TPLP **25** (2): 225–255, 2025.
doi:10.1017/S1471068424000164 First published online 28 October 2024

Practical Reasoning in DatalogMTL[†]

DINGMIN WANG and BERNARDO CUENCA GRAU
Department of Computer Science, University of Oxford, Oxford, UK
(e-mails: dingmin.wang@cs.ox.ac.uk, bernardo.cuenca.grau@cs.ox.ac.uk)

PRZEMYSŁAW A. WAŁĘGA
Department of Computer Science, University of Oxford, Oxford, UK
School of Electronic Engineering and Computer Science, Queen Mary University of London, London, UK
(e-mail: przemyslaw.walega@cs.ox.ac.uk)

PAN HU
Department of Computer Science, University of Oxford, Oxford, UK
School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China,
(e-mail: pan.hu@sjtu.edu.cn)

submitted 13 March 2023; revised 20 October 2023; accepted 9 June 2024

Abstract

DatalogMTL is an extension of Datalog with metric temporal operators that has found an increasing number of applications in recent years. Reasoning in DatalogMTL is, however, of high computational complexity, which makes reasoning in modern data-intensive applications challenging. In this paper we present a practical reasoning algorithm for the full DatalogMTL language, which we have implemented in a system called MeTeoR. Our approach effectively combines an optimised (but generally non-terminating) materialisation (a.k.a. forward chaining) procedure, which provides scalable behaviour, with an automata-based component that guarantees termination and completeness. To ensure favourable scalability of the materialisation component, we propose a novel seminaïve materialisation procedure for DatalogMTL enjoying the non-repetition property, which ensures that each rule instance will be applied at most once throughout its entire execution. Moreover, our materialisation procedure is enhanced with additional optimisations which further reduce the number of redundant computations performed during materialisation by disregarding rules as soon as it is certain that they cannot derive new facts in subsequent materialisation steps. Our extensive evaluation supports the practicality of our approach.

KEYWORDS: temporal reasoning, metric temporal logic, datalogMTL

[†]This paper extends our conference publications at the AAAI Conference on Artificial Intelligence (Wang *et al.* 2022) and at the International Joint Conference on Rules and Reasoning (Wang *et al.* 2022).

1 Introduction

DatalogMTL (Brandt *et al.* 2018) is an extension of the core rule-based language Datalog (Ceri *et al.* 1989) with operators from metric temporal logic (MTL) (Koymans 1990) interpreted over the rational timeline. For example, the following DatalogMTL rule states that travellers can enter the US if they had a negative COVID-19 test sometime in the last 2 days ($\diamondminus_{[0,2]}$) and have held fully vaccinated status continuously throughout the last 15 days ($\boxminus_{[0,15]}$)

$$\mathit{Authorised}(x) \leftarrow \diamondminus_{[0,2]} \mathit{NegativeLFT}(x) \wedge \boxminus_{[0,15]} \mathit{FullyVaccinated}(x).$$

DatalogMTL is a powerful temporal knowledge representation language, which has recently found applications in ontology-based query answering (Brandt *et al.* 2018; Kikot *et al.* 2018; Kalaycı *et al.* 2018; Koopmann 2019) and stream reasoning (Wałęga *et al.* 2019b), amongst others (Nissl and Sallinger 2022; Mori *et al.* 2022). Reasoning in DatalogMTL is, however, of high complexity, namely ExpSpace-complete (Brandt *et al.* 2018) and PSpace-complete with respect to data size (Wałęga *et al.* 2019a), which makes reasoning in data-intensive applications challenging. Thus, theoretical research has focused on establishing a suitable trade-off between expressive power and complexity of reasoning, by identifying lower complexity fragments of DatalogMTL (Wałęga *et al.* 2021,2023) and studying alternative semantics with favourable computational behaviour (Wałęga *et al.* 2020; Ryzhikov *et al.* 2019).

The design and implementation of practical reasoning algorithms for DatalogMTL remains, however, a largely unexplored area – something that has so far prevented its widespread adoption in applications. Brandt *et al.* (2018) implemented a prototype reasoner based on query rewriting that is applicable only to non-recursive DatalogMTL programmes; in turn, the temporal extension of the Vadalog system (Bellomarini *et al.* 2018) described by Bellomarini *et al.* (2022) implements the full DatalogMTL language, but without termination guarantees. The lack of available reasoners for DatalogMTL is in stark contrast with plain Datalog, for which a plethora of (both academic and commercial) systems have been developed and successfully deployed in practice (Motik *et al.* 2014; Bellomarini *et al.* 2018; Carral *et al.* 2019).

In this paper, we present the first practical reasoning algorithm for the full DatalogMTL language, which is sound, complete, and terminating. Our algorithm combines materialisation (a.k.a. forward chaining) and automata-based reasoning. On the one hand, materialisation is the reasoning paradigm of choice in numerous Datalog systems (Bry *et al.* 2007; Motik *et al.* 2014; Bellomarini *et al.* 2018; Carral *et al.* 2019); facts logically entailed by an input programme and dataset are derived in successive rounds of rule applications (also called *materialisation steps*) until a fixpoint is reached or a fact of interest (or a contradiction) is derived; both this process and its output are often referred to as *materialisation*. A direct implementation of materialisation-based reasoning in DatalogMTL is, however, problematic since forward chaining may require infinitely many rounds of rule applications (Wałęga *et al.* 2021,2023). An alternative to materialisation-based reasoning ensuring completeness and termination relies on constructing B FC;chi automata and checking non-emptiness of their languages (Wałęga *et al.* 2019a). This procedure, however, was mainly proposed for obtaining tight complexity bounds, and not with efficient implementation in mind; in particular, the constructed automata are of exponential size, which makes direct implementations impractical. Our approach deals with these difficulties by providing optimised materialisation-based algorithms together with an effective way of combining the scalability of materialisation-based reasoning and the completeness guaranteed by automata-based procedures, thus providing "the best of both worlds."

After discussing related work in Section 2 and preliminary definitions for DatalogMTL in Section 3, we will present the following contributions of our work.

1. In Section 4 we recapitulate different techniques available for reasoning in DatalogMTL, which we will take as a starting point for the development of our approach. On the one hand, we present a variant of the (non-terminating) naïve materialisation procedure (Wałęga *et al.* 2023), and prove its soundness and completeness; on the other hand, we describe existing algorithms based on exponential reductions to reasoning in linear temporal logic (LTL) and to emptiness checking of B FC;chi automata.
2. In Section 5.1 we present a *seminaïve* materialisation procedure for DatalogMTL. In contrast to the naïve procedure provided in Section 4.1 and analogously to the classical seminaïve algorithm for plain Datalog (Abiteboul *et al.* 1995; Motik *et al.* 2019), our procedure aims at minimising redundant computation by keeping track of newly derived facts in each materialisation step and ensuring that rule applications in the following materialisation step involve at least one of these newly derived facts. In this way, each rule instance is considered at most once, and so our procedure is said to enjoy the *non-repetition* property.
3. In Section 5.2 we present an optimised variant of our seminaïve procedure which further reduces the number of redundant computations performed during materialisation by disregarding rules during the execution of the procedure as soon as we can be certain that their application will never derive new facts in subsequent materialisation steps.
4. In Section 5.3 we propose a practical reasoning algorithm combining optimised seminaïve materialisation and our realisation of the automata-based reasoning approach of (Wałęga *et al.* 2019a). Our algorithm is designed to delegate the bulk of the computation to the scalable materialisation component and resort to automata-based techniques only as needed to ensure termination and completeness.
5. We have implemented our approach in the MeTeoR (Metric Temporal Reasoner) system, which we have made publicly available.[1] In Section 6 we describe our implementation and present its evaluation on a temporal extension of the Lehigh University Benchmark (LUBM), the Weather Benchmark used by Brandt *et al.* (2018), and the iTemporal benchmark generator developed by Bellomarini *et al.* (2022). In Section 6.4 we describe the results of our evaluation and draw the following conclusions.

   – Although completeness and termination in our approach can be ensured by relying on either automata construction or on a reduction to LTL satisfiability followed by the application of a LTL reasoner, our experiments show that automata construction yields superior performance and can be used to solve non-trivial problems; this justifies our choice of automata over LTL in MeTeoR.

[1] See https://github.com/wdimmy/DatalogMTL˙Practical˙Reasoning for the version of MeTeoR described and evaluated in this paper.

- Our proposed seminaïve materialisation strategy offers significant performance and scalability gains over naïve materialisation, and it can be successfully applied to non-trivial programmes and datasets with tens of millions of temporal facts.
- The performance of MeTeoR is superior to that of the query rewriting approach proposed by Brandt *et al.* (2018) on non-recursive programmes.
- The vast majority of queries for programmes generated by iTemporal can be answered using materialisation only. This supports the practicality of our approach since the bulk of the workload is delegated to the scalable materialisation component and the use of automata-based reasoning is only required in exceptional cases.

This paper is supplemented by a technical appendix containing all the proofs to our claims.

2 Related work

DatalogMTL was first introduced by Brandt *et al.* (2017) and it has found applications in areas as diverse as temporal stream reasoning (Wałęga *et al.* 2019b), temporal ontology-based data access (Brandt *et al.* 2017), specification and verification of banking agreements (Nissl and Sallinger 2022), fact-checking economic claims (Mori *et al.* 2022), and the description of human movements (Raheb *et al.* 2017). The complexity of standard reasoning tasks in both the full language and its fragments has been investigated in depth (Brandt *et al.* 2018; Wałęga *et al.* 2019a,2021; Bellomarini *et al.* 2021). Alternative semantics to the standard continuous semantics over the rational timeline have also been considered; these include the semantics based on the integer timeline studied by Wałęga *et al.* (2020) and the event-based semantics by Ryzhikov *et al.* (2019). DatalogMTL has also been recently extended with negation-as-failure under the stable model semantics, first for stratified programmes (Cucala *et al.* 2021) and subsequently for the general case (Wałęga *et al.* 2021).

There have been numerous alternative proposals for extending Datalog with temporal constructs. For example, Datalog$_{1S}$ (Chomicki and Imielinski 1988) is a prominent early extension, in which predicates are allowed to contain terms of an additional temporal sort and a single successor function symbol over this sort. A number of recent temporal extensions of Datalog feature operators from LTL (Artale *et al.* 2015) as well as from the Halpern-Shoham logic of intervals (Kontchakov *et al.* 2016). Extensions of DatalogMTL with non-monotonic negation are closely related to temporal extensions of answer set programming (ASP) (Aguado *et al.* 2021). Particularly relevant is a recently introduced extension of ASP with MTL operators (Cabalar *et al.* 2020) as well as the LARS framework (Beck *et al.* 2018), which combines ASP and MTL operators for reasoning over data streams. It is worth observing, however, that all these temporal extensions of ASP are interpreted over the integer timeline. Finally, operators from MTL have also been exploited in temporal extensions of description logics (Gutiérrez-Basulto *et al.* 2016; Artale and Franconi 1998; Baader *et al.* 2017; Thost 2018).

The first system with DatalogMTL support was an extension of the Ontop platform (Kalaycı *et al.* 2018) restricted to non-recursive programmes and based on reasoning via rewriting into SQL. Very recently, the Vadalog reasoning system (Bellomarini *et al.* 2018) has also been extended with DatalogMTL support (Bellomarini *et al.* 2022); the algorithm implemented by this extension is, however, non-terminating in general. Fragments of DatalogMTL for which materialisation-based reasoning is guaranteed to terminate have been identified and studied by Wałęga *et al.* (2023); these fragments, however, impose restrictions which effectively disallow programmes expressing "recursion through time." Finally, there is a plethora of reasoners available for solving satisfiability checking problems in LTL, with techniques involving reduction to model checking, tableau systems, and automata-based techniques. Prominent recent examples of highly-optimised LTL reasoners include nuXmv (Cavada *et al.* 2014) and BLACK (Geatti *et al.* 2019), where the latter was our LTL reasoner of choice in the experiments reported in Section 6.

3 Preliminaries

In this section, we recapitulate the syntax, semantics, and key reasoning problems in DatalogMTL.

3.1 Syntax

We consider a *timeline* consisting of ordered rational numbers, denoted by $\mathbb{Q}$, which we also call *time points*. A *(rational) interval* ρ is a set of rational numbers that is either empty or such that

- for all $t_1, t_2, t_3 \in \mathbb{Q}$ satisfying $t_1 < t_2 < t_3$ and $t_1, t_3 \in \rho$, it must be the case that $t_2 \in \rho$, and
- the greatest lower bound ρ^- and the least upper bound ρ^+ of ρ belong to $\mathbb{Q} \cup \{-\infty, \infty\}$.[2]

The bounds ρ^- and ρ^+ are called the *left* and the *right endpoints* of ρ, respectively. An interval is *punctual* if it contains exactly one time point. Intervals ρ_1 and ρ_2 are *union-compatible* if $\rho = \rho_1 \cup \rho_2$ is also an interval. We represent a non-empty interval ρ using the standard notation $\langle \rho^-, \rho^+ \rangle$, where the *left bracket* "$\langle$" is either "[" or "(", and the *right bracket* $\rangle$ is either "]" or ")". We assume that each rational endpoint is given as a (not necessary reduced) fraction with an integer numerator and a positive integer denominator, both encoded in binary. As usual, the brackets "[" and "]" indicate that the corresponding endpoints are included in the interval, whereas "(" and ")" indicate that they are not included. Observe that every interval has multiple representations due to possibly non-reduced fractions. Each representation, however, uniquely determines an interval and so, if it is clear from the context, we will abuse notation and identify an interval representation with the unique interval it represents.

We consider a *signature* consisting of pairwise disjoint countable sets of constants, variables, and predicates with non-negative integer arities. As usual, a term is either a constant or a variable. A *relational atom* is an expression of the form $P(\mathbf{s})$, with P a predicate and $\mathbf{s}$ a tuple of terms whose length matches the arity of P. A *metric atom* is

[2] By this restriction, for example, the set of all rational numbers between 0 and π is not an interval, as it has no least upper bound in $\mathbb{Q}$. Also note that in contrast to Brandt *et al.* (2018), we do not require finite endpoints of intervals to be dyadic rational numbers.

an expression given by the following grammar, where $P(\mathbf{s})$ is a relational atom, $\top$ and $\bot$ are logical truth and falsehood respectively, and $\diamondminus$, $\diamondplus$, $\boxminus$, $\boxplus$, $\mathcal{S}$, and $\mathcal{U}$ are MTL operators that can be indexed with any intervals ρ containing only non-negative rationals:

$$M ::= \top \mid \bot \mid P(\mathbf{s}) \mid \diamondminus_\rho M \mid \diamondplus_\rho M \mid \boxminus_\rho M \mid \boxplus_\rho M \mid M \mathcal{S}_\rho M \mid M \mathcal{U}_\rho M.$$

We refer to $\diamondminus$, $\boxminus$, and $\mathcal{S}$ as *past operators* and we refer to $\diamondplus$, $\boxplus$, and $\mathcal{U}$ as *future operators*. A *rule* is an expression of the form

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

with each M_i a metric atom, and M' generated by the following grammar:

$$M' ::= \bot \mid P(\mathbf{s}) \mid \boxminus_\rho M' \mid \boxplus_\rho M'.$$

The conjunction $M_1 \wedge \cdots \wedge M_n$ in Expression (1) is the rule's *body* and M' is the rule's *head*. A rule mentioning $\bot$ in the head is referred to as a $\bot$-*rule*. A rule is *forward-propagating* (respectively *backwards propagating*) if it does not mention $\top$ or $\bot$, mentions only past (respectively, future) operators in the body, and only future (respectively, past) operators in the head. A rule is *safe* if each variable in its head also occurs in the body, and this occurrence is not in a left operand of $\mathcal{S}$ or $\mathcal{U}$; for instance, a rule such as $P(x) \leftarrow Q(x)\mathcal{S}_{[0,0]}\top$ is not safe; in fact, it can be equivalently rewritten as $P(x) \leftarrow \top$ (see the following section on DatalogMTL semantics). A *programme* is a finite set of safe rules; it is forward or backwards propagating if so are all its rules.

An expression is *ground* if it mentions no variables. A *fact* is an expression of the form $M@\rho$ with M a ground relational atom and ρ an interval; a *dataset* is a finite set of facts. The *coalescing* of facts $M@\rho_1$ and $M@\rho_2$ with union-compatible intervals ρ_1 and ρ_2 is the fact $M@(\rho_1 \cup \rho_2)$. The *coalescing* of a dataset $\mathcal{D}$, denoted by $\mathsf{coal}(\mathcal{D})$, is the unique dataset obtained by iteratively coalescing facts in $\mathcal{D}$ until no more facts can be coalesced. The *grounding* $\mathsf{ground}(\Pi, \mathcal{D})$ of a programme Π with respect to a dataset $\mathcal{D}$ is the set of all ground rules that can be obtained by assigning constants in Π or $\mathcal{D}$ to variables in Π.

The *dependency graph* of a programme Π is the directed graph G_Π, with a vertex v_P for each predicate P in Π and an edge (v_Q, v_R) whenever there is a rule in Π mentioning Q in the body and R in the head. Programme Π is *recursive* if G_Π has a cycle. A predicate P is *recursive* in Π if G_Π has a path ending in v_P and including a cycle (the path can be a self-loop); otherwise, it is *non-recursive*. A metric atom is *non-recursive* in Π if so are all its predicates; otherwise it is *recursive*. The (non-)recursive fragment of Π is the subset of rules in Π with (non-)recursive atoms in heads. Furthermore, for a predicate P, a rule r is P-*relevant* in Π if there exists a rule r' in Π mentioning P or $\bot$ in the head and a path in G_Π starting from a vertex representing the predicate in the head of r and ending in a vertex representing some predicate from the body of r'.

3.2 Semantics and reasoning problems

An *interpretation* $\mathfrak{I}$ is a function which assigns to each time point t a set of ground relational atoms; if an atom $P(\mathbf{c})$ belongs to this set, we say that $P(\mathbf{c})$ is *satisfied* at t in $\mathfrak{I}$ and we write $\mathfrak{I}, t \models P(\mathbf{c})$. This extends to arbitrary metric atoms as shown in Table 1.

Table 1. *Semantics of ground metric atoms*

$\mathfrak{I}, t \models \top$		for each t
$\mathfrak{I}, t \models \bot$		for no t
$\mathfrak{I}, t \models \diamondminus_\rho M$	iff	$\mathfrak{I}, t' \models M$ for some t' with $t - t' \in \rho$
$\mathfrak{I}, t \models \diamondplus_\rho M$	iff	$\mathfrak{I}, t' \models M$ for some t' with $t' - t \in \rho$
$\mathfrak{I}, t \models \boxminus_\rho M$	iff	$\mathfrak{I}, t' \models M$ for all t' with $t - t' \in \rho$
$\mathfrak{I}, t \models \boxplus_\rho M$	iff	$\mathfrak{I}, t' \models M$ for all t' with $t' - t \in \rho$
$\mathfrak{I}, t \models M_1 \mathscr{S}_\rho M_2$	iff	$\mathfrak{I}, t' \models M_2$ for some t' with $t - t' \in \rho$ and $\mathfrak{I}, t'' \models M_1$ for all $t'' \in (t', t)$
$\mathfrak{I}, t \models M_1 \mathscr{U}_\rho M_2$	iff	$\mathfrak{I}, t' \models M_2$ for some t' with $t' - t \in \rho$ and $\mathfrak{I}, t'' \models M_1$ for all $t'' \in (t, t')$

An interpretation $\mathfrak{I}$ satisfies a fact $M@\rho$ if $\mathfrak{I}, t \models M$ for all $t \in \rho$. Interpretation $\mathfrak{I}$ satisfies a ground rule r if, whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $\mathfrak{I}$ also satisfies the head of r at t. Interpretation $\mathfrak{I}$ satisfies a (non-ground) rule r if it satisfies each ground instance of r. Interpretation $\mathfrak{I}$ is a *model* of a programme Π if it satisfies each rule in Π, and it is a *model* of a dataset $\mathscr{D}$ if it satisfies each fact in $\mathscr{D}$. Each dataset $\mathscr{D}$ has a unique least model $\mathfrak{I}_{\mathscr{D}}$, and we say that dataset $\mathscr{D}$ *represents* interpretation $\mathfrak{I}_{\mathscr{D}}$. Programme Π and dataset $\mathscr{D}$ are *consistent* if they have a model, and they *entail* a fact $M@\rho$ if each model of both Π and $\mathscr{D}$ is a model of $M@\rho$.

Consistency checking is the problem of checking whether a given programme and a dataset admit a common model. *Fact entailment* is the problem of checking whether a programme and a dataset entail a given relational fact. Consistency checking and fact entailment in DatalogMTL reduce to the complements of each other. Specifically, to check whether a programme Π and a dataset $\mathscr{D}$ are inconsistent, it suffices to check whether they entail fact $P@0$, for P a fresh proposition occurring neither in Π nor in $\mathscr{D}$. In turn, to check whether Π and $\mathscr{D}$ entail a fact $P(\mathbf{c})@\rho$, it suffices to check whether the following programme and dataset are inconsistent, with P' a fresh predicate of the same arity as P, $\mathbf{x}$ a tuple of distinct variables, and t an arbitrary time point belonging to the interval ρ:

$$\Pi'' = \Pi \cup \{\bot \leftarrow P'(\mathbf{x}) \wedge \boxminus_{\rho_1} P(\mathbf{x}) \wedge \boxplus_{\rho_2} P(\mathbf{x})\}, \qquad \mathscr{D}'' = \mathscr{D} \cup \{P'(\mathbf{c})@t\}.$$

Intervals ρ_1 and ρ_2 are constructed using ρ and t; for example, if $\rho = [t_1, t_2)$, then $\rho_1 = [0, t - t_1]$ and $\rho_2 = [0, t_2 - t)$, whereas if $t_2 = \infty$, then $t_2 - t$ stands for ∞.

3.3 Fixpoint characterisation and the canonical interpretation

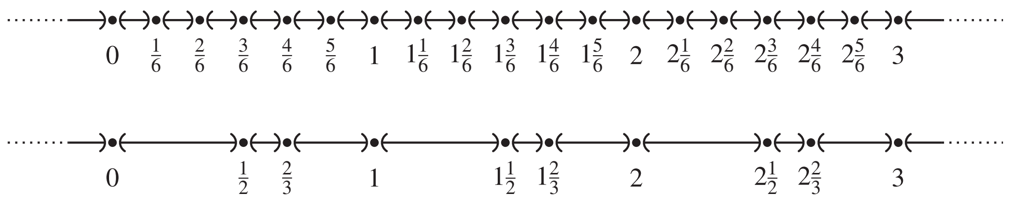
Each pair of a consistent programme Π and a dataset $\mathscr{D}$ admits a unique least model, which we refer to as their *canonical interpretation* $\mathfrak{C}_{\Pi,\mathscr{D}}$ (Brandt *et al.* 2018). As in plain Datalog, we can construct this interpretation by applying rules of Π to $\mathscr{D}$ in a forward-chaining manner until a fixpoint is reached. Unlike plain Datalog, however, the fixpoint in DatalogMTL may only be reachable after infinitely many materialisation steps; for example, given a fact $P@0$, the rule $\boxplus_1 P \leftarrow P$ propagates P to all positive integers, which requires ω materialisation steps.

Materialisation is performed using the *immediate consequence operator* T_Π; intuitively, $T_\Pi(\mathfrak{I})$ can be seen as the interpretation obtained from $\mathfrak{I}$ by adding all relational facts that can be derived by applying once all rules in Π which are not $\bot$-rules in all possible ways to $\mathfrak{I}$. Formally, we define T_Π, for a programme Π, as the operator mapping each interpretation $\mathfrak{I}$ to the least interpretation containing $\mathfrak{I}$ and satisfying the following property for each ground instance r of a rule in Π that is not a $\bot$-rule: whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $T_\Pi(\mathfrak{I})$ satisfies the head of r at t. Subsequent applications of T_Π to the (least) model $\mathfrak{I}_\mathscr{D}$ of $\mathscr{D}$ define the following sequence of interpretations, for all ordinals α:

$$\begin{aligned} T_\Pi^0(\mathfrak{I}_\mathscr{D}) &= \mathfrak{I}_\mathscr{D}, & \\ T_\Pi^\alpha(\mathfrak{I}_\mathscr{D}) &= T_\Pi\left(T_\Pi^{\alpha-1}(\mathfrak{I}_\mathscr{D})\right), & &\text{for } \alpha \text{ a successor ordinal,} \\ T_\Pi^\alpha(\mathfrak{I}_\mathscr{D}) &= \bigcup_{\beta<\alpha} T_\Pi^\beta(\mathfrak{I}_\mathscr{D}), & &\text{for } \alpha \text{ a limit ordinal.} \end{aligned}$$

The canonical interpretation is obtained after at most ω_1 (i.e. the first uncountable ordinal) applications of T_Π; that is, $\mathfrak{C}_{\Pi,\mathscr{D}} = T_\Pi^{\omega_1}(\mathfrak{I}_\mathscr{D})$ (Brandt *et al.* 2017); in turn, if Π and $\mathscr{D}$ are consistent then $\mathfrak{C}_{\Pi,\mathscr{D}}$ is the least common model of Π and $\mathscr{D}$. Furthermore, $\bot$-rules can be treated as constraints, and so Π and $\mathscr{D}$ are consistent if and only if $\mathfrak{C}_{\Pi,\mathscr{D}}$ is a model of all $\bot$-rules in Π.

Although the timeline in DatalogMTL is dense, it can be divided into regularly distributed intervals which are uniform in the sense that, in the canonical interpretation, the same relational atoms hold in all time points belonging to the same interval. This observation was first exploited to partition the rational timeline, for a programme Π and dataset $\mathscr{D}$, into punctual intervals $[i \cdot d, i \cdot d]$ and open intervals $((i-1) \cdot d, i \cdot d)$, for each $i \in \mathbb{Z}$, where d is the greatest common divisor (gcd) of the numbers occurring as interval endpoints in Π and $\mathscr{D}$ (Brandt *et al.* 2018). Later, an alternative partitioning of the timeline was proposed (Wałęga *et al.* 2019a), where punctual intervals of the form $[i \cdot d, i \cdot d]$ are replaced with punctual intervals $[t + i \cdot d', t + i \cdot d']$, for all rational numbers t in $\mathscr{D}$, $i \in \mathbb{Z}$, and d' the gcd of numbers occurring in Π; in turn, open intervals of the form $((i-1) \cdot d, i \cdot d)$ were replaced with open intervals located between the new punctual intervals. Below we present example partitionings of the timeline into intervals stemming from both discretisation methods, for the case where the only rationals occurring in $\mathscr{D}$ are $\frac{1}{2}$ and $\frac{2}{3}$ and the gcd of Π is 1 (therefore, $d = \frac{1}{6}$ and $d' = 1$). The second partitioning has the advantage that the gcd is computed independently from $\mathscr{D}$; this was exploited to devise reasoning techniques with a better computational behaviour in data size (Wałęga *et al.* 2019a).



Procedure 1: Naïve(Π, $\mathscr{D}$, $P(\mathbf{c})@\rho$)

```
  Input: A program $\Pi$, a dataset $\mathscr{D}$, and a fact $P(\mathbf{c})@\rho$
  Output: Either inconst, or a pair of a truth value and a dataset
1 Initialise $\mathscr{N}$ to $\emptyset$ and $\mathscr{D}'$ to $\mathscr{D}$;
2 loop
3     $\mathscr{N} := \Pi[\mathscr{D}']$;                                   // derive new facts
4     $\mathscr{C} := \mathsf{coal}(\mathscr{D}' \cup \mathscr{N})$;        // add new facts to partial materialisation
5     if the least model of $\mathscr{D}'$ does not satisfy some $\bot$-rule in $\Pi$ then return inconst;
        // inconsistency is derived
6     if $\mathscr{D}' \models P(\mathbf{c})@\rho$ then return $(\mathsf{true}, \mathscr{D}')$;   // input fact is derived
7     if $\mathscr{C} = \mathscr{D}'$ then return $(\mathsf{false}, \mathscr{D}')$;          // fixpoint is reached
8     $\mathscr{D}' := \mathscr{C}$;
```

4 Reasoning techniques for DatalogMTL

In this section we recapitulate different techniques available for reasoning in DatalogMTL, which we take as a starting point for the development of our approach; we present a variant of the naïve materialisation procedure (Wałęga *et al.* 2023,2021), the reduction of DatalogMTL reasoning to LTL satisfiability proposed by Brandt *et al.* (2018), and the automata-based procedure exploited by Wałęga *et al.* 2019a to establish optimal complexity bounds for reasoning.

4.1 Naïve materialisation

The fixpoint characterisation from Section 3.3 suggests the naïve materialisation-based approach specified in Procedure 1. For a programme Π, a dataset $\mathscr{D}$, and a fact $P(\mathbf{c})@\rho$, the procedure applies the immediate consequence operator T_Π to $\mathfrak{I}_\mathscr{D}$ (Lines 3, 4, and 8) until one of the following holds:

- the partial materialisation of Π and $\mathscr{D}$ yields inconsistency due to satisfaction of the body of some $\bot$-rule, in which case inconst is returned (Line 5),
- the partial materialisation entails the input fact $P(\mathbf{c})@\rho$, in which case the truth value true and the obtained partial materialisation are returned (Line 6), or
- the application of T_Π reaches a fixpoint and the obtained materialisation does not satisfy $P(\mathbf{c})@\rho$ nor yields an inconsistency, in which case the value false and the full materialisation are returned (Line 8).

Example 4.1.
As a running example, consider the input fact $R_1(c_1, c_2)@[4, 4]$, *the dataset*

$$\mathscr{D}_{\mathsf{ex}} = \{R_1(c_1, c_2)@[0, 1], \quad R_2(c_1, c_2)@[1, 2], \quad R_3(c_2, c_3)@[2, 3], \quad R_5(c_2)@[0, 1]\},$$

and the programme Π_{ex} *consisting of the following rules:*

$$R_1(x, y) \leftarrow \diamondminus_{[1,1]} R_1(x, y), \qquad (\mathrm{r}_1)$$

$$\boxplus_{[1,1]} R_5(y) \leftarrow R_2(x,y) \wedge \boxplus_{[1,2]} R_3(y,z), \tag{r$_2$}$$

$$R_4(x) \leftarrow \diamondminus_{[0,1]} R_5(x), \tag{r$_3$}$$

$$R_6(y) \leftarrow R_1(x,y) \wedge \boxminus_{[0,2]} R_4(y) \wedge R_5(y). \tag{r$_4$}$$

In naïve materialisation, rules are applied by first identifying the facts that can ground the rule body, and then determining the maximal intervals for which all the ground body atoms hold simultaneously. For instance, Rule r_2 in Example 4.1 is applied to $\mathscr{D}_{\mathsf{ex}}$ by matching $R_2(c_1,c_2)$ and $R_3(c_2,c_3)$ to the rule's body atoms, and determining that $[1,1]$ is the maximal interval in which the body atoms $R_2(c_1,c_2)$ and $\boxplus_{[1,2]} R_3(c_2,c_3)$ hold together. As a result, the head $\boxplus_{[1,1]} R_5(c_2)@[1,1]$ holds, and so the fact $R_5(c_2)@[2,2]$ is derived. We formalise it below.

Definition 4.2.
Let r be a rule of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$ for some $n \geq 1$, which is not a $\bot$-rule, and let $\mathscr{D}$ be a dataset. The set of instances for r and $\mathscr{D}$ is defined as follows:

$$\begin{aligned}\mathsf{inst}_r[\mathscr{D}] = \{(M_1\sigma@\rho_1, \ldots, M_n\sigma@\rho_n) \mid \sigma \text{ is a substitution and, for each } i \in \{1,\ldots,n\}\\ \rho_i \text{ is a subset-maximal interval such that } \mathscr{D} \models M_i\sigma@\rho_i\}.\end{aligned}$$

The set $r[\mathscr{D}]$ of facts derived by r from $\mathscr{D}$ is defined as follows:

$$\begin{aligned}r[\mathscr{D}] = \{M\sigma@\rho \mid \sigma \text{ is a substitution, } M \text{ is the relational atom in } M'\sigma, \text{ and}\\ \text{there is } (M_1\sigma@\rho_1,\ldots,M_n\sigma@\rho_n) \in \mathsf{inst}_r[\mathscr{D}] \text{ such that } \rho \text{ is the unique}\\ \text{subset-maximal interval satisfying } M'\sigma@(\rho_1 \cap \ldots \cap \rho_n) \models M\sigma@\rho\}.\end{aligned} \tag{2}$$

The set of facts derived from $\mathscr{D}$ by one-step application of Π is

$$\Pi[\mathscr{D}] = \bigcup_{r \in \Pi} r[\mathscr{D}]. \tag{3}$$

By Definition 4.2, the set $\mathscr{N}$ of facts derived by Procedure 1 in Line 3 in the first iteration of the loop on our running example is $\Pi_{\mathsf{ex}}[\mathscr{D}_{\mathsf{ex}}] = \{R_1(c_1,c_2)@[1,2], R_4(c_2)@[0,2], R_5(c_2)@[2,2]\}$. The partial materialisation $\mathscr{D}^1_{\mathsf{ex}}$ that will be passed on to the next materialisation step is obtained in Line 4 as $\mathscr{D}^1_{\mathsf{ex}} = \mathsf{coal}(\mathscr{D}_{\mathsf{ex}} \cup \Pi_{\mathsf{ex}}[\mathscr{D}_{\mathsf{ex}}])$ where, as we described in Section 3.1, $\mathsf{coal}(\mathscr{D})$ is the unique dataset obtained by exhaustively coalescing facts in $\mathscr{D}$. In our example, $R_1(c_1,c_2)@[0,1]$ in and $R_1(c_1,c_2)@[1,2]$ will be coalesced into $R_1(c_1,c_2)@[0,2]$. Thus,

$$\begin{aligned}\mathscr{D}^1_{\mathsf{ex}} = \{R_1(c_1,c_2)@[0,2], \quad R_2(c_1,c_2)@[1,2], \quad R_3(c_2,c_3)@[2,3],\\ R_4(c_2)@[0,2], \quad R_5(c_2)@[0,1], \quad R_5(c_2)@[2,2]\}.\end{aligned}$$

In the second round, rules are applied to $\mathscr{D}^1_{\mathsf{ex}}$. The application of r_1 derives $R_1(c_1,c_2)@[1,3]$ (from $R_1(c_1,c_2)@[0,2]$) and the application of r_2 rederives a redundant fact $R_5(c_2)@[2,2]$. In contrast to the previous step, Rule r_4 can now be applied to derive the new fact $R_6(c_2)@[2,2]$. Finally, r_3 derives the new fact $R_4(c_2)@[2,3]$ and rederives $R_4(c_2)@[0,2]$. After coalescing, the second step of materialisation yields the following partial materialisation:

$$\mathscr{D}^2_{\mathsf{ex}} = \{R_1(c_1,c_2)@[0,3], \quad R_2(c_1,c_2)@[1,2], \quad R_3(c_2,c_3)@[2,3],$$
$$R_4(c_2)@[0,3], \quad R_5(c_2)@[0,1], \quad R_5(c_2)@[2,2], \quad R_6(c_2)@[2,2]\}.$$

In the third materialisation step, rules are applied to $\mathscr{D}^2_{\mathsf{ex}}$, and derive the new fact $R_1(c_1,c_2)@[1,4]$ using Rule r_1, as well as all facts which were already derived in the second materialisation step (with the only exception of $R_1(c_1,c_2)@[1,3]$). After coalescing we obtain:

$$\mathscr{D}^3_{\mathsf{ex}} = \{R_1(c_1,c_2)@[0,4], \quad R_2(c_1,c_2)@[1,2], \quad R_3(c_2,c_3)@[2,3],$$
$$R_4(c_2)@[0,3], \quad R_5(c_2)@[0,1], \quad R_5(c_2)@[2,2], \quad R_6(c_2)@[2,2]\}.$$

At this point, the algorithm detects that $\mathscr{D}^3_{\mathsf{ex}}$ entails the input fact $R_1(c_1,c_2)@[4,4]$ and stops.

Procedure 1 is both sound and complete. To show this, for each $k \in \mathbb{N}$, let $\mathscr{N}_k$ and $\mathscr{D}_k$ denote the contents of, respectively, $\mathscr{N}$ and $\mathscr{D}$ in Procedure 1 upon the completion of the kth iteration of the loop. Then we prove soundness by showing inductively on $k \in \mathbb{N}$, that $\mathfrak{I}_{\mathscr{D}_k} \subseteq T^k_\Pi(\mathfrak{I}_\mathscr{D})$.

Theorem 4.3 (Soundness).
Consider Procedure 1 running on input Π and $\mathscr{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 1, it holds that $\mathfrak{I}_{\mathscr{D}'} \subseteq T^k_\Pi(\mathfrak{I}_\mathscr{D})$.

By Theorem 4.3, if the least model $\mathfrak{I}_{\mathscr{D}'}$ of $\mathscr{D}'$ is not a model of a $\bot$-rule in Π, then the canonical interpretation $\mathfrak{C}_{\Pi,\mathscr{D}}$ is also not a model of such a rule; thus, if the algorithm returns $\mathsf{inconst}$, then Π and $\mathscr{D}$ are inconsistent. Next, to show completeness, we prove inductively that $T^k_\Pi(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}_k}$.

Theorem 4.4 (Completeness).
Consider Procedure 1 running on input Π and $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 1, it holds that $T^k_\Pi(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}'}$.

If Π and $\mathscr{D}$ are inconsistent, the canonical interpretation $\mathfrak{C}_{\Pi,\mathscr{D}}$ is not a model of some $\bot$-rule r in Π; thus, there is an ordinal α such that $T^\alpha_\Pi(\mathfrak{I}_\mathscr{D})$ is also not a model of r, in which case Theorem 4.4 ensures that the algorithm returns $\mathsf{inconst}$, as required.

Furthermore, if a fixpoint is reached without encountering an inconsistency, then our procedure ensures that the input fact is not entailed and also that we have obtained a representation of the full canonical interpretation which can be kept in memory and subsequently exploited for checking entailment of any other fact. The procedure is, however, not always terminating as reaching a fixpoint may require infinitely many rounds of rule applications.

4.2 Translation to LTL

The discretisation of the rational timeline described in Section 3.3 was exploited by Brandt *et al.* (2018) to reduce reasoning in DatalogMTL to reasoning in LTL. The reduction transforms a programme Π and a dataset $\mathscr{D}$ into an LTL formula $\varphi_{\Pi,\mathscr{D}}$ such that Π

and $\mathcal{D}$ are consistent if and only if $\varphi_{\Pi,\mathcal{D}}$ is LTL-satisfiable. Here, LTL is a propositional modal logic interpreted over the integer time points and equipped with temporal operators $\bigcirc_P$ for *at the previous time point*, $\Box_P$ for *always in the past*, $\mathcal{S}$ for *since*, $\bigcirc_F$ for *at the next time point*, $\Box_F$ for *always in the future*, and $\mathcal{U}$ for *until*. An LTL formula φ is *satisfiable* if it holds at time point 0 in some LTL model.

Since, in contrast to DatalogMTL, the language of LTL is propositional, the first step in the translation is to ground Π with all constants occurring in Π or $\mathcal{D}$. Then, every relational atom $P(\mathbf{c})$ occurring in the grounding of Π with constants from Π and $\mathcal{D}$ is translated into a propositional symbol $P^{\mathbf{c}}$. Moreover, the MTL operators occurring in Π are rewritten using LTL operators. Both the grounding of the input programme Π and the translation of the MTL operators (with binary-encoded numbers) into sequences of LTL operators yield an exponential blow-up. For instance, assume that Π mentions an atom $\boxminus_{[0,60)}A(x)$, both Π and $\mathcal{D}$ mention constants $c_1, \dots, c_n$, and that interval $(0, 60)$ contains m intervals after discretising the timeline. Then $\boxplus_{(0,60)}A(x)$ is translated to an LTL formula containing n conjuncts (one conjunct for each c_i), each of the form $\bigcirc_F A^{c_i} \wedge \bigcirc_F \bigcirc_F A^{c_i} \wedge \cdots \wedge \underbrace{\bigcirc_F \cdots \bigcirc_F}_{m} A^{c_i}$. Consequently, $\varphi_{\Pi,\mathcal{D}}$ is exponentially large. Since satisfiability checking in LTL is PSpace-complete, this approach provides a (worst-case optimal) ExpSpace reasoning procedure for DatalogMTL.

4.3 Automata-based reasoning

An alternative approach to exploit the time discretisation of DatalogMTL is to directly apply automata-based techniques, without the need of constructing an LTL formula. Such automata-based constructions are well-studied in the context of LTL, where checking satisfiability of a formula φ reduces to checking non-emptiness of a generalised non-deterministic B FC;chi automaton where states are sets of formulas relevant to φ, the alphabet consists of sets of propositions, and the transition relation and accepting conditions ensure that words accepted by the automaton are exactly the models of φ (Baier and Katoen 2008).

This technique can be adapted to the DatalogMTL setting (Wałęga *et al.* 2021); each state now represents ground metric atoms holding at all time points within a fragment of the timeline called a *window*. Since the timeline can be discretised in DatalogMTL, each window can be finitely represented as a sequence consisting of sets of metric atoms which hold in the consecutive intervals from the window. Additionally, for such a sequence to be a state of the automaton, it is required that the involved metric atoms are *locally consistent*; for example, if $\boxplus_{[0,\infty)}A$ – which states that A holds always in the future – holds in some interval ρ, then $\boxplus_{[0,\infty)}A$ needs to hold also in all the intervals in the window which are to the right of ρ in the timeline. The remaining components of the automaton are defined analogously to the case of LTL.

Consistency checking in DatalogMTL reduces to checking non-emptiness of (pairs of) automata, and this reduction provides a PSpace upper bound for data complexity (Wałęga *et al.* 2021). In particular, automata states are polynomially large in the size of the dataset since windows can be chosen so that the number of intervals in each window is polynomially large. Moreover, the number of ground metric atoms that can hold in each of these intervals is also polynomially bounded. Thus, each state is polynomially representable and non-emptiness of the automata can be checked with the standard "on-the-fly" approach (Baier and Katoen 2008) in PSpace.

5 Our practical reasoning algorithm

In this section we propose a scalable approach for deciding both consistency and fact entailment in DatalogMTL. The key elements of our approach are as follows:

- an optimised materialisation procedure using a seminaïve strategy which efficiently applies the immediate consequence operator while minimising repeated inferences,
- a realisation of the automata-based reasoning approach, and
- an effective way of combining materialisation with automata-based reasoning that aims at resorting to materialisation-based reasoning whenever possible.

In the remainder of this section we discuss each of the key components of our approach in detail. For convenience of presentation, and without loss of generality, we will assume that input programmes do not contain rules whose body is vacuously satisfied (i.e. satisfied in the empty interpretation). Observe that each such rule can be formulated as a fact and added to the input dataset instead; for example, rule $P \leftarrow \top$ is equivalent to a fact $P@(-\infty, \infty)$.

5.1 Seminaïve materialisation

In this section we present a seminaïve materialisation procedure for DatalogMTL. Analogously to the classical seminaïve algorithm for plain Datalog (Abiteboul *et al.* 1995), the main idea behind our procedure is to keep track of newly derived facts in each materialisation step by storing them in a set Δ, and to make sure that each rule application in the following materialisation step involve at least one fact in Δ. In this way, the procedure will consider each instance (for a rule and dataset, as we introduced in Definition 4.2) *at most once* throughout its entire execution, and so, such a procedure is said to enjoy the *non-repetition* property. The same fact, however, can still be derived multiple times by *different* instances; this type of redundancy is difficult to prevent and is not addressed by the standard seminaïve strategy.

Our aim is to lift the seminaïve rule evaluation strategy to the setting of DatalogMTL. As we have seen in Section 4.1, a rule instance can be considered multiple times in the naïve materialisation procedure; for example, the instance $(\diamondminus_{[0,1]} R_5(c_2)@[0,2])$ of Rule r_3 in Example 4.1 is considered by the naïve materialisation procedure in Section 4.1 both in the first and second materialisation steps to derive $R_4(c_2)@[0,2]$ since the naïve procedure cannot detect that the fact $R_5(c_2)@[0,1]$ used to instantiate r_3 in the second step had previously been used to instantiate r_2. Preventing such redundant computations, however, involves certain challenges. First, by including in Δ just newly derived facts as in Datalog, we may overlook information obtained by coalescing newly derived facts with previously derived ones. Second, restricting application to relevant rule instances requires consideration of the semantics of metric operators in rule bodies.

Procedure 2: Seminaïve $\Pi, \mathscr{D}, P(\mathbf{c})@\rho)$

```
    Input: A program $\Pi$, a dataset $\mathscr{D}$, and a fact $P(\mathbf{c})@\rho$
    Output: Either inconst, or a pair of a truth value and a dataset
  1 Initialise $\mathscr{N}$ to $\emptyset$, and both $\varnothing$and $\mathscr{D}'$ to $\mathscr{D}$;
  2 loop
  3     $\mathscr{N} := \Pi[\mathscr{D}' \vdots \varnothing]$;
  4     $\mathscr{C} := \mathsf{coal}(\mathscr{D}' \cup \mathscr{N})$;
  5     $\varnothing := \{M@\rho \in \mathscr{C} \mid M@\rho$ entails some fact in $\mathscr{N}$ which is not entailed by $\mathscr{D}'\}$;
  6     if the least model of $\mathscr{D}'$ does not satisfy some $\bot$-rule in $\Pi$ then return inconst;
  7     if $\mathscr{D}' \models P(\mathbf{c})@\rho$ then return $(\mathsf{true}, \mathscr{D}')$;
  8     if $\varnothing =$ / then return $(\mathsf{false}, \mathscr{D}')$;
  9     $\mathscr{D}' := \mathscr{C}$;
```

Procedure 2 extends the seminaïve strategy to the setting of DatalogMTL while overcoming the aforementioned difficulties. Analogously to the naïve approach, each iteration of the main loop captures a single materialisation step consisting of a round of rule applications (Line 3) followed by the coalescing of relevant facts (Line 4); as before, dataset $\mathscr{D}'$ stores the partial materialisation resulting from each iteration and is initialised as the input dataset, whereas the dataset $\mathscr{N}$ stores the facts obtained as a result of rule application and is initialised as empty. Following the rationale behind the seminaïve strategy for Datalog, newly derived information in each materialisation step is now stored as a dataset Δ, which is initialised as the input dataset $\mathscr{D}$ and which is suitably maintained in each iteration (Line 5); furthermore, Procedure 2 ensures in Line 3 that only rule instances for which it is essential to involve facts from Δ (as formalised in the following definition) are taken into account during rule application.

Definition 5.1.
Let r be a rule of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, for some $n \geq 1$, and let $\mathscr{D}$ and Δ be datasets. The set of instances for r and $\mathscr{D}$ relative to Δ is defined as follows:

$$\mathsf{inst}_r[\mathscr{D} \vdots \Delta] = \{(M_1\sigma@\rho_1, \ldots, M_n\sigma@\rho_n) \in \mathsf{inst}_r[\mathscr{D}] \mid \mathscr{D} \setminus \Delta \not\models M_i\sigma@\rho_i,$$
$$\text{for some } i \in \{1, \ldots, n\}\}.$$

The set $r[\mathscr{D} \vdots \Delta]$ of facts derived by r from $\mathscr{D}$ relative to Δ is defined analogously to $r[\mathscr{D}]$ in Definition 4.2, with the exception that $\mathsf{inst}_r[\mathscr{D}]$ is replaced with $\mathsf{inst}_r[\mathscr{D}] \vdots \Delta$ in Expression (2). Finally, the set $\Pi[\mathscr{D} \vdots \Delta]$ of facts derived from $\mathscr{D}$ by one-step seminaïve application of Π is defined as $\Pi[\mathscr{D}]$ in Expression (3), by replacing $r[\mathscr{D}]$ with $r[\mathscr{D} \vdots \Delta]$.

In each materialisation step, Procedure 2 exploits Definition 5.1 to identify as relevant those rule instances with some "new" element (i.e. one that cannot be entailed without the facts in Δ). The facts derived by such relevant rule instances in each iteration are stored in set $\mathscr{N}$ (Line 3).

As in the naïve approach, rule application is followed by a coalescing step where the partial materialisation is updated with the facts derived from rule application (Line 4). In contrast to the naïve approach, however, Procedure 2 needs to maintain set Δ to ensure that it captures only new facts. This is achieved in Line 5, where a fact in the updated partial materialisation is considered new if it entails a fact in $\mathscr{N}$ that was not already entailed by the previous partial materialisation. The procedure terminates in Line 6 if an inconsistency has been derived, in Line 7 if the input fact has been derived, or in Line 8 if Δ is empty. Otherwise, the procedure carries over the updated partial materialisation and the set of newly derived facts to the next materialisation step.

We next illustrate the application of the procedure to $\mathscr{D}_{\mathsf{ex}}$ and Π_{ex} and our example fact $R_1(c_1, c_2)@4$ from Example 4.1. In the first materialisation step, all input facts are considered as newly derived (i.e. $\Delta = \mathscr{D}$) and hence $\mathscr{N} = \Pi[\mathscr{D}' \,\vdots\, \Delta] = \Pi[\mathscr{D}']$ and the result of coalescing coincides with the partial materialisation computed by the naïve procedure (i.e. $\mathscr{C} = \mathscr{D}^1_{\mathsf{ex}}$). Then, the procedure identifies as new all facts in $\mathscr{D}^1_{\mathsf{ex}}$ which are not entailed by $\mathscr{D}_{\mathsf{ex}}$, namely:

$$\Delta = \{R_1(c_1, c_2)@[0,2], \quad R_4(c_2)@[0,2], \quad R_5(c_2)@[2,2]\}.$$

In the second step, rule evaluation is performed in Line 3; since Δ is used for this, the procedure no longer considers the redundant instance $(R_2(c_1, c_2)@[0,2], \boxplus_{[1,2]} R_3(c_2, c_3)@[1,2])$ of r_2, since $\boxplus_{[1,2]} R_3(c_2, c_3)@[1,2]$ is entailed already by $\mathscr{D}^1_{\mathsf{ex}} \setminus \Delta$. Similarly, the redundant instance $(\diamondplus_{[0,1]} R_5(c_2)@[0,2])$ of Rule r_3 is not considered, as $\mathscr{D}^1_{\mathsf{ex}} \setminus \Delta \models \diamondplus_{[0,1]} R_5(c_2)@[0,2]$. In contrast, all non-redundant facts derived by the naïve strategy are also derived by the seminaïve procedure and after coalescing dataset $\mathscr{C} = \mathscr{D}^2_{\mathsf{ex}}$. The set Δ is now updated as follows:

$$\Delta = \{R_1(c_2, c_2)@[0,3], \qquad R_4(c_2)@[0,3], \qquad R_6(c_2)@[2,2]\}.$$

In particular, note that Δ contains the coalesced fact $R_4(c_2)@[0,3]$ instead of fact $R_4(c_2)@[2,3]$ derived from rule application. Datasets Δ and $\mathscr{D}' = \mathscr{D}^2_{\mathsf{ex}}$ are passed on to the third materialisation step, where all redundant computations identified in Section 4.1 are avoided with the only exception of fact $R_6(c_2)@[2,2]$, which is re-derived using the instance of r_4 consisting of $R_1(c_2, c_2)@[0,3]$, $\boxminus_{[0,2]} R_4(c_2)@[2,3]$, and $R_5(c_2)@[2,2]$. Note that this is a new instance which was not used in previous iterations, and hence it does not violate the non-repetition property. Note also that, as with the naïve strategy, our seminaïve procedure terminates on our running example in this materialisation step since input fact $R_1(c_1, c_2)@[4,4]$ has been derived.

We conclude this section by establishing correctness of our procedure, similarly as we did in Theorems 4.3 and 4.4 for the naïve strategy from Procedure 1.

Theorem 5.2 (Soundness).
Consider Procedure 2 running on input Π and $\mathscr{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 2, it holds that $\mathfrak{I}_{\mathscr{D}'} \subseteq T^k_\Pi(\mathfrak{I}_{\mathscr{D}})$.

Completeness is proved by induction on the number k of iterations of the main loop. In particular, we show that if $T^{k+1}_\Pi(\mathfrak{I}_{\mathscr{D}})$ satisfies a new fact $M@t$, then there must be a rule r and an instance in $\mathsf{inst}_r[\mathscr{D}_k \,\vdots\, \Delta_k]$ witnessing the derivation of $M@t$. Note that since $\mathsf{inst}_r[\mathscr{D}_k \,\vdots\, \Delta_k] \subseteq \mathsf{inst}_r[\mathscr{D}_k]$, it requires strengthening the argument from the proof of

Theorem 4.4, where it was sufficient to show that such an instance is in $\mathsf{inst}_r[\mathscr{D}_k]$. This will imply that each fact satisfied by $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D})$ is derived in the $k+1$st iteration of our procedure.

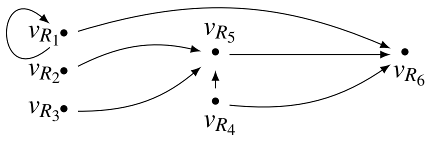
Theorem 5.3 (Completeness).
Consider Procedure 2 with input programme Π and input dataset $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}'}$.

5.2 Optimised seminaïve evaluation

Although the seminaïve procedure enjoys the non-repetition property, it can still re-derive facts already obtained in previous materialisation steps; as a result, it can incur in a potentially large number of redundant computations. In particular, as discussed in Section 5.1, fact $R_6(c_2)@[2,2]$ is re-derived using Rule r_4 in the third materialisation step of our running example, and (if the procedure did not terminate) it would also be re-derived in all subsequent materialisation steps (by different instances of Rule r_4).

In this section, we present an optimised variant of our seminaïve procedure which further reduces the number of redundant computations performed during materialisation. The main idea is to disregard rules during the execution of the procedure as soon as we can be certain that their application will never derive new facts in subsequent materialisation steps. In our example, Rule r_4 can be discarded after the second materialisation step as its application will only continue to re-derive fact $R_6(c_2)@[2,2]$ in each subsequent materialisation step.

To this end, we will exploit the distinction between recursive and non-recursive predicates in a programme, as defined in Section 3. In the case of our Example 4.1, predicates R_2, R_3, R_4, and R_5 are non-recursive, whereas R_1 and R_6 are recursive, since in the dependency graph of the programme Π_{ex} – depicted below – only vertices corresponding to R_1 and R_6 can be reached by paths containing a cycle. Hence, Rules r_2 and r_3 are non-recursive in Π_{ex} since their heads mention non-recursive predicates, whereas r_1 and r_4 are recursive.



Now, in contrast to recursive predicates, for which new facts can be derived in each materialisation step, the materialisation of non-recursive predicates will be completed after linearly many materialisation steps; from then on, the rules will no longer derive any new facts involving these predicates. This observation can be exploited to optimise our seminaïve evaluation as follows. Assume that the procedure has fully materialised all non-recursive predicates in the input programme. At this point, we can safely discard all non-recursive rules; furthermore, we can also discard a recursive rule r with a non-recursive body atom M if the current partial materialisation does not entail any grounding of M (in this case, r cannot apply in further materialisation steps). An additional optimisation applies to forward-propagating programmes, where rules cannot propagate information "backwards" along the timeline; in this case, we can compute the maximal time points for which each non-recursive body atom in r may possibly hold, select the minimum t_r amongst such values, and discard r as soon as we can determine that the materialisation up to time point t_r has been fully completed.

In the case of our Example 4.1, materialisation of the non-recursive predicates R_2, R_3, R_4, and R_5 is complete after two materialisation steps. Hence, at this point we can disregard non-recursive Rules r_2 and r_3 and focus on the recursive forward-propagating Rules r_1 and r_4. Furthermore, the maximum time point at which the non-recursive predicates R_4 and R_5 from r_4 can hold are 3 and 2, respectively, and hence $t_{r_4} = 2$; as upon the completion of the second materialisation step we can be certain that the remaining body atom of r_6, namely R_1, has been materialised up to t_{r_6}, we can discard the rule r_6. In subsequent materialisation steps we can apply only Rule r_1, thus avoiding many redundant computations.

Procedure 3 implements these ideas by extending our seminaïve materialisation from Lines 3–8. In particular, in each materialisation step, the procedure checks whether all non-recursive predicates have been fully materialised (Line 9), in which case it removes all non-recursive rules from the input programme as well as all recursive rules with an unsatisfied non-recursive body atom (Lines 10–12). It also sets the flag to 1, which activates the additional optimisation for forward-propagating programmes, which is applied in Lines 13–18.

We conclude this section by establishing correctness of our procedure. We first observe that, as soon as the algorithm switches the flag to 1, we can be certain that all facts with non-recursive predicates have been fully materialised.

Lemma 5.4.
Consider Procedure 3 running on input programme Π and dataset $\mathscr{D}$. Whenever $flag = 1$, dataset $\mathscr{D}'$ satisfies all facts over a non-recursive predicate in Π that are entailed by Π and $\mathscr{D}$.

This lemma shows us that once the $flag$ is changed to 1, Procedure 3 can safely ignore all the non-recursive rules, as they can no longer be used to derive any new facts. Thus, such rules are deleted from Π in Line 12. The deletion of rules in Line 18, on the other hand, is based on a simple observation that a derivation of a new fact $M@t$ by a forward-propagating programme is based only on facts holding at time points smaller-or-equal to t. Thus, if for a forward-propagating programme Π' (condition in Line 13) it holds that $\mathfrak{I}_{\mathscr{D}'}$ and $T_{\Pi'}(\mathfrak{I}_{\mathscr{D}'})$ satisfy the same facts within $(-\infty, t_r]$ (condition in Line 18), then $\mathfrak{I}_{\mathscr{D}'}$ and $T^{\alpha}_{\Pi'}(\mathfrak{I}_{\mathscr{D}'})$ will coincide over $(-\infty, t_r]$, for each ordinal number α. By the construction, t_r is the maximum time point in which the body of rule r can hold, so if the above conditions are satisfied, we can safely delete r from the programme in Line 18. Consequently, we obtain the following result.

Procedure 3: OptimisedSeminaïve Π, $\mathscr{D}$, $P(\mathbf{c})@\rho)$)

```
Input: A program $\Pi$, a dataset $\mathscr{D}$, and a fact $P(\mathbf{c})@\rho$
Output: Either inconst, or a pair of a truth value and a dataset
1  Initialise $\mathscr{N}$ to $\emptyset$, $\Delta$ and $\mathscr{D}'$ to $\mathscr{D}$, $\Pi'$ to $\Pi$, $flag$ to 0, and $S_r$ (for each $r \in \Pi$) to the set of
     body atoms in $r$ that are non-recursive in $\Pi$;
2  loop
3      $\mathscr{N} := \Pi'[\mathscr{D}' \vdots \Delta]$;
4      $\mathscr{C} := \mathsf{coal}(\mathscr{D}' \cup \mathscr{N})$;
5      $\Delta := \{M@\rho \in \mathscr{C} \mid M@\rho$ entails some fact in $\mathscr{N}$ but which is not entailed by $\mathscr{D}'\}$;
6      if the least model of $\mathscr{D}'$ does not satisfy some $\bot$-rule in $\Pi$ then return inconst;
7      if $\mathscr{D}' \models P(\mathbf{c})@\rho$ then return $(\mathsf{true}, \mathscr{D}')$;
8      if $\Delta = \emptyset$ then return $(\mathsf{false}, \mathscr{D}')$;
9      if $flag = 0$ and the datasets $\mathscr{D}'$ and $\mathscr{C}$ satisfy the same facts with non-recursive in $\Pi'$
         predicates then
10         Set $flag$ to 1 and $\Pi'$ to the recursive fragment of $\Pi$;
11         for each $r \in \Pi'$ do
12             if there is $M \in S_r$ such that $\mathscr{D}' \not\models M\sigma@t$, for each substitution $\sigma$ and time
                 point $t$ then $\Pi' := \Pi' \setminus \{r\}$;
13     if $flag = 1$ and $\Pi'$ is forward propagating then
14         for each $r \in \Pi'$ do
15             for each $M \in S_r$ do
16                 $t^M_{max}$ := maximum right endpoint amongst all intervals $\rho$ satisfying
                     $\mathscr{D}' \models M\sigma@\rho$, for some substitution $\sigma$;
17             $t_r$ := minimum value in $\{t^M_{max} \mid M \in S_r\}$;
18             if for each $M@\rho \in \mathscr{C}$ with $\rho$ overlapping $(-\infty, t_r]$, there is $M@\rho' \in \mathscr{D}'$ with
                 $\rho \cap (-\infty, t_r] \subseteq \rho'$ then $\Pi' := \Pi' \setminus \{r\}$;
19     $\mathscr{D}' := \mathscr{C}$;
```

Lemma 5.5.
Whenever Procedure 3 removes a rule r from Π' in either Line 12 or 18, $\mathfrak{C}_{\Pi',\mathscr{C}} = \mathfrak{C}_{\Pi'\setminus\{r\},\mathscr{C}}$.

Finally, using Lemma 5.5 together with the soundness and completeness of our seminaïve evaluation (established in Theorems 5.2 and 5.3), we can show soundness and completeness of the optimised version of the procedure.

Theorem 5.6 (Soundness and Completeness).
Consider Procedure 3 with input programme Π and input dataset $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T^k_\Pi(\mathfrak{I}_\mathscr{D}) = \mathfrak{I}_{\mathscr{D}'}$.

Note that our optimisation for forward-propagating programmes in Lines 13–18 can be adopted in a straightforward way for backwards-propagating programmes, as these cases are symmetric.

Algorithm 4: Practical reasoning algorithm

```
Input: A programme $\Pi$, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\rho$
Output: Either inconst, or a pair of a truth value and a dataset
1  $\Pi' :=$ the set of $P$-relevant rules in $\Pi$;
2  $\mathcal{D}' :=$ the set of facts in $\mathcal{D}$ which mention predicates occurring in $\Pi'$;
3  Out $:=$ OptimisedSeminaïveHalt$(\Pi', \mathcal{D}', P(\mathbf{c})@\rho)$;
4  if Out $= (\Pi'', \mathcal{D}'')$ then assign $\Pi' := \Pi''$ and $\mathcal{D}' := \mathcal{D}''$
5  else return Out;
6  Run two threads in parallel:
7  Thread 1:
8      return OptimisedSeminaïve$(\Pi', \mathcal{D}', P(\mathbf{c})@\rho)$;
9  Thread 2:
10     $\Pi'', \mathcal{D}'' :=$ EntailToInconsist$(\Pi', \mathcal{D}', P(\mathbf{c})@\rho)$;
11     $b :=$ AutomataProcedure$(\Pi'', \mathcal{D}'')$;
12     return the negation of $(b, \mathcal{D}')$;
```

We conclude this section by recalling that Procedure 3 is non-terminating, as it is possible for the main loop to run for an unbounded number of iterations. In our approach, we will also exploit a terminating (but incomplete) variant of the procedure, which we call OptimisedSeminaïveHalt. This variant is obtained by adding an additional termination condition just before Line 19 which breaks the loop and returns Π' and $\mathcal{D}'$ (without a truth value) if the flag has been set to 1. Thus, such a variant of the procedure ensures termination with a (partial) materialisation once all the non-recursive predicates in the input programme have been fully materialised, and also returns a subset of relevant rules in the programme.

5.3 Combining materialisation and automata

We are now ready to provide a practical and scalable reasoning algorithm combining optimised seminaïve materialisation and automata construction. Our algorithm delegates the bulk of the computation to the materialisation component and resorts to automata-based techniques only as needed to ensure termination and completeness.

Our approach is summarised in Algorithm 4. When given as input a programme Π, a dataset $\mathcal{D}$, and a fact $P(\mathbf{c})@\rho$, the algorithm starts by identifying the subset Π' of the input rules and the subset $\mathcal{D}'$ of the input data that are relevant to deriving the input fact $P(\mathbf{c})@\rho$ (Lines 1 and 2). This optimisation is based on the observation that a programme Π and dataset $\mathcal{D}$ entail a fact of the form $P(\mathbf{c})@\rho$ if and only if so do the subset Π^P of P-relevant rules in Π and the subset $\mathcal{D}^P$ of facts in $\mathcal{D}$ mentioning only predicates from Π^P. Then, the algorithm proceeds according to the following steps.

1. Pre-materialise Π' and $\mathcal{D}'$ using the terminating (but possibly incomplete) version of the seminaïve materialisation procedure OptimisedSeminaïveHalt discussed at the end of Section 5.2. If the procedure terminates because an inconsistency has been found, or the input fact has been derived, or a fixpoint has been reached,

then we can terminate and report output (Line 5); otherwise, we have obtained a partial materialisation where non-recursive predicates have been fully materialised and we can proceed to the next stage.

2. Run two threads in parallel. In the first (possibly non-terminating) thread, continue performing optimised semi-naïve materialisation until the input fact (or an inconsistency) is derived or a fixpoint is reached. In turn, in the second thread we resort to automata-based techniques applied to the partial materialisation obtained in the first stage of the algorithm; in particular, this second thread applies in Line 10 the reduction from fact entailment to inconsistency checking as described in Section 3.2, and then applies in Line 11 the automata-based decision procedure of (Wałęga *et al.* 2019a), as described in Section 4.3.

It follows directly from the results proved earlier in this section and the theoretical results of Wałęga *et al.* (2019a) that Algorithm 4 is sound, complete, and terminating. Furthermore, the algorithm is designed so that, on the one hand, materialisation is favoured over automata-based reasoning and, on the other hand, the automata construction relies on a recursive programme that is as small as possible as well as on a partial materialisation that is as complete as possible. The favourable computational behaviour of this approach will be confirmed by our experiments.

6 Implementation and evaluation

We have implemented our practical decision procedure from Algorithm 4 in a system called MeTeoR. Our system is implemented in Python 3.8 and does not depend on third-party libraries. MeTeoR is available for download and its demo can also be accessed via an online interface.[3] In this section, we describe the implementation choices made in MeTeoR and report the results of an extensive empirical evaluation on available benchmarks.

6.1 Implementation details

A scalable implementation of materialisation-based reasoning requires suitable representation and storage schemes for facts. In MeTeoR, we associate to each ground relational atom a list of intervals sorted by their left endpoints, which provides a compact account of all facts mentioning a given relational atom. Each ground relational atom is further indexed by a composite key consisting of its predicate and its tuple of constants. This layout is useful for fact coalescing and fact entailment checking; for instance, to check if a fact $M@\rho$ is entailed by a dataset $\mathscr{D}$, it suffices to find and interval ρ' such that $M@\rho' \in \mathscr{D}$ and $\rho \subseteq \rho'$; this can be achieved by first scanning the sorted list of intervals for M using the index and checking if ρ is a subset of one of these intervals. Additionally, each ground relational atom is also indexed by each of its term arguments to facilitate joins. Finally, when a fact is inserted into the dataset, the corresponding list of intervals is sorted to facilitate subsequent operations.

[3] The version of MeToeR used in this paper is available at https://github.com/wdimmy/DatalogMTL˙Practical˙Reasoning and its online demo is available at https://meteor.cs.ox.ac.uk/.

MeTeoR also implements an optimised version of (temporal) joins, which are required to evaluate rules with multiple body atoms. A naïve implementation of the join of metric atoms $M_1, \ldots, M_n$ occurring in the body of a rule would require computing all intersections of intervals $\rho_1, \ldots, \rho_n$ such that $M_i@\rho_i$ occurs in the so-far constructed dataset, for each $i \in \{1, \ldots, n\}$. Since each M_i may hold in multiple intervals in the dataset, the naïve approach is ineffective in practice. In contrast, we implemented a variant of the classical sort-merge join algorithm: we first sort all n lists of intervals corresponding to $M_1, \ldots, M_n$, and then scan the sorted lists to compute the intersections, which improves performance. Our approach to temporal joins can be seen as a generalisation of the idea sketched by Brandt *et al.* (2018, Section 5), which deals with two metric atoms at a time but has not been put into practice to the best of our knowledge.

6.2 Baselines and machine configuration

We compared MeTeoR with two baselines: the query rewriting approach by Brandt *et al.* (2018) and the reduction to LTL reasoning proposed by Brandt *et al.* (2018) which was later implemented by Yang (2022). We could not compare our approach to the temporal extension of the Vadalog system recently proposed by Bellomarini *et al.* (2022) as the system is not accessible.

The query rewriting approach by Brandt *et al.* (2018) is based on rewriting a target predicate P with respect to an input programme Π into an SQL query that, when evaluated over the input dataset $\mathscr{D}$, provides the set of all facts with maximal intervals over P entailed by Π and $\mathscr{D}$. The only implementation of this approach that we are aware of is the extension of the Ontop system described by Kalaycı *et al.* (2018); this implementation, however, is no longer available[4] and hence we have produced our own implementation. Following Brandt *et al.* (2018), we used temporary tables (instead of subqueries) to compute the extensions of predicates appearing in Π on the fly, which has two implications. First, we usually have not just one SQL query but a set of queries for computing the final result; second, similarly to MeTeoR, the approach essentially works bottom-up rather than top-down. The implementation by Brandt *et al.* (2018) provides two variants for coalescing: the first one uses standard SQL queries by Zhou *et al.* (2006), whereas the second one implements coalescing explicitly. For our baseline we adopted the standard SQL approach, which is less dependent on implementation details. Finally, we chose Postgresql 10.18 as the underpinning database for all our baseline experiments.

As shown by Brandt *et al.* (2018), DatalogMTL reasoning can be reduced to LTL reasoning by means of an exponential translation. This approach has been implemented by Yang (2022) using BLACK (Geatti *et al.* 2019) as the LTL reasoner of choice, and we used their implementation as a baseline for our experiments.

All experiments have been conducted on a Dell PowerEdge R730 server with 512 GB RAM and two Intel Xeon E5-2640 2.6 GHz processors running Fedora 33, kernel version 5.8.17. In each experiment, we verified that the answers returned by MeTeoR and the baselines coincide.

[4] Personal communication with the authors.

6.3 Benchmarks

The *Temporal LUBM Benchmark* is an extension of the Lehigh University Benchmark (Guo *et al.* 2005) with temporal rules and data, which we developed (Wang *et al.* 2022a). LUBM's data generator, which provides means for generating datasets of different sizes, has been extended to randomly assign an interval to each generated fact; the rational endpoints of each interval belong to a range, which is a parameter. In addition, the plain Datalog rules from the OWL 2 RL fragment of LUBM have been extended with 29 rules involving recursion and mentioning all metric operators in DatalogMTL. The resulting DatalogMTL programme is denoted by Π_L.

The *Weather Benchmark* is based on freely available data with meteorological observations in the US (Maurer *et al.* 2002);[5] in particular, we used a dataset $\mathscr{D}_W$ consisting of 397 million temporal facts spanning over the years 1949–2010, and then considered smaller subsets of this dataset. Similar data was used in experiments of Brandt *et al.* (2018), however the format of data has slightly changed, so we adapted their (non-recursive) programme (consisting of 4 rules) for detecting US states affected by excessive heat and heavy wind, so that the programme matches the data. The obtained programme is still non-recursive and, in what follows, it is denoted by Π_W.

The *iTemporal Benchmark* provides a synthetic generator for DatalogMTL programmes and datasets (Bellomarini *et al.* 2022).[6] The benchmark can generate different types of DatalogMTL programmes and equips them with datasets. We used iTemporal to generate 10 recursive programmes $\Pi_I^1, \dots, \Pi_I^{10}$ with different structures and containing 20–30 rules each, as well as corresponding datasets of various size.

6.4 Experiments

In this section, we describe experiments that we performed for benchmarking MeTeoR.[7]

6.4.1 Automata vs LTL.

Although MeTeoR utilises automata to ensure completeness and termination, we could have alternatively relied on the translation to LTL implemented by Yang (2022) together with BLACK (Geatti *et al.* 2019) as the LTL reasoner of choice. To determine which choice is favourable, we considered programmes $\Pi_I^1, \dots, \Pi_I^{10}$ from the iTempora benchmark, together with datasets containing $1,000$ facts each. For each pair of a dataset and a programme, we constructed 10 query facts; these facts were generated by first randomly choosing a predicate that appears in a programme or in a dataset; second, constants are randomly chosen among those present in the programme and in the dataset; third, an interval for the fact is generated. Next, we performed fact entailment by first reducing to inconsistency checking and then either applying the automata-based approach or constructing an LTL formula and checking it's satisfiability with BLACK. Average running times per 10 query facts for both approaches are reported in Figure 1 (left),

[5] https://www.engr.scu.edu/emaurer/gridded˙obs/index˙gridded˙obs.html
[6] https://github.com/kglab-tuwien/iTemporal.git
[7] All the datasets and programmes that we used in the experiments are available together with exploited MeTeoR implementation at https://github.com/wdimmy/DatalogMTL˙Practical˙Reasoning.

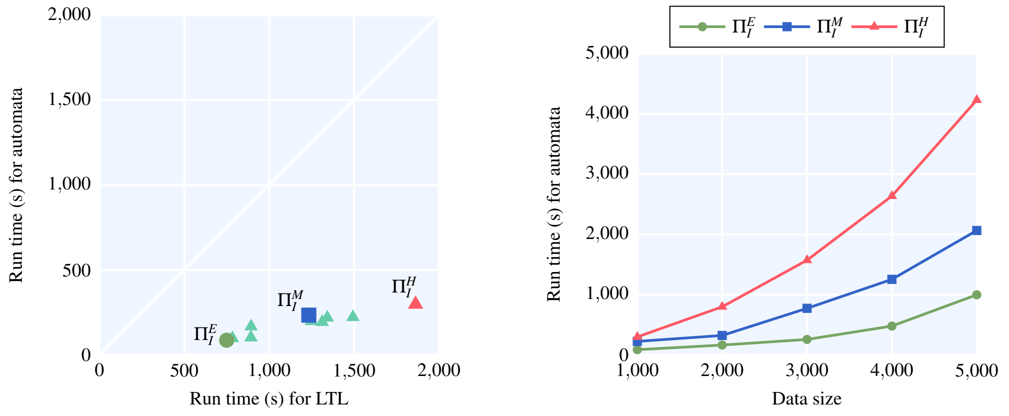


Fig. 1. Comparison of automata- and LTL-based approaches (left) and scalability of the automata approach (right).

where each point represents the running times for the automata-based approach (vertical coordinate) and LTL-based approach (horizontal coordinate) for a single programme and dataset. We can observe that the automata-based approach is consistently about an order of magnitude faster, which justifies our choice over an LTL-based reasoner.

6.4.2 Scalability of automata.

We have conducted scalability experiments to understand the practical limitations of the automata-based approach. Based on the results from the previous experiment, we chose an "easy" programme Π_I^E, a "medium" programme Π_I^M, and a "hard" programme Π_I^H amongst programmes $\Pi_I^1, \ldots, \Pi_I^{10}$, as depicted in Figure 1 (left). We used iTemporal to generate increasing size datasets for these programmes and we randomly generated query facts as in the previous experiment. The runtimes of automata-based approach for such inputs are summarised in Figure 1 (right), where we observe that the automata-based procedure is able to solve non-trivial problems, but struggles to scale beyond datasets with a few thousand temporal facts.

6.4.3 Comparison of materialisation strategies.

We compared the naïve, seminaïve, and optimised seminaïve materialisation procedures described in Sections 4.1, 5.1, and 5.2, respectively. To this end, for each of the programmes Π_i^E, Π_i^M, and Π_i^H we used iTemporal to generate a dataset containing one million facts. In Figure 2 we present running time and memory consumption (measured as the maximum number of facts stored in memory) over the first 30 materialisation steps in each case. As we can see, the seminaïve procedure consistently outperforms the naïve approach both in terms of running time and memory consumption, especially as materialisation progresses. In turn, the optimised seminaïve approach disregards rules after 6th, 7th, and 8th materialisation steps for Π_I^E, Π_I^M, and Π_I^H, respectively; from then onwards, the optimised procedure outperforms the basic seminaïve procedure. We can note that after several steps of materialisation the memory usage stabilises; the number of facts in

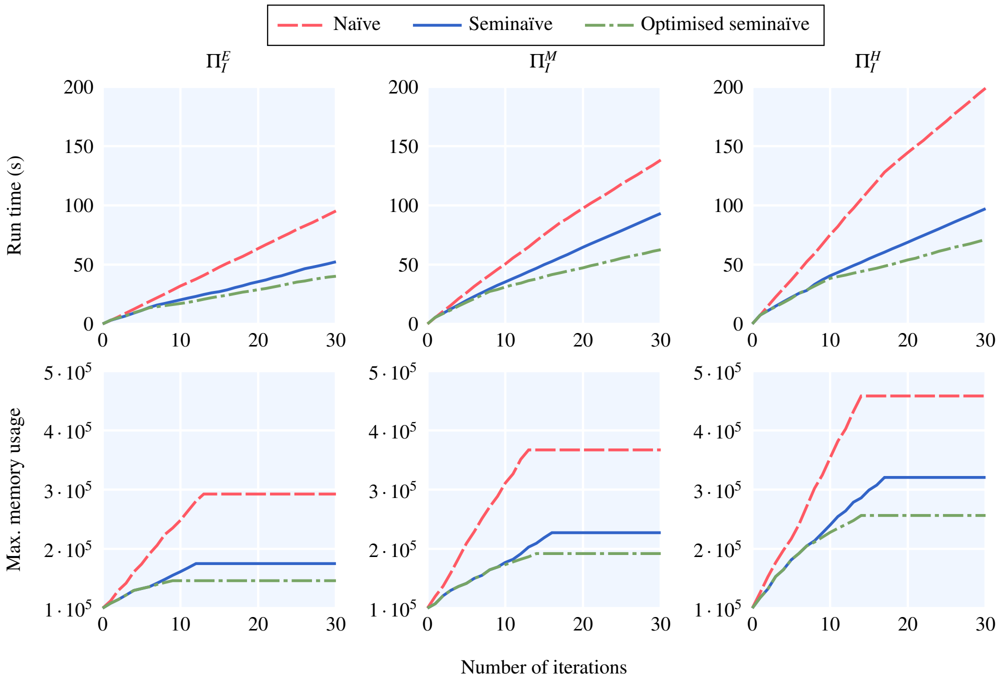


Fig. 2. Comparison of time and memory consumption in various materialisation strategies.

memory stops increasing but their intervals keep growing. This behaviour seems to be specific to programmes we used in this experiment.

6.4.4 Scalability of optimised seminaïve materialisation.

Our previous experiments provide evidence of the superiority of optimised seminaïve materialisation over the two alternative approaches. We further evaluated the scalability of optimised seminaïve materialisation as the size of the data increases. To this end, we considered the three iTemporal programmes Π_I^E, Π_I^M, and Π_I^H as well as programme Π_L from the Temporal LUBM Benchmark, and Π_W from the Weather Benchmark. In Figure 3 we show running times and memory consumption after the first 10 materialisation steps for datasets with up to 10 million temporal facts. We note here that the programme Π_W is non-recursive and its full materialisation is obtained already after 3 materialisation steps; all other programmes are recursive. Our results suggest that the optimised seminaïve materialisation procedure exhibits favourable scalability behaviour as data size increases.

6.4.5 Comparison with query rewriting.

We compared the performance of MeTeoR with the query rewriting baseline. As the latter does not support recursive programmes, we considered non-recursive programmes only. For this, we generated non-recursive subsets of the programme Π_L from the Temporal LUBM Benchmark and two subsets of the programme Π_W from the Weather Benchmark. In particular, we considered fragments Π_L^1 (with 5 rules), Π_L^2 (10 rules), and Π_L^3 (21 rules)

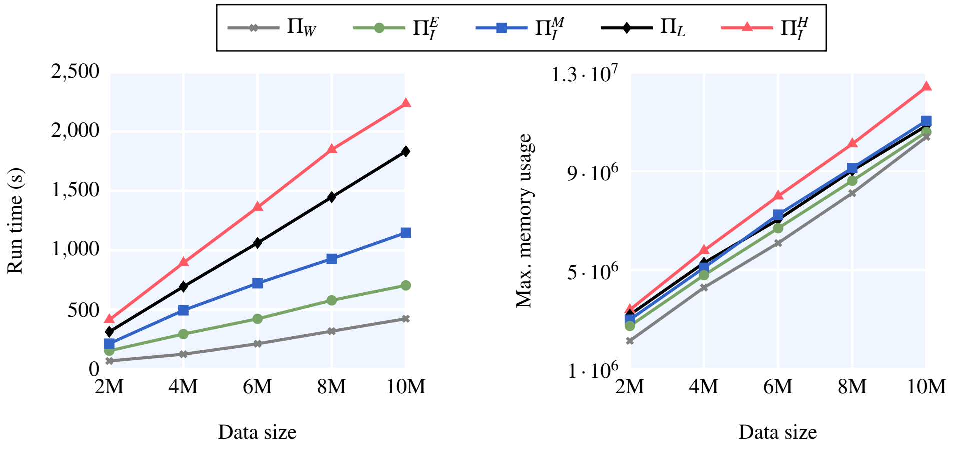


Fig. 3. Scalability of our optimised seminaïve materialisation.

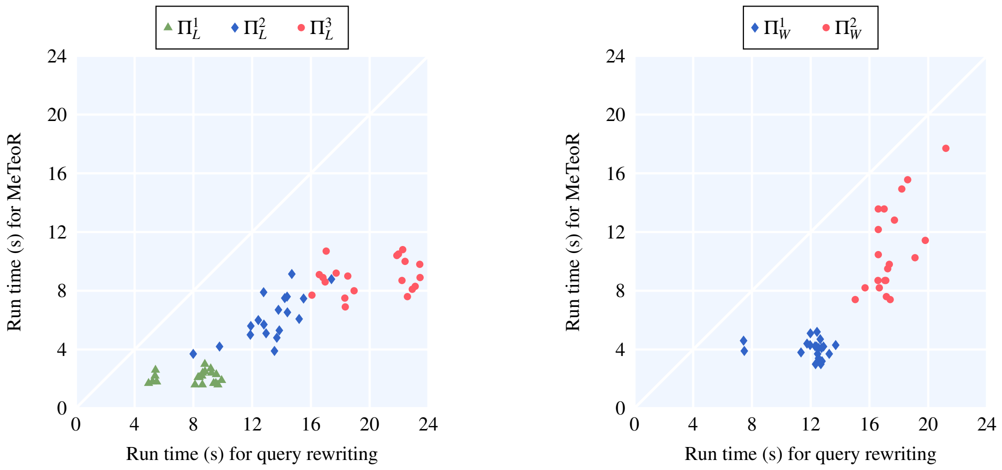


Fig. 4. Comparison between meTeoR and the query rewriting baseline.

of Π_L with rules relevant (see the end of Section 3.1 for the precise definition of this notion) to predicates *ResearchAssistant*, *Lecturer*, and *FullProfessorCandidate*, respectively. For the Weather Benchmark we used fragments Π_W^1 and Π_W^2 (each with 2 rules) of Π_W, with rules relevant to predicates *HeatAffectedState* and *HeavyWindAffectedState*, respectively. For each programme we generated a collection of datasets with 100, 000 facts each and compared the runtime of MeTeoR with the baseline in the task of computing all entailed facts over the predicates mentioned above. Note that since the programmes are non-recursive, MeTeoR runs only (optimised seminaïve) materialisation without exploiting automata. Figure 4 presents results, where each point represents the running times for MeTeoR (vertical coordinate) and the baseline (horizontal coordinate) for particular pairs of a programme and a dataset; MeTeoR consistently outperformed the baseline.

6.4.6 Usage of materialisation and automata in meTeoR.

Our previous experiments provide strong indication that the optimised seminaïve materialisation algorithm implemented in MeTeoR is more scalable than the automata-based component and that it can successfully deal with complex rules and datasets containing millions of temporal facts. Next, we assess whether Algorithm 4 is indeed effective in delegating most of the reasoning workload to the scalable materialisation component. To this end, we considered programmes $\Pi_I^1, \dots, \Pi_I^{10}$ generated by iTemporal together with their corresponding datasets with 1000 facts each. For each pair of a programme and a dataset we randomly generated 100 query facts and checked, for which facts entailment can be checked by MeTeoR using materialisation (i.e. Algorithm 4 terminates in either Line 3 or Line 8) and for which by the automata component (i.e. Algorithm 4 terminates in Line 12). It turned out that in 99.7% cases MeTeoR decided entailment using materialisation only, and so, only in 0.3% cases the automata component was used. Although these results are obviously biased by the way iTemporal generates programmes and datasets, they do support our hypothesis that focusing on the materialisation component in MeTeoR can provide a practically efficient reasoning mechanism.

7 Conclusions

In this paper, we presented a practical reasoning algorithm for full DatalogMTL, which combines materialisation with an automata-based approach. To achieve favourable performance, our algorithm delegates most of the computations to the materialisation component, which we optimised using seminaïve reasoning strategies. In turn, the computationally-expensive automata-based approach is used only to guarantee termination and completeness of reasoning. We have implemented our algorithm in the MeTeoR reasoner and made it publicly available. Our extensive evaluation supports our practicality and scalability claims.

We see many avenues for future research. First, DatalogMTL has been extended with stratified negation-as-failure (Cucala *et al.* 2021) and our seminaïve procedure could be extended accordingly. We are also working on blocking conditions that exploit the periodic structure of canonical models to ensure termination of materialisation-based reasoning. Finally, incremental materialisation-based reasoning has been studied in context of Datalog (Motik *et al.* 2019), and it would be interesting to lift such approaches to the DatalogMTL setting.

Acknowledgments

Our research was funded by the following EPSRC projects: OASIS (EP/S032347/1), and UK FIRES (EP/S019111/1), as well as by SIRIUS Centre for Scalable Data Access, Samsung Research UK, and NSFC grant No. 62206169. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

Abiteboul, S., Hull, R. and Vianu, V. 1995. *Foundations of Databases.* Addison-Wesley.

Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., Schaub, T., Schuhmann, A. and Vidal, C. 2021 Linear-time temporal answer set programming. In *Theory and Practice of Logic Programming*, 1–55.

Artale, A. and Franconi, E. 1998. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research* 9, 463–506.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F. and Zakharyaschev, M. 2015. First-order rewritability of temporal ontology-mediated queries. In *Proc. of the International Joint Conference on Artificial Intelligence*, 25 July 2015, 2706–2712.

Baader, F., Borgwardt, S., Koopmann, P., Ozaki, A. and Thost, V. 2017. Metric temporal description logics with interval-rigid names.In *Frontiers of Combining Systems*, C. Dixon and M. Finger, Eds. Springer Verlag, 60–76.

Baier, C. and Katoen, J.-P. 2008. *Principles of Model Checking.* MIT Press.

Beck, H., Dao-Tran, M. and Eiter, T. 2018. LARS: A logic-based framework for analytic reasoning over streams. *Artificial Intelligence* 261, 16–70.

Bellomarini, L., Blasi, L., Nissl, M. and Sallinger, E. 2022, The temporal Vadalog system. In *International Joint Conference on Rules and Reasoning*, 130–145.

Bellomarini, L., Nissl, M. and Sallinger, E. 2021. Query evaluation in DatalogMTL – taming infinite query results. *CoRR abs/2109.10691* .

Bellomarini, L., Nissl, M. and Sallinger, E. 2022. iTemporal: An extensible temporal benchmark generator. In *2022 IEEE 38th International Conference on Data Engineering*, 2021–2033.

Bellomarini, L., Sallinger, E. and Gottlob, G. 2018. The Vadalog system: Datalog-based reasoning for knowledge graphs. In *Proc. of the VLDB Endowment*, 975–987.

Brandt, S., Kalayci, 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 *Proc. of the AAAI Conference on Artificial Intelligence*, 1070–1076.

Brandt, S., Kalayci, E. G., Ryzhikov, V., Xiao, G. and Zakharyaschev, M. 2018. Querying log data with metric temporal logic. *Journal of Artificial Intelligence Research* 62, 829–877.

Bry, F., Eisinger, N., Eiter, T., Furche, T., Gottlob, G., Ley, C., Linse, B., Pichler, R. and Wei, F. 2007. Foundations of rule-based query answering, In *Reasoning Web*, 1–153.

Cabalar, P., Diéguez, M., Schaub, T. and Schuhmann, A. 2020. Towards metric temporal answer set programming. *Theory and Practice of Logic Programming* 20, 5, 783–798.

Carral, D., Dragoste, I., González, L., Jacobs, C. J. H., Krötzsch, M. and Urbani, J. 2019. Vlog: A rule engine for knowledge graphs. In *International Semantic Web Conference*, 19–35.

Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M. and Tonetta, S. 2014. The nuXmv symbolic model checker. In *Computer Aided Verification*, 334–342.

Ceri, S., Gottlob, G. and Tanca, L. 1989. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1, 1, 146–166.

Chomicki, J. and Imielinski, T. 1988. Temporal deductive databases and infinite objects. In *Proc. of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 61–73.

Cucala, D. J. T., Wałęga, P. A., Cuenca Grau, B. and Kostylev, E. V. 2021. Stratified negation in Datalog with metric temporal operators. In *Proc. of the Thirty-Third AAAI Conference on Artificial Intelligence*, 6488–6495.

Geatti, L., Gigante, N. and Montanari, A. 2019. A sat-based encoding of the one-pass and tree-shaped tableau system for LTL. In *Automated Reasoning with Analytic Tableaux and Related Methods*, 3–20.

Guo, Y., Pan, Z. and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3, 2-3, 158–182.

Gutiérrez-Basulto, V., Jung, J. C. and Ozaki, A. 2016. On metric temporal description logics. In *22nd European Conference on Artificial Intelligence*, 837–845.

Kalayci, E. G., Xiao, G., Ryzhikov, V., Kalayci, T. E. and Calvanese, D. 2018. Ontop-temporal: a tool for ontology-based query answering over temporal data. In *Proc. of the 27th ACM International Conference on Information and Knowledge Management*, 1927–1930.

Kikot, S., Ryzhikov, V., Wałęga, P. A. and Zakharyaschev, M. 2018. On the data complexity of ontology-mediated queries with MTL operators over timed words. In *Description Logics*

Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V. and Zakharyaschev, M. 2016. Temporal and spatial OBDA with many-dimensional Halpern-Shoham logic. In *Proc. of the International Joint Conference on Artificial Intelligence*, 1160–1166.

Koopmann, P. 2019. Ontology-based query answering for probabilistic temporal data. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2903–2910.

Koymans, R. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Systems* 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 *Proc. of the Fifth Fact Extraction and VERification Workshop*, 78–82.

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

Motik, B., Nenov, Y., Piro, R., Horrocks, I. and Olteanu, D. 2014. Parallel materialisation of Datalog programs in centralised, main-memory RDF systems. In *Proc. of the AAAI Conference on Artificial Intelligence*.

Nissl, M. and Sallinger, E. 2022. Modelling smart contracts with datalogmtl. In *Proc. of the Workshops of the EDBT/ICDT*.

Raheb, K. E., Mailis, T., Ryzhikov, V., Papapetrou, N. and Ioannidis, Y. E. 2017. Balonse: Temporal aspects of dance movement and its ontological representation. In *European Semantic Web Conference*, 49–64.

Ryzhikov, V., Wałęga, P. A. and Zakharyaschev, M. 2019a. Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics. In *International Joint Conferences on Artificial Intelligence*, 1851–1857.

Thost, V. 2018. Metric temporal extensions of DL-Lite and interval-rigid names. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 665–666.

Wałęga, P. A., Cuenca Grau, B., Kaminski, M. and Kostylev, E. V. 2019a. DatalogMTL: computational complexity and expressive power. In *International Joint Conferences on Artificial Intelligence*, 1886–1892.

Wałęga, P. A., Cuenca Grau, B., Kaminski, M. and Kostylev, E. V. 2020. DatalogMTL over the integer timeline. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 768–777.

Wałęga, P. A., Cuenca Grau, B., Kaminski, M. and Kostylev, E. V. 2021. Tractable fragments of Datalog with metric temporal operators. In *Proc. of the AAAI Conference on Artificial Intelligence*, 1919–1925.

Wałęga, P. A., Kaminski, M. and Cuenca Grau, B. 2019b. Reasoning over streaming data in metric temporal Datalog. In *Proc. of the AAAI Conference on Artificial Intelligence*, 3092–3099.

Wałęga, P. A., Tena Cucala, D. J., Kostylev, E. V. and Cuenca Grau, B. 2021. DatalogMTL with negation under stable models semantics. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 609–618.

Wałęga, P. A., Zawidzki, M. and Cuenca Grau, B. 2021. Finitely materialisable Datalog programs with metric temporal operators. In *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 619–628.

Wałęga, P., Zawidzki, M. and Grau, B. C. 2023. Finite materialisability of datalog programs with metric temporal operators. *Journal of Artificial Intelligence Research.* 76, 471–521.

Wang, D., Hu, P., Wałęga, P. A. and Grau, B. C. 2022a. MeTeoR: Practical reasoning in Datalog with metric temporal operators. In *Proc. of the AAAI Conference on Artificial Intelligence*, 5906–5913.

Wang, D., Wałęçga, P. A. and Grau, B. C. 2022. Seminaive materialisation in DatalogMTL. In *International Joint Conference on Rules and Reasoning*.

Yang, J. 2022. Translation of datalogMTL into PLTL, M.S. thesis, University of Oxford,UK.

Zhou, X., Wang, F. and Zaniolo, C. 2006. Efficient temporal coalescing query support in relational database systems. In *International Conference on Database and Expert Systems Applications*, 676–686.

Appendix A Proofs

Theorem 4.3: Consider Procedure 1 running on input Π and $\mathscr{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 1, it holds that $\mathfrak{I}_{\mathscr{D}'} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$.

Proof.
For each $k \in \mathbb{N}$, we let $\mathscr{N}_k$ and $\mathscr{D}_k$ denote the contents of, respectively, $\mathscr{N}$ and $\mathscr{D}$ in Procedure 1 upon the completion of the kth iteration of the loop. Thus, it suffices to show, inductively on $k \in \mathbb{N}$, that $\mathfrak{I}_{\mathscr{D}_k} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$.

In the base case, we have $\mathscr{D}_0 = \mathscr{D}$. Moreover, $T_\Pi^0(\mathfrak{I}_\mathscr{D}) = \mathfrak{I}_\mathscr{D}$, and so, $\mathfrak{I}_{\mathscr{D}_0} \subseteq T_\Pi^0(\mathfrak{I}_\mathscr{D})$, as required. For the inductive step, we assume that $\mathfrak{I}_{\mathscr{D}_k} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$, for some $k \in \mathbb{N}$, and that the procedure enters the $k+1$st iteration of the loop. If the $k+1$st iteration of the loop breaks in Line 7, then $\mathscr{D}_{k+1} = \mathscr{D}_k$. By the inductive assumption we have $\mathfrak{I}_{\mathscr{D}_k} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$, and therefore, $\mathfrak{I}_{\mathscr{D}_{k+1}} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$, and thus, $\mathfrak{I}_{\mathscr{D}_{k+1}} \subseteq T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D})$. Now, assume that the $k+1$st iteration of the loop does not break in Line 7. Hence, by Lines 4 and 8, we have $\mathscr{D}_{k+1} = \mathsf{coal}(\mathscr{D}_k \cup \mathscr{N}_{k+1})$. To show that $\mathfrak{I}_{\mathscr{D}_{k+1}} \subseteq T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D})$, we assume that $\mathfrak{I}_{\mathscr{D}_{k+1}} \models M@t$, for some relational fact $M@t$. Hence, we have $\mathscr{D}_k \models M@t$ or $\mathscr{N}_{k+1} \models M@t$.

Case 1: $\mathscr{D}_k \models M@t$. By the inductive assumption, $\mathfrak{I}_{\mathscr{D}_k} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$, so $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \models M@t$. Clearly, $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D})$, and so, $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$, as required.

Case 2: $\mathscr{N}_{k+1} \models M@t$. By Line 3 of Procedure 1, we obtain that $\mathscr{N}_{k+1} = \Pi(\mathscr{D}_k)$. Thus, by Expression (3) from Definition 4.2, there exists a rule $r \in \Pi$, say of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, such that $r[\mathscr{D}_k] \models M@t$. Therefore, by Expression (2) from Definition 4.2, there are a substitution σ and some intervals $\rho_1, \ldots, \rho_n$ such that $(M_1\sigma@\rho_1, \ldots, M_n\sigma@\rho_n) \in \mathsf{inst}_r[D_k]$ and $M'\sigma@(\rho_1 \cap \cdots \cap \rho_n) \models M@t$. Since $(M_1\sigma@\rho_1, \ldots, M_n\sigma@\rho_n)$ belongs to $\mathsf{inst}_r[\mathscr{D}_k]$, by Expression (2) from Definition 4.2, we obtain that $\mathscr{D}_k \models M_i@\rho_i$, for each $i \in \{1, \ldots, n\}$. Therefore, by the definition of T_Π, we obtain that $T_\Pi(\mathfrak{I}_{\mathscr{D}_k}) \models M'\sigma@(\rho_1 \cap \cdots \cap \rho_n)$ and so $T_\Pi(\mathfrak{I}_{\mathscr{D}_k}) \models M@t$. Finally, by the inductive assumption, $\mathfrak{I}_{\mathscr{D}_k} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$, so $T_\Pi(\mathfrak{I}_{\mathscr{D}_k}) \subseteq T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D})$, and therefore we obtain that $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$.

Theorem 4.4: Consider Procedure 1 running on input Π and $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 1, it holds that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}'}$.

Proof.
We use the same symbols $\mathscr{N}_k$ and $\mathscr{D}_k$ as in the proof of Theorem 4.3 and we define Δ_k analogously. We will show, inductively on natural numbers k, that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}_k}$. The base case holds trivially, because we have $\mathscr{D}_0 = \mathscr{D}$, and so, $T_\Pi^0(\mathfrak{I}_\mathscr{D}) = \mathfrak{I}_{\mathscr{D}_0}$. For the inductive step assume that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}_k}$, for some $k \in \mathbb{N}$. Moreover, let $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$, for some relational fact $M@t$. We will show that $\mathfrak{I}_{\mathscr{D}_{k+1}} \models M@t$. We have either $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \models M@t$ or $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \not\models M@t$.

Case 1: $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \models M@t$. Then, by the inductive assumption, $\mathfrak{I}_{\mathscr{D}_k} \models M@t$. Since $\mathscr{D}_{k+1} \models \mathscr{D}_k$, we obtain that $\mathfrak{I}_{\mathscr{D}_{k+1}} \models M@t$.

Case 2: $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \not\models M@t$. Since $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$, there exist a rule $r \in \Pi$, say of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, and a time point t' such that an application of r at t' yields $M@t$. More precisely, it means that there is a substitution σ such that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \models M_i\sigma@t'$, for each $i \in \{1, \ldots, n\}$, and $M'\sigma@t' \models M@t$. Therefore, by the fact that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}_k}$ and by Expression (2) from Definition 4.2, there need to exist some intervals $\rho_1, \ldots, \rho_n$ such that $t' \in \rho_1 \cap \cdots \cap \rho_n$ and $(M_1\sigma@\rho_1, \cdots, M_n\sigma@\rho_n) \in \mathsf{inst}_r[\mathscr{D}_k]$. Thus, as $M'\sigma@t' \models M@t$, we obtain by Definition 4.2 that $r[\mathscr{D}_k] \models M@t$, and so, $\Pi[\mathscr{D}_k] \models M@t$. Finally, recall that $\mathscr{D}_{k+1} = \mathsf{coal}(\mathscr{D}_k \cup \mathscr{N}_{k+1})$ and $\mathscr{N}_{k+1} = \Pi[\mathscr{D}_k]$. Therefore, the fact that $\Pi[\mathscr{D}_k] \models M@t$ implies that $\mathfrak{I}_{\mathscr{D}_{k+1}} \models M@t$.

Theorem 5.2: Consider Procedure 2 running on input Π and $\mathscr{D}$. Upon the completion of the kth (for some $k \in \mathbb{N}$) iteration of the loop of Procedure 2, it holds that $\mathfrak{I}_{\mathscr{D}'} \subseteq T_\Pi^k(\mathfrak{I}_\mathscr{D})$.

Proof.
The proof relies on the observation that rule instances processed by seminaïve evaluation are also processed by the naïve evaluation. In particular, directly by Definition 5.1 we obtain that $\mathsf{inst}_r[\mathscr{D}' \mathbin{\vdots} \Delta] \subseteq \mathsf{inst}_r[\mathscr{D}']$, for each r, $\mathscr{D}'$, and Δ. Hence, in each loop iteration, the sets $\mathscr{N}$ and $\mathscr{C}$ in Procedure 2 are subsets of the corresponding sets in Procedure 1. Consequently, the same holds for the set $\mathscr{D}'$, and so, soundness follows from Theorem 4.3.

Theorem 5.3: Consider Procedure 2 with input programme Π and input dataset $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}'}$.

Proof.
We will use $\mathscr{N}_k$, Δ_k, and $\mathscr{D}'_k$, for the contents of, respectively, $\mathscr{N}$, Δ, and $\mathscr{D}'$ in Procedure 2 upon the completion of the kth iteration of the loop. Now, as in the proof of Theorem 4.4, we will show inductively on natural numbers k, that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \subseteq \mathfrak{I}_{\mathscr{D}_k}$.

The only difference with respect to the proof of Theorem 4.4 lies in Case 2, where we assume that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \not\models M@t$ and $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$, for some relational fact $M@t$. By the inductive assumption, as in the proof of Theorem 4.4, there needs to be an instance $(M_1\sigma@\rho_1, \cdots, M_n\sigma@\rho_n) \in \mathsf{inst}_r[\mathscr{D}_k]$ of some rule r of the form $M' \leftarrow M_1 \wedge \cdots \wedge M_n$, and a time point $t' \in \rho_1 \cap \cdots \cap \rho_n$ such that $M'\sigma@t' \models M@t$. Now, we argue that this instance is not only in $\mathsf{inst}_r[\mathscr{D}_k]$, but also in $\mathsf{inst}_r[\mathscr{D}_k \mathbin{\vdots} \Delta_k]$, namely that $(M_1\sigma@\rho_1, \cdots, M_n\sigma@\rho_n) \in \mathsf{inst}_r[\mathscr{D}_k \mathbin{\vdots} \Delta_k]$. To this end, by Definition 5.1, it suffices to show that there is $i \in \{1, \ldots, n\}$ such that $\mathscr{D}_k \setminus \Delta_k \not\models M_i\sigma@\rho_i$. We will consider two cases, namely, when $k = 0$ and when $k > 0$.

Case 2.1: $k = 0$. By the initialisation (Line 1) of the of Procedure 2,$\Delta_0 = \mathscr{D}$, so $\mathscr{D}_0 \setminus \Delta_0 = \emptyset$. Recall that we assumed in the paper that input programmes do not have rules with vacuously satisfied bodies, so there needs to exist $i \in \{1, \ldots, n\}$ such that the empty interpretation does not satisfy $M_i\sigma@\rho_i$, that is, $\mathscr{D}_0 \setminus \Delta_0 \not\models M_i\sigma@\rho_i$.

Case 2.2: $k > 0$. By Lines 9 and 4, we have $\mathscr{D}_k = \mathsf{coal}(\mathscr{D}_{k-1} \cup \mathscr{N}_k)$. Moreover, by the definition of Δ in Line 5, we obtain $\mathscr{D}_k = \mathsf{coal}(\mathscr{D}_{k-1} \cup \Delta_k)$. Thus $\mathfrak{I}_{\mathscr{D}_k} = \mathfrak{I}_{\mathscr{D}_{k-1}} \cup \mathfrak{I}_{\Delta_k}$, so $\mathfrak{I}_{\mathscr{D}_k} \setminus \mathfrak{I}_{\Delta_k} = \mathfrak{I}_{\mathscr{D}_{k-1}} \setminus \mathfrak{I}_{\Delta_k}$, and therefore $\mathfrak{I}_{\mathscr{D}_k} \setminus \mathfrak{I}_{\Delta_k} \subseteq \mathfrak{I}_{\mathscr{D}_{k-1}}$. Hence, to show that for some $i \in \{1, \ldots, n\}$ we have that $\mathscr{D}_k \setminus \Delta_k \not\models M_i\sigma@\rho_i$, it suffices to show that $\mathscr{D}_{k-1} \not\models M_i\sigma@\rho_i$. Suppose towards a contradiction that $\mathscr{D}_{k-1} \models M_i\sigma@\rho_i$, for all $i \in \{1, \ldots, n\}$. By Theorem 5.2, $T_\Pi^{k-1}(\mathfrak{I}_\mathscr{D}) \models M_i\sigma@\rho_i$, for all $i \in \{1, \ldots, n\}$. Thus, $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \models M@t$, which raises a contradiction.

Lemma 5.4: Consider Procedure 3 running on input programme Π and dataset $\mathscr{D}$. Whenever $flag = 1$, dataset $\mathscr{D}'$ satisfies all facts over a non-recursive predicate in Π that are entailed by Π and $\mathscr{D}$.

Proof.
In Procedure 3, $flag$ is initialised to 0 and once it is changed to 1 (in Line 10) its value cannot be reverted. Moreover, the consecutive contents of $\mathscr{D}'$ are obtained by coalescing the previous contents with facts in $\mathscr{N}$ (in Line 4), so once a fact is entailed by $\mathscr{D}'$, it will remain entailed by all the consecutive contents of $\mathscr{D}'$. Therefore, to prove the lemma, it suffices to consider the step of computation in which $flag$ becomes 1. Let k be this step and let $\mathscr{D}'_k$, be the contents of $\mathscr{D}'$ upon the completion of the kth iteration of the loop. Hence, we need to show that $\mathscr{D}'_k$ satisfies all the facts which are satisfied in $\mathfrak{C}_{\Pi,\mathscr{D}}$ and which mention non-recursive predicates in Π.

For this, we observe that before $flag$ changes to 1, Procedure 3 works as Procedure 2 so, by Theorems 5.2 and 5.3, we have $\mathfrak{I}_{\mathscr{D}'_k} = T_\Pi^k(\mathfrak{I}_\mathscr{D})$. Hence, as $\mathfrak{C}_{\Pi,\mathscr{D}} = T_\Pi^{\omega_1}(\mathfrak{I}_\mathscr{D})$, it suffices to show by a transfinite induction that, for each ordinal $\alpha \geq k$, the interpretations $T_\Pi^k(\mathfrak{I}_\mathscr{D})$ and $T_\Pi^\alpha(\mathfrak{I}_\mathscr{D})$ satisfy the same facts with non-recursive predicates in Π. For the base case assume that $T_\Pi^{k+1}(\mathfrak{I}_\mathscr{D}) \models M@t$, for some relational fact $M@t$ whose predicate is non-recursive in Π. Hence, there is a rule $r \in \mathsf{ground}(\Pi, \mathscr{D})$ and a time point t' such that $T_\Pi^k(\mathfrak{I}_\mathscr{D})$ entails each body atom of r at t', and the head of r holding at t' entails $M@t$. Thus the head atom of r has a non-recursive predicate in Π; therefore, by the definition, each body atom in r also mentions only non-recursive predicates in Π. Now, as $T_\Pi^k(\mathfrak{I}_\mathscr{D}) \not\models M@t$, we obtain that $T_\Pi^k(\mathfrak{I}_\mathscr{D})$ and $T_\Pi^{k-1}(\mathfrak{I}_\mathscr{D})$ do not satisfy the same relational facts with non-recursive predicates in Π, which directly contradicts the condition from Line 9. The inductive step for a successor ordinal α uses the same argument; indeed, if $T_\Pi^\alpha(\mathfrak{I}_\mathscr{D}) \models M@t$ and $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathscr{D}) \models M@t$, for some relational fact $M@t$ whose predicate is non-recursive in Π, then $T_\Pi^\alpha(\mathfrak{I}_\mathscr{D})$ and $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathscr{D})$ do not satisfy the same relational facts with non-recursive predicates, contradicting the inductive assumption. The inductive step for the transfinite ordinal α holds trivially as $T_\Pi^\alpha(\mathfrak{I}_\mathscr{D}) = \bigcup_{\beta<\alpha} T_\Pi^\beta(\mathfrak{I}_\mathscr{D})$.

Theorem 5.6: Consider Procedure 3 with input programme Π and input dataset $\mathscr{D}$. For each $k \in \mathbb{N}$, upon the completion of the kth iteration of the loop of Procedure 2, it holds that $T_\Pi^k(\mathfrak{I}_\mathscr{D}) = \mathfrak{I}_{\mathscr{D}'}$.

Proof.
If for both Lines 12 and 18, the condition in the "if" statement never applies, then Procedure 3 works in the same way as Procedure 2. Hence, by Theorems 5.2 and 5.3, we get $T_\Pi^k(\mathfrak{I}_\mathscr{D}) = \mathfrak{I}_{\mathscr{D}'}$. Otherwise, the loop from Procedure 3 works similarly as Procedure 2, except that it can delete in Lines 12 and 18 some rules. By Lemma 5.5, however, such rules can be safely deleted from the programme, without losing the properties established in Theorems 5.2 and 5.3.