Goal-Driven Reasoning in DatalogMTL with Magic Sets

Shaoyu Wang[1*], Kaiyue Zhao[1*], Dongliang Wei[1*], Przemysław Andrzej Wałęga[2,3], Dingmin Wang[2], Hongming Cai[1], Pan Hu[1†]

[1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, China
[2]Department of Computer Science, University of Oxford, UK
[3]School of Electronic Engineering and Computer Science, Queen Mary University of London, UK
{royalrasins, Horizon52183, weidongliang9, hmcai, pan.hu}@sjtu.edu.cn, {przemyslaw.walega, dingmin.wang}@cs.ox.ac.uk

Abstract

DatalogMTL is a powerful rule-based language for temporal reasoning. Due to its high expressive power and flexible modeling capabilities, it is suitable for a wide range of applications, including tasks from industrial and financial sectors. However, due its high computational complexity, practical reasoning in DatalogMTL is highly challenging. To address this difficulty, we introduce a new reasoning method for DatalogMTL which exploits the magic sets technique—a rewriting approach developed for (non-temporal) Datalog to simulate top-down evaluation with bottom-up reasoning. We have implemented this approach and evaluated it on publicly available benchmarks, showing that the proposed approach significantly and consistently outperformed state-of-the-art reasoning techniques.

Introduction

DatalogMTL (Brandt et al. 2018) extends the well-known declarative logic programming language Datalog (Ceri, Gottlob, and Tanca 1989) with operators from metric temporal logic (MTL) (Koymans 1990), allowing for complex temporal reasoning. It has a number of applications, including ontology-based query answering (Brandt et al. 2018; Güzel Kalayci et al. 2018), stream reasoning (Wałęga et al. 2023a; Wałęga, Kaminski, and Cuenca Grau 2019), and reasoning for the financial sector (Colombo et al. 2023; Nissl and Sallinger 2022; Mori et al. 2022), among others.

To illustrate capabilities of DatalogMTL, consider a scenario in which we want to reason about social media interactions. The following DatalogMTL rule describes participation of users in the circulation of a viral social media post:

$$\boxplus_{[0,2]} P(x) \leftarrow I(x,y) \wedge P(y), \tag{1}$$

namely, it states that if at a time point t a user x interacted with a user y over the post (expressed as $I(x,y)$), and y participated in the post's circulation (expressed as $P(y)$), then in the time interval $[t, t+2]$, user x will be continuously participating in the post's circulation ($\boxplus_{[0,2]} P(x)$).

One of the main reasoning tasks considered in DatalogMTL is fact entailment, which involves checking whether a program-dataset pair $(\Pi, \mathcal{D})$ logically entails a given temporal fact. This task was shown to be ExpSpace-complete in combined complexity (Brandt et al. 2018) and PSpace-complete in data complexity (Wałęga et al. 2019). To decrease computational complexity, various syntactical (Wałęga et al. 2020b) and semantic (Wałęga et al. 2020a) modifications of DatalogMTL have been introduced. DatalogMTL was also extended with stratified negation (Tena Cucala et al. 2021), non-stratified negation (Wałęga et al. 2021, 2024), and with temporal aggregation (Bellomarini, Nissl, and Sallinger 2021).

A sound and complete reasoning approach for DatalogMTL can be obtained using an automata-based construction (Wałęga et al. 2019). Another approach for reasoning is to apply materialisation, that is, successively apply to the initial dataset $\mathcal{D}$ rules in a given program Π, to derive all the facts entailed by a program-dataset pair $(\Pi, \mathcal{D})$. It is implemented, for example, within Vadalog reasoner (Bellomarini et al. 2022). Materialisation, however, does not guarantee termination since DatalogMTL programs can express (infinite) recursion over time. To marry completeness and efficiency of reasoning, a combination of automata- and materialisation-based approaches was introduced in the MeTeoR system (Wang et al. 2022). Materialisation-based approach typically outperforms the automata-based approach, so cases in which materialisation-based reasoning is sufficient were studied (Wałęga, Zawidzki, and Cuenca Grau 2021, 2023). More recently, a very efficient reasoning approach was introduced, which combines materialisation with detection of repeating periods (Wałęga et al. 2023b).

Despite recent advancement in the area, the currently available reasoning methods for DatalogMTL (Wang et al. 2022; Wałęga et al. 2023b; Bellomarini et al. 2022) can suffer from deriving huge amounts of temporal facts which are irrelevant for a particular query. This, in turn, can make the set of derived facts too large to store in memory or even in a secondary storage. For example, if the input dataset mentions millions of users, together with interactions between them and their participation in posts' circulation, application of Rule (1) may result in a huge set of derived facts. However, if we only want to check whether the user Arthur participated on post's circulation on September 10th, 2023, most of the derived facts are irrelevant. Note that determining which exactly facts are relevant is not easy, especially

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

when a DatalogMTL program contains a number of rules with complex interdependencies.

To address these challenges, we take inspiration from the *magic set* rewriting technique, which was developed for non-temporal Datalog (Abiteboul, Hull, and Vianu 1995; Ullman 1989b; Faber, Greco, and Leone 2007; Ullman 1989a). We extend it to the temporal setting of DatalogMTL, thereby proposing a new variant of magic set rewriting. For example, consider again Rule (1) and assume that we want to check if Arthur participated in a post's circulation on $10/9/2023$. Observe that if there is no chain of interactions between Arthur and another user Beatrice in the input dataset, then facts about Beatrice are irrelevant for our query, and so, we do not need to derive them. We will achieve it by rewriting the original program-dataset pair into a new pair (using additional magic predicates) such that the new pair provides the same answers to the query as the original pair, but allows us for a more efficient reasoning. As a result we obtain a goal-driven (i.e. query-driven) reasoning approach. To the best of our knowledge, our work is the first that allows to prune irrelevant derivations in DatalogMTL reasoning.

To implement the magic sets technique in the DatalogMTL setting, two main challenges need to be addressed. First, it is unclear how MTL operators affect sideways information passing in magic sets. We address this challenge by carefully examining each type of MTL operators and designing corresponding transformations. Second, for the transformed program-dataset pair established optimisations for reasoning in DatalogMTL may no longer apply or become less efficient. This is indeed the case, and so, we show how these optimisations can be adapted to our new setting. We implemented our approach and tested it using the state-of-the-art DatalogMTL solver MeTeoR. Experiments show that compared with existing methods, our approach achieves a significant performance improvement.

Preliminaries

We briefly summarise the core concepts of DatalogMTL and provide a recapitulation of magic set rewriting in the non-temporal setting.

Syntax of DatalogMTL. Throughout this paper, we assume that time is continuous, in particular, that the timeline is composed of rational numbers. A time interval is a set of continuous time points ϱ of the form $\langle t_1, t_2\rangle$, where $t_1, t_2 \in \mathbb{Q} \cup \{-\infty, \infty\}$, whereas $\langle$ is [or (and likewise $\rangle$ is] or). An interval is punctual if it is of the form $[t, t]$, for some $t \in \mathbb{Q}$; we will often represent it as t. An interval is bounded if both of its endpoints are rational numbers (i.e. neither ∞ or $-\infty$). Although time points are rational numbers, we will sometimes use dates instead. For example, $10/9/2023$ (formally it can be treated as the number of days that passed since $01/01/0000$). When it is clear from the context, we may abuse distinction between intervals (i.e. sets of time points) and their representation $\langle t_1, t_2\rangle$.

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

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

where $\top$ and $\bot$ are constants for, respectively, truth and falsehood, whereas ϱ are any intervals containing only non-negative rationals. A DatalogMTL rule, r, is of the form

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

where each M_i is a metric atom and M' is a metric atom not mentioning $\bot$, $\diamondplus$, $\diamondminus$, $\mathcal{U}$, and $\mathcal{S}$. Note that similarly as Wałęga et al. (2023b), we do not allow for $\bot$ in M' to ensures consistency and to focus on derivation of facts. We call M' the head of r and the set $\{M_i \mid i \in \{1, \ldots, n\}\}$ the body of r, and represent them as $head(r)$ and $body(r)$, respectively. For an example of a DatalogMTL rule see Rule (1).

We call a predicate intensional (or *idb*) in Π if it appears in the head of some rule in Π, and we call it extensional (or *edb*) in Π otherwise. Rule r is safe if each variable in r's head appears also in r's body. A DatalogMTL program, Π, is a finite set of safe rules. An expression (e.g. a relational atom, a rule, or a program) is ground if it does not mention any variables. A fact is of the form $R(\boldsymbol{t})@\varrho$, where $R(\boldsymbol{t})$ is a ground relational atom and ϱ is an interval. A query is of a similar form as a fact, but its relational atom $R(\boldsymbol{t})$ does not need to be ground. A dataset, $\mathcal{D}$, is a finite set of facts. A dataset or program is bounded, if all the intervals they mention are bounded.

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

Interpretation $\mathfrak{I}$ satisfies a fact $R(\boldsymbol{t})@\varrho$, if $\mathfrak{I}, t \vDash R(\boldsymbol{t})$ for each $t \in \varrho$, and $\mathfrak{I}$ satisfies a set B of ground metric atoms, in symbols $\mathfrak{I}, t \vDash B$, if $\mathfrak{I}, t \vDash M$ for each $M \in B$. Interpretation $\mathfrak{I}$ satisfies a ground rule r if $\mathfrak{I}, t \vDash body(r)$ implies $\mathfrak{I}, t \vDash head(r)$ for each $t \in \mathbb{Q}$. A ground instance of r is any ground rule r' such that there is a substitution σ mapping variables into constants so that $r' = \sigma(r)$ (for any expression e, we will use $\sigma(e)$ to represent the expression obtained by applying σ to all variables in e). Interpretation $\mathfrak{I}$ satisfies

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

Table 1: Semantics for ground metric atoms

a rule r, if it satisfies all ground instances of r, and it is a model of a program Π, if it satisfies all rules of Π. If $\mathfrak{I}$ satisfies every fact in a dataset $\mathcal{D}$, then $\mathfrak{I}$ is a model of $\mathcal{D}$. $\mathfrak{I}$ is a model of a pair $(\Pi, \mathcal{D})$ if $\mathfrak{I}$ is both a model of Π and a model of $\mathcal{D}$. A pair $(\Pi, \mathcal{D})$ entails a fact $R(\boldsymbol{t})@\varrho$, in symbols $(\Pi, \mathcal{D}) \models R(\boldsymbol{t})@\varrho$, if all models of $(\Pi, \mathcal{D})$ satisfy $R(\boldsymbol{t})@\varrho$. Given $(\Pi, \mathcal{D})$, an answer to a query q is any fact $R(\boldsymbol{t})@\varrho$ such that $(\Pi, \mathcal{D})$ entails $R(\boldsymbol{t})@\varrho$ and $R(\boldsymbol{t})@\varrho = \sigma(q)$, for some substitution σ. An interpretation $\mathfrak{I}$ contains interpretation $\mathfrak{I}'$ if $\mathfrak{I}$ satisfies every fact that $\mathfrak{I}'$ does and $\mathfrak{I} = \mathfrak{I}'$ if they contain each other. Each dataset $\mathcal{D}$ has a unique least interpretation $\mathfrak{I}_\mathcal{D}$, which is the minimal (with respect to containment) model of $\mathcal{D}$.

Given a program Π, we define the immediate consequence operator T_Π as a function mapping an interpretation $\mathfrak{I}$ into the least interpretation containing $\mathfrak{I}$ and satisfying for each time point t and each ground instance r (which does not mention $\bot$ in the head) of a rule in Π the following: $\mathfrak{I}, t \models \mathit{body}(r)$ implies $T_\Pi(\mathfrak{I}), t \models \mathit{head}(r)$. Now we are ready to define the canonical interpretation for a program-dataset pair $(\Pi, \mathcal{D})$. Repeated application of T_Π to $\mathfrak{I}_\mathcal{D}$ yields a transfinite sequence of interpretations $\mathfrak{I}_0, \mathfrak{I}_1, \dots$ such that (i) $\mathfrak{I}_0 = \mathfrak{I}_\mathcal{D}$, (ii) $\mathfrak{I}_{\alpha+1} = T_\Pi(\mathfrak{I}_\alpha)$ for α a successor ordinal, and (iii) $\mathfrak{I}_\beta = \bigcup_{\alpha<\beta} \mathfrak{I}_\alpha$ for β a limit ordinal and successor ordinals α (here union of two interpretations is the least interpretations satisfying all facts satisfied in these interpretations); then, $\mathfrak{I}_{\omega_1}$, where ω_1 is the first uncountable ordinal, is the canonical interpretation of $(\Pi, \mathcal{D})$, denoted as $\mathfrak{C}_{\Pi,\mathcal{D}}$ (Brandt et al. 2018).

Reasoning Task. We focus on fact entailment, the main reasoning task in DatalogMTL, which involves checking whether a given pair of DatalogMTL program and dataset $(\Pi, \mathcal{D})$ entails a fact $R(\boldsymbol{t})@\varrho$. This task is ExpSpace-complete (Brandt et al. 2018) in combined complexity and PSpace-complete in data complexity (Wałęga et al. 2019).

Magic Set Rewriting. We now recapitulate magic set rewriting for Datalog. For a detailed description of magic sets we refer ar reader to the work of Abiteboul, Hull, and Vianu (1995); Ullman (1989b); Faber, Greco, and Leone (2007); Ullman (1989a). Datalog can be seen as a fragment of DatalogMTL that disallows metric temporal operators and discards the notion of time. In the context of Datalog, given a query (which is now a relational atom) q and a program-dataset pair $(\Pi, \mathcal{D})$, the magic set approach constructs a pair $(\Pi', \mathcal{D}')$ such that $(\Pi', \mathcal{D}')$ and $(\Pi, \mathcal{D})$ provide the same answers to q.

The first step of magic set rewriting involves program adornment (adding superscripts to predicates). Adornments will guide construction of Π'. The adorned program Π_a is constructed as follows:

- Step 1: assume that the query is of the form $q = Q(\boldsymbol{t})$. We adorn Q with a string γ of letters *b* and *f*, which stand for 'bound' and 'free' terms. The length of γ is the arity of Q; the ith element of γ is *b* if the ith term in $\boldsymbol{t}$ is a constant, and *f* otherwise. For example $Q(x, y, Arthur)$ yields Q^{ffb}. We set $A = \{Q^\gamma\}$ and $\Pi_a = \varnothing$.
- Step 2: we remove one adorned predicate R^γ from A (in the first application of Step 2, set A contains only one element, but on later stages A will be modified). For each rule r whose head predicate is R, we generate an adorned rule r_a and add it to Π_a. The rule r_a is constructed from r as follows. We start by replacing the head predicate R in r with R^γ. Next we label all occurrences of terms in r as bound or free as follows:
  - (i) terms in $\mathit{head}(r)$ are labelled according to γ (i.e. the ith term is bound iff the ith element of γ is *b*),
  - (ii) each constant in $\mathit{body}(r)$ is labelled as bound,
  - (iii) each variable that is labelled as bound in $\mathit{head}(r)$ is labelled as bound in $\mathit{body}(r)$,
  - (iv) if a variable occurs in a body atom, all its occurrences in the subsequent atoms (i.e. located to the right of this body atom) are labelled as bound,
  - (v) all remaining variables in $\mathit{body}(r)$ are labelled as free.

  Then each body atom $P(\boldsymbol{t}')$ in r is replaced with $P^{\gamma'}(\boldsymbol{t}')$ such that the ith letter of γ' is b if the ith term of $\boldsymbol{t}'$ is labelled as bound, and it is f otherwise. Moreover, if P is *idb* in Π and $P^{\gamma'}$ was never constructed so far, we add $P^{\gamma'}$ to A. For example if R^γ is Q^{ffb} and r is $Q(x, Arthur, y) \leftarrow P(x, Beatrice) \wedge R(x, y)$, then r_a is $Q^{\mathit{ffb}}(x, Arthur, y) \leftarrow P^{\mathit{fb}}(x, Beatrice) \wedge R^{\mathit{bb}}(x, y)$. Then, we keep repeating Step 2 until A becomes empty.

Next, we generate the final Π' from the above constructed Π_a. For a tuple of terms $\boldsymbol{t}$ and an adornment γ of the same length, we let $\mathit{bt}(\boldsymbol{t}, \gamma)$ and $\mathit{bv}(\boldsymbol{t}, \gamma)$ be, respectively, the sequences of terms and variables in $\boldsymbol{t}$, which are bound according to γ (so $\mathit{bv}(\boldsymbol{t}, \gamma)$ is always a subsequence of $\mathit{bt}(\boldsymbol{t}, \gamma)$). For example, $\mathit{bt}((x, y, Arthur), \mathit{bfb}) = (x, Arthur)$ and $\mathit{bv}((x, y, Arthur), \mathit{bfb}) = (x)$. Then, for each rule $r_a \in \Pi_a$, represented as

$$R^\gamma(\boldsymbol{t}) \leftarrow R_1^{\gamma_1}(\boldsymbol{t}_1) \wedge \dots \wedge R_n^{\gamma_n}(\boldsymbol{t}_n),$$

we add to Π' the rule

$$R(\boldsymbol{t}) \leftarrow m\_R^\gamma(\boldsymbol{t}') \wedge R_1(\boldsymbol{t}_1) \wedge \dots \wedge R_n(\boldsymbol{t}_n) \tag{2}$$

and the following rules, for all $i \in \{1, \dots, n\}$ such that R_i is an *idb* predicate in Π:

$$m\_R_i^{\gamma_i}(\boldsymbol{t}_i') \leftarrow m\_R^\gamma(\boldsymbol{t}') \wedge R_1(\boldsymbol{t}_1) \wedge \dots \wedge R_{i-1}(\boldsymbol{t}_{i-1}), \tag{3}$$

where $\boldsymbol{t}' = \mathit{bv}(\boldsymbol{t}, \gamma)$, $\boldsymbol{t}_i' = \mathit{bv}(\boldsymbol{t}_i, \gamma_i)$, and m_R^γ and $m\_R_i^{\gamma_i}$ are newly introduced 'magic predicates' with arities $|\boldsymbol{t}'|$ and $|\boldsymbol{t}_i'|$, respectively. Intuitively, magic predicates are responsible for storing tuples relevant for the derivations of query answers. In particular, Rule (2) considers only derivations that are (i) considered by the original rule r and (ii) allowed by the magic predicate m_R^γ. Rule (3) is responsible for deriving facts about magic predicates.

Finally, assuming that the input query is of the form $q = Q(\boldsymbol{t})$, we construct $\mathcal{D}'$ by adding $m\_Q^\gamma(\mathit{bt}(\boldsymbol{t}, \gamma))$ to $\mathcal{D}$, where γ is the adornment from Step 1. The new pair $(\Pi', \mathcal{D}')$ provides the same answers to query q as $(\Pi, \mathcal{D})$, but allows us to provide answers to q faster than $(\Pi, \mathcal{D})$.

As an example of application of magic set rewriting, consider a Datalog dataset $\mathcal{D}$, a Datalog version of Rule (1):

$$P(x) \leftarrow I(x, y) \wedge P(y), \tag{4}$$

and a query $P(Arthur)$. After applying Step 1 we get $A = \{P^b\}$ and $\Pi_a = \varnothing$. Then, in Step 2, we remove P^b from A, and adorn Rule (4) according to conditions (i)–(v). As a result, we obtain the following rule:

$$P^b(x) \leftarrow I^{bf}(x, y) \wedge P^b(y)$$

as a single rule in Π_a. No predicate is added to A, so $A = \varnothing$, and so we do not apply Step 2 any more. Next, we generate the following program Π' from Π_a:

$$P(x) \leftarrow m\_P^b(x) \wedge I(x, y) \wedge P(y),$$
$$m\_P^b(y) \leftarrow m\_P^b(x) \wedge I(x, y).$$

Finally, we set $\mathcal{D}' = \mathcal{D} \cup \{m\_P^b(Arthur)\}$, which concludes the construction of $(\Pi', \mathcal{D}')$. The newly constructed $(\Pi', \mathcal{D}')$ and the initial $(\Pi, \mathcal{D})$ have the same answers to the query $P(Arthur)$. However, $(\Pi', \mathcal{D}')$ entails a smaller amount of facts about P. Indeed, if in the dataset $\mathcal{D}$ there is no chain of interactions I connecting a constant $Beatrice$ with $Arthur$, then neither $P(Beatrice)$ nor $m\_P^b(Beatrice)$ will be entailed by $(\Pi', \mathcal{D}')$.

Magic Sets for DatalogMTL

In this section we explain how do we extend magic set rewriting to the DatalogMTL setting. Since the approach is relatively complex, we start by providing an example how, given a particular query q and a DatalogMTL program-dataset pair $(\Pi, \mathcal{D})$, we construct $(\Pi', \mathcal{D}')$. For simplicity, in the example we will use a ground query q (i.e. a fact), but it is worth emphasising that our approach supports also non-ground queries.

Example. Consider a query $q = P(Arthur)@10/9$, a DatalogMTL dataset $\mathcal{D}$, and a program consisting of the rules

$$\boxplus_{[0,2]} P(x) \leftarrow I(x, y) \wedge P(y), \qquad (r_1)$$
$$\boxplus_{[0,1]} P(x) \leftarrow I(x, y) \wedge \diamondminus_{[0,1]} P(y). \qquad (r_2)$$

We construct $\mathcal{D}'$ by adding $m\_P^b(Arthur)@10/9$ to $\mathcal{D}$. Hence, the construction of $\mathcal{D}'$ is similar as in the case of (non-temporal) Datalog, but now the fact with magic predicate mentions also the time point from the input query. This will allow us not only to determine which atoms, but also which time points are relevant for answering the query.

Next, we construct an adorned program Π_a as below:

$$\boxplus_{[0,2]} P^b(x) \leftarrow I^{bf}(x, y) \wedge P^b(y), \qquad (r_{a_1})$$
$$\boxplus_{[0,1]} P^b(x) \leftarrow I^{bf}(x, y) \wedge \diamondminus_{[0,1]} P^b(y). \qquad (r_{a_2})$$

The construction of Π_a is analogous to that in Datalog (see the previous section), but Π_a now contains temporal operators. As in the case of Datalog, we will obtain Π' by constructing two types of rules, per a rule in Π_a. However, the construction of these rules will require further modifications.

Rules of the first type are extensions of rules in Π_a obtained by adding atoms with magic predicates to rule bodies. We can observe, however, that the straightforward adaptation of Rule (2) used in Datalog is not appropriate in the temporal setting. To observe an issue, let us consider Rule (r_{a_1}). The naïve adaptation of Rule (2) would construct from Rule (r_{a_1}) the following rule:

$$\boxplus_{[0,2]} P(x) \leftarrow m\_P^b(x) \wedge I(x, y) \wedge P(y).$$

Recall that our goal is to construct $(\Pi', \mathcal{D}')$ which will have exactly the same answers to the input query as the original $(\Pi, \mathcal{D})$. The rule above, however, would not allow us to achieve it. Indeed, if $\mathcal{D}$ consists of facts $I(Arthur, Beatrice)@8/9$ and $P(Beatrice)@8/9$, then the original Rule (r_1) allows us to derive $P(Arthur)@[8/9, 10/9]$, and so, the query fact $P(Arthur)@10/9$ is derived. However, the rule constructed above does not allow us to derive any new fact from $\mathcal{D}'$, because we do not have $m\_P^b(Arthur)@8/9$ in $\mathcal{D}'$, which is required to satisfy the rule body. Our solution is to use $\oplus_{[0,2]} m\_P^b(x)$ instead of $m\_P^b(x)$ in the rule body, namely we construct the rule

$$\boxplus_{[0,2]} P(x) \leftarrow \oplus_{[0,2]} m\_P^b(x) \wedge I(x, y) \wedge P(y).$$

This allows us to obtain missing derivations; in particular, in our example above, this new rule allows us to derive $P(Arthur)@[8/9, 10/9]$, and so, $P(Arthur)@10/9$, as required. Similarly, for Rule (r_{a_2}) we construct

$$\boxplus_{[0,1]} P(x) \leftarrow \oplus_{[0,1]} m\_P^b(x) \wedge I(x, y) \wedge \diamondminus_{[0,1]} P(y).$$

Next we construct rules of the second type, by adapting the idea from Rule (3). For Rule (r_{a_1}) we use the form of Rule (3), but we additionally precede the magic predicate in the body with the diamond operator, to prevent the observed issue with missing derivations. Hence, we obtain

$$m\_P^b(y) \leftarrow \oplus_{[0,2]} m\_P^b(x) \wedge I(x, y).$$

The case of Rule (r_{a_2}), however, is more challenging as this rule mentions a temporal operator in the body. Because of that, we observe that the below rule would not be appropriate

$$m\_P^b(y) \leftarrow \oplus_{[0,1]} m\_P^b(x) \wedge I(x, y).$$

To observe the issue, let us consider a program Π, which instead of Rule (r_1) has Rule (r_3) which is of the form $P(x) \leftarrow S(x)$. Therefore, Rule (r_{a_3}) is $P^b(x) \leftarrow S^b(x)$ and its first-type-rule is $P(x) \leftarrow m\_P^b(x) \wedge S(x)$. Assume moreover, that $\mathcal{D}$ consists of $S(Beatrice)@8/9$ and $I(Arthur, Beatrice)@9/9$. Hence, Rule (r_3) allows us to derive $P(Beatrice)@8/9$, and then Rule (r_2) derives $P(Arthur)@[9/9, 10/9]$. Although with the rule generated above from Rule (r_{a_2}), we can derive from $\mathcal{D}'$ the fact $m\_P^b(Beatrice)@9/9$, we cannot derive $m\_P^b(Beatrice)@8/9$. This, in turn, disallows us to derive $P(Arthur)@10/9$. In order to overcome this problem, it turns out that it suffices to add a temporal operator in the head of the rule. In our case, we construct the following rule:

$$\boxminus_{[0,1]} m\_P^b(y) \leftarrow \oplus_{[0,1]} m\_P^b(x) \wedge I(x, y).$$

Hence, the finally constructed Π' consists of the rules

$$\boxplus_{[0,2]} P(x) \leftarrow \oplus_{[0,2]} m\_P^b(x) \wedge I(x, y) \wedge P(y),$$
$$\boxplus_{[0,1]} P(x) \leftarrow \oplus_{[0,1]} m\_P^b(x) \wedge I(x, y) \wedge \diamondminus_{[0,1]} P(y),$$
$$m\_P^b(y) \leftarrow \oplus_{[0,2]} m\_P^b(x) \wedge I(x, y),$$
$$\boxminus_{[0,1]} m\_P^b(y) \leftarrow \oplus_{[0,1]} m\_P^b(x) \wedge I(x, y).$$

In what follows we will provide details of this construction and show that $(\Pi', \mathcal{D}')$ and $(\Pi, \mathcal{D})$ have the same answers to the input query.

Algorithm Overview. Our approach to constructing $(\Pi', \mathcal{D}')$ is provided in Algorithm 1. The algorithm takes as input a DatalogMTL program Π, dataset $\mathcal{D}$, and query $q = Q(\boldsymbol{t})@_\varrho$, and it returns the rewritten pair $(\Pi', \mathcal{D}')$. For simplicity of presentation we will assume that rules in Π do not have nested temporal operators and that their heads always mention either $\boxplus$ or $\boxminus$. Note that these assumptions are without loss of generality, as we can always flatten nested operators by introducing additional rules, whereas an atom M with no temporal operators can be written as $\boxplus_0 M$ (or equivalently as $\boxminus_0 M$) (Brandt et al. 2018). Note, moreover, that our rewriting technique does not require the input query to be a fact, namely the sequence $\boldsymbol{t}$ in the query $Q(\boldsymbol{t})@_\varrho$ can contain variables.

Line 1 of Algorithm 1 computes $\mathcal{D}'$ as an extension of $\mathcal{D}$ with the fact $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@_\varrho$. It is obtained from the query $Q(\boldsymbol{t})@_\varrho$ using a sequence γ_0 of b and f such that the ith element is b if and only if the ith element of $\boldsymbol{t}$ is a constant.

Lines 2–3 compute the adorned program Π_a. In particular, Line 2 initialises the set A of adorned *idb* predicates that are still to be processed and the set Π_a of adorned rules. Then, Line 3 constructs Π_a in the same way as in the case of Datalog (see Preliminaries).

Line 4 initialises, as an empty set, Π' and two auxiliary sets of rules Π'_1 and Π'_2. Then, Lines 5–9 compute a set Π'_1 of rules of the first type and Lines 10–15 compute a set Π'_2 of rules of the second type (to be precise, Π'_1 and Π'_2 become rules of the first and the second type after removing adornments in Line 17).

In particular, the loop from Lines 5–9 constructs from each $r \in \Pi_a$ a rule r' and adds it to Π'_1. To construct r', Lines 7–8 check if the head of r mentions $\boxplus$ or $\boxminus$. In the first case, Line 7 adds an atom with $\oplus$ and a magic predicate to the rule body. In the second case, Line 8 adds an atom with $\ominus$ and a magic predicate to the rule body. Then, Line 9 adds r' to Π'_1.

Lines 10–15 construct rules Π'_2 of the second type from rules in Π'_1. To this end, for each rule r in Π'_1 (Line 10) and each body atom M in r which mentions an *idb* predicate (Line 11) the algorithm performs the following computations. In Line 12 it computes a set H of atoms with magic predicates which will be used to construct rule heads. This is performed with a function **MagicHeadAtoms** implemented with Algorithm 2, which we will describe later. Then, the loop in Lines 13–15 computes a rule r' for each atom M' in H and adds this r' to Π'_2. Rule r' is constructed from r in Line 14, by replacing the head with M' and deleting from the rule body M as well as all the other body atoms located to the right of M.

To construct the final program Π' from Π'_1 and Π'_2, Line 16 constructs the union of these two sets and Line 17 deletes from these rules all the adornments of predicates which are non-magic, that is, not preceded by $m\_$. Line 18 returns the pair consisting of Π' and $\mathcal{D}'$.

Algorithm 1: MagicRewriting

Input : A DatalogMTL program Π, dataset $\mathcal{D}$, and query $q = Q(\boldsymbol{t})@_\varrho$
Output: A DatalogMTL program Π' and dataset $\mathcal{D}'$

1 $\mathcal{D}' := \mathcal{D} \cup \{m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@_\varrho\}$;
2 $A := \{Q^{\gamma_0}\}$; $\Pi_a := \varnothing$;
3 $\Pi_a :=$ a program obtained from Π by applying Step 2 from Section "Magic Set Rewriting" until $A = \varnothing$;
4 $\Pi' := \varnothing$; $\Pi'_1 := \varnothing$; $\Pi'_2 := \varnothing$;
5 **foreach** *rule* $r \in \Pi_a$ **do**
6 Let $\Box_{\varrho'} R^\gamma(\boldsymbol{t})$, with $\Box \in \{\boxplus, \boxminus\}$, be the head of r ;
7 **if** $\Box = \boxplus$ **then** $r' := r$ with $\oplus_{\varrho'} m\_R^\gamma(bv(\boldsymbol{t}, \gamma))$ added as the first body atom ;
8 **if** $\Box = \boxminus$ **then** $r' := r$ with $\ominus_{\varrho'} m\_R^\gamma(bv(\boldsymbol{t}, \gamma))$ added as the first body atom ;
9 $\Pi'_1 := \Pi'_1 \cup \{r'\}$;
10 **foreach** *rule* $r \in \Pi'_1$ **do**
11 **foreach** $M \in body(r)$ *with an idb predicate* **do**
12 $H :=$ **MagicHeadAtoms**(M) ;
13 **foreach** *atom* $M' \in H$ **do**
14 $r' := r$ with head replaced by M' and body modified by deleting M and all body atoms to the right of M ;
15 $\Pi'_2 := \Pi'_2 \cup \{r'\}$;
16 $\Pi' = \Pi'_1 \cup \Pi'_2$;
17 Delete adornments from non-magic predicates in Π' ;
18 **return** $(\Pi', \mathcal{D}')$;

MagicHeadAtoms. The function **MagicHeadAtoms** is used to construct rules of the second type (i.e. the set Π'_2), as presented in Algorithm 2. The algorithm takes as input a metric atom M and returns a set H of metric atoms, whose predicates are magic predicates. Line 1 initialises H as the empty set and introduces an auxiliary M' which is set to M. The loop in Lines 2–3 modifies M' by replacing *idb* predicates with magic predicates. Then, Lines 4-18 construct the set H depending on the form of M'. Line 4 consider the cases when M' mentions $\boxplus$ or $\boxminus$. Line 5 considers the case when M' mentions $\oplus$, and Line 6 when M' mentions $\ominus$. Lines 7-12 are for the case when M' mentions $\mathcal{S}$ and Lines 13-18 when M' mentions $\mathcal{U}$. The final set H is returned in Line 19.

Correctness of the Algorithm. Next we aim to show that, given a program Π, a dataset $\mathcal{D}$, and a query $q = Q(\boldsymbol{t})@_\varrho$, the pair $(\Pi', \mathcal{D}')$ returned by Algorithm 1 entails the same answers to q as $(\Pi, \mathcal{D})$. This result is provided by the following theorem.

Theorem 1. *Let Π be a DatalogMTL program, $\mathcal{D}$ a dataset, $Q(\boldsymbol{t})@_\varrho$ a query, and $(\Pi', \mathcal{D}')$ the output of Algorithm 1 when run on Π, $\mathcal{D}$, and $Q(\boldsymbol{t})@_\varrho$. For each time point $t \in \varrho$ and each substitution σ mapping variables in $\boldsymbol{t}$ to constants we have*

$$(\Pi, \mathcal{D}) \models Q(\sigma(\boldsymbol{t}))@t \quad \textit{iff} \quad (\Pi', \mathcal{D}') \models Q(\sigma(\boldsymbol{t}))@t.$$

Proof Sketch. For the if part, we leverage the fact that the first kind of rules in Π' (i.e. rules in Π'_1) are constructed by

Algorithm 2: MagicHeadAtoms

Input : A metric atom M
Output: A set H of metric atoms with magic predicates

$H := \varnothing$; $M' := M$;
foreach *adorned idb relational atom* $R^{\gamma}(\boldsymbol{t})$ *in* M **do**
  Replace $R^{\gamma}(\boldsymbol{t})$ in M' with $m\_R^{\gamma}(bv(\boldsymbol{t}, \gamma))$;
if $M' = \boxplus_{\varrho} M_1$ *or* $M' = \boxminus_{\varrho} M_1$ **then** $H := \{M'\}$;
if $M' = \diamondplus_{\varrho} M_1$ **then** $H := \{\boxplus_{\varrho} M_1\}$;
if $M' = \diamondminus_{\varrho} M_1$ **then** $H := \{\boxminus_{\varrho} M_1\}$;
if $M' = M_2 \, \mathcal{S}_{\varrho} M_1$ **then**
  t_{max} := the right endpoint of ϱ ;
  if M_2 *contains a magic predicate* **then**
    $H := H \cup \{\boxminus_{[0,t_{max}]} M_2\}$;
  if M_1 *contains a magic predicate* **then**
    $H := H \cup \{\boxminus_{\varrho} M_1\}$;
if $M' = M_2 \, \mathcal{U}_{\varrho} M_1$ **then**
  t_{max} := the right endpoint of ϱ ;
  if M_2 *contains a magic predicate* **then**
    $H := H \cup \{\boxplus_{[0,t_{max}]} M_2\}$;
  if M_1 *contains a magic predicate* **then**
    $H := H \cup \{\boxplus_{\varrho} M_1\}$;
return H ;

adding body atoms with magic predicates to the bodies of rules in Π. Hence, we can show that each fact entailed by $(\Pi', \mathcal{D}')$ is also entailed by $(\Pi, \mathcal{D})$.

The opposite direction is more challenging. The main part of the proof is to show that if $(\Pi, \mathcal{D}) \models R(\boldsymbol{t})@t$ and $(\Pi', \mathcal{D}') \models m\_R^{\gamma}(bt(\boldsymbol{t}, \gamma))@t$ for some $R(\boldsymbol{t})@t$ and γ, then $(\Pi', \mathcal{D}') \models R^{\gamma}(bt(\boldsymbol{t}, \gamma))@t$. This is sufficient to finish the proof. Indeed, note that by the construction of $\mathcal{D}'$, it contains $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@\varrho$. Therefore, by the implication above we have $(\Pi', \mathcal{D}') \models m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@t$, as required. To show the needed implication we conduct a proof by a transfinite induction on the number of T_{Π} applications to $\mathfrak{I}_{\mathcal{D}}$ and $T_{\Pi'}$ applications to $\mathfrak{I}_{\mathcal{D}'}$. □

Reasoning with the Algorithm. Theorem 1 shows us that to answer a query q with respect to Π and $\mathcal{D}$, we can instead answer q with respect to Π' and $\mathcal{D}'$ constructed by Algorithm 1. In what follows we will show how to efficiently answer q with respect to Π' and $\mathcal{D}'$. For this, we will use a recently introduced approach by Wałęga et al. (2023b), which constructs a finite representation of (usually infinite) canonical model of a program and a dataset. Importantly, this approach is guaranteed to terminate, which stands in contrast with a pure materialisation approach in DatalogMTL that often requires a transfinite number of rule applications.

Although the approach of Wałęga et al. (2023b), presented in their Algorithm 1, overperforms other reasoning techniques for DatalogMTL, it is worth observing that it is guaranteed to terminate only if the input program and dataset are both bounded (i.e. mention only bounded intervals). Moreover, the runtime of their algorithm heavily depends on the distance between the minimal and maximal time points in the input dataset. This makes applying their algorithm in our setting challenging, because time points of our $\mathcal{D}'$ depend on the input query. Indeed, $\mathcal{D}'$ constructed in Line 1 of our Algorithm 1 contains the fact $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@\varrho$, whose interval ϱ is copied from the input query $q = Q(\boldsymbol{t})@\varrho$. Hence, if ϱ is large, then applying the approach of Wałęga et al. (2023b) may be inefficient. Moreover, if ϱ is unbounded, then so is $\mathcal{D}'$, and we loose the guarantee of termination.

We will address these challenges below. First, we observe that given a program Π, dataset $\mathcal{D}$, and query $q = Q(\boldsymbol{t})@\varrho$, instead of computing **MagicRewriting**$(\Pi, \mathcal{D}, Q(\boldsymbol{t})@\varrho)$ we can compute **MagicRewriting**$(\Pi, \mathcal{D}, Q(\boldsymbol{t})@(-\infty, +\infty))$ and the new program-dataset pair will still entail the same answers to q as the the input Π and $\mathcal{D}$. This is a simple consequence of Theorem 1 as stated formally below.

Corollary 2. *Let* Π *be a DatalogMTL program,* $\mathcal{D}$ *a dataset,* $Q(\boldsymbol{t})@\varrho$ *a query, and* $(\Pi', \mathcal{D}')$ *the output of Algorithm 1 when run on* Π*,* $\mathcal{D}$*, and* $Q(\boldsymbol{t})@(-\infty, +\infty)$*. For each time point* $t \in \varrho$ *and each substitution* σ *mapping variables in* $\boldsymbol{t}$ *to constants we have*

$$(\Pi, \mathcal{D}) \models Q(\sigma(\boldsymbol{t}))@t \quad \textit{iff} \quad (\Pi', \mathcal{D}') \models Q(\sigma(\boldsymbol{t}))@t.$$

We note that **MagicRewriting**$(\Pi, \mathcal{D}, Q(\boldsymbol{t})@(-\infty, +\infty))$ constructs a pair $(\Pi', \mathcal{D}')$ such that $\mathcal{D}'$ contains $m\_R(\boldsymbol{t})@(-\infty, +\infty)$. Hence, $\mathcal{D}'$ is not bounded, and so, application of Algorithm 1 by Wałęga et al. (2023b) to Π', $\mathcal{D}'$, and q is not guaranteed to terminate. What we show next, however, is that unboundedness of $m\_R(\boldsymbol{t})@(-\infty, +\infty)$ does not lead to non-termination. Indeed, if Π and $\mathcal{D}$ are bounded, then the algorithm of Wałęga et al. (2023b) is guaranteed to terminate on Π', $\mathcal{D}'$, and q, as stated next.

Theorem 3. *Let* Π *be a bounded program,* $\mathcal{D}$ *a bounded dataset,* $q = Q(\boldsymbol{t})@\varrho$ *a fact, and* $(\Pi', \mathcal{D}')$ *the output of Algorithm 1 when run on* Π*,* $\mathcal{D}$*, and* $Q(\boldsymbol{t})@(-\infty, +\infty)$*. Application of Algorithm 1 by Wałęga et al. (2023b) is guaranteed to terminate on* Π'*,* $\mathcal{D}'$*, and* q*, even though* $\mathcal{D}'$ *is not bounded.*

Proof Sketch. Observe that the program Π' constructed by our Algorithm 1 is bounded and the only unbounded fact in $\mathcal{D}'$ is $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@(-\infty, +\infty)$. To show that application of Algorithm 1 by Wałęga et al. (2023b) terminates on Π', $\mathcal{D}'$, and q, we replace the unbounded fact in $\mathcal{D}'$ with an additional rule in Π', namely we construct Π'' by extending Π' with a single rule $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0)) \leftarrow \top$, which simulates $m\_Q^{\gamma_0}(bt(\boldsymbol{t}, \gamma_0))@(-\infty, +\infty)$. We can show that the number of iterations of their algorithm running on $(\Pi'', \mathcal{D}, q)$ is equal to or greater than the number of its iterations on $(\Pi', \mathcal{D}', q)$ by one. Since Π'' and $\mathcal{D}$ are bounded, both of the above mentioned numbers of iterations need to be finite. Hence, the algorithm is guaranteed to terminate on Π', $\mathcal{D}'$, and q, as stated in the theorem. □

Hence, we obtain a terminating approach for reasoning in DatalogMTL with magic set rewriting. To sum up, our approach to check entailment of a fact $q = Q(\boldsymbol{t})@\varrho$ with respect to a program Π and a dataset $\mathcal{D}$ consists of two steps. The first step is to construct Π' and $\mathcal{D}'$ by applying our Algorithm 1 to Π, $\mathcal{D}$, and $Q(\boldsymbol{t})@(-\infty, +\infty)$. The second step is to apply Algorithm 1 of Wałęga et al. (2023b) to Π', $\mathcal{D}'$, and q. This method is guaranteed to terminate and correctly

answer if Π and $\mathcal{D}$ entail q. In the next section we will study performance of this new approach.

Empirical Evaluation

We implemented our algorithm and evaluated it on LUBM_t (Wang et al. 2022)—a temporal version of LUBM (Guo, Pan, and Heflin 2005)—, on iTemporal (Bellomarini, Nissl, and Sallinger 2022), and on the meteorological (Maurer et al. 2002) benchmarks. The aim of our experiments is to check how the performance of reasoning with magic programs compare with reasoning using original programs. The three programs we used have 85, 11, and 4 rules, respectively. Moreover, all our test queries are facts (i.e. they do not have free variables) since current DatalogMTL system do not allow for answering queries with variables (they are designed to check entailment). Given a program Π and a dataset $\mathcal{D}$ we used MeTeoR system (Wałęga et al. 2023b) to perform reasoning. The engine first loads the pair and preprocesses the dataset before it starts reasoning. Then, for each given query fact, we record the wall-clock time that the baseline approach (Wałęga et al. 2023b) takes to answer the query. For our approach, we record the time taken for running Algorithm 1 to generate the transformed pair $(\Pi', \mathcal{D}')$ plus the time of answering the query via the materialisation of $(\Pi', \mathcal{D}')$. We ran the experiments on a server with 256 GB RAM, Intel Xeon Silver 4210R CPU @ 2.40GHz, Fedora Linux 40, kernel version 6.8.5-301.fc40.x86_64. In each test case, we verified that both our approach and the baseline approach produced the same answer to the same query. Moreover, we observed that the time taken by running Algorithm 1 was in all test cases less than 0.01 second, so we do not report these numbers separately. Note that our results are not directly comparable to those presented by Wałęga et al. (2023b) and Wang et al. (2022), since the datasets and queries are generated afresh, and different hardware is used. The datasets we used, the source code of our implementation, as well as an extended technical report, are available online.[1]

Comparison With Baseline. We compared the performance of our approach to that of the reasoning with original programs on datasets from LUBM_t with 10^6 facts, iTemporal with 10^5 facts, and ten year's worth of meteorological data. We ran 119, 35, and 28 queries on them respectively. These queries cover every *idb* predicate and are all facts. The results are summarised on the box plot in Figure 1. The red and orange boxes represent time statistics for magic programs and the original program, respectively, while the blue box represents the per-query time ratios. Since computing finite representations using the baseline programs requires too much time for LUBM_t and iTemporal (more than 100,000s and 50,000s) and memory and time consumption should theoretically stay the same across each computation, we used an average time for computing finite representations as the query time for each not entailed query instead of actually running them.

For the LUBM_t case, we achieved a performance boost on every query, the least enhanced (receiving the least speedup among other queries compared with the original program) of these received a 1.95 times speedup. For the entailed queries, half were accelerated by more than 3.46 times, 25% were answered more than 112 times faster. The most enhanced query were answered 2,110 times faster. For the not entailed queries, every query was accelerated by more than 12 times. Three quarters of the queries were answered more than 128 times faster using magic programs than the original program, 25% were answered more than 9,912 times faster. The most enhanced query were answered 111,157 times faster. It is worth noting that for the queries where a performance uplift are most needed, which is when the original program's finite representation needs to be computed, especially when the query fact is not entailed, our rewritten pairs have consistently achieved an acceleration of more than 12 times, making the longest running time of all tested queries 12 times shorter. This can mean that some program-dataset pairs previously considered beyond the capacity of the engine may now be queried with acceptable time costs.

For the meteorological dataset case, the results resemble those of LUBM_t, with the least accelerated query being answered 1.58 times faster. Half of the queries were answered more than 3.54 times. One particular query was answered more than 1 million times faster.

For the iTemporal case, we achieved a more than 69,000 times performance boost on every query, some over a million times. As can be observed from the chart on the right in Figure 1, the general performance boost is considerable and rather consistent across queries. The huge performance boost is because in this case, an *idb* fact has very few relevant facts and derivations of the vast majority of *idb* facts were avoided by magic programs.

Scalability Experiments. We conducted scalability experiments on both LUBM_t and iTemporal benchmarks as they are equipped with generators allowing for constructing datasets of varying sizes. For LUBM_t, we generated datasets of 5 different sizes, 10^2 – 10^6, and used the program provided along with the benchmark; for each dataset size, we selected 30 queries. For iTemporal, the datasets are of sizes 10 – 10^5 and the program was automatically generated; for each dataset size, we selected 270 queries. To rule out the impact of early termination and isolate the effect that magic sets rewriting has on the materialisation time, we only selected queries that cannot be entailed so that the finite representation is always fully computed. These queries were executed using both approaches, with the average execution time computed and displayed as the red (Π for original program) and blue (Π' for magic program) lines in Figure 2.

As the chart shows, our magic programs scale better for LUBM_t than the original program in terms of canonical model computation. For iTemporal, the first two datasets are too small that both approaches compute the results instantaneously. As the size of a dataset increases from 10^3 to 10^5, we observe a similar trend as LUBM_t.

[1] https://github.com/RoyalRaisins/Magic-Sets-for-DatalogMTL

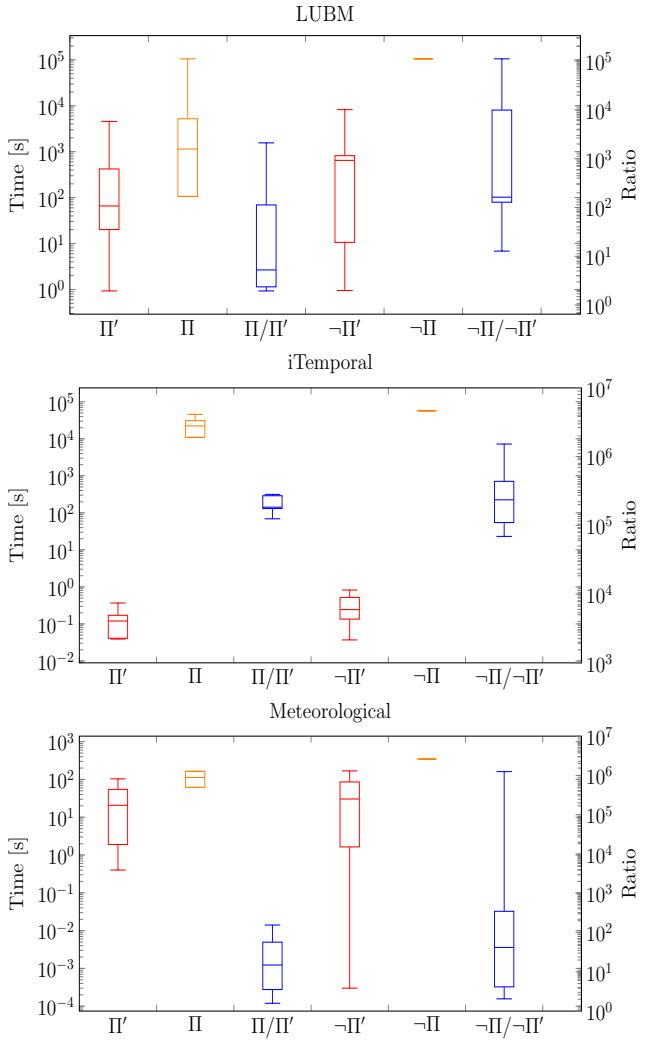

Figure 1: Comparison with baseline; (¬)Π′, (¬)Π, (¬)Π/(¬)Π′ refer to time consumed by (not) entailed queries on magic programs, the baseline program, and to per-query time ratios. Scales for ratios are on the right

Conclusion and Future Work

We have extended the magic set rewriting technique to the temporal setting, which allowed us to improve performance of query answering in DatalogMTL. We have obtained a goal-driven approach, providing an alternative to the state-of-the-art materialisation-based method in DatalogMTL. Our new approach can be particularly useful in applications where the input dataset changes frequently or the materialisation of the entire program and dataset is not feasible due to too high time and space consumption. In future, we plan to consider developing a hybrid approach that combines magic sets technique with materialisation. Another interesting avenue is to extend the magic sets approach to languages including such features such as negation and aggregation.

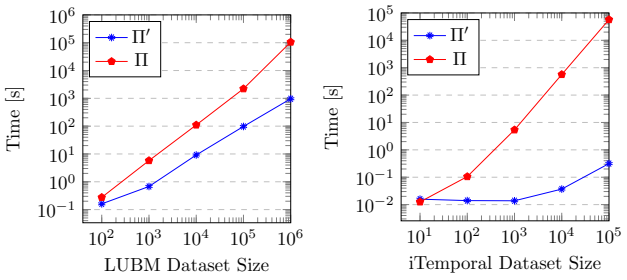

Figure 2: Scalability results, where Π′ stands for a magic program and Π for an original program

Acknowledgements

This work was funded by NSFC grants 62206169 and 61972243. It is also partially supported by the EPSRC projects OASIS (EP/S032347/1), ConCuR (EP/V050869/1) and UK FIRES (EP/S019111/1), as well as 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. We thank Prof. Boris Motik for proofreading the manuscript and providing insightful comments.

References

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

Bellomarini, L.; Blasi, L.; Nissl, M.; and Sallinger, E. 2022. The Temporal Vadalog System. In *Rules and Reasoning - International Joint Conference on Rules and Reasoning, RuleML+RR*, volume 13752, 130–145.

Bellomarini, L.; Nissl, M.; and Sallinger, E. 2021. Monotonic Aggregation for Temporal Datalog. In *15th International Rule Challenge, 7th Industry Track, and 5th Doctoral Consortium @ RuleML+RR*, volume 2956.

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

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

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

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

Faber, W.; Greco, G.; and Leone, N. 2007. Magic Sets and their application to data integration. *J. Comput. Syst. Sci.*, 73(4): 584–609.

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

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

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

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

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

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

Tena Cucala, D. J.; Wałęga, P. A.; Cuenca Grau, B.; and Kostylev, E. V. 2021. Stratified Negation in Datalog with Metric Temporal Operators. In *AAAI Conference on Artificial Intelligence*, 6488–6495.

Ullman, J. D. 1989a. Bottom-up beats top-down for datalog. In *Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS, 140–149.

Ullman, J. D. 1989b. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press.

Wałęga, P.; Zawidzki, M.; and Cuenca Grau, B. 2021. Finitely Materialisable Datalog Programs with Metric Temporal Operators. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, 619–628.

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

Wałęga, P. A.; Cuenca Grau, B.; Kaminski, M.; and Kostylev, E. V. 2019. DatalogMTL: Computational Complexity and Expressive Power. In *International Joint Conference on Artificial Intelligence, IJCAI*, 1886–1892.

Wałęga, P. A.; Cuenca Grau, B.; Kaminski, M.; and Kostylev, E. V. 2020a. DatalogMTL over the integer timeline. In *International Conference on Principles of Knowledge Representation and Reasoning, 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 *International Joint Conference on Artificial Intelligence, IJCAI*, 1919–1925.

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

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

Wałęga, P. A.; Tena Cucala, D. J.; Cuenca Grau, B.; and Kostylev, E. V. 2024. The stable model semantics of datalog with metric temporal operators. *Theory and Practice of Logic Programming*, 24(1): 22–56.

Wałęga, P. A.; Tena Cucala, D. J.; Kostylev, E. V.; and Cuenca Grau, B. 2021. DatalogMTL with Negation Under Stable Models Semantics. In *International Conference on Principles of Knowledge Representation and Reasoning, KR*, 609–618.

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

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