Early Detection of Temporal Constraint Violations

Isaac Mackey
University of California, Santa Barbara, CA, USA

Raghubir Chimni
University of California, Santa Barbara, CA, USA

Jianwen Su
University of California, Santa Barbara, CA, USA

Abstract

Software systems rely on events for logging, system coordination, handling unexpected situations, and more. Monitoring events at runtime can ensure that a business service system complies with policies, regulations, and business rules. Notably, detecting violations of rules *as early as possible* is much desired as it allows the system to reclaim resources from erring service enactments. We formalize a model for events and a logic-based rule language to specify temporal and data constraints. The primary goal of this paper is to develop techniques for detecting each rule violation as soon as it becomes inevitable. We further develop optimization techniques to reduce monitoring overhead. Finally, we implement a monitoring algorithm and experimentally evaluate it to demonstrate our approach to early violation detection is beneficial and effective for processing service enactments.

2012 ACM Subject Classification Information systems → Information systems applications

Keywords and phrases temporal constraints, monitoring, events, early violation detection

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.4

1 Introduction

Events are unorchestrated, asynchronous messages about the states of processes and situations like action and change. Events are a fundamental component in software systems including workflow systems, cyber-physical systems, IOT devices, decision support systems, etc., and a focus of research communities (e.g., [20]). These systems use events to *i*) identify time-critical exceptional situations that need attention, and *ii*) choreograph/orchestrate collaborative systems [5]. This paper studies a technical problem concerning *i*).

In runtime monitoring [2, 12], system policies for exceptional situations, i.e., violations of constraints, are specified in a formal language and algorithms monitor events from the system as they occur to detect and report violations. Violations of system policies can be divided into two categories depending on when the violation is detected: a violation can be detected once the system is finished executing or it can be reported when the system's execution is not yet finished but *as soon as* the violation becomes inevitable; we call the latter *early (violation) detection*.

We investigate the early detection problem in the context of workflow systems, where events report execution of activities in a workflow. In this setting, constraints are set by business rules, organization policies, regulations, and service-level agreements (SLAs) and specify temporal relationships between events in a workflow enactment, with "gap constraints" [24] to restrict time gaps between events. Constraints can also reference and compare data values in events. We call the growing set of events in an enactment a "log". A naive monitoring approach would (re)evaluate constraints over the entire log with each arrival of new events, but this is intractable for large logs, so we evaluate constraints incrementally.

This paper makes the following technical contributions:

29th International Symposium on Temporal Representation and Reasoning (TIME 2022).
Editors: Alexander Artikis, Roberto Posenato, and Stefano Tonetta; Article No. 4; pp. 4:1–4:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- A technique for calculating the earliest time a violation is inevitable (the "deadline"),
- Algorithms and data structures for incrementally maintaining and detecting violations, along with batch algorithms for processing incoming events,
- Optimization techniques for those algorithms, including expiring useless data and improving batch processing, and
- Experimental findings illustrating the benefits and costs of early violation detection.

This paper is organized as follows. Subsection 1.1 discusses related work. Section 2 motivates the early violation detection problem with an example. Section 3 defines the technical framework. Section 4 presents the key techniques for computing "deadlines", maintaining assignments and relationships between them, and detecting violations. Section 5 presents two optimization techniques. Section 6 presents the findings of an experimental evaluation. Finally, Section 7 concludes the paper.

1.1 Related Work

To identify when a violation is first inevitable, we distinguish between potential violations, which may or may not remain a violation in the future, and permanent violations, which are violations in all possible futures. This distinction is formalized for monitoring LTL formulas in [4], which notes that knowing if a trace satisfies or violates a constraint can be refined by knowing if the satisfaction or violation is permanent. [18] shows that this distinction can be monitored for propositional constraints in the Declare language using an encoding of violation status, i.e., potential or permanent, in states of automata derived from the constraint language. [22] uses a similar classification of violations (potential violations are called pending violations). The status of a violation is represented by a fluent in event calculus (EC) and changes to violations' status are encoded as EC axioms that initialize and change fluents. We distinguish potential and permanent violations based on satisfiability checking for partial initializations of constraint variables. Partial initialization is not a new technique (e.g., [3, 9]), though [9] does not monitor events with data and neither attempts early violation detection. Our work is more similar to that of [17], where satisfiability checking determines if constraints in MP-Declare (a variant of Declare that supports conditions on data and time) are permanently violated, however we provide an algorithm that calculates deadlines, rather than offloading the calculation to a solver.

Specifying conditions on data carried by events, such as matching the user opening an order to the user charged payment, is an important functionality of monitoring constraints [14]. [21] adds data conditions to Declare and the conditions are incorporated into the EC formalization [22]. Another approach to include data is found in [7, 8], which monitor FO-LTL and Declare constraints, resp., using automata whose states are augmented with data stores, and potential and permanent violations are distinguished in the same manner as [18], but these works assume a fixed, finite domain for data values. Incremental view maintenance for Datalog offers relevant incremental algorithms. [11] maintains non-recursive views but does not have any time or inequality constraints; our language allows timestamps and gap constraints. [23] maintains recursive views; our language assumes a fixed set of atomic events, which does not allow recursion.

[14] also argues that quantitative time constraints are important for compliance specification. LTL, Declare, and their metric extensions [3, 17, 21, 22] can require the gap between a pair of event timestamps to fall in an interval. Our language gives each event atom in a constraint a time variable, thus allowing an unbounded number of gap (in)equalities

and constant offsets between any and all pairs of event timestamps. It is unknown if our constraints can be translated into LTL, though for a subset of our language, specifically dataless, "singly-linked" rules, [15, 16] provide a translation.

Controllability is another approach to manage temporal constraints in workflow enactments. [6] and [13] feature propagation of upper and lower bound constraints similar to our deadline calculation approach, but does not allow comparison of data values in events. [10] applies explicit time variables to the controllability problem for modular process models. Enforcing controllability is a design-time solution, however; we make no assumptions about the control structure of a service in order to afford managers and users maximum flexibility.

2 An Opportunity for Early Violation Detection

We illustrate the problem of detecting violations of business rules for workflows and motivate an approach based on reasoning about constraints. We sketch an example workflow from an Infrastructure-as-a-Service (IaaS) provider. Then, we explain how the constraints on the workflow are evaluated to determine the earliest time violations are permanent.

Consider an IaaS provider that offers high-performance cloud computing rentals. The service is managed by a workflow with the following activities: the user Requests a machine through an account and the provider grants Approval to the user. Then, the user Reserves a machine for their account, makes a Payment with their account and Launches the machine. The completion of each activity generates an event; events for the same rental instance form an *enactment*. Each event has a timestamp, an enactment identifier, and may have additional data, e.g., a user. We view a set of events as a relational database. Fig. 1 shows a database S_9 at time 9, with eight events from two enactments with ids π_1 and π_2. For example, the first row of the Request table indicates a Request event with enactment id π_1 from user Alice with account $a3$ at time 1.

Request

ID	*user*	*account*	ts
π_1	Alice	a3	1
π_1	Alice	a4	3
π_2	Bob	b6	7

Approval

ID	*user*	ts
π_1	Alice	6

Reserve

ID	*user*	*account*	ts
π_1	Alice	a4	8
π_1	Alice	a3	9

Payment

ID	*user*	*account*	ts
π_1	Alice	a3	8
π_1	Alice	a4	9

Launch

ID	*user*	*account*	ts

Figure 1 Database S_9 with events from two enactments π_1 and π_2.

The provider checks each enactment against specified business rules; these may measure service availability, quality, etc. For example, we use a few rules, including: when a user's Request is approved within 7 days and the machine is Reserved within 7 days of Approval by the same account as the request, the user should make a Payment for the machine through that account within 3 days of Approval and Launch it within 7 days of Reserve and 4 days of Payment. In this rule, events generated by either the provider or the user may lead to rule violation. We write this rule as $\varphi \to \psi$ where φ is the rule body and ψ is the rule head:

$$\begin{aligned}&\mathsf{Request}(u,a)@x, \mathsf{Approval}(u)@y, x \leqslant y \leqslant x+7, \mathsf{Reserve}(u,a)@z, y \leqslant z \leqslant y+7\\ &\to \mathsf{Payment}(u,a)@w, \mathsf{Launch}(u,a)@v, y \leqslant w \leqslant y+3, z \leqslant v \leqslant z+7, v \leqslant w+4\end{aligned}$$

The core idea of detecting a violation is checking whether each body assignment for the body variables u, a, x, y, z satisfying φ has a matching head assignment for the head variables u, a, w, v satisfying ψ. In order to detect violations incrementally, we build assignments that satisfy the rule's subformulas. Fig. 2(a) lists the partial and complete assignments for φ

Aid	ID	u	a	x	y	z	
μ_1	π_1	Alice	a3	1	-	-	
μ_2	π_1	Alice	a4	3	-	-	
μ_3	π_1	Alice	-	-	6	-	
μ_4	π_1	Alice	a3	1	6	-	$\mu_1 + \mu_3$
μ_5	π_1	Alice	a4	3	6	-	$\mu_2 + \mu_3$
μ_6	π_2	Bob	b6	7	-	-	
μ_7	π_1	Alice	a4	-	-	8	
μ_8	π_1	Alice	a4	3	-	8	$\mu_2 + \mu_7$
μ_9	π_1	Alice	a4	-	6	8	$\mu_3 + \mu_7$
μ_{10}	π_1	Alice	a4	3	6	8	$\mu_2 + \mu_9$
μ_{11}	π_1	Alice	a3	-	-	9	
μ_{12}	π_1	Alice	a3	1	-	9	$\mu_1 + \mu_{11}$
μ_{13}	π_1	Alice	a3	-	6	9	$\mu_3 + \mu_{11}$
μ_{14}	π_1	Alice	a3	1	6	9	$\mu_4 + \mu_{12}$

(a) All body assignments at time 9.

Aid	ID	u	a	x	y	z	
μ_{15}	π_1	Alice	-	-	10	-	
μ_{16}	π_1	Alice	a4	3	10	-	$\mu_2 + \mu_{15}$
μ_{17}	π_2	Bob	-	-	10	-	
μ_{18}	π_2	Bob	b6	7	10	-	$\mu_6 + \mu_{17}$

(b) New body assignments added at time 10.

Figure 2 Assignments for the rule body φ and events S_9, events $S_9 \cup \{e_1, e_2\}$.

and S_9. For example, assignment μ_1 is generated by the first row (event) of the Request table. Assignment μ_3 is generated by the first row of the Approval table, and is combined with μ_1 to form μ_4. Then, μ_{10} and μ_{14} makes φ true.

Suppose two events happen at time 10, e_1:Approval(π_1, [Alice], 10), e_2:Approval(π_2, [Bob], 10). Event e_1 generates a partial assignment μ_{15}, which combines with μ_2 into μ_{16}. Event e_2 yields new assignments μ_{17} and μ_{18}. Fig. 2(b) lists four assignments generated by e_1 and e_2.

ψ has six variables u, a, y, z, w, v, but Payment and Launch events only supply values for the four "event variables" u, a, w, v. We consider assignments for ψ in the same manner as for φ but ignoring y and z. The Payment events at times 8, 9 (Fig. 1) create partial assignments β_1: [π_1, Alice, a3, 8, -] and β_2: [π_1, Alice, a4, 9, -].

One interesting problem is to determine *when* to report rule violations. In Fig. 3, three events create a potential violation μ_{10}. It is natural to report this violation when the *END* of enactment π_1 arrives, after which no more events for π_1 will arrive; if μ_{10} is not extended by a head assignment by that time, it represents a permanent violation. We aim to detect violations as soon as they become permanent, which may be well before the *END* event. Given the rule's constraints $y \leqslant w \leqslant y+3$ and $z \leqslant v \leqslant z+7$, and $\mu_{10}(y) = 6$ and $\mu_{10}(z) = 8$, the violation is known to be permanent at time 9 because no Payment event arrives with a timestamp for w to extend μ_{10}. Note also that 9 is the earliest time we can be certain this is a violation.

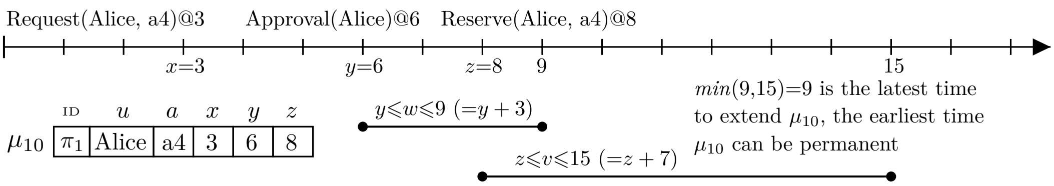

Figure 3 Deadline for extending potential violation μ_{10}.

Fig. 4 shows a Payment event at time 9 that creates the partial assignment β_2.

The main focus of this paper is to calculate these earliest times, which we call "deadlines", and use them in a monitoring algorithm.

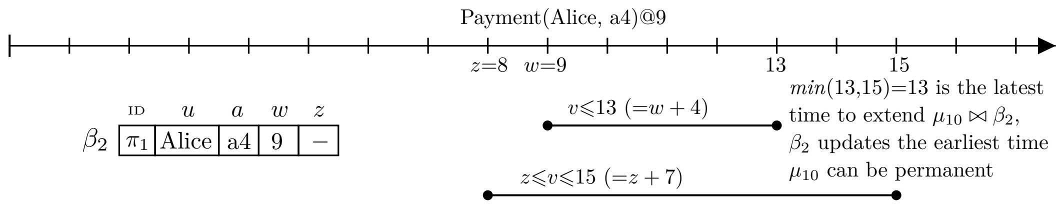

Figure 4 Deadline for extending β_2 as match for μ_{10}.

3 Rules and the Detection Problem

In this section, we present key notions needed for technical development, including "activities" in workflows, "events" of activities, "enactments", "batches", and "rules".

Activities are atomic units of work in a workflow. Each activity has a name and a set of data attributes. An activity's execution yields an event, which carries values for the data attributes and a timestamp. We use *identifiers* $\mathbf{I}$ (or simply ID's), for (workflow) enactments; each event has an identifier from the workflow instance that generated it. We assume a countably infinite set of timestamps $\mathbf{T}$ with a discrete total order and addition of constants. For technical development, we use natural numbers for timestamps. We also assume a countably infinite set of (data) values $\mathbf{D} = \{a, b, c, ..., a_1, ...\}$ with equality.

▶ **Definition.** *An* event *of an activity $A(\mathrm{C}_1, ..., \mathrm{C}_n)$ is a named tuple $A(\xi, \nu, \tau)$ where ξ is an* ID *from $\boldsymbol{I}$, $\nu : \{\mathrm{C}_1, ..., \mathrm{C}_n\} \to \boldsymbol{D}$ is a mapping from A's attributes to data values, and τ is a timestamp from $\boldsymbol{T}$.*

An instance of a workflow is a finite set η of events called an *enactment*, such that (i) each event has the same enactment ID, (ii) η has exactly one special *START* event that marks its beginning and of workflow enactments and at most one *END* event that marks its completion, (iii) the timestamp of the *START* event is less than that of all other events in η, and (iv) the timestamp of the *END* event, if it occurs, is greater than that of all other events in η. The rows in Fig. 1 with the same ID form an enactment (the *START*/*END* events are not shown).

This paper focuses on monitoring enactments as they are updated by new events. Constraints to be monitored are specified as "rules". In the following, we define and illustrate the notions of a "batch" (new events arriving) and a rule.

▶ **Definition.** *A* batch *for an enactment η is a finite set Δ of events such that (i) all events in Δ have the same timestamp, denoted as ts_Δ, greater than the timestamps of all events in η, (ii) for each event e in Δ, the* ID *of e is the* ID *of η, (iii) Δ has a* START *event or η has a* START *event, but not both, and (vi) if an* END *event is in η, no events are in Δ.*

Fig. 1 shows events from two enactments of the workflow in the IaaS example. Suppose that at time 10 exactly two events happened, e_1:$\mathsf{Approval}(\pi_1, [\text{Alice}], 10)$ and e_2:$\mathsf{Approval}(\pi_2, [\text{Bob}], 10)$. Then $\{e_1\}$ is a batch for π_1, $\{e_2\}$ a batch for π_2.

We describe a language for specifying rules, starting with atomic formulas. An *event atom* is an expression "$A(v_1, ..., v_n)@x$" where $A(\mathrm{C}_1, ..., \mathrm{C}_n)$ is an activity, $v_1, ..., v_n, x$ are variables, where x is a *time variable*. A *gap atom* is an expression "$x \pm \epsilon\, \theta\, y$" where x, y are time variables, ϵ (the gap) is a timestamp in $\mathbf{T}$, and $\theta \in \{<, \leqslant, \geqslant, >, =\}$ is an equality or inequality predicate. We denote the variables in a set of gap atoms φ as $var(\varphi)$.

▶ **Definition.** *A* rule *is an expression "$\varphi \to \psi$" where the* body *φ and the* head *ψ are finite sets of event and gap atoms such that each variable in a gap atom in φ occurs in an event atom in φ and each variable in a gap atom in ψ occurs in an event atom in $\varphi \cup \psi$.*

Rule satisfaction is defined as follows: An assignment is a mapping from variables to values in $\mathbf{D}\cup\mathbf{T}$. Time variables take values from $\mathbf{T}$; we use $\mathbb{N}$ as timestamps for technical development. All other variables take values from $\mathbf{D}$. An assignment is *complete* if it is a total mapping for the variables in a given set of atoms, *partial* otherwise. An assignment β extends an assignment α if $\alpha \subseteq \beta$. An enactment η satisfies an event atom $A(v_1, ..., v_n)@x$ for the activity $A(\mathrm{C}_1, ..., \mathrm{C}_n)$ with a complete assignment μ if $A(\eta.\mathrm{ID}, \{\mathrm{C}_1 \mapsto \mu(v_1), \ldots, \mathrm{C}_n \mapsto \mu(v_n)\}, \mu(x))$ is an event in η. An assignment satisfies a gap atom with the obvious interpretation.

An enactment η *satisfies* a set of atoms ϕ with a complete assignment μ if η satisfies every atom in ϕ with μ. An enactment η *satisfies* a rule $r\colon \varphi \to \psi$ if for every complete assignment μ such that η satisfies φ with μ, there is a complete assignment β that extends μ such that η satisfies ψ with β.

▶ **Example 1.** As shown in Fig. 3, the assignment μ_{10} satisfies φ. Then, to satisfy the rule w.r.t. μ_{10}, there must be an assignment extending μ that satisfies ψ; i.e., two events $\mathsf{Payment}(\pi_1, [\mathrm{Alice}, \mathrm{a4}], t_1)$ and $\mathsf{Launch}(\pi_1, [\mathrm{Alice}, \mathrm{a4}], t_2)$ with $6 \leqslant t_1 \leqslant 6+3=9$, $8 \leqslant t_2 \leqslant 8+7=15$, and $t_2 \leqslant t_1+4$ must happen.

An assignment μ is a *potential violation* of a rule $r\colon \varphi \to \psi$ in an enactment η if η satisfies φ with μ and there is no assignment β that extends μ such that η satisfies ψ with β. A *(permanent) violation* μ of a rule $r\colon \varphi \to \psi$ is a potential violation where for every sequence of batches of future events $\Delta_1, ..., \Delta_n$ (where Δ_i is a batch for $\eta \cup (\cup_{j<i}\Delta_j)$ for each $1 \leqslant i \leqslant n$), μ is a violation of r in $\eta \cup (\cup_{i=1}^{n}\Delta_i)$. In the next section, we develop algorithms to identify when violations become permanent.

4 Techniques for Early Violation Detection

In this section, we develop key techniques for early violation detection. First, we define the concept of a "deadline" and present an algorithm to calculate deadlines. Next, we define data structures to store variable assignments and algorithms to create new assignments from arriving events. Finally, we detail how violations are detected. A monitoring algorithm using these techniques was implemented and experimentally evaluated in Sec. 6.

We aim to detect permanent violations as early as possible. Since an enactment is an accumulation of events with increasing timestamps, it may be that a complete body assignment derived from the current enactment can only be extended at or before a specific future time called a deadline. We now formulate the notion of a deadline.

▶ **Definition.** *Let Θ be a set of gap atoms over variables $x_1, ..., x_n$ and μ a (partial) assignment for variables x_i's. We use DEF_μ for the variables μ assigns a value; $\mu(\Theta)$ the gap atoms obtained by replacing each variable $x \in \mathrm{DEF}_\mu$ with $\mu(x)$, and $\max(\mu) = \max\{\mu(x) \mid x \in \mathrm{DEF}_\mu\}$. A timestamp $\tau \in \mathbb{N}$ is the* deadline *for $\Theta, x_1, ..., x_n, \mu$ if (1) $\tau \geqslant \max(\mu)$, and (2) either $\mu(\Theta)$ is unsatisfiable and $\tau = \max(\mu)$ or conditions (i) and (ii) both hold: (i) for each complete extension μ' of μ such that $\mu'(x) > \tau$ for each $x \notin \mathrm{DEF}_\mu$, $\mu'(\Theta)$ is false, and (ii) there is a complete extension μ'' of μ such that $\mu''(\Theta)$ is true.*

▶ **Example 2.** In the running example in Section 2, μ_{10} is created at time 8, where $\mu_{10}(x)=3$, $\mu_{10}(y)=6$, and $\mu_{10}(z)=8$. As shown in Fig. 3, applying μ_{10} to the head atoms yields upper bounds $w \leqslant 9$ $(=y+3)$ and $v \leqslant 15$ $(=z+7)$. From these bounds, it is clear that extensions of μ_{10} must have a $\mathsf{Payment}$ event whose time variable w is no later than time 9. Thus, the time 9 is a "deadline" for μ_{10}: the latest time μ_{10} can be extended w.r.t. w, and the earliest time μ_{10} could be recognized as a permanent violation. Fortunately, a $\mathsf{Payment}$ event happened

Algorithm 1 Deadline$(\Theta, x_1, ..., x_n, \mu)$.

Input: A set of gap atoms Θ over time variables $x_1, ..., x_n$ and an assignment μ
Output: A timestamp τ

```
1: if If μ(Θ) is unsatisfiable then return τ := max(μ);
                /* max(μ) is the largest timestamp μ assigns to x_1, ..., x_n */
2: Rewrite each atom in μ(Θ) in the form u ± k ⩽ v;
                /* u, v either a time variable or in ℕ, k ∈ ℤ */
3: Let UpperBd be a map from x_1, ..., x_n to {∞};
4: for each u ± k ⩽ v in μ(Θ) with v ∈ ℕ and u ∈ {x_1, ..., x_n} do
5:     UpperBd(u) := v ∓ k;
6: for |Θ| iterations do
7:     for each gap atom u ± k ⩽ v in μ(Θ) do
8:         if UpperBd(v) is finite and UpperBd(u) ± k > UpperBd(v) ⩾ 0 then
9:             UpperBd(u) := UpperBd(v) ∓ k;
10: return τ := min{UpperBd(x_i) | 1⩽i⩽n}
```

at time 9, which satisfies $w \leqslant 9$. However, v remains unresolved and thus the subsequent deadline to extend μ_{10} is the latest time to observe a value for v: $v \leqslant 13$ $(=w+4)$ and $v \leqslant 15$ $(=z+7)$, so the deadline to extend μ_{10} is changed to 13.

We compute deadlines with function Deadline (Alg. 1). Deadline determines for each x_i the least τ_i such that $\mu(\Theta) \to x_i \leqslant \tau_i$, and the deadline τ is the least of τ_i's. First, if $\mu(\Theta)$ is unsatisfiable, μ is a violation at the time of its creation, i.e., at its largest timestamp. Otherwise, an array *UpperBd* is initialized with constants (Lines 3-5), then tightened with the initial bounds and the gap atoms in Θ: a gap atom $u \pm k \leqslant v$ indicates $UpperBd(v) \mp k$ is an upper bound for u. For each gap atom $u \pm k \leqslant v$ for which $UpperBd(v)$ is defined, we update $UpperBd(u)$ as $\max(UpperBd(v) \mp k, UpperBd(u))$ (Lines 7-9).

The Deadline function (Alg. 1) can compute deadlines for complete body assignments and for complete body assignments with matching partial head assignments. For a complete body assignment μ and a partial head assignment β, we compute the latest time $\mu \cup \beta$ can be extended. This time is, in fact, the earliest time μ becomes a permanent violation. In the following lemma, we state a property of deadlines for a complete body assignment and partial head assignment.

▶ **Lemma 3.** *Let $r : \varphi \to \psi$ be a rule, φ_g, ψ_g the gap atoms in φ, ψ (resp.), μ a complete body assignment such that $\mu(\varphi_g)$ is true, β an incomplete head assignment extending μ such that $\beta(\mu(\psi_g))$ is satisfiable, and U the variables in ψ_g undefined by β. Let $\tau = \mathsf{Deadline}(\psi_g, var(\varphi_g \cup \psi_g), \mu \cup \beta)$. The following hold:*

1. *If $\tau \in \mathbb{N}$, then there is a complete head assignment β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is true,*
2. *If $\tau \in \mathbb{N}$, then for all complete head assignments β' extending $\mu \cup \beta$ such that $\min(\beta'(U)) > \tau$, $\beta'(\psi_g)$ is false, and*
3. *If $\tau = \infty$, then for all timestamps n in $\mathbb{N}$, there is a complete head assignment β' extending $\mu \cup \beta$ such that $\max(\beta'(U)) > n$ and $\beta'(\psi_g)$ is true.*

A sketch of the proof is given in Appendix A. The key idea is that for the combined assignment $\mu \cup \beta$ and atoms ψ, either for some time variable z and timestamp τ, $\mu(\beta(\psi)) \wedge (z \geqslant \tau')$ is unsatisfiable (so τ is a deadline) or no such time variable z and timestamp τ exists (there is no deadline). Lemma 3 is applied in the following way: for a complete body assignment μ, we try to extend μ with each partial head assignment β when it is created. For each pair μ and β, we calculate a deadline using μ, β, and the rule head. According to Lemma 3, the output of Deadline is the time after which β cannot extend μ.

The discussions in Section 2 also suggest maintaining partial and complete assignments for variables. We define three tables BA_r for body assignments, HA_r for head assignments, and EXT_r (extensions) to track pairings of body and head assignments. BA_r and HA_r consist of the following columns: (*i*) one column for the assignment identifier (*Aid*) from $\mathbf{I}$, (*ii*) one column for the enactment identifier (ID) from $\mathbf{I}$, (*iii*) one column in BA_r for each variable in φ and one column in HA_r for each event variable in ψ (resp.) (a variable in the head ψ is an *event variable* if it occurs in an event atom in ψ.) to hold a value from $\mathbf{D}$ or $\mathbf{T}$, and (*iv*) one column for gap atoms in φ and ψ (resp.) simplified with the assigned values as possible. Additionally, BA_r has one more column (*v*) *match?* indicating with *yes* or *no* the presence or absence, resp., of a complete head assignment extending the complete body assignment. For convenience, we refer to rows in these two tables as assignments. EXT_r has three columns: (*i*) one column for a body *Aid* from BA_r, (*ii*) one column for a head *Aid* from HA_r that extends the row's body assignment, and (*iii*) one column for the *deadline*, calculated using the row's assignments and the head gap atoms as inputs for Deadline.

For each enactment η, $\text{BA}_r(\eta)$ and $\text{HA}_r(\eta)$ store all assignments that can be generated from η and satisfy φ and ψ (resp.). Specifically, for a rule $r\colon\varphi\to\psi$ and an enactment η, $\text{BA}_r(\eta)$ contains every assignment μ such that for a non-empty subset P of the event atoms in φ, μ is defined for all variables in P, $\mu(P)\subseteq\eta$, and η satisfies all atoms in φ having only variables in P with μ. $\text{HA}_r(\eta)$ is similar, using ψ instead of φ. Fig. 5(a) shows the assignments inserted into BA_r table at time 10 (those from Fig. 2(b)) with columns for gap atoms and the possibility of matching. $\text{EXT}_r(\eta)$ stores each pair of assignments from $\text{BA}_r(\eta)$ and $\text{HA}_r(\eta)$, resp., such that the body assignment can be extended by the head assignment only at or before the row's deadline.

Aid	u	a	x	y	z	gap atoms	match?
μ_{10}	Alice	a4	3	6	8	-	No
μ_{11}	Alice	a3	-	-	9	$x\leqslant y\leqslant x{+}7$, $y\leqslant 9\leqslant y{+}7$	No
μ_{12}	Alice	a3	1	-	9	$1\leqslant y\leqslant 8$, $y\leqslant 9\leqslant y{+}7$	No
μ_{13}	Alice	a3	-	6	9	$x\leqslant 6\leqslant x{+}7$	No
μ_{14}	Alice	a3	1	6	9	-	No

(a) Some assignments in $\text{BA}_r(\pi_1)$ (Fig.2(a)) at $\mathsf{ts}=9$.

Aid	u	a	w	v	gap atoms
β_1	Alice	a3	8	-	$v\leqslant 12$
β_2	Alice	a4	9	-	$v\leqslant 13$
β_3	Alice	a3	-	12	$8\leqslant w$
β_4	Alice	a3	8	10	-

(b) Some assignments in $\text{HA}_r(\pi_1)$ (Fig.2(b)) at $\mathsf{ts}=10$.

Figure 5 Body and Head Table Examples.

body Aid	head Aid	deadline
μ_{10}	-	9
μ_{10}	β_2	13
μ_{14}	-	9
μ_{14}	β_1	12
μ_{14}	β_3	12
μ_{14}	β_4	-

Figure 6 Extensions of μ_{10} and μ_{14} in $\text{EXT}_r(\pi_1)$ at $\mathsf{ts}=13$.

We next present an algorithm called Update to create and combine assignments as batches of events arrive. This algorithm maintains BA and HA incrementally without accessing enactments directly; Update (Alg. 2) does not take an enactment as input. This is important since enactments may be very large.

We now outline the behavior of Update. Given atoms Θ (here, the head of a rule), a batch Δ, and either BA_r or HA_r for an enactment η, Lines 2-6 generate assignments from the events in Δ and Θ, adding them to the table if they are satisfiable (extendible to

Algorithm 2 Update$(\Theta, \Delta, T(\eta))$.

Input: A set of atoms Θ, a batch Δ, a table $T(\eta)$ (T is BA_r or HA_r for enactment η)
Output: Updated table $T(\eta \cup \Delta)$ for the new enactment $\eta \cup \Delta$

1: $\Gamma := T(\eta)$;
2: **for** each event $e \in \Delta$ **do**
3:   **for** each event atom γ in Θ with the same activity as e **do**
4:     Create a (partial) assignment μ from e, γ such that $\mu(\gamma) = e$;
5:     **if** $\mu(\Theta)$ is satisfiable **then**
6:       Add to Γ the row $s = \langle a, e.\text{ID}, \mu(v_1), ..., \mu(v_n), \text{B}, (\text{no})\rangle$, where a is a fresh assignment identifier, $v_1, ..., v_n$ are the event variables in Θ, and B the gap atoms in Θ, evaluated and simplified under μ;
7: **while** Γ *changes* **do**
8:   **for** each pair of unique and *consistent* rows μ_1 and μ_2 in T **do**
9:     $\mu := \text{MERGE}(\mu_1, \mu_2)$; /* *consistent*, MERGE explained in the text */
10:     Add to Γ the row: $s = \langle a, \mu_1.\text{ID}, \max(t_1, t_2), \mu(v_1), ..., \mu(v_n), \text{B}, (\text{no})\rangle$ where a is a fresh assignment identifier and B is the union of gap atoms in μ_1, μ_2, evaluated with μ ;
11: **output** Γ

complete assignments). The **while** loop in Lines 7-10 searches for pairs of *consistent* partial assignments, Two assignments are *consistent* if they agree on the variables for which they are both defined, e.g., in Fig. 2 μ_1 and μ_2 agree on u but not on a. If two assignments are consistent and satisfy the necessary gap atoms, a new assignment is created with MERGE, which combines their variable mappings and gap atoms and recomputes their deadline. For example, assignment μ_5 in Fig. 2 is the *merge* of μ_2 and μ_3. The loop only creates assignments whose data values are pre-existing in Γ or the batch Δ, i.e., it doesn't introduce new data values, so the **while** loop terminates.

▶ **Example 4.** For the enactment and rule in Section 2, consider the enactment's event $\mathsf{Request}(\pi_1, [\text{Alice}, \text{a3}], 1)$ and the rule's atom $\mathsf{Request}(\text{user } u, \text{account } a)@x$. The mapping $[\text{ID} \mapsto \pi_1, u \mapsto \text{Alice}, v \mapsto \text{a3}, x \mapsto 1]$ maps the atom to this event; the assignment corresponding to this mapping is added to BA_r as μ_1 in Fig. 2(a). For the same example in Section 2 and Fig. 2(a), assignments μ_2: $[\pi_1, \text{Alice}, \text{ a4}, 3, \text{-}, \text{-}, \{3 \leqslant y \leqslant 10, y \leqslant z \leqslant y+7\}]$ and μ_3: $[\pi_1, \text{Alice}, \text{-}, \text{-}, 6, \text{-}, \{x \leqslant 6 \leqslant x+7,\ 6 \leqslant z \leqslant 13\}]$ are in $\text{BA}_r(\pi_1)$ at $\mathsf{ts} = 9$ and agree on u. Their combination $\text{MERGE}(\mu_2, \mu_3)$ satisfies $x \leqslant 6 \leqslant x+7$ and $3 \leqslant y \leqslant 10$, so a row corresponding to $\text{MERGE}(\mu_2, \mu_3)$ is added to BA_r as μ_5.

The following lemma states that Update refreshes the body and head tables by adding the partial and complete assignments with values from Δ as expected.

▶ **Lemma 5.** *Let $r: \varphi \to \psi$ be a rule, η an enactment, and Δ a batch for η.* Update$(\varphi, \Delta, \text{BA}_r(\eta))$ *(or* Update$(\psi, \Delta, \text{HA}_r(\eta))$*) computes $\text{BA}_r(\eta \cup \Delta)$ (resp. $\text{HA}_r(\eta \cup \Delta)$).*

A sketch of the proof is given in Appendix A. The key idea is that for an assignment in $\text{BA}_r(\eta \cup \Delta)$, some data values may come from events in η so they will be in $\text{BA}_r(\eta)$ and some may come from events in Δ, in which case they will be introduced in Line 4 of Alg. 2 and merged with other assignments in the loop of Line 7 of Alg. 2. The proof is similar for $\text{HA}_r(\eta \cup \Delta)$.

The EXT table pairs complete body assignments with partial and complete head assignments along with a deadline. When a batch arrives, Update-E (Alg. 3) adds new complete body assignments to EXT (Lines 2-3), and then adds pairs using head assignments (Lines 4-8), computing a deadline for each pair (Line 8). Line 9 checks if there is a match between complete body and head assignments, and updates BA if so.

Algorithm 3 Update-E($\Delta, \textsc{ext}_r(\eta), \textsc{ba}_r(\eta\cup\Delta), \textsc{ha}_r(\eta\cup\Delta)$).

Input: A batch Δ, un-updated table $\textsc{ext}_r(\eta)$,
updated tables $\textsc{ba}_r(\eta\cup\Delta)$ and $\textsc{ha}_r(\eta\cup\Delta)$ for an enactment η
Output: Updated table $\textsc{ext}_r(\eta \cup \Delta)$

$\Gamma := \textsc{ext}_r(\eta)$;
for each complete body assignment μ in $\textsc{ba}_r(\eta\cup\Delta)$ **do**
  if $\max(\mu) = \mathsf{ts}_\Delta$ **then** Add $\langle \mu, \text{-}, \mathsf{Deadline}(\psi, var(\psi), \mu)\rangle$ to Γ ;
for each assignment γ in $\textsc{ha}_r(\eta\cup\Delta)$ **do**
  if $\max(\gamma) = \mathsf{ts}_\Delta$ **then**
    for each row $\langle \mu, \beta, d\rangle$ in Γ **do**
      if γ extends $\mu \cup \beta$ and $\gamma(\mu(\psi))$ is satisfiable **then**
        Add $\langle \mu, \gamma, \mathsf{Deadline}(\psi, var(\psi), \mu \cup \gamma)\rangle$ to Γ ;
        if γ is complete **then** Update $\textsc{ba}_r(\eta\cup\Delta)$ to indicate μ has a match ;
output Γ ; /* $= \textsc{ext}_r(\eta \cup \Delta)$ */

For all complete body assignments, EXT stores each head assignment that extends it and indicates the latest time the pair can be further extended. The following lemma characterizes the conditions and time whereby a violation can be detected using EXT.

▶ **Lemma 6.** *Let η be an enactment with no* END *event, r a rule, τ a timestamp, and μ a complete body assignment for r. Then, μ is a permanent violation of r in η at τ iff μ occurs in $\textsc{ext}_r(\eta)$ but no rows in $\textsc{ext}_r(\eta)$ pairs μ with a complete head assignment, and each row in $\textsc{ext}_r(\eta)$ with μ has a deadline no greater than τ.*

A sketch of the proof is given in Appendix A. The key idea is that by Lemma 3, the largest deadline τ for μ and a partial match β in $\textsc{ext}_r(\eta)$ represent the time beyond which any assignment extending β derived from a future event will have a timestamp that is inconsistent with ψ. Thus, μ must be extended on or before time τ in order to be matched with β.

▶ **Example 7.** In Section 2, μ_{10} satisfies φ and must be extended no later than 9. Then, the deadline for matching the unpaired μ_{10} in $\textsc{ext}_r(\eta_{\leqslant 9})$ is 9. At time 9, a Payment event creates β_2 (Fig. 5), and μ_{10} and β_2 are inserted into $\textsc{ext}_r(\eta_{\leqslant 9})$ with deadline 13 because $\beta_2(w) = 9$ and ψ contains $v \leqslant w + 4$. Assuming no matching Launch event arrives, μ_{10} can be reported as a violation at time 13.

We now present the algorithm Detect (Algorithm 4) that detects permanent violations. These are unmatched body assignments in EXT (1) whose largest deadline is less than or equal to the current time or (2) whose enactments have ended.

Algorithm 4 Detect($\Delta, \textsc{ext}_r(\eta\cup\Delta)$).

Input: A batch Δ, updated $\textsc{ext}_r(\eta\cup\Delta)$
Output: A set of assignments indicating rule violations

Violations := {};
for each complete body assignment μ in $\textsc{ext}_r(\eta\cup\Delta)$ **do**
  if μ is not extended by any complete head assignment **then**
    if Δ contains an *END* event e with $e.\textsc{id} = \mu.\textsc{id}$ **then**
      Add μ to *Violations* ;
    Let τ be the maximum *deadline* for the rows in $\textsc{ext}_r(\eta\cup\Delta)$ with μ;
    if $\mathsf{ts}_\Delta \geqslant \tau > max(\eta)$ **then**
      Add μ to *Violations* ;
output Violations ;

In the following theorem, we assert that applying Algorithm 4 reports rule violations at the earliest possible time.

▶ **Theorem 8.** *Let r be a rule, η an enactment, and Δ a batch for η. Then, μ is a violation in $\eta\cup\Delta$ but not in η iff* Detect*$(\Delta, \textsc{ext}_r(\eta\cup\Delta))$ reports μ.*

A sketch of the proof is given in Appendix A. The key idea is that for a given body assignment μ in $\textsc{ext}_r(\eta\cup\Delta)$, by Lemma 6, if μ is a violation, it will be in exclusively unmatched rows in $\textsc{ext}_r(\eta\cup\Delta)$ with a deadline of at most ts_Δ. Then, when Δ is processed, μ can be recognized and reported.

From Theorem 8, we see that our monitoring algorithm reports exactly the set of violations in the enactment as soon as they are permanent. This concludes the presentation of the data structures and sub-routines used in our monitoring algorithm.

5 Optimizations

While the algorithms presented in Section 4 handle the monitoring task, their time and space complexities can be improved. We present one optimization to remove useless assignments using a similar reasoning to deadline calculation, another to avoid repeated computation by tracking which data is new. We report their improvement of relevant algorithms as a factor of the log size $|L|$, the batch size $|\Delta|$, the number of active enactments as approximated by $|\Delta|$, and the number of event atoms in the rule body or head e.

Expiring partial assignments. Early violation detection motivates a similar technique for discarding useless assignments. Partial assignments in BA and HA are *expired* (i.e., useless) if (1) they can no longer be extended because their timestamps and unresolved gap atoms are inconsistent with all possible future assignments, or (2) they are derived from an enactment that has ended. It is much desired to remove expired assignments, and thus reduce the sizes of BA and HA. Calculating expiration times resembles deadline calculation; in fact, the Deadline function is reused. To incorporate expiration time, we augment BA and HA (resp.) with an *expiration* column as new tables BAE and HAE, requiring that incomplete assignments in BAE and HAE be extendable by future events to complete assignments. To maintain this property, Deadline calculates its expiration time for each incomplete assignment with respect to its unresolved gap atoms. Removing expired assignments reduces the size of the BAE and HAE tables from $O(|L|^e)$ to $O(|\Delta|^e)$, which benefits the algorithms in § 4 by reducing the number of computations in Update from $O(|L|^{2e})$ to $O(|\Delta|^{2e})$, and that in Update-E from $O(|L|^e)$ to $O(|\Delta|^e)$. It also improves Update-E by decreasing the number of assignments checked for insertion into EXT (Lines 2 and 4), from $O(|L|^e)$ to $O(|\Delta|^e)$.

Semi-naive MERGE **of assignments.** We can also decrease the number of computations in the Update algorithm by tracking which data generated by the most recent batch. The **while** loop (Lines 7-10) in Update tests pairs of assignments to merge. For each batch Δ, we only need to try pairs that have at least one assignment added from events in Δ, because all other pairs of assignments were considered before Δ arrived. To make Update to reflect this, we use a queue Γ_{new} to hold new assignments generated at Line 6. We exchange the **for** loop in Update (Lines 8-10) to a doubly nested for-loop that iterates through each assignment μ_n in Γ_{new} (outer loop) and each row μ_o in Γ (inner loop), adding the new assignment to the queue Γ_{new}, moving μ_n from Γ_{new} to Γ after processing μ_n. This resembles “semi-naive” evaluation of Datalog programs [1] and reduces the search for matching assignments from considering $O(|L|^{2e}|)$ pairs to only pairs involving some new data: $O(|L|^e|\Delta|^e)$ pairs.

6 Experimental Evaluation

We implemented (Python 3.8.2) a monitoring algorithm using the data structures and algorithms in Section 4 and optimizations in Section 5. Moreover, our implementation handles multiple enactments simultaneously. Using this implementation, we experimentally evaluated the benefits and costs of early violation detection (EVD) and the overall batch processing times. We used logs created by simulating workflow models of the IaaS application in Section 2 with a simulator [25], varying the size of enactments from *normal* enactments (10 events per enactment) to *large* enactments (100 events per enactment) and using batch sizes of 100, 1,000, and 10,000 events. We used logs with an average of 100 concurrent enactments and monitored both *simple* rules (1-2 body atoms, 1-2 head atoms) and *complex* rules (2-4 body atoms, 2-4 head atoms). Our test data is motivated by discovering the feasible ranges for monitoring for enactment and batch size in five target applications areas: (1) healthcare information systems that manage medical services for compliance with patients' medical history, (2) drone management services that enforce geographic fencing and limits on flight time, (3) college admissions portals that manage application due dates and admission decisions, (4) IaaS providers, as illustrated above, and (5) retail websites where customers' orders must be paid for, filled, and delivered in a timely manner. For all experiments, we used a Mac laptop (MacOS Big Sur 12.2.1) with a 3.2 GHz, 8-core Apple M1 processor with 8GB memory.

Our experimental results indicate that early violation detection yields a significant resource savings (Finding 1) with a negligible overhead (Finding 2), and is feasible for enactments with up to 100 events and batches up to 10,000 events (Finding 3). Additionally, we can conclude that our algorithms are appropriate for some application areas of business services.

Finding 1. 16% of events in normal-length violating enactments and 66% of events in large violating enactments may be ignored.

First, we examine how soon violations could be detected with respect to each enactment's events. We report the average percentage of events observed in violating enactments before and after their first reported violation. This number represents the percentage of events that could be ignored, or even prevented, in the case that detecting a violation early halts the enactment's execution. This finding is partially dependent on the percentage of enactments that are violating and the size of gaps in rules as a proportion of enactment duration; future work could analyze these dimensions as factors of the potential savings. Fig. 7 shows the percentage of events observed in violating enactments before and after their first detected violation.

	normal-length enactments		large enactments	
rules	% events before first violation	% after	% events before first violation	% after
simple	74.9	25.1	33.5	66.5
complex	83.7	16.3	69.4	30.6

Figure 7 Percentages of events observed before and after the first detected violation.

Finding 2. The overhead of detecting violations early is $\leqslant$15% compared with the overall processing time, even for large enactments and rules with up to 8 atoms.

The benefits of early violation detection could be nullified if the time to calculate deadlines and find matches is a significant percentage of the overall processing time. As a baseline, we used an algorithm that does not calculate deadlines, and instead, detects and reports violations only once the enactment's *END* event arrives. Fig. 8 compares our monitoring algorithm with EVD to the baseline algorithm (without EVD). The increase in processing time with EVD for normal enactments (⩽2%) is less than the increase with EVD for large enactments (⩽15%). This is attributed to the higher number of events with matching data values in large enactments, which increases the number of assignment pairs, thus more deadlines are calculated in Lines 3 and 8 of Algorithm 3.

	normal-length enactments		large enactments	
rules	without EVD	with EVD	without EVD	with EVD
simple	4.27×10^{-2}	4.73×10^{-2} (+0.2%)	6.65×10^{-2}	7.60×10^{-2} (+14.3%)
complex	9.19×10^{-2}	9.31×10^{-2} (+1.3%)	1.840×10^{-1}	2.084×10^{-1} (+13.3%)

Figure 8 Batch processing times (seconds) with and without early violation detection.

Finding 3. Monitoring is feasible for enactments with up to 100 events, and batches of up to 10,000 events, with an arrival rate of 1 second.

We report the average batch processing time for normal and large enactments, simple and complex rules, and batches of 100, 1,000, and 10,000 events. Logs with larger batches were not obtained due to limitations of the simulator. We assume a batch arrival interval of 1 second, thus an average processing time ⩽1 second indicates monitoring is feasible for some application areas, because each batch can (on average) be processed before the following batch arrives, thus no backlog of events accumulates over time. Fig. 9 shows that the average processing time is ⩽1 second for all trials.

As the batch size grows, the number of events processed by Algorithm 2 grows proportionally. Given that most events in a batch are from different enactments, larger batches do not have proportionally more pairs of assignments to compare in Line 8 of Algorithm 2, so these times grow linearly with the batch size as expected. As the enactment length grows, the number of compatible events, and thus partial assignment pairs, grows, increasing the number of matches in Line 8 of Algorithm 2 and the number of updates to the EXT table in Line 4 of Algorithm 3. Then, enactment length accounts for the increase in processing time.

Lastly, we place the results in context for the five application areas. Given that the batch processing times in Finding 3 for enactments with 100 events, batches of 10,000 events, and rules with 8 atoms are below our assumed batch interval of 1 second, applying our algorithms to applications in areas (1) and (2), which feature similar dimensions for enactments and constraints, is feasible. It is also feasible for small applications in areas (3), (4), and (5), though monitoring larger applications with hundreds of thousands of concurrent users or enactments with thousands of events may not be possible. Additionally, Finding 2 suggests whenever monitoring is feasible, early violation detection is also feasible, as it has negligible computational overhead.

	enactment length			
	normal	large	normal	large
batch size	simple rules		complex rules	
100	4.55×10^{-4}	6.19×10^{-4}	7.74×10^{-4}	1.363×10^{-3}
1,000	4.330×10^{-3}	6.177×10^{-3}	7.534×10^{-3}	1.3509×10^{-2}
10,000	4.2414×10^{-2}	6.0769×10^{-2}	7.4925×10^{-2}	1.35218×10^{-1}

Figure 9 Batch processing times (seconds) for different enactments and rules.

7 Conclusions

Techniques for event monitoring are increasing in demand as more software systems generate and/or rely on events. This paper contributes monitoring and violation detection techniques for temporal constraints in workflow systems. More study is needed of the trade-offs of expressiveness of temporal constraints, specifically a comparison of our language's gap atoms with LTL and MTL, as well with extensions of our language with negation for modeling the absence of events. Additionally, it remains to be seen if early violation detection is possible, and then more effective and efficient, with respect to sets of rules, where deadlines may appear earlier due to interactions of "conflicting" constraints, as in [19]. Also, our techniques only consider whether or not a violation is certain; it may be useful to reason about violations probabilistically, which could allow them to be anticipated farther in advance and thus better mitigated.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2 Howard Barringer, Ylies Falcone, Bernd Finkbeiner, Klaus Havelund, Insup Lee, Gordon Pace, Grigore Rosu, Oleg Sokolsky, and Nikolai Tillmann. *Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418. Springer, 2010.
3 David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring metric first-order temporal properties. *Journal of the ACM (JACM)*, 62(2):1–45, 2015.
4 Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.
5 Saoussen Cheikhrouhou, Slim Kallel, Nawal Guermouche, and Mohamed Jmaiel. On enabling time-aware consistency of collaborative cross-org. business processes. In *ICSOC 2014*, pages 351–358. Springer, 2014.
6 Carlo Combi and Roberto Posenato. Towards temporal controllabilities for workflow schemata. In *TIME 2010*, pages 129–136. IEEE, 2010.
7 Riccardo De Masellis, Fabrizio M Maggi, and Marco Montali. Monitoring data-aware business constraints with finite state automata. In *Proceedings of the 2014 International Conference on Software and System Process*, pages 134–143, 2014.
8 Riccardo De Masellis and Jianwen Su. Runtime enforcement of first-order ltl properties on data-aware business processes. In *International Conference on Service-Oriented Computing*, pages 54–68. Springer, 2013.
9 Christophe Dousson and Pierre Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI*, volume 7, pages 324–329, 2007.
10 Johann Eder, Marco Franceschetti, and Julius Köpke. Controllability of business processes with temporal variables. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 40–47, 2019.
11 Ashish Gupta, Inderpal Singh Mumick, and Venkatramanan Siva Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166, 1993.
12 Klaus Havelund and Grigore Rosu. Monitoring programs using rewriting. In *ASE 2001*, pages 135–143. IEEE, 2001.
13 Luke Hunsberger and Roberto Posenato. Sound-and-complete algorithms for checking the dynamic controllability of conditional simple temporal networks with uncertainty. In *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
14 Linh Thao Ly, Fabrizio Maria Maggi, Marco Montali, Stefanie Rinderle-Ma, and Wil MP Van Der Aalst. Compliance monitoring in business processes: Functionalities, application, and tool-support. *Information systems*, 54:209–234, 2015.

15 Isaac Mackey and Jianwen Su. Mapping business rules to ltl formulas. In *ICSOC 2019*, pages 563–565, 2019.

16 Isaac Mackey and Jianwen Su. Mapping singly-linked, acyclic rules to linear temporal logic formulas. In submission, 2022.

17 Fabrizio Maria Maggi, Marco Montali, and Ubaier Bhat. Compliance monitoring of multi-perspective declarative process models. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 151–160. IEEE, 2019.

18 Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil MP Van Der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *International Conference on Business Process Management*, pages 132–147. Springer, 2011.

19 Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil MP van der Aalst. Runtime verification of ltl-based declarative process models. In *International Conference on Runtime Verification*, pages 131–146. Springer, 2011.

20 Alessandro Margara, Emanuele Della Valle, Alexander Artikis, Nesime Tatbul, and Helge Parzyjegla, editors. *International Conference on Distributed and Event-Based Systems*. ACM, ACM, 2021.

21 Marco Montali, Federico Chesani, Paola Mello, and Fabrizio M Maggi. Towards data-aware constraints in declare. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1391–1396, 2013.

22 Marco Montali, Fabrizio M Maggi, Federico Chesani, Paola Mello, and Wil MP van der Aalst. Monitoring business constraints with the event calculus. *ACM TIST 2014*, 5(1):1–30, 2014.

23 Milos Nikolic, Mohammad Dashti, and Christoph Koch. How to win a hot dog eating contest: Distributed incremental view maintenance with batch updates. In *Proceedings of the 2016 International Conference on Management of Data*, pages 511–526, 2016.

24 Peter Z Revesz. A closed-form evaluation for datalog queries with integer (gap)-order constraints. *Theoretical Computer Science*, 116(1):117–149, 1993.

25 Gabriel Siqueria. Log generator. `https://github.com/GabrielSiq/LogGenerator`, 2020.

A Proof Sketches of Lemmas 3, 5, 6 and Theorem 8

▶ **Lemma 3.** *Let $r\colon\varphi\to\psi$ be a rule, φ_g, ψ_g the gap atoms in φ, ψ (resp.), μ a complete body assignment such that $\mu(\varphi_g)$ is true, β an incomplete head assignment extending μ such that $\beta(\mu(\psi_g))$ is satisfiable, and U the variables in ψ_g undefined by β. Let $\tau = \mathsf{Deadline}(\psi_g, \mathit{var}(\varphi_g \cup \psi_g), \mu\cup\beta)$. The following hold:*

1. *If $\tau \in \mathbb{N}$, then there is a complete head assignment β' extending $\mu\cup\beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is true,*
2. *If $\tau \in \mathbb{N}$, then for all complete head assignments β' extending $\mu\cup\beta$ such that $\min(\beta'(U)) > \tau$, $\beta'(\psi_g)$ is false.*
3. *If $\tau = \infty$, then for all timestamps n in $\mathbb{N}$, there is a complete head assignment β' extending $\mu\cup\beta$ such that $\max(\beta'(U)) > n$ and $\beta'(\psi_g)$ is true.*

Proof Sketch for Lemma 3. To show (1), assume there is no complete head assignment β' extending $\mu\cup\beta$ such that $\min(\beta'(U)) \leqslant \tau$ and $\beta'(\psi_g)$ is *true*. Then, $(\mu \cup \beta)(\psi) \wedge (z = \tau)$ is not satisfiable. Then, there is a gap atom in $\mu \cup \beta(\psi)$ that provides an upper bound for z below τ. Then, τ is not the minimum of the upper bounds in *UpperBd*. Thus Algorithm 1 on $\mu \cup \beta$ and ψ should not output τ. This is a contradiction. To show (2), assume some complete head assignment β' extends $\mu\cup\beta$ such that $\min(\beta'(U)) > \tau$ and $\beta'(\psi_g)$ is true. Then, $(\mu \cup \beta)(\psi) \wedge (z = \tau')$ is satisfiable for some z in $\mathit{var}(\psi)$. Then, $\mu(\psi)$ does not imply $z_i \leqslant \tau$ for all variables z_i. Thus Algorithm 1 on $\mu \cup \beta$ and ψ should not output τ. This is a contradiction. To show (3), assume $\tau = \infty$. Algorithm 1 only produces ∞ when $\mu(\psi)$ is satisfiable and for some variable z_i and for all $n \in \mathbb{N}$, $\mu(\psi) \not\to (z_i \leqslant n)$ Then, for all timestamps n in $\mathbb{N}$, there is some complete assignment that extends μ, satisfies ψ, and uses some n' larger than n. Then, μ can be extended arbitrary far in the future. ◀

▶ **Lemma 5.** *Let $r\colon \varphi \to \psi$ be a rule, η an enactment, and Δ a batch for η.* $\mathsf{Update}(\varphi, \Delta, \textsc{ba}_r(\eta))$ *(or* $\mathsf{Update}(\psi, \Delta, \textsc{ha}_r(\eta))$*) computes $\textsc{ba}_r(\eta \cup \Delta)$ (resp. $\textsc{ha}_r(\eta \cup \Delta)$).*

Proof Sketch for Lemma 5. We argue this for $\textsc{ba}_r(\eta)$; adapting this argument for $\textsc{ha}_r(\eta)$ is trivial. For an assignment μ in $\textsc{ba}_r(\eta{\cup}\Delta)$ created by $\mathsf{Update}(\varphi, \Delta, \textsc{ba}_r(\eta))$, some events C in η and some D in Δ provide values for μ. Then an assignment μ_C for C is present in $\textsc{ba}_r(\eta)$ and Lines 2–5 of Alg. 2 generates $|D|$ assignments for each event in D. Next, these $|D| + 1$ assignments will merge with each other in the loop of Line 7 of Alg. 2 until μ is created and added to Γ. Alternatively, consider any assignment μ that is not in $\textsc{ba}_r(\eta{\cup}\Delta)$ after Algorithm 2. Then, no subset of events in $\eta{\cup}\Delta$ can create μ on Line 4 or μ is inconsistent with the rule body or head and will not proceed past Lines 5 or 8 of Alg. 2. ◀

▶ **Lemma 6.** *Let η be an enactment with no* END *event, r a rule, τ a timestamp, and μ a complete body assignment for r. Then, μ is a violation of r in η iff μ occurs in $\textsc{ext}_r(\eta)$ but no rows in $\textsc{ext}_r(\eta)$ pairs μ with a complete head assignment, and each row in $\textsc{ext}_r(\eta)$ with μ has a deadline no greater than τ.*

Proof Sketch for Lemma 6. Let τ be the largest timestamp in η. $\textsc{ext}_r(\eta)$ contains all possible pairs for μ and head assignments from $\textsc{ha}_r(\eta)$, so if μ is unmatched in $\textsc{ba}_r(\eta)$, there is no assignment with $min(\beta) \leqslant \tau$ that extends μ and satisfies ψ. Alternatively, let τ be the largest deadline for μ in $\textsc{ext}_r(\eta)$, by Lemma 3, for all rows with μ and β in $\textsc{ext}_r(\eta)$, for all complete head assignments β' that extend $\mu \cup \beta$, such that $min(\beta'(U)) > \tau$, $\beta'(\psi)$ is inconsistent. Thus, no future (i.e., with a value greater than τ) complete head assignment can extend μ and satisfy ψ. Then, μ will never be extended by a complete head assignment that satisfies ψ, so μ is a violation for η. ◀

▶ **Theorem 8.** *Let r be a rule, η be an enactment, and Δ a batch for η. Then, μ is a violation in $\eta{\cup}\Delta$ but not in η iff* Detect*$(\Delta, \textsc{ext}_r(\eta{\cup}\Delta))$ reports μ.*

Proof Sketch for Theorem 8. Let μ be a violation in $\eta{\cup}\Delta$. $\eta{\cup}\Delta$ may contain an *END* event and will have no later events, in which case, η.*END* is in Δ and μ will be added to *Violations* on Line 5 of Algorithm 4. Otherwise, by Lemma 6, μ is complete and in exclusively unmatched rows in $\textsc{ext}_r(\eta{\cup}\Delta)$ with a deadline of, at most, ts_Δ. Then, μ will be added to *Violations* on Line 8 of Algorithm 4.

Conversely, if $\mathsf{Detect}(\Delta, \textsc{ext}_r(\eta{\cup}\Delta))$ reports μ, then μ is added to *Violations* on Line 5 or Line 8 of Algorithm 4. Given Line 2 of the algorithm, μ must be a complete assignment in $\textsc{ext}_r(\eta{\cup}\Delta)$ that is not extended by any complete head assignment. Then, either (1) η.*END* in Δ or (2) ts_Δ is greater than or equal to the deadline for μ in all rows in $\textsc{ext}_r(\eta{\cup}\Delta)$. If (1), then μ is a violation because $\eta{\cup}\Delta$ will have no later events. If (2), μ is a violation in $\eta{\cup}\Delta$ by Lemma 6. ◀