Noname manuscript No.
(will be inserted by the editor)

Observation Strategies for Event Detection with Incidence on Runtime Verification: Theory, Algorithms, Experimentation

**Marco Alberti · Pierangelo Dell'Acqua ·
Luís Moniz Pereira**

Received: date / Accepted: date

Abstract Many applications (such as system and user monitoring, runtime verification, diagnosis, observation-based decision making, intention recognition) all require to detect the occurrence of an event in a system, which entails the ability to observe the system. Observation can be costly, so it makes sense to try and reduce the number of observations, without losing full certainty about the event's actual occurrence. In this paper, we propose a formalization of this problem. We formally show that, whenever the event to be detected follows a discrete spatial or temporal pattern, then it is possible to reduce the number of observations. We discuss exact and approximate algorithms to solve the problem, and provide an experimental evaluation of them. We apply the resulting algorithms to verification of linear temporal logics formulæ. Finally, we discuss possible generalizations and extensions, and, in particular, how event detection can benefit from logic programming techniques.

Keywords Event detection · Runtime verification · Temporal logic · Logic programming · Complexity

1 Introduction

An event can be defined as a combination of observable states in a system, that may occur at some points in space and time, in known configurations.

M. Alberti and L. Moniz Pereira
Centro de Inteligência Artificial (CENTRIA), Departamento de Informática
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Caparica (Portugal)
E-mail: m.alberti@fct.unl.pt,lmp@di.fct.unl.pt

P. Dell'Acqua
Dept. of Science and Technology, ITN
Linköping University
Norrköping (Sweden)
E-mail: pierangelo.dellacqua@liu.se

Event detection is a crucial process in many applications. For instance, in system and user monitoring [CDPT10], events need to be detected in order to check the correct functioning of the system or even the user's behavior; in runtime verification [LS09], observation is necessary to verify whether the system's behavior is exhibiting specified desirable properties; in diagnosis [LP06, PL07], events must be observed before an explanation for them is sought; in observation-based decision making [PR10], event detection decidedly influences the decision making process.

In order to detect events, it is of course necessary to observe the very system where the event may occur, and to try and match observations to the available knowledge about the events. Generally speaking, a system will be observable at some given points in a set of spatial/temporal coordinates, and not in others; each observation will consist in giving a description of the state of the system at a particular spatial/temporal point.

For example, in a system to monitor a patient's temperature, the event being monitored for detection might be the temperature being higher than 37°C, and the coordinate points would represent time points, each observation being the pair consisting of a time point and a temperature value. The designer may choose to represent time points by a discrete (say, one point for each minute) or a continuous sequence. As another example, consider the widely studied problem of intention recognition, defined as the process of becoming aware of the intention of another agent and, more technically, as the problem of inferring an agent's intention through the observation of some of its actions and their effects on the environment [KA86, HSB09]. In this case, what is detected are performed actions, events being collections of actions that accord with possible plans or action sequences known to achieve some goal.

However, observation may be a costly process: for instance, because of required instruments, because of needed analytical resources, of logging deluge, or because of time constraints on observations requiring fast or impossible to meet demands, or because of excessive and swamping logging. In these cases, observing the system at all spatial or temporal points may be undesirable or unfeasible, and the question naturally arises whether it is possible, whenever events follow a known spatial/temporal pattern, to reduce the number of required observations, without losing certainty about an event's occurrence. In other words, can a portion of system states be safely ignored without loss of significant knowledge? For example, in a monitoring system, such a possibility would let the designer employ cheaper hardware, required to support milder operation frequencies, and also to reduce excessive communication traffic, as well as logging storage and related costs.

Such considerations are relevant to the aforementioned applications. In intention recognition, for instance, not all actions required to achieve a goal need to have been observed in order to conclude about possible intentions. Nor are all actions necessarily observable; which underscores our point of concern about efficacy and efficiency of observation attempts, and correspondent resource allocation. In the last decade, as agent systems have been increasingly used in several real-world applications (see [FBHT07] and the many references therein), the issue of runtime verification of properties, which express the correct functioning of the system with respect to its specification, has become increasingly important. In previous work, Costantini et al. developed a framework for logic evolving agents (cf. [CTTT07,CDP08]), where every agent is seen as the composition of an object level program plus one or more metalevels. In this framework, agents interact with the environment and are able to modify themselves and evolve according to both external and internal stimuli. To address this issue, in [CDPT09] the authors introduced an approach to runtime verification of properties of agent behavior, wherein they adopted a temporal logic representation of such properties. The proposed framework was employed to develop suitable case-studies in significant application realms in ambient intelligence, like for example surveillance systems for elderly people or those with disabilities, where the system is intended to monitor their ongoing behavior. In such application it is often possible to reduce the run time monitoring effort by making use of the available knowledge about behavior patterns; as a simple example, if it is known that a behavior is observable for at least ten consecutive time points, it is possible to detect it by observing the system only every tenth time point, instead of at all time points. A desire to formally prove and generalize such an intuitive result originally inspired the present work. And since the other applications mentioned at the beginning of this introduction also require event detection, we believe that they too can potentially benefit from the work presented herein.

Indeed, in this paper we formalize the problem of whether it is possible to safely reduce the number of necessary observations in the general case, and show that the answer is yes for the simple, but sufficiently significant case where the observation space can be modeled as a set of (not necessarily consecutive) natural numbers. This being the case in the aforementioned temperature monitoring or ambient intelligence examples, or, more generally, in domains where events occur in systems of discrete spatial or temporal coordinates, such as those in which events can be observed at a sequence of temporal instants, or at points in a spatial curve. We prove that, in this case, the problem of finding a smaller set of observation points can be reduced to a well known NP-complete problem (minimum hitting set); and we show experimentally that solving it by means of a polynomial approximation algorithm yields noteworthy reductions in the number of necessary observation points. We also show that the results proven here can be expressed in the formalism of Linear Temporal Logics [Pnu77], widely utilized in model checking and, more recently, in runtime verification [BLS11], thereby opening a wider application range.

A considerable effort has been spent to develop techniques for discovering patterns in time series data. For instance, [GS99] proposes an iterative algorithm for discovering events in time series data by means of point of change detection (that is, a change in the parameters of the system), while [HKM$^+$96] and [MTV97] focus on discovering episodes (defined as a collection of events that occur relatively close to each other) in event sequences, by data mining; [PT96] extends those works, in applying a temporal logic programming approach to pattern discovery. [JDtL06] uses data mining to learn spatial-temporal traffic patterns, in order to identify possible accidents, by comparing real-time data with the historical models. Our contribution can be regarded as downstream of those on pattern discovery, in that it uses knowledge about patterns to sparingly strive to detect the events that follow those patterns.

In this paper, we distinctly focus on the problem of reducing the number of necessary observations to detect an event of already *known* pattern (by design or by prior discovery, say). This distinguishes the setting of our work from that of previous work on spatial-temporal cluster detection, such as [Iye04] and [NMSD05], because, in those works, the knowledge about the cluster pattern is not complete (in [NMSD05] clusters are rectangles of unknown length and width, and in [Iye04] they can expand or shrink over time). Moreover, in a particular case we study in more detail (of a coordinate system that can be represented as a set of natural numbers) we do not assume the observation points for the event to be contiguous.

Attempts to reduce the number of observations have been applied to problems, such as string matching [SCC05]), which are different from ours, although related (as we shall discuss in the sequel).

The paper is structured as follows. In Section 2, we provide a general formulation of the problem. In Section 3, we study the problem for the special case where the temporal or spatial event structure can be described by a set of natural numbers; for this case, we show the reduction of the problem to a well known NP-complete problem (minimum hitting set). In Section 4, we discuss and compare experimentally an exact and an approximation algorithm to solve the problem. In Section 5, we consider the problem in the context of runtime verification of Linear Temporal Logic formulæ, and we apply the results proved in Section 3 to this setting. In Section 6, we discuss possible extensions and generalizations of our approach, and the application of techniques borrowed from logic programming to problems beyond plain event detection.

2 Event detection in general

In this section, we introduce the definitions of the basic concepts, and a general formulation of the problem we address.

2.1 Coordinate system

A coordinate system defines points where events can happen. Coordinates can be interpreted, for instance, as spatial, temporal, or both. We only require the operation of sum (used in Definition 3, of event) to be defined over coordinates.

Definition 1 A **coordinate system** $\mathcal{S}$ is the cartesian product of N sets, closed under the sum operation.

Each of the sets can be continuous (e.g., the set $\mathbb{R}$ of real numbers), or discrete (e.g., the sets $\mathbb{N}$ of natural numbers or $\mathbb{Z}$ of integer numbers).

2.2 Patterns and events

A pattern associates a value taken from a **value set** $\mathcal{V}$ to (some) points in the coordinate system.

We do not impose any restriction on admissible value sets. In the simplest case, the value set will have only one element; this representation is satisfactory in cases where the value is not significant, and the relevant information is the set of points where the pattern is defined.

Definition 2 Given a coordinate system $\mathcal{S}$, a **pattern** $\mathcal{P}$ over $\mathcal{S}$ is a partial function from $\mathcal{S}$ to some value set $\mathcal{V}$.

As usual, a pattern $\mathcal{P}$'s **domain** (*dom* $\mathcal{P}$) is the set of points where $\mathcal{P}$ is defined, in general a subset of $\mathcal{S}$.

$\mathcal{S}$	$\mathcal{V}$	$\mathcal{P}$	*dom* $\mathcal{P}$
$\mathbb{N}$	$\{1\}$	$\{\langle 0,1\rangle, \langle 4,1\rangle, \langle 5,1\rangle\}$	$\{0,4,5\}$
$\mathbb{R}^2$	$\mathbb{R}$	$\{\langle\langle 0,0\rangle,0\rangle\}$	$\{\langle 0,0\rangle\}$
$\mathbb{R}^2$	$\mathbb{R}$	$\{\langle\langle x,y\rangle,x\rangle \mid x \in \mathbb{R} \wedge y \in \mathbb{R}\}$	$\mathbb{R}^2$
$\mathbb{R}^2$	$\mathbb{R}$	$\{\langle\langle x,y\rangle,1\rangle \mid 0 \le x \le 1 \wedge 0 \le y \le 1\}$	$\{\langle x,y\rangle \mid 0 \le x \le 1 \wedge 0 \le y \le 1\}$

Table 1 Example patterns

An event is an association of values to points in a coordinate system, that follows a given pattern. It is defined as the translation of its pattern by an offset.

Definition 3 Given a pattern $\mathcal{P}$ and $\tau \in \mathcal{S}$, the **event** $\mathcal{E}_\mathcal{P}(\tau)$ of pattern $\mathcal{P}$ and offset τ is the set $\{\langle \tau + p, \mathcal{P}(p)\rangle \mid p \in \mathit{dom}\ \mathcal{P}\}$.

We call *unfolding* the set of all possible events of a given pattern.

Definition 4 Given a pattern $\mathcal{P}$ in a coordinate system $\mathcal{S}$, the **unfolding** $\mathcal{U}_\mathcal{P}$ of $\mathcal{P}$ is the set $\{\mathcal{E}_\mathcal{P}(\tau) \mid \tau \in \mathcal{S}\}$.

The observability set of an event e is the set of points in the coordinate system where e can be observed.

Definition 5 For any event e, the **observability set** of e, denoted by $\mathcal{O}(e)$, is the projection of e over $\mathcal{S}$.

Example 1 If $\mathcal{P}$ is the first pattern in Table 1, the event $\mathcal{E}_\mathcal{P}(5)$ is $\{\langle 5,1\rangle, \langle 9,1\rangle, \langle 10,1\rangle\}$, and its observability set is $\{5,9,10\}$.

A set of points in the coordinate system that is guaranteed to intersect the observability set of any event with a given pattern can be understood as a suitable set of points to observe in order to decide about the occurrence of that event. We say that such a set *covers* the pattern, or is one of the pattern's *covering sets*. For example, if the coordinate system represents a spatial area, a covering set can be understood as a set of positions to deploy sensors (for example, in wireless sensor networks applications). Note that the covering of a pattern by a set does not depend on the pattern's value set.

Definition 6 A set $\mathcal{C} \subseteq \mathcal{S}$ **covers** a pattern $\mathcal{P}$ (or, equivalently, is a **covering set** of $\mathcal{P}$'s) if and only if for each $\tau \in \mathcal{S}$ there exists an element of $\mathcal{O}(\mathcal{E}_{\mathcal{P}}(\tau))$ that belongs to $\mathcal{C}$.

Example 2 Consider a pattern $\mathcal{P}$, whose domain is $\{0, 1, 2\}$, on the coordinate system of the set $\mathbb{N}$ of natural numbers. $\{3n \mid n \in \mathbb{N}\}$ covers $\mathcal{P}$. $\mathcal{C} = \{1, 2\}$ does not cover $\mathcal{P}$, because, for instance, 3 belongs to the observability set of the event of pattern $\mathcal{P}$ and offset 3 and does not belong to $\mathcal{C}$. For the same reason, $\{4n \mid n \in \mathbb{N}\}$ does not cover $\mathcal{P}$.

Example 3 $\mathcal{C} = \mathbb{Z}\times\mathbb{Z}$ covers the last pattern in Table 1. Indeed, the observability set of a generic event of offset $\tau = \langle \tau_x, \tau_y \rangle$ is
$\{\langle x, y\rangle \mid \tau_x \leq x \leq \tau_x + 1 \ \wedge \ \tau_y \leq y \leq \tau_y + 1\}$; it contains $\langle \lfloor \tau_x \rfloor + 1, \lfloor \tau_y \rfloor + 1\rangle$ (an element of $\mathcal{C}$), because, for any $\alpha \in \mathbb{R}$, $\alpha \leq \lfloor \alpha \rfloor + 1 \leq \alpha + 1$.

2.3 The sampling problem

If the application requires to determine the occurrence of an event with a given pattern, it is of interest to determine a set of points in the coordinate system that guarantees that, if an event occurs, its occurrence will be detected[1].

Definition 7 Given a pattern $\mathcal{P}$ over $\mathcal{S}$, the **sampling problem** consists of finding a set $\mathcal{C} \subseteq \mathcal{S}$ that covers $\mathcal{P}$.

The problem has a trivial solution ($\mathcal{S}$ itself, i.e., to observe the system at all points), but the solution may be required to satisfy some property. A reasonable requirement is for it to minimize a cost function; if the cost of observation is a constant, that translates to minimum cardinality. However, applications can impose further or different requirements. For instance, if the points in the coordinate system represent time points, then it may be required for the covering set to intersect observability sets at or near their minimum, in order to detect events as soon as possible.

Comparison with string matching String matching consists of finding one or all the occurrences of a string P, called a pattern, in a text T. String matching is an important problem, it has been studied for decades, and has many application areas. The string-matching algorithms developed can be divided into four categories, depending on the order of comparisons of the characters in P and T: (*i*) from left to right, (*ii*) from right to left, (*iii*) in a specific order, and (*iv*) in any order.

[1] Note that detecting the occurrence is different from observing the whole event. For example, if the event consists in a red light staying on for a continuous interval in time, to detect the event it is sufficient to observe the light at any one time point in the interval.

Event detection, at first sight, may appear to be similar to a string matching problem where P is given before T, and comparisons can be done in any order. However, it differs substantially from string matching, since our aim (using string matching terminology) is to determine whether a character in P occurs in T, knowing that T fulfills some property. For instance, the problem of detecting an event of pattern $\mathcal{P}=\{\langle 0,a\rangle,\langle 1,a\rangle,\ldots,\langle M,a\rangle\}$ in a stream T can be formulated as follows: given an alphabet $\Sigma=\{a,x\}$, let T be described by a regular expression of the form $x^*(a^{M+1}x^n)^*$, and determine whether the character a occurs in T. In other words, in our problem we can exploit knowledge about T, which is not assumed in string matching.

For this reason, techniques to reduce the number of observations developed for string matching (see, for instance, [SCC05]) cannot be applied to the problem studied here.

3 Sampling problem for natural patterns

In this section, we consider the case where the coordinate system is the set $\mathbb{N}$ of natural numbers, and we discuss the concomitant sampling problem.

Definition 8 A **natural pattern** is a pattern over the set $\mathbb{N}$ of natural numbers.

A natural interpretation for this case is that the coordinate system represents a sequence of time points at which the system can be observed. However, other interpretations are possible: for instance, spatial points, or states in a path on a Kripke structure (see Section 5).

For applications, it seems reasonable to consider patterns whose domain has 0 as its minimum: after all, the representation of the pattern domain is the designer's choice. However, the following result lets us reduce to this case, for generic pattern domains, by means of a translation.

Definition 9 Given a natural pattern $\mathcal{P}$, let $m = \min\ \mathit{dom}\ \mathcal{P}$.
Then the **normalization** of $\mathcal{P}$ is the pattern
$\overline{\mathcal{P}} = \{\langle p,\mathcal{P}(p+m)\rangle \mid p+m \in \mathit{dom}\ \mathcal{P}\}$.

Obviously, for any natural pattern $\mathcal{P}$, $\mathit{dom}\ \overline{\mathcal{P}}$ is $\mathit{dom}\ \mathcal{P}$ translated by $-m$, and $\min\ \mathit{dom}\ \overline{\mathcal{P}}=0$.

Theorem 1 *Let $\mathcal{P}$ be a natural pattern. Then an event of pattern $\mathcal{P}$ is also an event of pattern $\overline{\mathcal{P}}$.*

Proof $\mathcal{E}_{\mathcal{P}}(\tau) = \{\langle \tau+p,\mathcal{P}(p)\rangle \mid p \in \mathit{dom}\ \mathcal{P}\}$ which, by substituting $q+m$ for p, where $m = \min\ \mathit{dom}\ \mathcal{P}$, is equal to
$\{\langle \tau+q+m,\mathcal{P}(q+m)\rangle \mid q+m \in \mathit{dom}\ \mathcal{P}\} = \mathcal{E}_{\overline{\mathcal{P}}}(\tau+m)$.

Therefore, the unfolding of the original pattern is a subset of the unfolding of the normalized pattern, and a set that covers the normalized pattern also covers the original pattern. This lets us solve the sampling problem for the normalized pattern, whose domain's maximum element is in general smaller, and thus reduce complexity and the approximation ratio, if an approximation algorithm is used (see Section 4).

The following (intuitive) result shows that the covering set for a natural pattern cannot be finite.

Theorem 2 *A natural pattern cannot be covered by a finite set.*

Proof Let $\mathcal{P}$ be a pattern and $\mathcal{I}$ a generic finite subset of $\mathbb{N}$. If $\mathcal{I}$ is finite, it has a maximum M. Since dom $\mathcal{P}$ is composed of natural numbers, all the elements of the observablility set of the event e of pattern $\mathcal{P}$ and offset $M+1$ are strictly greater than M; therefore, none of them belongs to $\mathcal{I}$. Since $\mathcal{I}$ does not intersect the observability set of an event of pattern $\mathcal{P}$, $\mathcal{I}$ does not cover $\mathcal{P}$.

The covering set must be infinite, which makes it dubious how to approach its computation, depending on the properties that it is required to satisfy (such as minimum cardinality).

To address this issue, we now prove a result that shows how the sampling problem for natural patterns can be reduced to finding a *finite* set.

First, two definitions are provided.

Definition 10 Let $\mathcal{I}$ be a finite set of natural numbers, and let $M = max\ \mathcal{I}$.

For an integer k, let $\sigma_k = \{(p+k) \texttt{ mod } (M+1) \mid p \in \mathcal{I}\}$.

$\mathcal{I}$'s **circular repetition** is the collection of sets

$\mathcal{Z} = \{\sigma_k \mid k \in [0 \,..\, M]\}$, where $[0 \,..\, M]$ is the interval of integers from 0 to M.

Definition 11 Let $\mathcal{P}$ be a natural pattern.

Then a set with non-empty intersection with each element of the circular repetition of *dom* $\mathcal{P}$ is a **covering shape** of $\mathcal{P}$'s.

Theorem 3 *Let $\mathcal{P}$ be a natural pattern and let $M = max\ dom\ \mathcal{P}$.*

Let $\mathcal{S}_C$ be a covering shape of $\mathcal{P}$'s. Then the set
$\mathcal{C} = \{q + k(M+1) \mid q \in \mathcal{S}_C \wedge k \in \mathbb{N}\}$ covers $\mathcal{P}$.

Proof Consider a generic event e of pattern $\mathcal{P}$ and offset τ.

Let $R = \tau \texttt{ mod } (M+1)$. Consider σ_R. By hypothesis, it intersects $\mathcal{S}_C$ in (at least) a natural number q. Since $q \in \sigma_R$, there exists $p \in dom\ \mathcal{P}$ such that $q = (p+R) \texttt{ mod } (M+1)$. Since $p \in dom\ \mathcal{P}$, $\tau + p \in \mathcal{O}(e)$. Since τ is equal to R modulo $M+1$, $\tau + p$ is equal to $R + p$ modulo $M+1$, which is equal to q modulo $M+1$. Besides, since for any natural numbers a and b (with $b \geq 0$), $a \texttt{ mod } b \leq a$, $R \leq \tau$, so $R+p \leq \tau+p$, thus $(R+p) \texttt{ mod } (M+1) \leq \tau+p$, i.e., $q \leq \tau + p$. Summarizing, (i) $\tau + p$ is equal to q modulo $M+1$, and (ii) $q \leq \tau+p$; thus, $\exists k \in \mathbb{N} \mid \tau + p = q + k(M+1)$. Therefore, also $\tau + p \in \mathcal{C}$.

In conclusion, the observability set of a generic event of pattern $\mathcal{P}$ intersects $\mathcal{C}$; that is, $\mathcal{C}$ covers $\mathcal{P}$.

Therefore, the sampling problem can be reduced to finding a finite set $\mathcal{S}_C$, the pattern's covering shape, that intersects a collection of $M+1$ subsets of $[0 \mathinner{..} M]$. The periodic repetition of $\mathcal{S}_C$, with period $(M+1)$, covers the pattern. A trivial, and uninteresting, solution is $[0 \mathinner{..} M]$ itself (which corresponds to observing the system at all time points). However, finding such a set with minimum cardinality is a well known NP-hard problem: the *minimum hitting set* problem. In Section 4 we discuss exact and approximate solutions to this problem.

The following definition gives a measure of how much the number of necessary observations is reduced by observing the system only at the periodic repetition of a pattern's covering set, instead of at all points.

Definition 12 Given a normalized pattern $\mathcal{P}$ and a covering shape $\mathcal{S}_C$ of $\mathcal{P}$'s, the corresponding **sampling ratio** is defined as $\mathcal{R}_{\mathcal{S}_C,\mathcal{P}} = \frac{|\mathcal{S}_C|}{1+\max \mathit{dom}\ \mathcal{P}}$.

$\lvert \mathit{dom}\ \mathcal{P} \rvert$	2	4	6	8	10	12	14	16	18	20
$\mathcal{R}_{\mathcal{S}_C,\mathcal{P}}$	0.52	0.32	0.24	0.19	0.15	0.14	0.12	0.1	0.1	0.1

Table 2 Sampling ratios.

Example 4 Table 2 shows average sampling ratios for random patterns with domain maximum of 20, for different values of the domain cardinality. In particular, for each random pattern, a covering shape of minimum cardinality was computed with the exact algorithm introduced in Section 4.1, and the sampling ratio was computed per its definition.

Example 5 Let $[0 \mathinner{..} M]$be the domain of a pattern $\mathcal{P}$. By Definition 10, $\mathit{dom}\ \mathcal{P}$'s circular repetition is $\mathcal{Z} = \{\sigma_0, \ldots, \sigma_M\}$, where for $i \in [0 \mathinner{..} M]$ $\sigma_i = [0 \mathinner{..} M]$. A minimum hitting set for $\mathcal{Z}$ is $\{i\}$, for any $i \in [0 \mathinner{..} M]$. Therefore, for each i, by Theorem 3 a covering set for $\mathcal{P}$ is $\{i + k(M+1) \mid k \in \mathbb{N}\}$. The corresponding sampling ratio, by Definition 12 is $\frac{1}{M+1}$, confirming the intuitive idea that, if an event lasts $M+1$ points, it is sufficient to observe every $(M+1)$-th point to detect its occurrence.

Example 6 Consider a system that has to monitor whether a user is pressing a button, and to take some action in response; for instance, a piece of equipment that should be reset when the user pushes the reset button, and that checks the status of the reset button by periodic polling. Let the system clock be 1MHz. At each clock step, the signal corresponding to the button will be 1 if the button is pressed, and 0 if the button is not pressed. It is known that the releasing of the button, once it has been pressed, takes at least 1 ms (1000 clock cycles); is it worth checking the signal at each clock cycle? Of course not. With reference to Example 5, the pressing is an event whose pattern has domain $[0 \mathinner{..} M]$ with $M = 999$; and in order to decide about the button having been pressed, it is sufficient to observe the signal every 1000-th clock cycle, saving computing and communication resources.

Example 7 Consider a pattern $\mathcal{P}$ whose domain is $\{8, 10, 13\}$. Then $dom\ \overline{\mathcal{P}} = \{0, 2, 5\}$.

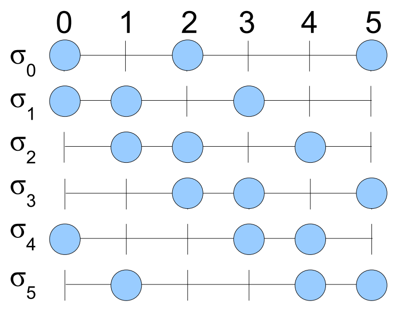

Fig. 1 Circular repetition of $dom\ \overline{\mathcal{P}}$.

By Definition 10, $dom\ \overline{\mathcal{P}}$'s circular repetition is $\mathcal{Z} = \{\sigma_0, \ldots, \sigma_5\}$, where $\sigma_0 = \{0, 2, 5\}, \sigma_1 = \{0, 1, 3\}, \sigma_2 = \{1, 2, 4\}, \sigma_3 = \{2, 3, 5\}, \sigma_4 = \{0, 3, 4\}$, and $\sigma_5 = \{1, 4, 5\}$ (see Figure 1).

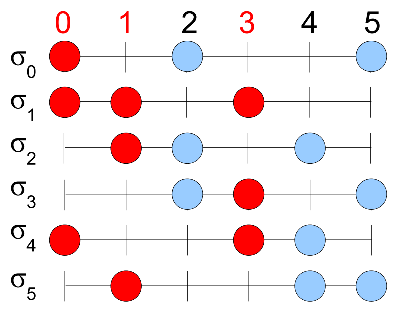

Fig. 2 A covering shape of $\mathcal{P}$ is $\{0, 1, 3\}$, which intersects the element of the circular repetition of $dom\ \mathcal{P}$ in the red (darker in grayscale) circles.

By Definition 11, $\overline{\mathcal{P}}$'s covering shape $\mathcal{S}_C$ is required to have non-empty intersection with all the sets in $\mathcal{Z}$; for instance, a possible covering shape is $\mathcal{S}_C = \{0, 1, 3\}$ (Figure 2). It can be easily verified that its cardinality is minimal.

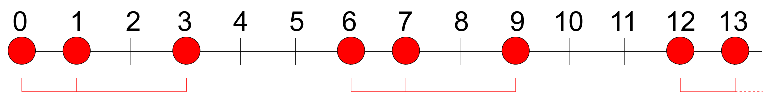

Fig. 3 A covering set for $\mathcal{P}$.

By Theorem 3, the corresponding covering set of $\overline{\mathcal{P}}$'s is $\mathcal{C} = \{q + 6k \mid q \in \{0, 1, 3\} \wedge k \in \mathbb{N}\} = \{0, 1, 3, 6, 7, 9, 12, \ldots\}$ (Figure 3), which also covers $\mathcal{P}$.

The sampling ratio, by Definition 12, is $\mathcal{R}_{\mathcal{S}_C, \overline{\mathcal{P}}} = 3/6 = 1/2$, which corresponds to a reduction in necessary observation points of 50%.

Example 8 With reference to Example 7, consider an event that is known to activate a sensor

– at the beginning of the event;
– again after 2 time steps;
– again after 3 more time steps.

It is also known that nothing else can cause the activation of the sensor.

In order to detect the event, one can of course observe the sensor at each time step; or one can note that the activation of the sensor is an event whose pattern has domain $\{0, 2, 5\}$ and exploit the result of Example 7, and so observe the sensor at the time steps in the set $\mathcal{C} = \{q + 6k \mid q \in \{0, 1, 3\} \wedge k \in \mathbb{N}\}$, thus saving half of the observations.

Example 9 Consider again Example 6, but suppose now that, in order to save energy, the signal from the reset button has a duty-cycle of 1/100 (i.e., while the button is pressed, the signal is 1 for 1 microsecond and 0 for 99 microseconds). Suppose, moreover, that it can be safely assumed that the button stays pressed for 1 millisecond. Then the domain of the pattern of the reset signal is $\{0, 100, 200, 300, 400, 500, 600, 700, 800, 900\}$. The sampling ratio of a corresponding covering shape (computed with the approximation algorithm described in Section 4.2[2]) is 1/10; this means that the number of observations necessary to detect whether the reset button is being pressed can be safely reduced by 90%.

4 Algorithms and experimental evaluation

In this section, we discuss solution methods for the minimum hitting set problem to which the sampling problem can be reduced, for natural patterns, as shown in section 3. We describe an exact algorithm for the minimum hitting set problem, and we discuss reduction of the minimum hitting set problem to the set cover problem, and the solution of the latter by means of a well-known approximation polynomial algorithm. We compare experimentally the two algorithms.

```
input  : A natural pattern $\mathcal{P}$'s domain $D$
output : A covering shape of $\mathcal{P}$'s with minimal cardinality
begin
    $circ\_rep \leftarrow$ circular_repetition($D$)
    $best \leftarrow [0 .. max\ D]$
    $candidate \leftarrow \emptyset$
    while $candidate \leftarrow$ gen_next_set($candidate$,cardinality($best$) $- 1$) do
        if hits($candidate$,$circ\_rep$) then
            $best \leftarrow candidate$
        end
    end
    return $best$
end
```

Algorithm 1: Algorithm to compute a natural pattern's covering shape with minimal cardinality.

4.1 Exact algorithm

Algorithm 1 is based on a total order of all the elements of $2^{[0\ ..\ max\ dom\ \mathcal{P}]}$; one of these elements is returned as a minimal hitting set.

The algorithm tries, as a first attempt, the full integer interval from 0 to *max dom* $\mathcal{P}$, which obviously hits the pattern domain's circular repetition. Then, at each iteration, it tests for hitting the next set with lesser cardinality than the current best, according to the aforementioned order.

The algorithm uses the following functions:

- `cardinality` takes a set as an argument, and returns its cardinality;
- `circular_repetition` takes a finite natural set as an argument, and computes its circular repetition (see Definition 10);
- `gen_next_set` takes a set $\mathcal{I}$ and a natural number n as arguments, and returns the first set of natural numbers with at most n elements that follows $\mathcal{I}$ in the chosen enumeration of $2^{[0\ ..\ max\ dom\ \mathcal{P}]}$, or *false* if no such set exists;
- `hits` takes a set $\mathcal{I}$ and a collection $\mathcal{Z}$ of sets as arguments, and returns *true* if $\mathcal{I}$ has non-empty intersection with all elements of $\mathcal{Z}$.

4.2 Reduction to set cover and approximate solution

Each minimum hitting set problem can be transformed into a *set cover* problem: given a collection $\mathcal{S} \subseteq 2^{[1\ ..\ n]}$, find a subset of $\mathcal{S}$ of minimal cardinality whose union is $[1\ ..\ n]$.

Given a collection $\mathcal{R} = r_1 \ldots r_n$ of sets, whose union is $[1\ ..\ k]$, the minimum hitting set problem for $\mathcal{R}$ can be formulated as a set cover problem as

[2] Such covering shape is $\{0, 1, 2, 3, 32, 33, 34, 35, 64, 65, 66, 67, 96, 97, 98, 99, 128, 129, 130, 131, 160, 161, 162, 163, 192, 193, 194, 195, 224, 225, 226, 227, 256, 257, 258, 259, 288, 289, 290, 291, 320, 321, 322, 323, 352, 353, 354, 355, 384, 385, 386, 387, 416, 417, 418, 419, 448, 449, 450, 451, 480, 481, 482, 483, 512, 513, 514, 515, 544, 545, 546, 547, 576, 577, 578, 579, 608, 609, 610, 611, 640, 641, 642, 643, 672, 673, 674, 675, 704, 705, 706, 707, 736, 737, 738, 739, 768, 769, 770, 771\}$.

follows. For each $e \in [1 .. k]$, let $s_e = \{i \mid e \in r_i\}$, and let $\mathcal{S}$ be the collection of all such sets. The union of each subset $\mathcal{T}$ of $\mathcal{S}$ is the set of indices of the sets in $\mathcal{R}$ which contain one of the indices of the sets in $\mathcal{T}$. Thus, the indices of a solution $\mathcal{T}$ of a set cover problem for $\mathcal{S}$ form a minimal hitting set for $\mathcal{R}$, because $\mathcal{T}$'s union is $[1 .. n]$, so all elements of $\mathcal{R}$ have non-empty intersection with the set of $\mathcal{T}$'s indices, and the union of no proper subset of $\mathcal{T}$ is $[1 .. n]$, or $\mathcal{T}$ would not be minimal.

The set cover problem has been widely studied in the literature; in particular, approximation algorithms have been proposed for it. A very simple *greedy* algorithm, at each step, chooses the set with higher cardinality, and it deletes such set's elements from the other sets, until the universe is covered.

Its performance (expressed as the ratio between the cardinality of the output and of the minimal covering collection) is $\ln n - \ln\ln n + \Theta(1)$ [Sla96]: in particular, as shown in that paper, it will not be worse than $\ln n - \ln\ln n + 0.78$, and it cannot be guaranteed to be better than $\ln n - \ln\ln n - 0.31$.

Several inapproximability results for set cover have been established: to the best of our knowledge, the best lower bound for the performance ratio has been proved by Alon et al. [AMS06], who proved that set cover cannot be approximated efficiently to less than $c \ln n$, for a constant c, unless $P = NP$. Therefore, the greedy algorithm performance is close to the best one can hope for, among polynomial algorithms.

In our case, for the aforementioned reduction of minimum hitting set to set cover, given a natural pattern $\mathcal{P}$, n is equal to $max\ dom\ \mathcal{P} + 1$.

4.3 Experimental results

In the following, we show experimental results. All experiments were run on a 2GHz dual-core CPU, 4GB RAM laptop.

Comparison of exact and approximation algorithms In order to assess performance for our application, we experimented with the two aforementioned solution methods: the exact algorithm for minimum hitting set, and the translation to set cover and solution with the greedy algorithm.

We fixed $max\ dom\ \mathcal{P} = 20$; $|dom\ \mathcal{P}|$ varies. For each parameter value, we generated a random pattern and we computed a covering shape with the two methods; the performance of the approximation algorithm on each experiment is the ratio between the cardinality of the covering shape computed by the approximation algorithm, and the cardinality of the minimal covering shape, computed by the exact algorithm.

We repeated each such computation 50 times, every time with a different random pattern. Of course, performance ratio may vary for different patterns; we are interested in the average and worst-case performance. Results are shown in Table 3. In each line, we show the average and maximum performance ratio (greater ratio corresponds to worse performance), as well as the average computation times for the two algorithms.

$\lvert dom\ \mathcal{P}\rvert$	Ratio (average)	Ratio (max)	Time (exact)	Time (greedy)
2	1.0	1.0	23534.002	1.02
4	1.063	1.333	1472.15	1.204
6	1.04	1.2	251.889	1.276
8	1.032	1.333	59.132	1.744
10	1.047	1.333	25.887	1.649
12	1.0	1.0	8.152	1.992
14	1.0	1.0	7.203	2.458
16	1.0	1.0	1.852	2.152
18	1.0	1.0	2.148	2.743
20	1.0	1.0	1.776	2.173

Table 3 Comparison of the exact and greedy algorithms (times are in milliseconds). For each pattern domain's cardinality, we show the average and maximum performance ratio, and computation times for the exact and the approximation algorithms.

As expected, computation time for the exact algorithm grows exponentially with $max\ dom\ \mathcal{P} - \lvert dom\ \mathcal{P}\rvert$, which suggests it is not applicable to bigger problems. The greedy algorithm is obviously faster; and its performance ratio appears to be better than its lower bound which, for $M = 20$, would be $\ln 21 - \ln\ln 21 - 0.31 = 1.62$.

Scalability of the greedy algorithm We also performed experiments on random patterns to assess the approximation algorithm's scalability and the reduction of observation points that can be expected, for increasing values of the pattern domain's maximum and the pattern's cardinality. Results (average over 50 computations) are shown in Table 4. Note that some cells are empty because $\lvert dom\ \mathcal{P}\rvert$ cannot be greater than $max\ dom\ \mathcal{P} + 1$.

Computation times increase with both $\lvert dom\ \mathcal{P}\rvert$ and $max\ dom\ \mathcal{P}$, but they appear to scale reasonably (as is to be expected for a polynomial algorithm), as shown in Figure 4. Sampling ratios become lesser (corresponding to a bigger reduction in necessary observation points) for denser pattern domains (i.e., for smaller values of $max\ dom\ \mathcal{P} - \lvert dom\ \mathcal{P}\rvert$). For the same value of $\lvert dom\ \mathcal{P}\rvert$, sampling ratios vary slightly for different values of $max\ dom\ \mathcal{P}$.

5 Runtime verification of Linear Temporal Logic formulæ

Intuitively described, runtime verification is the problem of checking if an execution trace of some system (or a finite set of execution traces) satisfies some property [LS09]. Compared with static verification techniques such as model checking, runtime verification is computationally less costly (essentially because only the execution traces that actually occur, rather than all the possible execution traces, are considered), although, in principle, less conclusive (precisely because the results hold for the actual traces, rather than for all the possible ones). However, static and runtime verification are not necessarily mutually exclusive: a statically verified system may need to be verified at

	max dom $\mathcal{P}$				
$\lvert dom\ \mathcal{P}\rvert$	100	200	300	400	500
20	90.59%, 54	90.58%, 237	90.59%, 527	90.62%, 975	90.57%, 1480
40	94.79%, 79	94.58%, 377	94.56%, 890	94.6%, 1640	94.56%, 2666
60	96.16%, 100	96.08%, 513	96.05%, 1267	96.04%, 2365	96.04%, 3739
80	97.17%, 113	97.02%, 605	96.94%, 1557	96.89%, 2768	96.86%, 4454
100	98.02%, 114	97.52%, 641	97.4%, 1685	97.42%, 3278	97.41%, 5259
120		98.01%, 686	97.87%, 1856	97.78%, 3625	97.8%, 6013
140		98.5%, 729	98.12%, 2015	98.04%, 4010	98.03%, 6695
160		98.51%, 760	98.35%, 2149	98.27%, 4338	98.25%, 7337
180		99.0%, 795	98.66%, 2263	98.5%, 4624	98.42%, 7922
200		99.0%, 830	98.68%, 2359	98.68%, 4894	98.6%, 8437
220			98.99%, 2442	98.75%, 5117	98.75%, 8914
240			99.0%, 2516	98.91%, 5318	98.81%, 9357
260			99.08%, 2596	99.0%, 5508	98.94%, 9693
280			99.34%, 2665	99.02%, 5645	99.0%, 10041
300			99.34%, 2748	99.25%, 5803	99.02%, 10327
320				99.25%, 5948	99.2%, 10946
340				99.25%, 6487	99.2%, 12302
360				99.48%, 7144	99.2%, 11960
380				99.5%, 6523	99.39%, 11293
400				99.5%, 6515	99.4%, 11501
420					99.4%, 11717
440					99.42%, 11914
460					99.6%, 12121
480					99.6%, 12328
500					99.6%, 12584

Table 4 Greedy algorithm performance. For each combination of *max dom* $\mathcal{P}$ and $\lvert dom\ \mathcal{P}\rvert$ the percentage of observations saved corresponding to the average sampling ratio and the average computation time (in milliseconds) are shown.

runtime, for example because the assumptions used in static verification may turn out not to be verified at runtime.

Runtime verification approaches usually compute a *monitor* for each property, i.e., a procedure that tells whether a given execution trace satisfies the property. The monitor is applied while the system is executing (*online* verification) or to a finite set of observed execution traces (*offline* verification).

A common choice for a formalism to express temporal properties of a system's execution traces is temporal logics and, in particular, Linear Temporal Logics (LTL) [Pnu77]. LTL is appropriate to express significant properties of execution traces, because it provides operators for existential and universal quantification over time points, among others; for instance, it is possible to express properties such as "it is never the case that the temperature is too high" or "if the temperature was too high, the cooling system will be activated at the next time step and the temperature will be normal within three time steps".

For this reason, many runtime verification systems for properties expressed in LTL have been proposed [Bod04, GCZ08]. Recently, suitable semantics of LTL for runtime verification have been considered [BLS10], and a variant of LTL has been proposed to express properties in runtime verification [BLS11];

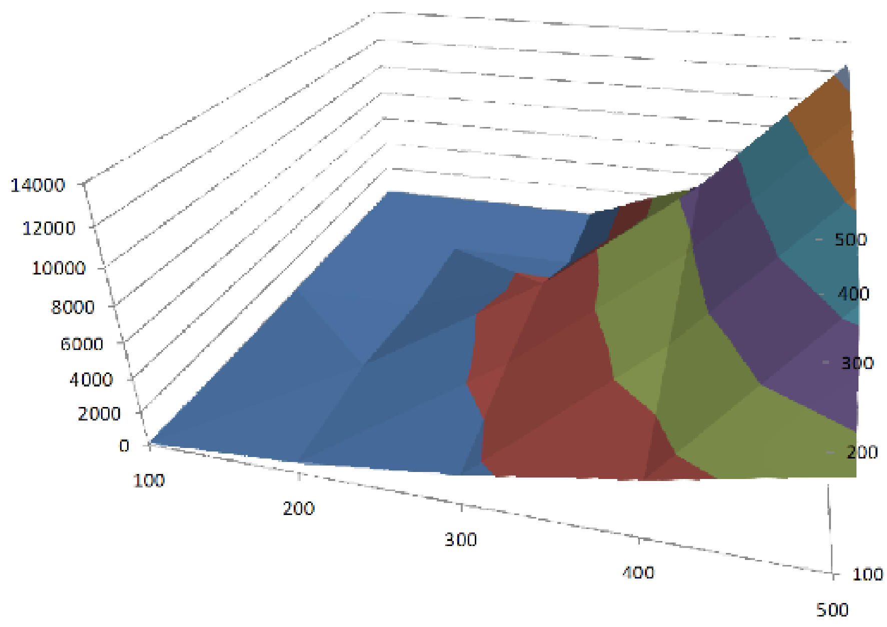

Fig. 4 Computation times, from Table 4. $|dom\ \mathcal{P}|$ is represented on the X axis, $max\ dom\ \mathcal{P}$ is on the Y axis, and computation time (in milliseconds) on the Z axis.

a related temporal formalism, I-METATEM, oriented to express properties of autonomous evolving agents, has been proposed in [CDPT09].

Event detection can be understood as a special case of runtime verification, where the property to be verified is a description of the state of affairs that defines the event, quantified existentially over time points (*diamond* operator in LTL, see Section 5.1). In the former of the previous example instances, the property would be verified if the temperature were too high at one time point at least.

To the best of our knowledge, such approaches assume observation of the whole execution traces to be given as input to the monitor. What if observation is costly, as discussed in the introduction to this article, or not even feasible? For instance, the property could be specified over a finer temporal grain than the available sampling system can support. In such cases, runtime verification could benefit from our contribution towards reducing the number of necessary observations. Even when complete execution traces are available, considering only part of them without losing certainty about property satisfaction is generally beneficial, in that it reduces computational and communication costs.

In the following, we discuss runtime verification of LTL formulæ by observing partial, rather than complete, execution traces.

We show that the results proved in Section 3 (in particular, Theorem 3) can be applied: in the special case of existential quantification over time, which, as we mentioned above, is arguably the most relevant to event detection, it is possible to draw conclusions about the complete execution trace by observing certain partial traces.

5.1 Background

In this section, we briefly recall standard definitions of syntax and semantics for LTL.

Definition 13 Let $\mathcal{A}$ be a countable set of atomic propositions, $\top$ represent truth and $\bot$ represent falsity. Then a **formula** is defined recursively as follows:

- $\top$ and $\bot$ are formulæ;
- if $\varphi \in \mathcal{A}$, φ is a formula;
- if φ is a formula, $\neg\varphi$, $\mathbf{X}\varphi$, $\Box\varphi$, and $\Diamond\varphi$ are formulæ;
- if φ and ψ are formulæ, then $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\varphi\mathbf{U}\psi$ are formulæ.

$\neg$, $\wedge$, and $\vee$ represent, respectively, negation, conjunction, and disjunction.

The informal meaning of the temporal operators is as usual in the literature. Consider a system evolving through states (formally defined below), each at one time point. Then

- $\mathbf{X}\varphi$ is the *next* operator applied to φ, true iff φ is true at the next time point;
- $\Box\varphi$ is the *necessity* of φ, true iff φ is true at all time points;
- $\Diamond\varphi$ is the *possibility* of φ, true iff φ is true at least at one time point;
- and $\varphi\mathbf{U}\psi$ (φ *until* ψ) is true iff φ is true until the first time point when ψ is true.

Formally, a formula's truth value is defined on a *path* in a *Kripke structure*.

Definition 14 A **Kripke structure** is a tuple $\langle S, S_0, R, \mathcal{A}, V\rangle$, where S is a finite **set of states**, $S_0 \subseteq S$ is a **set of initial states**, $R \subseteq S \times S$ is a **transition relation**, $\mathcal{A}$ is a countable **set of atomic propositions**, and $V : S \to 2^{\mathcal{A}}$ is a **labeling function**.

Definition 15 Given a Kripke structure $\mathcal{K} = \langle S, S_0, R, \mathcal{A}, V\rangle$, a **path** on $\mathcal{K}$ is a sequence of states $\pi = s_0 s_1 \dots s_n \dots$ such that
$\forall i \in \mathbb{N} \mid (s_{i+1} \in \pi \to \langle s_i, s_{i+1}\rangle \in R)$.

Definition 16 Given a path $\pi = s_0 s_1 \dots s_n, \dots$ on a Kripke structure $\mathcal{K}$, the k-th **postfix** of π is the sequence π^k of states $s_k s_{k+1} \dots s_n, \dots$.

Definition 17 For LTL formulæ φ and ψ, a Kripke structure
$\mathcal{K} = \langle S, S_0, R, \mathcal{A}, V\rangle$ and a path $\pi = s_0 s_1 \dots s_n$ in $\mathcal{K}$,

- if $\varphi \in \mathcal{A} \cup \{\top, \bot\}$, $\pi \models \varphi$ iff $\varphi \in V(s_0)$ or $\varphi = \top$;
- $\pi \models \varphi \vee \psi$ iff $\pi \models \varphi$ or $\pi \models \psi$;
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$;

- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$;
- $\pi \models \mathbf{X}\varphi$ iff $\pi^1 \models \varphi$;
- $\pi \models \varphi\mathbf{U}\psi$ iff $\pi \models \psi$, or
  $\exists k > 0 \mid (\pi^k \models \psi \wedge (\forall i \in [0 \,..\, k-1]\; \pi^i \models \varphi))$;
- $\pi \models \Diamond\varphi$ iff $\exists k \geq 0 \mid \pi^k \models \varphi$;
- $\pi \models \Box\varphi$ iff $\forall k \geq 0 \mid \pi^k \models \varphi$.

Definition 18 Given a state s in a Kripke structure and a formula φ, $s \models \varphi$ if and only if there exists a path π in $\mathcal{K}$ whose first state is s and $\pi \models \varphi$.

5.2 Partial paths

In this section, we introduce definitions for partial paths, which formalize the idea of incomplete execution traces. Partial paths are relevant to our work, because we want to characterize partial paths that allow to draw conclusions about the full path.

Definition 19 Given a Kripke structure $\mathcal{K}$ and a path π in $\mathcal{K}$, a **sub-path** η of π (in symbols, $\eta \sqsubset \pi$) is a sub-sequence (i.e., the sequence obtained possibly removing some states, but maintaining the order) of π; η is called a **partial path** on $\mathcal{K}$.

In general, a partial path is not a path in the original Kripke structure (whenever two consecutive states in the partial path are not in the original transition relation). Therefore, in order to give semantics to formulæ on the partial path, an associated Kripke structure can be defined as follows.

Definition 20 Let $\mathcal{K} = \langle S, S_0, R, \mathcal{A}, V\rangle$ be a Kripke structure and π a path in $\mathcal{K}$. Then, for each partial path $\eta = s_0 s_1 \ldots s_n$, with $\eta \sqsubset \pi$, the **associated Kripke structure** $\mathcal{K}_\eta = \langle S_\eta, S_{0\eta}, R_\eta, \mathcal{A}_\eta, V_\eta\rangle$ is defined, where S_η is the set of states in η, $S_{0\eta} = \{s_0\}$, $R_\eta = \{\langle s_i, s_{i+1}\rangle \mid 0 \leq i < n\}$, $\mathcal{A}_\eta = \mathcal{A}$, and V_η is the restriction of V to S_η.

The semantics of formulæ on a partial path can then be defined as in Definitions 17 and 18, where the relevant Kripke structure is the one associated with the partial path.

Definition 21 Given a Kripke structure $\mathcal{K}$, a path $\pi = s_0 s_1 \ldots s_n \ldots$ on $\mathcal{K}$, and a finite set $\mathcal{I}$ of natural numbers whose maximum is M, π's **sampling** of shape $\mathcal{I}$, $\pi_\mathcal{I}$, is the sequence of states s_i such that s_i is a state in π and, for some $k \in \mathbb{N}$ and $q \in \mathcal{I}$, $i = q + k(M+1)$.

Example 10 The sampling of a path $\pi = s_0 s_1 \ldots s_{16}$[3] of shape $\mathcal{I} = \{0, 1, 3, 6\}$ is $s_0 s_1 s_3 s_6 s_7 s_8 s_{10} s_{13} s_{14} s_{15}$. Using the symbols in Definition 21, $M = 6$, and the index i of each state in the sampling results from a choice of $q \in \mathcal{I}$ and $k \in \mathbb{N}$, as shown in this table:

[3] Note that paths in Kripke structures may be finite, depending on the transition relation.

q	k	i
0	0	0
1	0	1
3	0	3
6	0	6
0	1	7
1	1	8
3	1	10
6	1	13
0	2	14
1	2	15

Graphical representation of paths In the following, we represent paths graphically: a state s is represented as a node with label s; the set of atomic propositions satisfied by each state is written above the corresponding node; solid edges represent the sequence in the full path, and dashed edges represent the sequence in the partial path.

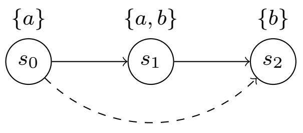

Fig. 5 Example path representation.

For example, in Figure 5 the complete path is composed of the states s_0, s_1 and s_2, while the partial path is composed of the states s_0 and s_2. State s_0 satisfies the atomic proposition a, s_1 satisfies a and b, and s_2 satisfies b.

5.3 Formula entailment in partial and full paths

In this section, we discuss logical relations between entailment of a LTL formula in full and partial paths.

We consider the fundamental LTL operators ($\Box$, $\Diamond$, $\mathbf{X}$, $\mathbf{U}$); we prove general positive results for the (few, as is to be expected) cases where it is possible, and provide counterexamples for the other cases.

Box operator ($\Box$) If a formula is always true in a full path, it will be true in any partial path.

Theorem 4 *If $\eta \sqsubset \pi$, then $\pi \models \Box\varphi \to \eta \models \Box\varphi$.*

Proof Let s be a state in η; then, by $\eta \sqsubset \pi$, $s \in \pi$. By hypothesis, $s \models \varphi$. The same holds for any state in η.

The opposite does not hold:

$$\begin{tikzcd}
\{a\} & \{\} & \{a\} \\
s_0 \arrow[r] \arrow[rr, dashed, bend right] & s_1 \arrow[r] & s_2
\end{tikzcd}$$

$\eta \models \Box a$, but $\pi \not\models \Box a$.

Diamond operator ($\Diamond$) If a formula is true in at least one state in a partial path, it will be true in at least one state in the full path.

Theorem 5 *If $\eta \sqsubset \pi$, then $\pi \models \Diamond\varphi \Leftarrow \eta \models \Diamond\varphi$.*

Proof Let s be the state in η such that $s \models \varphi$(such a state exists by hypothesis). Since $\eta \sqsubset \pi$, $s \in \pi$; i.e., $\exists s \in \pi \mid s \models \varphi$.

The opposite does not hold:

$$\begin{tikzcd}
\{a\} & \{a,b\} & \{a\} \\
s_0 \arrow[r] \arrow[rr, dashed, bend right] & s_1 \arrow[r] & s_2
\end{tikzcd}$$

$\pi \models \Diamond b$, but $\eta \not\models \Diamond b$.

Next operator ($\mathbf{X}$) No general result can be proved about $\mathbf{X}$: if a formula is true in the next state in a full path, it might not be in the next state in a partial path, and vice-versa, as exemplified in the following.

$$\begin{tikzcd}
\{\} & \{a\} & \{b\} \\
s_0 \arrow[r] \arrow[rr, dashed, bend right] & s_1 \arrow[r] & s_2
\end{tikzcd}$$

– $\pi \models \mathbf{X}a$, but $\eta \not\models \mathbf{X}a$
– $\eta \models \mathbf{X}b$, but $\pi \not\models \mathbf{X}b$.

Until operator ($\mathbf{U}$) In general, no implication can be proved about $\mathbf{U}$: φ can be true until ψ in the full path but not in a partial path, and vice-versa.

$$\begin{tikzcd}
\{a,b\} & \{b\} & \{b,c\} & \{d\} \\
s_0 \arrow[r] \arrow[rr, dashed, bend right] & s_1 \arrow[r] & s_2 \arrow[r] & s_3
\end{tikzcd}$$

– $\pi \models b\mathbf{U}d$, but $\eta \not\models b\mathbf{U}d$
– $\eta \models a\mathbf{U}c$, but $\pi \not\models a\mathbf{U}c$

5.4 Event detection in LTL

The positive results in Section 5.3 (Theorems 4 and 5) only cover a small portion of the possible LTL formulæ. The counter-examples provided for the other logical relations between entailment in partial and full paths exclude that conclusions about the full path can be drawn from the partial path, in the general case.

Of course, stronger results can be proved in special cases.

One significant such case is an application where (i) it is of interest to detect if a formula φ is entailed by at least one state in a path (that is, to verify $\Diamond\varphi$[4]), and (ii) it is known that φ's truth follows a given pattern (e.g., φ is entailed by a number of consecutive states, or it is entailed by a state, then by the following state, then by the state that comes after four more states). In this case, the result of Theorem 3 can be applied.

First, we define formally what it means for a LTL formula's truth to follow a given (natural) pattern. For this case, the value set is not relevant; the formula is true in all states whose position in the path is an element of the pattern domain, plus an offset.

Definition 22 Given a natural pattern $\mathcal{P}$, a formula φ is an **LTL-event** of pattern $\mathcal{P}$ in a Kripke structure $\mathcal{K}$ if and only if for any path $\pi = s_0 s_1 \ldots s_n \ldots$ in $\mathcal{K}$,
$(\exists k \mid s_k \models \varphi) \rightarrow \exists \tau (\forall j \in \mathit{dom}\,\mathcal{P} \mid s_{\tau+j} \models \varphi)$

The following results allow to verify the occurrence of a LTL formula by observing a sampling of a path, rather than the full path.

Theorem 6 *Let φ be a LTL-event of pattern $\mathcal{P}$ on a Kripke structure $\mathcal{K}$. Let $\mathcal{S}_C$ be a covering shape of $\mathcal{P}$'s. Then, for each path $\pi = s_0 s_1 \ldots s_n \ldots$ on $\mathcal{K}$, $\pi \models \Diamond\varphi$ if and only if $\pi_{\mathcal{S}_C} \models \Diamond\varphi$, where $\pi_{\mathcal{S}_C}$ is π's sampling of shape $\mathcal{S}_C$ (Definition 21).*

Proof If: by Theorem 5, because $\pi_{\mathcal{S}_C}$ is a subpath of π.

Only if: $\pi \models \Diamond\varphi$, so $\exists m \mid s_m \models \varphi$. By Definition 22,
$\exists i \mid (\forall j \in \mathit{dom}\,\mathcal{P} \mid s_l \models \varphi, l = i + j)$. Such ls are the observability set of an event of pattern $\mathcal{P}$ and offset i (see Definition 5). Since $\mathcal{S}_C$ is $\mathcal{P}$'s covering shape, by Theorem 3 the set $\mathcal{C} = \{q + k(M+1) \mid q \in \mathcal{S}_C \wedge k \in \mathbb{N}\}$, where $M = \max \mathcal{S}_C$, intersects the observability sets of all the events of shape $\mathcal{P}$, that is, it contains at least one of the aforementioned ls.

Summarizing, there exists some l with the following properties: (i) $s_l \in \pi$, (ii) $l = q + k(M+1)$, for some $q \in \mathcal{S}_C$ and $k \in \mathbb{N}$, and (iii) $s_l \models \varphi$. But properties (i) and (ii) define indices of states that belong to $\pi_{\mathcal{S}_C}$, so $s_l \in \pi_{\mathcal{S}_C}$; and because of property (iii), $\pi_{\mathcal{S}_C} \models \Diamond\varphi$.

Example 11 Suppose that a formula φ is known to remain true for 5 consecutive states. Then it is a LTL-event of pattern
$\mathcal{P} = \{\langle 0,1\rangle, \langle 1,1\rangle, \langle 2,1\rangle, \langle 3,1\rangle, \langle 4,1\rangle\}$. Since $\{0\}$ is a covering shape of $\mathcal{P}$'s (to see why, consider Example 5, with $M = 1$), to verify $\Diamond\varphi$ on a path $s_0 s_1 \ldots s_n \ldots$, it is sufficient to observe the states in the set $\{s_{5k} \mid k \in \mathbb{N}\}$.

Example 12 If we represent as the LTL formula φ the fact that the button of Example 6 is being pressed, then it is an event of pattern
$\{\langle 0,1\rangle, \langle 1,1\rangle, \ldots \langle 999,1\rangle\}$. By Theorem 6, $\Diamond\varphi$, which represent in LTL the fact that the button has been pushed at least once, can be verified observing the system at states in a path $s_0 s_1 \ldots s_n \ldots$ whose index is a multiple of 1000.

[4] If the formula is understood as a logical representation of an event, this amounts to verifying the event's occurrence.

Example 13 Suppose that a is true at a state if and only if b is true 2 states later, and if and only if c is true 5 states later. Then the formula $\varphi = a \vee b \vee c$ is a LTL-event of pattern $\mathcal{P} = \{\langle 0,1\rangle, \langle 2,1\rangle, \langle 5,1\rangle\}$.

By Theorem 6 and the results of Example 7, $\{0,1,3\}$ is a covering shape of $\mathcal{P}$'s. Then, to detect that a or b or c become true at a state (that is, to verify $\Diamond\varphi$), it is sufficient to observe the system at states s_0, s_1, s_3, s_6, s_7, s_9 and so on.

Example 14 In the scenario of Example 9, the formula φ representing that the button is being pushed is an LTL-event of pattern $\mathcal{P}$, where $\mathit{dom}\ \mathcal{P} = \{0, 100, 200, 300, 400, 500, 600, 700, 800, 900\}$. Then, by Theorem 6, and exploiting the result of Example 9, $\Diamond\varphi$ can be verified by observing only 1/10th of the states in any path.

Example 15 Consider the example in [GCZ08], where the output phase must be followed (possibly not immediately) by an input phase. The property to be verified can be written in LTL as $\Box(\mathit{SyncOutput} \rightarrow \Diamond\mathit{SyncInput})$, which is equivalent to $\neg\Diamond\neg(\mathit{SyncOutput} \rightarrow \Diamond\mathit{SyncInput})$, or
$\neg\Diamond(\mathit{SyncOutput} \wedge \neg\Diamond\mathit{SyncInput})$

Therefore, if the formula $\varphi = \mathit{SyncOutput} \wedge \neg\Diamond\mathit{SyncInput}$ is detected true at at least one state, the property does not hold.

Suppose that the output phase lasts for at least M consecutive states, and that only afterwards can the input be on. Therefore, if φ is true at a state, it is true for at least M consecutive states, and, by Definition 22, φ is an LTL-event of pattern $\mathcal{P}$, with $\mathit{dom}\ \mathcal{P} = \{0, 1, \ldots, M-1\}$. Since, by the result in Example 5, $\{0\}$ is a covering shape for $\mathcal{P}$, $\Diamond\varphi$ can be detected observing the system only every M-th state.

6 Beyond plain event detection: future research lines

In this section, we discuss possible generalizations and extensions of the research issues presented in this paper.

6.1 Generalizations

Multi-dimensional patterns The definition of coordinate system in Section 2.1 is very general. In this paper, we focussed and proved results on one-dimensional natural patterns, but other classes of coordinate systems can be considered. For instance, it seems reasonable that the results of this paper can be generalized to multi-dimensional natural coordinate systems, which could represent more complex spatial-temporal patterns.

Multiple events We assumed that applications require detection of events of only one pattern. In future work, we want to generalize to the case of different possible patterns. In this circumstance, we can distinguish two cases: one where the value sets of the patterns are disjoint (so an event can be immediately recognized), and the other where, once an event is detected, more observations are required in order to determine its pattern. Which further observations best discriminate amongst the still standing competing patterns is a germane issue, which can benefit from existing probabilistic approaches, such as Bayesian networks, as already applied to intention recognition in [AP10].

6.2 Complex events

In this paper, we considered simple, atomic events with a given pattern. Detection of complex events (composed of atomic ones) can benefit from logic programming techniques, as we argue in the following.

Representation of complex events An *episode* has been defined in the literature [MTV97] as “a collection of events that occur relatively close to each other in a given partial order.”

More generally, complex events can be composed from simple ones by conjunction, disjunction, sequence or negation (as in event algebras [CL04]). A suitable approach would be to describe a complex event by means of a grammar, for instance DCGs [Per81], well supported in logic programming, allowing for a more compact and declarative representation, supporting recursiveness, and incremental processing. Also, if an observed sub-pattern has a certain form, the grammar might immediately distinguish what are the full pattern candidates still possible, i.e., still under consideration according to the grammar, and even make proactive observations looking for events that discriminate between the still outstanding patterns, e.g. to rule out patterns. For instance, if a pattern is known to be symmetric and with a certain length, the grammar can predict what should come as the second half of the pattern given the first half, and rule out the pattern if the symmetry is broken. Also an LP grammar can tie recognition to other elaborate LP techniques, including the semantic compilation of justifications of pattern detection, say for debugging.

This brings out the opportunity to employ model-based diagnosis techniques in general, namely efficient kernel diagnosis algorithms that detect maximal intersections of hitting sets [dKMR92], and allow for detection of multiple persistent and intermittent diagnosis [dK09].

6.3 Focussing detection efforts

Tools and techniques developed in the field of logic programming can be used to guide event detection at a higher level, determining which atomic events to look for in an execution trace, thereby further saving observation effort.

Application of Abductive Logic Programming In particular, approaches based on abductive logic programming [KKT93] and its efficient implementation [APS04] appear appropriate, as abduction can be used to formulate hypotheses on the state of the system at points not yet observed.

In applications where more than one type of event can occur, the occurrence of each event can be represented by an abducible predicate. It can be of interest to express permissible combinations of events, which may be expressed by means of integrity constraints; or to express relations of expectancy of an event depending on the detection of another.

For instance, the $\mathcal{S}$CIFF abductive logic language [ACG$^+$08] supports special predicate $\mathbf{H}(a)$, meaning that event a happened, and abducible predicates $\mathbf{E}(a)$ and $\mathbf{EN}(a)$, meaning, respectively, that event a is expected to happen, or expected not to happen. Integrity constraints as $\mathbf{H}(a) \rightarrow \mathbf{E}(b) \vee \mathbf{E}(c)$ can be used as a guidance mechanism to perform detection: a's occurrence would trigger an attempt to detect the occurrence of b or c, in order to satisfy the integrity constraint.

Side-effects More generally, events sometimes have tell-tale side-effects, but which are not guaranteed to happen – i.e., the side-effects either don't always follow if the event happens, or they may be side-effects of only a subset of the whole event, so that while the side-effects may actually happen (and even then with a probability) the complete event might not be there.

Also, if detected they may assure us the event occurred (though not being formally part of the event) or simply allow to hypothesize (abduce) that event or some other event. The classical example in this case, framed using abduction, is "the grass is wet": did it rain during the night or was the sprinkler left on or both? On the other hand, it may have rained but the grass no longer being wet.

These tell-tale side-effects can be used heuristically as triggers to direct attention, and speed up event detection by going for the more likely events first, given the side-effects. A Bayesian net could be employed to code such heuristic and probabilistic information. In particular, side-effects can be tell-tale signs of intentions, which in turn lead to actions and events. This connects the event detection issues to intention recognition, combining Bayesian nets with plan recognition, and combining uncertainty in KR and rule KR in general [PA10, PA09]. [KNF03] reports on a logic programming based system which infers team of agents' intentions from traces of agents' action events and observations.

Similar to side-effects, in a mirror-like way, one may have premonitions, that is tell-tale prior-effects that may be omens of (sub-)events to follow, for which all of the above can likewise be re-stated, with consequences for heuristics leading to faster or priority event recognition.

The importance of the tell-tale signs, prior or posterior, can be evaluated for significance and preferential compatibility. A case in point might be the tell-tale signs concerning a possible natural disaster or deliberate attack, with consequences for preventive or palliative measures. This raises the issue of how can events be countered or forestalled. On the other hand, positive events might be leveraged for greater benefit.

In summary, the topic of runtime verification and event detection can benefit from declarative logic programming because of the natural use of logic programming and abduction for representing observations and actions, as well as its implementation tools in general, in order to support the topic's ramifications and applications. The latter two concern not just opening declarative programming to the new areas of monitoring and control, but also the very use of runtime verification in program correctness and runtime safety.

6.4 Further developments

Finally, we mention developments of our research that we would like to pursue in the long-term.

Tradeoff between accuracy and observation cost In case detection of all events is not critical, one may consider further reducing the number of observations, and estimate the probability of missing an event; decision theory can be applied to find a trade-off between the costs of observation and of failed detection.

Uncertainty In case execution traces are being observed, but there is uncertainty about observations, predictive lookahead can be used to remove uncertainty. Same goes, regarding uncertainty removal, for retrospective hindsight (by making observations about the past, cf. astronomy) in order to confirm or disconfirm hypotheses about traces. Feature extraction techniques, such as the wavelet-based approach of [WA01], can be used to de-noise and cluster uncertain data.

Synthesis If the system being discussed can be controlled (such as a logic program that can be evolved [DLP06]), the problem can be viewed in the other direction, i.e., how to update the system in order to have it generate the desired traces.

Such problems can have more than one solution. Preferences (now well understood in the context of logic programming [PDL09]) can be used to rank among solutions, or as a heuristics along with abduction or evolution [DP07].

Implementation Thanks to the recent advances in logic programming technology, one may also think about trying to confirm and disconfirm an event at the same time (using multi-threaded computations) and take decisions depending on the first goal that succeeds.

7 Conclusions

In this paper, we considered the problem of reducing the number of observations necessary for event detection when the event's pattern is known, by design or by previous discovery.

We showed that, when the observation space can be modeled as the set of natural numbers, the number of observations required can be significantly reduced. In this case, the problem can be reduced to a NP-hard problem, but we showed that a polynomial approximation algorithm with good (theoretical and experimental) performance can be applied.

We showed too how the result can be applied to runtime verification of temporal logic formulæ.

Finally, we also discussed possible extensions and generalizations of the problem, and argued for the use of logic programming techniques and systems to handle increased problem sophistication.

Acknowledgments

We thank the anonymous reviewers for valuable comments on previous versions of this article.

References

ACG$^+$08. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: the $\mathcal{S}$CIFF framework. *ACM Transactions on Computational Logic (TOCL)*, 9(4), 2008.

AMS06. Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

AP10. Han The Anh and Luís Moniz Pereira. Collective intention recognition and elder care. In *Proactive Assistant Agents: Papers from the AAAI Fall Symposium (FS-10-07)*, pages 26–31. AAAI, November 2010.

APS04. José Júlio Alferes, Luís Moniz Pereira, and Terrance Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming*, 4(4):383–428, 2004.

BLS10. Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *J. Log. and Comput.*, 20:651–674, June 2010.

BLS11. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology*, 20(4), 2011.

Bod04. Eric Bodden. A lightweight LTL runtime verification tool for Java. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*, pages 306–307. ACM, October 2004. ACM Student Research Competition.

CDP08. Stefania Costantini, Pierangelo Dell'Acqua, and Luís Moniz Pereira. A multi-layer framework for evolving and learning agents. In A. Raja M. T. Cox, editor, *Proceedings of Metareasoning: Thinking about thinking workshop at AAAI 2008, Chicago, USA*, 2008.

CDPT09. Stefania Costantini, Pierangelo Dell'Acqua, Luís Moniz Pereira, and Pangiota Tsintza. Runtime verification of agent properties. In *Proc. of the Int. Conf. on Applications of Declarative Programming and Knowledge Management (INAP09)*, 2009.

CDPT10. Stefania Costantini, Pierangelo Dell'Acqua, Luís Moniz Pereira, and Francesca Toni. Learning and evolving agents in user monitoring and training. In *Atti del Congresso Nazionale AICA 2010*, L'Aquila, Italy, September-October 2010. Available at `http://centria.di.fct.unl.pt/~lmp/publications/slides/aica10/aica-2010.pdf`.

CL04. Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 147–154, New York, NY, USA, 2004. ACM.

CTTT07. Stefania Costantini, Arianna Tocchio, Francesca Toni, and Pangiota Tsintza. A multi-layered general agent model. In *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing, 10th Congress of the Italian Association for Artificial Intelligence*, LNCS 4733. Springer-Verlag, Berlin, 2007.

dK09. Johan de Kleer. Diagnosing multiple persistent and intermittent faults. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 733–738, 2009.

dKMR92. Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artif. Intell.*, 56(2-3):197–222, 1992.

DLP06. Pierangelo Dell'Acqua, Anna Lombardi, and Luís Moniz Pereira. Modelling adaptive controllers with evolving logic programs. In Juan Andrade-Cetto, Jean-Louis Ferrier, José Dias Pereira, and Joaquim Filipe, editors, *ICINCO 2006, Setúbal, Portugal*. INSTICC Press, 2006.

DP07. Pierangelo Dell'Acqua and Luís Moniz Pereira. Preferential theory revision. *J. Applied Logic*, 5(4):586–601, 2007.

FBHT07. Michael Fisher, Rafael H. Bordini, Benjamin Hirsch, and Paolo Torroni. Computational logics and agents: a road map of current technologies and future trends. *Computational Intelligence Journal*, 23(1):61–91, 2007.

GCZ08. Heather J. Goldsby, Betty H. Cheng, and Ji Zhang. Amoeba-rt: Run-time verification of adaptive software. In Holger Giese, editor, *Models in Software Engineering*, pages 212–224. Springer-Verlag, Berlin, Heidelberg, 2008.

GS99. Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 33–42, New York, NY, USA, 1999. ACM.

HKM$^+$96. Kimmo Hätönen, Mika Klemettinen, Heikki Mannila, Pirjo Ronkainen, and Hannu Toivonen. Knowledge discovery from telecommunication network alarm databases. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE '96, pages 115–122, Washington, DC, USA, 1996. IEEE Computer Society.

HSB09. Annika Hinze, Kai Sachs, and Alejandro Buchmann. Event-based applications and enabling technologies. In *DEBS '09: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pages 1–15, New York, NY, USA, 2009. ACM.

Iye04. Vijay S. Iyengar. On detecting space-time clusters. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 587–592, New York, NY, USA, 2004. ACM.

JDtL06. Ying Jin, Jing Dai, and Chang tien Lu. Spatial-temporal data mining in traffic incident detection. In *In Proc. SIAM DM 2006 Workshop on Spatial Data Mining*, 2006.

KA86. H. Kautz and J.F. Allen. Generalized plan recognition. In *Proceedings of the Conference of the American Association of Artificial Intelligence (AAAI-86)*, pages 32–38. AAAI, 1986.

KKT93. A. C. Kakas, R. A. Kowalski, and Francesca Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.

KNF03. Taro Kanno, Keiichi Nakata, and Kazuo Furuta. A method for team intention inference. *Int. J. Hum.-Comput. Stud.*, 58(4):393–413, 2003.

LP06. Gonçalo Lopes and Luís Moniz Pereira. Prospective programming with acorda. In *Empirically Successful Computerized Reasoning (ESCoR'06) workshop at The 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, Seattle, USA, August 2006.

LS09. Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5):293 – 303, 2009.

MTV97. Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining And Knowledge Discovery*, 1:259–289, 1997.

NMSD05. Daniel B. Neill, Andrew W. Moore, Maheshkumar Sabhnani, and Kenny Daniel. Detection of emerging space-time clusters. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 218–227, New York, NY, USA, 2005. ACM.

PA09. Luís Moniz Pereira and Han The Anh. Intention recognition via causal bayes networks plus plan generation. In L. Seabra Lopes, Nuno Lau, P. Mariano, and L. Rocha, editors, *14th Portuguese Intl.Conf. on Artificial Intelligence*, volume 5816 of *LNAI*, pages 138–149. Springer, October 2009.

PA10. Luís Moniz Pereira and Han The Anh. Intention recognition with evolution prospection and causal bayesian networks. In Ana Madureira, Judite Ferreira, and Zita Vale, editors, *Computational Intelligence for Engineering Systems: Emergent Applications*, volume 46 of *Intelligent Systems, Control and Automation: Science and Engineering Book Series*, pages 1–33. Springer, December 2010. Select updated papers from Intl. Symp. on Computational Intelligence for Engineering Systems. http://www.amazon.ca/Computational-Intelligence-Engineering-Systems-Applications/dp/940070092X.

PDL09. Luís Moniz Pereira, Pierangelo Dell'Acqua, and Gonçalo Lopes. On preferring and inspecting abductive models. In Andy Gill and Terrance Swift, editors, *Practical Aspects of Declarative Languages, 11th International Symposium, PADL 2009, Savannah, GA, USA, January 19-20, 2009. Proceedings*, volume 5418 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2009.

Per81. Fernando Pereira. Extraposition grammars. *Computational Linguistics*, 7:243–256, October 1981.

PL07. Luís Moniz Pereira and Gonçalo Lopes. Prospective logic agents. In J. Maia Neves, M. F. Santos, and J. M. Machado, editors, *Progress in Artificial Intelligence, Procs. 13th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'07)*, LNAI 4874, pages 73–86, Guimarães, December 2007. Springer.

Pnu77. Amir Pnueli. The temporal logics of programs. In *Proceedings of the 18th Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

PR10. Luís Moniz Pereira and Carroline D. P. Kencana Ramli. Modelling decision making with probabilistic causation. *Intelligent Decision Technologies*, 4(2):133–148, 2010. http://centria.di.fct.unl.pt/ lmp/publications/online-papers/IDT-Probabilistic-Causation.pdf.

PT96. Balaji Padmanabhan and Alexander Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *roceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 351–354, 1996.

SCC05. Simon Sheu, Chang-Yeng Cheng, and Alan Chang. Fast pattern detection in stream data. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications - Volume 1*, AINA '05, pages 125–130, Washington, DC, USA, 2005. IEEE Computer Society.

Sla96. Petr Slavík. A tight analysis of the greedy algorithm for set cover. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 435–441, New York, NY, USA, 1996. ACM.

WA01. M. Wu and H. Adeli. Wavelet-neural network model for automatic traffic incident detection. *Mathematical and Computational Applications*, 6(2):85–96, 2001.