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:
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 improv-
ing 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 specific-
ation. 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 $\pi_1$ and $\pi_2$. For example, the first row of the Request table indicates a Request event with enactment id $\pi_1$ from user Alice with account $a_3$ at time 1.

![Figure 1 Database $S_9$ with events from two enactments $\pi_1$ and $\pi_2$.](image)

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 $\varphi \rightarrow \psi$ where $\varphi$ is the rule body and $\psi$ is the rule head:

\[
\begin{align*}
\text{Request}(u, a)@x, & \quad \text{Approval}(u)@y, x \leq y \leq x+7, \quad \text{Reserve}(u, a)@z, y \leq z \leq y+7 \\
& \rightarrow \quad \text{Payment}(u, a)@w, \quad \text{Launch}(u, a)@v, y \leq w \leq y+3, \quad z \leq v \leq z+7, \quad v \leq w + 4
\end{align*}
\]

The core idea of detecting a violation is checking whether each body assignment for the body variables $u, a, x, y, z$ satisfying $\varphi$ has a matching head assignment for the head variables $u, a, w, v$ satisfying $\psi$. 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 $\varphi$.
4:4 Early Detection of Temporal Constraint Violations

and $S_o$. For example, assignment $\mu_1$ is generated by the first row (event) of the Request table. Assignment $\mu_3$ is generated by the first row of the Approval table, and is combined with $\mu_1$ to form $\mu_e$. Then, $\mu_{10}$ and $\mu_{14}$ makes $\varphi$ true.

Suppose two events happen at time 10, $e_1$: Approval($\pi_i$, [Alice], 10), $e_2$: Approval($\pi_j$, [Bob], 10). Event $e_1$ generates a partial assignment $\mu_{15}$, which combines with $\mu_2$ into $\mu_{16}$. Event $e_2$ yields new assignments $\mu_{17}$ and $\mu_{18}$. Fig. 2(b) lists four assignments generated by $e_1$ and $e_2$.

$\psi$ 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 $\psi$ in the same manner as for $\varphi$ but ignoring $y$ and $z$. The Payment events at times 8, 9 (Fig. 1) create partial assignments $\beta_1$: [$\pi_1$, Alice, a3, 8, -] and $\beta_2$: [$\pi_1$, Alice, a4, 9, -].

One interesting problem is to determine when to report rule violations. In Fig. 3, three events create a potential violation $\mu_{10}$. It is natural to report this violation when the end of enactment $\pi_i$ arrives, after which no more events for $\pi_i$ will arrive; if $\mu_{10}$ 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 $y \leq w \leq y + 3$ and $z \leq v \leq z + 7$, 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 $\mu_{10}$, and that 9 is the earliest time we can be certain this is a violation.

Fig. 4 shows a Payment event at time 9 that creates the partial assignment $\beta_2$.

The main focus of this paper is to calculate these earliest times, which we call “deadlines”, and use them in a monitoring algorithm.
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 \( I \) (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 \( T \) with a discrete total order and addition of constants. For technical development, we use natural numbers for timestamps.

An instance of a workflow is a finite set of (data) values \( D = \{ a, b, c, ..., a_1, ... \} \) with equality.

**Definition.** An event of an activity \( A(c_1, ..., c_n) \) is a named tuple \( A(\xi, \nu, \tau) \) where \( \xi \) is an ID from \( I \), \( \nu: \{ c_1, ..., c_n \} \rightarrow D \) is a mapping from \( A \)’s attributes to data values, and \( \tau \) is a timestamp from \( T \).

An instance of a workflow is a finite set \( \eta \) of events called an enactment, such that (i) each event has the same enactment ID, (ii) \( \eta \) has exactly one special \( \text{START} \) event that marks its beginning and of workflow enactments and at most one \( \text{END} \) event that marks its completion, (iii) the timestamp of the \( \text{START} \) event is less than that of all other events in \( \eta \), and (iv) the timestamp of the \( \text{END} \) event, if it occurs, is greater than that of all other events in \( \eta \). The rows in Fig.1 with the same ID form an enactment (the \( \text{START/END} \) 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 \( \eta \) is a finite set \( \Delta \) of events such that (i) all events in \( \Delta \) have the same timestamp, denoted as \( ts_{\Delta} \), greater than the timestamps of all events in \( \eta \), (ii) for each event \( e \) in \( \Delta \), the ID of \( e \) is the ID of \( \eta \), (iii) \( \Delta \) has a \( \text{START} \) event or \( \eta \) has a \( \text{START} \) event, but not both, and (iv) if an \( \text{END} \) event is in \( \eta \), no events are in \( \Delta \).

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

We describe a language for specifying rules, starting with atomic formulas. An event atom is an expression “\( A(v_1, ..., v_n)@r \)” where \( A(c_1, ..., c_n) \) is an activity, \( v_1, ..., v_n \) are variables, where \( x \) is a time variable. A gap atom is an expression “\( x \pm \theta y \)” where \( x, y \) are time variables, \( \epsilon \) (the gap) is a timestamp in \( T \), and \( \theta \in \{ <, \leq, >, \geq, = \} \) is an equality or inequality predicate. We denote the variables in a set of gap atoms \( \varphi \) as \( \text{var}(\varphi) \).

**Definition.** A rule is an expression “\( \varphi \rightarrow \psi \)” where the body \( \varphi \) and the head \( \psi \) are finite sets of event and gap atoms such that each variable in a gap atom in \( \varphi \) occurs in an event atom in \( \varphi \) and each variable in a gap atom in \( \psi \) 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 \( D \cup T \). Time variables take values from \( T \); we use \( \mathbb{N} \) as timestamps for technical development. All other variables take values from \( D \). An assignment is complete if it is a total mapping for the variables in a given set of atoms, partial otherwise. An assignment \( \beta \) extends an assignment \( \alpha \) if \( \alpha \subseteq \beta \). An enactment \( \eta \) satisfies an event atom \( A(v_1, \ldots, v_n) \in \alpha \) for the activity \( A(c_1, \ldots, c_n) \) with a complete assignment \( \mu \) if \( \mu(A, \{ c_1 \mapsto \mu(v_1), \ldots, c_n \mapsto \mu(v_n) \}, \mu(x)) \) is an event in \( \eta \). An assignment satisfies a gap atom with the obvious interpretation.

An enactment \( \eta \) satisfies a set of atoms \( \phi \) with a complete assignment \( \mu \) if \( \eta \) satisfies every atom in \( \phi \) with \( \mu \). An enactment \( \eta \) satisfies a rule \( r: \varphi \rightarrow \psi \) if for every complete assignment \( \mu \) such that \( \eta \) satisfies \( \varphi \) with \( \mu \), there is a complete assignment \( \beta \) that extends \( \mu \) such that \( \eta \) satisfies \( \psi \) with \( \beta \).

**Example 1.** As shown in Fig. 3, the assignment \( \mu_{10} \) satisfies \( \varphi \). Then, to satisfy the rule w.r.t. \( \mu_{10} \), there must be an assignment extending \( \mu \) that satisfies \( \psi \); i.e., two events \( \text{Payment}(\pi, [\text{Alice}, a4], t1) \) and \( \text{Launch}(\pi, [\text{Alice}, a4], t2) \) with \( 6 \leq t_1 \leq 6+3=9 \), \( 8 \leq t_2 \leq 8+7=15 \), and \( t_2 \leq t_1 + 4 \) must happen.

An assignment \( \mu \) is a potential violation of a rule \( r: \varphi \rightarrow \psi \) in an enactment \( \eta \) if \( \eta \) satisfies \( \varphi \) with \( \mu \) and there is no assignment \( \beta \) that extends \( \mu \) such that \( \eta \) satisfies \( \psi \) with \( \beta \). A (permanent) violation \( \mu \) of a rule \( r: \varphi \rightarrow \psi \) is a potential violation where for every sequence of batches of future events \( \Delta_1, \ldots, \Delta_n \) (where \( \Delta_i \) is a batch for \( \eta \cup (\cup_{j<i} \Delta_j) \) for each \( 1 \leq i \leq n \), \( \mu \) 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 \( \Theta \) be a set of gap atoms over variables \( x_1, \ldots, x_n \) and \( \mu \) a (partial) assignment for variables \( x_i \)’s. We use \( \text{DEF}_\mu \) for the variables \( \mu \) assigns a value; \( \mu(\Theta) \) the gap atoms obtained by replacing each variable \( x \in \text{DEF}_\mu \) with \( \mu(x) \), and \( \max(\mu) = \max\{\mu(x) \mid x \in \text{DEF}_\mu\} \). A timestamp \( \tau \in \mathbb{N} \) is the deadline for \( \Theta, x_1, \ldots, x_n, \mu \) if (1) \( \tau \geq \max(\mu) \), and (2) either \( \mu(\Theta) \) is unsatisfiable and \( \tau = \max(\mu) \) or conditions (i) and (ii) both hold: (i) for each complete extension \( \mu' \) of \( \mu \) such that \( \mu'(x) > \tau \) for each \( x \notin \text{DEF}_\mu \), \( \mu'(\Theta) \) is false, and (ii) there is a complete extension \( \mu'' \) of \( \mu \) such that \( \mu''(\Theta) \) is true.

**Example 2.** In the running example in Section 2, \( \mu_{10} \) 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 \( \mu_{10} \) to the head atoms yields upper bounds \( w \leq 9 \) (\( =y+3 \)) and \( v \leq 15 \) (\( =z+7 \)). From these bounds, it is clear that extensions of \( \mu_{10} \) must have a Payment event whose time variable \( w \) is no later than time 9. Thus, the time 9 is a “deadline” for \( \mu_{10} \); the latest time \( \mu_{10} \) can be extended w.r.t. \( w \), and the earliest time \( \mu_{10} \) could be recognized as a permanent violation. Fortunately, a Payment event happened
Lemma 3. Let $r: \varphi \rightarrow \psi$ be a rule, $\varphi, \psi$ the gap atoms in $\varphi, \psi$ (resp.), $\mu$ a complete body assignment such that $\mu(\varphi_g)$ is true, $\beta$ an incomplete head assignment extending $\mu$ such that $\beta(\mu(\psi_g))$ is satisfiable, and $U$ the variables in $\psi_g$ undefined by $\beta$. Let $\tau = \text{Deadline}(\psi_g, \text{var}(\varphi_q \cup \psi_y), \mu \cup \beta)$. The following hold:

1. If $\tau \in \mathbb{N}$, then there is a complete head assignment $\beta'$ extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leq \tau$ and $\beta'(\psi_g)$ is true,
2. If $\tau \in \mathbb{N}$, then for all complete head assignments $\beta'$ 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 $\beta'$ 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 $\psi$, either for some time variable $z$ and timestamp $\tau$, $\mu(\beta(\psi)) \land (z \geq \tau')$ is unsatisfiable (so $\tau$ is a deadline) or no such time variable $z$ and timestamp $\tau$ exists (there is no deadline). Lemma 3 is applied in the following way: for a complete body assignment $\mu$, we try to extend $\mu$ with each partial head assignment $\beta$ when it is created. For each pair $\mu$ and $\beta$, we calculate a deadline using $\mu$, $\beta$, and the rule head. According to Lemma 3, the output of $\text{Deadline}$ is the time after which $\beta$ cannot extend $\mu$. 

---

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

**Input:** A set of gap atoms $\Theta$ over time variables $x_1, \ldots, x_n$ and an assignment $\mu$

**Output:** A timestamp $\tau$

1. if $\mu(\Theta)$ is unsatisfiable then return $\tau := \max(\mu)$;
   // * max$(\mu)$ is the largest timestamp $\mu$ assigns to $x_1, \ldots, x_n*$
2. Rewrite each atom in $\mu(\Theta)$ in the form $u \pm k \leq v$;
   // * $u, v$ either a time variable or in $\mathbb{N}$, $k \in \mathbb{Z}*$
3. Let $\text{UpperBd}$ be a map from $x_1, \ldots, x_n$ to $\{\infty\}$;
4. for each $u \pm k \leq v$ in $\mu(\Theta)$ with $v \in \mathbb{N}$ and $u \in \{x_1, \ldots, x_n\}$ do
   5. $\text{UpperBd}(u) := v \mp k$
6. for $|\Theta|$ iterations do
   7. for each gap atom $u \pm k \leq v$ in $\mu(\Theta)$ do
   8. if $\text{UpperBd}(v)$ is finite and $\text{UpperBd}(u) \pm k > \text{UpperBd}(v) \geq 0$ then
   9. $\text{UpperBd}(u) := \text{UpperBd}(v) \mp k$
10. return $\tau := \min\{\text{UpperBd}(x_i) | 1 \leq i \leq n\}$

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

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

The $\text{Deadline}$ 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 $\mu$ and a partial head assignment $\beta$, we compute the latest time $\mu \cup \beta$ can be extended. This time is, in fact, the earliest time $\mu$ becomes a permanent violation. In the following lemma, we state a property of deadlines for a complete body assignment and partial head assignment.

---

I. Mackey, R. Chimni, and J. Su
The discussions in Section 2 also suggest maintaining partial and complete assignments for variables. We define three tables \( \mathcal{BA}_r \) for body assignments, \( \mathcal{HA}_r \) for head assignments, and \( \mathcal{EXT}_r \) (extensions) to track pairings of body and head assignments. \( \mathcal{BA}_r \) and \( \mathcal{HA}_r \) consist of the following columns: (i) one column for the assignment identifier (Aid) from \( \mathcal{I} \), (ii) one column for the enactment identifier (Id) from \( \mathcal{I} \), (iii) one column in \( \mathcal{BA}_r \) for each variable in \( \varphi \) and one column in \( \mathcal{HA}_r \) for each event variable in \( \psi \) (resp.) (a variable in the head \( \psi \) is an event variable if it occurs in an event atom in \( \psi \)) to hold a value from \( \mathcal{D} \) or \( \mathcal{T} \), and (iv) one column for gap atoms in \( \varphi \) and \( \psi \) (resp.) simplified with the assigned values as possible. Additionally, \( \mathcal{BA}_r \) 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. \( \mathcal{EXT}_r \) has three columns: (i) one column for a body Aid from \( \mathcal{BA}_r \), (ii) one column for a head Aid from \( \mathcal{HA}_r \) 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 \( \eta \), \( \mathcal{BA}_r(\eta) \) and \( \mathcal{HA}_r(\eta) \) store all assignments that can be generated from \( \eta \) and satisfy \( \varphi \) and \( \psi \) (resp.). Specifically, for a rule \( r: \varphi \rightarrow \psi \) and an enactment \( \eta \), \( \mathcal{BA}_r(\eta) \) contains every assignment \( \mu \) such that for a non-empty subset \( P \) of the event atoms in \( \varphi \), \( \mu \) is defined for all variables in \( P, \mu(P) \subseteq \eta \), and \( \eta \) satisfies all atoms in \( \varphi \) having only variables in \( P \) with \( \mu \). \( \mathcal{HA}_r(\eta) \) is similar, using \( \psi \) instead of \( \varphi \). Fig. 5(a) shows the assignments inserted into \( \mathcal{BA}_r \) table at time 10 (those from Fig. 2(b)) with columns for gap atoms and the possibility of matching. \( \mathcal{EXT}_r(\eta) \) stores each pair of assignments from \( \mathcal{BA}_r(\eta) \) and \( \mathcal{HA}_r(\eta) \), resp., such that the body assignment can be extended by the head assignment only at or before the row’s deadline.

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
\text{Aid} & u & a & x & y & z & \text{match}\? \\
\hline
\mu_{10} & \text{Alice} & a3 & 4 & 8 & - & \text{No} \\
\hline
\mu_{11} & \text{Alice} & a3 & - & 9 & \{x \leq y \leq x+7, y \leq 9 \leq y+7\} & \text{No} \\
\hline
\mu_{12} & \text{Alice} & a3 & 3 & - & 9 & \{1 \leq y \leq 8, y \leq 9 \leq y+7\} & \text{No} \\
\hline
\mu_{13} & \text{Alice} & a3 & - & 6 & 9 & \{x \leq 6 \leq x+7\} & \text{No} \\
\hline
\mu_{14} & \text{Alice} & a3 & 1 & 6 & 9 & - & \text{No} \\
\hline
\end{array}
\]

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

\( \Box \) **Figure 5** Body and Head Table Examples.

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Aid} & u & a & x \\
\hline
\beta_1 & \text{Alice} & a3 & 8 & - & \{v \leq 12\} \\
\beta_2 & \text{Alice} & a4 & 9 & - & \{v \leq 13\} \\
\beta_3 & \text{Alice} & a3 & - & 12 & \{8 \leq w\} \\
\beta_4 & \text{Alice} & a3 & 8 & 10 & - \\
\hline
\end{array}
\]

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

We next present an algorithm called **Update** to create and combine assignments as batches of events arrive. This algorithm maintains \( \mathcal{BA} \) and \( \mathcal{HA} \) 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 \( \Theta \) (here, the head of a rule), a batch \( \Delta \), and either \( \mathcal{BA}_r \) or \( \mathcal{HA}_r \) for an enactment \( \eta \), Lines 2-6 generate assignments from the events in \( \Delta \) and \( \Theta \), adding them to the table if they are satisfiable (extendible to
Algorithm 2 Update(\(\Theta, \Delta, T(\eta)\)).

Input: A set of atoms \(\Theta\), a batch \(\Delta\), a table \(T(\eta)\) (\(T\) is BA\(_{r}\) or HA\(_{r}\) for enactment \(\eta\))
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 \(\gamma\) in \(\Theta\) with the same activity as \(e\) do
4:     Create a (partial) assignment \(\mu\) from \(e, \gamma\) such that \(\mu(\gamma) = e\);
5:     if \(\mu(\Theta)\) is satisfiable then
6:       Add to \(\Gamma\) the row \(s = \langle a, e, \text{id}, \mu(v_1), ..., \mu(v_n), b, (\text{no})\rangle\),
       where \(a\) is a fresh assignment identifier, \(v_1, ..., v_n\) are the event variables
       in \(\Theta\), and \(b\) is the gap atoms in \(\Theta\), evaluated and simplified under \(\mu\);
7: while \(\Gamma\) changes do
8:   for each pair of unique and consistent rows \(\mu_1\) and \(\mu_2\) in \(T\) do
9:     \(\mu := \text{MERGE}(\mu_1, \mu_2)\); /* consistent, MERGE explained in the text */
10:  Add to \(\Gamma\) the row: \(s = \langle a, \text{id}, \max(t_1, t_2), \mu(v_1), ..., \mu(v_n), b, (\text{no})\rangle\)
     where \(a\) is a fresh assignment identifier and \(b\) is the union of gap atoms in \(\mu_1, \mu_2\), evaluated with \(\mu\);
11: output \(\Gamma\)

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 \(\mu_1\) and \(\mu_2\) agree on \(a\) but not on \(c\). 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 \(\mu_1\) in Fig. 2 is the merge of \(\mu_2\) and \(\mu_3\). The loop only creates assignments whose data values are pre-existing in \(\Gamma\) or the batch \(\Delta\), 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 \(\text{Request}(\pi, [\text{Alice, a3}], 1)\) and the rule’s atom \(\text{Request}(\text{user u, account a})\). The mapping \([\text{id} \rightarrow \pi, \text{u} \rightarrow \text{Alice, v} \rightarrow \text{a3, x} \rightarrow 1]\) maps the atom to this event; the assignment corresponding to this mapping is added to BA\(_{r}\) as \(\mu_1\) in Fig. 2(a). For the same example in Section 2 and Fig. 2(a), assignments \(\mu_2, \mu_3\); \([\pi, \text{Alice, a4, 3, -}, \{3 \leq y \leq 10, y \leq z \leq y + 7\}]\) and \([\pi, \text{Alice, -}, 6, -; \{x \leq 6 \leq x + 7, 6 \leq z \leq 13\}]\) are in BA\(_{r}\)(\(\pi_1\)) at \(ts = 9\) and agree on \(u\). Their combination \(\text{MERGE}(\mu_2, \mu_3)\) satisfies \(x \leq 6 \leq x + 7\) and \(3 \leq y \leq 10\), so a row corresponding to \(\text{MERGE}(\mu_2, \mu_3)\) is added to BA\(_{r}\) as \(\mu_3\).

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

Lemma 5. Let \(\varphi; \phi \rightarrow \psi\) be a rule, \(\eta\) an enactment, and \(\Delta\) a batch for \(\eta\). \(\text{Update}(\varphi, \Delta, \text{BA}_{r}(\eta))\) (or \(\text{Update}(\psi, \Delta, \text{HA}_{r}(\eta))\)) 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 \(\eta\) so they will be in \(\text{BA}_{r}(\eta)\) and some may come from events in \(\Delta\), 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 \text{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 \text{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.
4:10 Early Detection of Temporal Constraint Violations

**Algorithm 3 Update-E(Δ, EXTr(η), BAr(η∪Δ), HAr(η∪Δ)).**

**Input:** A batch Δ, un-updated table EXTr(η),
updated tables BAr(η∪Δ) and HAr(η∪Δ) for an enactment η

**Output:** Updated table EXTr(η∪Δ)

1: \( Γ := EXTr(η) \);
2: for each complete body assignment \( μ \) in BAr(η∪Δ) do
3: if \( \text{max}(μ) = ts_Δ \) then Add \( ⟨μ, r, Deadline(τ, \var(ψ, μ))⟩ \) to \( Γ \);
4: for each assignment \( γ \) in HAr(η∪Δ) do
5: if \( \text{max}(γ) = ts_Δ \) then
6: for each row \( ⟨μ, β, d⟩ \) in \( Γ \) do
7: if \( γ \) extends \( μ ∪ β \) and \( γ(μ(ψ)) \) is satisfiable then
8: Add \( ⟨μ, γ, Deadline(υ, \var(ψ, μ ∪ γ))⟩ \) to \( Γ \);
9: if \( γ \) is complete then Update BAr(η∪Δ) to indicate \( μ \) has a match;
10: output \( Γ \);
/* = EXTr(η∪Δ) */

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 \( EXTr(η) \) but no rows in \( EXTr(η) \) pairs \( μ \) with a complete head assignment, and each row in \( EXTr(η) \) 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 \( EXTr(η) \) 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 \( EXTr(η≤9) \) is 9. At time 9, a Payment event creates \( β_2 \) (Fig. 5), and \( μ_{10} \) and \( β_2 \) are inserted into \( EXTr(η≤9) \) with deadline 13 because \( β_2(w) = 9 \) and \( ψ \) contains \( v ≤ w + 4 \). 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(Δ, EXTr(η∪Δ)).**

**Input:** A batch Δ, updated EXT(η∪Δ)

**Output:** A set of assignments indicating rule violations

1: \( \text{Violations} := \{\} \);
2: for each complete body assignment \( μ \) in EXT(η∪Δ) do
3: if \( μ \) is not extended by any complete head assignment then
4: if \( Δ \) contains an END event \( e \) with \( e.ID = μ.ID \) then
5: Add \( μ \) to \( \text{Violations} \);
6: Let \( τ \) be the maximum deadline for the rows in EXT(η∪Δ) with \( μ \);
7: if \( ts_Δ ≥ τ > max(η) \) then
8: Add \( μ \) to \( \text{Violations} \);
9: output \( \text{Violations} \);

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, \( \eta \) an enactment, and \( \Delta \) a batch for \( \eta \). Then, \( \mu \) is a violation in \( \eta \cup \Delta \) but not in \( \eta \) iff \( \text{Detect}(\Delta, \text{EXT}_{\vartriangle}(\eta \cup \Delta)) \) reports \( \mu \).

A sketch of the proof is given in Appendix A. The key idea is that for a given body assignment \( \mu \) in \( \text{EXT}_{\vartriangle}(\eta \cup \Delta) \), by Lemma 6, if \( \mu \) is a violation, it will be in exclusively unmatched rows in \( \text{EXT}_{\vartriangle}(\eta \cup \Delta) \) with a deadline of at most \( \text{ts} \) of \( \Delta \). Then, when \( \Delta \) is processed, \( \mu \) 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 \( |L| \) of the log, 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 \( \text{BAE} \) and \( \text{HAE} \), requiring that incomplete assignments in \( \text{BAE} \) and \( \text{HAE} \) 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 \( \text{BAE} \) and \( \text{HAE} \) 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 \( O(|\Delta|^c) \). 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 \( O(|\Delta|^c) \).

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 \( \Delta \), we only need to try pairs that have at least one assignment added from events in \( \Delta \), because all other pairs of assignments were considered before \( \Delta \) arrived. To make Update to reflect this, we use a queue \( \Gamma_{\text{new}} \) 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 \( \mu_n \) in \( \Gamma_{\text{new}} \) (outer loop) and each row \( \mu_o \) in \( \Gamma \) (inner loop), adding the new assignment to the queue \( \Gamma_{\text{new}} \) moving \( \mu_n \) from \( \Gamma_{\text{new}} \) to \( \Gamma \) after processing \( \mu_n \). 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: \( O(|L|^e|\Delta|^c) \) 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.

<table>
<thead>
<tr>
<th>rules</th>
<th>normal-length enactments</th>
<th>large enactments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% events before first violation</td>
<td>% after</td>
</tr>
<tr>
<td>simple</td>
<td>74.9</td>
<td>25.1</td>
</tr>
<tr>
<td>complex</td>
<td>83.7</td>
<td>16.3</td>
</tr>
</tbody>
</table>

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

Finding 2. The overhead of detecting violations early is \(\leq 15\%\) 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 ($\leq 2\%$) is less than the increase with EVD for large enactments ($\leq 15\%$). 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.

<table>
<thead>
<tr>
<th>rules</th>
<th>without EVD</th>
<th>with EVD</th>
<th>without EVD</th>
<th>with EVD</th>
</tr>
</thead>
<tbody>
<tr>
<td>simple</td>
<td>4.27$\times 10^{-2}$</td>
<td>4.73$\times 10^{-2}$ (+0.2%)</td>
<td>6.65$\times 10^{-2}$</td>
<td>7.60$\times 10^{-2}$ (+14.3%)</td>
</tr>
<tr>
<td>complex</td>
<td>9.19$\times 10^{-2}$</td>
<td>9.31$\times 10^{-2}$ (+1.3%)</td>
<td>1.840$\times 10^{-1}$</td>
<td>2.084$\times 10^{-1}$ (+13.3%)</td>
</tr>
</tbody>
</table>

**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 $\leq 1$ 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 $\leq 1$ 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.

<table>
<thead>
<tr>
<th>enactment length</th>
<th>normal</th>
<th>large</th>
<th>normal</th>
<th>large</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch size</td>
<td>simple rules</td>
<td>complex rules</td>
<td>simple rules</td>
<td>complex rules</td>
</tr>
<tr>
<td>100</td>
<td>4.55$\times 10^{-3}$</td>
<td>6.19$\times 10^{-3}$</td>
<td>7.74$\times 10^{-3}$</td>
<td>1.363$\times 10^{-3}$</td>
</tr>
<tr>
<td>1,000</td>
<td>4.330$\times 10^{-3}$</td>
<td>6.177$\times 10^{-3}$</td>
<td>7.534$\times 10^{-3}$</td>
<td>1.3509$\times 10^{-2}$</td>
</tr>
<tr>
<td>10,000</td>
<td>4.2414$\times 10^{-2}$</td>
<td>6.0769$\times 10^{-2}$</td>
<td>7.4925$\times 10^{-2}$</td>
<td>1.35218$\times 10^{-1}$</td>
</tr>
</tbody>
</table>

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

Then, Proof Sketch for Lemma 3.

Let $\tau$ be a rule, $\phi \rightarrow \psi$ be a rule, $\phi, \psi \in \text{gap atoms in } \tau$ (resp.), $\mu$ a complete body assignment such that $\mu(\phi)$ is true, $\beta$ an incomplete head assignment extending $\mu$ such that $\beta(\mu(\psi))$ is satisfiable, and $\psi$ the variables in $\psi$ undefined by $\beta$. Let $\tau = \text{Deadline}(\psi, \text{var}(\phi \cup \psi), \mu \cup \beta)$. The following hold:

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

Proof Sketch for Lemma 3. To show (1), assume there is no complete head assignment $\beta'$ extending $\mu \cup \beta$ such that $\min(\beta'(U)) \leq \tau$ and $\beta'(\psi)$ is true. Then, $(\mu \cup \beta)(\psi) \land (z = \tau)$ is not satisfiable. Then, there is a gap atom in $\mu \cup \beta(\psi)$ that provides an upper bound for $z$ below $\tau$. Then, $\tau$ is not the minimum of the upper bounds in $\text{UpperBd}$. Thus Algorithm 1 on $\mu \cup \beta$ and $\psi$ should not output $\tau$. This is a contradiction. To show (2), assume some complete head assignment $\beta'$ extends $\mu \cup \beta$ such that $\min(\beta'(U)) > \tau$ and $\beta'(\psi)$ is true. Then, $(\mu \cup \beta)(\psi) \land (z = \tau')$ is satisfiable for some $z$ in $\text{var}(\psi)$. Then, $\mu(\psi)$ does not imply $z_i \leq \tau$ for all variables $z_i$. Thus Algorithm 1 on $\mu \cup \beta$ and $\psi$ should not output $\tau$. This
Early Detection of Temporal Constraint Violations

is a contradiction. To show (3), assume \( \tau = \infty \). Algorithm 1 only produces \( \infty \) when \( \mu(\psi) \) is satisfiable for some variable \( z_i \) and for all \( n \in \mathbb{N} \), \( \mu(\psi) \not\rightarrow (z_i \leq i) \) then, for all timestamps \( n \in \mathbb{N} \), there is some complete assignment that extends \( \mu \), satisfies \( \psi \), and uses some \( n' \) larger than \( n \). Then, \( \mu \) can be extended arbitrary far in the future.

\[ ▶ \text{Lemma 5. Let } r; \varphi \rightarrow \psi \text{ be a rule, } \eta \text{ an enactment, and } \Delta \text{ a batch for } \eta. \] \[ \text{Update}(\varphi, \Delta, \text{BA}_{r}(\eta)) \text{ (or } \text{Update}(\psi, \Delta, \text{HA}_{r}(\eta)) \text{) computes } \text{BA}_{r}(\eta \cup \Delta) \text{ (resp. } \text{HA}_{r}(\eta \cup \Delta)). \]

\[ \text{Proof Sketch for Lemma 5. We argue this for } \text{BA}_{r}(\eta); \text{ adapting this argument for } \text{HA}_{r}(\eta) \text{ is trivial. For an assignment } \mu \text{ in } \text{BA}_{r}(\eta \cup \Delta) \text{ created by } \text{Update}(\varphi, \Delta, \text{BA}_{r}(\eta)), \text{ some events } C \text{ in } \eta \text{ and } D \text{ in } \Delta \text{ provide values for } \mu. \text{ Then an assignment } \mu_{C} \text{ for } C \text{ is present in } \text{BA}_{r}(\eta) \text{ and Lines 2–5 of Alg. 2 generates } |D| \text{ assignments for each event in } D. \text{ Next, these } |D| + 1 \text{ assignments will merge with each other in the loop of Line 7 of Alg. 2 until } \mu \text{ is created and added to } \Gamma. \text{ Alternatively, consider any assignment } \mu \text{ that is not in } \text{BA}_{r}(\eta \cup \Delta) \text{ after Algorithm 2. Then, no subset of events in } \eta \cup \Delta \text{ can create } \mu \text{ on Line 4 or } \mu \text{ is inconsistent with the rule body or head and will not proceed past Lines 5 or 8 of Alg. 2.} \]

\[ ▶ \text{Lemma 6. Let } \eta \text{ be an enactment with no END event, } r \text{ a rule, } \tau \text{ a timestamp, and } \mu \text{ a complete body assignment for } r. \text{ Then, } \mu \text{ is a violation of } r \text{ in } \eta \text{ iff } \mu \text{ occurs in } \text{EXT}_{r}(\eta) \text{ but no rows in } \text{EXT}_{r}(\eta) \text{ pairs } \mu \text{ with a complete head assignment, and each row in } \text{EXT}_{r}(\eta) \text{ with } \mu \text{ has a deadline no greater than } \tau. \]

\[ \text{Proof Sketch for Lemma 6. Let } \tau \text{ be the largest timestamp in } \eta. \text{ EXT}_{r}(\eta) \text{ contains all possible pairs for } \mu \text{ and head assignments from } \text{HA}_{r}(\eta), \text{ so if } \mu \text{ is unmatched in } \text{BA}_{r}(\eta), \text{ there is no assignment with } \min(\beta) \leq \tau \text{ that extends } \mu \text{ and satisfies } \psi. \text{ Alternatively, let } \tau \text{ be the largest deadline for } \mu \text{ in } \text{EXT}_{r}(\eta), \text{ by Lemma 3, for all rows with } \mu \text{ and } \beta \text{ in } \text{EXT}_{r}(\eta), \text{ for all complete head assignments } \beta' \text{ that extend } \mu \cup \beta, \text{ such that } \min(\beta'(U)) > \tau, \text{ is inconsistent. Thus, no future (i.e., with a value greater than } \tau \text{) complete head assignment can extend } \mu \text{ and satisfy } \psi. \text{ Then, } \mu \text{ will never be extended by a complete head assignment that satisfies } \psi, \text{ so } \mu \text{ is a violation for } \eta. \]

\[ ▶ \text{Theorem 8. Let } r \text{ be a rule, } \eta \text{ be an enactment, and } \Delta \text{ a batch for } \eta. \text{ Then, } \mu \text{ is a violation in } \eta \cup \Delta \text{ but not in } \eta \text{ iff } \text{Detect}(\Delta, \text{EXT}_{r}(\eta \cup \Delta)) \text{ reports } \mu. \]

\[ \text{Proof Sketch for Theorem 8. Let } \mu \text{ be a violation in } \eta \cup \Delta. \text{ } \eta \cup \Delta \text{ may contain an END event and will have no later events, in which case, } \eta.\text{END} \text{ is in } \Delta \text{ and } \mu \text{ will be added to Violations on Line 5 of Algorithm 4. Otherwise, by Lemma 6, } \mu \text{ is complete and in exclusively unmatched rows in } \text{EXT}_{r}(\eta \cup \Delta) \text{ with a deadline of, at most, } ts_{\Delta}. \text{ Then, } \mu \text{ will be added to Violations on Line 8 of Algorithm 4.} \]

\[ \text{Conversely, if } \text{Detect}(\Delta, \text{EXT}_{r}(\eta \cup \Delta)) \text{ reports } \mu, \text{ then } \mu \text{ is added to Violations on Line 5 or Line 8 of Algorithm 4. Given Line 2 of the algorithm, } \mu \text{ must be a complete assignment in } \text{EXT}_{r}(\eta \cup \Delta) \text{ that is not extended by any complete head assignment. Then, either (1) } \eta.\text{END} \text{ in } \Delta \text{ or (2) } ts_{\Delta} \text{ is greater than or equal to the deadline for } \mu \text{ in all rows in } \text{EXT}_{r}(\eta \cup \Delta). \text{ If (1), then } \mu \text{ is a violation because } \eta \cup \Delta \text{ will have no later events. If (2), } \mu \text{ is a violation in } \eta \cup \Delta \text{ by Lemma 6.} \]