

MeTeoR: Practical Reasoning in Datalog with Metric Temporal Operators

Dingmin Wang¹, Pan Hu^{1,2}, Przemysław Andrzej Wałęga¹, Bernardo Cuenca Grau¹

¹Department of Computer Science, University of Oxford, UK

²School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
{dingmin.wang, przemyslaw.walega, bernardo.cuenca.grau}@cs.ox.ac.uk, pan.hu@sjtu.edu.cn

Abstract

DatalogMTL is an extension of Datalog with operators from metric temporal logic which has received significant attention in recent years. It is a highly expressive knowledge representation language that is well-suited for applications in temporal ontology-based query answering and stream processing. Reasoning in DatalogMTL is, however, of high computational complexity, making implementation challenging and hindering its adoption in applications. In this paper, we present a novel approach for practical reasoning in DatalogMTL which combines materialisation (a.k.a. forward chaining) with automata-based techniques. We have implemented this approach in a reasoner called MeTeoR and evaluated its performance using a temporal extension of the Lehigh University Benchmark and a benchmark based on real-world meteorological data. Our experiments show that MeTeoR is a scalable system which enables reasoning over complex temporal rules and datasets involving tens of millions of temporal facts.

Introduction

Temporal data is ubiquitous in many application scenarios, such as stock trading (Nuti et al. 2011), network flow anomaly detection (Munz and Carle 2007), and equipment malfunction monitoring (Doherty, Kvarnström, and Heintz 2009). In order to represent knowledge and subsequently reason in the presence of such temporal data, Brandt et al. (2018) proposed DatalogMTL—an extension of Datalog (Ceri, Gottlob, and Tanca 1989) with operators from metric temporal logic (Koymans 1990) interpreted over the rational timeline. DatalogMTL is a powerful KR language, which has 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, Kaminski, and Cuenca Grau 2019). For example, the following DatalogMTL rule can be used to analyse equipment data:

$$\exists_{[0,1]} \text{ExcHeat}(x) \leftarrow \exists_{[0,1]} \text{Temp24}(x) \wedge \Diamond_{[0,1]} \text{Temp41}(x).$$

The rule states that a device x has been under excessive heat continuously within a past interval of length 1 ($\exists_{[0,1]}$) if the temperature recorded in this interval was always above 24 degrees (Temp24) and also, at some point in this interval ($\Diamond_{[0,1]}$), the temperature was above 41 degrees (Temp41).

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. 2019), which makes reasoning in practice challenging. Thus, research has recently focused on establishing a suitable trade-off between expressive power and complexity of reasoning, by identifying lower complexity fragments of DatalogMTL (Wałęga et al. 2020b; Wałęga, Zawidzki, and Cuenca Grau 2021) as well as studying alternative semantics with more favourable computational behaviour (Wałęga et al. 2020a; Ryzhikov, Wałęga, and Zakharyashev 2019).

The design and implementation of practical reasoning algorithms for the full DatalogMTL language, however, remains a largely unexplored area—something that has prevented its widespread adoption in applications. In particular, the only implementation we are aware of is the prototype by Brandt et al. (2018), which is limited to non-recursive DatalogMTL programs. This is in stark contrast with plain Datalog, for which a plethora of systems have been developed and successfully deployed in practice (Motik et al. 2014; Carral et al. 2019; Bellomarini, Sallinger, and Gottlob 2018).

In this paper, we present the first practical reasoning algorithm for the full DatalogMTL language, which combines materialisation (a.k.a. forward chaining) and automata-based reasoning. On the one hand, materialisation is the reasoning paradigm adopted in most Datalog systems (Bry et al. 2007); to check fact entailment in this setting, one first computes in a forward chaining manner all facts logically entailed by the input program and dataset, and then verifies whether the input fact is included amongst the entailed facts. A direct implementation of materialisation-based reasoning in DatalogMTL is, however, problematic since forward chaining may require infinitely many rounds of rule applications (Wałęga, Zawidzki, and Cuenca Grau 2021). On the other hand, Wałęga et al. (2019) introduced a decision procedure for DatalogMTL which relies on constructing Büchi automata and checking non-emptiness of their languages. This procedure has been introduced 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 new algorithm deals with these difficulties by providing an effective way of combining the scalability of materialisation-based reasoning and the completeness guar-

anted by automata-based procedures, thus bringing together ‘the best of both worlds’. To achieve this, our algorithm aims at minimising reasoning workload by resorting to materialisation-based reasoning whenever possible while minimising the use of automata. Furthermore, we propose a suite of optimisation techniques aimed at reducing the workload involved in rule application during materialisation-based reasoning, as well as in Büchi automata construction and the subsequent non-emptiness checks.

We have implemented our approach in a new reasoner called MeTeoR (<https://meteor.cs.ox.ac.uk>), which supports fact entailment over arbitrary (i.e., potentially recursive) DatalogMTL programs and large-scale temporal datasets. We have evaluated the performance of our reasoner on an extension of the Lehigh University Benchmark (Guo, Pan, and Heflin 2005) with temporal rules and data, as well as on a benchmark based on a real-world meteorological dataset (Maurer et al. 2002). Our results show that MeTeoR is a scalable system that can successfully reason over complex recursive programs and datasets including tens of millions of temporal facts; furthermore, it consistently outperformed the approach by Brandt et al. (2018) when reasoning over non-recursive programs.

Preliminaries

We recapitulate the standard definition of DatalogMTL, interpreted with the standard continuous semantics over the rational timeline (Brandt et al. 2018).

Syntax. A *relational atom* is a function-free first-order atom of the form $P(s)$, with P a predicate and s a tuple of *terms*. A *metric atom* is an expression given by the following grammar, where $P(s)$ is a relational atom, and $\diamond, \oplus, \boxplus, \boxminus, S, \mathcal{U}$ are MTL operators indexed with positive rational intervals ϱ (i.e., containing only non-negative rationals):

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

A *rule* is an expression of the form of

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

with each M_i a metric atom, and M' a metric atom not mentioning operators \diamond, \oplus, S , and \mathcal{U} ; metric atom M' is the rule’s *head* and the conjunction $M_1 \wedge \dots \wedge M_n$ is its *body*. A rule is *safe* if each variable in its head also occurs in its body; a *program* is a finite set of safe rules. An expression (metric atom, rule, etc.) is *ground* if it mentions no variables. A *fact* is an expression $M@_{\varrho}$ with M a ground relational atom and ϱ a rational interval; a *dataset* is a finite set of facts. The *coalescing* of facts $M@_{\varrho_1}$ and $M@_{\varrho_2}$, with ϱ_1 and ϱ_2 intervals that are either adjacent or have a non-empty intersection, is the fact $M@_{\varrho_3}$ where ϱ_3 is the union of ϱ_1 and ϱ_2 . The *grounding* $\text{ground}(\Pi, \mathcal{D})$ of a program Π with respect to a dataset \mathcal{D} is a set of all ground rules obtained by assigning constants from Π and \mathcal{D} to variables in Π .

The *dependency graph* of a program 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

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

Table 1: Semantics of ground metric atoms

Q in the body and R in the head. Program 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). Furthermore, for a predicate P , a rule r is *P-relevant* in Π if there exists a rule r' in Π mentioning P or \perp 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' . Intuitively, *P-relevant* rules may be used for deriving facts about P or \perp (i.e., inconsistency).

Semantics. An *interpretation* \mathfrak{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 $\mathfrak{I}, t \models M$. This extends to metric atoms with MTL operators as shown in Table 1. An interpretation \mathfrak{I} satisfies a fact $M@_{\varrho}$ if $\mathfrak{I}, t \models M$ for all $t \in \varrho$. An 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 . An interpretation \mathfrak{I} satisfies a (non-ground) rule r if it satisfies each ground instance of r . An interpretation \mathfrak{I} is a *model* of a program Π if it satisfies each rule in Π , and it is a *model* of a dataset \mathcal{D} if it satisfies each fact in \mathcal{D} . A program Π and a dataset \mathcal{D} are *consistent* if they have a model, and they *entail* a fact $M@_{\varrho}$ if each model of both Π and \mathcal{D} is also a model of $M@_{\varrho}$. Each dataset \mathcal{D} has the least model $\mathfrak{I}_{\mathcal{D}}$, and we say that dataset \mathcal{D} *represents* interpretation $\mathfrak{I}_{\mathcal{D}}$.

Canonical Interpretation. The *immediate consequence operator* T_{Π} for a program Π is a function mapping an 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 atom of r at time point t , then $T_{\Pi}(\mathfrak{I})$ satisfies the head of r at t . The successive application of T_{Π} to $\mathfrak{I}_{\mathcal{D}}$ defines a transfinite sequence of interpretations $T_{\Pi}^{\alpha}(\mathfrak{I}_{\mathcal{D}})$ for ordinals α as follows: (i) $T_{\Pi}^0(\mathfrak{I}_{\mathcal{D}}) = \mathfrak{I}_{\mathcal{D}}$, (ii) $T_{\Pi}^{\alpha+1}(\mathfrak{I}_{\mathcal{D}}) = T_{\Pi}(T_{\Pi}^{\alpha}(\mathfrak{I}_{\mathcal{D}}))$ for α an ordinal, and (iii) $T_{\Pi}^{\alpha}(\mathfrak{I}_{\mathcal{D}}) = \bigcup_{\beta < \alpha} T_{\Pi}^{\beta}(\mathfrak{I}_{\mathcal{D}})$ for α a limit ordinal. The *canonical interpretation* $\mathfrak{C}_{\Pi, \mathcal{D}}$ of Π and \mathcal{D} is the interpretation $T_{\Pi}^{\omega_1}(\mathfrak{I}_{\mathcal{D}})$, with ω_1 the first uncountable ordinal. If Π and \mathcal{D} have a model, the canonical interpretation $\mathfrak{C}_{\Pi, \mathcal{D}}$ is the least model of Π and \mathcal{D} (Brandt et al. 2018).

Reasoning. The main reasoning tasks in DatalogMTL are *fact entailment* and *consistency checking*. The former is to check whether a program and dataset entail a given fact, and the latter is to check whether a program and dataset are consistent. These problems are reducible to the complement of each other; both of them are ExpSpace-complete (Brandt et al. 2018) and PSpace-complete in data complexity (i.e. with respect to the size of a dataset) (Wałęga et al. 2019).

Materialisation vs. Automata-based Reasoning

Materialisation (a.k.a. forward chaining) is a standard reasoning technique used in scalable implementations of plain Datalog, which consists of successive rounds of rule applications to compute all consequences of an input program and dataset (Motik et al. 2019; Bry et al. 2007). The resulting set of facts represents the canonical model over which all queries can be answered directly. As in plain Datalog, each consistent pair of a DatalogMTL program Π and a dataset \mathcal{D} admits a canonical model $\mathfrak{C}_{\Pi, \mathcal{D}}$ defined as the least fixpoint of the immediate consequence operator T_{Π} capturing a single round of rule applications. The use of metric operators in rules, however, introduces two main practical difficulties. First, DatalogMTL interpretations are intrinsically infinite and hence a materialisation-based algorithm must be able to finitely represent (e.g., as a dataset) the result of each individual round of rule applications. Second, reaching the least fixpoint of the immediate consequence operator in DatalogMTL may require an infinite number of rule applications; hence, a direct implementation of materialisation-based reasoning is non-terminating. In contrast, materialisation is guaranteed to terminate for non-recursive programs, thus providing a decision procedure (Brandt et al. 2018).

Wałęga et al. (2019) provided an automata-based decision procedure for consistency checking applicable to unrestricted DatalogMTL programs. This approach relies on reducing consistency checking of an input program Π and dataset \mathcal{D} to checking non-emptiness of two Büchi automata. These automata are responsible for accepting parts of a model located, respectively, to the left of the least number mentioned in \mathcal{D} , and to the right of the greatest number in \mathcal{D} . Thus, Π and \mathcal{D} have a model if and only if both automata have non-empty languages. States of these automata are polynomially representable in the size of \mathcal{D} , so the reduction yields a PSpace data complexity bound for consistency checking. The idea behind the reduction is based on splitting a model witnessing consistency of Π and \mathcal{D} into fragments corresponding to segments of the timeline; automata states describe metric atoms holding in such segments and the transition functions guarantee that atoms from successive fragments do not violate the semantics of metric operators. Although this automata-based approach provides tight complexity bounds, it does not yield a practical decision procedure since the automata contain a large (exponential in the size of \mathcal{D}) number of states.

A Practical Decision Procedure

In this section, we present our approach to practical reasoning in the full DatalogMTL language. Our algorithm decides fact entailment, that is, checks whether a given fact is entailed by

a given program and dataset. To this end, we optimise and combine materialisation—which is a scalable technique, but is not guaranteed to terminate—and the automata-based approach—which is terminating in general, but less efficient in practice. The key novelties of our approach are as follows:

- an optimised implementation of the materialisation approach, which aims at applying the immediate consequence operator efficiently and storing a succinct representation of the fragment of the canonical interpretation constructed thus far;
- an optimised implementation of the automata-based reasoning approach of Wałęga et al. (2019), which aims at minimising the size of the automata; and
- an effective way of combining materialisation with automata-based reasoning that aims at reducing reasoning workload and achieving early termination.

In the remainder of this section we will discuss each of these components in detail.

Optimising Materialisation

The key component of materialisation-based reasoning is an effective implementation of the immediate consequence operator capturing a single round of rule applications. Our implementation takes as input a program Π and a dataset \mathcal{D} and computes a dataset \mathcal{D}' representing the interpretation $T_{\Pi}(\mathfrak{I}_{\mathcal{D}})$ obtained by application of T_{Π} to the least model of \mathcal{D} . Thus, it addresses one of the difficulties associated to materialisation-based reasoning by ensuring that the result of a single round of rule applications is a finite object.

To facilitate the presentation of our rule application procedure, we first briefly discuss details regarding the representation and storage of datasets. 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 this ground relational atom. Moreover, each ground relational atom is 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 fact $M @ \varrho$ is entailed by dataset \mathcal{D} , it suffices to find ϱ' such that $M @ \varrho' \in \mathcal{D}$ and $\varrho \subseteq \varrho'$; 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.

We perform a single round of rule applications with a procedure `ApplyRules` presented in Algorithm 1. To construct a representation of $T_{\Pi}(\mathfrak{I}_{\mathcal{D}})$, for an input program Π and a dataset \mathcal{D} , our algorithm performs two main steps. The first step is performed for each ground rule in $\text{ground}(\Pi, \mathcal{D})$ (Line 2), where rule grounding is performed using a standard index nested loop join algorithm (Garcia-Molina, Ullman, and Widom 2009) since facts in \mathcal{D} are indexed. Then, the first step consists of extending \mathcal{D} with all facts that can be derived by applying r to \mathcal{D} (we also add \perp to \mathcal{D} if some rule leads to inconsistency). Hence, the new dataset \mathcal{D}' represents the interpretation $T_{\{r\}}(\mathfrak{I}_{\mathcal{D}})$ (Line 3). To implement the

Algorithm 1: ApplyRules

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

```
1  $\mathcal{D}' := \mathcal{D}$ ;  
2 for each rule  $r \in \text{ground}(\Pi, \mathcal{D})$  do  
3    $\mathcal{D}' := \text{representation of } \mathfrak{I}_{\mathcal{D}'} \cup T_{\{r\}}(\mathfrak{I}_{\mathcal{D}})$ ;  
4 for each ground metric atom  $M$  appearing in  $\mathcal{D}'$  do  
5   Retrieve all  $M@_{\varrho_1}, \dots, M@_{\varrho_n}$  occurring in  $\mathcal{D}'$ ;  
6   Merge  $\varrho_1, \dots, \varrho_n$  into  $\varrho'_1, \dots, \varrho'_m$ ;  
7   Replace  $M@_{\varrho_1}, \dots, M@_{\varrho_n}$  with  
      $M@_{\varrho'_1}, \dots, M@_{\varrho'_m}$  in  $\mathcal{D}'$ ;  
8 Return  $\mathcal{D}'$ ;
```

first step, we modified the completion rules for normalised programs defined by Brandt et al. (2018, Section 3)—our approach is suitable for arbitrary DatalogMTL rules and thus not only for rules in the aforementioned normal form, which disallows nested MTL operators, the use of \boxplus and \boxminus in rule bodies, and the use of \boxplus and \boxminus in rule heads (Brandt et al. 2018). Since our approach does not require program normalisation as a pre-processing step, we avoid the computation and storage of auxiliary predicates introduced by the normalisation. Furthermore we implement an optimised version of (temporal) joins that is required to evaluate rules with several body atoms. A naïve implementation of the join of metric atoms M_1, \dots, M_n occurring in the body of a rule would require computing all intersections of intervals $\varrho_1, \dots, \varrho_n$ such that $M_i@_{\varrho_i}$ occurs in the so-far constructed dataset, for each $i \in \{1, \dots, n\}$. 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 sort-merge join algorithm: we first sort all n lists of intervals corresponding to M_1, \dots, 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.

The second step of Algorithm 1 consists of coalescing facts in the so-far constructed dataset (Lines 4–7). Conceptually, whenever there exist facts $M@_{\varrho}$ and $M@_{\varrho'}$ in \mathcal{D}' such that ϱ and ϱ' can be coalesced, we replace them with $M@_{\varrho \cup \varrho'}$. To achieve this in practice, for each ground relational atom occurring in \mathcal{D}' we iterate through the corresponding sorted list of intervals and merge them as needed (Lines 5–6). Replacing intervals with coalesced ones reduces memory usage while preventing redundant computations in subsequent rounds of rule applications. Finally, in Line 8, Algorithm 1 returns a dataset, which represents $T_{\Pi}(\mathfrak{I}_{\mathcal{D}})$.

Optimising Automata-based Reasoning

We have implemented an optimised variant of the automata-based approach for checking consistency of a program Π and dataset \mathcal{D} introduced by Wałęga et al. (2019), which is based on verifying non-emptiness of two Büchi automata. States of these automata are of size polynomial in \mathcal{D} and exponential in the combined size of Π and \mathcal{D} , which makes automata

construction and the non-emptiness checks hard in practice. To address this difficulty, our implementation introduces a number of optimisations and we next highlight two of them.

First, instead of directly checking consistency of Π and \mathcal{D} , our implementation checks consistency of Π and the ‘relevant’ part of \mathcal{D} , namely the subset \mathcal{D}' of facts in \mathcal{D} mentioning predicates occurring in the bodies of rules in Π . This optimisation is based on the straightforward observation that Π and \mathcal{D} are consistent if and only if Π and \mathcal{D}' are. In practice, \mathcal{D}' can be significantly smaller than \mathcal{D} , with the subsequent reduction in the size of the constructed automata.

Second, we have optimised the construction of states of the automata when searching for accepting runs. Since automata states represent fragments of a model, we can exploit the semantics of MTL operators to restrict possible locations of metric atoms holding in the same state. For example, if $\boxplus_{[0, \infty)} P$ holds at a time point t , then it needs to hold in all time points greater than t ; similarly, if $\boxminus_{[0, \infty)} P$ holds at t , then it needs to hold at all time points smaller than t , and analogous statements hold for metric atoms with past operators \boxplus and \boxminus . We have incorporated a suite of such restrictions, which resulted in a more efficient construction of automata states.

Combining Materialisation and Automata

Our approach aims at minimising the reasoning workload by considering only relevant parts of the input during reasoning, and by resorting to materialisation-based reasoning whenever possible while minimising the use of automata.

Given as input a program Π , a dataset \mathcal{D} , and a fact $P(s)@_{\varrho}$, our procedure returns a truth value stating whether Π and \mathcal{D} entail $P(s)@_{\varrho}$. To this end, we proceed as summarised in Algorithm 2. The algorithm starts by checking whether $P(s)@_{\varrho}$ is already entailed by \mathcal{D} alone, in which case True is returned in Line 1. As already discussed, this check can be realised very efficiently using suitable indexes.

In Line 2, we identify (using the program’s dependency graph G_{Π}) the subset Π^P of P -relevant rules in Π , that can be potentially used to derive $P(s)@_{\varrho}$ (or \perp , i.e., inconsistency). This allows us to disregard a potentially large number of irrelevant rules during reasoning, optimise automata construction, and also identify cases where materialisation naturally terminates and automata-based reasoning is not required. In particular, if Π^P is non-recursive, fact entailment can be decided using only materialisation (Lines 3–8), namely rules of Π^P are applied (Line 5) until entailment of $P(s)@_{\varrho}$ is detected (Line 6) or a fixpoint of the immediate consequence operator is reached (Line 7).

Even if Π^P is recursive, we can still benefit from using materialisation. For this, we start by performing a ‘pre-materialisation’ step (Lines 10–16), where we materialise until all facts mentioning non-recursive predicates in Π^P have been derived, in which case we break the pre-materialisation loop and continue to the next stage. Upon completion of the pre-materialisation step, we run two threads in parallel, where the first thread continues applying materialisation on a best-effort basis (Lines 18–23) and the second thread resorts to automata (Lines 24–26); the algorithm terminates as soon as one of the threads generates an output truth value.

Algorithm 2: Practical fact entailment

Input: A program Π , a dataset \mathcal{D} , and a fact $P(s)@q$ **Output:** A truth value

```
1 if  $\mathcal{D} \models P(s)@q$  then Return True;
2  $\Pi^P :=$  the set of all  $P$ -relevant rules in  $\Pi$ ;
3 if  $\Pi^P$  is non-recursive then
4   loop
5      $\mathcal{D}' := \text{ApplyRules}(\Pi^P, \mathcal{D})$ ;
6     if  $\mathcal{D}' \models P(s)@q$  then Return True;
7     if  $\mathcal{D}' = \mathcal{D}$  then Return False;
8      $\mathcal{D} := \mathcal{D}'$ ;
9 else
10  loop
11     $\mathcal{D}' := \text{ApplyRules}(\Pi^P, \mathcal{D})$ ;
12    if  $\mathcal{D}' \models P(s)@q$  then Return True;
13    if  $\mathcal{D}' = \mathcal{D}$  then Return False;
14     $\mathcal{D} := \mathcal{D}'$ ;
15    if all facts with non-recursive predicates are
      derived then break;
16   $\mathcal{D}^{pre} := \mathcal{D}$ ;
17  run two threads in parallel:
18  Thread 1:
19  loop
20     $\mathcal{D}' := \text{ApplyRules}(\Pi^P, \mathcal{D})$ ;
21    if  $\mathcal{D}' \models P(s)@q$  then Return True;
22    if  $\mathcal{D}' = \mathcal{D}$  then Return False;
23     $\mathcal{D} := \mathcal{D}'$ ;
24  Thread 2:
25   $\Pi', \mathcal{D}' := \text{EntailToInconsist}(\Pi^P, \mathcal{D}^{pre}, P(s)@q)$ ;
26  Return negation of AutomataProcedure( $\Pi', \mathcal{D}'$ );
```

The second thread reduces fact entailment to inconsistency of a new program Π' and dataset \mathcal{D}' using the reduction from the literature (Brandt et al. 2018; Wałęga et al. 2019) and then runs the automata-based procedure for consistency checking described in the previous section. Our reduction from entailment to inconsistency (Line 25) uses the set of relevant rules Π^P instead of Π , which reduces the size of the automata; furthermore we use the pre-materialised dataset \mathcal{D}^{pre} instead of the input dataset \mathcal{D} to optimise the non-emptiness tests for the constructed automata. Indeed, with \mathcal{D}^{pre} we can determine entailment of all facts with non-recursive predicates, and this allows us to reduce the search space when checking existence of accepting runs of the automata.

Algorithm 2 is clearly sound and complete. Furthermore, it is designed so that, on the one hand, the majority of entailment tests are handled via materialisation and, on the other hand, the recursive program passed on as input to the automata-based approach is small. These features of the algorithm will be empirically confirmed by our experiments described in the following section.

Implementation and Evaluation

We have implemented a prototype DatalogMTL reasoner, called MeTeoR, which exploits our practical decision proce-

dures from Algorithm 2 to decide fact entailment. Our implementation uses existing libraries in the Python 3.8 eco-system without depending on other third-party libraries. To the best of our knowledge, no reasoning benchmark is currently available for DatalogMTL¹, so to evaluate the performance of MeTeoR we have designed two new benchmarks.

LUBM Benchmark. We obtain the first benchmark by extending the Lehigh University Benchmark (LUBM) (Guo, Pan, and Hefflin 2005) with temporal rules and data. To construct temporal datasets we modified LUBM’s data generator, which provides means for generating datasets of different sizes, to randomly assign an interval to each generated fact; the rational endpoints of each interval belong to a range, which is a customised parameter. We used the same approach to generate input query facts for entailment checks. In addition, we extended the 56 plain Datalog rules obtained from the OWL 2 RL fragment of LUBM with 29 temporal rules involving recursion and mentioning all metric operators available in DatalogMTL. We denote the resulting DatalogMTL program with 85 rules as Π_L .

Meteorological Benchmark. For this benchmark, we used a freely available dataset with meteorological observations (Maurer et al. 2002); in particular, we used a set \mathcal{D}_W of 397 millions of facts from the years 1949–2010. We then adapted the DatalogMTL program used by Brandt et al. (2018) to reason with DatalogMTL about weather, which resulted in a non-recursive program Π_W with 4 rules.

Machine Configuration. 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. Our first experiment tests the scalability of MeTeoR; the second experiment compares the performance of MeTeoR and the approach of Brandt et al. (2018) when reasoning over non-recursive programs. In both experiments, each test was conducted once.

Experiment 1: Scalability of MeTeoR

In Experiment 1a, to evaluate the scalability of MeTeoR, we used the recursive program Π_L from our LUBM benchmark and four datasets (obtained with LUBM’s data generator) \mathcal{D}_L^1 , \mathcal{D}_L^2 , \mathcal{D}_L^3 , and \mathcal{D}_L^4 of increasing size, which consist of 5, 10, 15, and 20 million facts, respectively. As already mentioned, input query facts are randomly generated. We hypothesise that performance critically depends on the type of these input query facts and, in particular, on which block within Algorithm 2 the answer is returned.

To check our hypothesis we have identified five disjoint types of input query facts for an input program Π and dataset \mathcal{D} , depending on where in the execution of Algorithm 2 an answer is returned. In particular, Algorithm 2 terminates in

- Line 1 for facts of type **T₁**, so these facts are already entailed by the input dataset \mathcal{D} ;
- Lines 6 or 7 for facts of type **T₂**, so these facts are not entailed by \mathcal{D} and mention a predicate whose relevant rules in Π are non-recursive;

¹The datasets used by Brandt et al. (2018) to test their reasoning approach for non-recursive programs are not publicly available.

| | T₁ | T₂ | T₃ | T₄ | T₅ |
|-------------------|---------------------------|--------------------------------|--------------------------------|---------------------------------|--------------------------|
| \mathcal{D}_L^1 | 0.6s($c = 0.6s, n = 0$) | 70.1s($c = 2.1s, n = 1.7$) | 66.9s($c = 4.1s, n = 1.9$) | 97.8s($c = 3.2s, n = 21.4$) | 7.6min($p = 2.7s$) |
| \mathcal{D}_L^2 | 0.8s($c = 0.8s, n = 0$) | 152.5s($c = 8.9s, n = 1.5$) | 167.9s($c = 11.2s, n = 3.1$) | 229.3s($c = 10.2s, n = 17.1$) | 50.2min($p = 10.5s$) |
| \mathcal{D}_L^3 | 1.9s($c = 1.9s, n = 0$) | 201.4s($c = 13.1s, n = 1.6$) | 269.2s($c = 18.3s, n = 2.9$) | 284.8s($c = 27.3s, n = 11.1$) | 92.1min ($p = 19.7s$) |
| \mathcal{D}_L^4 | 2.5s($c = 2.5s, n = 0$) | 301.2s($c = 21.3s, n = 1.8$) | 341.9s($c = 26.1s, n = 2.6$) | 312.5s($c = 41.3s, n = 7.8$) | 172.6min ($p = 28.6s$) |

Table 2: Results of Experiment 1a

- Line 13 or 22 for facts of type **T₃**, so materialisation of the relevant (recursive) subprogram reaches a fixpoint, and the fact is not entailed by the resulting materialisation;
- Line 12 or 21 for facts of type **T₄**, so these facts have a recursive relevant subprogram and are found to be entailed after finitely many rounds of rule application; and
- Line 26 for facts of type **T₅**, so the entailment of these facts is checked using automata.

The results of Experiment 1a are reported in Table 2, where we have generated 10 facts of each type and recorded the average runtime of MeTeoR for such sets of 10 facts; additionally, we have stated in brackets how much time was consumed for fact coalescing (c) and how many rounds of rule applications (n) were performed on average. In the case of type **T₅** we report, instead, the time consumed by the pre-materialisation step (p). The experiment shows that the performance of MeTeoR is dependent on the type of an input fact, and in particular, on whether the system can verify it using materialisation only (types **T₁–T₄**) or automata are needed (type **T₅**). As expected, runtimes increase with the size of the dataset and the number of rounds of rule applications needed. Furthermore, although materialisation was more efficient, it is worth noting that the automata-based approach could also be successfully applied to such large-scale datasets. Finally, coalescing times were relatively small, and so were pre-materialisation times for facts of type **T₅**.

If the majority of input facts can be verified using materialisation only, then we can expect robust and scalable performance; we hypothesise that this is likely to be the case in many practical situations. To verify this hypothesis on our benchmark, we have generated 1,000 random facts for each of the datasets \mathcal{D}_L^1 – \mathcal{D}_L^4 and calculated the percentage of facts that belong to each of the types **T₁–T₅** (with respect to the considered dataset and the program Π_L). We found that in 46.0% cases the facts were of type **T₁**, 28.3% of type **T₂**, 16.6% of type **T₃**, 8.3% of type **T₄**, and only 0.8% of type **T₅**. This supports our hypothesis that input facts requiring the use of automata may rarely occur in practice.

Finally, in Experiment 1b, we have stress tested MeTeoR. To this end, we proceeded as in Experiment 1a, but we keep constructing bigger datasets, until the average run times exceed 40 minutes. Note that this threshold is already exceeded by facts of type **T₅** in Experiment 1a, whereas facts of type **T₁** do not require rule applications. Therefore, in Experiment 1b we focused on types **T₂**, **T₃**, and **T₄**. Our results, which are shown in Figure 1, suggest that MeTeoR scales very well and it is capable of handling datasets containing up to 150 million temporal facts using materialisation.

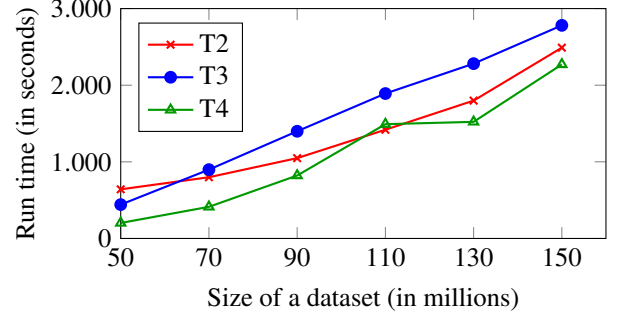


Figure 1: Results of Experiment 1b

Experiment 2: Comparison with Baseline

We performed two experiments to compare the performance of MeTeoR and the baseline approach of Brandt et al. (2018) for reasoning with non-recursive programs. The approach by Brandt et al. (2018) is based on query rewriting—given a target predicate P and an input program Π the algorithm generates a SQL query that, when evaluated over the input dataset \mathcal{D} , provides the set of all facts with maximal intervals over P entailed by Π and \mathcal{D} . To the best of our knowledge there is no publicly available implementation of this approach; thus, 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, Wang, and Zaniolo (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 used PostgreSQL 10.18 for all our baseline experiments. In each test, we could verify that the answers returned by MeTeoR and the baseline coincided.

In Experiment 2a, we compared the performance of our system with that of the baseline using three target predicates from our LUBM benchmark program Π_L , whose relevant sets of rules constitute non-recursive programs Π_L^1 , Π_L^2 , and Π_L^3 consisting of 5, 10, and 21 rules, respectively. For each program Π_L^1 – Π_L^3 and each dataset \mathcal{D}_L^1 – \mathcal{D}_L^4 from our benchmark, we used the baseline approach and MeTeoR to

| | Π_L^1 | | Π_L^2 | | Π_L^3 | |
|-------------------|-----------|--------|-----------|--------|-----------|--------|
| | baseline | MeTeoR | baseline | MeTeoR | baseline | MeTeoR |
| \mathcal{D}_L^1 | 160.3s | 51.3s | 345.9s | 146.8s | 523.9s | 213.4s |
| \mathcal{D}_L^2 | 331.8s | 90.5s | 747.8s | 321.9s | 1118.5s | 455.2s |
| \mathcal{D}_L^3 | 503.2s | 114.2s | 1154.0s | 425.8s | 1688.5s | 760.1s |
| \mathcal{D}_L^4 | 683.0s | 170.3 | 1575.6s | 582.7s | 2254.6s | 953.4s |

Table 3: Results of Experiment 2a

compute all facts over the target predicates entailed by the corresponding program and dataset—this was achieved by MeTeoR using materialisation only since these programs are non-recursive. As a result, MeTeoR computed a representation of the entire canonical model, and not just the extension of the target predicates. As shown in Table 3, both approaches exhibit good performance and scalable behaviour as the size of the data increases, with our system consistently outperforming the baseline.

In Experiment 2b, we performed similar tests as in Experiment 2a, but using the meteorological benchmark. We chose two target predicates from the program Π_W which correspond to non-recursive programs Π_W^1 and Π_W^2 , each with two rules. In addition, we constructed subsets \mathcal{D}_W^i of the entire meteorological dataset \mathcal{D}_W , where the superscript i indicates the number of years covered by the dataset; in particular \mathcal{D}_W^{62} is the full dataset covering all 62 years, so $\mathcal{D}_W^{62} = \mathcal{D}_W$. The results, as presented in Table 4, show that MeTeoR consistently outperforms the baseline approach.

| | Π_W^1 | | Π_W^2 | |
|----------------------|-----------|---------|-----------|--------|
| | baseline | MeTeoR | baseline | MeTeoR |
| \mathcal{D}_W^{10} | 376.9s | 141.8s | 38.8s | 34.8s |
| \mathcal{D}_W^{20} | 761.3s | 324.0s | 78.5s | 77.9s |
| \mathcal{D}_W^{30} | 1059.5s | 430.8s | 119.4s | 101.8s |
| \mathcal{D}_W^{40} | 1406.5s | 782.8s | 163.23s | 140.6s |
| \mathcal{D}_W^{50} | 1765.1s | 929.2s | 203.7s | 171.8s |
| \mathcal{D}_W^{62} | 2234.0s | 1050.8s | 258.5s | 240.4s |

Table 4: Results of Experiment 2b

Although in Experiments 2a and 2b the performance of the baseline was comparable to that of MeTeoR, we hypothesised that the performance gap would significantly increase (in favour of MeTeoR) as the number of temporal joins involved during reasoning increases. This is because computing temporal joins requires intersecting intervals—something for which databases do not seem to be optimised. In contrast, we anticipate that the performance of MeTeoR is more robust due to the way we implement joins (by first sorting the sets of intervals involved in a join and then linearly scanning these sets to compute the relevant intersections).

To verify this hypothesis we performed Experiment 2c which is similar to Experiments 2a and 2b, but we considered only one target predicate from the LUBM benchmark with the corresponding (non-recursive) program Π_L^1 , and

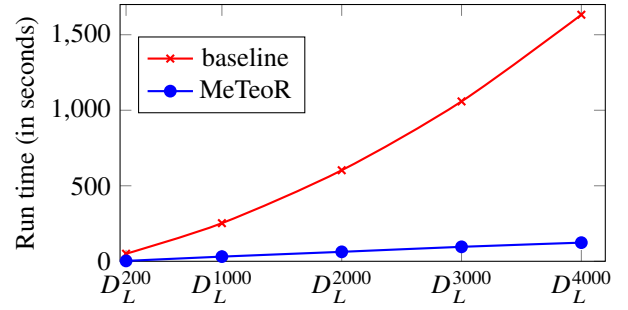


Figure 2: Results of Experiment 2c

datasets \mathcal{D}_L^{200} , \mathcal{D}_L^{1000} , \mathcal{D}_L^{2000} , \mathcal{D}_L^{3000} , and \mathcal{D}_L^{4000} , which, for each relational atom, contain increasing number of facts. In particular, these datasets contain facts mentioning 7,082 relational atoms— \mathcal{D}_L^{200} has 200 facts for each of these atoms (so in total 1,416,400 facts); whereas \mathcal{D}_L^{1000} , \mathcal{D}_L^{2000} , \mathcal{D}_L^{3000} , and \mathcal{D}_L^{4000} have 1000, 2000, 3000, and 4000 facts, respectively. In general, the larger is the number of facts mentioning the same relational atom, the larger the number of temporal joins involved in reasoning. Thus, our hypothesis is that, as we progress from \mathcal{D}_L^{200} towards \mathcal{D}_L^{4000} , we will see a larger performance gap between baseline and MeTeoR.

The results of Experiment 2c are shown in Figure 2. As anticipated, the performance of the baseline quickly degrades as the number of temporal joins needed for SQL query evaluation increases, whereas the performance of MeTeoR remains stable. We see these results as empirical validation for the need of specialised temporal join algorithms, which are not provided off-the-shelf by SQL engines.

Conclusion and Future Work

We have presented the first practical algorithm and scalable implementation for the full DatalogMTL language. Our system MeTeoR effectively combines optimised implementations of materialisation and automata-based reasoning, and was able to successfully decide fact entailment over complex recursive programs and large-scale datasets containing tens of millions of temporal facts.

We see many exciting avenues for future research. First, Cucala et al. (2021) have recently extended DatalogMTL with stratified negation as failure—a very useful feature for applications—and provided an automata-based decision procedure. It would be interesting to extend MeTeoR to support stratified negation, which will require a non-trivial revision of our algorithm since materialisation within each separate stratum may be non-terminating. Second, it would be interesting to explore incremental reasoning techniques (Wałęga, Kaminski, and Cuenca Grau 2019), which are especially relevant to stream reasoning applications. Third, we aim to explore additional optimisations to further improve scalability and also to apply MeTeoR in real-world applications in collaboration with our industrial partners.

Acknowledgments

This work was 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.

References

- Bellomarini, L.; Sallinger, E.; and Gottlob, G. 2018. The Vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. of VLDB Endow.*, 11(9): 975–987.
- Brandt, S.; Kalaycı, E. G.; Ryzhikov, V.; Xiao, G.; and Zakharyashev, M. 2018. Querying log data with metric temporal logic. *J. Artif. Intell. Res.*, 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.
- Carral, D.; Dragoste, I.; González, L.; Jacobs, C. J. H.; Krötzsch, M.; and Urbani, J. 2019. VLog: a rule engine for knowledge graphs. In *Proc. of ISWC*, 19–35.
- Ceri, S.; Gottlob, G.; and Tanca, L. 1989. What you always wanted to know about Datalog (and never dared to ask). *IEEE TKDE*, 1(1): 146–166.
- 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 AAAI*, 6488–6495.
- Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Proc. of AAMAS*, 19(3): 332–377.
- Garcia-Molina, H.; Ullman, J. D.; and Widom, J. 2009. *Database Systems - The Complete Book (2. ed.)*.
- Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: a benchmark for OWL knowledge base systems. *J. Web Semant.*, 3(2-3): 158–182.
- Kalaycı, E. G.; Xiao, G.; Ryzhikov, V.; Kalaycı, T. E.; and Calvanese, D. 2018. Ontop-temporal: a tool for ontology-based query answering over temporal data. In *Proc. of CIKM*, 1927–1930.
- Kikot, S.; Ryzhikov, V.; Wałęga, P. A.; and Zakharyashev, M. 2018. On the data complexity of ontology-mediated queries with MTL operators over timed words. In *Proc. of DL*.
- Koopmann, P. 2019. Ontology-based query answering for probabilistic temporal data. In *Proc. of AAAI*, 2903–2910.
- Koymans, R. 1990. Specifying real-time properties with metric temporal logic. *J. R Time Syst*, 2(4): 255–299.
- Maurer, E. P.; Wood, A. W.; Adam, J. C.; Lettenmaier, D. P.; and Nijssen, B. 2002. A long-term hydrologically based dataset of land surface fluxes and states for the conterminous United States. *JCLI*, 15(22): 3237–3251.
- Motik, B.; Nenov, Y.; Piro, R.; and Horrocks, I. 2019. Maintenance of Datalog materialisations revisited. *Artif. Intell.*, 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 AAAI*, 129–137.
- Munz, G.; and Carle, G. 2007. Real-time analysis of flow data for network attack detection. In *Proc. of FM*, 100–108.
- Nuti, G.; Mirghaemi, M.; Treleaven, P.; and Yingsaeree, C. 2011. Algorithmic trading. *Computer*, 44(11): 61–69.
- Ryzhikov, V.; Wałęga, P. A.; and Zakharyashev, M. 2019. Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics. In *Proc. of IJCAI*, 1851–1857.
- Wałęga, P. A.; Cuenca Grau, B.; Kaminski, M.; and Kostylev, E. V. 2019. DatalogMTL: computational complexity and expressive power. In *Proc. of IJCAI*, 1886–1892.
- Wałęga, P. A.; Cuenca Grau, B.; Kaminski, M.; and Kostylev, E. V. 2020a. DatalogMTL over the integer timeline. In *Proc. of KR*, 768–777.
- Wałęga, P. A.; Cuenca Grau, B.; Kaminski, M.; and Kostylev, E. V. 2020b. Tractable fragments of Datalog with metric temporal operators. In *Proc. of AAAI*, 1919–1925.
- Wałęga, P. A.; Kaminski, M.; and Cuenca Grau, B. 2019. Reasoning over streaming data in metric temporal Datalog. In *Proc. of AAAI*, 3092–3099.
- Wałęga, P. A.; Zawidzki, M.; and Cuenca Grau, B. 2021. Finitely materialisable Datalog programs with metric temporal operators. In *Proc. of KR*.
- Zhou, X.; Wang, F.; and Zaniolo, C. 2006. Efficient temporal coalescing query support in relational database systems. In *Proc. of DEXA*, 676–686.