Ontology-Based Data Access with a Horn Fragment of Metric Temporal Logic (Extended Version)

S. Brandt¹, E. Güzel Kalaycı², R. Kontchakov², V. Ryzhikov², G. Xiao² and M. Zakharyaschev³

¹Siemens CT, Germany  ²Free University of Bozen-Bolzano, Italy  ³Birkbeck, University of London, UK

Abstract

We advocate datalogMTL—a datalog extension of a Horn fragment of the metric temporal logic MTL—as a language for ontology-based access to temporal log data. We show that datalogMTL is EXPSPACE-complete even with punctual intervals, in which case MTL is known to be undecidable. Non-recursive datalogMTL turns out to be PSPACE-complete for combined complexity and in AC⁰ for data complexity. We demonstrate by two real-world use cases that nonrecursive datalogMTL programs can express complex temporal concepts from typical user queries and thereby facilitate access to log data. Our experiments with Siemens turbine data and MesoWest weather data show that datalogMTL ontology-mediated queries are efficient and scale on large datasets of up to 10GB.

1 Introduction

Data gathering at Siemens In order to prevent malfunctions and abnormal behavior, Siemens is operating remote-diagnostic centres in which data from installations worldwide, such as gas turbines for power generation, is being gathered and analysed. For the service engineers working in those centres, analysing the data often begins by running queries that aggregate sensor measurements such as the power output of the turbine, its maximum rotor speed, average exhaust temperature, etc. A typical query dealing with unexpected stops of a turbine might be find when an active power trip occurred, that is:

(ActivePowerTrip) the active power was above 1.5MW for a period of at least 10 seconds, maximum 3 seconds after which there was a period of at least one minute where active power was below 0.15MW.

Under the traditional workflow, an engineer would call an IT expert who would produce a specific script such as

\[
\text{message("active power TRIP") = }
\]
\[
\begin{align*}
&\text{eval( } >, \#\text{activePower}, 1.5 \text{ ) :} \\
&\quad \text{for( } > = 10s) \\
&\quad \quad \text{&}\  \\
&\quad \text{eval( } <, \#\text{activePower}, 0.15 \text{ ) :} \\
&\quad \quad \text{start( } \text{after[0s, 3s]} \text{ } \&\text{end } \text{for( } > = 1m); \\
\end{align*}
\]

Data gathering amounts to a major part of the time service engineers require for activities at Siemens remote-diagnostic centres, most of which due to the indirect access to data. The complexity of the task stems from the lacking of abstraction and the heterogeneity of the data sources, i.e., Siemens installations around the world.

OBDA Ontology-based data access (Poggi et al. 2008) offers a different workflow without the IT middleman. Domain experts develop an ontology providing definitions of the terms the engineers may be interested in together with mappings relating these terms to the database schemas. Modulo such an ontology, the query above could simply be ActivePowerTrip(t03)@x, where x is an answer variable over time intervals. Unfortunately, the OBDA ontology and query languages standardised by W3C—the OWL 2 QL profile of OWL 2 and SPARQL—are not suitable for the Siemens case as they were not designed to deal with essentially temporal data and concepts.

One approach to temporal OBDA is to use OWL 2 QL as an ontology language, assuming that ontology axioms hold at all times, and extend the query language with various temporal operators (Gutiérrez-Basulto and Klarmann 2012; Baader, Borgwardt, and Lippmann 2013; Borgwardt, Lippmann, and Thost 2013; Özçep et al. 2013; Klarmann and Meyer 2014; Özçep and Möller 2014; Kharlamov et al. 2016). However, OWL 2 QL is not able to define the temporal feature of ‘active power trip’, and so the engineer would have to capture it in a complex temporal query. Another known approach is to allow the temporal operators of linear temporal logic LTL in both queries and ontologies (Artale et al. 2013; 2015). However, sensor data come at irregular time intervals, which makes it impossible to adequately represent ‘10 seconds’ or ‘1 minute’ in LTL.

Metric temporal logic A more suitable formalism for capturing the meaning of concepts such as ‘active power trip’ is the logic MTL designed by Koymans (1990) and Alur and

<table>
<thead>
<tr>
<th>turbineId</th>
<th>dateTime</th>
<th>activePower</th>
<th>rotorSpeed</th>
<th>mainFlame</th>
</tr>
</thead>
<tbody>
<tr>
<td>t03</td>
<td>2015-04-04 12:20:48</td>
<td>...</td>
<td>1550</td>
<td>0</td>
</tr>
<tr>
<td>t03</td>
<td>2015-04-04 12:20:49</td>
<td>1.8</td>
<td>1400</td>
<td>null</td>
</tr>
<tr>
<td>t03</td>
<td>2015-04-04 12:20:52</td>
<td>1.7</td>
<td>1350</td>
<td>1</td>
</tr>
</tbody>
</table>

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
Henzinger (1993) for modelling and reasoning about real-time systems. MTLE can be interpreted over the reals (\(\mathbb{R}, \leq\)) and allows formulas such as \(\Box [5, 3] \varphi\) (or \(\Diamond [5, 3] \varphi\)) that holds at a moment \(t\) iff \(\varphi\) holds at every (respectively, some) moment in the interval \([t - 3, t - 1.5]\), which can easily capture the temporal feature of ‘active power trip’. Unfortunately, MTL turns out to be undecidable (Alur and Henzinger 1993) and EXPSPACE-complete if punctual operators such as \(\Diamond [1, 1]\) are disallowed (Alur, Feder, and Henzinger 1996).

Our contribution In this paper, we first investigate the Horn fragment of MTL (without the diamond operators in the head of rules) and its datalog extension datalogMTL, where ‘active power trip’ can be defined by the rule

\[
\text{ActivePowerTrip}(v) \leftarrow \text{Turbine}(v) \land \\
\Box[0,1\text{m}] \text{ActivePowerBelow0.15}(v) \land \\
\Diamond[0.05,0.634] \Box[0,1\text{m}] \text{ActivePowerAbove1.5}(v),
\]  

(1)

which is assumed to hold at all times. We show that answering ontology-mediated queries (\(\Pi, G(v)@x\)) is EXPSPACE-complete, where \(\Pi\) is a datalogMTL program, \(G(v)\) a goal with individual variables \(\nu\) and \(x\) a variable for intervals during which \(G(v)\) holds. We also observe that hornMTL becomes undecidable if the diamond operators are allowed in the head of rules.

From the practical point of view, most interesting are nonrecursive datalogMTL queries, where query answering is in \(AC^0\) for data complexity and PSPACE-complete for combined complexity (even NP-complete if the arity of predicates is bounded). In this case, we develop a query answering algorithm that can be implemented in standard SQL (with window functions). We also present a framework for practical OBDA with nonrecursive datalogMTL queries and temporal log data stored in databases as above.

Finally, we evaluate our framework on two use cases. We develop a datalogMTL ontology for temporal concepts used in typical queries at Siemens (e.g., NormalStop that takes place if events ActivePowerOff, MainFlameOff, CoastDown6600to1500, and CoastDown1500to200 happen in a certain temporal pattern). We also create a weather ontology defining standard meteorological concepts such as Hurricane (HurricaneForceWind, wind with the speed above 118 km/h, lasting 1 hour or longer). Using sensor databases from Siemens and MesoWest historical records of the weather stations across the US, we demonstrate experimentally that our algorithm is efficient in practice and scales on large datasets of up to 10GB.

Omitted proofs and details of the experiments can be found in the anonymous archive at http://tinyurl.com/zuwowu5.

2 DatalogMTL

In our applications, the intended flow of time is the real numbers \(\mathbb{R}\) (but the results of this section hold also for rational numbers \(\mathbb{Q}\)).

By an interval, \(\iota\), we mean any nonempty subset of \(\mathbb{R}\) of the form \([t_1, t_2]\), \((t_1, t_2]\), \([t_1, t_2)\), or \((t_1, t_2)\) where \(t_1, t_2 \in \mathbb{Q} \cup \{-\infty, \infty\}\) with \(t_1 \leq t_2\). A range, \(\varphi\), is an interval with non-negative endpoints. The end-points of intervals and ranges are represented in binary. An individual term, \(\tau\), is an individual variable, \(v\), or a constant, \(a\). A datalogMTL program, \(\Pi\), is a finite set of rules of the form

\[
A^+ \leftarrow A_1 \land \cdots \land A_k \text{ or } \bot \leftarrow A_1 \land \cdots \land A_k,
\]

where \(k \geq 1\), each \(A_i\) is either an inequality \(\tau \neq \tau'\) or defined by the grammar

\[
A ::= \Pi(\tau_1, \ldots, \tau_m) \mid \BoxF A \mid \BoxB A \mid \BoxF \varphi A \mid \BoxB \varphi A
\]

and \(A^+\) does not contain any diamond operators \(\BoxF\) and \(\BoxB\). The atoms \(A_1, \ldots, A_k\) constitute the body of the rule, while \(A^+\) or \(\bot\) its head. As usual, we assume that every variable in the head of a rule also occurs in its body.

A data instance, \(D\), is a finite set of facts of the form \(P(c)@ti\), where \(P(c)\) is a ground atom and \(t\) is an interval, stating that \(P(c)\) holds throughout \(t\).

An interpretation, \(M\), is based on a domain \(\Delta \neq \emptyset\) (for the individual variables and constants). For any \(m\)-ary predicate \(P\), \(m\)-tuple \(a\) from \(\Delta\), and moment of time \(t \in \mathbb{R}\), the interpretation \(M\) specifies whether \(P\) is true on \(a\) at \(t\), in which case we write \(M, t \models P(a)\). Let \(\nu\) be an assignment of elements of \(\Delta\) to the individual variables (we adopt the standard name assumption: \(\nu(c) = c\), for every individual constant \(c\)). We then set inductively:

\[
M, t \models^\nu P(\tau) \iff M, t \models P(\nu(\tau)),
\]

\[
M, t \models^\nu \tau \neq \tau' \iff \nu(\tau) \neq \nu(\tau'),
\]

\[
M, t \models \BoxF A \iff M, s \models A \text{ for all } s \text{ with } s - t \in \Delta,
\]

\[
M, t \models \BoxB A \iff M, s \models A \text{ for all } s \text{ with } s - t \in \Delta,
\]

\[
M, t \models \BoxF \varphi A \iff M, s \models A \text{ for some } s \text{ with } s - t \in \Delta,
\]

\[
M, t \models \BoxB \varphi A \iff M, s \models A \text{ for some } s \text{ with } s - t \in \Delta.
\]

We say that \(M\) satisfies \(\Pi\) under \(\nu\) if, for all \(t \in \mathbb{R}\) and all rules \(A \leftarrow A_1 \land \cdots \land A_k\) in \(\Pi\), we have

\[
M, t \models^\nu A \text{ whenever } M, t \models^\nu A_i \text{ for } 1 \leq i \leq k
\]

(usual \(\mathbb{R}, t \models^\nu \bot\)). \(M\) is a model of \(\Pi\) and \(D\) if it satisfies \(\Pi\) under every assignment, and \(M, t \models P(c)\) for any \(P(c)@ti\) in \(\Pi\) and any \(t \in \iota\). \(\Pi\) and \(D\) are consistent if they have a model.

Note that ranges \(\varphi\) in the temporal operators can be punctual \(\{t, \iota\}\), in which case \(\Box[t, t]\) is equivalent to \(\Box[t, t]\) \(A\) to \(\Diamond[t, t]\) \(A\).

A datalogMTL query takes the form (\(\Pi, q(v, x)\)), where \(q(v, x) = Q(\tau)@x\varphi\), for some predicate \(Q\). \(v\) is a tuple of individual variables occurring in the terms \(\tau\), and \(x\) an interval variable. A certain answer to (\(\Pi, q(v, x)\)) over a data instance \(D\) is a pair \((c, e)\) such that \(c\) is a tuple of constants from \(D\) (of the same length as \(v\)), \(e\) an interval and, for any \(t \in \iota\) and any model \(M\) of \(\Pi\) and \(D\), we have \(M, t \models^\nu Q(\tau), \nu\) maps \(v\) to \(c\). For example, suppose \(D\) consists of the facts Turbine(00)(0)(0)@ti, ActivePowerAbove1.5(00)(0)(0)@ti, ActivePowerBelow0.15(00)(0)(0)@ti, and \(\Pi\) is just rule (1). Then any subinterval of \([13:00:17, 13:01:18]\) is a certain answer to (\(\Pi, \text{ActivePowerTrip}(00)(0)\).

By answering datalogMTL queries we understand the problem of checking whether a given pair \((c, e)\) is a certain answer to a given datalogMTL query (\(\Pi, q(v, x)\)) over
a given data instance $D$. Our first result is the following theorem, which is to be put in the context of that $MTL$ is undecidable over $\mathbb{R}$, and $\text{EXPSPACE}$-complete over the integers $\mathbb{Z}$ (Alur and Henzinger 1993).

**Theorem 1.** Answering datalog$MTL$ queries is $\text{EXPSPACE}$-complete (even in the propositional case) for combined complexity.

To give the intuition behind the proof, we note first that every datalog$MTL$ program $\Pi$ can be transformed (using polynomially-many fresh predicates) to a datalog$MTL$ program in normal form that contains only rules of the form

$$P(\tau) \leftarrow \bigwedge_{i \in I} P_i(\tau), \quad \bot \leftarrow \bigwedge_{i \in I} P_i(\tau),$$

(2)

$$\func{\exists}_p P(\tau) \leftarrow P'(\tau'), \quad \func{\forall}_p P(\tau) \leftarrow P'(\tau'),$$

(3)

$$P(\tau) \leftarrow \func{\exists}_p P'(\tau'), \quad P(\tau) \leftarrow \func{\forall}_p P'(\tau').$$

(4)

gives the same certain answers as $\Pi$. For example, the rule $P(\tau) \leftarrow P_1(\tau) \land \func{\exists}_p P_2(\tau_2)$ can be replaced by $P(\tau) \leftarrow P_1(\tau) \land P'_2(\tau_2)$ and $\func{\exists}_p P_2(\tau_2) \leftarrow P_2(\tau_2)$, where $P'_2$ is a fresh predicate of the same arity as $P_2$.

We now require the following notation. Given an interval $\iota$ and a range $\rho$, we set

$$\iota + \rho = \{t + k \mid t \in \iota \text{ and } k \in \rho\}.$$

For example, if $\iota = (t_h, t_e)$ and $\rho = [q_h, q_e]$, then $\iota + \rho$ is the interval $(t_h + q_h, t_e + q_e)$; if $\iota = (t_h, t_e)$ and $\rho = (q_h, q_e]$, then $\iota + \rho = (t_h + q_h, t_e + q_e]$.

Next, by $\iota - \rho$ we denote the maximal interval $\iota'$ such that $\iota' + \rho = \iota$.

Note that $\iota - \rho$ is only defined if there is such a $\iota'$ with $\iota' + \rho = \iota$ for $\iota \in \iota$, in which case we write $\iota \preceq \iota$. If $\iota \preceq \iota'$, then $\iota'$ is defined uniquely. For example, if $\iota = (t_h, t_e)$ and $\rho = [q_h, q_e]$, then $\iota - \rho$ is the interval $(t_h - q_h, t_e - q_e)$; if $\iota = (t_h, t_e)$ and $\rho = (q_h, q_e]$, then $\iota - \rho = (t_h - q_h, t_e - q_e]$.

Now, given a data instance $D$, we denote by $\Pi(D)$ the closure (by transfinite induction) of $D$ under the rules:

- (coal) if $P(c) \at \in \Pi(D)$, for $i \in I$, then we add $P(c) \at \bigcup_{i \in I} \iota_i$ to $D$;
- (horn) if $P(c) \leftarrow \bigwedge_{i \in I} P_i(c_i) \at \in \Pi(D)$ and $\bigcap_{i \in I} \iota_i \neq \emptyset$, then we add $P(c) \at \bigcap_{i \in I} \iota_i$ to $D$;
- ($\exists$-) if $\exists \quad P(c) \at \leftarrow P'(c')$ is an instance of a rule in $\Pi$ with $P(c) \at \in \Pi(D)$ and $\iota \neq \emptyset$, then we add $P(c) \at (t \in \iota)$ to $D$ (and similarly for $\forall$);
- (TIL) if $P(c) \at \quad P'(c')$ is an instance of a rule in $\Pi$ with $P'(c') \at \in \Pi(D)$, then we add $P(c) \at (t - \rho)$ to $D$ (and similarly for $\exists$);
- (TIL) if $P(c) \at \quad P'(c')$ is an instance of a rule in $\Pi$ with $P'(c') \at \in \Pi(D)$ and $\rho \preceq \iota$, then we add $P(c) \at (t - \rho)$ to $D$ (and similarly for $\exists$).

Define a canonical interpretation $\mathcal{C}_{\Pi(D)}$ whose object domain consists of the individual constants in $\Pi$ and $D$, and $\mathcal{C}_{\Pi(D), t} \models P(c) \at \Pi(D)$, for some $t \ni i$.

**Lemma 2.** (i) If $\iota \cap \iota' = \emptyset$ for some $i$, then $\Pi$ and $D$ are inconsistent; otherwise, $\mathcal{C}_{\Pi(D)}$ is the minimal model of $\Pi$ and $D$ in the sense that $P(c) \at \Pi(D)$ implies $\mathcal{M}_t, t \models P(c)$, for any model $\mathcal{M}$ of $\Pi$ and $D$ and any $i$.

(ii) A pair $(c, i)$ is a certain answer to $(\Pi, q(\nu, x))$ over $D$ consistent with $\Pi$ iff $\mathcal{C}_{\Pi(D), t} \models q(c)$ for all $t \ni i$.

Let $\delta$ be the greatest common divisor of the (rational) numbers that occur in $\Pi$ and $D$. Let grid$(\Pi, D)$ be the closure of these numbers under the operations $+1$ and $-1$. It is not hard to see that the order $(\text{grid}(\Pi, D), \leq)$ is isomorphic to $(\mathbb{Z}, \leq)$.

**Lemma 3.** For any ground $P(c)$ and any $t \in \text{grid}(\Pi, D)$, we either have $\mathcal{C}_{\Pi(D), t} \models P(c)$, for all $t' \ni (t, t + 1)$, or $\mathcal{C}_{\Pi(D), t'} \models \lnot P(c)$, for all $t' \ni (t, t + 1)$.

The $\text{EXPSPACE}$ upper bound in Theorem 1 can now be obtained by (exponential) reduction to $\text{LTL}$ over $\mathbb{Z}$, which is known to be $\text{PSPACE}$-complete (Sistla and Clarke 1985).

The diamond operators $\Diamond_p$ and $\Box_p$ are disallowed in the head of datalog$MTL$ rules for the following reason. Denote by $\text{datalog}^{MTL}$ the extension of datalog$MTL$ that allows arbitrary temporal operators in the head of rules. The extended language turns out to be much more powerful and can encode 2-counter Minsky machines, which gives the following theorem; cf. (Madhani, Krishna, and Pandya 2013).

**Theorem 4.** Answering datalog$MTL$ queries is undecidable.

As none of the datalog$MTL$ programs required in our use cases is recursive, we now consider the class datalog$_{nr}MTL$ of nonrecursive datalog$MTL$ programs. More precisely, for a program $\Pi$, let $\prec$ be the dependence relation on the predicate symbols in $\Pi$; $P \prec Q$ if $\Pi$ has a clause with $P$ in the head and $Q$ in the body. $\Pi$ is called nonrecursive if $P \prec Q$ does not hold for any predicate symbol $P$ in $\Pi$, where $\prec$ is the transitive closure of $\prec$.

For datalog$_{nr}MTL$ queries, one can define a finite order grid$(\Pi, D)$ of exponential size, for which Lemma 3 holds with two additional infinite intervals $(-\infty, \min)$ and $(\max, +\infty)$. Since grid$(\Pi, D)$ can be encoded in polynomial space, we can use a tableau-like top-down procedure to obtain a $\text{PSPACE}$ upper bound for answering nonrecursive datalog$MTL$ queries:

**Theorem 5.** Answering datalog$_{nr}MTL$ queries is $\text{PSPACE}$-complete (even in the propositional case) and in $\text{AC}^0$ for data complexity.

The following example shows how a datalog$_{nr}MTL$ program can generate all possible assignments of truth-values to given propositional variables, which is required in the proof of the $\text{PSPACE}$ lower bound of Theorem 5.

**Example 6.** Suppose we are given three propositional variables $p$, $q$, and $r$. Let $\Pi_3$ be a datalog$MTL$ program with the following rules:

$$\begin{align*}
\rho^\sigma &\leftarrow \rho^\sigma_p, \\
q^\sigma &\leftarrow q_{0}^\sigma \lor \Box^4\rho^\sigma_p, \\
p^\sigma &\leftarrow p_{1}^\sigma \lor \Box^4\rho^\sigma_p, \\
p_{0}^\sigma &\leftarrow p_{0}^\sigma \lor \Box^2\rho_p, \end{align*}$$

where $\sigma \in \{0, 1\}$, $s^1 = s$ and $s^0 \equiv s$, for any propositional variable $s$, and $A \leftarrow B \lor C$ is a shorthand for two rules $A \leftarrow B$ and $A \leftarrow C$. Suppose also that a data instance $D$ consists of the facts

$$p_0 \at \{0, 1\}, \quad p_0 \at \{1, 2\}, \quad q_{0}^\sigma \at \{0, 2\}, \quad p_{0}^\sigma \at \{2, 4\}, \quad r_{0}^\sigma \at \{0, 4\}, \quad \tau_{0}^\sigma \at \{4, 8\}.$$
Then the canonical model $\mathcal{C}_{II,D}$ is shown below:

<table>
<thead>
<tr>
<th>$r$</th>
<th>$r_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$p$</td>
<td>$p_0$</td>
</tr>
<tr>
<td>$\bar{p}$</td>
<td>$\bar{p}_0$</td>
</tr>
</tbody>
</table>

Note that the Horn fragment of the Halpern-Shoham logic $\mathcal{H}S$ is P-complete over dense orders but undecidable over discrete ones (Bresolin et al. 2016), while the Horn fragment of $LT L$ is P-complete without the next operator and $PSPACE$-complete (P-complete in the nonrecursive case) with the next operator over $\mathcal{Z}$ (Artale et al. 2014).

3 Implementing $datalog_{nr}$ MTL in SQL

In our applications, instead of the $PSPACE$ top-down tableau procedure we use a rewriting approach that produces an SQL query implementing a bottom-up evaluation. Namely, we rewrite a given $datalog_{nr}$ MTL query $\langle II, Q(\tau) \vert \forall x \rangle$ (with II in normal form) to an SQL query computing the certain answers $(e, i)$ to the query with maximal intervals $i$.

In a nutshell, the rewriting algorithm produces SQL views that apply the rules (coal), (horn), $(\sqcap, \tau^{-})$ and $(\rightarrow, \tau)$ above to the facts extracted from the given database using mappings. The algorithm starts with tables $P$ containing these facts and having all the temporal intervals sorted (which is usually the case for log data and mappings such as $M$ in the next section). The tables $P$ are coalesced by the algorithm from (Zhou, Wang, and Zaniolo 2006) into new tables for $P$ in time $O(|P|)$. By applying a rule $(\sqcap, \tau^{-})$ or $(\rightarrow, \tau)$ to $P$, we construct a table $P'$ in time $O(|P|^2)$. The rule (horn) is applied by a variant of the merge join algorithm in time $O(|I|^n |I|)$, where $n$ is the maximal number of individual tuples in the tables $P_i$, for $i \in I$, and $m$ the maximal number of interval tuples. We then form the union of the $k$-many constructed tables for the same $P$; as the intervals in them are sorted, we obtain a sorted table for $P$ in time $O(|I|^n |I|^m k)$ and coalesce it in linear time. Observe that the time required to compute the resulting table $P$ is polynomial in $n$ and linear in $m$, which explains linear patterns in our experiments below, where the size of individual tuples is fixed. To compute the table for the goal $Q$, we iterate the described procedure $d$ times, where $d$ is the length of the longest chain of predicates ordered by the dependence relation $\prec$ for II. Thus, the overall time required to compute the goal predicate $Q$ is exponential in $d$ and the size of II itself, polynomial in $n$ and linear in $m$ (for a fixed $II$).

4 Use Cases

We test the feasibility of OBDA with $datalog_{nr}$ MTL by querying Siemens turbine log data and MesoWest weather data. First, we briefly describe these use cases.

Siemens service centres store aggregated turbine sensor data in tables such as TB_Sensor. The data comes with (not necessarily regular) timestamps $t_1, t_2, \ldots$, and it is deemed that the values remain constant in every interval $[t_i, t_{i+1})$. Using a set of mappings, we extract from these tables a data instance containing ground facts such as

- ActivePowerAbove1.5(tb0)@[12:04:48, 12:04:49),
- ActivePowerAbove1.5(tb0)@[12:04:49, 12:05:24),
- RotorSpeedAbove1500(tb0)@[12:04:49, 12:04:49),
- MainFlameBelow0.1(tb0)@[12:04:49, 12:05:24).

For example, the first two of them are obtained from the table TB_Sensor using the following SQL mapping $M!$:

```sql
SELECT turbineId AS x, 
    lag(dateTime) over (partition by turbineId order by dateTime) AS t1, 
    TIME AS t2, "[" AS (",") AS )
FROM TB_Sensor
WHERE lag(rotorSpeed) > 1500
    over (partition by turbineId order by dateTime)
```

In terms of the basic predicates above, we define more complex ones that are used in queries posed by the Siemens engineers. Below is a snippet of our $datalog_{nr}$ MTL ontology (the complete ontology can be found in the online appendix).

- **NormalStop**($v$) ← CoastDown1500to200($v$) ∧
- $\langle 0.9m \rangle [\text{CoastDown6600to1500}(v) \land
  \langle 0.2m \rangle [\text{MainFlameOff}(v) \land
  \langle 0.2m \rangle [\text{ActivePowerOff}(v)]]],

- **MainFlameOff**($v$) ← $[0.1 \text{m} \text{a}] \text{MainFlameBelow0.1}(v),
- **ActivePowerOff**($v$) ← $[0.1 \text{m} \text{a}] \text{MainPowerBelow0.15}(v),
- **CoastDown6600to1500**($v$) ←
- $[0.3 \text{m} \text{a}] \text{RotorSpeedBelow1500}(v) \land
  \langle 0.2m \rangle [\text{MainFlameBelow0.3}(v),
- **CoastDown1500to200**($v$) ←
- $[0.3 \text{m} \text{a}] \text{RotorSpeedBelow200}(v) \land
  \langle 0.9m \rangle [\text{MainFlameBelow0.3}(v),
- **NormalRestart**($v$) ←
- $\text{NormalStart}(v) \land \langle 0.1 \text{m} \rangle [\text{NormalStop}(v)].
```

Since all the rules above are guarded, the rewritings of their predicates into SQL queries can be evaluated in linear time for data complexity.

MesoWest. The MesoWest\(^1\) project makes publicly available historical records of the weather stations across the US showing such parameters of meteorological conditions as temperature, wind speed and direction, amount of precipitation, etc. Each station outputs its measurements with

\(^1\)http://mesowest.utah.edu/
some periodicity, with the output at a time $t_{i+1}$ containing the accumulative (e.g., for precipitation) or averaged (e.g., for wind speed) value over the interval $[t_i, t_{i+1}]$. The data comes in a table Weather, which looks as follows:

<table>
<thead>
<tr>
<th>stationId</th>
<th>dateTime</th>
<th>airTemp</th>
<th>windSpeed</th>
<th>windDr</th>
<th>hourPrecip</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBVY</td>
<td>2013-02-15:15:14</td>
<td>8</td>
<td>45</td>
<td>10</td>
<td>0.05</td>
</tr>
<tr>
<td>KMNI</td>
<td>2013-02-15:15:21</td>
<td>6</td>
<td>123</td>
<td>240</td>
<td>0</td>
</tr>
<tr>
<td>KBVY</td>
<td>2013-02-15:15:24</td>
<td>8</td>
<td>47</td>
<td>10</td>
<td>0.08</td>
</tr>
<tr>
<td>KMNI</td>
<td>2013-02-15:15:31</td>
<td>6.7</td>
<td>115</td>
<td>220</td>
<td>0</td>
</tr>
</tbody>
</table>

One more table, Metadata, provides some atemporal meta information about the stations:

<table>
<thead>
<tr>
<th>stationId</th>
<th>county</th>
<th>state</th>
<th>latitude</th>
<th>longitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAWS</td>
<td>Clarendon</td>
<td>S. Carolina</td>
<td>33.58333</td>
<td>-80.21667</td>
</tr>
<tr>
<td>KMNI</td>
<td>Essex</td>
<td>Massachusetts</td>
<td>42.58361</td>
<td>-70.91639</td>
</tr>
</tbody>
</table>

The monitoring and historical analysis of the weather involves answering queries such as ‘find showery counties, where one station observes precipitation at the moment, while another one does not, but observed precipitation 30 minutes ago’.

We use SQL mappings over the Weather table similar to those in the Siemens case to obtain ground atoms such as

- \( \text{EastWind}(\text{KBVY})@\langle 15:14, 15:24 \rangle \)
- \( \text{HurricaneForceWind}(\text{PAWS})@\langle 15:21, 15:31 \rangle \)
- \( \text{Precipitation}(\text{KBVY})@\langle 15:14, 15:24 \rangle \)
- \( \text{TempAbove0}(\text{KBVY})@\langle 15:14, 15:24 \rangle \)
- \( \text{TempAbove0}(\text{PAWS})@\langle 15:21, 15:31 \rangle \)

(according to the standard definition, the hurricane force wind is above 118 km/h). On the other hand, mappings to the Metadata table provide atoms such as

- \( \text{LocatedInCounty}(\text{KBVY}, \text{Essex})@\langle -\infty, \infty \rangle \)
- \( \text{LocatedInState}(\text{KBVY}, \text{Massachusetts})@\langle -\infty, \infty \rangle \)

Our ontology contains definitions of various meteorological terms; a few examples are given below:

\[
\begin{align*}
\text{ShoweryCounty}(v) & \leftarrow \text{LocatedInCounty}(u_1, v) \land \\
\text{LocatedInCounty}(u_2, v) & \land \text{Precipitation}(u_1) \land \\
\text{NoPrecipitation}(u_2) & \land \Phi_{[0,30m]} \text{Precipitation}(u_2), \\
\square_{[0,1h]} \text{Hurricane}(v) & \leftarrow \square_{[0,1h]} \text{HurricaneForceWind}(v), \\
\text{HurricaneAffectedState}(v) & \leftarrow \text{LocatedInState}(u, v) \land \\
\text{Hurricane}(u), \\
\square_{[0,24h]} \text{ExcessiveHeat}(v) & \leftarrow \square_{[0,24h]} \text{TempAbove24}(v) \land \\
\Phi_{[0,24h]} \text{TempAbove41}(v), \\
\text{HeatAffectedCounty}(v) & \leftarrow \text{LocatedInCounty}(u, v) \land \\
\text{ExcessiveHeat}(u), \\
\text{CyclonePatternState}(v) & \leftarrow \text{LocatedInState}(u_1, v) \land \\
\text{LocatedInState}(u_2, v) & \land \text{LocatedInState}(u_3, v) \land \\
\text{LocatedInState}(u_4, v) & \land \text{EastWind}(u_1) \land \\
\text{NorthWind}(u_2) & \land \text{WestWind}(u_3) \land \text{SouthWind}(u_4). \\
\end{align*}
\]

5 Experiments

To evaluate the performance of the SQL queries produced by the datalog

\[
\text{MTL rewriting algorithm outlined above, we developed two benchmarks for our use cases. ! To the best of our knowledge, this work is the first on practical OBDA with temporal ontologies. No other systems have similar functionalities. We ran the experiments on an HP Proliant server with 24 Intel Xeon CPUs (@3.47GHz), 106GB of RAM and five 1TB 15K RPM HD. We used PostgreSQL as a database engine and for the following experiments we observed that the maximum physical memory consumption is 12.9GB.}
\]

\textbf{Siemens} provided us with a sample of data for one running turbine, which we denote by \textit{tb0}, over 4 days in the form of the table \textit{TB Sensor}. The data table was rather sparse, containing a lot of nulls, because different sensors recorded data with different frequencies. For example, \textit{ActivePower} arrived most frequently with average periodicity of 7 seconds, whereas the values for the field \textit{Main Flame} arrived most rarely, every 1 minute on average. We replicated this sample to imitate the data for one turbine over 10 various periods ranging from 32 to 320 months. The statistics of the data sets can be found in Table 1a. With a timeout of 30 minutes, we evaluated four queries \textit{ActivePowerTrip}($\textit{tb0}$$@x$), \textit{NormalStart}($\textit{tb0}$$@x$), \textit{NormalStop}($\textit{tb0}$$@x$), and \textit{NormalRestart}($\textit{tb0}$$@x$). The execution times are given in the picture below, which shows their linear growth in the number of the months and consequently the size of the data (confirming theoretical results).

Note that the normal restart (start) query timeouts on the data for more than 15 (respectively, 20) years, which is more than enough for the monitoring and diagnostic tasks at Siemens:

\textbf{MesoWest}. In contrast to the Siemens case, the weather tables contain very few nulls. Normally, the data values arrive with periodicity from 1 to 20 minutes. We tested the performance of our algorithm by increasing (i) the temporal span (with some necessary increase of the spatial spread) and (ii) the geographical spread of data. For (i), we took the New York state data for the 10 continuous periods between 2005 and 2014; see Table 1b. As each year around 70 new weather stations were added, our 10 data samples increase more than linearly in size. For (ii),
we fixed the time period of one year (2012) and linearly increased the data from 1 to 19 states (NY, NJ, MD, DE, GA, RI, MA, CT, LA, VT, ME, WV, NH, NC, MS, SC, ND, KY, SD); see Table 1c. In both cases, with a timeout of 30 minutes, we executed four datalog\textit{nr}MTL queries \texttt{ShoweryCounty(v) @ x}, \texttt{HurricaneAffectedState(NY) @ x}, \texttt{HeatAffectedCounty(v) @ x}, \texttt{CyclonePatternState(NY) @ x}. The execution times are given in the pictures below.

In the experiment (i) we observe a mild non-linear dependency (the growth is even closer to linear if measured in the size of the data sets, as 70 new stations are added each year on average). The experiment (ii) exhibits linear behaviour even though about 500 stations are added in each new data set. The linear behaviour in that case can be explained by the fact that the data can be naturally partitioned into ‘chunks’ according to the location states of the stations so that the chunks are independent in the sense that, for any query \( Q \), data sets \( D \) and chunks \( D' \), we have \( Q(D \cup D') = Q(D) | Q(D') \). Such linear performance is then realised by PostgreSQL taking advantage of proper indexes over individuals and intervals. Note that the cyclone pattern state query is most expensive because its definition includes a join of four atoms for winds in four directions, each with a large volume of instances.

Overall, the results of the experiments look very encouraging: our datalog\textit{nr}MTL query rewriting algorithm produces SQL queries that are executable by a standard database engine PostgreSQL in acceptable time over large sets of real-world temporal data of up to 10GB. The relatively challenging queries such as NormalRestart and CyclonePatternState require a large number of temporal joins, which turn out to be rather expensive. One promising optimisation could be using distributed computing techniques to further exploit spatial/temporal partitions.

### Table 1: The size of the data sets used in the experiments.

<table>
<thead>
<tr>
<th># states</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>17</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw size (GB)</td>
<td>1.2</td>
<td>2.4</td>
<td>3.1</td>
<td>3.9</td>
<td>5.1</td>
<td>6.1</td>
<td>7.1</td>
<td>8.1</td>
<td>9.2</td>
<td>10.0</td>
</tr>
<tr>
<td>total size (GB)</td>
<td>2.0</td>
<td>4.1</td>
<td>5.3</td>
<td>6.5</td>
<td>8.6</td>
<td>10.0</td>
<td>12.0</td>
<td>14.0</td>
<td>16.0</td>
<td>18.0</td>
</tr>
</tbody>
</table>

Raw size: the size of the data itself stored in PostgreSQL reported by the \texttt{pg\_relation\_size} function.

Total size: the size of the total data (including the index) stored in PostgreSQL reported by the \texttt{pg\_total\_relation\_size} function.

6 Conclusions and Future Work

To facilitate access to sensor temporal data with the aim of monitoring and diagnostics, we suggested the ontology language datalogMTL, a combination of datalog with the Horn fragment of the metric temporal logic MTL. We showed that answering datalogMTL queries is EXPSPACE-complete for combined complexity, but becomes undecidable if the diamond operators are allowed in the head of rules. We also proved that answering nonrecursive datalogMTL queries is PSPACE-complete for combined complexity and in AC\textit{0} for data complexity. We tested feasibility and efficiency of OBDA with datalog\textit{nr}MTL on two real-world use cases by querying Siemens turbine data and MesoWest weather data. Namely, we designed datalog\textit{nr}MTL ontologies defining typical concepts used by Siemens engineers and various meteorological terms, developed and implemented an algorithm rewriting datalog\textit{nr}MTL queries into SQL queries,
and then executed the SQL queries obtained by this algorithm from our ontologies over the Siemens and MesoWest data, showing their acceptable efficiency and scalability.

Based on these encouraging results, we plan to extend datalogMTL with the since and until operators and include our temporal OBDA framework into the Ontop platform (Calvanese et al. 2016). We are also working on the streaming data setting, where the challenge is to continuously evaluate queries over the incoming data.

Acknowledgements
This paper is supported by the EU under the large-scale integrating project (IP) Optique (Scalable End-user Access to Big Data), grant agreement n. FP7-318338. We thank Prof. Diego Calvanese for insightful discussions.

References

http://ontop.inf.unibz.it/
A Appendix

An extended version of this technical report is available at https://arxiv.org/abs/1703.08982.

A.1 Proofs

Theorem 1. Answering datalogMTL queries is ExpSpace-complete (even in the propositional case) for combined complexity.

Proof. Consider a datalogMTL query \( (II, Q(\tau, x)) \) and a data instance \( D \). Denote by \( \Pi(D) \) the closure (by transfinite induction) of \( D \) under the following rules:

(coal) If \( P(c)@t_i \in D \), for \( i \in I \), and \( \bigcup_{i \in I} t_i \) forms an interval, e.g., \( (0, 1) \cup [1, 1] \cup (1, 2) \) forms \( (0, 2) \), then add \( P(c)@\bigcup_{i \in I} t_i \) to \( D \).

(horn) If \( P(c) \leftarrow \bigwedge_{i \in I} P_i(c_i) \) is an instance of a rule in \( \Pi \) with \( P_i(c_i)@t_i \in D \) and \( \bigcap_{i \in I} t_i \neq \emptyset \), then add \( P(c)@\bigcap_{i \in I} t_i \) to \( D \).

(\( \sqcup \)) If \( P(c) \leftarrow \bigwedge_{i \in I} P_i(c_i) \) is an instance of a rule in \( \Pi \) with \( P_i(c_i)@t_i \in D \), then add \( P(c)@\bigcap_{i \in I} (t_i - \emptyset) \) to \( D \). For \( \exists \), \( P(c) \leftarrow P'(c') \) is an instance of a rule in \( \Pi \) with \( P'(c')@t \in D \), and \( t \subsetneq \emptyset \), then we add \( P(c)@t \) to \( D \). For \( P(c) \leftarrow \exists \, P'(c') \), we apply the mirror image of this operation.

Define an interpretation \( \mathcal{C}_{\Pi, D} \) whose object domain consists of the individual constants in \( \Pi \) and \( D \), and \( \mathcal{C}_{\Pi, D}, t \models P(c) \) iff \( P(c)@t \in \Pi(D) \), for some \( t \supseteq t \). The proof of the following lemma is routine:

Lemma 2. (i) If \( \exists \bigwedge_{i \in I} P_i(c_i) \in \Pi(D) \) for some \( i \). Then, if \( D \) and \( \Pi \) are consistent; otherwise, \( \Pi \) and \( D \) are consistent and \( \mathcal{C}_{\Pi, D} \) is their minimal model in the sense that \( P(c)@t \in \Pi(D) \) implies \( \mathfrak{M}, t \models P(c) \), for any \( t \in t \) and any model \( \mathfrak{M} \) of \( \Pi \) and \( D \).

(ii) A pair \( (c, t) \) is a certain answer to \( (II, \not\exists \, (\forall, x)) \) over \( D \) consistent with \( II \) iff \( \mathcal{C}_{\Pi, D}, t \models \exists \, q(c) \), for every \( t \in t \).

Let 1 be the greatest common divisor of the rational numbers that occur in \( \Pi \) and \( D \). Let \( \text{grid}(\Pi, D) \) be the closure of these numbers under the operations +1 and −1.

Theorem 3. For any ground atom \( P(c) \) and any \( t \in \text{grid}(\Pi, D) \), either \( \mathcal{C}_{\Pi, D}, t \models P(c) \), for all \( t \in (t, t + 1) \), or \( \mathcal{C}_{\Pi, D}, t \not\models P(c) \), for all \( t \in (t, t + 1) \).

Clearly, \( \text{grid}(\Pi, D) \) is isomorphic to \( \mathbb{Z} \) (with standard <). This observation allows us to reduce the query answering problem to satisfiability of propositional LTL-formulas.

First, we construct the grounding \( \Pi_0 \) of \( \Pi \), whose size is bounded by \( |\Pi| \cdot |D|^{|\Pi|} \). So, in what follows we consider only predicates of arity 0. Then, we divide all rational numbers in \( \Pi_0 \) and \( D \) by 1 and denote the resulting integer-range program and data instance by \( \Pi' \) and \( D' \), respectively. Observe that \( \Pi_0 \) contains the same numbers as \( \Pi \). So, the size of the binary representation of 1 cannot exceed the product of the denominators of the numbers. Thus, the sizes of \( \Pi' \) and \( D' \) are bounded by \( |\Pi| \cdot |D|^{|\Pi|} \cdot ((|\Pi| + |D|)) \) and \( |D| \cdot ((|\Pi| + |D|)) \), respectively.

The resulting propositional integer-range query answering problem can be encoded in LTL in the following way: we encode punctual intervals \([n, n] \) for \( n \in \mathbb{Z} \), using even moments 2\( n \) and open intervals \((n, n + 1) \), for \( n \in \mathbb{Z} \), using odd moments \( 2n + 1 \) (by Lemma 3, the propositions of \( \Pi' \) and \( D' \) keep their values constant throughout such open intervals). Two propositions, punctual and open, are used to distinguish between the two situations:

\[ \square \land \Diamond \text{punctual,} \]

where \( \Diamond \) and \( \square \) are standard LTL operators that mean ‘true at all moments of time’ and ‘true at the next moment of time’, respectively. It should be clear that we can assume ‘true at all moments of time’ and ‘true at the next moment of time’, respectively. It should be clear that we can assume that all boxes in \( \Pi' \) have either open \((m, n) \) or punctual \([n, n] \) ranges (for example, \( \square \text{punctual} \leadsto \Box \text{punctual} \)). Similarly, we can assume \( D' \) contains atoms with punctual and open intervals only.

Then, a rule of the form \( p \leftarrow \bigwedge_{i \in I} p_i \) in \( \Pi' \) is encoded as

\[ \square \text{punctual} \land \Diamond \text{punctual} \rightarrow \bigwedge_{i \in I} p_i \rightarrow p. \]

A rule \( \square \text{punctual} \land p \rightarrow p' \) with an open interval range as

\[ \Diamond \text{punctual} \land p \rightarrow \bigwedge_{k=2n+1} p. \]

A rule \( p \leftarrow (\square, n) \text{punctual} \) with a punctual interval as

\[ \square (\Diamond, n) \rightarrow \square \text{punctual} \rightarrow \bigwedge_{k=2m} p. \]

A rule \( p \leftarrow \square \text{punctual} \rightarrow (\Diamond, n) \text{punctual} \) with an open interval range as

\[ \square (\Diamond, n) \rightarrow (\Diamond, n) \rightarrow \bigwedge_{k=2m+1} p. \]

A rule \( p \leftarrow (\square, n) \text{punctual} \) with a punctual interval as

\[ (\square, n) \rightarrow (\Diamond, n) \rightarrow \bigwedge_{k=2m} p. \]

(The cases of rules with \( \Box \) are symmetric.)

The data instance, \( D' \), can be encoded similarly: \( p \in (\emptyset, n) \) as \( \bigwedge_{k=2n+1} p \) and \( p \in [n, n] \) as \( \square \text{punctual} \rightarrow p' \rightarrow p. \)

The size of the resulting LTL-formula is bounded by \( (|\Pi| \cdot |D|^{|\Pi|}) \cdot ((|\Pi| + |D|) \cdot 2^{(|\Pi| + |D|)} \cdot A \text{depth of the formula}) \), which, with the polynomial-space complexity of LTL, gives us an ExpSpace upper bound.

The matching lower bound (even for the propositional case with predicates of arity 0) can be obtained by encoding computations of a deterministic Turing on exponential tape (similar to PSPACE-hardness of horn-LTL with the key observation that we can use an operator ‘holds in 2n’ moments’ of size polynomial in \( n \)).

Theorem 4. Answering propositional datalogMTL queries is undecidable.
Proof. The proof below uses ideas of the proof in [Mandani, Krishna & Pandya, 2013], where undecidability is shown with non-Horn formulas. We present the proof below in the syntax of the Horn fragment of MTL, the propositional metric temporal logic (the presented formulas can easily be translated into the syntax of datalogMTL).

We reduce the following problem, which is known to be undecidable:

• given a 2-counter Minsky machine, determine whether it halts starting from 0 in both counters.

Suppose the machine with counters $C_1$ and $C_2$ has $n - 1$ instructions of the form

1. $i$: Inc($C_k$), goto $j$,
2. $i$: Dec($C_k$), goto $j$,
3. $i$: If $C_k = 0$ then $j_1$ else $j_2$,

where $i$, $j$, $j_1$ and $j_2$ are instruction indexes, and the $n$-th instruction is $n$: Halt. We encode successive configurations of the machine along the timeline in intervals of length 5: each interval begins with the instruction index (propositions $p_1, \ldots, p_n$); the first counter is encoded using proposition $c$ between moments 1 and 2 in the interval and the second counter using variable $c$ between 3 and 4; proposition $z$ is used as the complement of $c$. We require that the $p_i$ are true at a single point in the interval, only one $p_i$ can be true at any moment and $c$ does not occur outside the parts of the intervals that encode the two counters: for all $i$, we take

$$\Box (p_i \rightarrow \bigwedge_{j \neq i} \neg p_j \land \bigwedge_j \Phi_{[0,5]} p_j \land \Phi_{[0,1]} z \land \Phi_{[2,3]} z \land \Phi_{[4,5]} z),$$

$$\Box (z \land c \rightarrow \perp),$$

where $\Box$ abbreviates $\Phi_{[0,\infty]}$. The initial state is 1 and the two counters are zeros:

$$p_1 \land \Phi_{[0,1]} z \land \Phi_{[0,1]} z,$$

where $\Phi_{[1,1]}$ and $\Phi_{[3,3]}$. The machine eventually halts:

$$\Phi_{[0,\infty]} p_n.$$  

Each of the instructions of the machine is encoded in the following way (the instructions for $C_2$ are obtained by swapping 1 and 2):

$$\Box (p_i \rightarrow \Phi_{[0,1]} z \rightarrow \Box p_j),$$

$$\Box (p_i \rightarrow \Phi_{[0,1]} z \rightarrow \Box p_j),$$

$$\Box (p_n \rightarrow \Box p_n),$$

where $\Box$ abbreviates $\Phi_{[5,5]}$ and the formulas for manipulations of the counter are as follows:

$$\text{COPY} = \Phi_{(0,1)} (c \leftrightarrow \bigcirc c) \land$$
$$\Phi_{(0,1)} (z \leftrightarrow \bigcirc z),$$

$$\text{INC} = \Phi_{(0,1)} (c \rightarrow \bigcirc c) \land$$
$$\Phi_{(0,1)} (c \land \Phi_{(0,1)} z \rightarrow \bigcirc \Phi_{(0,1)} (c \land \Phi_{(0,1)} z)) \land$$
$$\Phi_{(0,1)} (z \land \Phi_{(0,1)} c \rightarrow \Box z) \land$$
$$\Phi_{(0,1)} (\bigcirc c \land \Phi_{(0,1)} c \rightarrow c) \land$$
$$\Phi_{(0,1)} ((c \land \Phi_{(0,1)} z) \rightarrow z) \land$$
$$\Phi_{(0,1)} (\bigcirc z \rightarrow z)$$

and DEC$_i$ is the mirror image of INC$_i$ in which every $\bigcirc$ is replaced by $\Phi_{[5,5]}$.

It can be verified that the conjunction $\varphi$ of the formulas above is satisfiable iff the 2-counter machine halts.

Theorem 5. Answering $\text{datalog}_{\tau \cdot MTL}$ queries is PSPACE-complete for combined complexity (even in the propositional case) and in $\text{AC}^0$ for data complexity.

Proof. Consider a $\text{datalog}_{\tau \cdot MTL}$ query $(\Pi, Q(\tau, x))$, a data instance $D$ and a pair $(c, e)$, where $c$ is a tuple and $\tau$ an interval. Denote by $\min D$ and $\max D$ the minimum and the maximum, respectively, number that occurs in intervals in $D$. Let $K$ be the greatest number occurring in $\Pi$ and $d$ the depth of $\Pi$. Set

$$M_t = \min D - K^d \quad \text{and} \quad M_r = \max D + K^d.$$

Consider the closure $\Pi(D)$ of $D$ defined as in the proof of Theorem 1. It can be shown by induction that every number in $\Pi(D)$ is between $M_t$ and $M_r$, which can be stored (in binary) in space polynomial in $|D| + |\Pi|$.

Define a finite order Grid($\Pi, D$) similarly to the proof of Theorem 1, where the first interval is $(-\infty, M_t)$ and the last one is $(M_r, \infty)$. There are exponentially many points in Grid($\Pi, D$).

Top down derivation algorithm. In the propositional case, our derivation tree uses the atoms of the shape $P^{\ominus}(n, m)$ and $P^{\ominus}[n, n]$, where $(n, m)$ and $[n, n]$ are intervals from Grid($\Pi, D$). We process the rules (horn), $(\Box_{\neq}, \rightarrow)$ and $(\rightarrow \Box_{\neq})$ as expected. The rule (coal) makes the branching degree of the derivation tree exponential. Each branch, however, is of polynomial length, and the derivation process is in PSPACE.

$\text{AC}^0$. Let $\text{num}_{\Pi \cdot D}(P)$ be defined, for an intentional predicate $P$, as the set of the numbers occurring in the statements $P(c) \in D$. For an extensional predicate $P$, let
\[
\text{num}_{11, D}(P) = \bigcup_{P(\tau) \in \mathcal{P}} \text{num}_{11, D}(P(\tau)),
\]

\[
\bigcup_{P(\tau) \in \mathcal{P}, P(\tau) \vdash P(\tau')} \{ \varrho, e, t + \varrho, t + e \mid t \in \text{num}_{11, D}(P') \},
\]

\[
\bigcup_{P(\tau) \in \mathcal{P}, P(\tau) \vdash \neg P(\tau')} \{ \varrho, e, t - \varrho, t - e \mid t \in \text{num}_{11, D}(P') \},
\]

where \( \varrho \) and \( e \) denote the left and the right endpoints of range \( \varrho \).

We construct the rewriting inductively following the definition of \( D \)-predicates of arity higher than \( d \)-arity. The reader. We construct an FO-formula \( \Phi \) together with all the numbers from \( \text{num}_{11, D} \). Clearly, the size of \( (n, m) \) is linear in \( |D| \) but exponential in \( |\Pi| \).

We are going to show how to construct the rewriting for a propositional \( \text{datalog}_{n, m} \)-MTL query \( (\Pi, Q(x)) \). The case of predicates of arity higher than 0 is left to the reader as a straightforward extension of the technique below.

For simplicity it is also assumed that all the intervals in \( D \) and \( \Pi \) are of the shape \( [t, s] \) (the case of arbitrary intervals is left to the reader). We construct an FO-formula \( \psi_Q(n, m) \) using the predicates \( n < m, n \leq m \) and \( \text{PLUS}(n, m, \ell) \) for \( n \) equals \( m + \ell \), where \( \ell \) is either a variable or a constant. We consider as input (as the domain for evaluating \( \psi_Q(n, m) \)) \( D^* \), which is \( D \) together with all the numbers from \( \text{num}_{11, D} \). It will follow from our construction that

\[ D^* \models \psi_Q(t, s) \iff \{ t, s \} \text{ is a certain answer to } (\Pi, Q(x)) \text{ over } D. \]

We construct the rewriting inductively following the definitions of predicates in \( \Pi \). For the intentional predicate \( P \), the rewriting is \( \varphi_P(n, m) \) obtained by “coalescing” of the formula \( \psi_P(n, m) = P(n, m) \). More precisely, let \( \varphi_P(n, m) \) be the conjunction of the following two formulas:

\[ (n = m) \rightarrow \exists n', m'(\psi_P(n', m') \land (n' \leq n \leq m')), \]
\[ (n < m) \rightarrow \forall \ell, \ell' ((n \leq \ell < \ell' \leq m) \rightarrow \exists n', m'(\psi_P(n', m') \land (n' \leq \ell') \land (\ell' \leq m'))) \]

Now, suppose we have the rewritings \( \varphi_P(n, m) \) for all \( P \rightarrow \Pi \) of \( \Pi \). First, consider disjunction \( \psi_P(n, m) \) of the following formulas:

\[ \bigvee_{P \vdash \Pi, \tau} \bigwedge_{P(i) \in \Pi} \bigwedge_{\text{num}_{11, D}(P(i))} \exists n', m'(\varphi_P(n', m') \land (n' \leq n \leq m')) \]

\[ \bigwedge_{\text{num}_{11, D}(P(\tau))} \bigwedge_{\neg P(\tau')} \exists n', m'(\varphi_P(n', m') \land \text{PLUS}(n, m, a)) \]

\[ \bigwedge_{\text{num}_{11, D}(P(\tau))} \bigwedge_{\neg P(\tau')} \exists n', m'(\varphi_P(n', m') \land \text{PLUS}(n', m, a)) \]

Figure 1: Algorithm of evaluating \( \text{datalog}_{n, m} \)-MTL in SQL.

\[ \bigvee_{P \vdash \Pi, \tau} \bigwedge_{\text{num}_{11, D}(P(\tau))} \bigwedge_{\neg P(\tau')} \exists n', m'(\varphi_P(n', m') \land \text{PLUS}(n, m, a) \land \text{PLUS}(n', m, b)) \]

A.2 Rewriting Algorithm

We propose an algorithm that, for a given \( \text{datalog}_{n, m} \)-MTL program \( \Pi \) in the normal form, produces an SQL query simulating the bottom up evaluation of \( \Pi \). The pseudo code algorithm is outlined in Figure 1. The function \( \text{rew} \) produces the SQL query that can be used to compute the certain answers to \( (\Pi, Q(\tau) \oplus x) \). This SQL query is evaluated over raw data tables \( R \) such as \( \text{TB}\_\text{Sensor} \).

For each predicate \( P \) in \( \Pi \), the rewriting algorithm generates a view (query) \( V_P \) such that \( V_P(R) \) comprises all certain answers to \( (\Pi, P(\tau, x)) \) with maximal intervals. The fact \( P(c) \oplus \langle t_1, t_2 \rangle \) is encoded in \( V_P(R) \) as the tuple \( \langle \text{null}, \langle t_1, t_2 \rangle \rangle \). We denote the four columns \( \langle \text{null}, \langle t_1, t_2 \rangle \rangle \) in \( V_P \) by \( i \) and the query that projects \( V_P \) on \( i \) by \( V_P.i \).

For an extensional predicate \( P \), \( V_P \) is defined by the union of all mappings \( P(\tau, i) \in \mathcal{M} \) and an SQL function coalesce that implements the coalescing algorithm...
from (Zhou, Wang, and Zaniolo 2006). For intentional $P$, we generate a view $V_r$, for every rule $r$ in $\Pi P$, using the function $\text{create-view}$ as follows:

\[ r = \exists \_P (\sigma) \leftarrow P^* (\tau') : \text{we obtain } V_r \text{ by (i) selecting the rows in } V_P^* \text{ that satisfy the condition } cond(\tau') \text{ where } cond(\tau') \text{ is consisting of equalities from shared variables in } \tau' \text{ and filters from constants in } \tau', \text{(ii) substituting in them all the intervals } t \text{ by } t + \varrho \text{ and } t + \varrho^* \text{ for } \exists \_P \text{ where } \varrho^* \text{ is defined by prefixing end-points with } - \text{ and swapping them; e.g., } (1, 2]^* \text{ is } [-2, -1]), \text{ and (iii) projecting on the columns from } \tau. \]

\[ r = P(\tau) \leftarrow \exists \_P P_1 (\tau_1) \land P_2 (\tau_2) : \text{we obtain } V_r \text{ similarly to the case above by taking the intervals } t - \varrho \text{ instead of } t + \varrho \text{ (analogously for } r = P(\tau) \leftarrow \exists \_P P_1 (\tau_1) \text{ but using } t + \varrho^* \text{ instead}) \]

\[ r = P(\tau) \leftarrow P_1 (\tau_1) \land P_2 (\tau_2) : \text{we obtain } V_r \text{ by temporal join of } V_P^* \text{ and } V_P^*. \text{ We select all the pairs of tuples } ((c_1, t_1), (c_2, t_2)) \text{ from } V_P^* (\mathcal{R}) \times V_P^* (\mathcal{R}) \text{ that satisfy } \tau_1 \text{ and } \tau_2. \text{ From this product we only need to consider the rows for which } t = t_1 \land t_2 \neq \emptyset \text{ and return } (c, t) \text{ where } c \text{ is a projection on the columns from } \tau \text{ of } (c_1, c_2). \]

We can then take $V_P$ as the union of all $V_r$, for $r$ that have $P$ in the head. To provide the correct certain answers for the predicates $Q$ that are defined in terms of $P$, and to return answers with maximal intervals, we need to apply the coalesce operator again. Note that time to compute $\text{coalesce}(V(\mathcal{R}))$ is $O(V(\mathcal{R}))$, for any view query $V$.

### A.3 Complete Siemens Ontology

The Siemens use case ontology also includes the $\text{datalog}_{\text{MTL}}$ rules defining the concept NormalStart, which is outlined below:

\[
\begin{align*}
\text{NormalStart} (v) &\leftarrow \text{STCtoRUCReached} (v) \land \\
& \Theta_{0,30s} \left[ \text{RapidChange1-2Reached} (v) \land \\
& \Theta_{0,5m} \left[ \text{PurgingIsOver} (v) \land \\
& \Theta_{0,11m} \left[ \text{PurgeAndIgnitionSpeedReached} (v) \land \\
& \Theta_{0,15s} \left[ \text{FromStandStillTo180} (v) \right) \right] \\
\text{STCtoRUCReached} (v) &\leftarrow \\
& \Theta_{0,30s} \left[ \text{RotorSpeedAbove4800} (v) \land \\
& \Theta_{0,2m} \left[ \text{RotorSpeedBelow4400} (v) \right] \\
\text{Rampchange1-2Reached} (v) &\leftarrow \\
& \Theta_{0,30s} \left[ \text{RotorSpeedAbove4400} (v) \land \\
& \Theta_{0,6,5m} \left[ \text{RotorSpeedBelow1500} (v) \right] \\
\text{PurgingIsOver} (v) &\leftarrow \\
& \Theta_{0,10s} \left[ \text{MainFlameOn} (v) \land \\
& \Theta_{0,10m} \left[ \text{RotorSpeedAbove1260} (v) \land \\
& \Theta_{0,2m} \left[ \text{RotorSpeedBelow1000} (v) \right] \\
\end{align*}
\]

### A.4 More details of the Evaluation

In this online archive\(^3\), we also provide one additional excel file for the evaluation details and one folder of the rewritten SQL queries. In Figure 2 we show the performance of

\(^3\)https://github.com/ontop/ontop-examples/tree/master/aaai-2017-MTL-datalog
query answering with respect to the total size of the data. We observe linear performances in all evaluations.