Incremental Maintenance of DatalogMTL Materialisations

Kaiyue Zhao[1*], Dingqi Chen[1*], Shaoyu Wang[1, 2*], Pan Hu[1†]

[1]School of Computer Science, Shanghai Jiao Tong University, China
[2]Department of Computer Science, University of Oxford, UK

Abstract

DatalogMTL extends the classical Datalog language with metric temporal logic (MTL), enabling expressive reasoning over temporal data. While existing reasoning approaches, such as materialisation-based and automata-based methods, offer soundness and completeness, they lack support for handling efficient dynamic updates—a crucial requirement for real-world applications that involve frequent data updates. In this work, we propose DRed_{MTL}, an incremental reasoning algorithm for DatalogMTL with bounded intervals. Our algorithm builds upon the classical Delete/Rederive (DRed) algorithm, which incrementally updates the materialisation of a Datalog program. Unlike a Datalog materialisation which is in essence a finite set of facts, a DatalogMTL materialisation has to be represented as a finite set of facts plus periodic intervals indicating how the full materialisation can be constructed through unfolding. To cope with this, our algorithm is equipped with specifically designed operators to efficiently handle such periodic representations of DatalogMTL materialisations. We have implemented this approach and tested it on several publicly available datasets. Experimental results show that DRed_{MTL} often significantly outperforms rematerialisation, sometimes by orders of magnitude.

Code, datasets and instructions — github.com/Horizon12275/DREDmtl-for-DatalogMTL

Extended version with full proof — arxiv.org/abs/2511.12169

Introduction

DatalogMTL extends the well-known rule language Datalog (Ceri, Gottlob, and Tanca 1989) with metric temporal logic (MTL) (Koymans 1990). It has found applications in various domains, including ontology-based query answering (Brandt et al. 2018; Güzel Kalayci et al. 2018), stream reasoning (Wałęga et al. 2023; Wałęga, Kaminski, and Cuenca Grau 2019), and temporal reasoning in the financial sector (Colombo et al. 2023; Nissl and Sallinger 2022; Mori et al. 2022), among others.

To showcase the capabilities of DatalogMTL, consider an industrial application scenario in which DatalogMTL has been applied by our research group to automatically detect anomaly of power transformers. More concretely, gas concentration data is first collected in real time using existing gas-in-oil sensors and then fed into a DatalogMTL rule engine, which fires alarms whenever appropriate. As an example, the following rule shows how oil thermal faults can be detected:

$$\begin{aligned} OTF(x) \leftarrow & HasEthylene(x,y) \\ & \wedge HasEthane(x,z) \\ & \wedge \diamondminus_{[0,10]} AboveThirty(y) \\ & \wedge \diamondminus_{[0,10]} AboveSeventy(z) \end{aligned} \tag{1}$$

This rule states that at any time point t, if ethylene and ethane have both been detected in the transformer oil, and their gas concentration values surpassed 30 ppm and 70 ppm at any time point in the past ten minutes, respectively, then an oil thermal fault (OTF) is detected.

Reasoning in DatalogMTL can be implemented using top-down or bottom-up approaches, or a combination of the two. One typical top-down approach is based on Büchi automata: it ensures correctness but incurs high reasoning costs. Therefore, efforts have been made to devise efficient bottom-up (or materialisation-based) approach for DatalogMTL reasoning. The vadalog system (Bellomarini, Nissl, and Sallinger 2022) implements such an approach, but it may not terminate due to recursion. The MeTeoR system (Wang et al. 2022) combines bottom-up and top-down approaches, but only resorts to the top-down approach when necessary. More recently, magic set rewriting, which simulates top-down evaluation via bottom-up reasoning, has been extended to support DatalogMTL reasoning (Wang et al. 2025). While materialisation-based approaches are popular for DatalogMTL reasoning, to the best of our knowledge, no incremental maintenance algorithm for DatalogMTL reasoning has been developed: when the set of explict facts change, the above systems have no choice but to recompute the materialisation from scratch.

For plain Datalog, many incremental materialisation maintenance algorithms have been developed, including Delete/Rederive (DRed) and its variants (Gupta, Mumick, and Subrahmanian 1993; Staudt and Jarke 1996; Ren and Pan 2011; Urbani et al. 2013; Hu, Motik, and Horrocks 2018), Backward/Forward (B/F) (Motik et al. 2015),

[*]These authors contributed equally.
[†]Corresponding author

FBF (Motik et al. 2019), among others. However, adapting an incremental materialisation maintenance algorithm for Datalog to support DatalogMTL reasoning is nontrivial. Unlike Datalog, DatalogMTL supports recursion over time, which easily leads to unbounded time intervals, making termination problematic. As a result, changes to the dataset may not only affect immediate derivations but also propagate across time through periodic patterns. Although it is known that under certain restrictions, materialisations in DatalogMTL exhibit repeating structures that can be finitely represented, how to correctly update such repeating structures is still highly challenging, especially due to the complex interplay between consequence propagation and termination condition checks.

In this paper, building upon the well-known DRed algorithm, we propose DRed_{MTL}. It replaces standard Datalog materialisation maintenance with a novel set of operations over finite representations of DatalogMTL materialisations called *periodic materialisations*. Periodic materialisations compactly encode infinite sets of temporal facts by capturing their recurring periodic patterns, which allows us to reason over infinite time domains using finite representations.

To facilitate reasoning over periodic materialisations, we devised a novel seminaïve evaluation operator specifically designed to efficiently handle the propagation of facts in the new DatalogMTL setting. In addition, we developed a new period identification algorithm that effectively and correctly guarantees termination. Our experimental evaluation demonstrates that, compared to rematerialisation from scratch, our approach achieves significant performance improvements, especially for small updates.

Preliminaries

In this section, we first briefly recapitulate the syntax and semantics of DatalogMTL. We focus on DatalogMTL with bounded intervals, as this restriction enables materialisation-based reasoning. Then, we briefly discuss how the materialisation of a DatalogMTL program can be finitely represented: our incremental maintenance algorithm will have to incrementally update such finite representations in response to changes in the explicitly given data.

Syntax of DatalogMTL Throughout the paper, we assume that the timeline consists of rational numbers. A time interval is a set of continuous time points ϱ of the form $\langle t_1, t_2 \rangle$, where $t_1 \in \mathbb{Q} \cup \{-\infty\}$, $t_2 \in \mathbb{Q} \cup \{\infty\}$, $\langle$ is $[$ or $($ and likewise $\rangle$ is $]$ or $)$. An interval is bounded if both of its endpoints are rational numbers, i.e., neither ∞ nor $-\infty$. When it is clear from the context, we may abuse the distinction between intervals (i.e. sets of time points) and their representation $\langle t_1, t_2 \rangle$. If $\varrho = \langle t_1, t_2 \rangle$, let $\varrho^- = t_1$ and $\varrho^+ = t_2$.

A term is either a variable or a constant. A relational atom is an expression $R(\boldsymbol{t})$, where R is a predicate and $\boldsymbol{t}$ is a tuple of terms of arity matching that of R. Metric atoms extend relational atoms by allowing operators from metric temporal logic (MTL), namely $\boxplus_\varrho$, $\boxminus_\varrho$, $\diamondplus_\varrho$, $\diamondminus_\varrho$, $\mathcal{U}_\varrho$, and $\mathcal{S}_\varrho$, where ϱ is an interval. Formally, metric atoms, M, are generated by the grammar

$$M ::= \bot \mid \top \mid R(\boldsymbol{t}) \mid \boxplus_\varrho M \mid \boxminus_\varrho M \mid \diamondplus_\varrho M \mid \diamondminus_\varrho M \mid M \mathcal{U}_\varrho M \mid M \mathcal{S}_\varrho M,$$

where $\top$ and $\bot$ are the constants representing truth and falsehood, respectively, and ϱ is any arbitrary interval containing only nonnegative rationals. A DatalogMTL rule, r, is of the form

$$M' \leftarrow M_1 \wedge M_2 \wedge \cdots \wedge M_n, \quad \text{for } n \geq 1,$$

where each M_i is a metric atom and M' is a metric atom not mentioning $\bot$, $\diamondplus$, $\diamondminus$, $\mathcal{U}$, and $\mathcal{S}$. Metric atom M' and the set $\{M_i \mid i \in \{1, \ldots, n\}\}$ are the head and body of r, denoted as $head(r)$ and $body(r)$, respectively.

A rule r is safe if each variable in its head appears also in its body. A program Π is a finite set of safe rules. A substitution σ is a mapping of finitely many variables to constants. For α an expression (e.g., an atom, a rule, or a program thereof), $\alpha\sigma$ is the result of replacing each occurrence of a variable x in α with $\sigma(x)$, if the latter is defined. An expression is ground if it mentions no variable. A fact is of the form $R(\boldsymbol{t})@\varrho$, where $R(\boldsymbol{t})$ is a ground relational atom, ϱ is an interval, and $@$ indicates that the preceding atom holds over the time interval that follows. A dataset $\mathcal{D}$ is a finite set of facts. If all the intervals a dataset (resp., a program) mentions are bounded, then the dataset (resp., the program) is bounded. Our work focuses on bounded datasets and programs.

$\mathfrak{I}, t \vDash \top$		for every $t \in \mathbb{Q}$
$\mathfrak{I}, t \vDash \bot$		for no $t \in \mathbb{Q}$
$\mathfrak{I}, t \vDash \boxplus_\varrho M$	iff	$\mathfrak{I}, t_1 \vDash M$ for all t_1 s.t. $t_1 - t \in \varrho$
$\mathfrak{I}, t \vDash \boxminus_\varrho M$	iff	$\mathfrak{I}, t_1 \vDash M$ for all t_1 s.t. $t - t_1 \in \varrho$
$\mathfrak{I}, t \vDash \diamondplus_\varrho M$	iff	$\mathfrak{I}, t_1 \vDash M$ for some t_1 s.t. $t_1 - t \in \varrho$
$\mathfrak{I}, t \vDash \diamondminus_\varrho M$	iff	$\mathfrak{I}, t_1 \vDash M$ for some t_1 s.t. $t - t_1 \in \varrho$
$\mathfrak{I}, t \vDash M_2 \mathcal{U}_\varrho M_1$	iff	$\mathfrak{I}, t_1 \vDash M_1$ for some t_1 s.t. $t_1 - t \in \varrho$, and $\mathfrak{I}, t_2 \vDash M_2$ for all $t_2 \in (t, t_1)$
$\mathfrak{I}, t \vDash M_2 \mathcal{S}_\varrho M_1$	iff	$\mathfrak{I}, t_1 \vDash M_1$ for some t_1 s.t. $t - t_1 \in \varrho$, and $\mathfrak{I}, t_2 \vDash M_2$ for all $t_2 \in (t_1, t)$

Table 1: Semantics for ground metric atoms

Semantics of DatalogMTL A DatalogMTL interpretation $\mathfrak{I}$ is a function that maps each time point $t \in \mathbb{Q}$ to a set of ground relational atoms (essentially to atoms that hold at t). For a time point $t \in \mathbb{Q}$, if $R(\boldsymbol{t})$ belongs to this set, we write $\mathfrak{I}, t \vDash R(\boldsymbol{t})$. This extends to the ground metric atoms as presented in Table 1.

An interpretation $\mathfrak{I}$ satisfies a fact $R(\boldsymbol{t})@\varrho$, denoted as $\mathfrak{I} \vDash R(\boldsymbol{t})@\varrho$, if $\mathfrak{I}, t \vDash R(\boldsymbol{t})$ for each $t \in \varrho$. An interpretation $\mathfrak{I}$ is a model of a dataset $\mathcal{D}$, written $\mathfrak{I} \vDash \mathcal{D}$, if it satisfies every fact in $\mathcal{D}$. An interpretation $\mathfrak{I}$ satisfies a ground rule r_0 if $\mathfrak{I}, t \vDash body(r_0)$ implies $\mathfrak{I}, t \vDash head(r_0)$ for each $t \in \mathbb{Q}$. A ground rule r_0 is an instance of a rule r if there

is a substitution σ such that $r_0 = r\sigma$. An interpretation $\mathfrak{I}$ satisfies a rule r, if it satisfies all ground instances of r, and it is a model of a program Π, if it satisfies all rules in Π. $\mathfrak{I}$ is a model of a pair $(\Pi, \mathcal{D})$ if $\mathfrak{I}$ is both a model of $\mathcal{D}$ and a model of Π . A program-dataset pair $(\Pi, \mathcal{D})$ entails a fact $R(\boldsymbol{t})@\varrho$, written $(\Pi, \mathcal{D}) \models R(\boldsymbol{t})@\varrho$, if all models of $(\Pi, \mathcal{D})$ satisfy $R(\boldsymbol{t})@\varrho$. An interpretation $\mathfrak{I}$ contains another interpretation $\mathfrak{I}'$, written $\mathfrak{I}' \subseteq \mathfrak{I}$, if $\mathfrak{I}, t \models R(\boldsymbol{t})$ implies $\mathfrak{I}', t \models R(\boldsymbol{t})$ for each ground relational atom $R(\boldsymbol{t})$ and each $t \in \mathbb{Q}$; moreover, $\mathfrak{I} = \mathfrak{I}'$ if they contain each other. An interpretation $\mathfrak{I}$ is the least in a set of interpretations $\boldsymbol{\mathfrak{I}}$, if $\mathfrak{I} \in \boldsymbol{\mathfrak{I}}$, and $\forall \mathfrak{I}' \in \boldsymbol{\mathfrak{I}}$, $\mathfrak{I} \subseteq \mathfrak{I}'$. Similarly, $\mathfrak{I}$ is the greatest in $\boldsymbol{\mathfrak{I}}$, if $\mathfrak{I} \in \boldsymbol{\mathfrak{I}}$, and $\forall \mathfrak{I}' \in \boldsymbol{\mathfrak{I}}$, $\mathfrak{I}' \subseteq \mathfrak{I}$. Each dataset $\mathcal{D}$ has a unique least interpretation $\mathfrak{I}_\mathcal{D}$ such that $\mathfrak{I}_\mathcal{D}$ is the least in the set of all models of $\mathcal{D}$. For interpretations $\mathfrak{I}_1$, $\mathfrak{I}_2$, and $\mathfrak{I}_3$, we write $\mathfrak{I}_3 = \mathfrak{I}_1 \cup \mathfrak{I}_2$ if $\mathfrak{I}_3$ is the least in $\{\mathfrak{I} \mid \mathfrak{I}_1 \subseteq \mathfrak{I} \text{ and } \mathfrak{I}_2 \subseteq \mathfrak{I}\}$, and we write $\mathfrak{I}_3 = \mathfrak{I}_1 \cap \mathfrak{I}_2$, if $\mathfrak{I}_3$ is the greatest in $\{\mathfrak{I} \mid \mathfrak{I} \subseteq \mathfrak{I}_1 \text{ and } \mathfrak{I} \subseteq \mathfrak{I}_2\}$. The empty interpretation $\mathfrak{I}_\varnothing$ maps t to $\varnothing$ for each $t \in \mathbb{Q}$, and it is contained by any interpretation. Finally, we write $\mathfrak{I}_3 = \mathfrak{I}_1 - \mathfrak{I}_2$, if $\mathfrak{I}_3$ is the greatest in $\{\mathfrak{I} \mid \mathfrak{I} \subseteq \mathfrak{I}_1, \mathfrak{I} \cap \mathfrak{I}_2 = \mathfrak{I}_\varnothing\}$.

Materialisation for DatalogMTL We now briefly discuss how materialisation-based reasoning works for DatalogMTL with bounded intervals. To this end, we first introduce a few additional notations that will facilitate our discussion. For $\mathcal{D}$ a dataset, $t^-_\mathcal{D}$ and $t^+_\mathcal{D}$ denote the minimal and maximal interval endpoints appearing in $\mathcal{D}$, respectively, and $t^-_\mathcal{D} = t^+_\mathcal{D} = 0$ if $\mathcal{D}$ mentions no numbers. Moreover, the depth of a rule r, written $\mathsf{depth}(r)$, is defined as the sum of right endpoints of all intervals appearing in the operators of r, and $\mathsf{depth}(r) = 0$ if r mentions no intervals; for Π a program, $\mathsf{depth}(\Pi)$ is the maximum depth of its rules.

The immediate consequence operator T_Π for a program Π is the function that maps an interpretation $\mathfrak{I}$ to the least interpretation $T_\Pi(\mathfrak{I})$ containing $\mathfrak{I}$ and satisfying the following: for each r_0 a ground rule instance of a rule in Π and each t a time point, $\mathfrak{I}, t \models body(r_0)$ implies $\mathfrak{I}, t \models head(r_0)$. For a program Π and a dataset $\mathcal{D}$, a transfinite sequence of interpretations $T^\alpha_\Pi(\mathfrak{I}_\mathcal{D})$ can be defined for ordinals α by successively applying T_Π, the immediate consequence operator for Π, to $\mathfrak{I}_\mathcal{D}$, the unique least model for $\mathcal{D}$, as follows: (i) $T^0_\Pi(\mathfrak{I}_\mathcal{D}) = \mathfrak{I}_\mathcal{D}$, (ii) $T^{\alpha+1}_\Pi(\mathfrak{I}_\mathcal{D}) = T_\Pi(T^\alpha_\Pi(\mathfrak{I}_\mathcal{D}))$ for α an ordinal, and (iii) $T^\alpha_\Pi(\mathfrak{I}_\mathcal{D}) = \bigcup_{\beta<\alpha} T^\beta_\Pi(\mathfrak{I}_\mathcal{D})$ for α a limit ordinal; the canonical model $\mathfrak{C}_{\Pi,\mathcal{D}}$ of Π and $\mathcal{D}$ is the interpretation $T^{\omega_1}_\Pi(\mathfrak{I}_\mathcal{D})$ with ω_1 the first uncountable ordinal. In fact, $\mathfrak{C}_{\Pi,\mathcal{D}}$ is the least model of Π and $\mathcal{D}$ (Brandt et al. 2017).

For an interpretation $\mathfrak{I}$ and an interval ϱ, the projection $\mathfrak{I}\,|_\varrho$ of $\mathfrak{I}$ over ϱ is the interpretation that coincides with $\mathfrak{I}$ on ϱ and maps all relational atoms to false outside ϱ. Moreover, an interpretation $\mathfrak{I}$ is a shift of another interpretation $\mathfrak{I}'$, if there is a rational number s such that for each $R(\boldsymbol{t})$ a ground relational atom and each t a time point, $\mathfrak{I}', t \models R(\boldsymbol{t})$ implies $\mathfrak{I}, t + s \models R(\boldsymbol{t})$, and vice versa. Finally, for a given program Π and a dataset $\mathcal{D}$, we are not interested in dealing with arbitrary rational number time points on the time line; instead, it is sufficient to handle time points that are somewhat related to those appearing in Π and $\mathcal{D}$. This is formalised by the notion of $(\Pi, \mathcal{D})$-ruler. Concretely, $(\Pi, \mathcal{D})$-ruler is the set of time points of the value $t + i \times div(\Pi)$, where t is an endpoint mentioned in $\mathcal{D}$ and i is an integer, and $div(\Pi) = 1/k$, with k being the product of all denominators in the rational endpoints mentioned in Π; for generality, $k = 1$ and $div(\Pi) = 1$ if Π has no mention of rational endpoints.

We are now ready to define the notions of saturated interpretation and unfolding, which are key components for finitely representing the materialisation of a bounded program-dataset pair.

Definition 1. For a program Π and a dataset $\mathcal{D}$, interpretation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is saturated if there exist closed intervals $\varrho_1, \varrho_2, \varrho_3$ and ϱ_4 of length $2\mathsf{depth}(\Pi)$, whose endpoints are located on the $(\Pi, \mathcal{D})$-ruler and satisfy $\varrho^+_1 < \varrho^+_2 < t^-_\mathcal{D}$ and $t^+_\mathcal{D} < \varrho^-_3 < \varrho^-_4$, and such that the following properties hold:

- $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ satisfies Π in $[\varrho^-_1, \varrho^+_4]$;
- $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_1}$ and $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_3}$ are shifts of $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_2}$ and $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_4}$,respectively.

$[\varrho^-_1, \varrho^-_2)$ and $(\varrho^+_3, \varrho^+_4]$ are often referred to as the left period, written ϱ_left, and the right period, written ϱ_right, of the interpretation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$, respectively.

Definition 2. The $(\varrho_\mathsf{left}, \varrho_\mathsf{right})$-unfolding of a saturated interpretation $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ with periods $(\varrho_\mathsf{left}, \varrho_\mathsf{right})$ is the interpretation $\mathfrak{C}$ such that:

- $\mathfrak{C}\,|_{[\varrho^-_\mathsf{left}, \varrho^+_\mathsf{right}]} = T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{[\varrho^-_\mathsf{left}, \varrho^+_\mathsf{right}]}$,
- $\mathfrak{C}\,|_{\varrho_\mathsf{left} - n\cdot|\varrho_\mathsf{left}|}$ is a shift of $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_\mathsf{left}}$, for any $n \in \mathbb{N}$,
- $\mathfrak{C}\,|_{\varrho_\mathsf{right} + n\cdot|\varrho_\mathsf{right}|}$ is a shift of $T^k_\Pi(\mathfrak{I}_\mathcal{D})\,|_{\varrho_\mathsf{right}}$, for any $n \in \mathbb{N}$.

It has been shown that for an arbitrary pair of bounded program and dataset $(\Pi, \mathcal{D})$, there exists $k \in \mathbb{N}$ and intervals ϱ_left and ϱ_right such that $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ is a saturated interpretation with periods $(\varrho_\mathsf{left}, \varrho_\mathsf{right})$, and the $(\varrho_\mathsf{left}, \varrho_\mathsf{right})$-unfolding of $T^k_\Pi(\mathfrak{I}_\mathcal{D})$ coincides with $\mathfrak{C}_{\Pi,\mathcal{D}}$, the canonical model of Π and $\mathcal{D}$ (Wałęga et al. 2023). Intuitively speaking, a saturated interpretation finitely represents the canonical model of a program and a dataset. The goal of this work is to incrementally update a saturated interpretation in response to changes in the explicitly given dataset $\mathcal{D}$.

The Delete/Rederive Algorithm To incrementally update saturated interpretations for DatalogMTL, we draw inspirations from the Delete/Rederive (DRed) algorithm (Gupta, Mumick, and Subrahmanian 1993; Staudt and Jarke 1996), a well-known technique for maintaining Datalog materialisations. Given a Datalog program, a set of explicitly given facts, the original materialisation, and sets of facts to remove from and to add to the given facts, the DRed algorithm updates the materialisation to reflect changes in the explicitly given facts, without recomputing it from scratch. More specifically, the algorithm operates in three stages: in the overdeletion stage, the algorithm eagerly identifies all facts that depend on the set of deleted facts; it then enters the rederivation stage, in which it recognises which of the overdeleted facts can be rederived in one step; finally, during insertion, the algorithm computes the consequences of both the rederived facts and the newly inserted ones.

Motivation

Before we present the technical details of our incrmental update approach for DatalogMTL, in this section, we discuss at a high level challenges that need to be addressed in devising such an incremental update approach. We also provide key insights behind our incremental update algorithm. Finally, we provide a concrete example that highlights the benefits of applying our incremental maintenance algorithm for DatalogMTL reasoning compared with recomputing the materialisation from scratch.

In extending DRed for Datalog to support DatalogMTL reasoning, two major challenges arise. First, DRed relies on the seminaïve evaluation strategy applied in both the overdeletion and the insertion stages to be efficient in dealing with updates. More specifically, in each round of rule application, at least one body atom is required to be evaluated in the set of facts deleted/inserted in the previous round; this strategy ensures that only consequences dependent on the deleted/inserted facts are considered. Although seminaïve evaluation has been considered for DatalogMTL materialisation, its interplay with the construction of saturated interpretations in the context of DatalogMTL reasoning has not been studied, let alone the application of seminaïve evaluation strategy in DatalogMTL materialisation maintenance. Second, for the Datalog setting, the bulk of the work is devoted to iterative rule application, so it is sufficient to make rule application process 'incremental' in order to obtain an algorithm that is efficient for updates; however, in the DatalogMTL setting, a significant amount of time needs to be dedicated to periods identification (i.e., termination checks). Intuitively, after each round of rule application, the period identification procedure enumerates all pairs of intervals of a certain length, and then it compares pairwisely the facts in these intervals to detect the presence of a repeating pattern. Once two intervals coincide in their content, a period is identified. As such, the cost of this procedure naturally depends on the number of facts inside these intervals. Therefore, it is essential that the periods identification process is also designed to be somewhat 'incremental', so that the overall processing time aligns with the size of the updates rather than the size of the entire materialisation; this would require thorough understanding and careful treatment of the periodic structure of DatalogMTL materialisations.

Our incremental maintenance technique addresses both of these two challenges. For the first challenge, we devise a novel seminaïve evaluation operator that is tailored for incremental updates in the DatalogMTL setting: compared with the existing seminaïve operator, our new operator identifies more accurately the consequences affected by the update and also facilitates period identification. To tackle the second challenge, our algorithm is designed to identify periods in the updated facts and their consequences, rather than in the entire materialisation. Typically, deletions and insertions are small compared with the entire materialisation. As such, computing periods over the updated facts tend to be less costly than doing so for the entire materialisation; thus, our approach has the potential to significantly reduce redundant computations, especially in scenarios where updates only affect a limited portion of the materialisation. We use the following example to highlight the benefits of our approach; throughout the technical section, this example will be expanded to illustrate key operations in our algorithm.

Example 1. Let program Π consist of a single rule (2), and let E, E^-, and E^+ be the original dataset, the set of facts to remove from E, and set of facts to add to E, respectively. Now consider the materialisation of Π over E. According to the definition of $\mathsf{depth}(\Pi)$ and Definition 1, it is easy to verify that $\mathsf{depth}(\Pi) = 11$, and that when $k = 4$, interpretation $T^k_\Pi(\mathfrak{I}_E)$ is saturated, with $\varrho_{\mathsf{left}} = [-24, -23)$ and $\varrho_{\mathsf{right}} = [24, 34)$ being possible left and right periods, respectively. To arrive at the conclusion that $\varrho_{\mathsf{right}} = [24, 34)$, the materialisation procedure needs to verify that $T^k_\Pi(\mathfrak{I}_E)\,|_{[2,24]}$ is a shift of $T^k_\Pi(\mathfrak{I}_E)\,|_{[12,34]}$, and this clearly requires comparing $O(n)$ facts. To summarise, the materialisation procedure requires $O(n)$ time.

$$\boxplus_{[0,1]} R(x) \leftarrow \boxminus_{[9,10]} R(x) \tag{2}$$

$$E = \{R(a_i)@[0,1] \mid 0 < i < n\}$$

$$E^- = \{R(a_1)@[0,1]\}$$

$$E^+ = \{R(a_n)@[0,1]\}$$

Now consider the removal of E^- and the insertion of E^+. If we recompute the materialisation from scratch over the entire set $(E \setminus E^-) \cup E^+$, the procedure again requires $O(n)$ time. In contrast, our incremental update approach restricts the attention to only the updated facts and their consequences. For deletion, our approach tries to identify periodic structure only among facts of the form $R(a_1)@\varrho$. In our example, after a constant number of rule application, our algorithm will be able to identify the relevant left and right periods by only comparing $O(1)$ facts instead of $O(n)$ facts. In other words, (over)deletion requires only $O(1)$ time. Insertion follows the same principle, and so we omit the details.

DRed for DatalogMTL

Before we present the details of our algorithm, we first introduce the notion of periodic materialisations, which are finite representations of possibly unbounded models of DatalogMTL programs. Our update algorithm will have to incrementally maintain periodic materialisations in response to changes in the explicitly given data.

Definition 3. For (Π, E) a bounded DatalogMTL program-dataset pair, a periodic materialisation of (Π, E) is a triple $\mathbb{I}$ of the form $\langle I, \varrho_{\mathsf{L}}, \varrho_{\mathsf{R}}\rangle$, where I is a saturated interpretation of Π and E, and ϱ_{L} and ϱ_{R} are periods of this saturated interpretation; moreover, the unfolding of $\mathbb{I}$, written $\mathsf{unfold}(\mathbb{I})$, is defined as the $(\varrho_{\mathsf{L}}, \varrho_{\mathsf{R}})$-unfolding of I.

By Definition 3 and properties already discussed in the Preliminary Section, if $\mathbb{I}$ is a periodic materialisation of a program-dataset pair (Π, E), then $\mathsf{unfold}(\mathbb{I})$ coincides with $\mathfrak{C}_{\Pi,E}$, the canonical model of Π and E. Note that $\mathbb{I}$ can capture the interpretation of a bounded dataset I, in which case $\mathbb{I} = \langle I, \varrho_{\mathsf{L}}, \varrho_{\mathsf{R}}\rangle$ such that $I\,|_{\varrho_{\mathsf{L}}} = I\,|_{\varrho_{\mathsf{R}}} = \mathfrak{I}_\varnothing$, and $\mathsf{unfold}(\mathbb{I}) = I$.

Algorithm 1: DRED$_{MTL}(\Pi, E, \mathbb{I}, E^-, E^+)$

Input: program Π, original dataset E, periodic materialisation $\mathbb{I}$ of (Π, E), dataset to remove E^-, dataset to insert E^+

1 $D \coloneqq R \coloneqq A \coloneqq \varnothing,\ E^- \coloneqq (E^- \cap E) \setminus E^+,\ E^+ \coloneqq E^+ \setminus E$
2 OVERDELETE
3 REDERIVE
4 INSERT

5 **procedure** OVERDELETE
6 $\quad N_D \coloneqq E^-$
7 $\quad$ **loop**
8 $\quad\quad \Delta_D \coloneqq N_D \setminus D$
9 $\quad\quad (\varrho_{\mathsf{L}}^D,\ \varrho_{\mathsf{R}}^D) \coloneqq \textsc{Pds}(\Pi, E, \mathbb{I}.\varrho_{\mathsf{L}}, \mathbb{I}.\varrho_{\mathsf{R}}, D, \Delta_D)$
10 $\quad\quad$ **if** $(\varrho_{\mathsf{L}}^D,\ \varrho_{\mathsf{R}}^D) \neq (\varnothing, \varnothing)$ **then break**
11 $\quad\quad N_D \coloneqq \Pi\langle \mathsf{unfold}(\mathbb{I}) \setminus D : \Delta_D \rangle$
12 $\quad\quad D \coloneqq D \cup \Delta_D$
13 $\quad \mathbb{D} \coloneqq \langle D, \varrho_{\mathsf{L}}^D, \varrho_{\mathsf{R}}^D \rangle, \quad \mathbb{I} \coloneqq \mathbb{I} \setminus\!\!\setminus \mathbb{D}$

14 **procedure** REDERIVE
15 $\quad N_R \coloneqq \varnothing, \quad k = 1$
16 $\quad$ **loop**
17 $\quad\quad t_{\mathsf{L}} = (\mathbb{I}.\varrho_{\mathsf{L}})^+ - k \times \mathsf{max}(2\mathsf{depth}(\Pi), |\mathbb{I}.\varrho_{\mathsf{L}}|)$
18 $\quad\quad t_{\mathsf{R}} = (\mathbb{I}.\varrho_{\mathsf{R}})^- + k \times \mathsf{max}(2\mathsf{depth}(\Pi), |\mathbb{I}.\varrho_{\mathsf{R}}|)$
19 $\quad\quad N_R \coloneqq N_R \cup (\mathsf{unfd}(\mathbb{D})|_{[t_{\mathsf{L}}, t_{\mathsf{R}}]} \cap T_\Pi(\mathsf{unfd}(\mathbb{I})))$
20 $\quad\quad \Delta_R \coloneqq N_R \setminus R$
21 $\quad\quad (\varrho_{\mathsf{L}}^R, \varrho_{\mathsf{R}}^R) \coloneqq \textsc{Pds}(\Pi, E, \mathbb{I}.\varrho_{\mathsf{L}}, \mathbb{I}.\varrho_{\mathsf{R}}, R, \Delta_R)$
22 $\quad\quad$ **if** $(\varrho_{\mathsf{L}}^R,\ \varrho_{\mathsf{R}}^R) \neq (\varnothing, \varnothing)$ **then break**
23 $\quad\quad R \coloneqq R \cup \Delta_R, \quad k \coloneqq k + 1$
24 $\quad\quad N_R \coloneqq \Pi\langle \mathsf{unfold}(\mathbb{I}) \cup R : \Delta_R \rangle$
25 $\quad \mathbb{R} \coloneqq \langle R, \varrho_{\mathsf{L}}^R, \varrho_{\mathsf{R}}^R \rangle, \quad \mathbb{I} \coloneqq \mathbb{I} \uplus \mathbb{R}$

26 **procedure** INSERT
27 $\quad N_A \coloneqq E^+, \quad E = (E \setminus E^-) \cup E^+$
28 $\quad$ **loop**
29 $\quad\quad \Delta_A \coloneqq N_A \setminus (\mathsf{unfold}(\mathbb{I}) \cup A)$
30 $\quad\quad (\varrho_{\mathsf{L}}^A, \varrho_{\mathsf{R}}^A) \coloneqq \textsc{Pds}(\Pi, E, \mathbb{I}.\varrho_{\mathsf{L}}, \mathbb{I}.\varrho_{\mathsf{R}}, A, \Delta_A)$
31 $\quad\quad$ **if** $(\varrho_{\mathsf{L}}^A,\ \varrho_{\mathsf{R}}^A) \neq (\varnothing, \varnothing)$ **then break**
32 $\quad\quad A \coloneqq A \cup \Delta_A$
33 $\quad\quad N_A \coloneqq \Pi\langle \mathsf{unfold}(\mathbb{I}) \cup A : \Delta_A \rangle$
34 $\quad \mathbb{A} \coloneqq \langle A, \varrho_{\mathsf{L}}^A, \varrho_{\mathsf{R}}^A \rangle, \quad \mathbb{I} \coloneqq \mathbb{I} \uplus \mathbb{A}$

Algorithm Overview Our incremental materialisation maintenance algorithm for DatalogMTL is formalised in Algorithm 1. The algorithm takes as input a program Π, a dataset E, a periodic materialisation $\mathbb{I}$ of (Π, E), a dataset E^- to be removed from E, and a dataset E^+ to be added to E, and it updates $\mathbb{I}$ such that after the execution of the algorithm, $\mathbb{I}$ becomes a periodic materialisation of $(\Pi, (E \setminus E^-) \cup E^+)$, and this is achieved without recomputing the periodic materialisation from scratch. Similar to the DRed algorithm for plain Datalog, Algorithm 1 consists of three stages, which we outline below.

In the overdeletion stage, the dataset D is extended with all facts that depend on the deleted facts. The algorithm then applies the rules of Π iteratively (line 11) until a pair of periods are found (line 9–10). Line 11 makes use of our newly devised seminaïve evaluation operator $\Pi\langle I : \Delta \rangle$, which computes the union of all facts derived by $(r\sigma, t)\langle I : \Delta \rangle$, where $r\sigma$ is a ground rule instance with r a rule in Π and σ a substitution mapping each variable appearing in r to a constant appearing in I, and t is a rational time point; operator $(r', t)\langle I : \Delta \rangle$ where r' is a ground rule instance and t is a rational time point is in turn defined as the minimal set of punctual facts N such that $\mathfrak{I}_I, t \vDash body(r')$, $\mathfrak{I}_{I \setminus \Delta}, t \nvDash body(r')$, and $\mathfrak{I}_N, t \vDash head(r')$. When I and Δ are finite, $\Pi\langle I : \Delta \rangle$ can be efficiently evaluated by instantiating the query from facts inside Δ, which is typically much smaller than I. However, in line 21, by slight abuse of notation, we use $\mathsf{unfold}(\mathbb{I}) \setminus D$ as an operand of the operator: this does not mean that we have to compute the entire unfolding of $\mathbb{I}$ prior to the execution of the operator. In contrast, the evaluation is still instantiated from from facts in Δ, while interpretation $\mathbb{I}$ can be unfolded lazily as required.

In the rederivation stage, the algorithm recovers the facts that were overdeleted but should actually still hold after the update. It should be noted that overdeleted facts may span the entire timeline, and so it may not be sufficient to recover facts within a fixed interval. Lines 17–19 address this issue, in each round of the loop of lines 16–24, we extend the interval of interest $[t_{\mathsf{L}}, t_{\mathsf{R}}]$ so that overdeleted facts inside this interval could be successfully recovered (line 19); this is done in parallel to the propagation of already recovered facts (line 24). The loop terminates when a pair of periods are successfully identified (lines 21–22), and the periodic materialisation $\mathbb{I}$ is updated in line 25.

Insertion is analogous to deletion: the dataset A is populated with new facts that are derivable from E^+. In parallel to consequence propagation, the algorithm tries to detect the periodic structure within the dataset A. Ultimately, the algorithm updates the periodic materialisation in line 34 to incorporate the inserted facts and their consequences.

Period Identification The PDS procedure formalised in Algorithm 2 identifies the repeating patterns within a dataset being populated during iterative application of rules. More specifically, it takes as input a program Π, a dataset E, intervals ϱ_{L} and ϱ_{R}, the dataset being popuplated D, and a set of new facts Δ_D that are to be integrated into D after period detection. The procedure first examines facts in Δ_D: if there is a fact in Δ_D that overlaps with the interval $[t_E^-, t_E^+]$, it means facts inside this interval have not stabilised, and so the procedure terminates. Otherwise, u and v will be computed separately (lines 5–6) such that $[u, v]$ is largest interval containing $[t_E^-, t_E^+]$ and does not contain a fact satisfied by $\mathfrak{I}_{\Delta_D}$. Now if $[u, v]$ is nonempty, we search for the left and right periods with endpoints on the (Π, E)– ruler in $\gamma_{\mathsf{L}} = (u, \varrho_{\mathsf{L}}^+ + 2\mathsf{depth}(\Pi)]$ and $\gamma_{\mathsf{R}} = [\varrho_{\mathsf{right}}^- - 2\mathsf{depth}(\Pi), v)$, respectively (lines 9–16). The procedure returns the pair of periods $(\varrho_{\mathsf{L}}', \varrho_{\mathsf{R}}')$, or $(\varnothing, \varnothing)$ if no valid period is detected at either end (lines 17–18). Intuitively, the algorithm requires ϱ_1, ϱ_2 (resp., ϱ_3, ϱ_4) to be a multiple of $|\varrho_{\mathsf{L}}|$ (resp., $|\varrho_{\mathsf{R}}|$) apart so that it would be convenient align the new periods with the given ones.

Algorithm 2: $\textsc{Pds}(\Pi, E, \varrho_{\mathsf{L}}, \varrho_{\mathsf{R}}, D, \Delta_D)$

Input: program Π, dataset E, intervals ϱ_{L} and ϱ_{R}, dataset D of facts derived so far, and dataset Δ_D of facts derived in the last round

$\varrho'_{\mathsf{L}} \coloneqq \varrho'_{\mathsf{R}} \coloneqq \varnothing, \quad P \coloneqq (\Pi, E)$-ruler
foreach $M\sigma@\varrho \in \Delta_D$ **do**
  if $\varrho \cap [t_E^-, t_E^+] \neq \varnothing$ **then**
    return $(\varnothing, \varnothing)$
$u = \max(\{x < t_E^- \mid \exists y, M@\langle y, x\rangle \in \Delta_D\} \cup \{-\infty\})$
$v = \min(\{x > t_E^+ \mid \exists y, M@\langle x, y\rangle \in \Delta_D\} \cup \{\infty\})$
$\gamma_{\mathsf{L}} \coloneqq (u, \quad \varrho_{\mathsf{L}}^+ + 2\mathsf{depth}(\Pi)]$
$\gamma_{\mathsf{R}} \coloneqq [\varrho_{\mathsf{R}}^- - 2\mathsf{depth}(\Pi), \quad v)$
foreach $\varrho_1, \varrho_2 \subseteq \gamma_{\mathsf{L}}$ *with* $|\varrho_1| = |\varrho_2| = 2\mathsf{depth}(\Pi)$ **do**
  if $\varrho_1^- < \varrho_2^-, \varrho_1^- \in P$ *and* $\varrho_2^- \equiv \varrho_1^- \mod |\varrho_{\mathsf{L}}|$ **then**
    if $\mathfrak{I}_D|_{\varrho_1}$ *is a shift of* $\mathfrak{I}_D|_{\varrho_2}$ **then**
      $\varrho'_{\mathsf{L}} \coloneqq [\varrho_1^-, \varrho_2^-)$
foreach $\varrho_3, \varrho_4 \subseteq \gamma_{\mathsf{R}}$ *with* $|\varrho_3| = |\varrho_4| = 2\mathsf{depth}(\Pi)$ **do**
  if $\varrho_3^+ < \varrho_4^+, \varrho_3^+ \in P$ *and* $\varrho_3^+ \equiv \varrho_4^+ \mod |\varrho_{\mathsf{R}}|$ **then**
    if $\mathfrak{I}_D|_{\varrho_3}$ *is a shift of* $\mathfrak{I}_D|_{\varrho_4}$ **then**
      $\varrho'_{\mathsf{R}} \coloneqq (\varrho_3^+, \varrho_4^+]$
if $\varrho'_{\mathsf{L}} = \varnothing$ *or* $\varrho'_{\mathsf{R}} = \varnothing$ **then return** $(\varnothing, \varnothing)$
else return $(\varrho'_{\mathsf{L}}, \varrho'_{\mathsf{R}})$

Algorithm 3: Implementation of Periodic Minus

Input: Two periodic materialisations $\mathbb{D}_1$ and $\mathbb{D}_2$

if $\mathbb{D}_2.\varrho_{\mathsf{L}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2) \coloneqq \texttt{Ext}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{L})$
  $(\mathbb{D}_1, \mathbb{D}_2, \varrho_{\mathsf{L}}) \coloneqq \texttt{Aln}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{L})$
else $\varrho_{\mathsf{L}} \coloneqq \mathbb{D}_1.\varrho_{\mathsf{L}}$
if $\mathbb{D}_2.\varrho_{\mathsf{R}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2) \coloneqq \texttt{Ext}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{R})$
  $(\mathbb{D}_1, \mathbb{D}_2, \varrho_{\mathsf{R}}) \coloneqq \texttt{Aln}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{R})$
else $\varrho_{\mathsf{R}} \coloneqq \mathbb{D}_1.\varrho_{\mathsf{R}}$
$I \coloneqq \mathbb{D}_1.I \setminus \mathbb{D}_2.I$
return $\langle I, \varrho_{\mathsf{L}}, \varrho_{\mathsf{R}}\rangle$

Implementation of Periodic Operators Algorithm 1 has made frequent use of operators $\setminus$ and $\uplus$, which are responsible for taking the difference and union of two periodic materialisations, respectively. In practice, these two operators can be implemented arbitrarily, so long as $\mathsf{unfold}(\mathbb{D}_1) - \mathsf{unfold}(\mathbb{D}_2) = \mathsf{unfold}(\mathbb{D}_1 \setminus \mathbb{D}_2)$ and $\mathsf{unfold}(\mathbb{D}_1) \cup \mathsf{unfold}(\mathbb{D}_2) = \mathsf{unfold}(\mathbb{D}_1 \uplus \mathbb{D}_2)$. Next we describe our implementation of these operators, which utilises two auxiliary functions, `Ext` and `Aln`, responsible for extending and aligning periodic materialisations, respectively.

Consider two periodic materialisations $\mathbb{D}_1$ and $\mathbb{D}_2$, and let $\mathsf{end} \in \{\mathsf{L}, R\}$ denote which end of the periodic materialisation to operate on (either left or right). Given periodic materialisations $\mathbb{D}_1$ and $\mathbb{D}_2$ which both have valid (but potentially different) periods ϱ_{end}, function `Ext` computes a pair of periodic materialisations $(\mathbb{D}'_1, \mathbb{D}'_2) = \texttt{Ext}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{end})$ such that the periodic regions are extended to have the same length, which is the least common multiple (LCM) of the two; this can be easily achieved by copying the relevant periodic segments of the timeline. In contrast, function `Aln` tries to align the start and end points of periodic intervals of the two periodic materialisations at the target end, again through facts copying. If only one of the two periodic operators has a valid ϱ_{end}, an empty period is introduced for the other periodic materialisation so that alignment can still be performed. It should be noted that both `Ext` and `Aln` operations preserve the semantics of the interpretations: they do not change which facts hold at any time point but only alters the finite representations of the input periodic materialisations.

Algorithm 4: Implementation of Periodic Union

Input: Two periodic materialisations $\mathbb{D}_1$ and $\mathbb{D}_2$

$\varrho_{\mathsf{L}} \coloneqq \varrho_{\mathsf{R}} \coloneqq \varnothing$
if $\mathbb{D}_1.\varrho_{\mathsf{L}}$ *and* $\mathbb{D}_2.\varrho_{\mathsf{L}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2) \coloneqq \texttt{Ext}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{L})$
if $\mathbb{D}_1.\varrho_{\mathsf{L}}$ *or* $\mathbb{D}_2.\varrho_{\mathsf{L}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2, \varrho_{\mathsf{L}}) \coloneqq \texttt{Aln}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{L})$
if $\mathbb{D}_1.\varrho_{\mathsf{R}}$ *and* $\mathbb{D}_2.\varrho_{\mathsf{R}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2) \coloneqq \texttt{Ext}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{R})$
if $\mathbb{D}_1.\varrho_{\mathsf{R}}$ *or* $\mathbb{D}_2.\varrho_{\mathsf{R}}$ **then**
  $(\mathbb{D}_1, \mathbb{D}_2, \varrho_{\mathsf{R}}) \coloneqq \texttt{Aln}(\mathbb{D}_1, \mathbb{D}_2, \mathsf{R})$
$I \coloneqq \mathbb{D}_1.I \cup \mathbb{D}_2.I$
return $\langle I, \varrho_{\mathsf{L}}, \varrho_{\mathsf{R}}\rangle$

Theorem 1 states that Algorithms 3 and 4 correctly implement operators $\setminus$ and $\uplus$, respectively; Theorem 2 states that Algorithm 1 is correct. The full proofs for these theorems are lengthy, so we only provide proof sketches; the full proofs are given in the online technical report.

Theorem 1. *If given input* $\mathbb{D}_1$,$\mathbb{D}_2$*, Algorithm 3 and 4 output* $\mathbb{D}_m$ *and* $\mathbb{D}_u$*, respectively, then* $\mathsf{unfold}(\mathbb{D}_1) - \mathsf{unfold}(\mathbb{D}_2) = \mathsf{unfold}(\mathbb{D}_m)$ *and* $\mathsf{unfold}(\mathbb{D}_1) \cup \mathsf{unfold}(\mathbb{D}_2) = \mathsf{unfold}(\mathbb{D}_u)$

Proof Sketch. For minus, we show that the equation holds by leveraging that the periods of $\mathsf{unfold}(\mathbb{D}_1)$ and $\mathsf{unfold}(\mathbb{D}_2)$ can be extended and aligned to form a new pair of periods, and their difference also abides by the new periods. The case for union is similar. □

Theorem 2. *For a bounded DatalogMTL program* Π*, a bounded dataset* E*, a bounded deleting dataset* E^-*, a bounded inserting dataset* E^+*, a periodic materialisation* $\mathbb{I}$ *such that* $\mathsf{unfold}(\mathbb{I}) = \mathfrak{C}_{\Pi,E}$*, let* $E' = E \setminus E^- \cup E^+$*, then after calling* $\textbf{DRED}_{MTL}(\Pi, E, E^-, E^+, \mathbb{I})$*, we have*

$$\mathsf{unfold}(\mathbb{I}) = \mathfrak{C}_{\Pi,E'}$$

Proof Sketch. We first show that $\mathsf{unfold}(\mathbb{D})$ computed by the OVERDELETE contains $\mathfrak{C}_{\Pi,E} - \mathfrak{C}_{\Pi,E\setminus E^-\cup E^+}$, which means every fact that no longer holds because of deleting E^- will be removed, and then we show that the union of $\mathsf{unfold}(\mathbb{R}), \mathsf{unfold}(\mathbb{A})$ computed by the REDERIVE,INSERT and $\mathsf{unfold}(\mathbb{I})$ after deletion equals $\mathfrak{C}_{\Pi,E\setminus E^-\cup E^+}$, which involves proving mistakenly deleted facts are rederived and every new fact that holds because of E^+ are inserted. □

Evaluation

We implemented the proposed DRed_{MTL} algorithm based on an efficient DatalogMTL reasoner MeTeoR (Wang et al. 2022) and empirically tested the performance of our implementation on three publicly available datasets. We chose MeTeoR as its latest version supports materialisation for DatalogMTL with bounded intervals (Wałęga et al. 2023), which allows us to directly compare the performance of DRed_{MTL} with rematerialisation from scratch. The source code of our implementation, the benchmarks we used, as well as an extended technical report containing all detailed proofs, are available online.

Benchmarks LUBM_t (Wang et al. 2022) is a temporal version of the well-known LUBM benchmark (Guo, Pan, and Heflin 2005). It has a recursive program consisting of 85 rules, of which 29 have temporal operators in them (denoted as $|\Pi_{mtl}|$) and 56 do not (denoted as $|\Pi_{no\_mtl}|$). The iTemporal dataset is generated from a temporal benchmark generator developed by Bellomarini, Nissl, and Sallinger (2022). Its program is highly recursive and consists of 12 rules. Finally, the Meteorological dataset (Maurer et al. 2002) contains long-term meteorological observations. It has a nonrecursive program consisting of four rules, of which two contain temporal operators. These datasets were collected and made publicly available by Wang et al. (2025), and we used them without any modification. The statistics of these datasets is given in Table 2, where $|E|$ and $|I|$ are the number of explicitly given facts and the number of facts in the saturated interpretation, respectively. The ratio between $|I|$ and $|E|$ is usually a good indicator of the recursiveness of the corresponding rule set: the larger the ratio is, the more recursive and complex the rule set is and the more likely it is to generate a greater number of facts. Indeed, our choice of benchmarks is a nice mixture of highly recursive (iTemporal), mildly recursive (LUBM_t) and nonrecursive datasets, which allows us to test the potential of our reasoning algorithm in different possible application scenarios.

Test Settings We conducted our experiment on a server with 256GB RAM and an Intel Xeon Platinum 8269CL 2.50GHz CPU, running Fedora Linux 40, kernel version Linux 6.10.10-200.fc40.x86_64. Our evaluation primarily examines the capabilities of DRed_{MTL} in handling both deletions and insertions. Table 3 summarises the performance comparison of our algorithm against rematerialisation. For our approach, we record the wall-clock time of running Algorithm 1 to handle the updates. For the baseline, we record the wall-clock time that MeTeoR spends on computing the canonical representation over the updated set of explicitly given facts from scratch. In addition to the time metrics, to better reflect the workload of DRed_{MTL}, we also record the number of facts derived in the three stages of our algorithm, i.e., overdeletion ($|D|$), rederivation ($|R|$) and insertion ($|A|$). Note that for insertion, the sets of overdeleted and rederived facts are always empty, so we only record the number of inserted facts, $|A|$.

	LUBM_t	iTemporal	Meteorological
$\lvert\Pi_{mtl}\rvert$	29	3	2
$\lvert\Pi_{no\_mtl}\rvert$	56	8	2
$\lvert\Pi\rvert$	85	11	4
$Recursive$	Yes	Yes	No
$\lvert E\rvert$	630.5k	46.41k	62.01M
$\lvert I\rvert$	1.426M	30.62M	62.48M

Table 2: Dataset Statistics

Our evaluation considers both small-scale and large-scale updates. For small-scale update tests, we randomly selected 100 facts and ran DRed_{MTL} to deal with the deletion; then we added these facts back and recorded the time DRed_{MTL} took to handle the insertion. Large-scale tests were performed in a similar fashion, except that 10% of the original dataset were removed and added back; the exact numbers of deleted facts are shown in Table 3. For each test case, three test runs with different (randomly selected) updates were performed; the results showed no significant variation in terms of running time. Therefore, for the ease of presentation we only reported the results of one run for each test case. Finally, for each test run, we made sure that the periodic materialisation produced by DRed_{MTL} ($\mathbb{I}_1$) is equivalent to that produced by rematerialisation ($\mathbb{I}_2$): this is achieved by verifying that both $\mathbb{I}_1 \setminus \mathbb{I}_2$ and $\mathbb{I}_2 \setminus \mathbb{I}_1$ are empty.

Results Our evaluation shows that DRed_{MTL} consistently outperforms rematerialisation for all small deletions and insertions. As shown in Table 3, on LUBM_t, DRed_{MTL} is 69.4 times faster than the baseline for small deletion, and 121.3 times faster for small insertion. On the nonrecursive Meteorological dataset, DRed_{MTL} achieves similar performance improvement for small updates. On iTemporal, the performance improvement is more modest: the speedup is around six times. This is so since the program of iTemporal is highly recursive, making incremental maintenance especially challenging: as one can see, deleting only 100 facts leads to the overdeletion of over 244 thousand facts.

For large updates, DRed_{MTL} achieved a significant performance boost on the LUBM_t and iTemporal datasets. When deleting 10% of the data, DRed_{MTL} is 13.2 times faster on LUBM_t and 4.2 times faster on iTemporal. For large insertion, the improvements are 43.8 and 4.2 times, respectively. Interestingly, on the Meteorological dataset, DRed_{MTL} is slower than rematerialisation in dealing with large updates. Notice that for this dataset, the ratio between $|I|$ and $|E|$ is rather small, as depicted in Table 2, indicating that only a small fraction of the materialisation are inferred facts; moreover, the program is nonrecursive, so no effort is required for identifying the repeating pattern inside the materialisation. As such, for large updates on this dataset, DRed_{MTL}, which is based on efficient seminaive reasoning and period identification, losts its advantage.

To further analyse the performance of the proposed algorithm, we profiled our system on the deletion test cases and reported the runtime breakdown (by stage of reason-

'Remat' stands for Rematerialisation from the updated set of explicitly given facts.

Dataset	$\lvert E^{\pm}\rvert$	Deletion					Insertion		
		DRED$_{MTL}$				Remat	DRED$_{MTL}$		Remat
		Time(s)	$\lvert D\rvert$	$\lvert R\rvert$	$\lvert A\rvert$	Time(s)	Time(s)	$\lvert A\rvert$	Time(s)
LUBM$_t$	100	0.7k	11.5k	1.5k	11.4k	48.6k	0.4k	0.2k	48.5k
	63.1k	3.4k	372.1k	143.5k	251.9k	45.0k	1.1k	190.2k	48.2k
iTemporal	100	8.1k	244.5k	274.0k	274.4k	52.7k	8.7k	231.2k	52.8k
	4.6k	12.6k	8.963M	1.286M	1.288M	52.3k	12.4k	7.829M	52.6k
Meteorological	100	10	102	1	1	0.7k	11	101	0.7k
	6.201M	1.4k	6.271M	1105	1105	0.7k	1.2k	6.270M	0.7k

Table 3: Evaluation Results

Dataset	$\lvert E^{-}\rvert$	Overdeletion	Rederivation	Insertion
LUBM$_t$	100	59.0%	2.4%	38.6%
	63.1k	15.7%	67.9%	16.4%
iTemporal	100	70.5%	10.6%	18.9%
	4.6k	64.2%	29.9%	5.9%
Meteorological	100	62.8%	21.8%	15.4%
	6.201M	92.2%	7.7%	0.1%

Table 4: Deletion Test Runtime Breakdown

ing) in Table 4. It could be readily observed that in most cases, overdeletion is the most time consuming step of the algorithm. This is so since overdeletion produces the largest number of facts across the three stages. The only exception is the case of large deletion for LUBM$_t$: rederivation consumes more time than overdeletion and insertion combined. Indeed, although rederivation produces less facts than overdeletion, it involves evaluating rules 'backwards' from head to body, which, as observed by Hu, Motik, and Horrocks (2018), could be more expensive than the seminaïve evaluation taking place in overdeletion and insertion. In the Datalog setting, enhancing rederivation with counting could help alleviate this issue, but extending the counting technique to DatalogMTL is beyond the scope of this paper.

Overall our test results suggest that the proposed algorithm improves substantially and consistently over rematerialisation, for both small and large updates. In some test cases, the running time decreased from several hours to several minutes, or from an hour to a few seconds, demonstrating the potential of deploying the proposed approach in industrial applications where short service response time is highly desirable.

Conclusion and Future Work

In this paper, we have introduced a new technique for incrementally updating DatalogMTL materialisations. Compared with recomputing the materialisation from scratch, our technique achieves significant performance improvements, especially when the updates are small.

We see many exciting future research directions. From a practical perspective, it would be interesting to see how well the proposed approach works in industrial scenarios such as IoT anomaly detection (Zhang et al. 2024): in typical such applications, DatalogMTL can be used to model anomaly detection rules that are triggered by streams composed of timestamped sensor data; the ability to reason incrementally offered by our method is crucial for ensuring efficient and reliable fault alerts. Moreover, we shall consider comparing the performance of our system with well-established stream reasoning frameworks such as C-SPARQL (Barbieri et al. 2009) and CQELS (Phuoc et al. 2011). While the underlying languages are quite different, it should be possible to transform our reasoning workload (at least the nonrecursive cases) to a C-SPARQL or CQELS workload, and test it on the corresponding engine. Last but not least, DRed is not the only incremental update algorithm for Datalog materialisations; algorithms such as B/F, FBF, and DRed^c are popular alternatives of DRed and sometimes offer superior performance. It would thus be useful to consider extending these incremental maintenance algorithms for Datalog to support DatalogMTL reasoning, and to compare the performance of the adapted algorithms with that of ours.

Acknowledgements

This work was generously funded by National Science and Technology Major Project of China under grant number 2025ZD1600800 and National Natural Science Foundation of China under grant number 62206169. We thank the anonymous reviewers for their constructive comments that helped greatly in shaping the final version of this paper.

References

Barbieri, D. F.; Braga, D.; Ceri, S.; Della Valle, E.; and Grossniklaus, M. 2009. C-SPARQL: SPARQL for continuous querying. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, 1061–1062. New York, NY, USA: Association for Computing Machinery. ISBN 9781605584874.

Bellomarini, L.; Nissl, M.; and Sallinger, E. 2022. iTemporal: An Extensible Generator of Temporal Benchmarks. In *International Conference on Data Engineering, ICDE 2022*, 2021–2033.

Brandt, S.; Kalaycı, E. G.; Kontchakov, R.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2017. Ontology-based data access with a horn fragment of metric temporal logic. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 1070–1076. AAAI Press.

Brandt, S.; Kalayci, E. G.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2018. Querying Log Data with Metric Temporal Logic. *J. Artif. Intell. Res.*, 62: 829–877.

Ceri, S.; Gottlob, G.; and Tanca, L. 1989. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. Knowl. Data Eng.*, 1(1): 146–166.

Colombo, A.; Bellomarini, L.; Ceri, S.; and Laurenza, E. 2023. Smart Derivative Contracts in DatalogMTL. In *International Conference on Extending Database Technology, EDBT*, 773–781.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. Web Semant.*, 3(2-3): 158–182.

Gupta, A.; Mumick, I. S.; and Subrahmanian, V. S. 1993. Maintaining views incrementally. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, 157–166. New York, NY, USA: Association for Computing Machinery. ISBN 0897915925.

Güzel Kalayci, E.; Xiao, G.; Ryzhikov, V.; Kalayci, T. E.; and Calvanese, D. 2018. Ontop-temporal: a tool for ontology-based query answering over temporal data. In *ACM International Conference on Information and Knowledge Management*, 1927–1930.

Hu, P.; Motik, B.; and Horrocks, I. 2018. Optimised Maintenance of Datalog Materialisations. In McIlraith, S. A.; and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018*, 1871–1879. AAAI Press.

Koymans, R. 1990. Specifying Real-Time Properties with Metric Temporal Logic. *Real Time Syst.*, 2(4): 255–299.

Maurer, E. P.; Wood, A. W.; Adam, J. C.; Lettenmaier, D. P.; and Nijssen, B. 2002. A Long-Term Hydrologically Based Dataset of Land Surface Fluxes and States for the Conterminous United States. *Journal of Climate*, 15(22): 3237 – 3251.

Mori, M.; Papotti, P.; Bellomarini, L.; and Giudice, O. 2022. Neural Machine Translation for Fact-checking Temporal Claims. In *Fact Extraction and VERification Workshop*, 78–82.

Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2015. Incremental update of datalog materialisation: the backward/forward algorithm. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 1560–1568. AAAI Press. ISBN 0262511290.

Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2019. Maintenance of datalog materialisations revisited. *Artificial Intelligence*, 269: 76–136.

Nissl, M.; and Sallinger, E. 2022. Modelling Smart Contracts with DatalogMTL. In *Workshops of the EDBT/ICDT*, volume 3135.

Phuoc, D. L.; Dao-Tran, M.; Parreira, J. X.; and Hauswirth, M. 2011. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In Aroyo, L.; Welty, C.; Alani, H.; Taylor, J.; Bernstein, A.; Kagal, L.; Noy, N. F.; and Blomqvist, E., eds., *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, 370–388. Springer.

Ren, Y.; and Pan, J. Z. 2011. Optimising ontology stream reasoning with truth maintenance system. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, 831–836. New York, NY, USA: Association for Computing Machinery. ISBN 9781450307178.

Staudt, M.; and Jarke, M. 1996. Incremental Maintenance of Externally Materialized Views. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, 75–86. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558603824.

Urbani, J.; Margara, A.; Jacobs, C.; van Harmelen, F.; and Bal, H. 2013. DynamiTE: Parallel Materialization of Dynamic RDF Data. In Alani, H.; Kagal, L.; Fokoue, A.; Groth, P.; Biemann, C.; Parreira, J. X.; Aroyo, L.; Noy, N.; Welty, C.; and Janowicz, K., eds., *The Semantic Web – ISWC 2013*, 657–672. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-41335-3.

Wałęga, P.; Kaminski, M.; and Cuenca Grau, B. 2019. Reasoning over Streaming Data in Metric Temporal Datalog. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 3092–3099. AAAI Press.

Wałęga, P. A.; Zawidzki, M.; Wang, D.; and Cuenca Grau, B. 2023. Materialisation-Based Reasoning in DatalogMTL with Bounded Intervals. In *AAAI Conference on Artificial Intelligence*, 6566–6574.

Wałęga, P. A.; Kaminski, M.; Wang, D.; and Grau, B. C. 2023. Stream reasoning with DatalogMTL. *Journal of Web Semantics*, 76: 100776.

Wang, D.; Hu, P.; Wałęga, P. A.; and Cuenca Grau, B. 2022. MeTeoR: Practical Reasoning in Datalog with Metric Temporal Operators. In *AAAI Conference on Artificial Intelligence*, 5906–5913.

Wang, S.; Zhao, K.; Wei, D.; Walega, P. A.; Wang, D.; Cai, H.; and Hu, P. 2025. Goal-Driven Reasoning in DatalogMTL with Magic Sets. In Walsh, T.; Shah, J.; and Kolter, Z., eds., *The Thirty-Ninth AAAI Conference on Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, 15203–15211. AAAI Press.

Zhang, F.; Hu, P.; Cai, H.; and Jiang, L. 2024. Parallel Collaborative Reasoning Approaches Based on DatalogMTL in IoT Scenarios. In *2024 27th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 1055–1060.