Querying Log Data with Metric Temporal Logic

Sebastian Brandt SEBASTIAN-PHILIPP.BRANDT@SIEMENS.COM
Siemens CT, München, Germany

Elem Güzel Kalaycı KALAYCI@INF.UNIBZ.IT
KRDB Research Centre, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

Vladislav Ryzhikov VLAD@DCS.BBK.AC.UK
Department of Computer Science and Information Systems, Birkbeck, University of London, U.K.

Guohui Xiao[*] XIAO@INF.UNIBZ.IT
KRDB Research Centre, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy

Michael Zakharyaschev MICHAEL@DCS.BBK.AC.UK
Department of Computer Science and Information Systems, Birkbeck, University of London, U.K.

Abstract

We propose a novel framework for ontology-based access to temporal log data using a datalog extension *datalogMTL* of the Horn fragment of the metric temporal logic *MTL*. We show that *datalogMTL* is EXPSPACE-complete even with punctual intervals, in which case full *MTL* is known to be undecidable. We also prove that nonrecursive *datalogMTL* is PSPACE-complete for combined complexity and in AC^0 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 temporal 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.

1. Introduction

In this paper, we present a new ontology-based framework for querying temporal log data. We begin by outlining this framework in the context of data gathering and analysis at Siemens, a leading manufacturer and supplier of systems for power generation, power transmission, medical diagnosis, and industry automation.

Data gathering at Siemens. For the Siemens equipment, analytics services are usually delivered by remote diagnostic centres that store data from the relevant industrial sites or individual equipment around the globe. The analytics provided at these centres falls into three categories: descriptive, predictive, and prescriptive. Descriptive analytics describes or quantifies in detail what has happened after an event. Predictive analytics aims to anticipate events before they occur and provide a window of opportunity for countermeasures. Prescriptive analytics aims to automate the process of suggesting underlying reasons for the predicted events and carrying out appropriate countermeasures. All these types of analytics heavily rely on the ability to recognise interesting events using sensor measurements or other machine data such as the power output of a gas turbine, its maximum rotor speed, average exhaust temperature, etc. For example, a service engineer at a Siemens remote diagnostic centre could be interested in active power trips of the turbine, that is, events when

*. Corresponding author.

(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 the active power was below 0.15MW.

Under the standard workflow, when facing the task of finding the active power trips of the turbine, the engineer would call an IT expert who would then produce a specific script (in a proprietary signal processing language developed by Siemens) such as

$$
\begin{aligned}
&\texttt{message}(\text{“}\texttt{active power trip}\text{”}) = \\
&\quad \$t1 : \texttt{eval}(>, \#\texttt{activePower}, 1.5) : \\
&\qquad \texttt{for}(>= 10s) \\
&\qquad \&\& \\
&\qquad \texttt{eval}(<, \#\texttt{activePower}, 0.15) : \\
&\qquad \texttt{start}(\texttt{after}[0s, 3s]\$t1 : \texttt{end}) : \\
&\qquad \texttt{for}(>= 1m);
\end{aligned}
$$

for the turbine aggregated data stored in a table TB_Sensor, which looks as follows:

turbineId	dateTime	activePower	rotorSpeed	mainFlame	...
		...			
tb0	2015-04-04 12:20:48	2	1550	0	
tb0	2015-04-04 12:20:49	1.8	1400	null	
tb0	2015-04-04 12:20:52	1.7	1350	1	
		...			

The result of running the script is a log with records such as

“2015-04-04 12:22:17 `active power trip` tb0”

where information about all the events is accumulated.

When facing the same task but for a different turbine, the engineer may have to call the IT expert once again because different models of turbines and sensors may have different log/database formats. Moreover, the storage platform for the sensor data often changes (thus, currently Siemens are pondering over migrating certain data to a cloud-based storage). Maintaining a set of scripts, one for each data source, does not provide an efficient solution since a query such as ‘find all the turbines that had an active power trip in May 2017’ would require an intermediate database with integrated data of active power trips. Another difficulty is that the definitions of events the engineer is interested in can also change. Some changes are minor, say the pressure threshold or the number of seconds in the active power trip definition, but some could be more substantial, such as ‘find the active power trips that were followed by a high pressure within 3 minutes that lasted for 30 seconds’. This modification would require rewriting the script above into a much longer one rather than using it as a module in the new definition.

The permanent involvement of an IT expert familiar with database technology incurs high costs for Siemens, and data gathering accounts for a major part of the time the service engineers spend at Siemens remote diagnostic centres, most of which due to the indirect access to data.

Ontology-based data access (OBDA) offers a different workflow that excludes the IT middleman from data gathering (Poggi, Lembo, Calvanese, De Giacomo, Lenzerini, & Rosati, 2008);

consult also (Xiao, Calvanese, Kontchakov, Lembo, Poggi, Rosati, & Zakharyaschev, 2018) for a recent survey. In a nutshell, the OBDA workflow in the Siemens context looks as follows. Domain experts develop and maintain an ontology that contains terms for the events the engineers may be interested in. IT experts develop and maintain mappings that relate these terms to the database schemas. The engineer can now use familiar terms from the ontology and a graphical tool such as OptiqueVQS (Soylu, Giese, Jiménez-Ruiz, Vega-Gorgojo, & Horrocks, 2016) to construct and run queries such as $\mathsf{ActivePowerTrip}(\mathrm{tb0})@x$. The task of the OBDA system such as Ontop (Rodriguez-Muro, Kontchakov, & Zakharyaschev, 2013; Calvanese, Cogrel, Komla-Ebri, Kontchakov, Lanti, Rezk, Rodriguez-Muro, & Xiao, 2017) will be, using the mappings, to rewrite the engineer's *ontology-mediated query* into an SQL query over the database and then execute it returning the time intervals x where the turbine with the ID tb0 had active power trips.

Unfortunately, the ontology and query languages designed for OBDA and standardised by the W3C—the *OWL 2 QL* profile of *OWL 2* and SPARQL—are not suitable for the Siemens case because they were not meant to deal with essentially *temporal* data, concepts and properties. There have been several attempts to develop temporal OBDA.

One approach is to use the same *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 & Klarman, 2012; Baader, Borgwardt, & Lippmann, 2013; Borgwardt, Lippmann, & Thost, 2013; Özçep, Möller, Neuenstadt, Zheleznyakov, & Kharlamov, 2013; Klarman & Meyer, 2014; Özçep & Möller, 2014; Kharlamov, Brandt, Jiménez-Ruiz, Kotidis, Lamparter, Mailis, Neuenstadt, Özçep, Pinkel, Svingos, Zheleznyakov, Horrocks, Ioannidis, & Möller, 2016). Unfortunately, *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 (or call an expert in temporal logic). Another known approach is to allow the temporal operators of the linear-time temporal logic *LTL* in both queries and ontologies (Artale, Kontchakov, Wolter, & Zakharyaschev, 2013; Artale, Kontchakov, Kovtunova, Ryzhikov, Wolter, & Zakharyaschev, 2015; Gutiérrez-Basulto, Jung, & Kontchakov, 2016a). For more details and further references, consult the recent survey (Artale, Kontchakov, Kovtunova, Ryzhikov, Wolter, & Zakharyaschev, 2017)[1].

However, standard *LTL* over a *discrete* timeline such as $(\mathbb{N}, \leq)$ or $(\mathbb{Z}, \leq)$ is not able to adequately represent the temporal data and knowledge in the Siemens use case because measurements are taken and sent *asynchronously* by *multiple* sensors at *irregular* time intervals, which can depend on the turbine model, sensor type, etc. To model measurements and events using discrete time, one could take a sufficiently small time unit (quantum), say 1 second, and encode 'active power was below 0.15MW for a period of one minute' by an *LTL*-formula of the form $\bigcirc_P p \wedge \bigcirc_P^2 p \wedge \dots \wedge \bigcirc_P^{60} p$, where $\bigcirc_P$ is the previous-time operator. One problem with this encoding is that it is clearly awkward, not succinct, and only works under the assumption that the active power is measured *each and every second*. If, for some reason, a measurement is missing as in $\mathsf{TB\_Sensor}$, the formula becomes inadequate. This problem can be solved by using the (more succinct) metric temporal logic *MTL* with operators like $\boxminus_{[1,60]}$ interpreted as 'at every time instant within the previous minute when a measurement was taken'. The satisfiability problem for the description logic $\mathcal{ALC}$ extended with such operators over $(\mathbb{N}, \leq)$ was investigated by Gutiérrez-Basulto, Jung, & Ozaki (2016b). A more fundamental issue with modelling turbine events using discrete time is that it only applies to data complying with the chosen quantum and requires amendments every time the quantum has to be

1. Surveys of early developments in temporal deductive databases are given by Baudinet, Chomicki, and Wolper (1993), Chomicki and Toman (1998).

set to a different value because of a new equipment or because asynchronous sensor measurements start to happen more frequently. Thus, a better way of modelling the temporal data and events under consideration is by means of a suitable fragment of *MTL* interpreted over *dense* time such as the rationals $(\mathbb{Q}, \leq)$ or reals $(\mathbb{R}, \leq)$. This would allow us to capture, for example, that one event, say a sharp temperature rise, happened *just before* (maybe a fraction of a quantum), and so possibly caused another event, say an emergency shutdown, which is a typical feature of an asynchronous behaviour of real-time systems where the actual time of event occurrences cannot be predicted at the modelling stage.

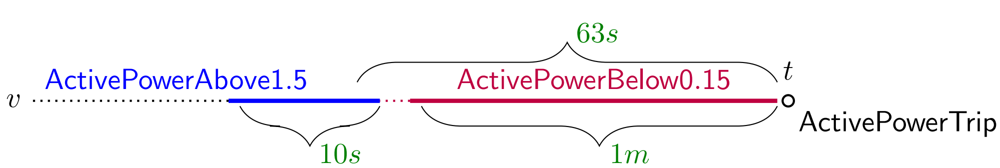
The *metric temporal logic MTL* was originally designed for modelling and reasoning about real-time systems (Koymans, 1990; Alur & Henzinger, 1993). *MTL* is equipped with two alternative semantics, *pointwise* and *continuous* (aka interval-based). In both semantics, the timestamps are taken from a dense timeline $(\mathbb{T}, \leq)$ such as $(\mathbb{Q}, \leq)$ or $(\mathbb{R}, \leq)$. Under the pointwise semantics, an interpretation is a *timed word*, that is, a finite or infinite sequence of pairs (Σ_i, t_i), where Σ_i is a subset of propositional variables that are assumed to hold at $t_i \in \mathbb{T}$ and $t_i < t_j$ for $i < j$. Under the continuous semantics, an interpretation is an assignment of a set of propositional variables to *each* $t \in \mathbb{T}$. *MTL* allows formulas such as $\boxplus_{[1.5,3]}\varphi$ (or $\diamondplus_{[1.5,3]}\varphi$) that holds at a moment t if and only if φ holds at every (respectively, some) moment in the interval $[t + 1.5, t + 3]$. However, under the pointwise semantics, t must be a timestamp from the timed word and φ must only hold at every (respectively, some) t_i with $1.5 \leq t_i - t \leq 3$. Thus, $\boxplus_{[1,1]}\bot$ is satisfiable under the pointwise semantics, for example, by a timed word with $t_{i+1} - t_i > 1$, but not under the continuous semantics.

In the Siemens case, we assume that the real-time system is being continuously monitored, the result of the next measurement of a sensor is only recorded when it exceeds the previous one by some fixed margin, and events such as active power trip can happen between measurements. This makes the continuous semantics a natural choice for temporal modelling. The satisfiability problem for *MTL* under this semantics turns out to be undecidable (Alur & Henzinger, 1993) and ExpSpace-complete if the punctual operators such as $\diamondplus_{[1,1]}$ are disallowed (Alur, Feder, & Henzinger, 1996); see also (Ouaknine & Worrell, 2005, 2008). Note that, under the pointwise semantics, *MTL* is decidable over finite timed words, though not primitive recursive (Ouaknine & Worrell, 2005).

Our contribution. Having analysed two real-world scenarios of querying asynchronous real-time systems (to be discussed in Section 6), we came to a conclusion that a basic ontology language for temporal OBDA should contain datalog rules with *MTL* operators in their bodies. In this language, for example, the event of active power trip can be defined by the rule

$$\begin{aligned}\mathsf{ActivePowerTrip}(v) \leftarrow \mathsf{Turbine}(v) \land \boxminus_{[0,1m]} \mathsf{ActivePowerBelow0.15}(v) \land \\ \diamondminus_{[60s,63s]} \boxminus_{[0,10s]} \mathsf{ActivePowerAbove1.5}(v). \quad (1)\end{aligned}$$

The variables of the predicates in such rules range over a (non-temporal) object domain. Thus, the intended domain for v in (1) comprises turbines, their parts, sensors, etc. The underlying (dense) timeline is implicit: we understand (1) as saying that $\mathsf{ActivePowerTrip}(v)$ holds at any given time instant t if the pattern shown in the picture below has occurred before t:



Unlike model-checking liveness properties (that some events eventually happen) in transition systems, our task is to query historical data for events that have already happened and are actually implicitly recorded in the data. As a consequence, we do not need ontology axioms with eventuality operators in the head such as $\diamondplus_{[0,3s]}\mathsf{ShutDown}(v) \leftarrow \mathsf{ActivePowerTrip}(v)$ saying that an active power trip must be followed by a shutdown within 3 seconds. *OWL 2 QL* allows existential quantification in the head of rules such as $\exists u\, \mathsf{hasRotor}(v, u) \leftarrow \mathsf{Turbine}(v)$ stating that every turbine has a rotor. Although axioms of this sort are present in the Siemens turbine configuration ontology (Kharlamov, Mailis, Mehdi, Neuenstadt, Özçep, Roshchin, Solomakhina, Soylu, Svingos, Brandt, Giese, Ioannidis, Lamparter, Möller, Kotidis, & Waaler, 2017), we opted not to include $\exists$ in the head of rules in our language. On the one hand, we have not found meaningful queries in the use cases for which such axioms would provide more answers. On the other hand, it is known that existential axioms may considerably increase the combined complexity of both atemporal (Gottlob, Kikot, Kontchakov, Podolskii, Schwentick, & Zakharyaschev, 2014; Bienvenu, Kikot, Kontchakov, Podolskii, & Zakharyaschev, 2018) and temporal ontology-mediated query answering (Artale et al., 2015). For these reasons, we do not allow existential rules in our ontology language and leave their investigation for future work.

The resulting temporal ontology language can be described as a datalog extension of the *Horn fragment* of *MTL* (without diamond operators in the head of rules). We denote this language by *datalogMTL* and prove in Section 3 that answering ontology-mediated queries of the form $(\Pi, G(\boldsymbol{v})@x)$ is ExpSpace-complete for combined complexity, where Π is a *datalogMTL* program, $G(\boldsymbol{v})$ a goal with individual variables $\boldsymbol{v}$, and x a variable over time intervals during which $G(\boldsymbol{v})$ holds. On the other hand, we show that *hornMTL* becomes undecidable if the diamond operators are allowed in the head of rules. We also prove that answering *propositional datalogMTL* queries is P-hard for data complexity. To compare, recall that answering ontology-mediated queries with propositional (not necessarily Horn) *LTL* ontologies is NC^1-complete for data complexity (Artale et al., 2015).

From the practical point of view, most interesting are *nonrecursive datalogMTL* queries. We show in Section 4 that answering such queries is in AC^0 for data complexity (assuming that data timestamps and the ranges of the temporal operators in *datalogMTL* programs are represented as finite binary fractions) 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 in Section 5 a framework for practical OBDA with nonrecursive *datalogMTL* queries and temporal log data stored in databases as shown above. Finally, in Section 6, 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 at least 1 hour). Using Siemens sensor databases and MesoWest historical records of the weather stations across the US, we experimentally demonstrate that our algorithm is efficient in practice and scales on large datasets of up to 8.3GB. We used two systems, PostgreSQL and Apache Spark, to evaluate our SQL programs. To our surprise, Apache Spark achieved tenfold better performance on the weather data than PostgreSQL. This effect can be attributed to the capacity of Spark to parallelise query execution as well as to the natural 'modularity' of weather data by location.

An extended abstract of this paper was presented at AAAI-17 (Brandt, Kalaycı, Kontchakov, Ryzhikov, Xiao, & Zakharyaschev, 2017).

2. Datalog*MTL*

In the standard metric temporal logic *MTL* (Alur et al., 1996), the temporal domain is the real numbers $\mathbb{R}$, while the intervals ϱ in the constrained temporal operators such as $\diamondplus_\varrho$ (sometime in the future within the interval ϱ from now) have natural numbers or ∞ as their endpoints. In the context of the applications of *MTL* we deal with in this paper, it is more natural to assume that the endpoints of ϱ are non-negative *dyadic rational numbers*—finite binary fractions[2] such as 101.011—or ∞. We denote the set of dyadic rationals by $\mathbb{Q}_2$ and remind the reader that $\mathbb{Q}_2$ is dense in $\mathbb{R}$ and, by Cantor's theorem, $(\mathbb{Q}_2, <)$ is isomorphic to $(\mathbb{Q}, <)$. By an *interval*, ι, we mean any nonempty subset of $\mathbb{Q}_2$ of the form $[t_1, t_2]$, $[t_1, t_2)$, $(t_1, t_2]$ or (t_1, t_2), where $t_1, t_2 \in \mathbb{Q}_2 \cup \{-\infty, \infty\}$ and $t_1 \leq t_2$. We identify $(t, \infty]$ with (t, ∞), $[-\infty, t]$ with $(-\infty, t]$, etc. A *range*, ϱ, is an interval with non-negative endpoints. The temporal operators of *MTL* take the form $\boxplus_\varrho$, $\diamondplus_\varrho$ and $\mathcal{U}_\varrho$, which refer to the future, and $\boxminus_\varrho$, $\diamondminus_\varrho$ and $\mathcal{S}_\varrho$, which refer to the past. The end-points of intervals and ranges are assumed to be represented in binary.

An *individual term*, τ, is an individual variable, v, or a constant, c. As usual, we assume that there is a countably-infinite list of predicate symbols, P, with assigned arities. A *datalogMTL program*, Π, is a finite set of *rules* of the form

$$A^+ \leftarrow A_1 \land \cdots \land A_k \qquad \text{or} \qquad \bot \leftarrow A_1 \land \cdots \land A_k,$$

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

$$A \ ::= \ P(\tau_1, \dots, \tau_m) \ \mid \ \top \ \mid \ \boxplus_\varrho A \ \mid \ \boxminus_\varrho A \ \mid \ \diamondplus_\varrho A \ \mid \ \diamondminus_\varrho A \ \mid \ A \,\mathcal{U}_\varrho\, A' \ \mid \ A \,\mathcal{S}_\varrho\, A'$$

and A^+ is given by the same grammar but *without* any 'non-deterministic' operators $\diamondplus_\varrho, \diamondminus_\varrho, \mathcal{U}_\varrho, \mathcal{S}_\varrho$. The atoms $A_1, \dots, A_k$ 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*, $\mathcal{D}$, is a finite set of *facts* of the form $P(\boldsymbol{c})@\iota$, where $P(\boldsymbol{c})$ is a ground atom (with a tuple $\boldsymbol{c}$ of individual constants) and ι an interval. The fact $P(\boldsymbol{c})@\iota$ states that $P(\boldsymbol{c})$ holds throughout the interval ι. We denote by $\mathsf{num}(\mathcal{D})$ the set of numbers (excluding $\pm\infty$) that occur in $\mathcal{D}$, and by $\mathsf{num}(\Pi, \mathcal{D})$ the set of number occurring in Π or $\mathcal{D}$.

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

$$\begin{aligned}
&\mathfrak{M}, t \models^\nu P(\boldsymbol{\tau}) \quad \text{iff} \quad \mathfrak{M}, t \models P(\nu(\boldsymbol{\tau})),\\
&\mathfrak{M}, t \models^\nu (\tau \neq \tau') \quad \text{iff} \quad \nu(\tau) \neq \nu(\tau'),\\
&\mathfrak{M}, t \models^\nu \boxplus_\varrho A \quad \text{iff} \quad \mathfrak{M}, s \models^\nu A \text{ for all } s \text{ with } s - t \in \varrho,\\
&\mathfrak{M}, t \models^\nu \boxminus_\varrho A \quad \text{iff} \quad \mathfrak{M}, s \models^\nu A \text{ for all } s \text{ with } t - s \in \varrho,\\
&\mathfrak{M}, t \models^\nu \diamondplus_\varrho A \quad \text{iff} \quad \mathfrak{M}, s \models^\nu A \text{ for some } s \text{ with } s - t \in \varrho,
\end{aligned}$$

2. In other words, a dyadic rational is a number of the form $n/2^m$, where $n \in \mathbb{Z}$ and $m \in \mathbb{N}$.

$\mathfrak{M}, t \models^\nu \diamondminus_\varrho A \quad$ iff $\quad \mathfrak{M}, s \models^\nu A$ for some s with $t - s \in \varrho$,

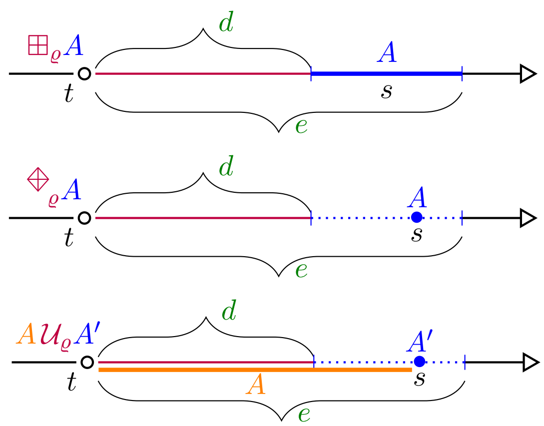
$\mathfrak{M}, t \models^\nu A \,\mathcal{U}_\varrho\, A' \quad$ iff $\quad \mathfrak{M}, t' \models^\nu A'$ for some t' with $t' - t \in \varrho$ and $\mathfrak{M}, s \models^\nu A$ for all $s \in (t, t')$,

$\mathfrak{M}, t \models^\nu A \,\mathcal{S}_\varrho\, A' \quad$ iff $\quad \mathfrak{M}, t' \models^\nu A'$ for some t' with $t - t' \in \varrho$ and $\mathfrak{M}, s \models^\nu A$ for all $s \in (t', t)$,

$\mathfrak{M}, t \models^\nu \top$,

$\mathfrak{M}, t \not\models^\nu \bot$.

The picture below illustrates the semantics of the 'future' operators for $\varrho = [d, e]$:



We say that $\mathfrak{M}$ *satisfies* a *datalogMTL* program Π under an assignment ν if, for *all* $t \in \mathbb{R}$ and all the rules $A \leftarrow A_1 \wedge \dots \wedge A_k$ in Π, we have

$$\mathfrak{M}, t \models^\nu A \quad \text{whenever} \quad \mathfrak{M}, t \models^\nu A_i \ \text{ for } 1 \leq i \leq k.$$

We call $\mathfrak{M}$ a *model* of Π and $\mathcal{D}$ and write $\mathfrak{M} \models (\Pi, \mathcal{D})$ if $\mathfrak{M}$ satisfies Π under every assignment, and $\mathfrak{M}, t \models P(\boldsymbol{c})$ for any $P(\boldsymbol{c})@\iota$ in $\mathcal{D}$ and any $t \in \iota$. Π and $\mathcal{D}$ are *consistent* if they have a model.

Note that ranges ϱ in the temporal operators can be punctual $[r, r]$, in which case $\boxplus_{[r,r]} A$ is equivalent to $\diamondplus_{[r,r]} A$, and $\boxminus_{[r,r]} A$ to $\diamondminus_{[r,r]} A$. We also observe that $\top \,\mathcal{S}_\varrho\, A$ is equivalent to $\diamondminus_\varrho A$ (that is, $\mathfrak{M}, t \models^\nu \top \,\mathcal{S}_\varrho\, A$ iff $\mathfrak{M}, t \models^\nu \diamondminus_\varrho A$ for all $\mathfrak{M}$, t and ν), and $\top \,\mathcal{U}_\varrho\, A$ is equivalent to $\diamondplus_\varrho A$.

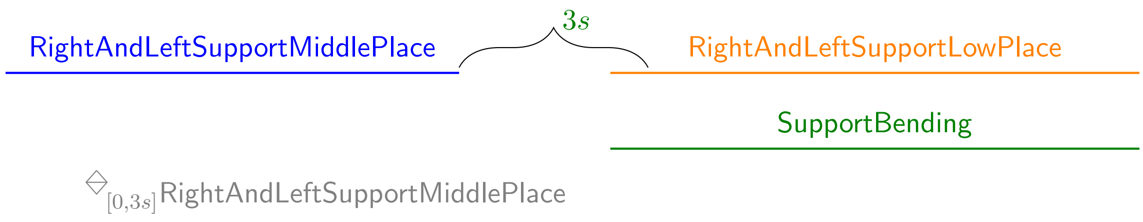
A *datalogMTL query* takes the form $(\Pi, \boldsymbol{q}(\boldsymbol{v}, x))$, where Π is a *datalogMTL* program and $\boldsymbol{q}(\boldsymbol{v}, x) = Q(\boldsymbol{\tau})@x$, for some predicate Q, $\boldsymbol{v}$ is a tuple of all individual variables occurring in the terms $\boldsymbol{\tau}$, and x an *interval variable*. A *certain answer* to $(\Pi, \boldsymbol{q}(\boldsymbol{v}, x))$ over a data instance $\mathcal{D}$ is a pair $(\boldsymbol{c}, \iota)$ such that $\boldsymbol{c}$ is a tuple of constants from $\mathcal{D}$ of the same length as $\boldsymbol{v}$, ι an interval and, for any $t \in \iota$, any model $\mathfrak{M}$ of Π and $\mathcal{D}$, and any assignment ν mapping $\boldsymbol{v}$ to $\boldsymbol{c}$, we have $\mathfrak{M}, t \models^\nu Q(\boldsymbol{\tau})$. In this case, we write $\mathfrak{M}, t \models \boldsymbol{q}(\boldsymbol{c})$. If the tuple $\boldsymbol{v}$ is empty (that is, $Q(\boldsymbol{\tau})$ does not have any individual variables), then we say that ι is a *certain answer* to $(\Pi, \boldsymbol{q}(x))$ over $\mathcal{D}$.

Example 1. Suppose that Π has one rule (1) and $\mathcal{D}$ consists of the facts

$$\begin{aligned}
&\mathsf{Turbine(tb0)}@(-\infty, \infty),\\
&\mathsf{ActivePowerAbove1.5(tb0)}@[13{:}00{:}00, 13{:}00{:}15),\\
&\mathsf{ActivePowerBelow0.15(tb0)}@[13{:}00{:}17, 13{:}01{:}25).
\end{aligned}$$

Then any subinterval of the interval $[13{:}01{:}17, 13{:}01{:}18)$ is a certain answer to the *datalogMTL* query $(\Pi, \mathsf{ActivePowerTrip(tb0)}@x)$.

Example 2. We illustrate the importance of the operators $\mathcal{S}$ (since) and $\mathcal{U}$ (until) using an example inspired by the ballet moves ontology (Raheb, Mailis, Ryzhikov, Papapetrou, & Ioannidis, 2017). Suppose we want to say that $\mathsf{SupportBending}$ is a move spanning from the beginning to the end of $\mathsf{RightAndLeftSupportLowPlace}$ provided that it is preceded by $\mathsf{RightAndLeftSupportMiddlePlace}$, which ends within $3s$ from the beginning of the $\mathsf{RightAndLeftSupportLowPlace}$, as shown below:



We can define the $\mathsf{SupportBending}$ move using the following rule:

$$\mathsf{SupportBending} \leftarrow \\ \mathsf{RightAndLeftSupportLowPlace}\ \mathcal{S}_{[0,\infty)}\ \big(\diamondminus_{[0,3s]}\mathsf{RightAndLeftSupportMiddlePlace}\big).$$

(note that a definition of $\mathsf{SupportBending}$ in *datalogMTL* would be problematic if only the $\Box$ and $\Diamond$ operators were available).

By *answering datalogMTL queries* we understand the problem of checking whether a given pair $(\boldsymbol{c}, \iota)$ is a certain answer to a given *datalogMTL* query $(\Pi, \boldsymbol{q}(\boldsymbol{v}, x))$ over a given data instance $\mathcal{D}$. The *consistency* (or *satisfiability*) *problem* is to check whether a given *datalogMTL* program Π is consistent with a given data instance $\mathcal{D}$. As usual in database theory (Vardi, 1982) and ontology-mediated query answering, we distinguish between the *combined complexity* and the *data complexity* of these problems: the former regards all the ingredients—Π, $\boldsymbol{q}(\boldsymbol{c}, \iota)$ and $\mathcal{D}$—as input, while the latter one assumes that Π and $\boldsymbol{q}$ are fixed and only $\mathcal{D}$ and $(\boldsymbol{c}, \iota)$ are the input.

Proposition 3. *Answering datalogMTL queries and consistency checking are polynomially reducible to the complement of each other.*

Proof. Suppose first that we want to check whether $(\boldsymbol{c}, \iota)$ is a certain answer to $(\Pi, \boldsymbol{q}(\boldsymbol{v}, x))$ over $\mathcal{D}$, where $\boldsymbol{q}(\boldsymbol{v}, x) = Q(\boldsymbol{\tau})@x$ and $\iota = [-t_1, t_2)$, $t_1, t_2 \in \mathbb{Q}_2^{\geq 0}$; other types of ι are considered analogously. Consider the following program Π' and data instance $\mathcal{D}'$:

$$\Pi' = \Pi \cup \{\bot \leftarrow P(\boldsymbol{v}) \wedge \boxminus_{[0,t_1]} Q(\boldsymbol{v}) \wedge \boxplus_{(0,t_2)} Q(\boldsymbol{v})\},$$
$$\mathcal{D}' = \mathcal{D} \cup \{P(\boldsymbol{c})@[0,0]\},$$

where P is a fresh predicate. It is readily seen that $(\boldsymbol{c}, \iota)$ is a certain answer to $(\Pi, \boldsymbol{q}(\boldsymbol{v}, x))$ over $\mathcal{D}$ iff Π' is *not* consistent with $\mathcal{D}'$. Conversely, Π and $\mathcal{D}$ are consistent iff $[0, 0]$ is *not* a certain answer to $(\Pi, P@x)$ over $\mathcal{D}$, where P is a fresh 0-ary predicate, that is, a propositional variable. $\Box$

We conclude this section by reminding the reader that, over the integer numbers $(\mathbb{Z}, <)$, *MTL* is as expressive as the *linear temporal logic LTL* with the operators $\bigcirc_F$ (at the next moment), $\mathcal{U}$ (until), $\Box_F$ (always in the future), $\Diamond_F$ (some time in the future) and their past counterparts $\bigcirc_P$, $\mathcal{S}$, $\Box_P$ and $\Diamond_P$. For example, the *LTL*-formula $\bigcirc_F A$ is equivalent to $\diamondplus_{[1,1]} A$ and $A\ \mathcal{U}\ B$ under the irreflexive semantics to $A \,\mathcal{U}_{(0,\infty)}\, B$; conversely, $\oplus_{[2,3]} A$ is clearly equivalent to the *LTL*-formula $\bigcirc_F \bigcirc_F A \vee \bigcirc_F \bigcirc_F \bigcirc_F A$. However, *MTL* operators are more succinct, which explains why *MTL*-satisfiability over $(\mathbb{Z}, <)$ is ExpSpace-complete (Alur & Henzinger, 1993; Furia & Spoletini, 2008) whereas *LTL*-satisfiability is PSpace-complete (Sistla & Clarke, 1985).

In the next section, we show that consistency checking for *datalogMTL* programs is ExpSpace-complete for combined complexity. It follows from Proposition 3 that answering *datalogMTL* queries is ExpSpace-complete as well. On the other hand, we also prove that answering propositional *datalogMTL* queries is P-hard for data complexity, and that the extension of *datalogMTL* with $\oplus$ in the head of rules leads to undecidability.

3. Complexity of answering *datalogMTL* queries

Observe first that every *datalogMTL* program Π can be transformed (using polynomially-many fresh predicates) to a *datalogMTL* program in *normal form* that only contains rules such as

$$P(\boldsymbol{\tau}) \leftarrow \bigwedge_{i \in I} P_i(\boldsymbol{\tau}_i), \qquad \bot \leftarrow \bigwedge_{i \in I} P_i(\boldsymbol{\tau}_i), \tag{2}$$

$$P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\, \mathcal{S}_\varrho\, P_2(\boldsymbol{\tau}_2), \qquad P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\, \mathcal{U}_\varrho\, P_2(\boldsymbol{\tau}_2), \tag{3}$$

$$P(\boldsymbol{\tau}) \leftarrow \boxminus_\varrho P_1(\boldsymbol{\tau}_1), \qquad P(\boldsymbol{\tau}) \leftarrow \boxplus_\varrho P_1(\boldsymbol{\tau}_1), \tag{4}$$

and gives the same certain answers as Π over any data instance. (In particular, *datalogMTL* programs in normal form do not contain occurrences of the diamond operators.) For example, we can replace the rule $\boxplus_{\varrho'} P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1) \wedge \boxminus_\varrho P_2(\boldsymbol{\tau}_2)$ in Π with three rules

$$\begin{aligned} &P'(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1) \wedge P_2'(\boldsymbol{\tau}_2), \\ &P_2'(\boldsymbol{\tau}_2) \leftarrow \boxminus_\varrho P_2(\boldsymbol{\tau}_2), \\ &P(\boldsymbol{\tau}) \leftarrow \top\, \mathcal{S}_{\varrho'}\, P'(\boldsymbol{\tau}), \end{aligned}$$

where P' is a fresh predicate of the same arity as P and P_2' a fresh predicate of the same arity as P_2. Moreover, we can only consider those programs and data instances where intervals take one of the following two forms:

– (t_1, t_2) with $t_1, t_2 \in \mathbb{Q}_2 \cup \{-\infty, \infty\}$,

– $[t, t]$ with $t \in \mathbb{Q}_2$; such intervals are called *punctual*.

For example, a data instance $\mathcal{D} = \mathcal{D}' \cup \{P(\boldsymbol{c})@(t_1, t_2]\}$ is equivalent to the data instance

$$\mathcal{D} = \mathcal{D}' \cup \{P(\boldsymbol{c})@(t_1, t_2),\ P(\boldsymbol{c})@[t_2, t_2]\}$$

in the sense that is gives the same certain answers as $\mathcal{D}$, the rule $P(v) \leftarrow \boxminus_{(r_1,r_2]} P'(v)$ is equivalent to $P(v) \leftarrow \boxminus_{(r_1,r_2)} P'(v) \wedge \boxminus_{[r_2,r_2]} P'(v)$, whereas the rule $P(v) \leftarrow P_1(v) \mathcal{U}_{(r_1,r_2]} P_2(v)$ is equivalent to the pair of rules

$$P(v) \leftarrow P_1(v)\, \mathcal{U}_{(r_1,r_2)}\, P_2(v), \quad P(v) \leftarrow P_1(v)\, \mathcal{U}_{[r_2,r_2]}\, P_2(v).$$

We use the following notations. We assume that $\langle$ is one of $($ and $[$, while $\rangle$ is one of $)$ and $]$. Given an interval $\iota = \langle \iota_b, \iota_e \rangle$ and a range ϱ, we set

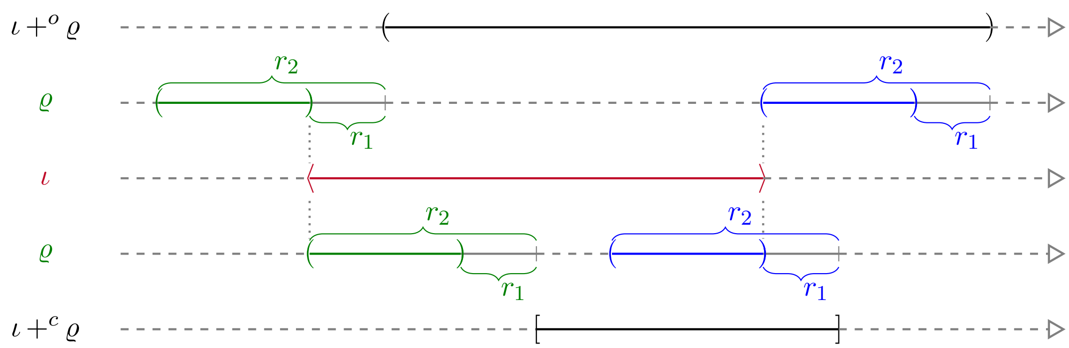
$$\iota +^o \varrho = \begin{cases} \langle \iota_b + r, \iota_e + r \rangle, & \text{if } \varrho = [r, r], \\ (\iota_b + r_1, \iota_e + r_2), & \text{if } \varrho = (r_1, r_2), \end{cases} \qquad \iota -^o \varrho = \begin{cases} \langle \iota_b - r, \iota_e - r \rangle, & \text{if } \varrho = [r, r], \\ (\iota_b - r_2, \iota_e - r_1), & \text{if } \varrho = (r_1, r_2). \end{cases}$$

In other words, $\iota +^o \varrho = \{t + k \mid t \in \iota \text{ and } k \in \varrho\}$ and $\iota -^o \varrho = \{t - k \mid t \in \iota \text{ and } k \in \varrho\}$. We also set

$$\iota -^c \varrho = \begin{cases} \langle \iota_b - r, \iota_e - r \rangle, & \text{if } \varrho = [r, r], \\ [\iota_b - r_1, \iota_e - r_2], & \text{if } \varrho = (r_1, r_2), \quad r_2, \iota_e \in \mathbb{Q}_2, \\ [\iota_b - r_1, \infty), & \text{if } \varrho = (r_1, r_2), \quad r_2 = \infty \text{ or } \iota_e = \infty, \end{cases}$$

$$\iota +^c \varrho = \begin{cases} \langle \iota_b + r, \iota_e + r \rangle, & \text{if } \varrho = [r, r], \\ [\iota_b + r_2, \iota_e + r_1], & \text{if } \varrho = (r_1, r_2), \quad r_2, \iota_b \in \mathbb{Q}_2, \\ (-\infty, \iota_e + r_1], & \text{if } \varrho = (r_1, r_2), \quad r_2 = \infty \text{ or } \iota_b = -\infty. \end{cases}$$

We assume that $\iota -^c \varrho$ and $\iota +^c \varrho$ are only defined if $r_2 - r_1 \leq \iota_e - \iota_b$, in which case we write $\varrho \sqsubseteq \iota$. Thus, $\iota -^c \varrho$ is defined if there is t' such that $t' + k \in \iota$, for all $k \in \varrho$. Symmetrically, $\iota +^c \varrho$ is defined if there is t' such that $t' - k \in \iota$. The picture below illustrates the intuition behind $\iota +^o \varrho$ and $\iota +^c \varrho$, for non-punctual ϱ, and the difference between them:



Furthermore, we write

- $\bigcap_{i \in I} \iota_i \neq \emptyset$ to say that the intersection of the intervals ι_i, for $i \in I$, is non-empty;
- $\bigcap_{i \in I} \iota_i$ for the intersection of the intervals ι_i provided that $\bigcap_{i \in I} \iota_i \neq \emptyset$; otherwise $\bigcap_{i \in I} \iota_i$ is undefined;
- $\bigcup_{i \in I} \iota_i$ for the union of the intervals ι_i provided that $\bigcup_{i \in I} \iota_i$ is a single interval; otherwise $\bigcup_{i \in I} \iota_i$ is undefined;
- ι^c for the *closure* of an interval ι, that is $\iota^c = [\iota_b, \iota_e]$ for any $\iota = \langle \iota_b, \iota_e \rangle$.

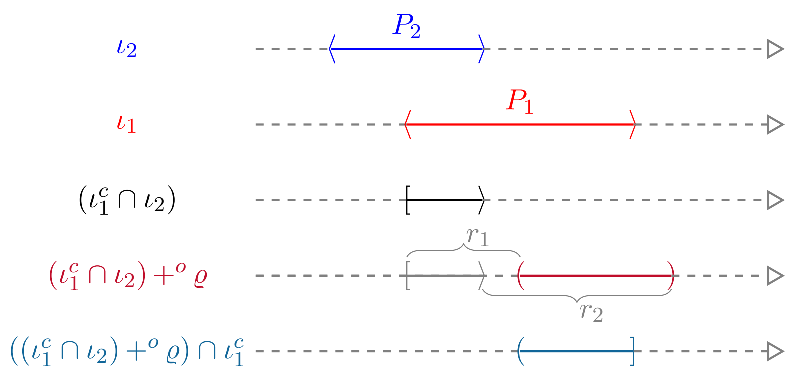
Suppose now that we are given a *datalogMTL* program Π (in normal form) and a data instance $\mathcal{D}$. We define a (possibly infinite) set $\mathfrak{C}_{\Pi,\mathcal{D}}$ of atoms of the form $P(\boldsymbol{c})@\iota$ or $\bot@\iota$ that contains all answers to *datalogMTL* queries with Π over $\mathcal{D}$. The construction is essentially the standard chase procedure from database theory (Abiteboul, Hull, & Vianu, 1995) adapted to time intervals and the temporal operators by mimicking their semantics. The only new chase rule is coalescing (coal) that merges—possibly infinitely-many—smaller intervals into the lager one they cover. Because of this rule, our chase construction requires transfinite recursion; cf. also (Bresolin, Kurucz, Muñoz-Velasco, Ryzhikov, Sciavicco, & Zakharyaschev, 2017; Artale et al., 2015).

Let $\mathcal{C}$ be some set of atoms of the form $P(\boldsymbol{c})@\iota$ or $\bot@\iota$ from Π and $\mathcal{D}$. Denote by $\mathsf{cl}(\mathcal{C})$ the result of applying exhaustively and non-recursively the following rules to $\mathcal{C}$:

(coal) if $P(\boldsymbol{c})@\iota_i \in \mathcal{C}$, for all $i \in I$ with a possibly infinite set I, and $\bigcup_{i\in I} \iota_i$ is defined, then we add $P(\boldsymbol{c})@\bigcup_{i\in I} \iota_i$ to $\mathcal{C}$;

(horn) if $P(\boldsymbol{c}) \leftarrow \bigwedge_{i\in I} P_i(\boldsymbol{c}_i)$ is an instance of a rule in Π with all $P_i(\boldsymbol{c}_i)@\iota_i$ in $\mathcal{C}$ and $\bigcap_{i\in I} \iota_i \neq \emptyset$, then we add $P(\boldsymbol{c})@\bigcap_{i\in I} \iota_i$ to $\mathcal{C}$; if $\bot \leftarrow \bigwedge_{i\in I} P_i(\boldsymbol{c}_i)$ is an instance of a rule in Π, then we add $\bot@\bigcap_{i\in I} \iota_i$ to $\mathcal{C}$;

$(\mathcal{S}_\varrho)$ if $P(\boldsymbol{c}) \leftarrow P_1(\boldsymbol{c}_1)\,\mathcal{S}_\varrho\,P_2(\boldsymbol{c}_2)$ is an instance of a rule in Π with $P_i(\boldsymbol{c}_i)@\iota_i \in \mathcal{C}$ for $i \in \{1,2\}$, $\iota_1^c \cap \iota_2 \neq \emptyset$, and $((\iota_1^c \cap \iota_2) +^o \varrho) \cap \iota_1^c \neq \emptyset$, then we add $P(\boldsymbol{c})@((\iota_1^c \cap \iota_2) +^o \varrho) \cap \iota_1^c$ to $\mathcal{C}$; see the picture below, where $\varrho = (r_1, r_2)$;



$(\boxplus_\varrho)$ if $P(\boldsymbol{c}) \leftarrow \boxplus_\varrho P_1(\boldsymbol{c}_1)$ is an instance of a rule in Π with $P_1(\boldsymbol{c}_1)@\iota \in \mathcal{C}$ and $\varrho \sqsubseteq \iota$, then we add $P(\boldsymbol{c})@(\iota -^c \varrho)$ to $\mathcal{C}$;

$(\mathcal{U}_\varrho)$ if $P(\boldsymbol{c}) \leftarrow P_1(\boldsymbol{c}_1)\,\mathcal{U}_\varrho\,P_2(\boldsymbol{c}_2)$ is an instance of a rule in Π with $P_i(\boldsymbol{c}_i)@\iota_i \in \mathcal{C}$, $\iota_1^c \cap \iota_2 \neq \emptyset$ and $((\iota_1^c \cap \iota_2) -^o \varrho) \cap \iota_1^c \neq \emptyset$, then we add $P(\boldsymbol{c})@((\iota_1^c \cap \iota_2) -^o \varrho) \cap \iota_1^c$ to $\mathcal{C}$;

$(\boxminus_\varrho)$ if $P(\boldsymbol{c}) \leftarrow \boxminus_\varrho P_1(\boldsymbol{c}_1)$ is an instance of a rule in Π with $P_1(\boldsymbol{c}_1)@\iota \in \mathcal{C}$ and $\varrho \sqsubseteq \iota$, then we add $P(\boldsymbol{c})@(\iota +^c \varrho)$ to $\mathcal{C}$.

We set $\mathsf{cl}^0(\mathcal{D}) = \mathcal{D} \cup \{\top(-\infty, \infty)\}$ and, for any successor ordinal $\xi + 1$ and limit ordinal ζ,

$$\mathsf{cl}^{\xi+1}(\mathcal{D}) = \mathsf{cl}(\mathsf{cl}^{\xi}(\mathcal{D})), \quad \mathsf{cl}^{\zeta}(\mathcal{D}) = \bigcup_{\xi<\zeta} \mathsf{cl}^{\xi}(\mathcal{D}) \quad \text{and} \quad \mathfrak{C}_{\Pi,\mathcal{D}} = \mathsf{cl}^{\omega_1}(\mathcal{D}), \tag{5}$$

where ω_1 is the first uncountable ordinal (as $\mathsf{cl}^{\omega_1}(\mathcal{D})$ is countable, there is an ordinal $\alpha < \omega_1$ such that $\mathsf{cl}^{\alpha}(\mathcal{D}) = \mathsf{cl}^{\beta}(\mathcal{D})$, for all $\beta \geq \alpha$). We regard $\mathfrak{C}_{\Pi,\mathcal{D}}$ as both a set of atoms of the form $P(\boldsymbol{c})@\iota$ or $\bot@\iota$ and an interpretation where, for any $t \in \mathbb{R}$, any P (different from $\bot$), and any tuple $\boldsymbol{c}$ of individual constants, we have $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P(\boldsymbol{c})$ iff $P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ and $t \in \iota$. The *domain* of $\mathfrak{C}_{\Pi,\mathcal{D}}$ is the set $\mathsf{ind}(\mathcal{D}) \cup \mathsf{ind}(\Pi)$ that comprises the individual constants occurring in $\mathcal{D}$ and Π.

We illustrate the definition above by a simple example:

Example 4. Let Π have two rules $P \leftarrow \boxminus_{[1,1]} P$ and $Q \leftarrow \boxplus_{(0,\infty)} P$, and let $\mathcal{D} = \{P(0,1]\}$. The first ω steps of the construction of $\mathfrak{C}_{\Pi,\mathcal{D}}$ will produce, using the rules $(\boxminus_\varrho)$ and (coal), the atoms $P(n, n+1]$ and $P(0, n+1]$, for $n < \omega$. In the step $\omega + 1$, (coal) will give $P(0,\infty)$ and then $(\boxplus_\varrho)$ will return $Q@[0,\infty)$.

Lemma 5. *Let Π be a datalogMTL program and $\mathcal{D}$ a data instance. Then, for any predicate symbol P from Π and $\mathcal{D}$, any tuple $\boldsymbol{c}$ of constants from $\mathcal{D}$ and Π, and any interval ι,*

(*i*) *$P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies $\mathfrak{M}, t \models P(\boldsymbol{c})$, for all $t \in \iota$ and all models $\mathfrak{M}$ of Π and $\mathcal{D}$;*

(*ii*) *if $\bot@\iota \notin \mathfrak{C}_{\Pi,\mathcal{D}}$ for any ι, then $\mathfrak{C}_{\Pi,\mathcal{D}} \models (\Pi, \mathcal{D})$; otherwise, Π and $\mathcal{D}$ are inconsistent.*

Proof. (*i*) Suppose that $\mathfrak{M}$ is a model of Π and $\mathcal{D}$, and that $P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$. Let ξ be the smallest ordinal such that $P(\boldsymbol{c})@\iota \in \mathsf{cl}^\xi(\mathcal{D})$. We show that $\mathfrak{M}, t \models P(\boldsymbol{c})$ for all $t \in \iota$ by induction of ξ. If $\xi = 0$, then $P(\boldsymbol{c})@\iota \in \mathcal{D}$, and since $\mathfrak{M}$ satisfies every assertion in $\mathcal{D}$, we are done. If $\xi = \xi' + 1$ then $P(\boldsymbol{c})@\iota$ was obtained from $\mathsf{cl}^{\xi'}(\mathcal{D})$ by applying one of the construction rules for $\mathfrak{C}_{\Pi,\mathcal{D}}$. Suppose $P(\boldsymbol{c})@\iota$ is $P(\boldsymbol{c})@\bigcup_{i \in I} \iota_i$ obtained by (coal). By the induction hypothesis, $\mathfrak{M}, t \models P(\boldsymbol{c})$ for all $t \in \iota_i$ and $i \in I$. Clearly, $\mathfrak{M}, t \models P(\boldsymbol{c})$ for all $t \in \bigcup_{i \in I} \iota_i$, and so for all $t \in \iota$. The case of (horn) is similar (with intersection in place of union).

Suppose $P(\boldsymbol{c})@\iota$ is obtained by $(\mathcal{S}_\varrho)$ from $P_i(\boldsymbol{c}_i)@\iota_i$, $i \in \{1,2\}$. By the induction hypothesis, $\mathfrak{M}, t \models P_i(\boldsymbol{c}_i)$ for every $t \in \iota_i$. Take an arbitrary $t \in ((\iota_1^c \cap \iota_2) +^o \varrho) \cap \iota_1^c$. Then there exists $t' \in \iota_1^c \cap \iota_2$ such that $t - t' \in \varrho$ and $\mathfrak{M}, t \models P_2(\boldsymbol{c}_2)$. Moreover, we have $\mathfrak{M}, s \models P_1(\boldsymbol{c}_1)$ for all $s \in (t', t)$. Therefore, $\mathfrak{M}, t \models P_1(\boldsymbol{c}_1) \, \mathcal{S} \, P_2(\boldsymbol{c}_2)$. If $P(\boldsymbol{c})@\iota$ is obtained by $(\boxplus_\varrho)$ from $P_1(\boldsymbol{c}_1)@\iota$, the proof is analogous by considering $t \in \iota -^c \varrho$. The remaining rules are treated similarly.

(*ii*) Suppose $\bot@\iota \notin \mathfrak{C}_{\Pi,\mathcal{D}}$ for any ι. By definition, $\mathcal{D} \subseteq \mathfrak{C}_{\Pi,\mathcal{D}}$, and so $\mathfrak{C}_{\Pi,\mathcal{D}} \models P(\boldsymbol{c})@\iota$ for every $P(\boldsymbol{c})@\iota \in \mathcal{D}$. To show that all the rules in Π are satisfied by $\mathfrak{C}_{\Pi,\mathcal{D}}$, we take an assignment ν, a rule $P(\boldsymbol{\tau}) \leftarrow \bigwedge_{i \in I} P_i(\boldsymbol{\tau}_i)$ from Π, and suppose that $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models^\nu P_i(\boldsymbol{\tau}_i)$, for all $i \in I$. By the definition of $\mathfrak{C}_{\Pi,\mathcal{D}}$, it follows that $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P_i(\nu(\boldsymbol{\tau}_i))$ and $P_i(\nu(\boldsymbol{\tau}_i)) \in \mathfrak{C}_{\Pi,\mathcal{D}}$, for some $\iota_i \ni t$. Moreover, there are ordinals ξ_i, $i \in I$, such that $P_i(\nu(\boldsymbol{\tau}_i))@\iota_i \in \mathsf{cl}^{\xi_i}(\mathcal{D})$. By the rule (horn), we then have $P(\nu(\boldsymbol{\tau}))@\bigcap_{i \in I} \iota_i \in \mathsf{cl}^{\max\{\xi_i \mid i \in I\}+1}(\mathcal{D})$, from which $P(\nu(\boldsymbol{\tau}))@\bigcap_{i \in I} \iota_i \in \mathfrak{C}_{\Pi,\mathcal{D}}$, and so $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P(\nu(\boldsymbol{\tau}))$. Now, consider a rule $\bot \leftarrow \bigwedge_{i \in I} P_i(\boldsymbol{\tau}_i)$ and suppose that $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models^\nu P_i(\boldsymbol{\tau}_i)$, for all $i \in I$. By the argument above, we then should have $\bot@\bigcap_{i \in I} \iota_i \in \mathfrak{C}_{\Pi,\mathcal{D}}$, which is a contradiction. For a rule $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1) \, \mathcal{S}_\varrho \, P_2(\boldsymbol{\tau}_2)$, take an arbitrary t and suppose that $\mathfrak{C}_{\Pi,\mathcal{D}}, t_2 \models^\nu P_2(\boldsymbol{\tau}_2)$ for some t_2 with $t - t_2 \in \varrho$ and $\mathfrak{C}_{\Pi,\mathcal{D}}, t_1 \models^\nu P_1(\boldsymbol{\tau}_1)$ for all $t_2 \in (t_2, t)$. By the construction of $\mathfrak{C}_{\Pi,\mathcal{D}}$, it follows that $P_2(\nu(\boldsymbol{\tau}_2))@\iota_2 \in \mathfrak{C}_{\Pi,\mathcal{D}}$ for some $\iota_2 \ni t_2$. Moreover, there are finitely many intervals ι_i', $i \in I$, such that $(t_2, t) \subseteq \bigcup_{i \in I} \iota_i'$ and $P_1(\nu(\boldsymbol{\tau}_1))@\iota_i' \in \mathfrak{C}_{\Pi,\mathcal{D}}$. By the rule (coal), $P_1(\nu(\boldsymbol{\tau}_1))@\iota_1 \in \mathfrak{C}_{\Pi,\mathcal{D}}$ for $\iota_1 = \bigcup_{i \in I} \iota_i'$. It follows then that $t_2, t \in \iota_1^c$, and so $\iota_1^c \cap \iota_2 \neq \emptyset$ and $t \in ((\iota_1^c \cap \iota_2) +^o \varrho) \cap \iota_1^c$. Thus, by the rule $(\mathcal{S}_\varrho)$, we have $P(\nu(\boldsymbol{\tau}))@((\iota_1^c \cap \iota_2) +^o \varrho) \cap \iota_1^c \in \mathfrak{C}_{\Pi,\mathcal{D}}$. Therefore, $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models^\nu P(\boldsymbol{\tau})$. The remaining rules are considered in the same manner.

That $\bot@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$, for some ι, implies inconsistency of $\mathcal{D}$ and Π follows from (*i*). $\square$

If $\bot@\iota \notin \mathfrak{C}_{\Pi,\mathcal{D}}$, we call $\mathfrak{C}_{\Pi,\mathcal{D}}$ the *canonical* (or *minimal*) *model of* Π *and* $\mathcal{D}$. We now establish an important property of $\mathfrak{C}_{\Pi,\mathcal{D}}$ that will allow us to reduce consistency checking for *datalogMTL* programs and data to the satisfiability problem for formulas in the linear temporal logic *LTL* over $(\mathbb{Z}, <)$.

Recall that the *greatest common divisor* of a finite set $N \subseteq \mathbb{Q}$ (at least one of which is not 0) is the largest number $\gcd(N) > 0$ such that every $n \in N$ is divisible by $\gcd(N)$ (in the sense that

$n/\gcd(N) \in \mathbb{Z}$). It is known that $\gcd(N)$ always exists and $\gcd(N) \leq \prod_{n\in N} |n|$. It is easy to see that, for any a finite set $N \subseteq \mathbb{Q}_2$ (at least one of which is not 0), we have $\gcd(N) = 2^m$, where m is the maximal natural number such that $n/2^m \in N$ is an irreducible fraction. Thus, $\gcd(N)$ can be computed and stored using space polynomial in $|N|$ (the size of the binary encoding of N). To make further definitions simpler, it will be convenient to assume that $\gcd(N) = 1$ if $N = \{0\}$.

Given a *datalogMTL* program Π and a data instance $\mathcal{D}$, we take $d = \gcd(\mathsf{num}(\Pi, \mathcal{D}))$. Denote by $\mathsf{sec}_{\Pi,\mathcal{D}}$ the set of all the intervals of the form $[kd, kd]$ and $((k-1)d, kd)$, for $k \in \mathbb{Z}$. Clearly, $\mathsf{sec}_{\Pi,\mathcal{D}}$ is a partition of $\mathbb{Q}_2$. We represent $\mathsf{sec}_{\Pi,\mathcal{D}}$ as

$$\mathsf{sec}_{\Pi,\mathcal{D}} = \{\dots, \sigma_{-3}, \sigma_{-2}, \sigma_{-1}, \sigma_0, \sigma_1, \sigma_2, \sigma_3, \dots\},$$

where $\sigma_0 = [0,0]$, $\sigma_1 = (0,d)$, $\sigma_2 = [d,d]$, $\sigma_3 = (d, 2d)$, $\sigma_{-1} = (-d, 0)$, etc. Thus, σ_i is punctual if i is even and non-punctual if i is odd. We refer to the σ_i as *sections* of $\mathsf{sec}_{\Pi,\mathcal{D}}$.

Lemma 6. *For every atom $P(\boldsymbol{c})$ and every $\sigma \in \mathsf{sec}_{\Pi,\mathcal{D}}$, we either have $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P(\boldsymbol{c})$ for all $t \in \sigma$, or $\mathfrak{C}_{\Pi,\mathcal{D}}, t \not\models P(\boldsymbol{c})$ for all $t \in \sigma$.*

Proof. It suffices to show that every interval ι such that $P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ takes one of the following forms: $(-\infty, \infty)$, $\langle dk, \infty)$, $(-\infty, dk\rangle$, $\langle dk, dk'\rangle$, where $k, k' \in \mathbb{Z}$. This can readily be done by induction on the construction of $\mathfrak{C}_{\Pi,\mathcal{D}}$. Indeed, when applied to a set of atoms of this form, the operator cl also results in a set of such atoms. $\square$

Our aim now is to encode the structure of $\mathfrak{C}_{\Pi,\mathcal{D}}$ given by Lemma 6 by means of an *LTL*-formula $\varphi_{\Pi,\mathcal{D}}$ that is satisfiable over $(\mathbb{Z}, <)$ iff Π and $\mathcal{D}$ are consistent. The *LTL*-formula $\varphi_{\Pi,\mathcal{D}}$ contains *propositional variables* of the form $P^{\boldsymbol{c}}$, where P is a predicate symbol from Π and $\mathcal{D}$ of arity m and $\boldsymbol{c}$ an m-tuple of individual constants from $\mathcal{D}$ and Π, as well as two additional propositional variables odd and even. We define $\varphi_{\Pi,\mathcal{D}}$ as a conjunction of the following clauses, where ν is any assignment of the individual constants from $\mathcal{D}$ and Π to the terms in Π, and $\Box\psi$ is a shorthand for $\Box_P\varphi \wedge \varphi \wedge \Box_F\varphi$:

- $\mathsf{even} \wedge \Box(\mathsf{even} \to \bigcirc_F \mathsf{odd}) \wedge \Box(\mathsf{odd} \to \bigcirc_F \mathsf{even})$;
- $\Box(P^{\nu(\boldsymbol{\tau})} \leftarrow \bigwedge\limits_{i\in I} P_i^{\nu(\boldsymbol{\tau}_i)})$, for every rule $P(\boldsymbol{\tau}) \leftarrow \bigwedge_{i\in I} P_i(\boldsymbol{\tau}_i)$ in Π;
- $\Box(\bot \leftarrow \bigwedge\limits_{i\in I} P_i^{\nu(\boldsymbol{\tau}_i)})$, for every rule $\bot \leftarrow \bigwedge_{i\in I} P_i(\boldsymbol{\tau}_i)$ in Π;
- for every rule $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\, \mathcal{S}_\varrho\, P_2(\boldsymbol{\tau}_2)$ in Π with $\varrho = [r, r]$, we require two clauses:

$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{even} \wedge \bigcirc^{-2r/d} P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigwedge_{-2r/d<j<0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big),$$
$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{odd} \wedge \bigcirc^{-2r/d} P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigwedge_{-2r/d\leq j\leq 0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big),$$

  where $\bigcirc^n\varphi = \underbrace{\bigcirc_F \dots \bigcirc_F}_{n} \varphi$ if $n > 0$, $\bigcirc^0\varphi = \varphi$, and $\bigcirc^n\varphi = \underbrace{\bigcirc_P \dots \bigcirc_P}_{|n|} \varphi$ if $n < 0$;

– for every rule $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\,\mathcal{S}_\varrho\,P_2(\boldsymbol{\tau}_2)$ in Π with $\varrho = (r_1, r_2)$, we require four clauses:

$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{even} \wedge \bigvee_{-2r_2/d<k<-2r_1/d} (\bigcirc^k P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigcirc^k \mathsf{even} \wedge \bigwedge_{k<j<0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big),$$
$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{even} \wedge \bigvee_{-2r_2/d<k<-2r_1/d} (\bigcirc^k P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigcirc^k \mathsf{odd} \wedge \bigwedge_{k\leq j<0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big),$$
$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{odd} \wedge \bigvee_{-2r_2/d\leq k\leq-2r_1/d} (\bigcirc^k P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigcirc^k \mathsf{even} \wedge \bigwedge_{k<j\leq 0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big),$$
$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{odd} \wedge \bigvee_{-2r_2/d\leq k\leq-2r_1/d} (\bigcirc^k P_2^{\nu(\boldsymbol{\tau}_2)} \wedge \bigcirc^k \mathsf{odd} \wedge \bigwedge_{k\leq j\leq 0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)}\big);$$

– for every rule $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\,\mathcal{S}_\varrho\,P_2(\boldsymbol{\tau}_2)$ in Π with $\varrho = (r_1, \infty)$,

$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{even} \wedge \bigwedge_{-2r_1/d\leq j<0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)} \wedge \bigcirc^{-2r_1/d}(P_1^{\nu(\boldsymbol{\tau}_1)}\,\mathcal{S}\,(\mathsf{even} \wedge P_2^{\nu(\boldsymbol{\tau}_2)}) \vee P_1^{\nu(\boldsymbol{\tau}_1)}\,\mathcal{S}\,(\mathsf{odd} \wedge P_1^{\nu(\boldsymbol{\tau}_1)} \wedge P_2^{\nu(\boldsymbol{\tau}_2)}))\big),$$
$$\Box\big(P^{\nu(\boldsymbol{\tau})} \leftarrow \mathsf{odd} \wedge \bigwedge_{-2r_1/d\leq j\leq 0} \bigcirc^j P_1^{\nu(\boldsymbol{\tau}_1)} \wedge \bigcirc^{-2r_1/d}(P_2^{\nu(\boldsymbol{\tau}_2)} \vee P_1^{\nu(\boldsymbol{\tau}_1)}\,\mathcal{S}\,(\mathsf{even} \wedge P_2^{\nu(\boldsymbol{\tau}_2)}) \vee P_1^{\nu(\boldsymbol{\tau}_1)}\,\mathcal{S}\,(\mathsf{odd} \wedge P_1^{\nu(\boldsymbol{\tau}_1)} \wedge P_2^{\nu(\boldsymbol{\tau}_2)}))\big)$$

(recall that $P\,\mathcal{S}\,Q$ holds at i iff there exists $k < i$, such that Q holds at k and P holds at all j with $k < j < i$);

– similar clauses for the rules of the form $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\,\mathcal{U}_\varrho\,P_2(\boldsymbol{\tau}_2)$ (here we need the 'until' operator $\mathcal{U}$), $P(\boldsymbol{\tau}) \leftarrow \boxminus_\varrho P_1(\boldsymbol{\tau}_1)$ and $P(\boldsymbol{\tau}) \leftarrow \boxplus_\varrho P_1(\boldsymbol{\tau}_1)$ in Π;

– for every fact $P(\boldsymbol{c})@\iota$ in $\mathcal{D}$, we need the clauses:

$$\begin{array}{ll}
\bigcirc^{2r/d} P^{\boldsymbol{c}}, & \text{if } \iota = [r, r],\\
\bigwedge_{2r_1/d<i<2r_2/d} \bigcirc^i P^{\boldsymbol{c}}, & \text{if } \iota = (r_1, r_2) \text{ and } r_1, r_2 \in \mathbb{Q}_2,\\
\bigcirc^{2r_1/d} \Box_F P^{\boldsymbol{c}}, & \text{if } \iota = (r_1, r_2), r_1 \in \mathbb{Q}_2 \text{ and } r_2 = \infty,\\
\bigcirc^{2r_2/d} \Box_P P^{\boldsymbol{c}}, & \text{if } \iota = (r_1, r_2), r_1 = -\infty \text{ and } r_2 \in \mathbb{Q}_2,\\
\Box_F \Box_P P^{\boldsymbol{c}}, & \text{if } \iota = (r_1, r_2), r_1 = -\infty \text{ and } r_2 = \infty.
\end{array}$$

Lemma 7. (Π, D) *is consistent iff* $\varphi_{\Pi,\mathcal{D}}$ *is satisfiable.*

Proof. $(\Rightarrow)$ If $\mathfrak{C}_{\Pi,\mathcal{D}}$ is a model of $(\Pi, \mathcal{D})$, we define an *LTL*-interpretation $\mathfrak{M}$ by taking

– $\mathfrak{M}, i \models P^{\boldsymbol{c}}$ iff $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P(\boldsymbol{c})$, for all $t \in \sigma_i$ and $i \in \mathbb{Z}$, all tuples of individual constants $\boldsymbol{c}$, and predicates P;

– $\mathfrak{M}, i \models \mathsf{even}$, for even $i \in \mathbb{Z}$;

– $\mathfrak{M}, i \models \mathsf{odd}$, for odd $i \in \mathbb{Z}$.

It is routine to check that $\mathfrak{M}, 0 \models \varphi_{\Pi,\mathcal{D}}$, taking into account that $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P_1(\boldsymbol{c}_1)\, \mathcal{S}_\varrho\, P_2(\boldsymbol{c}_2)$ for some (= all) $t \in \sigma_i$ iff the following conditions hold:

Case $\varrho = [r, r]$: $\mathfrak{C}_{\Pi,\mathcal{D}}, t' \models P_2(\boldsymbol{c}_2)$, for some $t' \in \sigma_{i-2r/d}$, and $\mathfrak{C}_{\Pi,\mathcal{D}}, s \models P_1(\boldsymbol{c}_1)$ for all $s \in \sigma_j$ such that

$$\begin{aligned} &i - 2r/d < j < i, && \text{if } i \text{ is even,}\\ &i - 2r/d \leq j \leq i, && \text{if } i \text{ is odd;} \end{aligned}$$

Case $\varrho = (r_1, r_2)$: there exists σ_k with $\mathfrak{C}_{\Pi,\mathcal{D}}, t' \models P_2(\boldsymbol{c}_2)$, for some $t' \in \sigma_k$, and $\mathfrak{C}_{\Pi,\mathcal{D}}, s \models P_1(\boldsymbol{c}_1)$ for all $s \in \sigma_j$ such that

$$\begin{aligned} &i - 2r_2/d < k < i - 2r_1/d, && k < j < i, && \text{if } i \text{ is even and } k \text{ is even,}\\ &i - 2r_2/d < k < i - 2r_1/d, && k \leq j < i, && \text{if } i \text{ is even and } k \text{ is odd,}\\ &i - 2r_2/d \leq k \leq i - 2r_1/d, && k < j \leq i, && \text{if } i \text{ is odd and } k \text{ is even,}\\ &i - 2r_2/d \leq k \leq i - 2r_1/d, && k \leq j \leq i, && \text{if } i \text{ is odd and } k \text{ is odd;} \end{aligned}$$

and similarly for the other temporal operators in $\varphi_{\Pi,\mathcal{D}}$.

($\Leftarrow$) Suppose now that $\varphi_{\Pi,\mathcal{D}}$ is satisfiable. Take the canonical model $\mathfrak{M}$ of $\varphi_{\Pi,\mathcal{D}}$ such that $\mathfrak{M}, 0 \models \varphi_{\Pi,\mathcal{D}}$; see (Artale, Kontchakov, Ryzhikov, & Zakharyaschev, 2013) for details. Using the observations above, it is not hard to check that $\mathfrak{M}, i \models P^{\boldsymbol{c}}$ iff $\mathfrak{C}_{\Pi,\mathcal{D}}, t \models P(\boldsymbol{c})$, for all $t \in \sigma_i$ and $i \in \mathbb{Z}$, all tuples of individual constants $\boldsymbol{c}$ and predicates P. Details are left to the reader. $\square$

We are now in a position to prove our first complexity result:

Theorem 8. *Consistency checking for datalogMTL programs is* EXPSPACE*-complete. The lower bound holds even for propositional datalogMTL.*

Proof. We first show the upper bound. By the two lemmas above, a *datalogMTL* program Π is consistent with a data instance $\mathcal{D}$ iff the *LTL* formula $\varphi_{\Pi,\mathcal{D}}$ is satisfiable. Thus, a consistency checking EXPSPACE algorithm can first construct $\varphi_{\Pi,\mathcal{D}}$, which requires exponential time in the size of Π and $\mathcal{D}$. Indeed, the greatest common divisor of the set $\mathsf{num}(\Pi, \mathcal{D})$ can be computed in polynomial time. The *LTL* formula $\varphi_{\Pi,\mathcal{D}}$ contains exponentially many clauses (as there are exponentially many assignments ν) of at most exponential size (as they contain $2t/d$ conjuncts or disjuncts, where t is a number from Π or $\mathcal{D}$). After that we can run a standard PSPACE satisfiability checking algorithm for *LTL*; see, e.g., (Sistla & Clarke, 1985).

We establish the matching lower bound by reduction of the non-halting problem for deterministic Turing machines with an exponential tape. Let M a deterministic Turing machine that requires $2^{f(m)}$ cells of the tape given an input of length m, for some polynomial f. Let $n = f(m)$. Without loss of generality, we can assume that M never runs outside the first 2^n cells. Suppose $M = (Q, \Gamma, \#, \Sigma, \delta, q_0, q_h)$, where Q is a finite set of states, Γ a tape alphabet, $\# \in \Gamma$ the blank symbol, $\Sigma \subseteq \Gamma$ a set of input symbols, $\delta\colon (Q \setminus \{q_h\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ a transition function, and $q_0, q_h \in Q$ are the initial and halting states, respectively. Let $\vec{a} = a_1 \dots a_m$ be an input for M. We construct a propositional *datalogMTL* program Π and a data instance $\mathcal{D}$ such that they are *not* consistent iff M accepts $\vec{a}$. In our encoding, we employ the following propositional variables, where $a \in \Gamma$, $q \in Q$:

– $H_{q,a}$ indicating that a cell is read by the head, the current state of the machine is q, and the cell contains a;

– N_a indicating that a cell is not read by the head and contains a,

– first and last marking the first and last cells of a configuration, respectively.

The program Π consists of the following rules, for $a, a', a'' \in \Gamma$, $q, q' \in Q$:

$$
\begin{array}{ll}
\boxplus_{2^n+1} H_{q',a''} \leftarrow H_{q,a} \land \boxplus_1 N_{a''}, \quad \boxplus_{2^n} N_{a'} \leftarrow H_{q,a}, & \text{if } \delta(q,a) = (q', a', R),\\
\boxplus_{2^n-1} H_{q',a''} \leftarrow H_{q,a} \land \boxminus_1 N_{a''}, \quad \boxplus_{2^n} N_{a'} \leftarrow H_{q,a}, & \text{if } \delta(q,a) = (q', a', L),\\
\boxplus_{2^n} N_a \leftarrow \boxminus_1 N_{a'} \land N_a \land \boxplus_1 N_{a''}, & \\
\boxplus_{2^n} N_a \leftarrow \boxminus_1 H_{q,a'} \land N_a \land \boxplus_1 N_{a''}, & \text{if } \delta(q,a') \neq (r,b,R) \text{ for all } r, b,\\
\boxplus_{2^n} N_a \leftarrow \boxminus_1 N_{a'} \land N_a \land \boxplus_1 H_{q,a''}, & \text{if } \delta(q,a'') \neq (r,b,L) \text{ for all } r, b,\\
\boxplus_{2^n} N_a \leftarrow N_a \land \mathsf{first} \land \boxplus_1 N_{a'}, & \\
\boxplus_{2^n} N_a \leftarrow N_a \land \mathsf{first} \land \boxplus_1 H_{q,a'}, & \text{if } \delta(q,a') \neq (r,b,L) \text{ for all } r, b,\\
\boxplus_{2^n} N_a \leftarrow \boxminus_1 N_{a'} \land N_a \land \mathsf{last}, & \\
\boxplus_{2^n} N_a \leftarrow \boxminus_1 N_{q,a'} \land N_a \land \mathsf{last}, & \text{if } \delta(q,a') \neq (r,b,R) \text{ for all } r, b,\\
\boxplus_{2^n} \mathsf{first} \leftarrow \mathsf{first}, & \\
\boxplus_{2^n} \mathsf{last} \leftarrow \mathsf{last}, & \\
\bot \leftarrow H_{q_h,a}, & \\
\boxplus_1 N_\# \leftarrow N_\# \land \diamondplus_{(0,\infty)} N_\#^<, &
\end{array}
$$

where $\boxplus_r$ is an abbreviation for $\boxplus_{[r,r]}$ and similarly for $\boxminus_r$. Let $\mathcal{D}$ contain the following facts:

$$
N_{a_i}@[i,i], \text{ for } 1 < i \leq m, \quad N_\#@[m+1, m+1], \quad N_\#^<@[2^n, 2^n],
$$
$$
H_{q_0,a_1}@[1,1], \quad \mathsf{first}@[1,1], \quad \mathsf{last}@[2^n, 2^n].
$$

The program represents the computation of M on $\vec{a}$ as a sequence of configurations. The initial one is spread over the time instants $1, \dots, 2^n$, from which the first m instants represent $\vec{a}$ and the remaining ones are $\#$. The second configuration uses the next 2^n instants (i.e., $2^n+1, \dots, 2^n+2^n$), etc. It is routine to check that M halts on $\vec{a}$ iff Π and $\mathcal{D}$ are inconsistent. $\square$

Note that *datalogMTL* allows *punctual intervals* of the form $[r,r]$ as ranges of temporal operators, and that full propositional *MTL* with such intervals is undecidable (Alur & Henzinger, 1993).

Now we turn to the data complexity of *datalogMTL* and show the following result:

Theorem 9. *Consistency checking and answering propositional* datalogMTL *queries is* P*-hard for data complexity* (*under* LOGSPACE *reductions*).

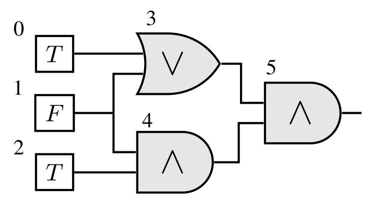
Proof. We establish this lower bound by reduction of the monotone circuit value problem, which is known to be P-complete (Arora & Barak, 2009). Let $\boldsymbol{C}$ be a monotone circuit with input gates having fan-in 1 and all other gates fan-in 2. We assume that the gates that are enumerated by consecutive positive integers, so that if there is an edge from n to m then $n < m$. Let $N = 2^k$, for some $k \in \mathbb{N}$, be the minimal number that is greater than or equal to the maximal gate number. We encode the computation of $\boldsymbol{C}$ on an input $\boldsymbol{\alpha}$ by a data instance $\mathcal{D}_{\boldsymbol{C}}$ with the following punctual facts, where $[n]$ stands for $[n,n]$:

- $V[2n + n/N]$, if n is an input gate and $\boldsymbol{\alpha}(n) = V \in \{T, F\}$;
- $D[2n + n/N]$, if n is an OR gate;
- $C[2n + n/N]$, if n is an AND gate;
- $I_0[2n + m/N], I_1[2n + k/N]$, if n is a gate with input gates m and k.

Let $\Pi_{\boldsymbol{C}}$ be a *datalogMTL* program with the rules

$$
\begin{array}{ll}
T \leftarrow \diamondminus_{[2,2]} T, & F \leftarrow \diamondminus_{[2,2]} F, \\
T \leftarrow \diamondminus_{[0,1]} (I_0 \wedge T) \wedge D, & F \leftarrow \diamondminus_{[0,1]} (I_0 \wedge F) \wedge C, \\
T \leftarrow \diamondminus_{[0,1]} (I_1 \wedge T) \wedge D, & F \leftarrow \diamondminus_{[0,1]} (I_1 \wedge F) \wedge C, \\
F \leftarrow \diamondminus_{[0,1]} (I_0 \wedge F) \wedge \diamondminus_{[0,1]} (I_1 \wedge F) \wedge D, & T \leftarrow \diamondminus_{[0,1]} (I_0 \wedge T) \wedge \diamondminus_{[0,1]} (I_1 \wedge T) \wedge C.
\end{array}
$$

Suppose n is the output gate. Then it is straightforward to check that the value of $\mathbf{C}$ on $\boldsymbol{\alpha}$ is T iff $(\Pi, \mathcal{D}) \models T[2n + n/N]$. This immediately implies the required hardness for the query answering problem. An example of a circuit $\boldsymbol{C}$ with an assignment $\boldsymbol{\alpha}$, and an initial part of the canonical model of $(\Pi_{\boldsymbol{C}}, \mathcal{D}_{\boldsymbol{C}})$ are shown below, with the black symbols above the timestamps indicating what is given in $\mathcal{D}_{\boldsymbol{C}}$ and the grey ones what is implied by $\Pi_{\boldsymbol{C}}$:



To show P-hardness of the consistency problem, it suffices to add the fact $P[2n + n/N]$ to $\mathcal{D}_{\boldsymbol{C}}$, for a fresh P, and the axiom $\bot \leftarrow P \wedge T$ to $\Pi_{\boldsymbol{C}}$. $\square$

The exact data complexity of answering propositional *datalogMTL* queries remains open. It is worth noting that answering ontology-mediated queries with propositional *LTL* ontologies is NC^1-complete for data complexity (Artale et al., 2015), while answering propositional datalog queries with the Halpern-Shoham operators is P-complete for data complexity (Kontchakov, Pandolfo, Pulina, Ryzhikov, & Zakharyaschev, 2016).

The diamond operators $\diamondplus_{\varrho}$ and $\diamondminus_{\varrho}$ are disallowed in the head of *datalogMTL* rules. Denote by *datalogMTL*$^{\diamond}$ the extension of *datalogMTL* that allows both box and diamond operators in the head of rules. We show now that this language has much more expressive power and can encode 2-counter Minsky machines, which gives the following theorem; cf. (Madnani, Krishna, & Pandya, 2013):

Theorem 10. *Consistency checking for propositional* datalogMTL$^{\diamond}$ *programs is undecidable.*

Proof. We use some ideas of (Madnani et al., 2013), where a non-Horn fragment of *MTL* was shown to be undecidable. The proof is by reduction of the undecidable non-halting problem for

Minksy machines: given a 2-counter Minsky machine, decide whether it *does not halt* starting from 0 in both counters.

Suppose we are given a Minsky machine with counters C_1 and C_2 that has $n-1$ instructions of the form

$$
\begin{aligned}
&i\text{: Increment}(C_k)\text{, goto } j,\\
&i\text{: Decrement}(C_k)\text{, goto } j,\\
&i\text{: If } C_k = 0 \text{ then } j_1 \text{ else } j_2,
\end{aligned}
$$

where i, j, j_1 and j_2 are instruction indexes, $k = 1, 2$, and the n-th instruction is

$$n\text{: Halt.}$$

We encode successive configurations of the machine using the sequence $[0,4), [4,8), [8,12), \dots$ of time intervals. The current instruction index is represented by a propositional variable P_i, for $1 \leq i \leq n$, that holds at the first point, say $4m$, of the interval $[4m, 4m+4)$. The current value, say k_1, of the counter C_1 is encoded by exactly k_1 moments of time in the interval $(4m+1, 4m+2)$ where the propositional variable C holds true. Similarly, the value k_2 of C_2 is encoded by exactly k_2 moments in the interval $(4m+3, 4m+4)$ where the propositional variable C holds true.

The initial configuration is encoded by the following data instance $\mathcal{D}$, where the variable Z indicates that both counters are 0:

$$P_1@[0,0], \quad Z@(1,2), \quad Z@(3,4). \tag{6}$$

For every i $(1 \leq i \leq n)$ we require the rules

$$\boxplus_{[0,1]} Z \leftarrow P_i, \quad \boxplus_{[2,3]} Z \leftarrow P_i, \quad \bot \leftarrow Z \wedge C, \quad \bot \leftarrow Z \wedge N \tag{7}$$

saying, in particular, that C cannot hold true outside the intended intervals (here N is an auxiliary variable). To simplify notation, we use the following abbreviations: $\textcircled{1} = \boxplus_{[1,1]}$, $\textcircled{2} = \boxplus_{[3,3]}$, and $\bigcirc = \boxplus_{[4,4]}$. The machine instructions are encoded as follows (the instructions for C_2 are obtained by replacing $\textcircled{1}$ with $\textcircled{2}$):

$$
\begin{array}{ll}
\bigcirc P_{j_1} \leftarrow P_i \wedge \textcircled{1} \boxplus_{(0,1)} Z, & \\
\bigcirc P_{j_2} \leftarrow P_i \wedge \textcircled{1} \diamondplus_{(0,1)} C, & i\text{: if } C_1 = 0 \text{ then } j_1\\
\textcircled{1} \boxplus_{(0,1)} \mathrm{CP} \leftarrow P_i, \quad \textcircled{2} \boxplus_{(0,1)} \mathrm{CP} \leftarrow P_i, & \qquad\qquad \text{else } j_2\\
\bigcirc P_j \leftarrow P_i, \quad \textcircled{1} \boxplus_{(0,1)} \mathrm{IC} \leftarrow P_i, \quad \textcircled{2} \boxplus_{(0,1)} \mathrm{CP} \leftarrow P_i, & i\text{: Inc}(C_1)\text{, goto } j\\
\bigcirc P_j \leftarrow P_i, \quad \textcircled{1} \boxplus_{(0,1)} \mathrm{DC} \leftarrow P_i, \quad \textcircled{2} \boxplus_{(0,1)} \mathrm{CP} \leftarrow P_i, & i\text{: Dec}(C_1)\text{, goto } j.
\end{array}
$$

Here the variable CP means copying of the counter value, DC means decrementing it by 1, and IC incrementing it by 1. To achieve this, we require the following rules:

$$
\begin{gathered}
\bigcirc C \leftarrow \mathrm{CP} \wedge C, \quad \bigcirc Z \leftarrow \mathrm{CP} \wedge Z,\\
\bigcirc C \leftarrow \mathrm{DC} \wedge C \wedge \diamondplus_{(0,1)} C,\\
\bigcirc Z \leftarrow \mathrm{DC} \wedge Z \wedge \diamondplus_{(0,1)} C, \quad \bigcirc \boxplus_{[0,1]} Z \leftarrow \mathrm{DC} \wedge C \wedge \boxplus_{(0,1)} Z,\\
\diamondplus_{(0,1)} N \leftarrow \boxplus_{(0,1)} \mathrm{IC} \wedge \boxplus_{(0,1)} Z, \quad \diamondplus_{(0,1)} N \leftarrow C \wedge \mathrm{IC} \wedge \boxplus_{(0,1)} Z, \qquad (8)\\
\bigcirc C \leftarrow \mathrm{IC} \wedge C, \quad \bigcirc C \leftarrow \mathrm{IC} \wedge N, \qquad (9)\\
\bigcirc Z \leftarrow \mathrm{IC} \wedge Z \wedge \diamondplus_{(0,1)} N, \quad \bigcirc \boxplus_{(0,1)} Z \leftarrow \mathrm{IC} \wedge N \wedge \boxplus_{(0,1)} Z, \qquad (10)
\end{gathered}
$$

We explain the intuition behind the most complex rules (8)–(10) that are used to model the increment of the counters. The rules (8) mark a new time-point with the variable N in a block located after the last C-time-point in this block (or, according the first axiom, N is placed anywhere in the block if the current value of a counter is 0). The rules (9) insert C in the next block, where in the current block we have either C or N. The rules (10) transfer Z from the current block to the next one excluding the time-point where N holds. Finally, we add the rule

$$\bot \leftarrow P_n, \qquad\qquad\qquad \text{n: Halt.}$$

It is not hard to check that the program and data instance above are consistent iff the given 2-counter Minsky machine does not halt. □

The diamond operators in the head of rules can encode disjunction and thereby ruin 'Horness'. Thus, the temporalised description logic $\mathcal{EL}$ with such rules is undecidable (Lutz, Wolter, & Zakharyaschev, 2008); cf. also (Gutiérrez-Basulto et al., 2016a). The addition of diamonds in the heads to the Horn fragment of the propositional Halpern-Shoham logic $\mathcal{HS}$ can make a P-complete logic undecidable (Bresolin et al., 2017). A distinctive feature of these formalisms is their two-dimensionality (Gabbay, Kurucz, Wolter, & Zakharyaschev, 2003), while propositional *datalogMTL* is one-dimensional. Diamonds in the head of rules also ruin FO-rewritability of answering ontology-mediated queries with temporalised *DL-Lite* ontologies by increasing their data complexity to coNP (Artale et al., 2013). The same construction actually shows that nonrecursive *datalogMTL* with binary predicates and diamonds in the heads is coNP-hard.

4. Nonrecursive *datalogMTL*

As none of the *datalogMTL* programs required in our use cases is recursive, we now consider the class *datalog$_{nr}$MTL* of nonrecursive *datalogMTL* programs. We first show that consistency checking (and so query answering) for *datalog$_{nr}$MTL* programs is PSPACE-complete for combined complexity. Then we regard a given *datalog$_{nr}$MTL* program as fixed and reduce these problems to evaluating a (data-independent) FO($<$)-formula over any given data, thereby establishing that *datalog$_{nr}$MTL* is in AC^0 for data complexity.

More precisely, for a program Π, let $\lessdot$ be the dependence relation on the predicate symbols in Π: we have $P \lessdot Q$ iff Π contains a clause with P in the head and Q in the body. Π is called *nonrecursive* if $P \lessdot^+ P$ does not hold for any predicate symbol P in Π, where $\lessdot^+$ is the transitive closure of $\lessdot$. We denote by $\mathsf{depth}_\Pi(P)$ the maximal number l such that $P_0 \lessdot P_1 \lessdot \cdots \lessdot P_l = P$. (Note that $\mathsf{depth}_\Pi(P) = 0$ iff either P does not occur in Π or P occurs only in the body of some rules.) The maximal $\mathsf{depth}_\Pi(P)$ over all predicates P is denoted by $\mathsf{depth}(\Pi)$. It should be clear that, for any nonrecursive Π and any data instance $\mathcal{D}$, there exists some $n \in \mathbb{N}$ such that $\mathsf{cl}^{n+1}(\mathcal{D}) = \mathsf{cl}^n(\mathcal{D}) = \mathfrak{C}_{\Pi,\mathcal{D}}$. Therefore, $\mathfrak{C}_{\Pi,\mathcal{D}}$ is finite.

Denote by $\min \mathcal{D}$ and $\max \mathcal{D}$ the minimal and, respectively, maximal *finite* numbers that occur in the intervals from $\mathcal{D}$. Let K be the largest number occurring in Π. We then set

$$M_l = \min \mathcal{D} - K \times \mathsf{depth}(\Pi) \quad \text{and} \quad M_r = \max \mathcal{D} + K \times \mathsf{depth}(\Pi).$$

Let $d = \gcd(\mathsf{num}(\Pi, \mathcal{D}))$. The next lemma will be required for our PSPACE algorithm checking consistency of *datalog$_{nr}$MTL* programs.

Lemma 11. *Let Π be a datalog$_{nr}$MTL program. Then every interval ι such that $P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ or $\bot(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ takes one of the following forms: $(-\infty,\infty)$, $\langle dk,\infty)$, $(-\infty,dk\rangle$, $\langle dk,dk'\rangle$, where $k,k' \in \mathbb{Z}$ and $M_l \leq dk \leq dk' \leq M_r$.*

Proof. That every interval in $\mathfrak{C}_{\Pi,\mathcal{D}}$ is of the form $(-\infty,\infty)$, $\langle dk,\infty)$, $(-\infty,dk\rangle$, $\langle dk,dk'\rangle$, where $k,k' \in \mathbb{Z}$, was observed in the proof of Lemma 6. Thus, we only need to establish the bounds on dk and dk'. For each P, let $\mathsf{hi}(P)$ and $\mathsf{lo}(P)$ be the maximal and, respectively, minimal number $dk \in \mathbb{Q}$ such that $P(\boldsymbol{c})@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ and dk is an end-point of ι. Note that $\mathsf{hi}(P)$ and $\mathsf{lo}(P)$ can be undefined. We are going to show that $\mathsf{hi}(P)$ is either undefined or $\mathsf{hi}(P) \leq \max\mathcal{D} + \mathsf{depth}_\Pi(P)K$. (That $\mathsf{lo}(P)$ is either undefined or $\mathsf{lo}(P) \geq \min\mathcal{D} - \mathsf{depth}_\Pi(P)K$ is left to the reader.) Clearly, this fact implies the required bounds on dk and dk'.

The proof is by induction on the construction of $\mathfrak{C}_{\Pi,\mathcal{D}}$. Let $\mathsf{hi}^n(P)$ be the maximal $dk \in \mathbb{Q}_2$ such that $P(\boldsymbol{c})@\iota \in \mathsf{cl}^n(\mathcal{D})$ and dk is an end-point of ι. We show by induction on n that either $\mathsf{hi}^n(P)$ is undefined or $\mathsf{hi}^n(P) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P)$.

For the basis of induction, if $\mathsf{hi}^0(P)$ is defined and $P(\boldsymbol{c})@\iota \in \mathsf{cl}^0(\mathcal{D})$ is an atom mentioning $\mathsf{hi}^0(P)$, then $P(\boldsymbol{c})@\iota \in \mathcal{D}$ and $\mathsf{hi}^0(P) \leq \max\mathcal{D}$. Assume next that $n = n'+1$. Suppose $\mathsf{hi}^n(P)$ is defined and let $P(\boldsymbol{c})@\iota \in \mathsf{cl}^n(\mathcal{D})$ be an atom mentioning $\mathsf{hi}^n(P)$. If $P(\boldsymbol{c})@\iota \in \mathsf{cl}^{n'}(\mathcal{D})$, we are done by the induction hypothesis. Otherwise, we consider how $P(\boldsymbol{c})@\iota$ was obtained. Suppose it was obtained by (coal) with $\iota = \bigcup_{i\in I}\iota_i$. By the induction hypothesis, $\mathsf{hi}^{n'}(P) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P)$, and so every number mentioned in $\{\iota_i \mid i \in I\}$ does not exceed $\max\mathcal{D} + K\mathsf{depth}_\Pi(P)$. Thus, we have $\mathsf{hi}^n(P) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P)$. Now suppose that $P(\boldsymbol{c})@\iota$ was obtained by (horn) from $P_i(\boldsymbol{c}_i)@\iota_i$, $i \in I$. Observe that $\mathsf{depth}_\Pi(P_i) < \mathsf{depth}_\Pi(P)$ and, by the induction hypothesis, $\mathsf{hi}^{n'}(P_i) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P_i)$. Since $\iota = \bigcap_{i\in I}\iota_i$, the maximal number mentioned in ι cannot exceed $\max\mathcal{D} + K\mathsf{depth}_\Pi(P)$. Thus, $\mathsf{hi}^n(P) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P)$. Consider now the case when $P(\boldsymbol{c})@\iota$ was obtained by applying $(\mathcal{S}_\varrho)$ to $P_i(\boldsymbol{c}_i)@\iota_i$, $i \in \{1,2\}$. By the induction hypothesis, the largest number mentioned in ι_i does not exceed $\max\mathcal{D} + K\mathsf{depth}_\Pi(P_i)$. On the other hand, $\mathsf{depth}_\Pi(P_i) < \mathsf{depth}_\Pi(P)$ and the maximal number in ι cannot be larger that the maximal number in $\{\iota_i \mid i \in \{1,2\}\}$ plus K. Thus, the maximal number in ι does not exceed

$$\max\mathcal{D} + K\mathsf{depth}_\Pi(P_i) + K \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P),$$

and so $\mathsf{hi}^n(P) \leq \max\mathcal{D} + K\mathsf{depth}_\Pi(P)$. The remaining temporal rules are similar and left to the reader. ☐

Suppose we are given a *datalog$_{nr}$MTL* program Π and a data instance $\mathcal{D}$. If Π and $\mathcal{D}$ are inconsistent then, by Lemmas 5 and 11, we have $\bot@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$, for some ι of the form $(-\infty,\infty)$, $\langle dk,\infty)$, $(-\infty,dk\rangle$, $\langle dk,dk'\rangle$, where $k,k' \in \mathbb{Z}$ and $M_l \leq dk \leq dk' \leq M_r$. Thus, there is a *derivation* of $\bot@\iota$ from Π and $\mathcal{D}$, that is, a tree whose root is $\bot@\iota$, whose leaves are some atoms from $\mathcal{D}$, and whose every non-leaf vertex results from applying one of the rules (coal), (horn), $(\mathcal{S}_\varrho)$, $(\boxplus_\varrho)$, $(\mathcal{U}_\varrho)$, $(\boxminus_\varrho)$ to the immediate predecessors of this vertex.

Lemma 12. *If $\bot@\iota \in \mathfrak{C}_{\Pi,\mathcal{D}}$ then there is a derivation of $\bot@\iota$ from Π and $\mathcal{D}$ such that*

(i) *the length of any branch in the derivation does not exceed $2|\Pi|$;*

(ii) *for some polynomial p, every non-leaf vertex, corresponding to the application of (coal) in the derivation, has at most $2^{p(|\Pi|,|\mathcal{D}|)}$ immediate predecessors.*

Proof. To show (i), it suffices to recall that Π is non-recursive (and so none of the rules in Π can be applied twice in the same branch of the derivation) and observe that we can always replace multiple successive applications of the rule (coal) with a single application.

(ii) follows from Lemma 11. $\square$

Theorem 13. *Consistency checking for* datalog$_{nr}$*MTL programs is* PSPACE*-complete for combined complexity. The lower bound holds even for propositional* datalog$_{nr}$*MTL.*

Proof. The upper bound is established by a standard algorithm (Ladner, 1977; Tobies, 2001) using Lemma 12 and Savitch's theorem according to which NPSPACE = PSPACE. In essence, the NPSPACE algorithm guesses branches of the derivation one by one and keeps only last two branches in memory. By Lemma 12 (i), each branch contains $\leq 2|\Pi|$ atoms of the form $P(\boldsymbol{c})@\iota$, where ι is as in Lemma 11, and so is stored in polynomial space. In addition, we store the axioms in Π that created these atoms, or (coal) if the atom was obtained by coalescing. In the latter case, we also need to guess a number k indicating how many distinct intervals are coalesced to obtain ι. By Lemma 12 (ii), $k \leq 2^{p(|\Pi|,|\mathcal{D}|)}$, and so it can be stored in polynomial space.

The lower bound is proved by reduction of the satisfiability problem for quantified Boolean formulas (QBFs), which is known to be PSPACE-complete. Let $\varphi = Q_n p_n \dots Q_0 p_0 \varphi_0$ be a QBF, where each Q_i is either $\forall$ or $\exists$, and $\varphi_0 = c_0 \land \dots \land c_m$ is a propositional formula in CNF with $c_i = l_0 \lor \dots \lor l_k$, with each l_i being either a variable p_j or its negation $\neg p_j$, for $0 \leq j \leq n$. In our *datalog$_{nr}$MTL* program, we use the following propositional variables:

- $P_0, \dots, P_n$ (to represent $p_0, \dots, p_n$ from φ);
- $\bar{P}_0, \dots, \bar{P}_n$ (to represent $\neg p_0, \dots, \neg p_n$);
- $P_0^0, \dots, P_0^n$ for p_0; $P_1^1, \dots, P_1^n$ for p_1, etc.; P_n^n for p_n, and similarly for $\neg p_i$;
- $F_0, \dots, F_{n+1}$;
- $C_0, \dots, C_m$ (to represent $c_0, \dots, c_m$).

We first take a data instance $\mathcal{D}$ with the following facts:

$$P_i^i@[0, 2^i),\ \bar{P}_i^i@[2^i, 2^{i+1}), \quad \text{for } 0 \leq i \leq n.$$

Starting from this data, we can generate all the truth-assignments for the variables $p_0, \dots, p_n$ using the following rules, where $0 \leq i \leq n$:

$$P_i \leftarrow P_i^n, \quad \bar{P}_i \leftarrow \bar{P}_i^n,$$
$$P_i^{j+1} \leftarrow P_i^j, \quad \boxplus_{2^{j+1}} P_i^{j+1} \leftarrow P_i^j, \quad \bar{P}_i^{j+1} \leftarrow \bar{P}_i^j, \quad \boxplus_{2^{j+1}} \bar{P}_i^{j+1} \leftarrow \bar{P}_i^j, \quad i \leq j < n.$$

The canonical model for $\mathcal{D}$ and the rules above for the variables p_0, p_1, p_2 (thus, $n = 2$) is shown in Fig. 1.

We then need the rules:

$$C_i \leftarrow P_j, \quad p_j \text{ occurs in } c_i, \tag{11}$$
$$C_i \leftarrow \bar{P}_j, \quad \neg p_j \text{ occurs in } c_i, \tag{12}$$
$$F_0 \leftarrow \bigwedge_{0 \leq i \leq m} C_i, \tag{13}$$

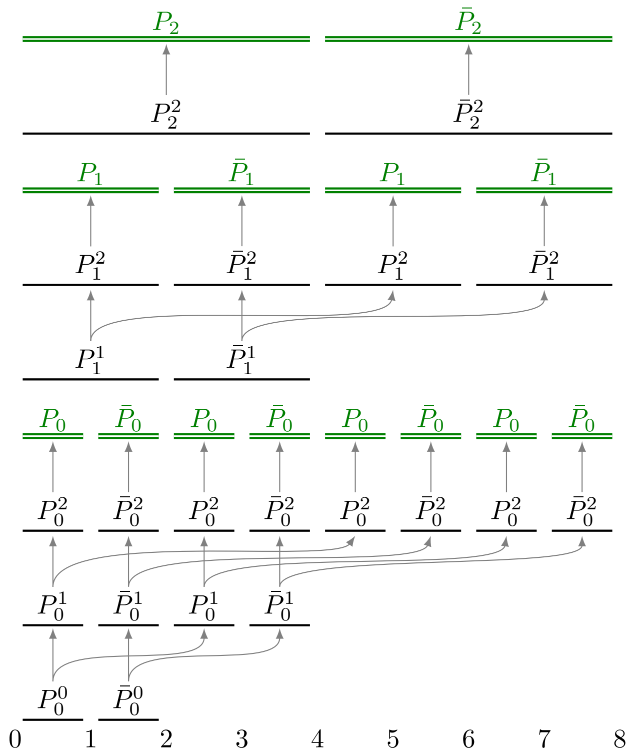


Figure 1: The canonical model for the proof of Theorem 13.

for $0 \leq i \leq m$, $0 \leq j \leq n$. Note that F_0 will hold at the moments of time corresponding to the assignments that make φ_0 true. Further, we consider the formula $\varphi_i = Q_{i-1}p_{i-1}\dots Q_0 p_0 \varphi_0$, for $1 \leq i \leq n+1$ (note that $\varphi_{n+1} = \varphi$), and provide rules that make F_i true precisely at the moments of time corresponding to the assignments that make φ_i true. We take

$$\boxplus_{[0,2^i]} F_{i+1} \leftarrow F_i \wedge P_i, \quad \boxminus_{[0,2^i]} F_{i+1} \leftarrow F_i \wedge \bar{P}_i, \quad \text{if } Q_i = \exists, \tag{14}$$

$$\boxplus_{[0,2^{i+1})} F_{i+1} \leftarrow \boxplus_{[0,2^i)} P_i \wedge \boxplus_{[0,2^{i+1})} F_i, \quad \text{if } Q_i = \forall, \tag{15}$$

for $0 \leq i \leq n$, and, finally,

$$\bot \leftarrow \boxplus_{[0,2^{n+1})} F_{n+1}.$$

All the rules above form the required *datalog$_{nr}$MTL* program Π. We now prove that Π is consistent with $\mathcal{D}$ iff φ is not satisfiable. By Lemma 5, it suffices to show that $F_{n+1}@[0, 2^{n+1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ iff φ is satisfiable. For $(\Rightarrow)$, suppose $F_{n+1}@[0, 2^{n+1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$. If $Q_n = \exists$ then, in view of (14), either $F_n@[0, 2^n), P_n@[0, 2^n) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ or $F_n@[2^n, 2^{n+1}), \bar{P}_n@[2^n, 2^{n+1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$. If the first option holds, we show that φ_n is satisfiable when p_n is true; if the second option holds, we show that φ_n is satisfiable when p_n is false. Similarly, if $Q_n = \forall$, then by (15), we have $F_n@[0, 2^n), P_n@[0, 2^n) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ and $F_n@[2^n, 2^{n+1}), \bar{P}_n@[2^n, 2^{n+1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$. In this case, we show that φ_n is satisfiable when p_n can be both false and true. To show that $F_n@[0, 2^n), P_n@[0, 2^n) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies that φ_n is satisfiable when p_n is true (the other case is analogous and left to the reader), suppose $Q_{n-1} = \exists$. By (14), either $F_{n-1}@[0, 2^{n-1}), P_{n-1}@[0, 2^{n-1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ or $F_n@[2^{n-1}, 2^n), \bar{P}_{n-1}@[2^{n-1}, 2^n) \in \mathfrak{C}_{\Pi,\mathcal{D}}$. (If

$Q_{n-1} = \forall$, by (14) both of these options hold.) Therefore, to show that φ is satisfiable, it now suffices to show that (*i*) $F_{n-1}@[0, 2^{n-1}), P_{n-1}@[0, 2^{n-1}) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies that φ_{n-1} is satisfiable when p_n is true and p_{n-1} is true; (*ii*) $F_{n-1}@[2^{n-1}, 2^n), \bar{P}_{n-1}@[2^{n-1}, 2^n) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies that φ_{n-1} is satisfiable when p_n is true and p_{n-1} is false. We only consider (*i*), leaving (*ii*) to the reader, and after applying the argument above n times, will need to show that (*i*) $F_0@[0, 1), P_0@[0, 1) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies that φ_0 is satisfiable when $p_n, \dots, p_1$ and p_0 are all true; (*ii*) $F_0@[1, 2), \bar{P}_0@[1, 2) \in \mathfrak{C}_{\Pi,\mathcal{D}}$ implies that φ_0 is satisfiable when $p_n, \dots, p_1$ are true while p_0 is false. That (*i*) holds follows from (11)–(13), and similarly for (*ii*). This concludes the proof of ($\Rightarrow$); the other direction is proved analogously. □

Using the techniques of (Artale, Kontchakov, Ryzhikov, & Zakharyaschev, 2014), it can be shown that nonrecursive Horn fragment of *LTL* is P-complete. The same complexity can be derived from (Bresolin et al., 2017) for the nonrecursive Horn fragment of the Halpern-Shoham logic $\mathcal{HS}$.

As we have just seen, the combined complexity of query answering drops from EXPSPACE for *datalogMTL* to PSPACE for *datalog$_{nr}$MTL*. We now show that the data complexity drops to AC^0, which is important for practical query answering using standard database systems. Note that this result is non-trivial in view of Theorem 9. The crux of the proof is encoding coalescing by FO-formulas with $\forall$ (which is typically not needed for rewriting atemporal ontology-mediated queries).

Theorem 14. *Consistency checking and answering datalog$_{nr}$MTL queries is in AC^0 for data complexity.*

Proof. We only consider a *propositional datalog$_{nr}$MTL* program Π. The proof can be straightforwardly adapted to the case of arity ≥ 1 by adding more (object) variables to the predicates used below. Let N be a set of comprising numbers or $\infty, -\infty$. We use $N + r$ as a shorthand for $\{t + r \mid t \in N\}$ and similarly for $N - r$ (we assume that $t + \infty = \infty$ and $t - \infty = -\infty$). For a propositional variable P in Π, we define two sets $\mathsf{le}(P)$ and $\mathsf{ri}(P)$ as follows:

– $\mathsf{le}(P) = \mathsf{ri}(P) = \{0\}$ if there is no P' such that $P \lessdot P'$;

– otherwise, $\mathsf{le}(P)$ is the union of:

  – $\bigcup_{i \in I} \mathsf{le}(P_i)$, for each $P \leftarrow \bigwedge_{i \in I} P_i$ in Π,
  – $\mathsf{le}(P_2) + r_1 \cup \mathsf{ri}(P_1)$, for each $P \leftarrow P_1 \,\mathcal{S}_{\langle r_1, r_2 \rangle}\, P_2$ in Π,
  – $\mathsf{le}(P_2) - r_2 \cup \mathsf{le}(P_1)$, for each $P \leftarrow P_1 \,\mathcal{U}_{\langle r_1, r_2 \rangle}\, P_2$ in Π,
  – $\mathsf{le}(P_1) + r_2$, for each $P \leftarrow \boxminus_{\langle r_1, r_2 \rangle} P_1$ in Π,
  – $\mathsf{le}(P_1) - r_1$, for each $P \leftarrow \boxplus_{\langle r_1, r_2 \rangle} P_1$ in Π,

  and $\mathsf{ri}(P)$ is the union of:

  – $\bigcup_{i \in I} \mathsf{ri}(P_i)$, for each $P(\tau) \leftarrow \bigwedge_{i \in I} P_i$ in Π,
  – $\mathsf{ri}(P_2) + r_2 \cup \mathsf{ri}(P_1)$, for each $P \leftarrow P_1 \,\mathcal{S}_{\langle r_1, r_2 \rangle}\, P_2$ in Π,
  – $\mathsf{ri}(P_2) - r_1 \cup \mathsf{le}(P_1)$, for each $P \leftarrow P_1 \,\mathcal{U}_{\langle r_1, r_2 \rangle}\, P_2$ in Π,
  – $\mathsf{ri}(P_1) + r_1$, for each $P \leftarrow \boxminus_{\langle r_1, r_2 \rangle} P_1$ in Π,
  – $\mathsf{ri}(P_1) - r_2$, for each $P \leftarrow \boxplus_{\langle r_1, r_2 \rangle} P_1$ in Π.

Using an argument that is similar to the proof of Lemma 11, one can show the following:

Lemma 15. *For any datalog$_{nr}$MTL program Π, any data instance $\mathcal{D}$, and any $P@\langle t_1, t_2\rangle \in \mathfrak{C}_{\Pi,\mathcal{D}}$,*

- $t_1 = t_1' + n_1$, *for some* $n_1 \in \mathsf{le}(P)$ *and some* t_1' *such that* $P'[t_1', t_1'] \in \mathcal{D}$ *or* $P'(t_1', s_2) \in \mathcal{D}$,
- $t_2 = t_2' + n_2$, *for some* $n_2 \in \mathsf{ri}(P)$ *and some* t_2' *such that* $P'[t_2', t_2'] \in \mathcal{D}$ *or* $P'(s_1, t_2') \in \mathcal{D}$.

In view of Lemma 15, we can prove Theorem 14 by constructing FO-formulas $\varphi_P^{\langle m,n\rangle}(x, y)$ with $m \in \mathsf{le}(P)$ and $n \in \mathsf{ri}(P)$ such that, for any data instance $\mathcal{D}$,

$$P@\langle t_1 + m, t_2 + n\rangle \in \mathfrak{C}_{\Pi,\mathcal{D}} \qquad \text{iff} \qquad \mathfrak{A}_{\mathcal{D}} \models \varphi_P^{\langle m,n\rangle}(t_1, t_2), \tag{16}$$

where $\mathfrak{A}_{\mathcal{D}}$ is the FO-structure defined below. To slightly simplify presentation (and without much loss of generality), we assume that all numbers in $\mathsf{num}(\mathcal{D})$ are positive, and set

$$\mathfrak{A}_{\mathcal{D}} = \big(\Delta, <, P_1^{[]}, P_1^{()}, \dots, P_l^{[]}, P_l^{()}, \mathsf{bit}^{in}, \mathsf{bit}^{fr}\big),$$

where

- Δ is a set of $(\ell + 1)$-many elements strictly linearly ordered by $<$, ℓ is the maximum of the number of distinct timestamps in $\mathcal{D}$ and the number of bits in the longest binary fraction in $\mathcal{D}$ (excluding the binary point); for simplicity, we assume that $\Delta = \{0, \dots, \ell\}$, $<$ is the natural order, and denote by $\bar{n}$ the nth fraction in $(\mathsf{num}(\mathcal{D}), <)$, counting from 0;
- $P_i^{[]}(n, n)$ holds in $\mathfrak{A}_{\mathcal{D}}$ iff $P_i@[\bar{n}, \bar{n}] \in \mathcal{D}$ and $P_i^{()}(n, m)$ holds in $\mathfrak{A}_{\mathcal{D}}$ iff $P_i@(\bar{n}, \bar{m}) \in \mathcal{D}$, for any P_i occurring in $\mathcal{D}$;
- for $\bar{n} \neq \infty$, $\mathsf{bit}^{in}(n, i, 0)$ ($\mathsf{bit}^{fr}(n, i, 0)$) holds in $\mathfrak{A}_{\mathcal{D}}$ iff the ith bit of the integer (respectively, fractional) part of $\bar{n}$ is 0, and $\mathsf{bit}^{in}(n, i, 1)$ ($\mathsf{bit}^{fr}(n, i, 1)$), for $i \in \Delta$, holds in $\mathfrak{A}_{\mathcal{D}}$ iff the ith bit of the integer (respectively, fractional) part of $\bar{n}$ is 1 (as usual, we start counting bits from the least significant one);
- for $\bar{n} = \infty$, $\mathsf{bit}^{in}(n, i, 1)$ and $\mathsf{bit}^{fr}(n, i, 1)$ for all $i \in \Delta$.

For example, the data instance $\mathcal{D} = \{P[110.001, 110.001], P(10000, \infty)\}$ is given as the FO structure

$$\mathfrak{A}_{\mathcal{D}} = \big(\Delta, <, P^{[]}, P^{()}, \mathsf{bit}^{in}, \mathsf{bit}^{fr}\big),$$

where $\Delta = \{0, \dots, 6\}$, $P^{[]} = \{(0, 0)\}$, $P^{()} = \{(1, 2)\}$, and

$$\begin{aligned}
\mathsf{bit}^{in} =& \{(0,0,0), (0,1,1), (0,2,1)\} \cup \{(0,i,0) \mid 3 \le i \le 6\} \cup \\
& \{(1,i,0) \mid 0 \le i \le 3\} \cup \{(1,4,1)\} \cup \{(1,5,0)\} \cup \{(1,6,0)\} \cup \\
& \{(2,i,1) \mid 0 \le i \le 6\}. \\
\mathsf{bit}^{fr} =& \{(0,4,1)\} \cup \{(0,i,0) \mid 0 \le i \le 6,\ i \neq 4\} \cup \\
& \{(1,i,0) \mid 0 \le i \le 6\} \cup \\
& \{(2,i,1) \mid 0 \le i \le 6\}.
\end{aligned}$$

To construct the required $\varphi_P^{\langle m,n\rangle}(x, y)$, suppose that we have FO-formulas

– $\mathsf{coal}_P^{\langle m,n\rangle}(x,y)$ saying that $P@\langle x+m, y+n\rangle$ is added to $\mathfrak{C}_{\Pi,\mathcal{D}}$ by an application of the rule (coal);

– $\psi_P^{\langle m,n\rangle}(x,y)$ saying that

either $P@\langle x+m, y+n\rangle$ is added to $\mathfrak{C}_{\Pi,\mathcal{D}}$ because it belongs to the given data instance (in which case we can assume that $m=n=0$, and $\langle\rangle$ is either $()$ or $[]$),

or $P@\langle x+m, y+n\rangle$ is added to $\mathfrak{C}_{\Pi,\mathcal{D}}$ as a result of an application of one of the 'logical' rules.

In this case we can set

$$\varphi_P^{\langle m,n\rangle}(x,y) \;=\; \psi_P^{\langle m,n\rangle}(x,y) \vee \mathsf{coal}_P^{\langle m,n\rangle}(x,y).$$

Using the predicate $\mathsf{is}_{a,b}$, which is $\top$ if $a=b$ and $\bot$ otherwise, we can define $\psi_P^{\langle m,n\rangle}(x,y)$ as a disjunction of the following formulas:

– $\mathsf{is}_{\langle,[} \wedge \mathsf{is}_{\rangle,]} \wedge \mathsf{is}_{m,0} \wedge \mathsf{is}_{n,0} \wedge P^{[]}(x,y)$;

– $\mathsf{is}_{\langle,(} \wedge \mathsf{is}_{\rangle,)} \wedge \mathsf{is}_{m,0} \wedge \mathsf{is}_{n,0} \wedge P^{()}(x,y)$;

– for every $P \leftarrow \bigwedge\limits_{1\leq i\leq k} P_i$ in Π,

$$\exists x_1, y_1, \dots, x_k, y_k \bigvee_{\substack{m_1\in\mathsf{le}(P_1)\\ n_1\in\mathsf{ri}(P_1)\\ \lceil_1\in\{[,(\},\ \rceil_1\in\{],)\}}} \Big(\varphi_{P_1}^{\lceil_1 m_1,n_1\rceil_1}(x_1,y_1) \wedge \cdots \wedge \bigvee_{\substack{m_k\in\mathsf{le}(P_k)\\ n_k\in\mathsf{ri}(P_k)\\ \lceil_k\in\{[,(\},\ \rceil_k\in\{],)\}}} \big(\varphi_{P_k}^{\lceil_k m_k,n_k\rceil_k}(x_k,y_k) \wedge \\ \mathsf{inter}^{\langle m,n\rangle}_{\lceil_1 m_1,n_1\rceil_1,\dots,\lceil_k m_k,n_k\rceil_k}(x,y,x_1,y_1,\dots,x_k,y_k)\big)\dots\Big),$$

where $\mathsf{inter}^{\langle m,n\rangle}_{\lceil_1 m_1,n_1\rceil_1,\dots,\lceil_k m_k,n_k\rceil_k}(x,y,x_1,y_1,\dots,x_k,y_k)$ says that $\langle x+m, y+n\rangle$ is an intersection of $\lceil_1 x_1+m_1, y_1+n_1\rceil_1, \dots, \lceil_k x_k+m_k, y_k+n_k\rceil_k$ (this formula can easily be defined in terms of the predicates $x+m=y+n$ and $x+m<y+n$ given below);

– for every $P \leftarrow P_1\ \mathcal{S}_\varrho\ P_2$ in Π, the formula $\sigma^{\langle m,n\rangle}_{\varrho,P,P_1,P_2}(x,y)$ saying that $\langle x+m, y+n\rangle$ is $((\iota_1^c \cap \iota_2) -^o \varrho) \cap \iota_1^c$ for some ι_1 and ι_2, where P_1 and P_2 hold, respectively (we give a definition of $\sigma^{\langle m,n\rangle}_{\varrho,P,P_1,P_2}(x,y)$ in the Appendix);

– analogous formulas encoding the relevant operations on intervals for the other temporal operators.

The formula $\mathsf{coal}_P^{\langle m,n\rangle}(x,y)$ is defined as follows:

$$\mathsf{coal}_P^{\langle m,n\rangle}(x,y) \;=\; \forall z \bigwedge_{l\in\mathsf{le}(P)\cup\mathsf{ri}(P)} \big((x+m \leq z+l)\wedge(z+l\leq y+m) \to \mathsf{nogap}^l_{P,\langle m,n\rangle}(z,x,y)\big), \quad (17)$$

where $\mathsf{nogap}^{l}_{P,\langle m,n\rangle}(z,x,y)$ is the formula

$$\exists x_1,y_1,x_2,y_2,x_3,y_3 \bigvee_{\substack{m_1\in\mathsf{le}(P)\\ n_1\in\mathsf{ri}(P)\\ \lceil_1\in\{[,(\},\ \rceil_1\in\{],)\}}} \Big(\psi_P^{\lceil_1 m_1,n_1\rceil_1}(x_1,y_1)\wedge \mathsf{sub}^{\lceil_1 m_1,n_1\rceil_1}_{\langle m,n\rangle}(x_1,y_1,x,y)\wedge$$

$$\bigvee_{\substack{m_2\in\mathsf{le}(P)\\ n_2\in\mathsf{ri}(P)\\ \lceil_2\in\{[,(\},\ \rceil_2\in\{],)\}}} \Big(\psi_P^{\lceil_2 m_2,n_2\rceil_2}(x_2,y_2)\wedge \mathsf{sub}^{\lceil_2 m_2,n_2\rceil_2}_{\langle m,n\rangle}(x_2,y_2,x,y)\wedge$$

$$\Big((x_1+m_1<z+l<y_1+n_1)\vee \tag{18}$$

$$(x_1+m_1<y_1+l_1=z+l=x_2+m_2<y_2+n_2)\wedge \mathsf{is}_{\rceil_1,]}\vee \mathsf{is}_{\lceil_2,[})\vee \tag{19}$$

$$\bigvee_{\substack{m_3\in\mathsf{le}(P)\\ n_3\in\mathsf{ri}(P)}} \big(\psi_P^{[m_3,n_3]}(x_3,y_3)\wedge \mathsf{sub}^{[m_3,n_3]}_{\langle m,n\rangle}(x_3,y_3,x,y)\wedge$$

$$\big[(x_3+m_3=y_3+n_3=z+l=x+m=x_1+m_1<y_1+n_1)\vee \tag{20}$$

$$(x_1+m_1<y_1+n_1=z+l=y+n=x_3+m_3=y_3+n_3)\vee \tag{21}$$

$$(x_1+m_1<y_1+l_1=z+l=x_3+m_3=y_3+n_3=x_2+m_2<y_2+n_2)\big]\big)\Big)\Big) \tag{22}$$

and $\mathsf{sub}^{\lceil m',n'\rceil}_{\langle m,n\rangle}(x',y',x,y)$ says that $\lceil x'+m',y'+n'\rceil$ is a subinterval of $\langle x+m,y+n\rangle$. Intuitively, $\mathsf{nogap}^{l}_{P,\langle m,n\rangle}(z,x,y)$ says that around the time instant $z+l$ (that is, to the left and right of it as well as at $z+l$ itself), their is no subinterval of $\langle x+m,y+n\rangle$ that is not covered by P. The five cases considered in the formula $\mathsf{nogap}^{l}_{P,\langle m,n\rangle}(z,x,y)$ are illustrated in Fig. 2.

When evaluating $\varphi^{\langle m,n\rangle}(x,y)$ over $\mathfrak{A}_\mathcal{D}$, we need to compute the truth-values of $x+m=y+n$ and $x+m<y+n$ (for fixed m and n). We regard the former as a formula with the predicates bit^{in}, bit^{fr} and $<$ that is true just in case $x=y+(n-m)$ if $n\geq m$, and $y=x+(m-n)$ otherwise. We provide a definition of $x=y+c$, for a positive c, in the Appendix. A formula expressing $x+m<y+n$ is constructed similarly and left to the reader.

Finally, we show how the formulas $\varphi_P^{\langle m,n\rangle}(x,y)$ defined above can be used to check whether an interval $\iota=\langle\iota_b,\iota_e\rangle$ is a certain answer to $(\Pi,P@x)$ over $\mathcal{D}$. As follows from Lemma 15, if $\bot@\lceil t_1,t_2\rceil\in\mathfrak{C}_{\Pi,\mathcal{D}}$ then, for some $m\in\mathsf{le}(\bot)$, $n\in\mathsf{ri}(\bot)$ and some numbers $t_1',t_2'\in\mathsf{num}(\mathcal{D})$ such that t_1' (t_2') occurs as the left (right) end of some interval, we have $t_1=t_1'+m$ and $t_2=t_2'+n$. Take the structure $\mathfrak{A}^\iota_\mathcal{D}$ that extends $\mathfrak{A}_\mathcal{D}$ with the numbers ι_b and ι_e. By (16), ι is a certain answer to $(\Pi,P@x)$ over $\mathcal{D}$ iff the formula

$$\exists x,y \bigvee_{\substack{m\in\mathsf{le}(\bot)\\ n\in\mathsf{ri}(\bot)}} \varphi_\bot^{\lceil m,n\rceil}(x,y)\vee$$

$$\exists x,y,x_1,y_1 \bigvee_{\substack{m_1\in\mathsf{le}(\bot)\\ n_1\in\mathsf{ri}(\bot)\\ \lceil_1\in\{[,(\},\ \rceil_1\in\{],)\}}} \big(\varphi_P^{\lceil_1 m,n\rceil_1}(x_1,y_1)\wedge \mathsf{sub}^{\langle 0,0\rangle}_{\lceil_1 m,n\rceil_1}(x,y,x_1,y_1)\wedge(x=\iota_b)\wedge(y=\iota_e)\big) \tag{23}$$

holds true in $\mathfrak{A}^\iota_\mathcal{D}$. $\square$

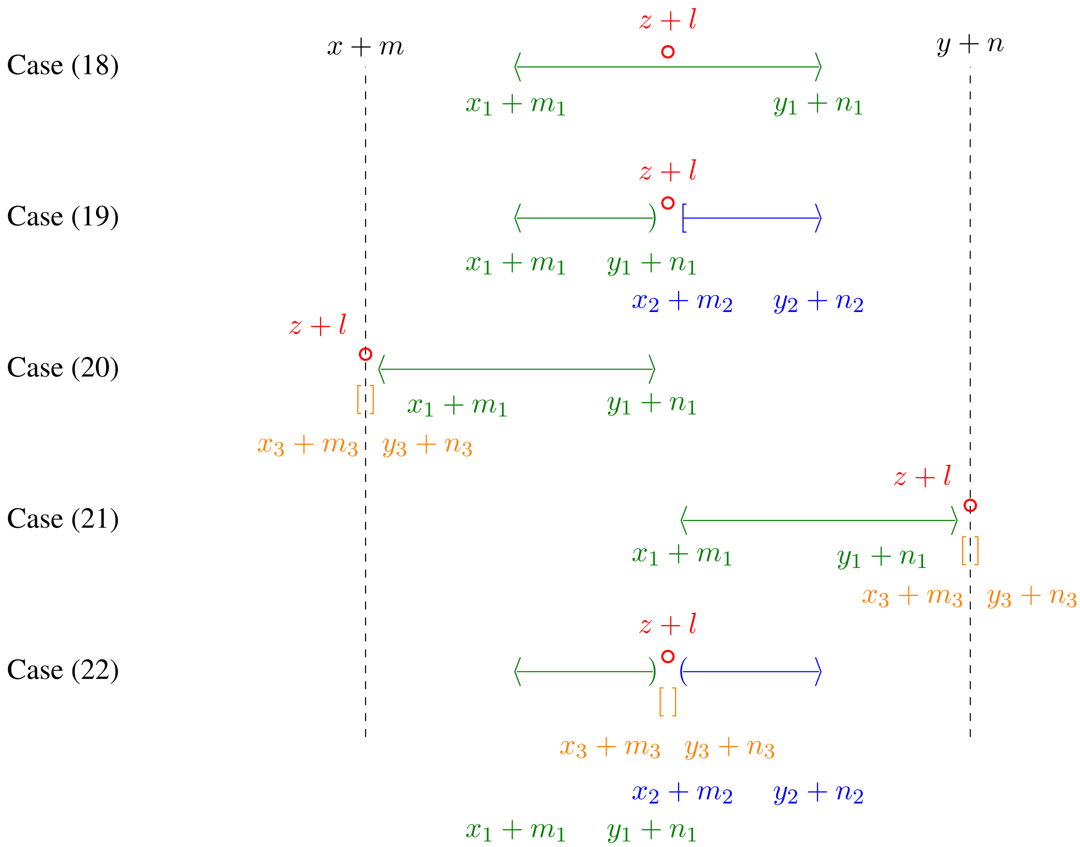


Figure 2: Five cases of the formula $\mathsf{nogap}^l_{P,\langle m,n\rangle}(z,x,y)$.

5. Implementing *datalog*$_{nr}$*MTL*

Unfortunately, the (data independent) FO-rewriting (23) turns out to be impractical because of the universal quantifier used for coalescing in (17). It is well known that $\forall$ is implemented in SQL as $\neg\exists\neg$ resulting in suboptimal performance in general. Having experimented with a few different approaches, we decided to use a materialisation (bottom-up) technique. In this section, we first present a bottom-up algorithm whose worst-case running time is linear in the number of intervals of an input data instance $\mathcal{D}$, under a practically motivated assumption that the order of occurrence of the intervals in $\mathcal{D}$ coincides with the natural temporal order on those intervals. Then we describe how our algorithm can be implemented in SQL (with views). In particular, we consider two alternative implementations of coalescing in SQL.

5.1 Bottom-up algorithm

We first introduce some notation and obtain a few results about *temporal tables* T with column names $\mathsf{attr}_1, \dots, \mathsf{attr}_m, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}$. A temporal table with $m = 0$ will be called *purely temporal*. We refer to the i-th row of T as $T[i]$, to the value of the column attr_j in the i-th row as $T[i, \mathsf{attr}_j]$, and set $T[i, \mathsf{attr}_j, \dots, \mathsf{attr}_k] = (T[i, \mathsf{attr}_j], \dots, T[i, \mathsf{attr}_k])$. We assume that the columns ledge and redge store timestamps or special values for ∞, $-\infty$, lpar stores [or (, and rpar stores $]$ or $)$. Define an order $\prec$ on intervals by taking $\langle t_1, t_2\rangle \prec \lceil s_1, s_2\rceil$ iff one of the following conditions holds:

– $t_1 < s_1$;

– $t_1 = s_1$, $\langle$ is $[$, and $\lceil$ is $($;

– $t_1 = s_1$, $\langle$ and $\lceil$ are the same, and $t_2 < s_2$;

– $t_1 = s_1$, $\langle$ and $\lceil$ are the same, $t_2 = s_2$, $\rangle$ is $)$, and $\rceil$ is $]$.

It should be clear that $\prec$ is a strict linear order on the set of all intervals. For example, we have $[3, 8) \prec [4, 7) \prec (4, 6) \prec (4, 7) \prec (4, 7]$. (In fact, the results of this section will work with any other linear order over intervals.) We write $T[i, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}] \prec T'[j, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}]$ to say that the interval defined by the ith row of a temporal table T $\prec$-precedes the interval given by the jth row of a temporal table T'.

We make the following *temporal ordering assumption* (or TOA), for any temporal table T with m attributes:

$$\text{if } T[i, \mathsf{attr}_1, \dots, \mathsf{attr}_m] = T[j, \mathsf{attr}_1, \dots, \mathsf{attr}_m] \text{ and } i < j, \\ \text{then } T[i, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}] \preceq T[j, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}].$$

For a purely temporal table T, this assumption means that the rows of T respect $\preceq$.

Let $T[\mathsf{attr}_j, \dots, \mathsf{attr}_k]$ be the *projection* of T on the columns $\mathsf{attr}_j, \dots, \mathsf{attr}_k$ that keeps only distinct tuples. We define $|T|_o$ to be the cardinality of $T[\mathsf{attr}_1, \dots, \mathsf{attr}_m]$ and $|T|_t$ to be the cardinality of $T[\mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}]$. The first measure estimates how large the table is in terms of individual constants, while the second measure concerns the number of timepoints. For the tables of extensional predicates in our use-cases, $|T|_o$ is much smaller than $|T|_t$.

We say that a table T is *coalesced* if it does not contain distinct tuples $(c_1, \dots, c_m, \langle, t_1, t_2, \rangle)$ and $(c_1, \dots, c_m, \lceil, t'_1, t'_2, \rceil)$ such that $\langle t_1, t_2\rangle \cap \lceil t'_1, t'_2\rceil \neq \emptyset$. For a tuple of individual constants $(c_1, \dots, c_m)$, let $T_{c_1,\dots,c_m}$ be the set of all intervals $\langle t_1, t_2\rangle$ such that $(c_1, \dots, c_m, \langle, t_1, t_2, \rangle)$ occurs in T. For a set $\mathcal{I}$ of intervals, we then denote by $\mathsf{coalesce}(\mathcal{I})$ the (minimal) set of intervals that results from coalescing $\mathcal{I}$. Finally, a *coalescing* of T is a minimal table, T^*, with the same columns as T such that the following condition holds:

(coalesce) for any $(c_1, \dots, c_m)$ in $T[\mathsf{attr}_1, \dots, \mathsf{attr}_m]$ and $\langle t_1, t_2\rangle$ in $\mathsf{coalesce}(T_{c_1,\dots,c_m})$, there exists $(c_1, \dots, c_m, \langle t_1, t_2\rangle)$ in T^*.

Clearly, T^* is a coalesced table.

Lemma 16. *Suppose a table T satisfies TOA. Then its coalescing T^* satisfying TOA and such that $|T^*|_o = |T|_o$ and $|T^*|_t \leq |T|_t$ can be computed in time $O(|T|_o^2 \times |T|_t)$.*

Proof. Consider first a purely temporal table S that satisfies temporal ordering. There is a simple linear-time algorithm to produce a coalesced table S^* that also satisfies temporal ordering. Indeed, initially we set $\langle b, e\rangle = S[0, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}]$. In a loop, we take each $\lceil t_1, t_2\rceil = S[i, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}]$ (clearly, $\langle b, e\rangle \prec \lceil t_1, t_2\rceil$). If $\lceil t_1, t_2\rceil$ and $\langle b, e\rangle$ are disjoint, we add $\langle b, e\rangle$ to S^* and set $\langle b, e\rangle = \lceil t_1, t_2\rceil$. If they are not disjoint, we set $\langle b, e\rangle = \lceil t_1, t_2\rceil \cup \langle b, e\rangle$ and move on. It is easily checked that the resulting table S^* is as required. Below, we refer to this algorithm as an *imperative* coalescing algorithm.

It only remains to explain how the algorithm above can be applied to T in order to obtain the required complexity. Note that $|T| \leq |T|_o \times |T|_t$ and we can construct $|T|_o$-many separate tables $T_{c_1,\dots,c_m}$, for each $(c_1, \dots, c_m)$, in time $|T| \times |T|_o$. Then, we can apply the algorithm described above to each $T_{c_1,\dots,c_m}$ in time $|T|_t$ and merge the results. Therefore, the overall running time is $|T| \times |T|_o + |T|_t \times |T|_o = O(|T|_o^2 \times |T|_t)$. $\square$

Before presenting our query answering algorithm, we determine the complexity of computing *temporal joins*. Let T be a table with attributes $\mathsf{attr}_1, \dots, \mathsf{attr}_m, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}$ and let T' be a table with attributes $\mathsf{attr}'_1, \dots, \mathsf{attr}'_n, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}$. A *temporal join* of T and T' is a table T'' with attributes $\mathsf{attr}''_1, \dots, \mathsf{attr}''_k, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}$ such that

$$\{\mathsf{attr}''_1, \dots, \mathsf{attr}''_k\} = \{\mathsf{attr}_1, \dots, \mathsf{attr}_m\} \cup \{\mathsf{attr}'_1, \dots, \mathsf{attr}'_n\}$$

and $(c''_1, \dots, c''_k, \langle, t''_1, t''_2, \rangle)$ is in T'' iff there exist two tuples $(c_1, \dots, c_m, \lceil, t_1, t_2, \rceil)$ from T and $(c'_1, \dots, c'_n, \lfloor, t'_1, t'_2, \rfloor)$ from T' satisfying the following conditions:

– $c''_i = c_j$, for all i, j such that $\mathsf{attr}''_i = \mathsf{attr}_j$;

– $c''_i = c'_j$, for all i, j such that $\mathsf{attr}''_i = \mathsf{attr}'_j$;

– $\lceil t_1, t_2 \rceil \cap \lfloor t'_1, t'_2 \rfloor \neq \emptyset$ and $\langle t''_1, t''_2 \rangle = \lceil t_1, t_2 \rceil \cap \lfloor t'_1, t'_2 \rfloor$.

Lemma 17. *If T, T' satisfy TOA, then a temporal join T'' of T and T' satisfying TOA and such that $|T''|_o \leq |T|_o \times |T'|_o$, $|T''|_t \leq |T|_t + |T'|_t$ can be computed in time $O(|T|_o^2 \times |T'|_o^2 \times (|T|_t + |T'|_t))$.*

Proof. We first give an algorithm for computing the temporal join of purely temporal tables S and S'. We assume that these tables are coalesced (which can be done in time $O(|S|)$ and $O(|S'|)$). The algorithm works starting from the first tuples $S[i]$ and $S'[i']$ of the tables. If $S[i] \cap S'[i'] \neq \emptyset$, we write $S[i] \cap S'[i']$ to the output table S''. Then, if $S[i+1] \succeq S'[i'+1]$, we set $i' := i' + 1$ (without changing i); otherwise, $i := i + 1$. We iterate until we have considered all the tuples in both tables. Clearly, computing the full S'' requires time $O(|S| + |S'|)$.

The complete algorithm for the tables T and T' will first, similarly to the argument of Lemma 16, produce $|T|_o$-many purely temporal tables $T_{c_1,\dots,c_m}$, for each $(c_1, \dots, c_m)$ occurring in T. Note that $|T_{c_1,\dots,c_m}| \leq |T|_t$ for each of those tables. In the same way, we produce $|T'|_o$ purely temporal tables $T'_{c'_1,\dots,c'_n}$, for each $(c'_1, \dots, c'_n)$ occurring in T'. It remains to apply the temporal join algorithm described above to all pairs of tables $T_{c_1,\dots,c_m}$ and $T'_{c'_1,\dots,c'_n}$, which can be done in the required time. $\square$

Another operation on temporal tables we need is projection. Let T be a table with column names as above and let $\{\mathsf{attr}'_1, \dots, \mathsf{attr}'_n\} \subseteq \{\mathsf{attr}_1, \dots, \mathsf{attr}_m\}$. A *projection* of T on $\mathsf{attr}'_1, \dots, \mathsf{attr}'_n$ is a table with columns $\mathsf{attr}'_1, \dots, \mathsf{attr}'_n, \mathsf{lpar}, \mathsf{ledge}, \mathsf{redge}, \mathsf{rpar}$ containing all $(c'_1, \dots, c'_n, \langle t_1, t_2 \rangle)$ such that some $(c_1, \dots, c_m, \langle t_1, t_2 \rangle)$ is in T and $c'_i = c_j$ whenever $\mathsf{attr}'_i = \mathsf{attr}_j$. As we have to preserve the temporal order, our algorithm for computing projections requires some attention. To show that a naïve projection does not preserve the temporal order, consider a table T with two tuples $(a, [, 1, 1,])$ and $(b, [, 0, 0,])$, which satisfies our temporal order assumption. The projection of T that removes the first column results is the table with two tuples $([, 1, 1,])$ and $([, 0, 0,])$, which is not ordered.

Lemma 18. *If T satisfies TOA, then a projection of T satisfying TOA can be computed in time $O(|T|_o^2 \times |T|_t)$.*

Now, consider the *union* operation on pairs of tables T and T' with the same columns that returns a table with all the tuples from the set $T \cup T'$.

Lemma 19. *For any pair of tables T and T' satisfying TOA, their union table also satisfying TOA can be computed in time $O((|T|_o^2 + |T'|_o^2) \times (|T|_t + |T'|_t))$.*

The proofs of Lemmas 18 and 19 can be found in the Appendix.

We are now in a position to describe the bottom-up query answering algorithm. Suppose we are given a program Π in normal form. Suppose also that each extensional predicate P is given by a table T_P satisfying TOA. (This assumption can be made in all of our use-cases. Indeed, both tables TB_Sensor and Weather are naturally ordered by the timestamp, and our mappings (see Section 6) can be easily written in a way to take advantage of this order and produce tables T satisfying TOA.) Thus, we can assume that the given data instance $\mathcal{D}$ is represented by a set of T_P, where each T_P contains all the tuples $(c_1, \dots, c_m, \langle, t_1, t_2, \rangle)$ such that $P(c_1, \dots, c_m)@\langle t_1, t_2\rangle \in \mathcal{D}$.

Consider a predicate P and suppose that we have computed temporal tables T_{P_i} satisfying TOA, for each P_i with $P \lessdot P_i$ (see Section 4). We assume that the T_{P_i} have (non-temporal) columns $(P_i, 1), \dots, (P_i, m)$. For each rule α in Π with P in the head, we compute a table T_P^α satisfying TOA. If α is of the form (2), we first compute the temporal join T of $T_{P_1}, \dots, T_{P_I}$ (we change the names so that T_{P_i} has columns $(P_i, \tau_1, 1), \dots, (P_i, \tau_m, m)$, where $\boldsymbol{\tau}_i = (\tau_1, \dots, \tau_m)$, and so all the tables T_{P_i} have distinct column names). Then we select from T only those tuples $(c_1, \dots, c_n, \langle, t_1, t_2, \rangle)$ for which $c_i = c_j$ in case the column names for c_i and c_j mention the same variable x, and the tuples for which $c_i = a$ in case the column name for c_i mentions the constant a. These two steps can be done in time $O(\prod_i |T_{P_i}|_o^2 \times \sum_i |T_{P_i}|_t)$, and the size of the resulting table does not exceed $\prod_i |T_{P_i}|_o \times \sum_i |T_{P_i}|_t$. It remains to perform projection in the following way. Suppose $P(\boldsymbol{\tau})$ with $\boldsymbol{\tau} = (x_1, \dots x_m)$ is the head of α (if $\boldsymbol{\tau}$ also contains constants, the procedure below can be easily modified). Then we keep only one column among all the columns named (P_i, x_j, k), for each variable x_j. It remains to rename the remaining (P_i, x_j, k) to (P, j), for each j. The total time required to compute T_P^α is $O(\prod_i |T_{P_i}|_o^2 \times \sum_i |T_{P_i}|_t)$.

If α is of the form (4), provided that T_{P_1} is coalesced, computing T_P^α reduces to using arithmetic operations for $\iota +^c \varrho$, $\iota -^c \varrho$, and $\varrho \sqsubseteq \iota$ as in the rules $(\boxplus_\varrho)$/$(\boxminus_\varrho)$, and projection. Therefore, T_P^α satisfying TOA can be computed in time $|T_{P_1}|_o^2 \times |T_{P_1}|_t$. Computing T_P^α for rules of the form (3) can be done in time $O(|T_{P_1}|_o \times |T_{P_2}|_o \times (|T_{P_1}|_t + T_{P_2}|_t))$. Indeed, to construct T_P^α for a rule α of the form $P(\boldsymbol{\tau}) \leftarrow P_1(\boldsymbol{\tau}_1)\, \mathcal{S}_\varrho\, P_2(\boldsymbol{\tau}_2))$, we follow the rule $(\mathcal{S}_\varrho)$ and first produce a table $T_{P_1}^c$ with the same columns as T_{P_1}, where for each tuple of T_{P_1}, we apply the operation $\cdot^c$ to its interval. We then compute the temporal join T of $T_{P_1}^c$ and T_{P_2} after applying the renaming described above. Then we compute $T^{+^o \varrho}$ by applying the operation $+^o \varrho$ to the interval columns of each tuple in T, after which we compute the temporal join of $T^{+^o \varrho}$ and $T_{P_1}^c$ (with renaming applied to the columns of $T_{P_1}^c$). To produce T_P^α, it remains to perform projection and renaming as described above. Finally, to compute T_P, it is sufficient to compute the union of all T_P^α satisfying TOA. Thus, we obtain the following, where the *degree* of the rule (2) is $|I|$, of (3) is 2, and of (4) is 1:

Lemma 20. *Let Π be a program and P a predicate in it such that K-many rules have P in the head, with R being the maximal degree of those rules, m the maximum of $|T_{P'}|_t$ among P' such that*

$P \lessdot P'$, and n the maximum of $|T_{P'}|_o$ among those P'. Then T_P is of size at most $n^R mRK$ and can be computed in time $O(n^{2R} mRK)$.

To compute the table for the goal Q, we iterate the described procedure as many times as the length of the longest chain of predicates in the dependence relation $\lessdot$ for Π. Thus, we obtain:

Theorem 21. *Let m be the maximum of $|T_P|_t$ among the extensional predicates P, and n the maximum of $|T_P|_o$ among those P. The overall time required to compute the goal predicate Q of Π is exponential in the size of Π, polynomial in n, and linear in m.*

Note that if all T_P are extracted from one table $\mathcal{R}$, as in our use-cases, then n corresponds to the number of individual tuples in $\mathcal{R}$, whereas m to the number of temporal intervals. It is to be emphasised that, in practice, programs tend to be small, and the number of individual constants is also small compared to the number of temporal intervals. The theorem above explains the linear patterns in our experiments below, where the size of individual tuples is fixed.

5.2 Implementation in SQL

Now, we show how to rewrite a given *datalog$_{nr}$MTL* query $(\Pi, Q(\boldsymbol{\tau})@x)$ with Π in normal form (2)–(4) to an SQL query computing the certain answers $(\boldsymbol{c}, \iota)$ to the query with *maximal* intervals ι. We illustrate the idea by a (relatively) simple example.

Consider the *datalog$_{nr}$MTL* query $(\Pi, \mathsf{HeatAffectedCounty}(\mathsf{county})@x)$, where

$$\Pi = \{\, \boxminus_{[0,24h]} \mathsf{ExcessiveHeat}(v) \leftarrow \boxminus_{[0,24h]}\mathsf{TempAbove24}(v) \land \diamondminus_{[0,24h]}\mathsf{TempAbove41}(v),$$
$$\mathsf{HeatAffectedCounty}(v) \leftarrow \mathsf{LocatedInCounty}(u, v) \land \mathsf{ExcessiveHeat}(u)\}$$

is part of the meteorological ontology from Section 6. First, we transform Π to normal form:

$$\Pi = \{\mathsf{ExcessiveHeat}(v) \leftarrow \diamondplus_{[0,24h]}\mathsf{X}(v),\ \ \mathsf{X}(v) \leftarrow \mathsf{Y}(v) \land \mathsf{Z}(v),$$
$$\mathsf{Y}(v) \leftarrow \boxminus_{[0,24h]}\mathsf{TempAbove24}(v),\ \ \mathsf{Z}(v) \leftarrow \diamondminus_{[0,24h]}\mathsf{TempAbove41}(v),$$
$$\mathsf{HeatAffectedCounty}(v) \leftarrow \mathsf{LocatedInCounty}(u, v) \land \mathsf{ExcessiveHeat}(u)\}.$$

We regard TempAbove24, TempAbove41, LocatedInCounty as extensional predicates given by the tables $T_{\mathsf{TempAbove24}}$, $T_{\mathsf{TempAbove41}}$, $T_{\mathsf{LocatedInCounty}}$. The first two of these tables have columns station_id, ledge, redge, and the third one station_id, county, ledge, redge. To simplify presentation, we omit the columns lpar and rpar used in the previous section and assume that all the temporal intervals take the form $(t, t']$; see Section 6.

For each predicate P in Π, we also create a view (temporary table) V_P^* with the same columns as T_P. We set $V_P^* = \mathsf{coalesce}(T_P)$, where coalesce is a query that implements coalescing in SQL[3] We explain the idea behind this query for a temporal table T (as mentioned above, we omit columns lpar, rpar). For a moment of time t occurring in T, we denote by $b^{\geq}(T, t)$ the number of i such that $T[i, \mathtt{ledge}] \geq t$, and by $e^{\geq}(T, t)$ the number of i such that $T[i, \mathtt{redge}] \geq t$; the numbers $b^{\leq}(T, t)$ and $e^{\leq}(T, t)$ are defined analogously. It can be readily seen that every t in $T[\mathtt{ledge}]$ such that $b^{\geq}(T, t) = e^{\geq}(T, t)$ is the beginning of some interval in the coalesced table T^*. Similarly, every

3. It should not be confused with the standard coalesce function in SQL that returns the first of its arguments that is not null, or null if all of the arguments are null.

t' in $T[\texttt{redge}]$ such that $b^{\leq}(T, t') = e^{\leq}(T, t')$ is the end of some interval in T^*. The coalesced intervals of T^* can be then obtained as pairs $(t, t'']$, where t is as above and t'' is the minimum over those t' defined above that are $\geq t$. Thus, to coalesce $T_{\mathsf{TempAbove24}}$ we first use the query

$$\begin{aligned}
V_l = \ & \texttt{SELECT}\ T.\mathsf{station_id}\ \texttt{AS}\ \mathsf{station_id},\ T.\mathsf{ledge}\ \texttt{AS}\ \mathsf{ledge}\ \texttt{FROM}\ T_{\mathsf{TempAbove24}}\ T\ \texttt{WHERE}\\
& (\texttt{SELECT COUNT}(*)\ \texttt{from}\ T_{\mathsf{TempAbove24}}\ S\ \texttt{WHERE}\ S.\mathsf{ledge} \geq\ T.\mathsf{ledge}\ \texttt{AND}\\
& \qquad S.\mathsf{station_id} = T.\mathsf{station_id}) =\\
& (\texttt{SELECT COUNT}(*)\ \texttt{from}\ T_{\mathsf{TempAbove24}}\ S\ \texttt{WHERE}\ S.\mathsf{redge} \geq\ T.\mathsf{ledge}\ \texttt{AND}\\
& \qquad S.\mathsf{station_id} = T.\mathsf{station_id}),
\end{aligned}$$

which extracts the pairs (n, t), where t is as described above and $\mathsf{station\_id} = n$. An analogous query can be used to produce V_r, a table of pairs (n, t'), where t' is as described above and $\mathsf{station\_id} = n$. Finally, we set

$$\begin{aligned}
V^*_{\mathsf{TempAbove24}} = \ & \texttt{SELECT}\ V_l.\mathsf{station_id}\ \texttt{AS}\ \mathsf{station_id},\ V_l.\mathsf{ledge}\ \texttt{AS}\ \mathsf{ledge},\\
& \qquad (\texttt{SELECT MIN}\ (V_r.\mathsf{redge})\ \texttt{FROM}\ V_r\ \texttt{WHERE}\ V_r.\mathsf{redge} \geq V_l.\mathsf{ledge}\ \texttt{AND}\\
& \qquad\qquad V_l.\mathsf{station_id} = V_r.\mathsf{station_id})\ \texttt{AS}\ \mathsf{redge}\\
& \texttt{FROM}\ V_l.
\end{aligned}$$

A more efficient variant of this algorithm that uses window functions with sorting and partitioning allows us to avoid joins used, e.g., in the query V_l (Zhou, Wang, & Zaniolo, 2006). We will refer to this algorithm in Section 7 as a *standard SQL* algorithm. In contrast to the imperative algorithm described in the proof of Lemma 16, this algorithm can be implemented using standard SQL operators.

In addition, for each intensional predicate P of Π, we create a view V_P defined by an SQL query that reflects the definitions of P in Π. For example, we set

$$\begin{aligned}
V_{\mathsf{Y}} = \ & \mathsf{SELECT}\ V^*_{\mathsf{TempAbove24}}.\mathsf{station_id}\ \mathsf{AS}\ \mathsf{station_id},\\
& \qquad V^*_{\mathsf{TempAbove24}}.\mathsf{ledge} + 24h\ \mathsf{AS}\ \mathsf{ledge},\ V^*_{\mathsf{TempAbove24}}.\mathsf{redge}\ \mathsf{AS}\ \mathsf{redge},\\
& \mathsf{FROM}\ V^*_{\mathsf{TempAbove24}}\ \mathsf{WHERE}\ V^*_{\mathsf{TempAbove24}}.\mathsf{redge} - V^*_{\mathsf{TempAbove24}}.\mathsf{ledge} \geq 24h.
\end{aligned}$$

This query implements the $\iota +^c \varrho$ operation for $\varrho = [0, 24h]$, and the WHERE clause checks whether $\varrho \sqsubseteq \iota$ holds, where $\iota = (V^*_{\mathsf{TempAbove24}}.\mathsf{ledge}, V^*_{\mathsf{TempAbove24}}.\mathsf{redge}]$. We then set $V^*_{\mathsf{Y}} = \mathsf{coalesce}(V_{\mathsf{Y}})$ and note that the query

$$\mathsf{SELECT}\ \mathsf{station\_id},\ \mathsf{ledge}, \mathsf{redge}\ \mathsf{FROM}\ V^*_{\mathsf{Y}}, \tag{24}$$

when evaluated over the tables $T_{\mathsf{TempAbove24}}$, $T_{\mathsf{TempAbove41}}$ and $T_{\mathsf{LocatedInCounty}}$, would produce the answers to the query $(\Pi, \mathsf{Y}(\mathsf{station\_id}, \mathsf{county})@x)$ with *maximal* intervals $\iota = (\iota_b, \iota_e]$, where ι_b corresponds to ledge, and ι_e to redge.

We now explain how to construct queries for the concepts whose definitions involve $\wedge$ using the example of HeatAffectedCounty:

$$\begin{aligned}
V_{\texttt{HeatAffectedCounty}} = \ & \texttt{SELECT}\ V^*_{\texttt{LocatedInCounty}}.\texttt{county}\ \texttt{AS}\ \texttt{county},\\
& \qquad \texttt{MX}(V^*_{\texttt{LocatedInCounty}}.\texttt{ledge}, V^*_{\texttt{ExcessiveHeat}}.\texttt{ledge})\ \texttt{AS}\ \texttt{ledge},\\
& \qquad \texttt{MN}(V^*_{\texttt{LocatedInCounty}}.\texttt{redge}, V^*_{\texttt{ExcessiveHeat}}.\texttt{redge})\ \texttt{AS}\ \texttt{redge}\\
& \texttt{FROM}\ V^*_{\texttt{LocatedInCounty}}, V^*_{\texttt{ExcessiveHeat}}
\end{aligned}$$

```
function ans(q(v,x) = Q(τ)@x, Π, M, D):
  V = views(Π, M, Q)
  ans = SELECT projects(v,τ), ι AS x FROM V_Q
  return eval(ans ∧ V, D)

function views(Π, M, Q):
  V = ∅
  for each predicate P defined by M:
    V = V ∪ {V_P=coalesce(UNION({SELECT projects(τ,τ), T_1.ι AS ι FROM sql AS T_1
                                          | P(τ)@ι←sql ∈ M}))}
  let ⋖ be the dependence relation on the predicates in Π
  for each intensional predicate P with Q ⋖ P or Q = P:
    V = V ∪ {V_P=coalesce(UNION({view(r,P) | r ∈ Π_P}))}
  return V

function view(r,P):
  if r = P(τ) ← ⊞_ϱ P_1(τ_1):
    V_P^r = SELECT projects(τ,τ_1), T_1.ι −^c ϱ AS ι
             FROM V_P1 AS T_1 WHERE join-cond(τ_1) AND ϱ ⊑ T_1.ι
  else if r = P(τ) ← ⊟_ϱ P_1(τ_1):
    V_P^r = SELECT projects(τ,τ_1), T_1.ι +^c ϱ AS ι
             FROM V_P1 AS T_1 WHERE join-cond(τ_1) AND ϱ ⊑ T_1.ι
  else if r = P(τ) ← P_1(τ_1) S_ϱ P_2(τ_2):
    V_P^r = SELECT projects(τ), ((T_1.ι^c ∩ T_2.ι) +^o ϱ) ∩ T_1.ι^c AS ι
            FROM V_P1 AS T_1, V_P2 AS T_2
            WHERE join-cond(τ_1,τ_2)) AND ((T_1.ι^c ∩ T_2.ι) +^o ϱ) ∩ T_1.ι^c ≠ ∅
  else if r = P(τ) ← P_1(τ_1) U_ϱ P_2(τ_2):
    V_P^r = SELECT projects(τ,τ_1,τ_2), ((T_1.ι^c ∩ T_2.ι) −^o ϱ) ∩ T_1.ι^c AS ι
            FROM V_P1 AS T_1, V_P2 AS T_2
            WHERE join-cond(τ_1,τ_2) AND ((T_1.ι^c ∩ T_2.ι) −^o ϱ) ∩ T_1.ι^c ≠ ∅
  else if r = P(τ) ← P_1(τ_1),...,P_n(τ_n):
    V_P^r = SELECT projects(τ,τ_1,...,τ_n), T_1.ι ∩ ... ∩ T_n.ι AS ι
            FROM V_P1 AS T_1, ···, V_Pn AS T_n
            WHERE join-cond(τ_1,...,τ_n) AND T_1.ι ∩ ... T_n.ι ≠ ∅
  return V_P^r

function projects(τ,τ_1,...,τ_n):
  columns = {}
  for each v_i ∈ τ = v_1,...,v_m:
    let k,j be a pair of integers such that τ_k[j] = v_i
    columns.add(T_k.attr_j AS attr_i)
  return columns

function join-cond(τ_1,...,τ_n):
  cond = true
  for each pair of different positions τ_l[i] and τ_r[j] such that τ_l[i] = τ_r[j]
    cond = cond AND (T_l.attr_i = T_r.attr_j)
  return cond
```

Figure 3: The algorithm for evaluating *datalog*$_{nr}$*MTL* queries in SQL.

$$\begin{aligned}&\texttt{WHERE MX}(V^*_{\texttt{LocatedInCounty}}.\texttt{ledge}, V^*_{\texttt{ExcessiveHeat}}.\texttt{ledge}) <\\ &\quad\texttt{MN}(V^*_{\texttt{LocatedInCounty}}.\texttt{redge}, V^*_{\texttt{ExcessiveHeat}}.\texttt{redge})\\ &\quad\texttt{AND } V^*_{\texttt{LocatedInCounty}}.\texttt{county} = V^*_{\texttt{ExcessiveHeat}}.\texttt{county},\end{aligned}$$

where MN (MX) is the function that returns the earliest (latest) of any two given date/time values (it can be implemented in SQL as a user-defined function, or using the CASE operator). Finally, we use a query similar to (24) over $V^*_{\mathsf{HeatAffectedCounty}}$ to produce the answers to $(\Pi, \boldsymbol{q}(\mathsf{county}, x))$.

We are mostly interested in the scenario where the tables T_P are not available immediately, but extracted from raw timestamped data tables R by means of mappings. In this case, we use views V_P instead of T_P defined over R. For example, if the raw data is stored in the table Weather, we define the view:

```
V_TempAbove24 = SELECT sid, ledge, redge
                FROM (SELECT station_id AS sid,
                            LAG(date_time, 1) OVER (w) AS ledge,
                            date_time AS redge
                      FROM Weather
                      WINDOW w AS (PARTITION BY station_id ORDER BY date_time)
                ) tmp
                WHERE air_temp_set_1 >= 24.
```

Our general rewriting algorithm is outlined in Fig. 3, where the function ans produces an SQL query that computes the certain answers to $(\Pi, Q(\boldsymbol{\tau})@x)$ (with maximal intervals) by evaluating the query over the input database $\mathcal{D}$. The algorithm is a variation of the standard translation of non-recursive Datalog to relational algebra—see, e.g., (Ullman, 1988)—extended with the operations on temporal intervals described above (they are underlined in Fig. 3).

It is to be noted that the 'views' introduced by the algorithm do not require modifying the underlying database. They can be implemented in different ways: for example, by using subqueries, common table expressions (CTEs), or temporary tables. For the experiments in Section 7, we use the last approach, where temporary tables are generated on the fly and exist only within a transaction.

6. Use Cases

We test the feasibility of OBDA with *datalog$_{nr}$MTL* by querying Siemens turbine log data and MesoWest weather data. In this section, we briefly describe these use cases; detailed results of our experiments will presented in Section 7.

Siemens service centres store aggregated turbine sensor data in tables such as TB_Sensor. The data comes with (not necessarily regular) timestamps $t_1, t_2, \dots$, 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

$$\begin{aligned}&\mathsf{ActivePowerAbove1.5(tb0)}@[12{:}20{:}48, 12{:}20{:}49),\\ &\mathsf{ActivePowerAbove1.5(tb0)}@[12{:}20{:}49, 12{:}20{:}52),\\ &\mathsf{RotorSpeedAbove1500(tb0)}@[12{:}20{:}48, 12{:}20{:}49),\\ &\mathsf{MainFlameBelow0.1(tb0)}@[12{:}20{:}48, 12{:}20{:}52).\end{aligned}$$

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

$$
\begin{array}{l}
\mathsf{ActivePowerAbove1.5}(\mathsf{tbid})@[\mathtt{ledge}, \mathtt{redge}) \leftarrow \\
\quad \mathtt{SELECT}\ \mathtt{tbid},\ \mathtt{ledge},\ \mathtt{redge}\ \mathtt{FROM}\ (\\
\quad\quad \mathtt{SELECT}\ \mathtt{turbineId}\ \mathtt{AS}\ \mathtt{tbid}, \\
\quad\quad\quad \mathtt{LAG}(\mathtt{dateTime}, 1)\ \mathtt{OVER}\ (\mathtt{w})\ \mathtt{AS}\ \mathtt{ledge}, \\
\quad\quad\quad \mathtt{LAG}(\mathtt{activePower}, 1)\ \mathtt{OVER}\ (\mathtt{w})\ \mathtt{AS}\ \mathtt{lag\_activePower}, \\
\quad\quad\quad \mathtt{dateTime}\ \mathtt{AS}\ \mathtt{redge} \\
\quad\quad \mathtt{FROM}\ \mathsf{TB\_Sensor} \\
\quad\quad \mathtt{WINDOW}\ \mathtt{w}\ \mathtt{AS}\ (\mathtt{PARTITION}\ \mathtt{BY}\ \mathtt{turbineId}\ \mathtt{ORDER}\ \mathtt{BY}\ \mathtt{dateTime}) \\
\quad\quad)\ \mathtt{tmp}\ \mathtt{WHERE}\ \mathtt{lag\_activePower}\ >\ \mathtt{1.5}
\end{array}
$$

In terms of the basic predicates above, we define more complex ones that are used in queries posed by the Siemens engineers:

$$
\begin{array}{l}
\mathsf{NormalRestart}(v) \leftarrow \mathsf{NormalStart}(v) \land \diamondminus_{(0,1h]} \mathsf{NormalStop}(v), \\
\mathsf{NormalStop}(v) \leftarrow \mathsf{CoastDown1500to200}(v) \land \diamondminus_{(0,9m]} \big[\mathsf{CoastDown6600to1500}(v) \land \\
\hfill \diamondminus_{(0,2m]} \big(\mathsf{MainFlameOff}(v) \land \diamondminus_{(0,2m]} \mathsf{ActivePowerOff}(v)\big)\big], \\
\mathsf{MainFlameOff}(v) \leftarrow \boxminus_{[0s,10s]} \mathsf{MainFlameBelow0.1}(v), \\
\mathsf{ActivePowerOff}(v) \leftarrow \boxminus_{[0s,10s]} \mathsf{MainPowerBelow0.15}(v), \\
\mathsf{CoastDown6600to1500}(v) \leftarrow\ \boxminus_{[0s,30s]}\ \mathsf{RotorSpeedBelow1500}(v) \land \\
\hfill \diamondminus_{(0,2m]}\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedAbove6600}(v), \\
\mathsf{CoastDown1500to200}(v) \leftarrow\ \boxminus_{[0s,30s]}\ \mathsf{RotorSpeedBelow200}(v) \land \\
\hfill \diamondminus_{(0,9m]}\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedAbove1500}(v), \\
\mathsf{NormalStart}(v) \leftarrow \mathsf{STCtoRUCReached}(v) \land \diamondminus_{(0,30s]} \big[\mathsf{RampChange1\text{-}2Reached}(v) \land \\
\quad\quad \diamondminus_{(0,5m]} \big(\mathsf{PurgingIsOver}(v) \land \diamondminus_{(0,11m]} \big(\mathsf{PurgeAndIgnitionSpeedReached}(v) \land \\
\hfill \diamondminus_{(0,15s]} \mathsf{FromStandStillTo180}(v)\big)\big)\big], \\
\mathsf{STCtoRUCReached}(v) \leftarrow \boxminus_{(0,30s]} \mathsf{RotorSpeedAbove4800}(v) \land \\
\hfill \diamondminus_{(0,2m]}\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedBelow4400(v)}, \\
\mathsf{RampChange1\text{-}2Reached}(v) \leftarrow \boxminus_{(0s,30s]} \mathsf{RotorSpeedAbove4400}(v) \land \\
\hfill \diamondminus_{(0,6.5m]}\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedBelow1500}(v), \\
\mathsf{PurgingIsOver}(v) \leftarrow\ \boxminus_{[0s,10s]}\ \mathsf{MainFlameOn}(v) \land \\
\diamondminus_{(0,10m]} \big[\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedAbove1260}(v) \land \diamondminus_{(0,2m]}\ \boxminus_{(0,1m]}\ \mathsf{RotorSpeedBelow1000}(v)\big], \\
\mathsf{PurgeAndIgnitionSpeedReached}(v) \leftarrow\ \boxminus_{[0s,30s]}\ \mathsf{RotorSpeedAbove1260}(v) \land \\
\hfill \diamondminus_{(0,2m]}\ \boxminus_{(0,30s]}\ \mathsf{RotorSpeedBelow200}(v), \\
\mathsf{FromStandStillTo180}(v) \leftarrow\ \boxminus_{[0s,1m]}\ \mathsf{RotorSpeedAbove180}(v) \land \\
\hfill \diamondminus_{(0,1.5m]}\ \boxminus_{(0,1m]}\ \mathsf{RotorSpeedBelow60}(v).
\end{array}
$$

MesoWest. The MesoWest[4] 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 some periodicity, with the output at 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:

stationId	dateTime	airTemp	windSpeed	windDir	hourPrecip	...
		...				
KBVY	2013-02-15;15:14	8	45	10	0.05	
KMNI	2013-02-15;15:21	6	123	240	0	
KBVY	2013-02-15;15:24	8	47	10	0.08	
KMNI	2013-02-15;15:31	6.7	119	220	0	
		...				

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

stationId	county	state	latitude	longitude	...
		...			
KBVY	Essex	Massachusetts	42.58361	-70.91639	
KMNI	Essex	Massachusetts	33.58333	-80.21667	
		...			

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

$$\begin{aligned}
&\mathsf{NorthWind}(\mathsf{KBVY})@(15{:}14, 15{:}24],\\
&\mathsf{HurricaneForceWind}(\mathsf{KMNI})@(15{:}21, 15{:}31],\\
&\mathsf{Precipitation}(\mathsf{KBVY})@(15{:}14, 15{:}24],\\
&\mathsf{TempAbove0}(\mathsf{KBVY})@(15{:}14, 15{:}24],\\
&\mathsf{TempAbove0}(\mathsf{KMNI})@(15{:}21, 15{:}31]
\end{aligned}$$

(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

$$\begin{aligned}
&\mathsf{LocatedInCounty}(\mathsf{KBVY}, \mathsf{Essex})@(-\infty, \infty),\\
&\mathsf{LocatedInState}(\mathsf{KBVY}, \mathsf{Massachusetts})@(-\infty, \infty).
\end{aligned}$$

Our ontology contains definitions of various meteorological terms:

$$\begin{aligned}
&\mathsf{ShoweryCounty}(v) \leftarrow \mathsf{LocatedInCounty}(u_1, v) \land \mathsf{LocatedInCounty}(u_2, v) \land\\
&\qquad\qquad\qquad \mathsf{Precipitation}(u_1) \land \mathsf{NoPrecipitation}(u_2) \land \diamondminus_{(0,30m]}\mathsf{Precipitation}(u_2),\\
&\boxminus_{[0,1h]}\,\mathsf{Hurricane}(v) \leftarrow \boxminus_{[0,1h]}\mathsf{HurricaneForceWind}(v),\\
&\mathsf{HurricaneAffectedState}(v) \leftarrow \mathsf{LocatedInState}(u, v) \land \mathsf{Hurricane}(u),
\end{aligned}$$

4. `http://mesowest.utah.edu/`

(a) Siemens data for one turbine.

# of months	32	64	96	128	159	191	223	255	287	320
# of rows (approx.)	13 M	26 M	39 M	52 M	65 M	77 M	90 M	103 M	116 M	129 M
size (GB)	0.57	1.2	1.7	2.3	2.9	3.4	4.0	4.5	5.1	5.7

(b) NY weather stations from 2005 to 2014.

# of years # of stations	1 229	2 306	3 370	4 441	5 484	6 542	7 595	8 643	9 807	10 874
# of rows (approx.)	4 M	11 M	19 M	27 M	36 M	49 M	63 M	79 M	99 M	124 M
size (GB)	0.2	0.6	1.1	1.6	2.1	2.9	3.8	4.8	5.9	7.4

(c) Weather data for 1–19 states in 2012.

states	DE, GA	+NY	+MD	+NJ, RI	+MA, CT	+LA, VT	+ME, WV	+NH, NC	+MS,SC, ND	+KY, SD
# of states # of stations	2 408	3 659	4 1120	6 1476	8 1875	10 2305	12 2669	14 3019	17 3508	19 4037
# of rows (approx.)	17 M	32 M	41 M	52 M	67 M	81 M	93 M	106 M	121 M	141 M
size (GB)	0.9	1.9	2.5	3.1	4.0	4.8	5.5	6.4	7.2	8.3

Table 4: Data sets used in the experiments (the size measured for data in CSV format).

$$\boxminus_{[0,24h]} \mathsf{ExcessiveHeat}(v) \leftarrow \boxminus_{[0,24h]} \mathsf{TempAbove24}(v) \land \diamondminus_{[0,24h]} \mathsf{TempAbove41}(v),$$

$$\mathsf{HeatAffectedCounty}(v) \leftarrow \mathsf{LocatedInCounty}(u, v) \land \mathsf{ExcessiveHeat}(u),$$

$$\begin{aligned}\mathsf{CyclonePatternState}(v) \leftarrow{}& \mathsf{LocatedInState}(u_1, v) \land \mathsf{LocatedInState}(u_2, v) \land \\ & \mathsf{LocatedInState}(u_3, v) \land \mathsf{LocatedInState}(u_4, v) \land \mathsf{EastWind}(u_1) \land \\ & \mathsf{NorthWind}(u_2) \land \mathsf{WestWind}(u_3) \land \mathsf{SouthWind}(u_4).\end{aligned}$$

7. Experiments

To evaluate the performance of the SQL queries produced by the *datalog$_{nr}$MTL* rewriting algorithm outlined in Section 5.2, we developed two benchmarks for our use cases. We ran the experiments on an HP Proliant server with 2 Intel Xeon X5690 Processors (each with 12 logical cores at 3.47GHz), 106GB of RAM and five 1TB 15K RPM HD. We used both PostgreSQL 9.6 and the SQL interface (Armbrust, Xin, Lian, Huai, Liu, Bradley, Meng, Kaftan, Franklin, Ghodsi, & Zaharia, 2015) of Apache Spark 2.1.0. Apache Spark is a cluster-computing framework that provides distributed task dispatching, scheduling and data parallelisation. For each of these two systems, we provided two different implementations, imperative and standard SQL, which diverge in the computation of maximal intervals; see Section 5.

We run all the queries with a timeout of 30 minutes.

Siemens provided us with a sample of data for one running turbine, which we denote by $\mathsf{tb0}$, over 4 days in the form of the table $\mathsf{TB\_Sensor}$. The data table was rather sparse, containing a lot of nulls, because different sensors recorded data at different frequencies. For example, $\mathsf{ActivePower}$ arrived most frequently with average periodicity of 7 seconds, whereas the values for the field $\mathsf{MainFlame}$ arrived most rarely, every 1 minute on average. We replicated this sample to imitate

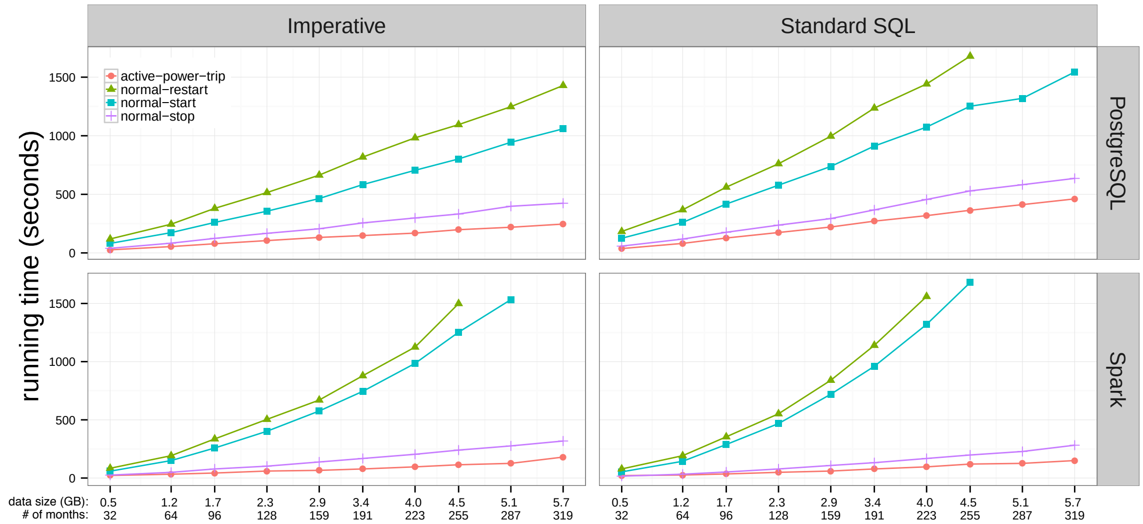


Figure 5: Experiment results for the Siemens use case.

the data for one turbine over 10 different periods ranging from 32 to 320 months. The statistics of the data sets are given in Tables 4a and 8a. We evaluated four queries $\mathsf{ActivePowerTrip}(\mathsf{tb0})@x$, $\mathsf{NormalStart}(\mathsf{tb0})@x$, $\mathsf{NormalStop}(\mathsf{tb0})@x$ and $\mathsf{NormalRestart}(\mathsf{tb0})@x$. The statistics of returned answers is given in Table 7a.

The execution times for the Siemens use case are given in Fig. 5. Although Apache Spark was designed to perform efficient parallel computations, it failed to take advantage of this feature due to the fact that the Siemens data could not be partitioned by mapping each part to a separate core. PostgreSQL 9.6 also supports parallel query execution in some cases. However, as many operators (e.g., scans of temporary tables) in our queries are classified either 'parallel unsafe' or 'parallel restricted' in the parallel safety documentation[5], the query planner failed to produce any parallel execution strategy in our case. The reason why PostgreSQL outperformed Apache Spark is that the latter does not provide a convenient way to define proper indexes over temporary tables, which leads to quadratically growing running times. On the other hand, PostgreSQL shows linear growth in the size of data (confirming theoretical results since we deal with a single turbine).

Note that the normal restart (start) query timeouts on the data for more than 18 (respectively, 21) years, which is more than enough for the monitoring and diagnostics tasks at Siemens, where the two most common application scenarios for sensor data analytics are daily monitoring (that is, analytics of high-frequency data of the previous 24 hours) and fleet-level analytics of key-performance indicators over one year. In both cases, the computation time of the results is far less a crucial cost factor than the lead-time for data preparation.

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

5. https://www.postgresql.org/docs/9.6/static/parallel-safety.html

between 2005 and 2014; see Tables 4b and 8b. 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 4c and 8c. In both cases, we executed four *datalog$_{nr}$MTL* queries $\mathsf{ShoweryCounty}(v)@x$, $\mathsf{HurricaneAffectedState}(\mathrm{NY})@x$, $\mathsf{HeatAffectedCounty}(v)@x$, $\mathsf{CyclonePatternState}(\mathrm{NY})@x$. The statistics of the returned answers is shown in Tables 7b and 7c.

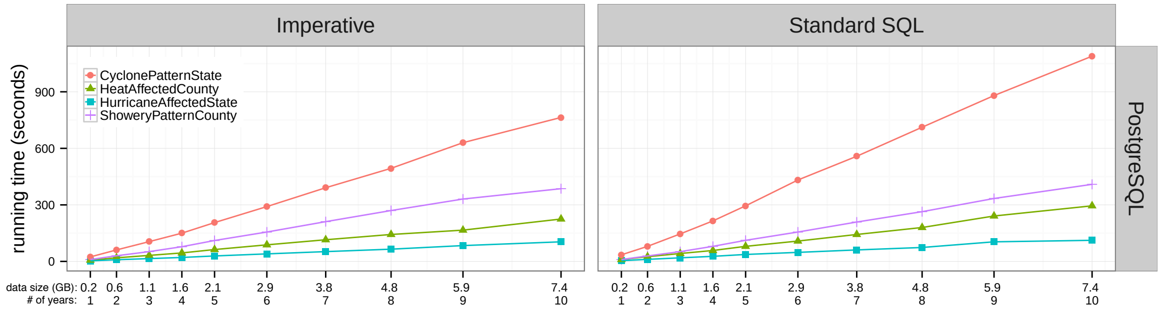
The execution times are shown in Fig. 6. All the four queries can be answered within the time limit. The most expensive one is the cyclone pattern state query because its definition includes a join of four atoms for winds in four directions, each with a large volume of instances. All the four sub-figures in Fig. 6 exhibit linear behaviour with respect to the size of data. The nearly tenfold better performance of Spark over PostgreSQL can be explained by the fact that, unlike the data in the Siemens case, the MesoWest data is highly parallelisable. Since it was collected from hundreds of different weather stations, it can be partitioned by station id, state, county, etc. to perfectly fit the MapReduce programming model extended with resilient distributed datasets (RDDs) (Zaharia, Xin, Wendell, Das, Armbrust, Dave, Meng, Rosen, Venkataraman, Franklin, Ghodsi, Gonzalez, Shenker, & Stoica, 2016). In this case, Apache Spark is able to take advantage of the multi-core and large memory hardware infrastructure, to compute mappings and coalescing in parallel, making it 10 times faster than PostgreSQL; see Figures 6b and 6d.

Overall, the results of the experiments look very encouraging: our *datalog$_{nr}$MTL* query rewriting algorithm produces SQL queries that are executable by a standard database engine PostgreSQL in acceptable time, and by a cluster-computing framework Apache Spark in better than acceptable time (in case data can be properly partitioned) over large sets of real-world temporal data of up to 8.3GB in CSV format. The relatively challenging queries such as NormalRestart and CyclonePatternState require a large number of temporal joins, which turn out to be rather expensive.

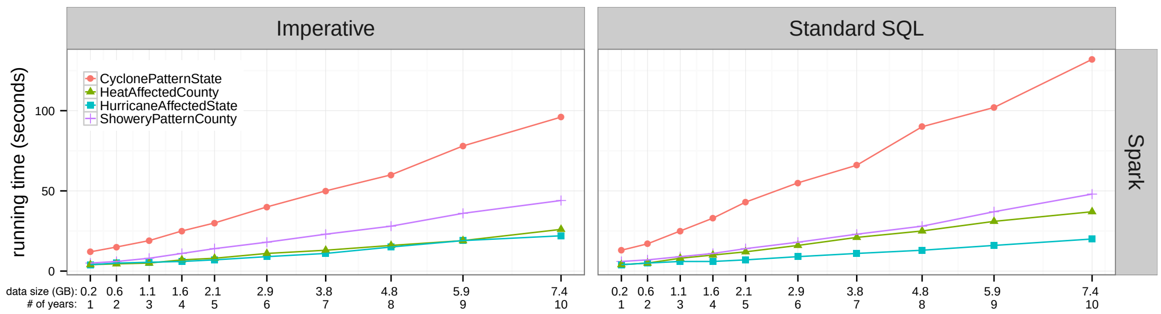
8. Conclusions and Future Work

To facilitate access to sensor temporal data with the aim of monitoring and diagnostics, we suggested the ontology language *datalogMTL*, an extension of datalog with the Horn fragment of the metric temporal logic *MTL* (under the continuous semantics). 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^0 for data complexity. We tested feasibility and efficiency of OBDA with *datalog$_{nr}$MTL* on two real-world use cases by querying Siemens turbine data and MesoWest weather data. Namely, we designed *datalog$_{nr}$MTL* ontologies defining typical concepts used by Siemens engineers and various meteorological terms, developed and implemented an algorithm rewriting *datalog$_{nr}$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. (To the best of our knowledge, this is the first work on practical OBDA with temporal ontologies, and so no other systems with similar functionalities are available for comparison.)

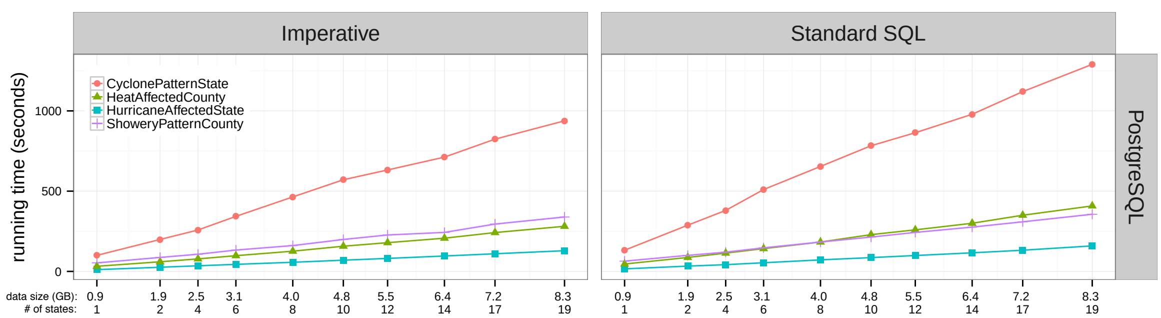
(a) Experiments over New York data of 2005–2014 (PostgreSQL).



(b) Experiments over New York data of 2005–2014 (Spark).



(c) Experiments over 1 year data from 1–19 states (PostgreSQL).



(d) Experiments over 1 year data from 1–19 states (Spark).

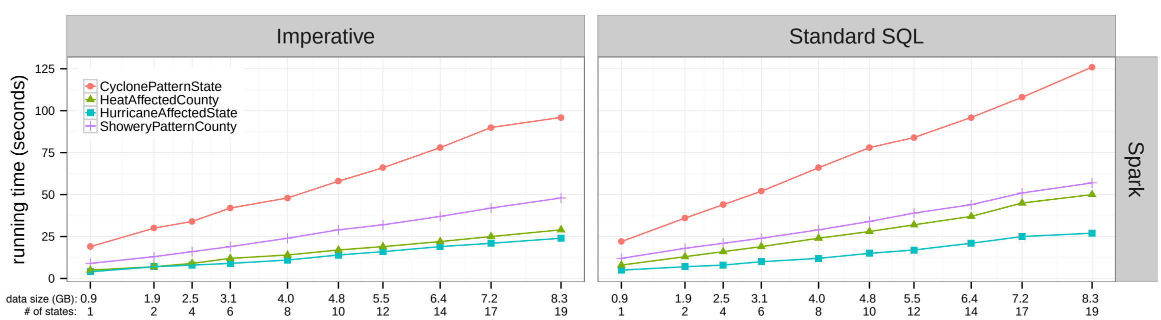


Figure 6: Experiment results for the MesoWest use case.

Based on these encouraging results, we plan to include our temporal OBDA framework into the Ontop platform[6] (Rodriguez-Muro et al., 2013; Kontchakov, Rezk, Rodriguez-Muro, Xiao, & Zakharyaschev, 2014; Calvanese et al., 2017). Note also that *datalogMTL* presented here has been recently used to develop an ontology of ballet moves (see Example 2) that underlies a search engine of annotated sequences in ballet videos (Raheb et al., 2017). This is a third use case for our framework (and we are aware of a few more emerging use cases), which makes an efficient and user-friendly implementation of the framework a top priority.

We are also working on the streaming data setting, where the challenge is to continuously evaluate queries over the incoming data. A rule-based language with window operators for analysing streaming data has been suggested by Beck, Dao-Tran, Eiter, & Fink (2015). This language is very expressive as it uses an abstract semantics for window operators (which does not have to guarantee decidability) and allows negation and disjunction in the rules. It would be interesting to identify and adapt a suitable fragment of this language in our temporal OBDA framework.

Acknowledgements

This work was supported by the UK EPSRC grant EP/M012670 'iTract: Islands of Tractability in Ontology-Based Data Access' and by the OBATS project at the Free University of Bozen-Bolzano.

References

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

Alur, R., Feder, T., & Henzinger, T. A. (1996). The benefits of relaxing punctuality. *J. ACM*, *43*(1), 116–146.

Alur, R., & Henzinger, T. A. (1993). Real-time logics: Complexity and expressiveness. *Inf. Comput.*, *104*(1), 35–77.

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., & Zaharia, M. (2015). Spark SQL: relational data processing in spark. In Sellis, T. K., Davidson, S. B., & Ives, Z. G. (Eds.), *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pp. 1383–1394. ACM.

Arora, S., & Barak, B. (2009). *Computational Complexity: A Modern Approach* (1st edition). Cambridge University Press, New York, USA.

Artale, A., Kontchakov, R., Wolter, F., & Zakharyaschev, M. (2013). Temporal description logic for ontology-based data access. In *Proc. of the 23rd Int. Joint Conf. on Artificial Intelligence, IJCAI 2013*. IJCAI/AAAI.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., & Zakharyaschev, M. (2015). First-order rewritability of temporal ontology-mediated queries. In *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence, IJCAI 2015*, pp. 2706–2712. IJCAI/AAAI.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., & Zakharyaschev, M. (2017). Ontology-mediated query answering over temporal data: A survey (invited talk). In *TIME*, Vol. 90 of *LIPIcs*, pp. 1:1–1:37. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

6. `http://ontop.inf.unibz.it/`

Artale, A., Kontchakov, R., Ryzhikov, V., & Zakharyaschev, M. (2013). The complexity of clausal fragments of LTL. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, pp. 35–52.

Artale, A., Kontchakov, R., Ryzhikov, V., & Zakharyaschev, M. (2014). A cookbook for temporal conceptual data modelling with description logics. *ACM Trans. Comput. Log.*, *15*(3), 25:1–25:50.

Baader, F., Borgwardt, S., & Lippmann, M. (2013). Temporalizing ontology-based data access. In *Proc. of the 24th Int. Conf. on Automated Deduction, CADE-24*, Vol. 7898 of *LNCS*, pp. 330–344. Springer.

Baudinet, M., Chomicki, J., & Wolper, P. (1993). Temporal deductive databases. In *Temporal Databases*, pp. 294–320.

Beck, H., Dao-Tran, M., Eiter, T., & Fink, M. (2015). LARS: A logic-based framework for analyzing reasoning over streams. In Bonet, B., & Koenig, S. (Eds.), *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pp. 1431–1438. AAAI Press.

Bienvenu, M., Kikot, S., Kontchakov, R., Podolskii, V. V., & Zakharyaschev, M. (2018). Ontology-mediated queries: Combined complexity and succinctness of rewritings via circuit complexity. *J. ACM*. In print.

Borgwardt, S., Lippmann, M., & Thost, V. (2013). Temporal query answering in the description logic DL-Lite. In *Proc. of the 9th Int. Symposium on Frontiers of Combining Systems, FroCoS'13*, Vol. 8152 of *LNCS*, pp. 165–180. Springer.

Brandt, S., Kalaycı, E. G., Kontchakov, R., Ryzhikov, V., Xiao, G., & Zakharyaschev, M. (2017). Ontology-based data access with a horn fragment of metric temporal logic. In Singh, S. P., & Markovitch, S. (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 1070–1076. AAAI Press.

Bresolin, D., Kurucz, A., Muñoz-Velasco, E., Ryzhikov, V., Sciavicco, G., & Zakharyaschev, M. (2017). Horn fragments of the halpern-shoham interval temporal logic. *ACM Trans. Comput. Log.*, *18*(3), 22:1–22:39.

Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., & Xiao, G. (2017). Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, *8*(3), 471–487.

Chomicki, J., & Toman, D. (1998). Temporal logic in information systems. In *Logics for Databases and Information Systems*, pp. 31–70. Kluwer.

Furia, C. A., & Spoletini, P. (2008). Tomorrow and all our yesterdays: MTL satisfiability over the integers. In Fitzgerald, J. S., Haxthausen, A. E., & Yenigün, H. (Eds.), *Theoretical Aspects of Computing - ICTAC 2008, 5th International Colloquium, Istanbul, Turkey, September 1-3, 2008. Proceedings*, Vol. 5160 of *Lecture Notes in Computer Science*, pp. 126–140. Springer.

Gabbay, D., Kurucz, A., Wolter, F., & Zakharyaschev, M. (2003). *Many-Dimensional Modal Logics: Theory and Applications*, Vol. 148.

Gottlob, G., Kikot, S., Kontchakov, R., Podolskii, V. V., Schwentick, T., & Zakharyaschev, M. (2014). The price of query rewriting in ontology-based data access. *Artif. Intell.*, *213*, 42–59.

Gutiérrez-Basulto, V., Jung, J., & Kontchakov, R. (2016a). Temporalized EL ontologies for accessing temporal data: Complexity of atomic queries. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press.

Gutiérrez-Basulto, V., Jung, J. C., & Ozaki, A. (2016b). On metric temporal description logics. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pp. 837–845.

Gutiérrez-Basulto, V., & Klarman, S. (2012). Towards a unifying approach to representing and querying temporal data in description logics. In *Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems, RR 2012*, Vol. 7497 of *LNCS*, pp. 90–105. Springer.

Kharlamov, E., Brandt, S., Jiménez-Ruiz, E., Kotidis, Y., Lamparter, S., Mailis, T., Neuenstadt, C., Özçep, Ö. L., Pinkel, C., Svingos, C., Zheleznyakov, D., Horrocks, I., Ioannidis, Y. E., & Möller, R. (2016). Ontology-based integration of streaming and static relational data with optique. In *Proc. of the 2016 Int. Conf. on Management of Data, SIGMOD Conference 2016*, pp. 2109–2112.

Kharlamov, E., Mailis, T., Mehdi, G., Neuenstadt, C., Özçep, Ö. L., Roshchin, M., Solomakhina, N., Soylu, A., Svingos, C., Brandt, S., Giese, M., Ioannidis, Y. E., Lamparter, S., Möller, R., Kotidis, Y., & Waaler, A. (2017). Semantic access to streaming and static data at siemens. *J. Web Sem.*, *44*, 54–74.

Klarman, S., & Meyer, T. (2014). Querying temporal databases via OWL 2 QL. In *Proc. of the 8th Int. Conf. on Web Reasoning and Rule Systems, RR 2014*, Vol. 8741 of *LNCS*, pp. 92–107. Springer.

Kontchakov, R., Rezk, M., Rodriguez-Muro, M., Xiao, G., & Zakharyaschev, M. (2014). Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *Proc. of the 13th Int. Semantic Web Conf. (ISWC 2014), Part I*, Vol. 8796 of *LNCS*, pp. 552–567. Springer.

Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V., & Zakharyaschev, M. (2016). Temporal and spatial OBDA with many-dimensional halpern-shoham logic. In *IJCAI*, pp. 1160–1166. IJCAI/AAAI Press.

Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-Time Systems*, *2*(4), 255–299.

Ladner, R. E. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*.

Lutz, C., Wolter, F., & Zakharyaschev, M. (2008). Temporal description logics: A survey. In *Proc. of the 15th Int. Symposium on Temporal Representation and Reasoning (TIME 2008)*, pp. 3–14.

Madnani, K., Krishna, S. N., & Pandya, P. K. (2013). On the decidability and complexity of some fragments of Metric Temporal Logic. *CoRR*, *abs/1305.6137*.

Ouaknine, J., & Worrell, J. (2005). On the decidability of metric temporal logic. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, LICS '05, pp. 188–197, Washington, DC, USA. IEEE Computer Society.

Ouaknine, J., & Worrell, J. (2008). Some recent results in metric temporal logic. In *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, pp. 1–13.

Özçep, Ö., Möller, R., Neuenstadt, C., Zheleznyakov, D., & Kharlamov, E. (2013). A semantics for temporal and stream-based query answering in an OBDA context. Tech. rep., Deliverable D5.1, FP7-318338, EU.

Özçep, Ö. L., & Möller, R. (2014). Ontology based data access on temporal and streaming data. In *the 10th Int. Summer School on Reasoning Web, RW 2014*, Vol. 8714 of *LNCS*, pp. 279–312. Springer.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. *J. on Data Semantics*, *X*, 133–173.

Raheb, K. E., Mailis, T., Ryzhikov, V., Papapetrou, N., & Ioannidis, Y. E. (2017). Balonse: Temporal aspects of dance movement and its ontological representation. In *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part II*, pp. 49–64.

Rodriguez-Muro, M., Kontchakov, R., & Zakharyaschev, M. (2013). Ontology-based data access: Ontop of databases. In *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, pp. 558–573.

Sistla, A., & Clarke, E. (1985). The complexity of propositional linear temporal logics. *J. ACM*, *32*, 733–749.

Soylu, A., Giese, M., Jiménez-Ruiz, E., Vega-Gorgojo, G., & Horrocks, I. (2016). Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, *15*(1), 129–152.

Tobies, S. (2001). Pspace reasoning for graded modal logics. *Journal of Logic and Computation*, *11*(1), 85–106.

Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

Vardi, M. (1982). The complexity of relational query languages (extended abstract). In *Proc. of the 14th ACM SIGACT Symp. on Theory of Computing (STOC'82)*, pp. 137–146.

Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., & Zakharyaschev, M. (2018). Ontology-based data access: A survey. In *Proc. of the 28th Int. Joint Conf. on Artificial Intelligence (IJCAI)*. IJCAI/AAAI.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., & Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Commun. ACM*, *59*(11), 56–65.

Zhou, X., Wang, F., & Zaniolo, C. (2006). Efficient temporal coalescing query support in relational database systems. In *Proc. of the 17th Int. Conf. on Database and Expert Systems Applications, DEXA 2006*, Vol. 4080 of *LNCS*, pp. 676–686. Springer.

Appendix A.

Proof of Theorem 14

The formula $\sigma^{\langle m,n \rangle}_{\varrho,P,P_1,P_2}(x,y)$ is defined as follows:

$$\exists x_1, y_1, \dots, x_5, y_5 \bigvee_{\substack{m_1 \in \mathsf{le}(P_1) \\ n_1 \in \mathsf{ri}(P_1) \\ \lceil_1 \in \{[,(\},\ \rceil_1 \in \{],)\}}} \Big(\varphi^{\lceil_1 m_1, n_1 \rceil_1}_{P_1}(x_1, y_1) \wedge \bigvee_{\substack{m_2 \in \mathsf{le}(P_2) \\ n_2 \in \mathsf{ri}(P_2) \\ \lceil_2 \in \{[,(\},\ \rceil_2 \in \{],)\}}} \Big(\varphi^{\lceil_2 m_2, n_2 \rceil_2}_{P_2}(x_2, y_2) \wedge$$
$$\bigvee_{\substack{m_3 = m_1 \\ n_3 = n_1 \\ \lceil_3 \in \{[,(\},\ \rceil_3 \in \{],)\}}} \Big((x_3 = x_1) \wedge (y_3 = y_1) \wedge \mathsf{is}_{\lceil_3, [} \wedge \mathsf{is}_{\rceil_3,]} \wedge$$
$$\bigvee_{\substack{m_4 \in \mathsf{le}(P_1) \cup \mathsf{le}(P_2) \\ n_4 \in \mathsf{ri}(P_1) \cup \mathsf{ri}(P_2) \\ \lceil_4 \in \{[,(\},\ \rceil_4 \in \{],)\}}} \Big(\mathsf{inter}^{\lceil_4 m_4, n_4 \rceil_4}_{\lceil_2 m_2, n_2 \rceil_2, \lceil_3 m_3, n_3 \rceil_3}(x_4, y_4, x_2, y_2, x_3, y_3) \wedge$$
$$\bigvee_{\substack{m_4 \in \mathsf{le}(P) \\ n_4 \in \mathsf{ri}(P) \\ \lceil_5 \in \{[,(\},\ \rceil_5 \in \{],)\}}} \big(\mathsf{pluso}^{\lceil_5 m_5, n_5 \rceil_5}_{\varrho, \lceil_4 m_4, n_4 \rceil_4}(x_5, y_5, x_4, y_4) \wedge$$
$$\mathsf{inter}^{\langle m,n \rangle}_{\lceil_5 m_5, n_5 \rceil_5, \lceil_3 m_3, n_3 \rceil_3}(x, y, x_5, y_5, x_3, y_3) \big) \Big) \Big) \Big) \Big),$$

where $\mathsf{pluso}^{\lceil_5 m_5, n_5 \rceil_5}_{\varrho, \lceil_4 m_4, n_4 \rceil_4}(x_5, y_5, x_4, y_4)$ is an (obvious) formula saying that $\lceil_5 x_5 + m_5, y_5 + n_5 \rceil_5$ is the interval $\lceil_4 x_4 + m_4, y_4 + n_4 \rceil_4 +^o \varrho$.

The formula $x = y + c$, for a non-negative c, is defined as follows. For $c = \infty$, we take the formula

$$\forall j \, (\mathsf{bit}^{in}(x, j, 1) \wedge \mathsf{bit}^{fr}(x, j, 1)),$$

whereas for a constant $c = h/2^k$, we can use

$$\forall j \, \Big(\big(\mathsf{bit}^{in}(x, j, 0) \wedge \mathsf{bit}^{in}_{+h/2^k}(y, j, 0)\big) \vee \big(\mathsf{bit}^{in}(x, j, 1) \wedge \mathsf{bit}^{in}_{+h/2^k}(y, j, 1)\big) \Big) \wedge$$
$$\forall j \, \Big(\big(\mathsf{bit}^{fr}(x, j, 0) \wedge \mathsf{bit}^{fr}_{+h/2^k}(y, j, 0)\big) \vee \big(\mathsf{bit}^{fr}(x, j, 1) \wedge \mathsf{bit}^{fr}_{+h/2^k}(y, j, 1)\big) \Big),$$

where predicates $\mathsf{bit}^{in}_{+h/2^k}(y, j, v)$, saying that v is the j-th bit of the integer part of $y + h/2^k$, and $\mathsf{bit}^{fr}_{+h/2^k}(y, j, v)$, saying that v is the j-th bit of the fractional part of $y + h/2^k$, are defined inductively as follows:

$$\begin{aligned}
\mathsf{bit}^{fr}_{+0/2^k}(y,j,v) &= \mathsf{bit}^{fr}(y,j,v),\\
\mathsf{bit}^{fr}_{+(d+1/2^k)}(y,j,v) &= \exists u\Big((u=\ell-k)\wedge\Big(\big((j\le u)\wedge \mathsf{bit}^{fr}_{+d}(y,j,v)\big)\vee\\
&\qquad\big((v=0)\wedge \mathsf{bit}^{fr}_{+d}(y,j,0)\wedge\exists j'((u<j'<j)\wedge \mathsf{bit}^{fr}_{+d}(y,j',0))\big)\vee\\
&\qquad\big((v=0)\wedge \mathsf{bit}^{fr}_{+d}(y,j,1)\wedge\forall j'((u<j'<j)\rightarrow \mathsf{bit}^{fr}_{+d}(y,j',1))\big)\vee\\
&\qquad\big((v=1)\wedge \mathsf{bit}^{fr}_{+d}(y,j,1)\wedge\exists j'((u<j'<j)\wedge \mathsf{bit}^{fr}_{+d}(y,j',0))\big)\vee\\
&\qquad\big((v=1)\wedge \mathsf{bit}^{fr}_{+d}(y,j,0)\wedge\forall j'((u<j'<j)\rightarrow \mathsf{bit}^{fr}_{+d}(y,j',1))\big)\Big)\Big),
\end{aligned}$$

$$\begin{aligned}
\mathsf{bit}^{in}_{+0/2^k}(y,j,v) &= \mathsf{bit}^{in}(y,j,v),\\
\mathsf{bit}^{in}_{+(d+1/2^k)}(y,j,v) &= \exists u\Big((u=\ell-k)\wedge\Big(\\
&\qquad\big((v=0)\wedge \mathsf{bit}^{in}_{+d}(y,j,0)\wedge\exists j'(((j'<j)\wedge \mathsf{bit}^{in}_{+d}(y,j',0))\vee\\
&\qquad\qquad((u<j'<j)\wedge \mathsf{bit}^{fr}_{+d}(y,j',0)))\big)\vee\\
&\qquad\big((v=0)\wedge \mathsf{bit}^{in}_{+d}(y,j,1)\wedge\forall j'(((j'<j)\rightarrow \mathsf{bit}^{in}_{+d}(y,j',1))\wedge\\
&\qquad\qquad(u<j'<j)\rightarrow \mathsf{bit}^{fr}_{+d}(y,j',1))\big)\vee\\
&\qquad\big((v=1)\wedge \mathsf{bit}^{in}_{+d}(y,j,0)\wedge\exists j'(((j'<j)\wedge \mathsf{bit}^{in}_{+d}(y,j',0))\vee\\
&\qquad\qquad((u<j'<j)\wedge \mathsf{bit}^{fr}_{+d}(y,j',0)))\big)\vee\\
&\qquad\big((v=1)\wedge \mathsf{bit}^{in}_{+d}(y,j,1)\wedge\forall j'(((j'<j)\rightarrow \mathsf{bit}^{in}_{+d}(y,j',1))\wedge\\
&\qquad\qquad((u<j'<j)\rightarrow \mathsf{bit}^{fr}_{+d}(y,j',1)))\big)\Big)\Big).
\end{aligned}$$

Here, $u=\ell-k$ can be easily defined using $<$ and k.

Proofs of Lemmas 18 and 19

Lemma. *If T satisfies TOA, then a projection of T satisfying TOA can be computed in time $O(|T|_o^2\times |T|_t)$.*

Proof. We first partition T into a set of purely temporal tables $T_{c_1,\dots,c_m}$ and compute the set of all individual tuples $(c'_1,\dots,c'_n)$ that will appear in the projection T'. Let $(c'_1,\dots,c'_n)$ be one such tuple, and consider the tables $T_{c^1_1,\dots,c^1_m},\dots,T_{c^k_1,\dots,c^k_m}$ such that the projection of each $(c^i_1,\dots,c^i_m)$ is precisely $(c'_1,\dots,c'_n)$. Clearly, we have at most $|T|_o$ such tables. It is well-known that, for a pair of ordered tables S and S', we can construct an ordered table that contains all the tuples $S\cup S'$ in time $|S|+|S'|$. We use this algorithm k times to obtain an ordered table containing all the tuples of $T_{c^1_1,\dots,c^1_m}\cup\dots\cup T_{c^k_1,\dots,c^k_m}$ in time $O(k|T|_o)$. We then write the tuples of the form $(c'_1,\dots,c'_n,\langle,t_1,t_2,\rangle)$, where $(\langle,t_1,t_2,\rangle)$ is a tuple from the united table, into the output table. It can be readily checked that the complete output table can be produced in the required time. □

Lemma. *For any pair of tables T and T' satisfying TOA, their union table also satisfying TOA can be computed in time $O((|T|_o^2+|T'|_o^2)\times(|T|_t+|T'|_t))$.*

Proof. We first partition T and T' into sets of purely temporal tables $T_{c_1,\dots,c_m}$ and, respectively, $T'_{c_1,\dots,c_m}$. While doing this partition, we make sure that the tables $T_{c_1,\dots,c_m}$ are stored sequentially with respect to some order on the tuples $(c_1, \dots, c_m)$ (it can be done in time $|T|_o^2 \times |T|_t$). We do the same for the tables $T'_{c_1,\dots,c_m}$. It remains to go through all the tuples $\langle, t_1, t_2, \rangle$ and $\lceil, t'_1, t'_2, \rceil$ in all the tables $T_{c_1,\dots,c_m}$ and $T'_{c_1,\dots,c_m}$ to produce the union table by an algorithm similar to the one applied to the tables S and S' in the proof of Lemma 18. □

Experimental Results

(a) Number of the results returned from the Siemens queries.

queries \ # of months	32	64	96	128	159	191	223	255	287	320
ActivePowerTrip	324	648	970	1294	1618	1940	2264	2588	2912	3236
NormalStop	648	1296	1940	2588	3236	3880	4528	5176	5824	6472
NormalStart	162	324	485	647	809	970	1132	1294	1456	1618
NormalRestart	0	0	0	0	0	0	0	0	0	0

(b) Number of the results returned from the NY weather stations from 2005 to 2014.

queries \ # of months	32	64	96	128	159	191	223	255	287	320
ShoweryPatternCounty	530	1221	1802	2647	3609	4349	5204	5912	6639	7655
HurricaneAffectedState	2	4	5	5	5	8	9	801	1523	1533
HeatAffectedCounty	0	5	7	14	21	33	39	51	57	59
CyclonePatternState	914	1574	1617	1851	1936	2139	2246	2307	2333	2359

(c) Number of the results returned from the Weather data for 1–19 states in 2012.

queries \ # of months	32	64	96	128	159	191	223	255	287	320
ShoweryPatternCounty	3769	4481	4928	10349	12709	13681	14470	14933	16381	16883
HurricaneAffectedState	2	784	789	789	790	790	798	811	813	813
HeatAffectedCounty	53	65	81	84	88	98	100	117	142	224
CyclonePatternState	9109	9179	9593	17577	30203	38421	40769	43662	54199	56303

Table 7: Number of the results returned.

(a) Siemens data for one turbine.

	# of months	32	64	96	128	159	191	223	255	287	320
	# of rows	12,935,538	25,871,076	38,726,765	51,662,303	64,597,841	77,453,530	90,389,068	103,324,606	116,260,144	129,195,682
CSV	size (GB)	0.57	1.2	1.7	2.3	2.9	3.4	4.0	4.5	5.1	5.7
PostgreSQL	raw size (GB)	0.7	1.4	2.2	2.9	3.7	4.4	5.2	5.9	6.7	7.4
	total size (GB)	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
Parquet	size (GB)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

(b) NY weather stations from 2005 to 2014.

	# of years	1	2	3	4	5	6	7	8	9	10
	# of stations	229	306	370	441	484	542	595	643	807	874
	# of rows	3,969,455	10,959,978	18,614,686	26,622,218	35,862,560	49,115,307	63,469,733	79,032,846	99,221,419	124,001,260
CSV	size (GB)	0.2	0.6	1.1	1.6	2.1	2.9	3.8	4.8	5.9	7.4
PostgreSQL	raw size (GB)	0.3	0.8	1.4	2.0	2.7	3.7	4.9	6.1	7.7	11.0
	total size (GB)	0.4	1.1	2.0	2.9	3.9	5.4	7.1	8.9	11.0	14.0
Parquet	size (GB)	0.03	0.08	0.15	0.2	0.3	0.4	0.5	0.6	0.8	0.9

(c) Weather data for 1–19 states in 2012.

	states	DE, GA	+NY	+MD	+NJ, RI	+MA, CT	+LA, VT	+ME, WV	+NH, NC	+MS,SC, ND	+KY, SD
	# of states	2	3	4	6	8	10	12	14	17	19
	# of stations	408	659	1120	1476	1875	2305	2669	3019	3508	4037
	# of rows	16,760,333	32,470,116	41,346,986	51,610,908	66,842,618	80,561,273	92.550.905	106,415,139	121,216,837	140,517,500
CSV	size (GB)	0.9	1.9	2.5	3.1	4.0	4.8	5.5	6.4	7.2	8.3
PostgreSQL	raw size (GB)	1.2	2.4	3.1	3.9	5.1	6.1	7.1	8.1	9.2	10.0
	total size (GB)	2.0	4.1	5.3	6.5	8.6	10.0	12.0	14.0	16.0	18.0
Parquet	size (GB)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.1

- CSV: the size of the data in CSV format;
- PostgreSQL (raw size): the size of the data itself stored in PostgreSQL reported by the `pg_relation_size` function;
- PostgreSQL (total size): the size of the total data (including the index) stored in PostgreSQL reported by the `pg_total_relation_size` function;
- Parquet: the size of the data in Apache Parquet format, used by Apache Spark.

Table 8: The size of the data sets used in the experiments.