intuition is formalized in the next chapter.

3.3 Application 2: Extraction of a temporal classifier

When we are given a set of finite (interval) temporal models, we can interpret such models as a temporal data set. This perspective allows us to imagine problems from the *machine learning* world [Mit97, Mur12]. The most representative classical machine learning problems are *supervised learning* and *unsupervised learning*. In the former case, the input is a set of already labelled instances (e.g., labelled timelines) from a set of countable labels and the goal is to learn how such instances are related to the labels so that when a new unlabelled instance is given as input, the algorithm should (statistically) label such instance based on the learned information/knowledge. Within this class of problems, one of the most studied problems is the *classification problem*, where each instance is classified with a *class* from a set of *classes* $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$. Moreover, when $l = 2$, the problem is called *binary classification*. In the latter case of unsupervised learning, the instances are not labelled and the goal is to identify patterns among such instances in order to find possible labels or possible association rules (see, e.g., [BCG+18, SSV19]), depending on the application-domain. Examples of algorithms for such class of problems are, among others, Clustering, Expectation-Maximization (EM), Principal Component Analysis (PCA), Generative Adversarial Networks (GANs).

Formal theories of mathematics and computer science, like temporal logics,

may seem unrelated with state-of-the-art algorithms borrowed from the machine learning zoo, which may (roughly) be seen as less theoretic counterparts. Recent results [BSS19, SSV19] have demystified such aspect, where there is a clear non-empty intersection between the theoretical computer science world (i.e., interval temporal logics) and the machine learning world (i.e., temporal tree-based supervised classifiers [BSS19], and temporal unsupervised rules discovery [SSV19]). Moreover, as mentioned, we use the abstraction algorithm presented in [SSV19], which turns a point-based time series, or a temporal database, into an interval-based timeline. Such adaptations [BSS19, SSV19] of the learning process to the case where the input data has a timeline representation are witnesses for the interval-based learning. In this subsection, we introduce a new witness where there is a strong evidence that an interval-based formalism must be used.

Among all methods that might be interesting to study there is also the problem of extracting a *rule-based classifier*, where the learned schemata is a set of Horn-like propositional rules which produces a new, unknown before, theory. That is, given a collection of labelled timelines $\mathcal{M} = \{M_1, \ldots, M_m\}$ each with its own class from the set of classes $\mathcal{C} = \{0, 1\}$, the goal is to learn the most (statistically) fitted rules $\Gamma = \{\rho_1, \ldots, \rho_r\}$ to $\mathcal{M}$, based on some *fitness function* [Mit97, Mur12]. Intuitively, a fitness function can be some indicator function which tells us the *goodness* of the classifier Γ. It is important to note that, in general, different fitness functions produce different models (e.g., classifiers) since the learning process is driven by such function. Another way of seeing such functions is by means of *cost/gain functions*, which are the bread and butter of the *Operational Research* (OR for short) world. In particular, using the OR nomenclature, such functions are called *objectives*.

A classification model is induced on a given structure: *i) function-based* classification models, where the underlying theory is modelled as a function whose output can be used to determine the class, *ii) tree-based* classification models, where the underlying theory is described as a tree, and *iii) rule-based* classification models, where the underlying theory is a set of disjunctive rules.

A propositional rule has the form:

$$\rho : p_1 \wedge p_2 \wedge \ldots \wedge p_s \rightarrow c, \tag{3.1}$$

where $A(\rho) = p_1 \wedge p_2 \wedge \ldots \wedge p_s$ is the *antecedent* having s conditions and $C(\rho) = c$ is the *consequent*, and a rule-based classifier model has the form:

$$\Gamma = \begin{cases} \rho_1 : p_1^1 \wedge p_2^1 \wedge \ldots \wedge p_{s_1}^1 \rightarrow c_1 \\ \rho_2 : p_1^2 \wedge p_2^2 \wedge \ldots \wedge p_{s_2}^2 \rightarrow c_2 \\ \ldots \\ \rho_r : p_1^r \wedge p_2^r \wedge \ldots \wedge p_{s_r}^r \rightarrow c_r \end{cases} \tag{3.2}$$

The i-th rule has an antecedent with s_i conditions and, if fired, it classifies an instance with class $c_i \in \mathcal{C}$. It is important to stress the fact that the implications are not logical, but statistical. Moreover, when the consequents are not defined the rules are called *association rules*, instead of *classification rules*.

Given an instance $M \in \mathcal{M}$, let $\Gamma(M)$ denote the class that Γ assigns to M. In general, more than one rule $\rho_i \in \Gamma$ may hold on M and, therefore, we need some *firing policy* (e.g., specificity, recency, etc.) in order to break potential ties. Since in supervised classification each instance $M \in \mathcal{M}$ has already been classified with a class, let $C(M)$ denote such class. To reader might be puzzled by the definition of $C(M)$ and $C(\rho)$; in the former case, we are asking for the class of the (pre)-labelled timeline M, and in the latter case, we are asking for the consequent of a rule ρ. To statistically evaluate the goodness of the classifiers, we need to compare $C(M)$ and $\Gamma(M)$ for each $M \in \mathcal{M}$. For a binary classification problem where $\mathcal{C} = \{0, 1\}$, four possibilities arise. In particular, we say that M is a *i) true positive* (TP for short) if $C(M) = \Gamma(M) = 1$, *ii) true negative* (TN) if $C(M) = \Gamma(M) = 0$, *iii) false positive* (FP) if $\Gamma(M) = 1$, but $C(M) = 0$, and *iv) false negative* (FN) if $\Gamma(M) = 0$, but $C(M) = 1$. These indicators can be combined in various ways, giving rise to a generic *performance function*. A classic performance function is the *accuracy* defined as:

$$Acc(\Gamma) = \frac{TP + TN}{TP + TN + FP + FN}. \tag{3.3}$$

A pictorial representation of these four indicators can be viewed in Figure 3.3. Such representation is canonical in the field of statistical classification also called *confusion matrix*, in the case of supervised learning, or *matching matrix*, in the

	$C(M) = 1$	$C(M) = 0$
$\Gamma(M) = 1$	TP	FP (type I error)
$\Gamma(M) = 0$	FN (type II error)	TN

Figure 3.3: Confusion matrix for a binary classification problem.

case of unsupervised learning. For the sake of the presentation, the accuracy, from Equation (3.3), is the sum of the diagonal entries divided by the sum of all entries of the confusion matrix.

Inspired by the fact that other classical machine learning algorithms have been generalized to the (interval) temporal case [BSS19, SSV19], the problem of extracting a temporal classifier can be also generalized. The generalization to the interval temporal case is simple. In particular, an *interval temporal rule* has the form:

$$\rho : p_1 \wedge \mathcal{O}(p_2 \wedge \mathcal{O}(\ldots \wedge \mathcal{O}p_s)\ldots) \rightarrow c, \tag{3.4}$$

where p_i is a propositional letter or the constant $\top$ and $\mathcal{O}$ is either an existential or an universal Allen relation, i.e., $\langle X \rangle$ or $[X]$ where $X \in \mathcal{A}$ (see Section 2.2), or it is $[=]$ representing the identity between intervals. Note that when using only $\{[=]\}$ for $\mathcal{O}$, then the (non-temporal) formulas following the pattern from Equation (3.4) is the propositional sub-fragment of HS and, thus, generalizes Equation (3.1) in terms of generated formulas. A *temporal rule-based classifier* can be defined analogously to Equation (3.2):

$$\Gamma = \begin{cases} \rho_1 : p_1^1 \wedge \mathcal{O}(p_2^1 \wedge \mathcal{O}(\ldots \wedge \mathcal{O}p_s^1)\ldots) \rightarrow c_1 \\ \rho_2 : p_1^2 \wedge \mathcal{O}(p_2^2 \wedge \mathcal{O}(\ldots \wedge \mathcal{O}p_s^2)\ldots) \rightarrow c_2 \\ \ldots \\ \rho_r : p_1^r \wedge \mathcal{O}(p_2^r \wedge \mathcal{O}(\ldots \wedge \mathcal{O}p_s^r)\ldots) \rightarrow c_r \end{cases} \tag{3.5}$$

The bad news is that inducing a rule-based classifier is usually defined as a *multi-objective non-linear optimization* problem (see, e.g., [CS04]), in contrast with a tree-based approach where the procedure is deterministic [BSS19]. In a multi-objective problem the goal is to minimize/maximize (i.e., optimize) a set of objectives. In addition to the objectives, such problems encompass a

(possibly empty) set of constraints. In the classical setting, if the constraints are linear, then the problem is called *linear optimization problem*; on the other hand, the most challenging setting has constraints which are non-linear, namely, *non-linear optimization problem*. The combination between multi-objective and non-linear problems produces multi-objective non-linear optimization problems. *Multi-objective evolutionary algorithms* are known to be particularly suitable to perform multi-objective optimization, as they search for optimal solutions in parallel [Deb01, DPAM02]. Generally speaking, each algorithm of such class of algorithms can be viewed as a big loop where the first population $\mathcal{P}$ of solutions is created at random, say $\mathcal{P} = \{\Gamma_1, \ldots, \Gamma_P\}$. At each iteration, a new population is generated by using four main operations: *i) selection*, *ii) crossover*, *iii) mutation*, *iv) elite-preservation*. After that, the members of the population are evaluated, the selection operator chooses the best solutions with a large probability to fill an intermediate mating pool. The crossover operator randomly picks two or more solutions (parents) from the mating pool and create one or more solutions (offsprings) by exchanging information among the parent solutions. The mutation operator randomly perturbates each child, created via crossover. The elitism operator combines the old population with the newly created population and chooses to keep the best solutions from the combined population (i.e., *survival of the fittest*). In the recent literature [JMMP+18] such problem has been approached even in a more general way, in which the object language is *fuzzy* propositional language. That is, the semantics of the propositions are not fixed to either true or false, but it varies in order to improve the classification performances. A more formal treatment is required to mathematically present a possible encoding of a HS-based temporal classifier Γ (see Equation (3.5)) by means of a multi-objective non-linear optimization problem, but such treatment is beyond the aim of this thesis. Nevertheless, the reader should refer to the references of this paragraph, if needed.

The good news is that evaluating the goodness (e.g., accuracy) of a HS-based classifier is just the adaptation of Algorithm 3. In particular, to obtain the four indicators, as depicted in Figure 3.3, in the case of a binary classification, we can call the $\boldsymbol{NaiveMMMC}$ algorithm on $\mathcal{M}$ and Γ, and, for each $M \in \mathcal{M}$, record the class induced by Γ (i.e., $\Gamma(M)$) to ultimately compute the accuracy.

A crucial observation is that solving a multi-objective non-linear optimization problem through a multi-objective evolutionary algorithms involves, as mentioned, the evaluation of random classifiers at the beginning of the evolution (e.g., the initial randomly generated population $\mathcal{P}$). Then, as the evolution goes on (i.e., iterating the evolutionary algorithm), the elements from the population will eventually have similar structures and, thus, similar formulas/sub-formulas because the search space is bounded by the objectives. The intuition is that calling algorithm $\boldsymbol{NaiveMMMC}$ at the beginning of the evolution will not suffer too much in terms of time complexity, but as the evolution goes on using such (naive) procedure will suffer since a huge fragment of the formulas of the population $\mathcal{P}$ have been already checked. The need of a less naive procedure naturally arises in this case, as in the (interval) temporal query evaluation case of the previous section.

Chapter 4

An alternative solution to the MMMC Problem

In this chapter, we effectively and efficiently solve the MMMC problem. First, we give the running time for the simple, naive case. Then, we introduce a data structure that we use to outperform the naive case in terms of time complexity. Finally, thanks to the presented data structure and thanks to some basic probability theory definitions, we give the expected time case analysis that hardly relies on such data structure.

4.1 Why is it a problem?

In the previous chapter, the usage of the MMMC approach has naturally emerged within the context of (temporal) databases and (temporal) machine learning. In [MdFM+17] the authors have pointed out some issues when solving the finite interval-based model checking for a single instance $\mathcal{I}$. For instance, as already mentioned, dealing with sparse or non-sparse models is beyond the goal of this thesis, nevertheless we can reasonably assume that our models are not sparse; thus, avoiding to use the sub-routine presented in [MdFM+17] that transforms a sparse model into an *equivalent* non-sparse

model. Moreover, model checking for full HS under the *homogeneity* assumption (i.e., a proposition letter holds on an interval if and only if it holds on every sub-interval) is EXPSPACE-hard [BMM+16]. Furthermore, the complexity of model checking HS fragments have been also studied (e.g., see [BMM+16]). In particular, the complexity for such fragments go from coNP-complete to non-elementary. Therefore, solving the MMMC problem is not a trivial task since it involves solving the simpler problem and the aim of this chapter is to go towards a (good) general solution. The reader should refer to Molinari's PhD thesis [Mol19] for a systematic, organic and in-depth study on the interval temporal logics model checking problem.

4.2 Complexity analysis of the naive solution

In this section we briefly analyse the time complexity of Algorithm 3.

Recall from [MdFM+17] that Algorithm 2 runs in time $O(|\mathcal{I}|^7)$ for some input instance $\mathcal{I} = (M, \varphi)$. In particular, the **for** loop at line 5 of Algorithm 2 is executed at most $|\mathcal{I}|$ since $|\varphi| \leq |\mathcal{I}|$, and the number of points of $[d]$ is bounded by $O(|\mathcal{I}|^3)$; thus, $|\mathbb{I}([d])|$ is bounded by $O(|\mathcal{I}|^6)$ [MdFM+17]. Moreover, $O(|\mathcal{I}|^6)$ gives an upper bound to $L(\psi)$, for each sub-formula ψ of φ, allowing us to conclude that Algorithm 2 runs in in $O(|\mathcal{I}|^7)$ time. We leave to the reader the proof of the following proposition.

Proposition 4.2.1. *For a set of n formulas Φ and a set of m models $\mathcal{M}$, Algorithm 3 runs in time $O(nm|\mathcal{I}|^7)$, where $|\mathcal{I}|$ is the size of an input instance $\mathcal{I} = (M, \varphi)$.*

4.3 A data structure to improve the naive solution

A rule of thumb to try to optimize the running time complexity of an algorithm is by using an auxiliary data structure that supports its computation. Our proposal to optimize the running time complexity of Algorithm 3 is to use an adapted *hashtable* [CLRS09]. Moreover, using the *universal hashing* methodology [CLRS09, MR95] each Insert, Delete and Search operation takes $O(1)$ expected time. In universal hashing a suitable *universal hash (functions) family* is (for example, algebraically) build and, then, a randomly chosen hash function

from such family is used to drive any operation on the table. We will silently assume that such methodology is used.

A hashtable is an associative map that maps *keys* to *values*. The values can be any secondary data structure (e.g., linked list, set). We make the following reasonable assumptions in order to present the idea. First, we consider formulas defined as in Equation (3.4), where each formula can be seen as a binary tree. In particular, for each temporal formula φ, we fix the modal depth $md(\varphi)$. Since the formulas are finite and the modal depth is fixed, we can easily compute the possible *universe* of formulas $\mathcal{U}$ that can be built using the pattern from Equation (3.4). The size of the universe $\mathcal{U}$ gives an upper bound to the total number of formulas that can be given as input (to be model checked), since any other formula (following the same pattern) will fall inside $\mathcal{U}$ (for the pigeonhole principle). Second, we also assume that the set of models $\mathcal{M}$ to be checked is also fixed. Third, each $M \in \mathcal{M}$ is defined over the same finite domain $[d]$, and, therefore, the set of all possible intervals is $\mathbb{I}([d])$; this will allow us to give an upper bound to the size of each model.

We depict in Figure 4.1 the proposed adapted hashtable $\mathcal{H}$ that maps each $\varphi_i \in \mathcal{U}$ to a secondary hashtable which maps each $M_j \in \mathcal{M}$ to a subset of $\mathbb{I}([d])$; in such case, we say that $\mathcal{H}$ has *type* $\mathcal{U} \mapsto (\mathcal{M} \mapsto \mathbb{I}(d))$. We refer to the *outermost* hashtable (i.e., the left hand side table of Figure 4.1) that maps keys from $\mathcal{U}$ to values of $\mathcal{M} \mapsto \mathbb{I}([d])$ as $\mathcal{H}_{\mathcal{U}}$. Let $|\mathcal{H}|$ denote the *real size* of the adapted hashtable, and let $|\mathcal{H}_{\mathcal{U}}|$ denote the number of *keys* of $\mathcal{H}_{\mathcal{U}}$. We anticipate the fact that the table $\mathcal{H}$ will grow at each iteration, thanks to the algorithm that we will present at the end of this chapter, and we need to know the effective size of $\mathcal{H}$ at each iteration. To this end, let $|\mathcal{H}^i|$ denote the real size of $\mathcal{H}$ at the i-th iteration, and let $|\mathcal{H}^i_{\mathcal{U}}|$ be defined in a similar manner. We can imagine $\mathcal{H}$ as a *bidimensional* data structure, where the first dimension represents the temporal formula $\varphi_i \in \mathcal{U}$ and the second dimension represents the finite interval model $M_j \in \mathcal{M}$. Such interpretation allows us to *array-like index* elements of $\mathcal{H}$. In particular, given $\varphi_i \in \mathcal{U}$ and $M_j \in \mathcal{M}$, we say that $\mathcal{H}[\varphi_i][M_j]$ is the set of intervals $\{[x, y] \in \mathbb{I}([d]) \mid M_j, [x, y] \Vdash \varphi_i\}$. As mentioned, accessing an element of $\mathcal{H}$ (i.e., $\mathcal{H}[\varphi_i][M_j]$) takes $O(1)$ expected running time. Moreover, when asking whether $\varphi_i \in \mathcal{H}$ or not, for some formula $\varphi_i \in \mathcal{U}$, we

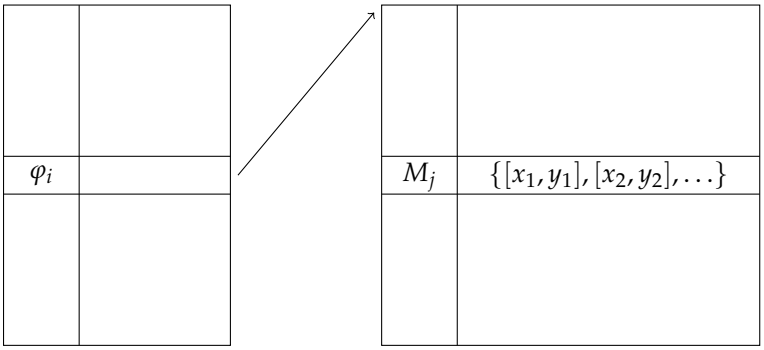


Figure 4.1: Adapted hashtable of type $\mathcal{U} \mapsto (\mathcal{M} \mapsto \mathbb{I}([d]))$.

are asking if there exists at least one $M_j \in \mathcal{M}$ such that $\mathcal{H}[\varphi_i][M_j]$ is defined, meaning that we have previously spent some modest time checking φ_i; this operation also takes $O(1)$ expected running time.

4.4 Expected time complexity analysis

In this section we exploit the usage of the adapted hashtable in solving the MMMC problem. Using a supporting data structure reduces the overall needed running time, and the goal is to give an upper bound to the overall expected running time. The intent of such data structure is to store as much as possible side information regarding the shared parts among formulas. In particular, instead of blindly verifying a particular formula, first, we can check how many sub-formulas have been already checked previously and, then, check only those that have not been already verified. To this end, such process can be probabilistically modelled and we now present such formalism.

4.4.1 Hypergeometric distribution

In this subsection we assume that the reader is familiar with basic probability theory. In particular, we assume that the reader knows the definition of *random variables* and *probability distributions* (along with their properties). The unfamiliar reader should refer, for example, to [EFHP10].

Suppose a set or a population of objects of size N consists of K objects of one type (e.g., having a certain attribute) and the remaining $N - K$ of another type

(e.g., not having that attribute). Suppose n objects are selected either *i)* one after another without replacement from the set of N objects, or *ii)* all at once without replacement from the set of N objects. If X is a (discrete) random variable that counts the number of objects in the sample of size n that have that specific attribute, then X has the *Hypergeometric distribution* with parameters N, K and n: $X \sim HG(N, K, n)$. The *probability (mass) function* of this random variable is:

$$\boldsymbol{P}[X = k] = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}. \tag{4.1}$$

Moreover, the mean or *expected value* of X is:

$$\mathbb{E}[X] = n\frac{K}{N}. \tag{4.2}$$

That is, the mean $\mathbb{E}[X]$ is the number of objects out of K in a sample of size n.

4.4.2 Complexity

We want to use the Hypergeometric distribution (and, in particular, its expected value) to give an expected time complexity result.

We use the universe $\mathcal{U}$, that follows the pattern from Equation (3.4), as an *urn* containing N objects; that is, $|\mathcal{U}| = N$. Given a formula $\varphi \in \mathcal{U}$, since the pattern for $\mathcal{U}$ is fixed, it is easy to observe that also $|cl(\varphi)|$ is fixed. Therefore, let $|cl(\varphi)| = K$; that is, K objects out of N have a specific property (i.e., each $\psi \in cl(\varphi)$ is a sub-formula of φ), and $N - K$ objects out of N do not have that property. Note that, in this simplified scenario, K is the same for all $\varphi \in \mathcal{U}$. Now, suppose sampling n formulas from $\mathcal{U}$ without replacement. Each sampled formula φ is model checked against the (fixed) set of models $\mathcal{M} = \{M_1, \ldots, M_m\}$, and we store such information in (the proposed data structure) $\mathcal{H}$. After the n-th sampled formula φ we can ask how many sub-formulas (out of K) of φ have been already sampled from $\mathcal{U}$, thanks to the expected value from Equation (4.2). This is a crucial observation because if we know how many sub-formulas (of a given formula) have been already sampled (i.e., model checked) in expected value, then the **for** loop at line 5 of Algorithm 2 can be executed fewer times than K meaning that it will be very likely that the

Algorithm 4: Algorithm $\boldsymbol{StepMC}$.

input : A timeline $M = \langle [d], V \rangle$.
input : A temporal formula ψ.
input : An adapted hashtable $\mathcal{H}$ of type $\mathcal{U} \mapsto (\mathcal{M} \mapsto \mathbb{I}([d]))$.
output: The set of intervals $\{[x,y] \in \mathbb{I}([d]) \mid M, [x,y] \Vdash \psi\}$.

1 $L(\psi) \leftarrow \emptyset$
2 **if** $\psi = p$, *with* $p \in \mathcal{AP}$ **then return** $V(p)$
3 **else if** $\psi = \neg\xi$ **then**
4 | **return** $\mathbb{I}([d]) \setminus \mathcal{H}[\xi][M]$
5 **else if** $\psi = \xi \vee \tau$ **then**
6 | **return** $\mathcal{H}[\xi][M] \cup \mathcal{H}[\tau][M]$
7 **else if** $\psi = \langle A \rangle \xi$ **then**
8 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z < x$ **do** $L(\psi) = L(\psi) \cup \{[z,x]\}$
9 | **return** $L(\psi)$
10 **else if** $\psi = \langle B \rangle \xi$ **then**
11 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z > y$ **do** $L(\psi) = L(\psi) \cup \{[x,z]\}$
12 | **return** $L(\psi)$
13 **else if** $\psi = \langle E \rangle \xi$ **then**
14 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z < x$ **do** $L(\psi) = L(\psi) \cup \{[z,y]\}$
15 | **return** $L(\psi)$
16 **else if** $\psi = \langle \overline{A} \rangle \xi$ **then**
17 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z > y$ **do** $L(\psi) = L(\psi) \cup \{[y,z]\}$
18 | **return** $L(\psi)$
19 **else if** $\psi = \langle \overline{B} \rangle \xi$ **then**
20 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z < y$ **do** $L(\psi) = L(\psi) \cup \{[x,z]\}$
21 | **return** $L(\psi)$
22 **else if** $\psi = \langle \overline{E} \rangle \xi$ **then**
23 | **for** $[x,y] \in \mathcal{H}[\xi][M]$ **and** $z > x$ **do** $L(\psi) = L(\psi) \cup \{[z,y]\}$
24 | **return** $L(\psi)$
25 **end**

information for those skipped loop iterations is stored in $\mathcal{H}$.

Algorithm 2 runs in time $O(|\mathcal{I}|^7)$ [MdFM+17], for some input instance $\mathcal{I} = (M, \varphi)$. In particular, the **for** loop at line 5 of Algorithm 2 is executed at most $|cl(\varphi)| = K$ since $|cl(\varphi)| \leq |\mathcal{I}|$, and the number of points of $[d]$ is bounded by $O(|\mathcal{I}|^3)$; thus, $|\mathbb{I}([d])|$ is bounded by $O(|\mathcal{I}|^6)$. We split the complexity in two parts: *i)* $O(K)$ for the formula φ, and *ii)* $O(|\mathcal{I}|^6)$ for the model M. Since we

Algorithm 5: Algorithm $\boldsymbol{FasterMMMC}$.

input : A collection of formulas $\Phi = \{\varphi_1, \ldots, \varphi_n\}$
input : A collection of finite interval models $\mathcal{M} = \{M_1, \ldots, M_m\}$.

1 $\mathcal{H} \leftarrow \emptyset$ ◁ Adapted hashtable of type $\mathcal{U} \mapsto (\mathcal{M} \mapsto \mathbb{I}([d]))$
2 **foreach** $\varphi_i \in \Phi$ **do**
3 **if** $\varphi_i \in \mathcal{H}$ **then** Do something.
4 **else**
5 **foreach** $\psi_i \in cl(\varphi_i)$ *(ordered by increasing size)* **do**
6 **if** $\psi_i \in \mathcal{H}$ **then** ◁ At least $|\mathcal{H}^i_{\mathcal{U}}| \cdot \frac{K}{N}$ checked sub-formulas
7 There is no need to model check ψ_i against any model.
8 **else**
9 **foreach** $M_j \in \mathcal{M}$ **do**
10 $\mathcal{H}[\psi_i][M_j] \leftarrow \boldsymbol{StepMC}(M_j, \psi_i, \mathcal{H})$
11 **end**
12 **end**
13 **end**
14 **end**
15 **end**

assumed that each model $M \in \mathcal{M}$ has a fixed finite domain $[d]$, then we can over-approximate $O(|\mathcal{I}|^6)$ to some (big) constant, say c. An algorithm that runs in $O(|\mathcal{I}|^6)$ time, which is essentially the projection of the **for** loop at line 5 of Algorithm 2 on a particular iteration (i.e., a sub-formula $\psi \in cl(\varphi)$), is depicted in Algorithm 4, where we also need $\mathcal{H}$ for the intervals at the previous step (since we are iterating over each element of $cl(\varphi)$ by increasing size).

Thanks to the above considerations, we can restate Proposition 4.2.1 as:

Proposition 4.4.1. *For a set of n formulas Φ and a set of m models $\mathcal{M}$, Algorithm 3 runs in time $O(cnmK)$, where $c = O(|\mathcal{I}|^6)$ and $K = |cl(\varphi)|$.*

The goal is to optimize such result in the remaining of this section. To this end, consider now Algorithm 5 which is the faster version of Algorithm 3, thanks to the supporting data structure $\mathcal{H}$ (defined in Section 4.3).

Theorem 4.4.2. *Given a set of n formulas Φ and a set of m models $\mathcal{M}$, then Algorithm 5 takes $O(cnmK(\frac{2N-n-1}{2N}))$ expected time, where $c = O(|\mathcal{I}|^6)$, for some instance $\mathcal{I} = (M, \varphi)$, $N = |\mathcal{U}|$ and $K = |cl(\varphi)|$.*

Proof. The most expensive instruction of the algorithm is at line 10 when calling the ***StepMC*** algorithm (depicted in Algorithm 4) which takes c time. Such instruction is inside the **foreach** loop at line 9. Thanks to Equation (4.2), such loop is executed *at most* $K - i\frac{K}{N}$ expected times. In general, the loop at line 9 is executed $(K - |\mathcal{H}^i_{\mathcal{U}}|\frac{K}{N})$ expected times because at each iteration i at least one sub-formula $\psi_i \in cl(\varphi_i)$ is inserted in the set of keys of $\mathcal{H}^i_{\mathcal{U}}$ (see Figure 4.1); that is, $|\mathcal{H}^i_{\mathcal{U}}| \geq i$. Moreover, the **if** instruction at line 6 takes $O(1)$ expected time. Finally, the **foreach** loop at line 2 takes $O(n)$ time. Now, putting all together we have the following:

$$\begin{aligned}
\sum_{i=1}^{n}((K - |\mathcal{H}^i_{\mathcal{U}}|\frac{K}{N}) \cdot \sum_{j=1}^{m} c) &\leq \sum_{i=1}^{n}((K - i\frac{K}{N}) \cdot \sum_{j=1}^{m} c) \\
&= cm \cdot \sum_{i=1}^{n}(\frac{K(N-i)}{N}) \\
&= cm\frac{K}{N} \cdot \sum_{i=1}^{n}(N-i) \\
&= cm\frac{K}{N} \cdot (nN - \frac{n(n+1)}{2}) \\
&= cnmK - cm\frac{K}{N}\frac{n(n+1)}{2} \\
&= cnmK(1 - \frac{1}{N}\frac{n+1}{2}) \\
&= cnmK(\frac{2N-n-1}{2N})
\end{aligned}$$

Thus, we obtain $O(cnmK(\frac{2N-n-1}{2N}))$ expected running time. □

The above theorem is an important result. In particular, it says that algorithm ***FasterMMMC*** outperforms algorithm ***NaiveMMMC*** as much as n grows. That is, it is better to very a larger set of formulas than a smaller one. The intuition is summarized in the following corollary.

Corollary 4.4.2.1. *If $n = N$, then Algorithm 5 runs in $O(\frac{cmK(n-1)}{2})$ expected time.*

If we are sampling all the formulas from $\mathcal{U}$, then using the adapted hashtable $\mathcal{H}$ gains at least half of the time with respect to the case where $\mathcal{H}$ is not involved. Moreover, we have proved a stronger result in Theorem 4.4.2. That is, if we

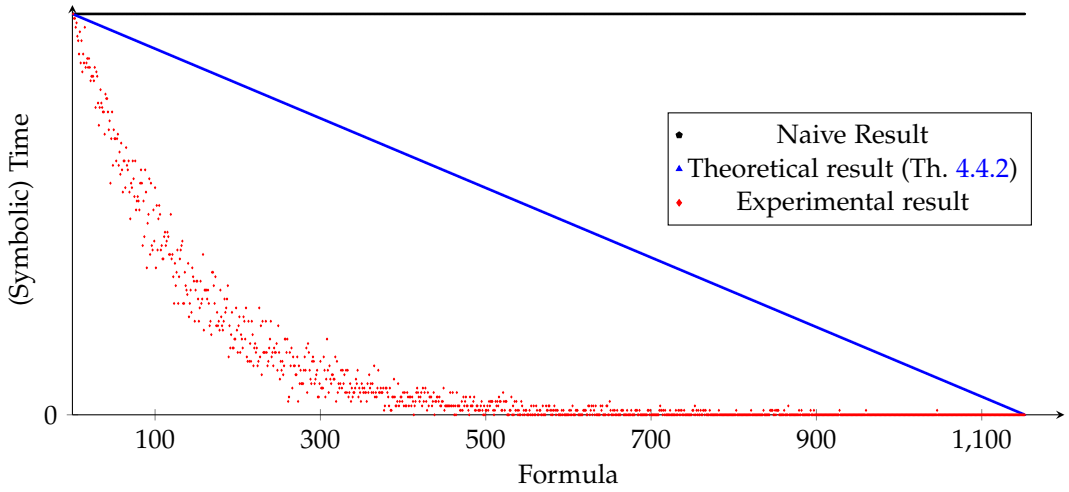


Figure 4.2: Time per evaluation plot: naive versus theoretical versus experimental trend using the proposed approach; the y axis represents the symbolic time since we are assuming some constants.

assume that the size of $\mathcal{H}^i_{\mathcal{U}}$ is increased by more than one key per iteration (i.e., $|\mathcal{H}^{i+1}_{\mathcal{U}}| > |\mathcal{H}^i_{\mathcal{U}}| + 1$), then, the above result essentially says that expected running time decreases more if we are willing to populate more the supporting data structure $\mathcal{H}$. In particular, the proposed algorithm has the nice feature that is highly parametrizable; e.g., the analyst can decide how many sub-formulas to add per iteration. Another interesting feature of such algorithm is that it is an *on-line algorithm*: since the data structure has been populated, a new set of formulas can be verified without any additional overhead, and we expect to evaluate only a small fragment of their sub-formulas after a reasonable number of already verified formulas. All these ideas are summarized in the following subsection.

4.4.3 Experimental evaluation

The expected time complexity result is given in *Big-Oh notation* [CLRS09, MR95] meaning that in practice we can expect better results. To this end, we can build an experiment to outline how the proposed approach performs in practice with respect to the theoretic upper bound. A bijective function that maps formulas to natural numbers can be virtually used to assign a (unique) number

to each formula. That is, for the sake of simplicity, suppose that each formula $\varphi \in \mathcal{U}$ is a natural number. Since each formula has K sub-formulas, we can randomly choose K natural numbers in the interval $[0, N-1]$ representing a particular formula φ, where $|\mathcal{U}| = N$. In a similar manner, other formulas can be created using the same approach. Now, suppose that $|\Phi| = N$; that is, N randomly generated formulas are ready to be verified. Because such formulas are randomly generated, it is very likely that many sub-formulas among the original formulas are the same, and, thus, it is very likely that in practice the upper bound expected time complexity is not reached. Once fixing the constants (and, in particular, the over-approximating constant for the ***StepMC*** algorithm), then running the algorithm ten times and taking the average time of all the executions, the theoretic and empirical trend are depicted in Figure 4.2. In particular, for a given formula, such figure represents the time needed to evaluate that formula against m models. Moreover, if we have N formulas to evaluate, then the evaluation of the N-th formula is null because (probabilistically) we have already seen all of the formulas from the universe $\mathcal{U}$. The important thing to point out is that, for example, other reasonable assumptions on how the input is distributed can be made in order to make the upper bound (recall Corollary 4.4.2.1) even tighter, but this treatment requires a proper investigation and it is beyond the purpose of this thesis. The same information can be checked in Figure 4.3 where the y axis represents the cumulative evaluation time for a given number of formulas.

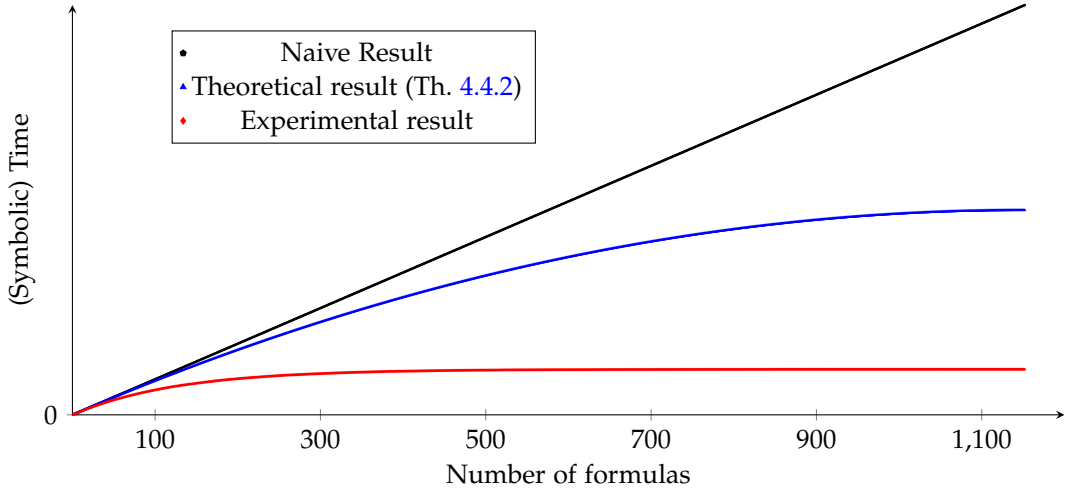


Figure 4.3: Cumulative time plot: naive versus theoretical versus experimental trend using the proposed approach; the y axis represents the symbolic time since we are assuming some constants.

Chapter 5

Conclusions

In this thesis, we proposed a generalization of the finite interval-based model checking problem. When reasoning about time, a mathematical formalism that can adequately model the evolution of a system over time is necessary. In our work, since we take time intervals as the primitive ontological entities, Halpern and Shoham's (multi-)modal interval temporal logic formalism is the natural choice. In order to suitably define the central notion of interval model, we introduced a general, domain-independent temporal abstraction that makes it possible to extract interval models from raw temporal data, that we call timelines.

Building on the concept of timeline, we generalized finite interval-based model checking of a single path/model to the case of multiple formulas and multiple models (many-to-many algorithm). We first showed that the obvious generalization of the original algorithm (one-to-one algorithm) can be successfully applied to at least two cutting-edge applications, which have been extensively described and analysed in detail. More precisely, from a technical point of view, we took the standard notion of temporal database and provided a suitable encoding of such a mathematical object into the timeline format that makes it possible the application of the generalized algorithm.

Then, we showed that the many-to-many generalized algorithm can actually be exploited to learn a temporal rule-based classifier, where the learning process is modelled as a multi-objective optimization problem, which intensively uses the one-to-one basic model checking algorithm [MdFM+17] to compute the *performance* of the classifier.

Next, by drawing inspiration from classic *caching lemmas*, we proposed an alternative many-to-many model checking algorithm, which benefits from some basic notion of probability theory, and proved that it allows us to get a better (expected) upper bound to time complexity.

As for future work, we aim at using the improved algorithm to learn a temporal rule-based classifier. It is important to observe the fact that a better upper bound to Theorem 4.4.2 can be achieved by simply assuming some probability distribution of the input, e.g., assuming that some sub-formulas are more frequent than others. This is something that we would like to investigate in future work. Moreover, a generalization of finite interval-based model checking to the infinite setting has been recently proposed in the literature [MMMS19], which assumes models to be non-sparse. An open question is to understand to which extent one can assume models to be sparse/non-sparse.

Bibliography

[All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[AMG+16] L. Aceto, D. Della Monica, V. Goranko, A. Ingólfsdóttir, A. Montanari, and G. Sciavicco. A complete classification of the expressiveness of interval logics of allen's relations: the general and the dense cases. *Acta Informatica*, 53(3):207–246, 2016.

[AMI+14] L. Aceto, D. Della Monica, A. Ingólfsdóttir, A. Montanari, and G. Sciavicco. On the expressiveness of the interval logic of allen's relations over finite and discrete linear orders. In *Logics in Artificial Intelligence - 14th European Conference, JELIA*, pages 267–281, 2014.

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In W. Rance Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207. Springer, 1999.

[BCG+18] D. Bresolin, E. Cominato, S. Gnani, E. Muñoz-Velasco, and G. Sciavicco. Extracting interval temporal logic rules: A first approach. In *Proc. of the 24th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 7:1–7:15, 2018.

[BHJ+06] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006.

[BMG+14] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014.

[BMM+14] D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Interval temporal logics over strongly discrete linear orders: Expressiveness and complexity. *Theoretical Computer Science*, 560:269 – 291, 2014. Games, Automata, Logic and Formal Verification.

[BMM+16] L. Bozzelli, A. Molinari, A. Montanari, A. Peron, and P. Sala. Interval temporal logic model checking: The border between good and bad HS fragments. In *Automated Reasoning - 8th International Joint Conference, IJCAR*, pages 389–405. Springer, 2016.

[BMM+19] D. Bresolin, D. Della Monica, A. Montanari, P. Sala, and G. Sciavicco. Decidability and complexity of the fragments of the modal logic of allen's relations over the rationals. *Information and Computation*, 266:97–125, 2019.

[BSS19] A. Brunello, G. Sciavicco, and I. E. Stan. Interval temporal logic decision tree learning. In *Logics in Artificial Intelligence - 16th European Conference (JELIA) 2019, Rende, Italy, May 7-11, 2019, Proceedings*, pages 778–793, 2019.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3 edition, 2009.

[CS04] Y. Collette and P. Siarry. *Multiobjective Optimization: Principles and Case Studies*. Springer, 2004.

[CT05] J. Chomicki and D. Toman. Temporal databases. In Michael David Fisher, Dov M. Gabbay, and Lluis Vila, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, chapter 14, pages 429–467. 2005.

[Deb01] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[EFHP10] M. Evans, C. Forbes, N. Hastings, and B. Peacock. *Statistical Distributions*. Wiley, 4 edition, 2010.

[Höp02] F. Höppner. Time series abstraction methods - A survey. In *Informatik bewegt: Informatik 2002 - 32. Jahrestagung der Gesellschaft für Informatik e.v. (GI), 30. September - 3.Oktober 2002 in Dortmund*, pages 777–786, 2002.

[HP14] F. Höppner and S. Peter. Temporal interval pattern languages to characterize time flow. *Wiley Int. Rev. Data Min. and Knowl. Disc.*, 4(3):196–212, 2014.

[HS91] J. Y. Halpern and Y. Shoham. A propositional modal logic of time intervals. *Journal of the ACM*, 38(4):935–962, 1991.

[JMMP+18] F. Jiménez, C. Martínez, L. Miralles-Pechuán, G. Sánchez, and G. Sciavicco. Multi-objective evolutionary rule-based classification with categorical data. *Entropy*, 20(9), 2018.

[LKWL07] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[LM13] A. R. Lomuscio and J. Michaliszyn. An epistemic halpern-shoham logic. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1010–1016, 2013.

[LM14] A. R. Lomuscio and J. Michaliszyn. Decidability of model checking multi-agent systems against a class of EHS specifications. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 543–548, 2014.

[LM16] A. R. Lomuscio and J. Michaliszyn. Model checking multi-agent systems against epistemic HS specifications with regular expressions. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016.*, pages 298–308, 2016.

[MdFM+17] D. Della Monica, D. de Frutos-Escrig, A. Montanari, A. Murano, and G. Sciavicco. Evaluation of temporal datasets via interval temporal logic model checking. In *Proc. of the 24th International Symposium on Temporal Representation and Reasoning (TIME)*, pages 11:1–11:18, 2017.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[MMMS19] D. Della Monica, A. Montanari, A. Murano, and G. Sciavicco. Ultimately-periodic interval model checking for temporal dataset evaluation. In *Proc. of the 5th Global Conference on Artificial Intelligence (GCAI)*, pages 30:30–30:43, 2019.

[Mol19] A. Molinari. *Model checking: the interval way*. PhD thesis, University of Udine, 2019.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[MS15] R. Moskovitch and Y. Shahar. Classification-driven temporal discretization of multivariate time series. *Data Min. Knowl. Discov.*, 29(4):871–913, 2015.

[MU05] F. Mörchen and A. Ultsch. Optimizing time series discretization for knowledge discovery. In *Proceedings of the Eleventh ACM*

SIGKDD International Conference on Knowledge Discovery in Data Mining, pages 660–665, 2005.

[Mur12] Kevin P Murphy. *Machine learning: A probabilistic perspective*. MIT press, 2012.

[MVPS+19] E. Muñoz-Velasco, M. Pelegrín, P. Sala, G. Sciavicco, and I. E. Stan. On coarser interval temporal logics. *Artificial Intelligence*, 266:1–26, 2019.

[SSV19] G. Sciavicco, I. E. Stan, and A. Vaccari. Towards a general method for logical rule extraction from time series. In *From Bioinspired Systems and Biomedical Applications to Machine Learning - 8th International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC) 2019, Almería, Spain, June 3-7, 2019, Proceedings, Part II*, pages 3–12, 2019.

[Var05] M. Y. Vardi. Model checking for database theoreticians. In *Proc. of the 10th International Conference on Database Theory (ICDT)*, pages 1–16. Springer, 2005.

[Zem12] F. Zemke. What's new in SQL:2011. *ACM SIGMOD Record*, 41(1):67–73, 2012.