

Reasoning Techniques in DatalogMTL

Przemysław Andrzej Wałęga^{1,*†}, Michał Zawidzki^{1,2†} and Bernardo Cuenca Grau^{1†}

¹Department of Computer Science, University of Oxford, 7 Parks Rd, Oxford, OX1 3QG, United Kingdom

²Department of Logic, University of Łódź, Lindleya 3/5, 91-131 Łódź, Poland

Abstract

DatalogMTL is a recently introduced extension of Datalog with operators from metric temporal logic (MTL). It allows for performing complex temporal reasoning tasks over the rational timeline, which makes it suitable for many practical applications. Although the main reasoning tasks in DatalogMTL are decidable, they have high computational complexity, and so, developing practically efficient reasoning techniques is challenging. Consequently, a number of approaches have been recently established; some of them have already been implemented and tested experimentally, but no comparison of these techniques has been provided yet. To fill this gap, we present an overview of reasoning techniques in DatalogMTL, sketch relations between them, and discuss their properties. Moreover, we present our ongoing research in this area and possible future directions thereof.

Keywords

DatalogMTL, temporal Datalog, reasoning techniques

1. Introduction

DatalogMTL [1] is a recently introduced extension of Datalog with operators from metric temporal logic (MTL) [2] interpreted over the rational timeline; for example an atom $\diamond_{[0,60]}A(x, y)$ states that $A(x, y)$ did hold at some time point in the past at least 0 and at most 60 seconds ago, whereas $\exists_{[0,60]}A(x, y)$ states that $A(x, y)$ did hold continuously in the above-mentioned interval. Such operators allow us to express complex temporal properties, like a turbine's *active power trip*, which is defined in Siemens remote diagnostic centre as an event when 'the active power was above 1.5MW for a period of at least 10 seconds, maximum 3 seconds after which there was a period of at least one minute where the active power was below 0.15MW' [1]. Indeed, we can express this concept with the rule

$$Active(x) \leftarrow Turbine(x) \wedge \exists_{[0,60]}Below0.15(x) \wedge \diamond_{[60,63]} \exists_{[0,10]} Above1.5(x).$$

Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy

*Corresponding author.

†These authors contributed equally.

✉ przemyslaw.walega@cs.ox.ac.uk (P. A. Wałęga); michal.zawidzki@cs.ox.ac.uk (M. Zawidzki);


bernardo.cuenca.grau@cs.ox.ac.uk (B. Cuenca Grau)

🌐 <https://sites.google.com/site/pawalega> (P. A. Wałęga); <https://www.cs.ox.ac.uk/people/bernardo.cuencagrau/>

(B. Cuenca Grau)

🆔 0000-0003-2922-0472 (P. A. Wałęga); 0000-0002-2394-6056 (M. Zawidzki); 0000-0003-2909-5923 (B. Cuenca Grau)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

Due to its high expressive power, DatalogMTL has found a number of potential applications, including stream reasoning [3], temporal ontology-based data access [4], and verification of banking agreements [5]. Consistency checking and fact entailment in DatalogMTL, however, are of high computational complexity (ExpSpace for combined [4], and PSpace for data complexity [6]), which makes the development of scalable implementations challenging. Despite these difficulties, there is currently growing interest in the development of practical reasoning algorithms for DatalogMTL, and new systems are becoming available.

In this paper, we provide a comprehensive overview of the different reasoning techniques available for DatalogMTL and discuss future research directions. In particular, we observe that reasoning techniques for DatalogMTL can be divided into two main groups:

1. Approaches based on *materialisation* (a.k.a. forward chaining), which derive and store in memory all facts that are entailed by a program and dataset. This is obtained by means of successive rounds of rule applications until a fixpoint is reached; the resulting set of facts is called the *materialisation*. In these approaches, fact entailment and query answering are performed directly over the materialisation. The main obstacle, however, is that the materialisation process in DatalogMTL is not guaranteed to terminate.
2. Approaches based on *discretisation* of the rational timeline, that is, on partitioning the dense timeline into a discrete sequence of intervals, within which all the time points satisfy the same relational atoms. Since this sequence of intervals is discrete, it allows us for exploiting techniques used for reasoning in temporal logics interpreted over the discrete timeline (e.g., linear temporal logic LTL). The problem of such approaches is that their implementations are usually inefficient in practice.

In the remainder of this paper, we will discuss in detail these approaches. We start by recapitulating the syntax, semantics, and main reasoning tasks for DatalogMTL in Section 2. In Section 3, we describe materialisation-based reasoning techniques, which exploit the fixpoint characterisation of the DatalogMTL semantics; in particular, we discuss the *naïve approach to materialisation* which does not guarantee termination, introduce *finitely materialisable* programs for which materialisation always terminates, and discuss algorithms based on *sliding windows* which are particularly useful in the stream reasoning setting. Next, in Section 4, we describe reasoning based on the discretisation of time; we show how the timeline can be discretised and how discretisation can be used to devise a *translation to LTL* or to provide *Büchi automata* and *arithmetic progressions* capturing the meaning of a DatalogMTL program. Finally, we present our ongoing research and outline its possible future directions in Section 5.

2. DatalogMTL

In this section, we introduce DatalogMTL; we will focus on the standard setting given by ‘continuous semantics’ and interpretations over the rational timeline. It is, however, worth mentioning that DatalogMTL was also studied under an alternative ‘pointwise’ semantics [7] and interpreted over the integer timeline [8].

2.1. Syntax

Syntactically, DatalogMTL is an extension of Datalog with the MTL operators \diamond , \diamond , \boxminus , \boxplus , \mathcal{S} , and \mathcal{U} , which are indexed with rational intervals ϱ containing only non-negative numbers. We distinguish two types of atoms. *Relational atoms* are standard Datalog atoms of the form $P(\mathbf{s})$, with an arbitrary arity predicate P and a tuple \mathbf{s} of terms (variables or constants). *Metric atoms* extend relational atoms by allowing MTL operators, and are generated by the following grammar:

$$M ::= \top \mid \perp \mid P(\mathbf{s}) \mid \diamond_{\varrho}M \mid \diamond_{\varrho}M \mid \boxminus_{\varrho}M \mid \boxplus_{\varrho}M \mid M\mathcal{S}_{\varrho}M \mid M\mathcal{U}_{\varrho}M.$$

Then, DatalogMTL *rules* are of the form

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

where each *body atom* M_i is a metric atom, and the *head atom* M' is a metric atom not mentioning any of the ‘non-deterministic’ operators \diamond , \diamond , \mathcal{S} , and \mathcal{U} . As usual, a *program* Π is a finite set of *safe* rules, but the definition of safety in DatalogMTL is slightly more elaborated than in Datalog. In particular, we do not only require that each variable in a rule’s head occurs in this rule’s body, but also that this occurrence in the body is not in a left operand of \mathcal{S} or \mathcal{U} . This additional requirement discards, for example, a rule of the form $B(x) \leftarrow A(x)\mathcal{S}_{[0,0]}\top$, which is equivalent to $B(x) \leftarrow \top$ (see the semantics of DatalogMTL operators from the next subsection), and so, should not be treated as safe.

A DatalogMTL *dataset* \mathcal{D} is a finite set of facts of the form $M@_{\varrho}$, where M is a *ground* (i.e., with no variables) relational atom and ϱ is an interval in which M holds true.

2.2. Semantics

A DatalogMTL interpretation \mathcal{I} can be seen as a set containing one standard Herbrand interpretation for each rational time point. More precisely, \mathcal{I} specifies, for each ground relational atom M and each time point $t \in \mathbb{Q}$, whether M holds at t , in which case we write $\mathcal{I}, t \models M$. Satisfaction of relational atoms determines satisfaction of (more complex) metric atoms, as presented below.

$\mathcal{I}, t \models \top$		for each t ,
$\mathcal{I}, t \models \perp$		for no t ,
$\mathcal{I}, t \models \diamond_{\varrho}M$	iff	$\mathcal{I}, t' \models M$ for some t' with $t - t' \in \varrho$,
$\mathcal{I}, t \models \diamond_{\varrho}M$	iff	$\mathcal{I}, t' \models M$ for some t' with $t' - t \in \varrho$,
$\mathcal{I}, t \models \boxminus_{\varrho}M$	iff	$\mathcal{I}, t' \models M$ for all t' with $t - t' \in \varrho$,
$\mathcal{I}, t \models \boxplus_{\varrho}M$	iff	$\mathcal{I}, t' \models M$ for all t' with $t' - t \in \varrho$,
$\mathcal{I}, t \models M_1\mathcal{S}_{\varrho}M_2$	iff	$\mathcal{I}, t' \models M_2$ for some t' with $t - t' \in \varrho$ and $\mathcal{I}, t'' \models M_1$ for all $t'' \in (t', t)$,
$\mathcal{I}, t \models M_1\mathcal{U}_{\varrho}M_2$	iff	$\mathcal{I}, t' \models M_2$ for some t' with $t' - t \in \varrho$ and $\mathcal{I}, t'' \models M_1$ for all $t'' \in (t, t')$.

A fact $M@q$ is satisfied if M is satisfied at all time points $t \in q$, whereas a rule r is satisfied if, for each of its ground instances r' and for each time point t , if all the body atoms of r' are satisfied at t , then so is the head of r' . As usual, a *model* of a dataset \mathcal{D} or of a program Π is an interpretation which satisfies all facts in \mathcal{D} or all rules in Π , respectively. Program Π and dataset \mathcal{D} are *consistent* if they have a model (note that if none of the rules mentions \perp in the head, then an inconsistency cannot occur—in the absence of negation in the language of DatalogMTL, \perp is the only means for expressing a contradiction), and they *entail* a fact $M@q$ if each model of both Π and \mathcal{D} is a model of $M@q$.

2.3. Complexity of Reasoning

The main reasoning tasks considered in DatalogMTL are *fact entailment* and *consistency checking*. These problems polynomially reduce to the complements of each other [1], so we identify them with *reasoning* in DatalogMTL. As usual, we distinguish between combined and data complexity, where in the latter case the input consists of a dataset only, while programs are considered fixed. Reasoning in DatalogMTL is decidable but of high complexity; in particular, it is ExpSpace-complete for combined complexity [1] and PSpace-complete for data complexity [6]. A number of lower-complexity fragments of DatalogMTL have been introduced, for example *finitely-materialisable* programs with ExpTime-complete combined complexity of reasoning [9], *linear* and *core* fragments with NL- and TC⁰-complete data complexity [10], and the *non-recursive* fragment with AC⁰ data complexity [1]. DatalogMTL has also been extended with non-monotonic negation, which does not increase the complexity for stratifiable programs [11], but leads to undecidability if a program is not stratifiable, unless it is interpreted over the integer timeline [12]. In contrast, in negation-free DatalogMTL reasoning over the integer timeline is as hard as over the rational timeline [8].

3. Approaches Based on Materialisation

We observe that there is a group of reasoning techniques for DatalogMTL whose main component is the materialisation process. This process mimics the fixpoint characterisation of DatalogMTL semantics, as we describe in the first part of this section.

3.1. Fixpoint Characterisation and Basic Materialisation

The fixpoint characterisation is based on the observation that each pair of a consistent program Π and a dataset \mathcal{D} admits a unique least model, which we refer to as their *canonical interpretation* $\mathfrak{C}_{\Pi, \mathcal{D}}$. As in Datalog, we can construct this interpretation by applying rules of Π to \mathcal{D} in a forward-chaining manner until a fixpoint is reached. However, unlike in Datalog, 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.

Formally, we define the *immediate consequence operator* T_{Π} for a program 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 : whenever \mathfrak{I} satisfies each body

Procedure 1: Materialisation-based reasoning

Input: A program Π and a dataset \mathcal{D}
Output: A dataset

```

1  $\mathcal{D}_{\text{new}} := \mathcal{D};$  // initialise  $\mathcal{D}_{\text{new}}$ 
2 repeat
3    $\mathcal{D}_{\text{old}} := \mathcal{D}_{\text{new}};$  // copy  $\mathcal{D}_{\text{new}}$  before applying rules
4    $\mathcal{D}_{\text{new}} :=$  the least dataset representation of  $T_{\Pi}(\mathcal{J}_{\mathcal{D}_{\text{new}}})$ ;
5 until  $\mathcal{D}_{\text{old}} = \mathcal{D}_{\text{new}};$ 
6 return  $\mathcal{D}_{\text{new}};$ 

```

atom of r at a time point t , then $T_{\Pi}(\mathcal{J})$ satisfies the head of r at t . Subsequent applications of T_{Π} to the (least) model $\mathcal{J}_{\mathcal{D}}$ of \mathcal{D} define the following sequence of interpretations, for all ordinals α :

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

If Π and \mathcal{D} are consistent, then the canonical interpretation is obtained after at most ω_1 (i.e., the first uncountable ordinal) applications of the immediate consequence operator; that is, $\mathfrak{C}_{\Pi, \mathcal{D}} = T_{\Pi}^{\omega_1}(\mathcal{J}_{\mathcal{D}})$ [4].

The fixpoint characterisation suggests a naïve materialisation-based reasoning approach presented in Procedure 1; given a program Π and a dataset \mathcal{D} , the procedure applies the immediate consequence operator T_{Π} until no new facts can be derived, that is, a fixpoint is reached.

We have recently implemented this approach in the metric temporal reasoner MeTeoR, and provided an efficient way of computing $T_{\Pi}(\mathcal{J}_{\mathcal{D}_{\text{new}}})$, which allowed us to handle datasets with hundreds of millions of facts [13]. The naïve materialisation procedure was also implemented by rewriting a program into a set of standard SQL queries (with views) [1], which was exploited in the Ontop ontology-based data access reasoning system to answer temporal queries [14].

The advantage of rewriting DatalogMTL programs into SQL is that constructed queries can be evaluated with standard systems such as PostgreSQL or Apache Spark. The downside of this approach, however, is that relying on SQL systems limits the possibility of controlling and optimising computations which are specific to DatalogMTL materialisation. Such materialisation requires, for example, computing numerous *temporal joins*, and the experimental evaluation [13] shows that a handcrafted implementation in MeTeoR of such computations (obtained by first sorting the sets of intervals involved in a join and then linearly scanning these sets to compute the relevant intersections) significantly outperforms the approach based on SQL rewritings (the latter is not optimised for intersecting time intervals).

Clearly, materialisation is a sound reasoning technique, but the procedure is not always terminating, as reaching the fixpoint may require an infinite number of materialisation steps. However, if a fixpoint is reached, then a representation of the full canonical interpretation can be kept in memory, which can be used for an efficient verification of entailment of any fact.

Algorithm 2: Checking finite materialisability for a single dataset

Input: A bounded program Π and a bounded dataset \mathcal{D}
Output: A Boolean value

- 1 $\mathcal{D}_{\text{new}} := \mathcal{D};$
- 2 **repeat**
- 3 $\mathcal{D}_{\text{old}} := \mathcal{D}_{\text{new}};$
- 4 $\mathcal{D}_{\text{new}} :=$ the least dataset representation of $T_{\Pi}(\mathcal{J}_{\mathcal{D}_{\text{new}}});$
- 5 **if** there is $M@_{\varrho} \in \mathcal{D}_{\text{new}}$ with $\varrho \not\subseteq \varrho_{\Pi, \mathcal{D}}$ **then** Return false;
- 6 **until** $\mathcal{D}_{\text{old}} = \mathcal{D}_{\text{new}};$
- 7 **return** true;

3.2. Finite Materialisability

Recent experiments suggest that, when materialisation terminates, it is a very scalable technique for DatalogMTL which is well suited for reasoning with large temporal datasets. Therefore, a natural question arises—for which reasoning instances does naïve materialisation terminate?

This question has recently been studied by introducing the notion of *finitely materialisable* programs. In particular, we can consider a data-dependent and a data-independent variant of this notion as follows. We say that a program Π is finitely materialisable for a dataset \mathcal{D} if, when applied to \mathcal{D} , the immediate consequence operator T_{Π} reaches a fixpoint in finitely many steps. The data-independent notion considers finite materialisability of Π for all datasets.

The problem of checking whether a program is finitely materialisable, as well as the complexity of reasoning in finitely materialisable programs have recently been studied for programs and datasets in which all intervals are bounded (i.e., do not mention infinities as endpoints) [9]. In this setting, it has been shown how to compute, for a program Π which is finitely materialisable for a dataset \mathcal{D} , an interval $\varrho_{\Pi, \mathcal{D}}$ in which are located all the facts entailed by Π and \mathcal{D} . Thus, checking finite materialisability boils down to performing materialisation until a fixpoint is reached or some fact is entailed outside $\varrho_{\Pi, \mathcal{D}}$, as depicted in Algorithm 2 (c.f. Procedure 1).

On the other hand, checking if a program Π is finitely materialisable for all datasets reduces to checking if Π is finitely materialisable for a specific *critical dataset* \mathcal{D}_{Π} consisting of facts $P(\mathbf{s})@[0, \text{depth}(\Pi)]$, for all predicates P occurring in Π , all tuples \mathbf{s} of constants in Π (and one additional fresh constant), and $\text{depth}(\Pi)$ a sufficiently large number depending on the intervals occurring in Π [9]. It turns out that checking data-dependent finite materialisability is PSpace-complete for data and ExpSpace-complete for combined complexity, whereas checking data-independent finite materialisability is ExpTime-complete. Furthermore, sufficient conditions for finite materialisability have been established, which allow for a limited form of temporal recursion and can be checked efficiently. In particular, *MTL-acyclicity* of a program requires that a generalisation of the program’s dependency graph, where edges are labelled with intervals occurring in the rules, does not contain cycles of certain types, and can be checked in NL [9].

Testing finite materialisability is a useful static analysis task, which can be performed offline. If the program is finitely materialisable, then a scalable materialisation-based algorithm with termination guarantees can be used. Reasoning with finitely materialisable programs is also sufficiently easier from the theoretical point of view, as it is ExpTime-complete for combined

complexity, in contrast to ExpSpace-completeness for full DatalogMTL.

3.3. Sliding Windows

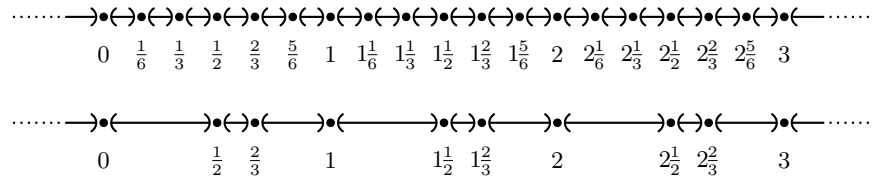
Another reasoning approach in DatalogMTL consists in materialising all the facts within a bounded fragment of the timeline—called a *window*—and then sliding the window by some distance towards the future. The algorithm ‘forgets’ all the facts outside the current window and repeats the process, by materialising all the facts in the current window, and then, sliding it further towards the future. The main advantage of this approach is that it keeps in memory only facts describing a fixed fragment of the timeline. Hence, the approach is particularly useful in the *stream reasoning* setting, where reasoning is performed based on a continuously increasing and potentially infinite stream of input facts, in contrast to a static dataset considered in the standard setting [3].

Like the other materialisation-based techniques for reasoning in DatalogMTL, using sliding windows constitutes a sound approach. Notice, however that, in general, materialisation based only on the information kept within a bounded window may lead to the incompleteness of reasoning. Nevertheless, completeness can be retained for *forward-propagating* programs, where rules can propagate information only towards the future, and which are a natural choice in the stream reasoning setting [3]. Moreover, it is worth noting that even though a sliding window is of a bounded length, it can contain an unbounded number of facts since DatalogMTL is interpreted over the rational timeline. One way to bound the memory usage is to disallow punctual intervals (i.e., containing a single time point) in a program which, in turn, allows for keeping in memory a succinct bounded-size representation of facts in a window [3].

4. Approaches Based on Discretisation of Time

4.1. Time Discretisation

Although the timeline in DatalogMTL is dense, it can be divided into regularly distributed intervals which are uniform in the sense that all time points belonging to the same interval satisfy exactly the same relational atoms. This observation was first exploited to partition the rational timeline, for a program Π and dataset \mathcal{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 \mathcal{D} [1]. Later, an alternative partitioning of the timeline was proposed [6], 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 \mathcal{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 exemplary partitionings of the timeline into intervals stemming from both discretisation methods, for the case where the only rational numbers occurring in \mathcal{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 main advantage of the second partitioning is that gcd is computed independently of \mathcal{D} , which was used in devising reasoning techniques with a better computational behaviour for data complexity [6].

4.2. Translation to LTL

Discretisation of time was exploited by Brandt et al. [1] to reduce reasoning in DatalogMTL to reasoning in linear temporal logic (LTL). The reduction consists in transforming a program Π and a dataset \mathcal{D} into an LTL formula $\varphi_{\Pi, \mathcal{D}}$ such that Π and \mathcal{D} are consistent if and only if $\varphi_{\Pi, \mathcal{D}}$ is LTL-satisfiable (over the integer timeline).

In its basic variant, LTL is a propositional modal logic interpreted over the ordered set of natural numbers, whose language involves boolean connectives and temporal operators \bigcirc_P for *at the previous time point*, \square_P for *always in the past*, \bigcirc_F for *at the next time point*, and \square_F for *always in the future*. An LTL formula φ is satisfiable if there exists an LTL model in which φ is satisfied at 0.

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^c . Moreover, since LTL does not allow for metric operators, the binary-encoded MTL operators occurring in Π need to be rewritten to basic LTL operators. Note that both the grounding of the initial program Π and then expanding the binary-encoded numbers involved in an MTL operator into sequences of LTL operators lead to an exponential blow-up. For example, assume that Π mentions an atom $\exists_{[0,60)} A(x)$. Moreover, let Π and \mathcal{D} mention 100 constants c_1, \dots, c_{100} and assume that after a discretisation of the timeline the interval $[0, 60)$ contains 600 intervals. Then $\exists_{[0,60)} A(x)$ is translated to an LTL-formula containing 100 conjuncts (one conjunct for each $c_i \in \{c_1, \dots, c_{100}\}$), each of the form

$$A(c_i) \wedge \bigcirc_P A(c_i) \wedge \bigcirc_P \bigcirc_P A(c_i) \wedge \dots \wedge \underbrace{\bigcirc_P \dots \bigcirc_P}_{599} A(c_i).$$

Consequently, $\varphi_{\Pi, \mathcal{D}}$ is exponentially large. Since satisfiability checking in LTL is PSpace-complete, this approach provides an ExpSpace reasoning procedure for DatalogMTL, which is worst-case optimal. Although it allows for using optimised off-the-shelf reasoning systems for LTL, it turns out that due to the exponential grounding of a program and the encoding of MTL operators, the approach is inefficient in practice [15]. Indeed, a recent implementation of the translation-based approach has been outperformed by the MeTeoR system by a considerable margin in a series of performed tests.

4.3. Automata-Based Techniques

Another method of reasoning in DatalogMTL was obtained by directly applying to DatalogMTL techniques known for LTL, instead of translating the former to the latter. In particular, one of the main approaches to checking satisfiability of an LTL-formula φ consists in constructing a generalised non-deterministic Büchi automaton whose states are sets of formulas relevant for φ , alphabet consists of sets of propositions, whereas transition relation and accepting conditions ensure that words accepted by the automaton are exactly models of φ [16].

To adapt this technique to the needs of DatalogMTL, the automaton was modified as follows [10]. Each state of the automaton now represents formulas (ground metric atoms) which hold not in a single time point but in all time points belonging to a fragment of the timeline called a *window* (c.f. the sliding windows technique from Section 3.3). Note that 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 $\boxplus_{[0,\infty)}A$ —stating 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 . The rest of the automaton is defined similarly to the way we do it in LTL, namely the alphabet consists of sets of ground relational atoms, whereas the transition relation and accepting conditions are analogous to those in the automaton for LTL.

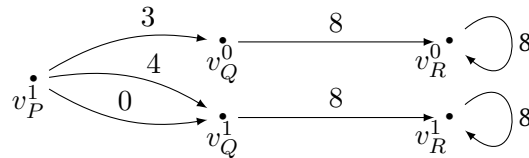
It was shown that consistency checking in DatalogMTL reduces to checking non-emptiness of (pairs of) the above-described automata, and that the latter is feasible in PSpace [10]. Indeed, it was shown that states of the automata are polynomially large in the size of the dataset. In particular, windows can be chosen so that, after the timeline discretisation, the number of intervals in each window is polynomially large in the size of the dataset—to obtain this property it is crucial to use the second of the time discretisation methods from Section 4.1. Moreover, the number of ground metric atoms that can hold in each of these intervals is also polynomially bounded. Hence, each state is polynomially large and non-emptiness of the automata can be checked with the standard on-the-fly approach [16] in PSpace.

This approach provides a worst-case optimal reasoning approach for full DatalogMTL, which was implemented as part of the MeTeoR system [13] and used for reasoning in cases where materialisation does not terminate. With suitable optimisations, automata construction is feasible in practice for inputs of moderate size and performs better than the translation to LTL, but its application to large-scale datasets remains problematic. The automata technique was also exploited to establish reasoning procedures in extensions of DatalogMTL with non-monotonic negation; both in the case of stratifiable [11] and general programs [12].

4.4. Arithmetic Progressions

Discretisation of time proved also useful in establishing low-complexity fragments of DatalogMTL. In particular, it allowed for constructing specific reasoning techniques for fragments in which \diamond is the only MTL operator occurring in a program, and moreover, each rule is *core*, that is, has at most one body atom ($\text{DatalogMTL}_{core}^{\diamond}$) or each rule is *linear* in the sense that it can mention at most one intensional (IDB) body atom ($\text{DatalogMTL}_{lin}^{\diamond}$).

The discussed reasoning technique for $\text{DatalogMTL}_{core}^{\diamond}$ is based on two main observations. First, since there are no conjunctions in rule bodies nor box operators (which simulate conjunctions), all derivations in $\text{DatalogMTL}_{core}^{\diamond}$ can be seen as sequences of facts, where each fact is derived only based on the previous fact in the sequence. Secondly, although the second type of the timeline discretisation from Section 4.1 yields intervals of different length, there are polynomially many *types* of intervals (c.f. the figure from Section 4.1 with four types of intervals). These observations allow us to represent all derivations as paths in a *temporal dependency graph* [10]. To illustrate, consider the following graph:



The graph shows that, for example, P holding at some interval ϱ of type 1 (vertex v_P^1) allows us to derive Q in the interval of type 0 (vertex v_Q^0) which is located 3 intervals (the weight of the edge from v_P^1 to v_Q^0) to the right from ϱ . Lengths of paths (i.e., derivations) connecting any pair of vertices in such a graph can be succinctly represented with a set of arithmetic progressions after transforming the initial graph into the Chrobak normal form for automata [17]. This makes it possible to check entailment in TC^0 for data complexity [10].

This reasoning technique can be extended to $\text{DatalogMTL}_{lin}^{\diamond}$. The structure of $\text{DatalogMTL}_{lin}^{\diamond}$ programs allows for decomposing the timeline into (polynomially many) sections in which reasoning can be performed with the technique established for $\text{DatalogMTL}_{core}^{\diamond}$. The existence of these sections, however, leads to an increase of the data complexity from TC^0 to NL.

Although fact entailment in both $\text{DatalogMTL}_{core}^{\diamond}$ and $\text{DatalogMTL}_{lin}^{\diamond}$ is tractable in data complexity, and the discussed algorithms are optimal, to the best of our knowledge they have not yet been implemented.

5. Ongoing and Future Research Directions

A promising direction of future work, thrusting from the ongoing research on reasoning techniques for DatalogMTL , stems from the observation that the canonical model of a program Π and a dataset \mathcal{D} has a *periodic structure* (which corresponds to *ultimately periodic* LTL models). In particular, by the translation to LTL and reasoning techniques that rely on Büchi automata, it follows that there exists a left period p_1 and a right period p_2 such that for all (sufficiently small) time points t , the same relational atoms hold in $\mathfrak{C}_{\Pi, \mathcal{D}}$ at t and $t - p_1$, whereas for all (sufficiently large) t , the same relational atoms hold in $\mathfrak{C}_{\Pi, \mathcal{D}}$ at t and $t + p_2$. If we knew p_1 and p_2 in advance, we could use them to significantly speed up the materialisation process, and potentially guarantee its termination for of all programs. The main problem, however, is that although an (impractically large) upper bound on the length of the periods can be determined [18], so far no efficient algorithm for computing values of these periods has been proposed. Some preliminary results in this direction have recently been established by Bellomarini et al. [19],

but computing periods in advance seems to be a computationally challenging task. Instead, we are currently working on an approach that computes periods from a partial materialisation; in particular, we aim to construct an algorithm that performs a bounded number of materialisation steps until the obtained materialisation satisfies specific properties that allow us to read periods from this partial materialisation.

Another interesting direction focuses on optimising materialisation, e.g., by developing *semi naïve* and *parallel* rule evaluation strategies and incremental *materialisation maintenance* techniques.

Acknowledgements

This work has been supported by the EPSRC projects OASIS (EP/S032347/1), AnaLOG (EP/P025943/1), and UK FIRES (EP/S019111/1), the SIRIUS Centre for Scalable Data Access, and Samsung Research UK. For the purpose of Open Access, the authors have applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

- [1] S. Brandt, E. G. Kalaycı, V. Ryzhikov, G. Xiao, M. Zakharyashev, Querying log data with metric temporal logic, *J. Artif. Intell. Res.* 62 (2018) 829–877. doi:10.1613/jair.1.11229.
- [2] R. Koymans, Specifying real-time properties with metric temporal logic, *Real-Time Syst.* 2 (1990) 255–299. doi:10.1007/BF01995674.
- [3] P. A. Wałęga, M. Kaminski, B. Cuenca Grau, Reasoning over streaming data in metric temporal Datalog, in: *Proc. of AAI*, 2019, pp. 3092–3099. doi:10.1609/aaai.v33i01.33013092.
- [4] S. Brandt, E. G. Kalaycı, R. Kontchakov, V. Ryzhikov, G. Xiao, M. Zakharyashev, Ontology-based data access with a Horn fragment of metric temporal logic, in: *Proc. of AAI*, 2017, pp. 1070–1076. doi:10.1609/aaai.v31i1.10696.
- [5] M. Nissl, E. Sallinger, Modelling smart contracts with DatalogMTL, in: M. Ramanath, T. Palpanas (Eds.), *Proc. of EDBT/ICDT*, 2022. URL: http://ceur-ws.org/Vol-3135/EcoFinKG_2022_paper4.pdf.
- [6] P. A. Wałęga, B. Cuenca Grau, M. Kaminski, E. V. Kostylev, DatalogMTL: Computational complexity and expressive power, in: *Proc. of IJCAI*, 2019, pp. 1886–1892. doi:10.24963/ijcai.2019/261.
- [7] V. Ryzhikov, P. A. Wałęga, M. Zakharyashev, Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics, in: *Proc. of IJCAI*, 2019, pp. 1851–1857. doi:10.24963/ijcai.2019/256.
- [8] P. A. Wałęga, B. Cuenca Grau, M. Kaminski, E. V. Kostylev, DatalogMTL over the integer timeline, in: *Proc. of KR*, 2020, pp. 768–777. doi:10.24963/kr.2020/79.
- [9] P. A. Wałęga, M. Zawidzki, B. Cuenca Grau, Finitely materialisable Datalog programs with metric temporal operators, in: *Proc. of KR*, 2021, pp. 619–628. doi:10.24963/kr.2021/59.

- [10] P. A. Wałęga, B. Cuenca Grau, M. Kaminski, E. V. Kostylev, Tractable fragments of Datalog with metric temporal operators, in: Proc. of IJCAI, 2020, pp. 1919–1925. doi:10.24963/ijcai.2020/266.
- [11] D. J. Tena Cucala, P. A. Wałęga, B. Cuenca Grau, E. V. Kostylev, Stratified negation in Datalog with metric temporal operators, in: Proc. of AAI, 2021, pp. 6488–6495. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16804/16611>.
- [12] P. A. Wałęga, D. J. Tena Cucala, E. V. Kostylev, B. Cuenca Grau, DatalogMTL with negation under stable models semantics, in: Proc. of KR, 2021, pp. 609–618. doi:10.24963/kr.2021/58.
- [13] D. Wang, P. Hu, P. A. Wałęga, B. Cuenca Grau, MeTeoR: Practical reasoning in Datalog with metric temporal operators, in: Proc. of AAI, 2022, pp. 5906–5913.
- [14] E. G. Kalayci, S. Brandt, D. Calvanese, V. Ryzhikov, G. Xiao, M. Zakharyashev, Ontology-based access to temporal data with Ontop: A framework proposal, *Int. J. Appl. Math.* 29 (2019) 17–30. doi:10.2478/amcs-2019-0002.
- [15] J. Yang, Translation of DatalogMTL Into PLTL, Technical Report, Department of Computer Science, University of Oxford, 2022.
- [16] C. Baier, J.-P. Katoen, Principles of model checking, MIT Press, 2008.
- [17] M. Chrobak, Finite automata and unary languages, *Theor. Comput. Sci.* 47 (1986) 149–158. doi:10.1016/0304-3975(86)90142-8.
- [18] A. Artale, R. Kontchakov, A. Kovtunova, V. Ryzhikov, F. Wolter, M. Zakharyashev, First-order rewritability of ontology-mediated queries in linear temporal logic, *Artif. Intell.* 299 (2021) 103536. doi:10.1016/j.artint.2021.103536.
- [19] L. Bellomarini, M. Nissl, E. Sallinger, Query evaluation in DatalogMTL – taming infinite query results, *CoRR abs/2109.10691* (2021). arXiv:2109.10691.