LogLogics: A logic for history-dependent business processes

Kees van Hee, Olivia Oanea, Alexander Serebrenik, Natalia Sidorova*, Marc Voorhoeve

Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Received 15 April 2006; received in revised form 1 July 2006; accepted 1 August 2006
Available online 18 October 2006

Abstract

Choices in business processes are often based on the process history saved as a log-file listing events and their time stamps. In this paper we introduce LogLogics, a finite-path variant of the Timed Propositional Temporal Logic with Past, which can be in particular used for specifying guards in business process models. The novelty is due to the presence of boundary points corresponding to the starting and current observation points, which gives rise to a three-valued logic allowing us to distinguish between temporal formulas that hold for any log extended with some possible past and future (true), those that do not hold for any extended log (false) and those that hold for some but not all extended logs (unknown). We reduce the check of the truth value of a LogLogics formula to a check on a finite abstraction and present an evaluation algorithm. We also define LogLogics patterns for commonly occurring properties.

Keywords: History-dependent processes; Log; Temporal logics; Business processes; Workflow

1. Introduction

Case management systems are an important and generic class of Enterprise Information Systems (EIS). An essential part of a case management system is a workflow management system (WfMS) [2] that takes care of the distribution of work to agents, which can be either human beings or application software. Decisions taken by a WfMS can depend on previous observations. For instance, a bank can propose more interesting loan conditions to those customers who paid off the previous loans on time. We call processes executed by such a WfMS history-dependent processes. Importance of history-based decisions in workflow management has been recognised in the past [18,19]. In history-dependent processes, actions can be guarded by conditions on the process history.

Although history-dependent processes are omnipresent in industrial EISs, only a few models (partially) support them [1,8]. Since a temporal logic is a natural way to express dependencies between the events observed in the history, those works are based on temporal logics. However, finiteness of the history at any given moment of time and,
hence, the inherent incompleteness of observations, should be taken into account, which is not quite adequately done in those works, since their formulas can be evaluated to true or false only.

To illustrate the resulting limitations consider a guard saying that every bill was paid within four weeks and a log documenting an unpaid bill issued two weeks ago. Following [1,8] this guard is evaluated to false, since there is a bill which is not paid yet. However, the payment term has not expired yet and we do not intend to blacklist the client whom the bill was sent to. Instead we would like to obtain unknown in this case, true in the case every bill was paid and it happened within four weeks, and false if there is a bill issued more than four weeks ago that was not paid on time. In some cases the WfMS takes a decision on the continuation of the process giving the benefit of the doubt, i.e. unknown leads to the same choice as true; in other cases unknown leads to the same choice as false, and in a number of cases evaluating a guard to unknown leads to enabling a special procedure to handle the case.

In this paper, we propose a temporal logic, called LogLogics, that overcomes the above limitation by reasoning with three truth values. A number of three-valued (untimed) temporal logics, including L-TL, have been proposed by Nakamura [16] and investigated in [15]. Similarly to L-TL, if a LogLogics-formula is evaluated to true or false at a given time point, this value cannot be changed in the future, while unknown can become true or false. Unlike L-TL, not every LogLogics-formula has to be eventually evaluated to true or false.

Since history is a finite linear sequence of timed events, we base LogLogics on linear timed temporal logics (defined on infinite sequences) that have been the subject of intensive research in the past, starting with [3,4,12]. More recent works on the subject include [5,20]. Due to the nature of history, we need to consider not only future but also past temporal operators. Therefore, we have chosen to adapt the Timed Propositional Logic with Past (TPTL+Past) [3,4].

An alternative to TPTL+Past might have been the metric timed logic (MTL) [12,20]. The reasons for opting for TPTL+Past rather than for MTL are twofold. First of all, TPTL is “more temporal”: it uses real clocks to express timed constraints. This allows us to express such common for EIS constraints as “event p occurred between January 1, 2005 and January 1, 2006”. Unlike TPTL, MTL reasons in terms of distances between events. Hence, in order to express the same constraint we need to introduce a special event q that occurred on January 1, 2005 and require that p followed q within one year. Second, as recently shown in [5], TPTL+Past is strictly more expressive than MTL+Past.

Two different semantics for timed temporal logics can be considered: point-wise semantics, where formulas are evaluated over discrete sequences of timed events, and interval-based semantics, where formulas are evaluated over the continuous time line [17]. We believe that discrete sequences of timed events, which are actually contained in logs, are better suited for specifying history-based guards in business processes and we choose the point-wise semantics.

We define a LogLogics-formula to be true for some word (log) ρ if it holds for all words containing ρ as a subword, i.e. for the log with all possible pasts and futures. A formula is evaluated to false if it does not hold for the log with any of the possible pasts and futures and unknown if it holds for the log with some but not all possible pasts and futures. Although defined in terms of infinitely many possible pasts and futures, checking the truth values of a LogLogics-formula can be reduced to checking the truth value of the formula on a finite abstraction. We list a number of patterns of commonly occurring guards in business processes and show how these patterns can be expressed in LogLogics.

The remainder of the paper is organised as follows. In Section 2 we present LogLogics. In Section 3 we introduce a finite abstraction that leads us to an evaluation algorithm presented in Section 4. In Section 5 we show some patterns expressed in our logics. In Section 6 we present directions for future research.

2. LogLogics

In this section we present LogLogics, which aims at the modelling of history-dependent processes based on log-files. Log-files record series of events such as “100 euro has been withdrawn from account X”, “a transaction has failed”, “loan Y has been determined to be uncollectible”. The set of all events possible in the system is denoted by Σ.

LogLogics is an adaptation of the Next-Free Timed Propositional Temporal Logic with Past [4,5] to finite sequences of events limited by two special points that refer to the beginning and the end of observations. While the absolute begin is well-suited for modelling the behaviour of software systems that have been invoked at some moment of time, it is less appropriate for business processes, where the observations could be available for a recent period of time only. Similarly, there is the last time point where observations are available.

Due to the finiteness of observations, the values of traditional temporal operators can become unknown. Consider, for instance, a predicate p that is true if a client is reliable. However, the fact that during the entire period of observations the client was reliable does not necessarily imply that “always reliable” is true. Nor, in fact does it imply
that “always reliable” is false. Indeed, there are two distinct possible futures: one where “always reliable” is true, and another one where “always reliable” is false. In such a situation we would like to say that the value of “always reliable” is unknown. To formalise this intuition we start by recapitulating definitions of the well-known Next-Free Timed Propositional Temporal Logics with Past and then define a semantics for finite traces.

We assume that a countable set $P$ of atomic propositions and a countable set $V$ of clock variables are given. Then, formulas $\phi$ are built from atomic propositions, boolean connectives, “until” $\mathcal{U}$ and “since” $\mathcal{S}$ operators, clock constraints and clock resets. Intuitively, $\phi_1\mathcal{U}\phi_2$ means that at some time point in the future an event happens for which $\phi_2$ holds and for all events happened before that event, $\phi_1$ holds. Similarly, $\phi_1\mathcal{S}\phi_2$ means that at some point of time in the past an event happens for which $\phi_2$ holds and from that point onwards $\phi_1$ holds. Finally, clock reset $x.\phi$, also known as “freeze”, sets the value of clock $x$ to the current time. Formally:

**Definition 1.** Formulas $\phi$ of LogLogics are inductively defined as:

$$\phi := p \mid x \sim y + c \mid x \sim c \mid x.\phi \mid false \mid \neg\phi \mid \phi_1 \land \phi_2 \mid \phi_1\mathcal{U}\phi_2 \mid \phi_1\mathcal{S}\phi_2,$$

where $x, y \in V, p \in P, \sim$ is one of $<, >, \leq, \geq, =, \neq$ and $c \in \mathbb{N}$.

We also assume that the abbreviations $\lor, \Rightarrow, \Leftrightarrow, true$ are defined as usual.

In order to define the formal semantics of LogLogics we introduce time sequences and timed words.

**Definition 2.** A finite time sequence $\tau = \tau_k\tau_{k+1}\ldots\tau_n$ with $k, n \in \mathbb{Z}$ is a finite sequence of times $\tau_i \in \mathbb{Z}$, $i \in \{k, \ldots, n\}$ such that $\tau_i \leq \tau_{i+1}$ for all $i \in \{k, \ldots, n-1\}$.

An infinite time sequence $\tau = \tau_k\tau_{k+1}\ldots$ with $k \in \mathbb{Z}$, is an infinite sequence of times $\tau_i \in \mathbb{Z}$ such that $\tau_i \leq \tau_{i+1}$ for all $i \geq k$.

A finite event sequence $\sigma = \sigma_k\sigma_{k+1}\ldots\sigma_n$ with $k, n \in \mathbb{Z}$ is a finite sequence of events $\sigma_i \in \Sigma, i \in \{k, \ldots, n\}$. For any atomic proposition $p \in P$ and any $i \in \{k, \ldots, n\}$, $\sigma_i \vdash p$ is either true or false.

An infinite event sequence $\sigma = \sigma_k\sigma_{k+1}\ldots$ with $k \in \mathbb{Z}$ is an infinite sequence of events $\sigma_i \in \Sigma, i \geq k$. For any atomic proposition $p \in P$, $\sigma_i \vdash p$ can be evaluated to true, or false.

A finite timed word $\rho = (\sigma, \tau)$ is a pair consisting of an event sequence $\sigma$ and a time sequence $\tau$ of the same length. We also write a timed word as a sequence of pairs $(\sigma_k, \tau_k)\ldots(\sigma_n, \tau_n)$.

An infinite timed word $\rho = (\sigma, \tau)$ is a pair consisting of an infinite event sequence $\sigma$ and an infinite time sequence $\tau$.

Note that dates are usual time stamps for business processes (i.e. the exact time is not necessarily indicated in the log), which naturally implies that multiple events can have the same time stamp. Still, also the events with equal time stamps remain ordered and can be in fact causally dependent.

We use the standard semantics of TPTL+Past for infinite traces. For the sake of brevity, given a set $S$ and a predicate $\pi$, we write $\exists x : x \in S : \pi(x)$ and $\forall x : x \in S : \pi(x)$ to denote $\exists x (x \in S \land \pi(x))$ and $\forall x (x \in S \Rightarrow \pi(x))$, respectively.

**Definition 3.** Let $\rho$ be an infinite timed word. Let $i \in \mathbb{Z}$ and $v : V \rightarrow \mathbb{Z}$ be a partial valuation for the clock variables. Then

- $\langle \rho, i, v \rangle \models p$ is equal to $\sigma_i \vdash p$;
- $\langle \rho, i, v \rangle \models false$ is false;
- $\langle \rho, i, v \rangle \models x \sim c$ iff $v(x) \sim c$;
- $\langle \rho, i, v \rangle \models x \sim y + c$ iff $v(x) \sim v(y) + c$, where $\sim$ and $+$ are the standard comparison and addition on $\mathbb{Z}$;
- $\langle \rho, i, v \rangle \models x.\phi$ iff $\langle \rho, i, v[x \mapsto \tau_i] \rangle \models \phi$;
- $\langle \rho, i, v \rangle \models \neg\phi$ iff $\langle \rho, i, v \rangle \models \phi$ is false;
- $\langle \rho, i, v \rangle \models \phi_1 \land \phi_2$ iff $\langle \rho, i, v \rangle \models \phi_1$ and $\langle \rho, i, v \rangle \models \phi_2$;
- $\langle \rho, i, v \rangle \models \phi_1\mathcal{U}\phi_2$ iff $\langle \rho, j, v \rangle \models \phi_2$ for some $j \geq i$ and $\langle \rho, k, v \rangle \models \phi_1$ for all $i \leq k < j$;
- $\langle \rho, i, v \rangle \models \phi_1\mathcal{S}\phi_2$ iff $\langle \rho, j, v \rangle \models \phi_2$ for some $j \leq i$ and $\langle \rho, k, v \rangle \models \phi_1$ for all $j < k \leq i$.

We say that a formula is closed if any occurrence of a clock variable $x$ is in the scope of the freeze operator “$x.$”. For instance, $x.((x > y + 1) \land p)$ is not a closed formula since $y$ appears not in the scope of “$y$”. One can show in the standard fashion that the truth value of a closed formula is completely defined by the timed word and the time
Definition 5. Let \( \bar{\rho} = (\bar{\sigma}, \bar{\tau}) \) be a finite timed word. An infinite word \( \rho = (\sigma_\ell, \tau_\ell), (\sigma_{\ell+1}, \tau_{\ell+1}), \ldots \) is called an extension of \( \bar{\rho} \) if

- \( \ell \leq k \);
- \( \bar{\sigma}_i = \sigma_i \) and \( \bar{\tau}_i = \tau_i \) for all \( i \in \{k, \ldots, n\} \);
- if \( \ell < k \) then \( \tau_{k-1} < \tau_k \);
- \( \tau_n < \tau_{n+1} \).

Definition 6. Let \( \bar{\rho} = (\bar{\sigma}, \bar{\tau}) \) be a finite timed word and \( \phi \) a LogLogics-formula. Then

- \( \langle \bar{\rho}, i, v \rangle \models \phi \) is true, if for any extension \( \rho \) of \( \bar{\rho} \), \( \langle \rho, i, v \rangle \models \phi \) is true.
- \( \langle \bar{\rho}, i, v \rangle \models \phi \) is false, if for any extension \( \rho \) of \( \bar{\rho} \), \( \langle \rho, i, v \rangle \models \phi \) is false.
- \( \langle \bar{\rho}, i, v \rangle \models \phi \) is unknown, if \( \langle \rho', i, v \rangle \models \phi \) is true and \( \langle \rho'', i, v \rangle \models \phi \) is false for some extensions \( \rho', \rho'' \) of \( \bar{\rho} \).

We abbreviate \( \langle \bar{\rho}, n, \varepsilon \rangle \models \phi \) to \( \bar{\rho} \models \phi \), where \( \varepsilon \) is the empty valuation function and \( n \) is such that \( \bar{\rho} = (\bar{\sigma}, \bar{\tau}) \ldots (\bar{\sigma}_n, \bar{\tau}_n) \).

3. Abstract timed words

The difficulty that arises with computing the truth values of a LogLogics formula \( \phi \) on a finite timed word \( \bar{\rho} \) is that the straightforward application of Definition 6 requires in general a check of \( \phi \) on an infinite number of infinite timed words (having \( \bar{\rho} \) as a subword). A well-studied approach allowing us to reduce the check of a property of an infinite object to a check of a property on a finite approximation of the object is known as abstraction [6,7,13]. In this section we introduce a notion of an abstract timed word, define a LogLogics semantics on abstract timed words and show that the required check can be reduced to a check on the corresponding abstract timed word.

Since by Definition 6 the truth value w.r.t. \( \bar{\rho} \) can be true, false or unknown, the semantics of a LogLogics-formula w.r.t. an abstract timed word should be three-valued as well. Recall that in the traditional three-valued logics (see e.g. [11]) the truth values are ordered as false \( \prec \) unknown \( \prec \) true and logical connectors and quantifiers are defined in Fig. 1. Note that \( \min(S) \) and \( \max(S) \) are defined for the set \( S \) w.r.t. \( \prec \). Note that the definitions of \( \neg \), \( \exists \) and \( \forall \) properly extend the corresponding definitions for the two-valued case. In other words, if \( \pi(x) \) takes only values true or false for all \( x \in S \) then the truth value of \( \exists x : x \in S : \pi(x) \) in the three-valued logic coincides with its truth value in the two-valued logic, and the same holds for \( \forall x : x \in S : \pi(x) \).
We start with introducing some basic notions we need here.

Abstract time domain. First we extend our time domain to $\mathbb{Z}^+ = \mathbb{Z} \cup \mathbb{Z}^+ \cup \mathbb{Z}^+$, where $\mathbb{Z}$ are whole numbers, $\mathbb{Z}^+ = \{\ldots, -1^+, 0^+, 1^+ \ldots\}$ and $\mathbb{Z}^+ = \{\ldots, -1^+, 0^+, 1^+ \ldots\}$. Now we pick some arbitrary $a, b \in \mathbb{Z}$, $a \leq b$, and define an abstraction function $\alpha_a^b : \mathbb{Z} \rightarrow \mathbb{Z}^+$ and a concretisation function $\gamma : \mathbb{Z}^+ \rightarrow 2^\mathbb{Z}$ in the following way:

$$\alpha_a^b(x) = \begin{cases} x & \text{if } a \leq x \leq b \\
 b^+ & \text{if } x > b \\
 a^+ & \text{if } x < a \end{cases}$$

$$\gamma(z) = \begin{cases} \{z\} & \text{if } z \in \mathbb{Z} \\
 \{x | x > y\} & \text{if } z = y^+ \\
 \{x | x < y\} & \text{if } z = y^-. \end{cases}$$

Following the definition of $\alpha$ and $\gamma$, we introduce addition $+_a$ on $\mathbb{Z}^+ \times \mathbb{Z} \rightarrow \mathbb{Z}^+$ and functions $\leq_a$ and $=_a$ on $\mathbb{Z}^+ \times \mathbb{Z}^+ \rightarrow \{\text{false}, \text{unknown}, \text{true}\}$:

$$z +_a y = \begin{cases} (x + y)^+ & \text{if } z = x^+; \\
 z + y & \text{if } z \in \mathbb{Z}; \\
 (x + y)^- & \text{if } z = x^-; \end{cases}$$

$$(z_1 \leq_a z_2) = \begin{cases} \text{true} & \text{if } x_1 \leq x_2 \text{ for any } x_1 \in \gamma(z_1), x_2 \in \gamma(z_2); \\
 \text{false} & \text{if } x_2 < x_1 \text{ for any } x_1 \in \gamma(z_1), x_2 \in \gamma(z_2); \\
 \text{unknown} & \text{otherwise}; \end{cases}$$

$$(z_1 =_a z_2) = \begin{cases} \text{true} & \text{if for any } x_i \in \gamma(z_i) \text{ there is } x_j \in \gamma(z_j) \text{ such that } x_1 = x_2, \\
 & \text{for } i, j = 1, 2, i \neq j; \\
 \text{false} & \text{if for any } x_i \in \gamma(z_i), x_j \in \gamma(z_j), x_1 \neq x_2, \\
 & \text{for } i, j = 1, 2, i \neq j; \\
 \text{unknown} & \text{otherwise}. \end{cases}$$

Example 7. Consider the expression $7^+ +_a 2 \leq_a 10^+$. By the definition of $\gamma$, $\gamma(7^+) = \{8, 9, 10, \ldots\}$, i.e. $7^+$ is an abstraction of a whole number which is greater than 7. By the definition of $+_a$, $7^+ +_a 2 = (7 + 2)^+ = 9^+$, which coincides with the intuition that tells us that by adding 2 to a number greater than 7, we obtain a number greater than 9.

Next we need to compare $9^+$ and $10^+$. By the definition of $\gamma$, $\gamma(9^+) = \{10, 11, 12, \ldots\}$ and $\gamma(10^+) = \{\ldots, 7, 8, 9\}$. Clearly, $x \leq y$ does not hold for any concretisation of $9^+$ and $10^+$. Hence, $7^+ +_a 2 \leq_a 10^+$ is false.

Note that the definitions of $=_a$ and $\leq_a$ can be rewritten by case enumeration. So, for the case $x^+ \leq y^+$ we obtain $\text{unknown}$ for any $x$ and $y$ since we can always find both concretisations $x_c, y_c$ of $x^+$ and $y^+$ for which the inequality holds, and the ones for which it does not. For the case $x^+ \leq y$ ($y \in \mathbb{Z}$) we obtain $\text{false}$ if $x + 1 > y$ and $\text{unknown}$ otherwise. For $x^+ \leq y^+$ we obtain $\text{false}$ if $x + 1 > y - 1$ and $\text{unknown}$ otherwise; etc.

Example 8. Reconsider $9^+ \leq_a 10^+$ from Example 7. Since $9 + 1 > 10 - 1$, we obtain $\text{false}$.

One can introduce $\geq_a$, $>_a$, $<_a$, $\neq_a$ in the standard way by using $\leq_a$ and $=_a$. We also use $\sim_a$ to refer to an arbitrary abstract comparison.

Abstract timed words. Next we introduce the notion of an abstract timed word. Abstract timed word is a finite word with the special first and last pairs. The first pair is an abstract representation of the period prior to the beginning of the observations while the last pair is an abstract representation of the period after the current moment. These two pairs contain a special event $\sigma^*$ that does not belong to $\Sigma$ and denotes an unknown event. For any atomic proposition $p$ we define $\sigma^* \vdash p$ to be $\text{unknown}$.
Definition 9. An abstract time sequence $\tau^a = \tau_k \tau_{k+1} \ldots \tau_n$ is a finite sequence of times such that $\tau_k \in \mathbb{Z}^+$, $\tau_n \in \mathbb{Z}^+$, and for all $i$, $k < i < n$, $\tau_i \in \mathbb{Z}$; moreover, $\tau_i \leq a \tau_{i+1}$ for all $i \in \{k, \ldots, n - 1\}$.

An abstract event sequence $\sigma^a = \sigma_k \sigma_{k+1} \ldots \sigma_n$ is a finite sequence of events such that $\sigma_k = \sigma^*$, $\sigma_n = \sigma^*$, and for all $i$, $k < i < n$, $\sigma_i \in \Sigma$. For any atomic proposition $p \in P$, $\sigma_k \vdash p$ is unknown, $\sigma_n \vdash p$ is unknown, and for any $k < i < n$, $\sigma_i \vdash p$ is true or false.

An abstract timed word $\rho^a = (\sigma^a, \tau^a)$ is a pair consisting of an abstract event sequence $\sigma^a$ and an abstract time sequence $\tau^a$ of the same length. We also write a timed word as a sequence of pairs $(\sigma_k, \tau_k) \ldots (\sigma_n, \tau_n)$.

Let $\rho = (\sigma, \tau)$ be an infinite timed word with $\tau = \tau_k \tau_{k+1} \ldots$ and $\sigma = \sigma_k \sigma_{k+1} \ldots$. To relate $\rho$ with an abstract timed word, we extend the abstraction and concretisation functions. Let $a \geq k$. Then, the abstraction of $\rho$ w.r.t. $a$ and $b$ is

$$\alpha^b_a(\rho) = (\sigma^*, (\tau_a)^\downarrow)(\sigma_a, \tau_a) \ldots (\sigma_b, \tau_b)(\sigma^*, (\tau_b)^\downarrow).$$

The concretisation function $\gamma$ maps the abstract timed word $(\sigma^*, (\tau_a)^\downarrow)(\sigma_a, \tau_a) \ldots (\sigma_b, \tau_b)(\sigma^*, (\tau_b)^\downarrow)$ to the set of all extensions of $(\sigma_a, \tau_a) \ldots (\sigma_b, \tau_b)$.

Lemma 10. Let $\overline{\rho} = (\overline{\sigma}_1, \overline{\tau}_1) \ldots (\overline{\sigma}_m, \overline{\tau}_m)$ be a finite timed word. Then for any extensions $\rho_1$, $\rho_2$ of $\overline{\rho}$, we have $\alpha^b_a(\rho_1) = \alpha^b_a(\rho_2)$.

Proof. Let $\rho_1$ be $(\sigma_1, \tau_1)$, $(\sigma_{i+1}, \tau_{i+1})$, $\ldots$ and $\rho_2$ be $(\sigma^*_i, \tau^*_i)$, $(\sigma^*_i, \tau^*_i)$, $\ldots$. Since both $\rho_1$ and $\rho_2$ are extensions of $\overline{\rho}$, $\overline{\sigma}_i = \sigma^*_i = \sigma^*_i$ and $\overline{\tau}_i = \tau^*_i = \tau^*_i$ for all $i \in \{a, \ldots, b\}$. Then, $\alpha^b_a(\rho_1) = (\sigma^*, (\tau^*_a)^\downarrow)(\sigma^*_a, \tau^*_a) \ldots (\sigma^*_b, \tau^*_b)(\sigma^*, (\tau^*_b)^\downarrow) = \alpha^b_a(\rho_2)$. □

Next we define the semantics of a LogLogics-formula w.r.t. an abstract timed word.

Definition 11. Let $\rho^a = (\sigma^*, (\tau_a)^\downarrow)(\sigma_a, \tau_a) \ldots (\sigma_b, \tau_b)(\sigma^*, (\tau_b)^\downarrow)$ be an abstract timed word, $i \in \mathbb{Z}$, $a - 1 \leq i \leq b + 1$, and $v : V \rightarrow \mathbb{Z}^+$ be a partial valuation for the clock variables. Then

- $\langle \rho^a, i, v \rangle \models p$ is equal to $\sigma_i \vdash p$;
- $\langle \rho^a, i, v \rangle \models \text{false} \text{ equals } false$;
- $\langle \rho^a, i, v \rangle \models x \sim c$ is equal to the value of $v(x) \sim a c$, where $\sim a$ is the relation on $\mathbb{Z}^+$ corresponding to $\sim$;
- $\langle \rho^a, i, v \rangle \models x \sim y + c$ is equal to the value of $v(x) \sim a v(y) + a c$, where $\sim a$ is as above;
- $\langle \rho^a, i, v \rangle \models x.\phi$ is equivalent to $\langle \rho^a, i, v[x \mapsto \tau_i] \rangle \models \phi$;
- $\langle \rho^a, i, v \rangle \models \neg \phi$ is equivalent to $\neg (\langle \rho^a, i, v \rangle \models \phi)$ (the latter $\neg$ being used as defined in Fig. 1);
- $\langle \rho^a, i, v \rangle \models \phi_1 \land \phi_2$ is equivalent to $\langle \rho^a, i, v \rangle \models \phi_1 \land \langle \rho^a, i, v \rangle \models \phi_2$ (the latter $\land$ being used as defined in Fig. 1);
- $\langle \rho^a, i, v \rangle \models \phi_1 \mathcal{U} \phi_2$ is equivalent to $\exists j : i \leq j : \langle \rho^a, j, v \rangle \models \phi_2 \land \forall k : i \leq k < j : \langle \rho^a, k, v \rangle \models \phi_1$ (the quantifiers and $\land$ being used as defined in Fig. 1);
- $\langle \rho^a, i, v \rangle \models \phi_1 \mathcal{S} \phi_2$ is equivalent to $\exists j : j \leq i : \langle \rho^a, j, v \rangle \models \phi_2 \land \forall k : j < k \leq i : \langle \rho^a, k, v \rangle \models \phi_1$ (the quantifiers and $\land$ being used as defined in Fig. 1).

We also abbreviate $\langle \rho^a, b, v \rangle \models \phi$ to $\rho^a \models \phi$.

Theorem 12. Let $\overline{\rho} = (\overline{\sigma}_1, \overline{\tau}_1) \ldots (\overline{\sigma}_n, \overline{\tau}_n)$ be a finite timed word, and $\rho$ an extension of $\overline{\rho}$. Then, for any LogLogics-formula $\phi$, the truth value of $\overline{\rho} \models \phi$ is equal to the truth value of $\alpha^a_\ell(\rho) \models \phi$.

Proof. The proof is easily obtained by induction on the structure of $\phi$ in the standard way. □

Recall that Definition 6 determines the truth value of a LogLogics-formula w.r.t. a finite timed word depending on its truth values w.r.t. all possible extensions. Hence, it cannot be used to compute the truth value. The theorem above resolves this problem by reducing the check of truth values on infinitely many extensions of the given timed word to checking the truth value on a finite object: the corresponding abstract timed word.

Example 13. Let us evaluate the formula $\exists x.(x \geq 0 \Rightarrow (p \Rightarrow \exists y.(q \land y \leq x + 4)))$ on the finite timed word $\overline{\rho} = ((\sigma_0, 0)(\sigma_1, 1)(\sigma_2, 1)(\sigma_3, 2)(\sigma_4, 5)(\sigma_5, 8))$ such that $\sigma_i \vdash p$ is true for $i = 1$ and $i = 4$, and false for $i \in \{0, 2, 3, 5\}$;
\( \sigma_i \vdash q \) is true for \( i = 3 \) and false for \( i \in \{0, 1, 2, 4, 5\} \) (see Fig. 2). Intuitively, this formula says that whenever \( p \) was encountered during the observation period, \( q \) was encountered not later than four time units after that.

By Theorem 12 we reduce our problem to evaluation of the formula w.r.t. the abstract timed word \( \rho^\alpha = ((\sigma^*, 0^1)(\sigma_0, 0)(\sigma_1, 1)(\sigma_2, 1)(\sigma_3, 2)(\sigma_4, 5)(\sigma_5, 8)(\sigma^*, 8^1)) \), which is the abstraction of any extension of \( \bar{p} \).

First, we observe that we need to minimise \( \langle \rho^\alpha, i, \epsilon \rangle \models x.(x \geq 0 \Rightarrow (p \Rightarrow \Diamond y.(q \land y \leq x + 4))) \) for all \( i \leq 5 \). This is equivalent to minimising the value of \( \langle \rho^\alpha, i, [x \mapsto \tau_i] \rangle \models x \geq 0 \Rightarrow (p \Rightarrow \Diamond y.(q \land y \leq x + 4)) \) for \( i \leq 5 \). For \( i = -1, \tau_{-1} \geq 0 \) is false and hence the implication is true. For \( i \in \{0, 2, 3, 5\} \), \( \langle \rho^\alpha, i, [x \mapsto \tau_j] \rangle \models p \) is false and therefore the inner implication is true, and so is the outer one. The cases left are:

- \( i = 4. \) Since \( \sigma_4 \vdash p \) is true, the truth value of the implication coincides with the truth value of \( \langle \rho^\alpha, 4, [x \mapsto 5] \rangle \models \Diamond y.(q \land y \leq x + 4) \). To determine the latter value we need to maximise the value of \( \langle \rho^\alpha, j, [x \mapsto 5] \rangle \models y.(q \land y \leq x + 4) \) for \( j \geq 4 \), i.e., the value of \( \langle \rho^\alpha, j, [x \mapsto 5, y \mapsto \tau_j] \rangle \models (q \land y \leq x + 4) \). For each \( j \geq 4 \) the value of the conjunction is the least value of \( \langle \rho^\alpha, j, [x \mapsto 5, y \mapsto \tau_j] \rangle \models q \) and \( \langle \rho^\alpha, j, [x \mapsto 5, y \mapsto \tau_j] \rangle \models y \leq x + 4 \).

  - If \( j = 4 \) or \( j = 5 \), \( \sigma_j \vdash q \) is false and, hence, the value of the conjunction is false as well. If \( j = 6 \), \( \sigma_j \vdash q \) is unknown, \( \tau_6 = 8^\dagger \) and \( \langle \rho^\alpha, j, [x \mapsto 5, y \mapsto 8^\dagger] \rangle \models y \leq x + 4 \) reduces to \( 8^\dagger \leq_{\alpha} 9 \) that evaluates to unknown. Hence, the value of the conjunction in this case is unknown. To determine the value of the implication for \( i = 4 \) we should take the maximal value, which is unknown, obtained for \( j = 6 \).

- \( i = 1 \). As above, since \( \sigma_1 \vdash p \) is true, the truth value of the implication coincides with the truth value of the maximum (on \( j \geq 1 \)) of the least of the two following values: \( \langle \rho^\alpha, j, [x \mapsto 1, y \mapsto \tau_j] \rangle \models (q \land y \leq x + 4) \).

  - If \( j \in \{1, 2, 4, 5\} \), then \( \sigma_j \vdash q \) is false, and so is the value of the conjunction. Since \( \tau_6 = 8^\dagger \), \( \langle \rho^\alpha, j, [x \mapsto 1, y \mapsto 8^\dagger] \rangle \models y \leq x + 4 \) reduces to \( 8^\dagger \leq_{\alpha} 5 \) that evaluates to false and the same is true for the conjunction. Finally, for \( j = 3 \), \( \sigma_j \vdash q \) is true and \( \tau_j = 2 \leq_{\alpha} 1 + 4 \). Hence, both conjuncts evaluate to true and the conjunction as well. Hence, the maximal value is true, \( \langle \rho^\alpha, 1, [x \mapsto 1] \rangle \models \Diamond y.(q \land y \leq x + 4) \) evaluates to true and the truth value of the implication is true.

To find the truth value of the original formula, we need to take the least value obtained. This value is “unknown”.

Example 13 also explains the true meaning of unknown. The formula is evaluated to unknown due to the behaviour on the boundaries of the observation sequence. Consider the finite timed word in Fig. 3 which differs from the one in Fig. 2 for \( i = 4 \) only. With respect to this finite timed word the response property from Example 13 is evaluated to true. However, if one considered \( \Box x.(p \Rightarrow \Diamond y.(q \land y \leq x + 4)) \) the truth value still would have been unknown. In such a case one might like to exclude the unknown prehistory and/or unknown future from the consideration. In fact, our formula in Example 13 excluded the prehistory. Since similar restrictions turn out to be useful for expressing interesting business properties, we introduce the following short-hand notation:

\[
\begin{align*}
\Box^b_{a_i} \phi(x) & \overset{\text{def}}{=} \Box x.(x < a \lor x > b \lor \phi(x)) \\
\Diamond^b_{a_i} \phi(x) & \overset{\text{def}}{=} \Diamond x.(x \geq a \land x \leq b \land \phi(x)) \\
\Box^b_{a_i} x. \phi(y) & \overset{\text{def}}{=} \Box x.(x < a \lor x > b \lor \phi(x)) \\
\Diamond^b_{a_i} x. \phi(y) & \overset{\text{def}}{=} \Diamond x.(x \geq a \land x \leq b \land \phi(x)).
\end{align*}
\]
Subscripts and superscripts of boxes and diamonds can be omitted when only one of the boundaries is of interest. Using the short-hand notation formula in Example 13 can be written as $\Box_{0,x}(p \Rightarrow \Diamond_{x}^4 y.(q \land y \leq x + 4))$. Note that the limit values $a$ and $b$ in the short-hand notation can depend on the values of the clock variables in whose scope the corresponding temporal operator appears. This means that the formula above can be further rewritten as $\Box_{0,x}(p \Rightarrow \Diamond_{x}^{4} x.q)$, which some users experience as more intuitive. We give more examples for the use of the abbreviations in Section 5.

**Lemma 14.** Let $\overline{p} = (\sigma_k, \tau_k) \ldots (\sigma_n, \tau_n)$ be a finite timed word. Let a restricted LogLogics-formula $\psi$ be inductively defined as:

$$\psi := p \mid x \sim y + c \mid x \sim c \mid x.\psi \mid false \mid \neg\psi \mid \psi_1 \land \psi_2 \mid \Box^b_a \psi \mid \Box^b_a \psi \mid \Diamond^b_\sigma \psi \mid \Diamond^b_\sigma \psi,$$

where $x, y \in V, p \in P, \sim$ is one of $<, >, \leq, \geq, =, \neq, c \in \mathbb{N}$ and $\tau_k \leq a \leq b \leq \tau_n$. Then $\overline{p} \models \psi$ is evaluated to true or false.

Note that by using restricted LogLogics-formulas only we obtain the logic that coincides with the logics from [1,8].

**4. Algorithm**

In this section we present an algorithm that evaluates a given LogLogics-formula $\phi$ w.r.t. a given abstract timed word and context $(\rho^\alpha, i, v)$.

We assume the existence of two auxiliary procedures $\text{EvalAtomic}(p, a, b, \rho^\alpha, i)$ and $\text{EvalClock}(\text{cvc}, v)$, where $p$ is an atomic proposition, $\text{cvc}$ a clock variable comparison ($x \sim c$ or $x \sim y + c$) and $a$ and $b$ correspond to pre-history and future indexes. In Fig. 4 we define the procedure $\text{Eval}$ with the following parameters: a LogLogics formula $\phi$, an abstract timed word $\rho^\alpha$, a pre-history event index $a$, a future event index $b$, a current event index $i$, a clock valuation $v$, a current minimum truth value $\min$ and a current maximum truth value $\max$. The evaluation of a closed formula $\phi$ w.r.t. an abstract timed word $\rho^\alpha = (\sigma^*, (\tau_{a+1})^\downarrow)(\sigma_{a+1}, \tau_{a+1}) \ldots (\sigma_{b+1}, \tau_{b+1})(\sigma^*, (\tau_{b+1})^\downarrow)$ is performed by calling $\text{Eval}(\phi, \rho^\alpha, a, b, \text{last}, \Box, \text{false, true})$, where $\text{last}$ gives the index of the last “non-abstract” entry in $\rho^\alpha$, i.e., $\text{last} = b - 1$. We assume that the formula $\phi$ is austere, i.e. the additional operators such as $\lor$ or $\Box$ have been replaced by their definitions.

Depending on the form of $\phi$, the procedure $\text{Eval}$ recursively calls itself until the subnodes have been exhausted or $\max = \min$. The formula is thus evaluated as a tree with atomic propositions $p$ and clock variable comparisons $x \sim c, x \sim y + c$ as leaves and operator symbols as other nodes. The algorithm makes a nondeterministic choice when evaluating conjunctions. A speedup may be possible by making better choices, choosing subnodes that can be evaluated fast and are likely to become $\text{false}$.

Recall that $\phi_1U\phi_2$ is true w.r.t. $(\rho^\alpha, i, v)$ if either $\phi_2$ is true w.r.t. $(\rho^\alpha, i, v)$ or both $\phi_1$ is true w.r.t. $(\rho^\alpha, i, v)$ and $\phi_1U\phi_2$ is true w.r.t. $(\rho^\alpha, i + 1, v)$. In our three-valued case, $(\rho^\alpha, i, v) \models \phi_1U\phi_2$ is $(\rho^\alpha, i, v) \models \phi_2 \lor ((\rho^\alpha, i, v) \models \phi_1 \land (\rho^\alpha, i + 1, v) \models \phi_1U\phi_2))$. The value of $i$ is limited by the length of the word. This observation is used in the algorithm to evaluate LogLogics-formulas of the form $\phi_1U\phi_2$. The case of $S$ is analogous.

Termination of the algorithm stems from the following fact: at each step of the computation, evaluating $(\rho^\alpha, i, v) \models \phi$ is reduced to evaluating a finite number of $(\rho^\alpha, i_1, v_1) \models \phi_1, \ldots, (\rho^\alpha, i_n, v_n) \models \phi_n$. The parameter $n$ is bounded by maximum of $b - a + 1$ (cases of $U$ and $S$) and 2 (conjunction). Each one of the $(\rho^\alpha, i_j, v_j) \models \phi_j$ is strictly smaller than $(\rho^\alpha, i, v) \models \phi$ w.r.t. the following order relation:

$$(\rho^\alpha, i_1, v_1) \models \phi_1) > (\rho^\alpha, i_2, v_2) \models \phi_2)$$

if $\phi_2$ is a subterm of $\phi_1$, or $\phi_2$ coincides with $\phi_1$ and $\phi_2 = \psi_1U\psi_2$ and $i_1 < i_2$ or $\phi_2 = \psi_1S\psi_2$ and $i_1 > i_2$.

Finally, observe that the parameters $\min$ and $\max$ express the information gained so far on the range of relevant values of the subformula. By relevant values we understand those values that can influence the truth value of the superformula. Moreover, one can show that $\min \leq \text{Eval} (\phi, \rho^\alpha, a, b, i, v, \min, \max) \leq \max$ for any values of the parameters.
Dec. 31, 2005 was a transaction for a sum exceeding 5,000,000 in 2005, which we can encode as ♦

The event occurs through the scope. Patterns belonging to this group are occurrence, bounded occurrence, absence and times. In particular, if ♦ undesired event. In the most general form it requires that in a given scope a given event occurs at least ♦ processes.

LogLogics ♦ a scope:

Typical guards of interest ♦

Dwyer et al. [9] have identified a number of property specification patterns for software verification and formalised them in LTL and CTL. In this section we analogously consider LogLogics guard specification patterns for business processes.

The first group of patterns concerns the occurrence of a certain desired event, or dually, the absence of a certain undesirable event. In the most general form it requires that in a given scope a given event occurs at least ♦ and at most ♦ times. In particular, if ♦ 0, the event does not occur at all, and if ♦ a equals the number of time points in a scope, the event occurs through the scope. Patterns belonging to this group are occurrence, bounded occurrence, absence and universality.

Occurrence ♦ pattern allows us to check whether some event happened in a certain time interval, e.g. whether there was a transaction for a sum exceeding 5,000,000 in 2005, which we can encode as ♦ x.(p ∧ x ≥ ‘Jan. 1, 2005’ ∧ x ≤ ‘Dec. 31, 2005’), where p stands for a transaction exceeding 5,000,000. Using the short-hand notation of Section 3, it can be also written as ♦ ‘Jan. 1, 2005’ ♦ x.p. In general, the occurrence pattern has the following form: ♦ ♦ x.φ, where ♦ a and/or ♦ b can be omitted.

Bounded occurrence ♦ is similar to the occurrence pattern but requires a certain event to occur at least ♦ k times within a scope:

\[ ♦ x_1.(φ \land x_1 ≥ t_1 \land x_1 ≤ t_2 \land \cdots \land x_k.(φ \land x_k ≥ t_1 \land x_k ≤ t_2 \land x_k \neq x_1 \land \cdots \land x_k \neq x_{k−1})], \]
or alternatively,
\[
\Diamond^k_a x_1 . (\phi \land \Diamond^b_a x_2 . (\phi \land x_2 \neq x_1 \land \cdots \Diamond^b_a x_k . (\phi \land x_k \neq x_1 \land \cdots \land x_k \neq x_{k-1}))).
\]

Variants of this pattern require the event to occur exactly \(k\) or at most \(k\) times. Using this pattern we can e.g. express the guard checking whether there were at least three transactions for a sum exceeding 5,000,000 between January 1, 2005 and December 31, 2005.

Absence pattern is dual to the occurrence pattern and can be written as \(\Box x . (\neg \phi(x) \lor x < a \lor x > b)\) or alternatively as \(\Diamond^b_a x . \neg \phi(x)\), where \(\phi\) denotes an event undesired between time points \(a\) and \(b\). In this way we can check that between \(a\) and \(b\) no transaction was rejected.

Universality pattern allows us to express properties that should hold throughout the period from \(a\) to \(b\): \(\Box x . (\phi(x) \lor x < a \lor x > b)\), i.e., \(\Diamond^b_a x . \phi(x)\). A property we could express with this pattern is “between \(a\) and \(b\) all transactions were executed successfully”.

The next group of patterns, called ordering patterns, expresses an ordering relationship between two (or more) events. Ordering patterns can be constructed from the occurrence patterns by demanding that one of them occurs in a scope within a time slot of another one.

Bounded response is an extremely common pattern, an instance of which we considered in Example 13. It allows us to express such guards as “every bill is paid within 30 days”. In general, the pattern has the form
\[
\Box^b_a x . (\phi_1(x) \Rightarrow \Diamond^{d(x)}_{c(x)} y . \phi_2(x)),
\]
where \(c(x)\) and \(d(x)\) are timed constraints, i.e., propositional formulas over clock variable comparisons \(x \sim c\) and \(x \sim y + c\).

Precedence pattern requires that any occurrence of \(p\) is preceded by an occurrence of \(q\) within a scope: \(\Diamond^b_a x . (\phi_1(x) \Rightarrow \Diamond^{d(x)}_{c(x)} y . \phi_2(x))\). An instance of this pattern allows us to express the guard that a loan was preceded by a credibility check with an outcome above a certain threshold.

Absence between pattern requires that between time points \(a\) and \(b\), no \(r\)-event happens between \(p\)-event and \(q\)-event, expressed as \(\Box^b_a x . ((p \land \neg q \land \Diamond x) \rightarrow (\neg r \land q))\). An example of a guard would be “no credit card transactions took place between the card issue and the report that the card was not received by the legal owner”.

Compound patterns, forming the last group of patterns, can be constructed from the patterns above by means of conjunction and disjunction.

6. Conclusion

In this paper we have proposed a logic that works on finite traces and is appropriate for specifying guards in models of history-dependent processes. Since at any given moment of time information is finite and inherently incomplete, we had to adapt existing timed temporal logics, which resulted in a three-valued logics, LogLogics, presented above. Although the straightforward application of the definition of the LogLogics semantics gives rise to a procedure with an infinite number of checks, we have shown that a check of the truth of an LogLogics-formula can be reduced to a check of its truth value on a finite abstraction. We have also shown how guard patterns common for business processes can be expressed in LogLogics.

For the future work, we will investigate the complexity of checking LogLogics formulas. As shown in [14], the complexity of checking whether a finite path \(u\) satisfies an LTL+Past formula \(\varphi\) is \(O(|u| \times |\varphi|)\). We expect a similar result to hold for our logic as well. Therefore the complexity of checking formulas of our logic will not form an obstacle for applying it in practice.

We plan to create a simple textual language for working with patterns targeted on non-specialists and to build a tool for checking LogLogics-formulas on history logs. The ultimate goal is to integrate the logic into existing workflow modelling frameworks, in particular in adaptive workflows [10].

Acknowledgement

Olivia Oanea was supported by the NWO Open Competition project ‘Modelling and Verification of Business Processes, MoveBP, project number 612.000.315.

References


