The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Reasoning over Streaming Data in Metric Temporal Datalog

Przemysław Andrzej Wałęga,[1,2] **Mark Kaminski,**[1] **Bernardo Cuenca Grau**[1]
[1]Department of Computer Science, University of Oxford, UK
[2]Institute of Philosophy, University of Warsaw, Poland
{przemyslaw.walega, mark.kaminski, bernardo.cuenca.grau}@cs.ox.ac.uk

Abstract

We study stream reasoning in $\mathit{datalogMTL}$—an extension of Datalog with metric temporal operators. We propose a sound and complete stream reasoning algorithm that is applicable to a fragment $\mathit{datalogMTL}^{\mathit{FP}}$ of $\mathit{datalogMTL}$, in which propagation of derived information towards past time points is precluded. Memory consumption in our algorithm depends both on the properties of the rule set and the input data stream; in particular, it depends on the distances between timestamps occurring in data. This is undesirable since these distances can be very small, in which case the algorithm may require large amounts of memory. To address this issue, we propose a second algorithm, where the size of the required memory becomes independent on the timestamps in the data at the expense of disallowing punctual intervals in the rule set. Finally, we provide tight bounds to the data complexity of standard query answering in $\mathit{datalogMTL}^{\mathit{FP}}$ without punctual intervals in rules, which yield a new PSPACE lower bound to the data complexity of the full $\mathit{datalogMTL}$.

Introduction

Decisions in industry increasingly depend on the analysis of large volumes of streaming data. Algorithmic trading requires real-time processing of stock tickers and news items (Nuti et al. 2011); oil companies monitor sensor readings to detect equipment malfunction and predict maintenance needs (Cosad et al. 2009); network providers analyse flow data to identify traffic anomalies (Münz and Carle 2007); and IoT applications such as Smart Cities analyse data coming from multiple devices (Lécué 2017).

Data analysis in these applications typically relies on the ability to recognise interesting events. For example, an analyst in a diagnostics centre could be interested in identifying whether a certain signal has been received with high frequency over a period of time. Metric Temporal Logic (MTL) has been proposed as a suitable formalism for specifying and reasoning over such complex events in real-time systems (Heintz and Doherty 2004; Thati and Roşu 2005; Doherty, Kvarnström, and Heintz 2009; Ničković and Piterman 2010; Baldor and Niu 2012; Ho, Ouaknine, and Worrell 2014; Basin, Klaedtke, and Zălinescu 2018). MTL allows for formulas such as $\boxminus_{[k_1,k_2]}\varphi$ and $\diamondminus_{[k_1,k_2]}\varphi$, with k_1 and k_2 rational numbers, which hold at time t if φ holds at each (respectively, some) moment in $[t-k_2, t-k_1]$. In particular, we can capture our previous example using an MTL formula $\boxminus_{[0,k_1]}\diamondminus_{[0,k_2]}\mathit{signal}$, which holds if the signal has been received continuously over an interval lasting at least k_1 with gaps between consecutive readings of length at most k_2.

MTL satisfiability checking, however, cannot be used to capture analysis tasks involving recursive propagation of information. Query answering in a recursive rule language with metric operators such as $\mathit{datalogMTL}$ (Brandt et al. 2018) is better suited to that effect.

Example 1. *Consider a network where nodes are monitoring different signals. A node receiving readings for a signal with sufficiently high frequency flags the signal by sending a message to all neighbouring nodes in the network, which will then also monitor the signal for a fixed period of time. The analyst is interested in tracking which nodes are monitoring which signals at any point in time. This generic monitoring task involves analysing how information propagates recursively over time throughout the topology of the network, and can be captured by the following* $\mathit{datalogMTL}$ *rules:*

$$\begin{aligned}\mathit{flags}(x,z) &\leftarrow \mathit{node}(x) \wedge \mathit{monitors}(x,z) \wedge \\ &\qquad \boxminus_{[0,k_1]}\diamondminus_{[0,k_2]}\mathit{signal}(z);\\ \boxplus_{[0,k]}\mathit{monitors}(x,z) &\leftarrow \mathit{connected}(x,y) \wedge \mathit{flags}(y,z).\end{aligned}$$

Here, the MTL formula $\boxminus_{[0,k_1]}\diamondminus_{[0,k_2]}\mathit{signal}(z)$ *is as before, whereas* $\boxplus_{[0,k]}\mathit{monitors}(x,z)$ *indicates that* x *will monitor signal* z *continuously for a period of time of length* k.

In contrast to the standard query answering problem, where input data is assumed to be finite, streaming data is seen as an unbounded sequence of facts that flow through the system and must be processed incrementally. *Stream reasoning*—the problem of answering queries over streaming data in the presence of background knowledge—has received a great deal of attention in the literature, and we refer the reader to our Related Work section for a detailed discussion (Barbieri et al. 2009; Anicic et al. 2011; Dell'Aglio et al. 2014; Özçep, Möller, and Neuenstadt 2014; Margara et al. 2014; Zaniolo 2012; Ronca et al. 2018b; Beck, Dao-Tran, and Eiter 2018). Stream reasoning in a recursive metric temporal language such as $\mathit{datalogMTL}$, however, has not been studied to the best of our knowledge, and raises a number of technical challenges discussed next.

To ensure correctness of query answering over streaming data, systems must compute results over received partial data as if the entire (infinite) stream had been available. Furthermore, to achieve low latency and satisfy memory restrictions, systems can only store a limited history of received information to perform further computations. Rules in *datalogMTL*, however, can propagate derived information both towards past and future time points and hence results can depend on data that has not yet been received or which arrived far in the past. This may force the system to keep in memory a large (or even unbounded) input history, and/or to delay the output of results (even indefinitely) to ensure correctness. This leads to a fundamental trade-off involving the basic properties that an algorithm must satisfy, namely *(i) soundness*—every computed answer follows logically from the rules and the entire infinite stream; *(ii) completeness*—each answer that follows from the rules and the infinite stream will eventually be streamed out; and *(iii) minimal storage and latency*—answers are streamed out as early as possible, and stored information is kept to a minimum.

Our contribution We investigate the problems surrounding stream reasoning for *datalogMTL*. In line with other works (Baldor and Niu 2012; Ronca et al. 2018a; Basin, Klaedtke, and Zălinescu 2018), we focus on a fragment that precludes propagation of derived information towards the past, which we call $datalogMTL^{FP}$. This fragment is motivated by the observation that received data in streaming applications typically influences only present and future events (e.g., a flag is raised only after a pattern in signal data has been detected).

We propose a sound and complete stream reasoning algorithm for $datalogMTL^{FP}$ that relies on a sliding window to limit memory usage. The analysis of our algorithm, however, reveals that memory usage not only depends on the rule set defining the query and the number of different objects mentioned in the stream, but also on the distances between timestamps in the stream. This is undesirable since the minimum distance between consecutive time points in a stream can be very small (i.e., bounded by the precision of the measuring devices) and, in particular, orders of magnitude smaller than the length of intervals in rules. As a result, the memory consumption of our algorithm may be too large in practice. To address this issue, we propose a modified algorithm, where memory consumption no longer depends on the distance between consecutive facts, at the expense of disallowing punctual operators such as $\diamondminus_{[1,1]}$ in rules.

We finally study the data complexity of standard query answering, which is in known to be in EXPSPACE and P-hard for *datalogMTL* (Brandt et al. 2018). We show tight P and PSPACE bounds for $datalogMTL^{FP}$ rule sets without punctual intervals under the assumption that numbers are encoded in unary and binary, respectively. These results yield a new PSPACE lower bound for the data complexity of the full *datalogMTL* language under binary encoding.

Preliminaries

We consider intervals over the non-negative rationals $\mathbb{Q}_{\geq 0}$ and ∞, and denote with ϱ^- the greatest lower bound (glb) and with ϱ^+ the least upper bound (lub) of interval ϱ, writing $\varrho^+ = \infty$ when ϱ is unbounded to the right; ϱ is left closed if $\varrho^- \in \varrho$, and right closed if $\varrho^+ \in \varrho$. The length $|\varrho|$ of ϱ is ∞ if $\varrho^+ = \infty$ and otherwise $|\varrho| = \varrho^+ - \varrho^-$; an interval $[t, t]$ of length 0 is punctual, and we denote it as t. For intervals ϱ_1 and ϱ_2, we define their union $\varrho_1 \cup \varrho_2$ and intersection $\varrho_1 \cap \varrho_2$ as usual; $\varrho_1 + \varrho_2$ is the interval with glb ${\varrho_1}^- + {\varrho_2}^-$ and lub ${\varrho_1}^+ + {\varrho_2}^+$ that is left (right) closed if and only if both ϱ_1 and ϱ_2 are left (right) closed. We define $\varrho_1 \oplus \varrho_2$ as the interval with glb ${\varrho_1}^- + {\varrho_2}^+$ and lub ${\varrho_1}^+ + {\varrho_2}^-$ that is left closed if and only if ϱ_1 is left closed and ϱ_2 is right closed, and that is right closed if and only if ϱ_1 is right closed and ϱ_2 is left closed; if such interval does not exist, then $\varrho_1 \oplus \varrho_2 = \emptyset$. We assume that each $t \in \mathbb{Q}$ is given by two integers (numerator and denominator), and say that t is encoded in unary (binary) if both integers are represented in unary (binary). A set $X \subseteq \mathbb{Q}$ is discrete if for each $t \in X$ that is not largest in X there is $t' > t$ in X such that there is no $t'' \in X$ with $t < t'' < t'$. For a discrete X and any $t \in \mathbb{Q}$ we define $\mathsf{succ}_X(t)$ as the least $t' \in X$ such that $t' > t$, or ∞ if no such t' exists. Finally, the greatest common divisor of X, written $\mathsf{gcd}(X)$, is the greatest rational number which divides all numbers in X to integer values.

Next, we introduce *datalogMTL*. As in (Brandt et al. 2018), we adopt the so-called continuous semantics of MTL as opposed to the pointwise semantics.

Syntax We assume countably infinite and pairwise disjoint sets of variables, constants, and predicates with a nonnegative integer arity. A term is a variable or a constant. An atom is of the form $P(\boldsymbol{\tau})$ with P a predicate symbol and $\boldsymbol{\tau}$ a tuple of terms of matching arity; an atom is ground if it contains no variables. A rule is an expression of the following form:

$$B \leftarrow A_1 \land \ldots \land A_k \qquad (k \geq 0), \tag{1}$$

where its elements are defined by the following grammar:

$$A := P(\boldsymbol{\tau}) \mid \boxplus_\varrho A \mid \boxminus_\varrho A \mid \diamondplus_\varrho A \mid \diamondminus_\varrho A \mid A\mathcal{U}_\varrho A \mid A\mathcal{S}_\varrho A;$$
$$B := P(\boldsymbol{\tau}) \mid \boxplus_\varrho B \mid \boxminus_\varrho B.$$

We assume that rules are safe: each variable occurs in some A_i. A *datalogMTL* program is a finite set Π of rules. Program Π is *datalogMITL* if it has no punctual intervals. The granularity $\mathsf{gran}(\Pi)$ of Π is the minimal length of the intervals in Π. A fact is of the form $\alpha@t$ with α a ground atom and $t \in \mathbb{Q}_{\geq 0}$. A dataset $\mathcal{D}$ is a (possibly infinite) set of facts.

Semantics An interpretation $\mathfrak{M}$ is based on a non-empty domain Δ; it determines, for each ground atom α and $t \in \mathbb{Q}$, whether α holds at t (in which case, we write $\mathfrak{M}, t \models \alpha$). Let ν map terms into Δ such that $\nu(c) = c$ for each constant c; then, satisfaction of complex expressions by $\mathfrak{M}$ is defined inductively in Table 1. Interpretation $\mathfrak{M}$ satisfies rule (1) if, for each ν, $\mathfrak{M}, t \models^\nu B$ whenever $\mathfrak{M}, t \models^\nu A_i$ for each $1 \leq i \leq k$. Interpretation $\mathfrak{M}$ is a model of program Π if it satisfies all its rules, and is a model of dataset $\mathcal{D}$ if $\mathfrak{M}, t \models \alpha$ for each $\alpha@t \in \mathcal{D}$. We say that Π and $\mathcal{D}$ entail $\alpha@t$, written $(\Pi, \mathcal{D}) \models \alpha@t$, if $\mathfrak{M}, t \models \alpha$ for each model $\mathfrak{M}$ of Π and $\mathcal{D}$.

Streams and Stream Queries

Streaming data is often represented as an unbounded sequence of timestamped facts. As in (Brandt et al. 2018), we

$\mathfrak{M}, t \models^\nu P(\boldsymbol{\tau})$	if $\mathfrak{M}, t \models P(\nu(\boldsymbol{\tau}))$;	
$\mathfrak{M}, t \models^\nu \boxplus_\varrho A$	if $\mathfrak{M}, s \models^\nu A$ for all s with $s - t \in \varrho$;	
$\mathfrak{M}, t \models^\nu \boxminus_\varrho A$	if $\mathfrak{M}, s \models^\nu A$ for all s with $t - s \in \varrho$;	
$\mathfrak{M}, t \models^\nu \diamondplus_\varrho A$	if $\mathfrak{M}, s \models^\nu A$ for some s with $s - t \in \varrho$;	
$\mathfrak{M}, t \models^\nu \diamondminus_\varrho A$	if $\mathfrak{M}, s \models^\nu A$ for some s with $t - s \in \varrho$;	
$\mathfrak{M}, t \models^\nu A \mathcal{U}_\varrho A'$	if $\mathfrak{M}, t' \models^\nu A'$ for some t' with $t' - t \in \varrho$	
	and $\mathfrak{M}, s \models^\nu A$ for all $s \in (t, t')$;	
$\mathfrak{M}, t \models^\nu A \mathcal{S}_\varrho A'$	if $\mathfrak{M}, t' \models^\nu A'$ for some t' with $t - t' \in \varrho$	
	and $\mathfrak{M}, s \models^\nu A$ for all $s \in (t', t)$.	

Table 1: Semantics of datalogMTL.

represent timestamps as non-negative rational numbers; this generalises other models of streaming data in the literature where timestamps are assumed to be natural numbers (Beck, Dao-Tran, and Eiter 2018; Ronca et al. 2018b). We also assume that the timeline of a stream is discrete as opposed to dense—that is, it is always possible to tell which is the next time point for which data is available in the stream.

Definition 2. A *stream* is a function S assigning a finite set $S(t)$ of ground atoms to each $t \in \mathbb{Q}_{\geq 0}$. The *object domain* of S is the set of constants $\mathcal{O}_S$ occurring in S; the *temporal domain* of S is the set of rationals $\mathcal{T}_S$ to which S assigns a non-empty set. For any stream S we require $\mathcal{T}_S$ to be a discrete set. By slight abuse of notation, we make no distinction between S and the (maybe infinite) dataset $\{\alpha@t \mid \alpha \in S(t)\}$.

A query defines a transformation of an input stream into an output stream by means of logical entailment.

Definition 3. A *stream query* is a pair (Π, Q), with Π a program and Q a predicate. Let $X \subseteq \mathbb{Q}_{\geq 0}$ be a discrete set. The *answer* to (Π, Q) over a stream I relative to X is the stream $\{Q(\mathbf{c})@t \mid t \in X \text{ and } (\Pi, I) \models Q(\mathbf{c})@t\}$.

Note that set X in the definition specifies the temporal domain of the answer stream—that is, the timepoints for which answers are required in the output.

Forward-propagating datalogMTL

We next introduce the language of forward-propagating *datalogMTL*—a fragment of *datalogMTL* that allows for recursive derivation of facts into present and future timepoints, while precluding propagation towards past points. Similar restrictions have been adopted in the stream reasoning (Ronca et al. 2018a), monitoring (Basin, Klaedtke, and Zălinescu 2018) and database view update literature (Chomicki 1995). The fragment is motivated by the observation that events in streaming applications are typically recognised based on previously received data only (and not based on future data). From an algorithmic perspective, this restriction will allow us to limit memory consumption by "forgetting" previously received facts without losing completeness.

Algorithm 1 A generic stream reasoning algorithm.

```
Parameters: Query $(\Pi, Q)$, discrete $X \subseteq \mathbb{Q}_{\geq 0}$, $step \in \mathbb{Q}_{>0}$
Input: Stream $I$
 1: $t := -step$                                          ▷ current time point
 2: Set $w$ to the maximal number in $\Pi$                 ▷ window size
 3: $W := \emptyset$ and $H := \emptyset$                  ▷ memory
 4: loop:
 5:     if $\mathsf{succ}_I(t) \leq t + step$ then
 6:         $t_{next} := \mathsf{succ}_I(t)$ and $W := W \cup I(t_{next})$
 7:     else
 8:         $t_{next} := t + step$
 9:     Exhaustively apply the rules from Table 2
10:     for $Q(\mathbf{c})@\varrho \in W$ do
11:         if $t \in \varrho \cap (t, t_{next}] \cap X$ then
12:             stream out $Q(\mathbf{c})@t$
13:     Exhaustively apply the rules from Table 3
14:     $t := t_{next}$
```

Definition 4. A rule (1) is *forward-propagating* if its elements are generated by the following grammar:

$$A := P(\boldsymbol{\tau}) \mid \boxminus_\varrho A \mid \diamondminus_\varrho A; \qquad B := P(\boldsymbol{\tau}) \mid \boxplus_\varrho B.$$

A program is *datalogMTL*^{FP} if it consists of forward-propagating rules and it is *datalogMITL*^{FP} if, additionally, it contains no punctual intervals.[1]

Note that the program given in Example 1 is forward-propagating. For convenience, we will assume w.l.o.g. that *datalogMTL*^{FP} are given in the normal form defined next, where there is no nesting of temporal operators.

Definition 5. A *datalogMTL*^{FP} program is in *normal form* if it contains only rules of the following forms:

$$P_0(\boldsymbol{\tau}_0) \leftarrow \diamondminus_\varrho P_1(\boldsymbol{\tau}_1); \tag{2}$$

$$P_0(\boldsymbol{\tau}_0) \leftarrow \boxminus_\varrho P_1(\boldsymbol{\tau}_1); \tag{3}$$

$$P_0(\boldsymbol{\tau}_0) \leftarrow P_1(\boldsymbol{\tau}_1) \wedge \ldots \wedge P_n(\boldsymbol{\tau}_n) \qquad (n \geq 0), \tag{4}$$

where $P_i(\boldsymbol{\tau}_i)$ are atoms and each interval is either left and right closed, or left and right open.

Proposition 6. *Each* *datalogMTL*^{FP} *program* Π *can be transformed in polynomial time into* Π' *in normal form such that for each dataset* $\mathcal{D}$ *and fact* $\alpha@t$ *over the vocabulary of* Π*:* $(\Pi, D) \models \alpha@t$ *if and only if* $(\Pi', D) \models \alpha@t$*.*

A Generic Stream Reasoning Algorithm

In this section, we present a stream reasoning algorithm for *datalogMTL*^{FP}. Our algorithm takes as input a stream I and outputs (also as a stream) the answers to a standing query (Π, Q), which we fix as a parameter. The algorithm is also parametrised with a discrete set X, which specifies the temporal domain of the answer stream, and a value *step* which controls the incrementality of query processing.

The algorithm is initialised in Lines 1–3. Line 1 initialises the pointer to the current time point in the input stream to a value smaller than all time points in $\mathcal{T}_I$. In Line 2, we define the size of a *sliding window* w as the maximal rational

[1] For simplicity, we omit from the language the operator $\mathcal{S}_\varrho$.

Table 2: Derivation rules.

($\diamondminus$) **If** $\alpha@\varrho_1 \in W$ and $\beta \leftarrow \diamondminus_{\varrho_2}\alpha$ is a ground instance of a rule in Π and $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$ is non-empty, **then** add $\beta@\varrho$ to W;

($\diamondminus^{\infty}$) **If** $\alpha \in H$ and $\beta \leftarrow \diamondminus_{\varrho_2}\alpha$ is a ground instance of a rule in Π with $\varrho_2^+ = \infty$, **then** add $\beta@(t, t_{next}]$ to W;

($\boxminus$) **If** $\alpha@\varrho_1 \in W$, $\beta \leftarrow \boxminus_{\varrho_2}\alpha$ is a ground instance of a rule in Π, and $\varrho = (\varrho_1 \oplus \varrho_2) \cap (t, t_{next}]$ is non-empty **then** add $\beta@\varrho$ to W;

($horn$) **If** $\beta \leftarrow \bigwedge_{i=1}^{n} \alpha_i$ is a ground instance of a rule in Π, for each i $\alpha_i@\varrho_i \in W$, and $\varrho = \bigcap_{i=1}^{n} \varrho_i$ is non-empty, **then** add $\beta@\varrho$ to W;

($\cup$) **If** $\{\alpha@\varrho_1, \alpha@\varrho_2\} \subseteq W$ and $\varrho = \varrho_1 \cup \varrho_2$ is an interval, **then** add $\alpha@\varrho$ to W;

(H) **If** $\alpha@\varrho \in W$ and $\varrho^- < t - w$, **then** add α to H.

number occurring as a bound of an interval in Π. Line 3 initialises the memory, which we partition into two parts:

- The *window* W stores information about the most recent w time points; it consists of *extended facts* of the form $\alpha@\varrho$ with α a ground atom and ϱ an interval, meaning that α holds in all rational numbers in ϱ.
- The *history* H stores information about older time points that are not covered by W; it consists of ground atoms α, meaning that α was true at some such older time point, without specifying exactly when.

The core of the algorithm is an infinite loop, where each iteration consists of the steps detailed next and where the current time point t is incremented at the end of each iteration. In Lines 5–8, the algorithm reads the next chunk of the input by either moving forward by $step$ (if there is no data point between t and $t + step$), or by moving to the next time point with data and loading the relevant facts into the window memory (Line 6). In Line 9, the algorithm applies exhaustively the rules in Table 2, which derive new information based on the rules in Π and the contents of the memory:

- Rules ($\diamondminus$), ($\boxminus$), and ($horn$) capture the semantics of the normalised $datalogMTL^{FP}$ rules of the form (2), (3), and (4), respectively, when applied to the contents of W; the application of rules ($\diamondminus$) and ($\boxminus$) is illustrated in Figure 1.
- Rule ($\diamondminus^{\infty}$) deals with normalised rules of the form (2) when applicable to "old" facts in the history H.
- Rule ($\cup$) merges extended facts in W about the same ground atom into an extended fact over a larger interval.
- Rule (H) extends the history with old information that is outside the window.

In Lines 10–12 the algorithm outputs all answers in the answer stream in-between t and t_{next}. Finally, in Line 13 the algorithm "compacts" the window memory W by exhaustively applying the rules in Table 3. In particular, Rule ($c1$) slides the window by deleting from W old information that falls outside the window; Rule ($c2$) eliminates redundancy by deleting from W all extended facts subsumed by others.

Table 3: Memory compacting rules.

($c1$) **If** $\alpha@\varrho \in W$ and $\varrho^- < t - w$, **then** delete $\alpha@\varrho$ from W. Moreover, **if** $\varrho' = \varrho \cap [t - w, t_{next}]$ is non-empty, **then** add $\alpha@\varrho'$ to W;

($c2$) **If** $\{\alpha@\varrho_1, \alpha@\varrho_2\} \subseteq W$ and $\varrho_1 \subsetneq \varrho_2$, **then** delete $\alpha@\varrho_1$ from W.

The following theorem establishes soundness and completeness of our algorithm.

Theorem 7. *Algorithm 1 outputs the answer to* (Π, Q) *over* I *relative to* X *for any value of the step parameter.*

Proof sketch. The derivation rules from Table 2 mimic the semantics of rules in a normalised $datalogMTL^{FP}$ program; as a result, it can be shown that in each iteration of Algorithm 1, $\alpha@\varrho \in W$ implies $(\Pi, I) \models \alpha@t'$ for all $t' \in \varrho$, and $\alpha \in H$ implies $(\Pi, I) \models \alpha@t'$ for some $t' < t - w$. Thus, $(\Pi, I) \models Q(\mathbf{c})@t$ whenever $Q(\mathbf{c})@t$ is streamed out, which implies soundness of the algorithm.

To show completeness, we prove that the algorithm never deletes from its memory information that may be used to derive new facts in future. In particular, if the algorithm has derived α in t' in some iteration, then in each further iteration we have $\alpha@\varrho \in W$ for some ϱ including t' (if t' is inside the window) or $\alpha \in H$ (if t' sticks out of the window). Then, we use the $datalogMTL$ fixpoint characterisation from (Brandt et al. 2018) and show by induction on the construction of the fixpoint of Π on the input stream I that, whenever a fact $\alpha@t'$ is in the fixpoint, memory W contains $\alpha@\varrho$ for some interval ϱ including t' in the iteration of the loop of Algorithm 1 for which $t' \in (t, t_{next}]$. □

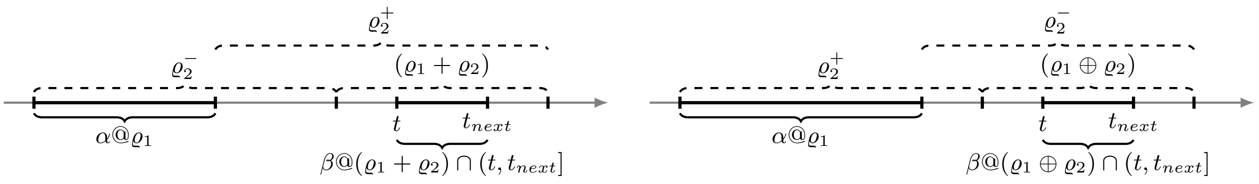
We next analyse the memory requirements of the algorithm in terms of the properties of the query (Π, Q), the input stream I, and the $step$ parameter.

Theorem 8. *Let* I *be a stream such that* $\mathsf{gcd}(\mathcal{T}_I)$ *exists. Then throughout each iteration of Algorithm 1, the number of elements in* $W \cup H$ *is at most*

$$\left(4 \cdot \left(\frac{w + 2 \cdot step}{\mathsf{gcd}(\mathcal{T}_I \cup \mathsf{N} \cup \{step\})} + 1\right)^2 + 1\right) \cdot \mathsf{P} \cdot |\mathcal{O}_I|^{\mathsf{A}},$$

with N *the set of rationals mentioned in* Π*,* P *the number of predicates in* Π *and* A *the maximum arity of such predicates.*

Proof sketch. Since each element of H is a ground atom, the cardinality of H is bounded by $\mathsf{P} \cdot |\mathcal{O}_I|^{\mathsf{A}}$. Similarly, for a given interval ϱ, there are at most $\mathsf{P} \cdot |\mathcal{O}_I|^{\mathsf{A}}$ extended facts of the form $\alpha@\varrho$ in W. Thus, it suffices to determine the number of intervals that may occur in W. We show by induction on the construction of W that $\alpha@\varrho \in W$ implies that ϱ^- and ϱ^+ divided by $\mathsf{gcd}(\mathcal{T}_I \cup \mathsf{N} \cup \{step\})$ yield integer values and that $\varrho \subseteq [t - step - w, t + step]$. As a result, there


ϱ_2^+
ϱ_2^-
$(\varrho_1 + \varrho_2)$
t
t_{next}
$\alpha@\varrho_1$
$\beta@(\varrho_1 + \varrho_2) \cap (t, t_{next}]$
ϱ_2^-
ϱ_2^+
$(\varrho_1 \oplus \varrho_2)$
t
t_{next}
$\alpha@\varrho_1$
$\beta@(\varrho_1 \oplus \varrho_2) \cap (t, t_{next}]$

(a)

(b)

Figure 1: Application of rules ($\diamondminus$) and ($\boxminus$), namely in (a) $\beta@(\varrho_1 + \varrho_2) \cap (t, t_{next}]$ is added to W by $\beta \leftarrow \diamondminus_{\varrho_2}\alpha$, whereas in (b) $\beta@(\varrho_1 \oplus \varrho_2) \cap (t, t_{next}]$ is added to W by $\beta \leftarrow \boxminus_{\varrho_2}\alpha$.

are $\frac{w+2\cdot step}{\mathsf{gcd}(\mathcal{T}_I \cup \mathsf{N} \cup \{step\})} + 1$ possible endpoints of ϱ, and thus ϱ is one of at most $4 \cdot (\frac{w+2\cdot step}{\mathsf{gcd}(\mathcal{T}_I \cup \mathsf{N} \cup \{step\})} + 1)^2$ intervals, where factor 4 takes into account whether the interval is left/right open or closed. Then, the bound in the theorem follows. □

We emphasise that Theorem 8 does not provide a finite bound on the number of elements in memory, even if the stream query and the step parameter are considered fixed. In particular, the algorithm may consume unbounded memory if the input stream I satisfies any of the following conditions: *(1)* the object domain $\mathcal{O}_I$ has infinitely many elements; or *(2)* $\mathcal{T}_I$ has no gcd. In many applications, it is reasonable to assume that these conditions do not hold—that is, the object domain of all streams is finite (e.g., the set of nodes and monitored signals in our example is finite over time) and that the timestamps occurring in a stream have a fixed gcd (e.g., the devices timestamping the data have a finite precision).

The gcd of the input stream however, can be very small compared to the size of the window, resulting in an impractically large bound on the number of elements in memory; furthermore, it may not be known to the designer of the query, which makes it difficult to estimate memory efficiency of the algorithm at design time. It is hence desirable to have an algorithm for which the size of the window and the history can be bounded independently of the temporal domain of the input stream. In the next section, we modify our algorithm for $\mathit{datalogMITL}^{FP}$ to a stream reasoning algorithm for $\mathit{datalogMITL}^{FP}$ that enjoys this desirable property.

An Algorithm for datalogMITL

We next present a variant of Algorithm 1 for which memory usage can be bounded independently of the temporal domain of the input streams whenever the query (Π, Q) is given in $\mathit{datalogMITL}^{FP}$ and hence has no punctual intervals. The only modification required to Algorithm 1 is an extension of the derivation and memory compacting rules from Tables 2 and 3; this will also require that we keep a new type of extended fact in the window memory.

Our modifications exploit the following observations:

1. Extended facts in W holding at "short" intervals cannot be used to derive new information using Rule ($\boxminus$) from Table 2. In particular, $\alpha@\varrho'$ in W cannot be used to apply ($\boxminus$) to a ground rule $\beta \leftarrow \boxminus_\varrho\alpha$ where $|\varrho'| < |\varrho|$.
2. If a ground atom α holds in W in a set S of intervals the gaps between which are all smaller than the granularity of Π, then to correctly derive new information with ground rules $\beta \leftarrow \diamondminus_\varrho\alpha$, it suffices to keep in memory the extended fact $\alpha@\varrho$ for the minimal ϱ containing all intervals in S. Note that this condition only makes sense if Π contains no punctual intervals, as otherwise $\mathit{gran}(\Pi) = 0$.

Table 4: Derivation rules for $\mathit{datalogMITL}^{FP}$

Rules ($\diamondminus^\infty$), ($\boxminus$), (horn), and ($\cup$) from Table 2 and the following rules:

($\#\diamondminus$) If $\alpha\#\varrho_1 \in W$, $\beta \leftarrow \diamondminus_{\varrho_2}\alpha$ is a ground instance of a rule in Π, and $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$ is non-empty, **then** add $\beta@\varrho$ to W;

($\#\cup$) If $\{\alpha\#\varrho_1, \alpha\#\varrho_2\} \subseteq W$ where $\varrho_1 \cup \varrho_2$ is an interval or $\varrho_2^- \geq \varrho_1^+$ and $\varrho_2^- - \varrho_1^+ < \mathsf{gran}(\Pi)$, **then** add $\alpha\#\varrho$ to W where ϱ is the smallest interval containing ϱ_1 and ϱ_2;

($\#H$) If $\alpha\#\varrho \in W$ and $\varrho^- < t - w$, **then** add α to H;

($@ \rightarrow \#$) If $\alpha@\varrho \in W$, **then** add $\alpha\#\varrho$ to W.

To capture the second observation, we allow in the window memory W a new type of extended fact of the form $\alpha\#\varrho$, with α a ground atom and ϱ an interval, meaning that α holds "sufficiently often" in ϱ as represented in Figure 2—that is, α holds in the endpoints of ϱ and at some point of each subinterval of ϱ of length at least $\mathit{gran}(\Pi)$.[2]

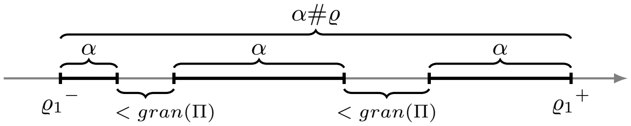


Figure 2: $\alpha\#\varrho$ means that α holds "sufficiently often" in ϱ.

The new derivation rules of the algorithm are given in Table 4. In order to capture the semantics of extended facts of the form $\alpha\#\varrho$, Rule ($\diamondminus$) in Table 2 is replaced with Rule ($\#\diamondminus$) and new rules ($\#\cup$), ($\#H$), and ($@ \rightarrow \#$) are added.

[2] The precise meaning of $\alpha\#\varrho$ is slightly more complicated, as it needs to cover the cases when ϱ is an open interval and when it was produced by Rule ($c3$) from Table 5.

Table 5: Memory compacting rules for $\mathit{datalogMITL}^{FP}$.

Rules from Table 3 and the following rules:

($c3$) **If** $\alpha\#\varrho \in W$ and $\varrho^- < t - w$, **then** delete $\alpha\#\varrho$ from W. Moreover, **if** $\varrho' = \varrho \cap [t - w, t_{next}]$ is non-empty, **then** add $\alpha\#\varrho'$ to W;

($c4$) **If** $\{\alpha\#\varrho_1, \alpha\#\varrho_2\} \subseteq W$ with $\varrho_1 \subsetneq \varrho_2$, **then** delete $\alpha\#\varrho_1$ from W;

($c5$) **If** $\alpha@\varrho \in W$, $\varrho^+ < t_{next}$, and $|\varrho| < \mathsf{gran}(\Pi)$ **then** delete $\alpha@\varrho$ from W.

The memory compacting rules in Table 3 are extended with those in Table 5. Rules ($c3$) and ($c4$) are the analogue to ($c1$) and ($c2$) in Table 3 for the new type of extended fact; in turn, Rule ($c5$) captures our first observation; it deletes from memory all extended facts $\alpha@\varrho$ where ϱ is too short since they cannot derive new information. Correctness of our modified algorithm is established in the following theorem.

Theorem 9. *Consider the variant of Algorithm 1 using the derivation rules from Table 4 and the compacting rules from Table 5. The algorithm outputs the answer to* (Π, Q) *over* I *relative to* X *for any value of the step parameter.*

Proof sketch. To show soundness we prove by simultaneous induction on the iterations of the algorithm that the two conditions from the soundness proof in Theorem 7 hold together with the following condition: if $\alpha\#\varrho \in W$, then each subinterval of ϱ of length at least $\mathsf{gran}(\Pi)$, includes a time point in which α holds. Hence, if $Q(\mathbf{c})@t$ is streamed out, we have $(\Pi, I) \models Q(\mathbf{c})@t$, so the algorithm is sound.

For completeness, we show that if α was derived in t' in some iteration of the algorithm, then in each further iteration we have: $\alpha\#\varrho \in W$ for some ϱ including t' (if t' is inside the window) or $\alpha \in H$ (if t' sticks out of the window). Moreover, if α was derived in ϱ in some iteration and $|\varrho| \geq \mathsf{gran}(\Pi)$, then in each further iteration for which ϱ^- is still in the window, we have $\alpha@\varrho' \in W$ for some ϱ' containing ϱ. Then, we show by induction on the construction of the fixpoint of Π on the input stream I that if a fact $\alpha@t'$ is in the fixpoint, then the memory W contains $\alpha@\varrho$ for some ϱ including t' in the iteration of the loop of Algorithm 1 for which $t' \in (t, t_{next}]$. Hence, the algorithm is complete. □

Finally, we obtain a bound on memory size that no longer depends on the temporal domain of input streams.

Theorem 10. *Consider the variant of Algorithm 1 using the derivation rules from Table 4 and the compacting rules from Table 5. For* Π *a* $\mathit{datalogMITL}^{FP}$ *program mentioning at least one interval,*[3] *in any step of the algorithm the number of elements in* $W \cup H$ *is at most*

$$4\left(\left(2\mathsf{G}\left(\frac{w + step}{\mathsf{gran}(\Pi)} + 1\right) + 2\right) \cdot \mathsf{R}^{\frac{w+2\cdot step}{\mathsf{K}}}\right)^2 + \mathsf{G},$$

where R *is the number of rules in* Π*,* K *is the least non-zero glb or lub of an interval and* $\mathsf{G} = \mathsf{P} \cdot |\mathcal{O}_I|^{\mathsf{A}}$ *for* P *the number, and* A *the maximum arity of predicates occurring in* Π*.*

Proof sketch. We first determine the size of W after compacting memory in Line 13. Consider $\alpha@\varrho$ in W; from ($c1$) it follows that $\varrho \subseteq [t - w, t_{next}]$ and, by ($c5$), $|\varrho| \geq \mathsf{gran}(\Pi)$ or $\varrho^+ = t_{next}$, which implies, by ($\cup$) and ($c2$), that W contains at most $\frac{w+step}{\mathsf{gran}(\Pi)}$ such facts. Next, consider $\alpha\#\varrho$ in W; it follows from ($c3$) that $\varrho \subseteq [t - w, t_{next}]$ and it follows from ($\#\cup$) and ($c4$) that the minimal distance between ϱ and any $\varrho' \neq \varrho$ with $\alpha'\#\varrho'$ in W is at least $\mathsf{gran}(\Pi)$. Hence, there are at most $\frac{w+step}{\mathsf{gran}(\Pi)} + 1$ elements $\alpha\#\varrho$ in W. Thus, W has at most $W_c = 2\mathsf{G}(\frac{w+step}{\mathsf{gran}(\Pi)} + 1)$ elements after Line 13.

Next, we analyse the number of elements in $W \cup H$ after applying derivation rules in Line 9. Set H contains at most G ground atoms. Assume that $\alpha@\varrho$ or $\alpha\#\varrho$ was added to W in Line 9; then, $\varrho \subseteq [t - w - step, t + step]$, and by analysing the rules in Table 5 we can show that each of ϱ^- and ϱ^+ can take at most $(W_c + 2) \cdot \mathsf{R}^{\frac{w+2\cdot step}{\mathsf{K}}}$ different values, which implies the bound from the theorem. □

Query Answering

We now turn our attention to the data complexity of standard query answering (i.e., over finite datasets). We first establish tight P and PSPACE bounds for $\mathit{datalogMITL}^{FP}$ under the assumption that numbers in the input are encoded in unary and binary, respectively. The upper bounds are obtained by applying our algorithm in the previous section by considering the input dataset as a finite stream.

Theorem 11. *Checking whether* $(\Pi, \mathcal{D}) \models \alpha@t$ *for* Π *a fixed* $\mathit{datalogMITL}^{FP}$ *program,* $\mathcal{D}$ *an input dataset, and* $\alpha@t$ *an input fact is* (i) PSPACE*-complete if time points in the input are encoded in binary; and* (ii) P*-complete if time points in the input are encoded in unary.*

Proof sketch. We can check $(\Pi, \mathcal{D}) \models \alpha@t$ using a straightforward adaptation of the algorithm for $\mathit{datalogMITL}^{FP}$ in the previous section: simply use $\mathcal{D}$ as a finite stream and terminate whenever the algorithm reaches time point t. The rules from Table 4 and Table 5 can be exhaustively applied in polynomial time in each iteration if Π is fixed; furthermore, the number of iterations is linear in the input under unary encoding, giving the P upper bound. Although the number of iterations is exponential under binary encoding, the size of the memory kept by the algorithm is polynomial if Π is fixed, so the PSPACE bound follows. The P lower bound follows from P-completeness of fact entailment in plain (non-temporal) Datalog. The PSPACE lower bound is obtained by reduction from the halting problem for a Turing Machine using polynomial space; it uses the observation that, on an input string w, the machine uses a number t_w of steps that is at most exponential in $|w|$ but t_w can be encoded in binary using polynomially many bits. □

The lower bounds in Theorem 11 yield a new PSPACE lower bound for the data complexity of the full $\mathit{datalogMTL}$ language (with time points encoded in binary), where the

[3] If Π mentions no intervals, it is Datalog, and the number of elements in $W \cup H$ is bounded by $7 \cdot \mathsf{P} \cdot |\mathcal{O}_I|^{\mathsf{A}}$.

best known lower bound was the straightforward P-hardness inherited from plain Datalog.

Corollary 12. *The data complexity of fact entailment in $datalogMTL^{FP}$ (and hence of $datalogMTL$) is* PSPACE-*hard if time points are encoded in binary.*

We conclude this section by studying data complexity for $datalogMTL^{FP}$. In contrast to our previous results, our bounds apply only under the assumption that the gcd of timestamps in the input stream is known in advance, which is the case, e.g., when the gap between consecutive time points in the input dataset is fixed to a constant. We leave open the problem of providing tight bounds in the general case.

Theorem 13. *Let g be a fixed positive rational. Checking whether $(\Pi, \mathcal{D}) \models \alpha@t$ for Π a fixed $datalogMTL^{FP}$ program, $\alpha@t$ an input fact, and $\mathcal{D}$ an input dataset satisfying $g = \mathsf{gcd}(\mathcal{T}_{\mathcal{D}})$ is* (i) PSPACE-*complete if time points in the input are encoded in binary; and* (ii) P-*complete if time points in the input are encoded in unary.*

Proof sketch. To check whether $(\Pi, \mathcal{D}) \models \alpha@t$ we use Algorithm 1 on input $\mathcal{D}$ with a parameter $step = \mathsf{gcd}(\{g\} \cup \mathsf{N})$, where N is the set of rationals occurring in Π. The rules from Table 2 and Table 3 can be exhaustively applied in polynomial time in each iteration. The number of iterations is linear in the input under unary encoding, giving the P upper bound. Under binary encoding the number of iterations is exponential but the size of the memory kept by the algorithm is polynomial, so the PSPACE bound follows.

The P lower bound follows from P-completeness of fact entailment in plain Datalog. PSPACE-hardness is obtained by the same reduction as in the proof of Theorem 11 since all datasets involved have the same temporal domain. □

Related Work

The evaluation of MTL formulas over streams has received significant attention in the literature. Basin, Klaedtke, and Zălinescu (2018) propose an algorithm for checking satisfiability of propositional MTL formulas w.r.t. a stream of facts under continuous semantics; similarly to our work, formulas are given in the past fragment of MTL, and memory consumption is limited by bounding the number of data points in a stream per unit of time. Baldor and Niu (2012) propose an algorithm which assumes a representation of streams as a set of signals capturing changes of truth values of propositional variables; memory consumption is bounded by assuming a finite number of signal changes in any finite length interval. Formula evaluation over streams under the pointwise semantics of MTL has also been extensively studied (Heintz and Doherty 2004; Thati and Roşu 2005; Doherty, Kvarnström, and Heintz 2009; Ničković and Piterman 2010; Baldor and Niu 2012; Ho, Ouaknine, and Worrell 2014; Basin, Klaedtke, and Zălinescu 2018). In all these works, reasoning amounts to model checking over an infinite model; in contrast, we study an entailment problem over a recursive rule set where rules can derive new information.

The complexity of query answering in $datalogMTL$ and its fragments under continuous semantics has been studied by Brandt et al. (2018), who showed that the problem is EXPSPACE-complete for combined complexity and P-hard for data complexity; furthermore, the authors studied the complexity of non-recursive $datalogMTL$ and proved PSPACE-completeness for combined complexity and subpolynomial (AC^0) data complexity bounds. Low complexity fragments of $datalogMTL$ have been studied in (Kikot et al. 2018). Finally, disallowing punctual intervals in MTL formulas is a well-known restriction to obtain more favourable computational properties; satisfiability and model checking of unrestricted MTL formulas are undecidable (Alur and Henzinger 1993), but become EXPSPACE-complete if punctual intervals are disallowed (Alur, Feder, and Henzinger 1996).

Reasoning over streams in the presence of an OWL 2 ontology has been widely studied (Barbieri et al. 2009; Anicic et al. 2011; Dell'Aglio et al. 2014; Özçep, Möller, and Neuenstadt 2014; Margara et al. 2014). OWL 2 ontologies, however, are non-temporal and hence their problem is very different to ours. Stream reasoning has also been recently studied for Datalog$_{1S}$ (Zaniolo 2012; Ronca et al. 2018b; 2018a) and LARS (Beck, Dao-Tran, and Eiter 2018). Similarly to our work, Ronca et al. (2018a) study a forward-propagating version of Datalog$_{1S}$ and define an algorithm based on a sliding window. Datalog$_{1S}$, however, does not allow for metric operators, and rules are evaluated over the natural numbers; this makes our technical results very different to theirs. The LARS framework (Beck, Dao-Tran, and Eiter 2018) aims to unify various approaches to stream reasoning and defines a rule language with (non-metric) modal temporal operators and windowing constructs; the framework considers only reasoning over finite data, and windows are used to select finite fragments of streams.

Conclusion and Future Work

In this paper, we have studied stream reasoning in the context of $datalogMTL$ and proposed and analysed two sound and complete algorithms. We have also investigated the data complexity of standard query answering for $datalogMTL^{FP}$ and $datalogMITL^{FP}$, and improved existing lower bounds on the data complexity of $datalogMTL$.

We see many avenues for future work. First, we will extend our algorithms to support the "Since" operator $\mathcal{S}_{\varrho}$, and also consider extending $datalogMTL$ with non-monotonic negation. Second, we will extend $datalogMTL^{FP}$ with a restricted form of future operators and extend our algorithms and complexity results accordingly. Finally, we believe that our algorithms can be made practical; stream processing algorithms based on MTL have been successfully applied to autonomous aerial vehicles (Heintz and Doherty 2004; Doherty, Kvarnström, and Heintz 2009), and query answering in a non-recursive fragment of $datalogMTL$ has been applied to monitoring turbine data (Brandt et al. 2018).

Acknowledgments

Research supported by the SIRIUS Centre for Scalable Data Access, the EPSRC projects DBOnto, MaSI^3, and ED^3, the NCN grant 2016/23/N/HS1/02168, and the Foundation for Polish Science (FNP).

References

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

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

Anicic, D.; Fodor, P.; Rudolph, S.; and Stojanovic, N. 2011. EP-SPARQL: a unified language for event processing and stream reasoning. In *WWW*, 635–644.

Baldor, K., and Niu, J. 2012. Monitoring dense-time, continuous-semantics, metric temporal logic. In *RV*, 245–259.

Barbieri, D. F.; Braga, D.; Ceri, S.; Della Valle, E.; and Grossniklaus, M. 2009. C-SPARQL: SPARQL for continuous querying. In *WWW*, 1061–1062.

Basin, D.; Klaedtke, F.; and Zălinescu, E. 2018. Algorithms for monitoring real-time properties. *Acta Inform.* 55(4):309–338.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2018. LARS: A logic-based framework for analytic reasoning over streams. *Artif. Intell.* 261:16–70.

Brandt, S.; Kalaycı, E. G.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2018. Querying log data with metric temporal logic. *J. Artif. Intell. Res.* 62:829–877.

Chomicki, J. 1995. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.* 20(2):149–186.

Cosad, C.; Dufrene, K. J.; Heidenreich, K.; McMillon, M.; Jermieson, A.; O'Keefe, M.; and Simpson, L. 2009. Wellsite support from afar. *Oilfield Rev.* 21(2):48–58.

Dell'Aglio, D.; Valle, E. D.; Calbimonte, J.; and Corcho, Ó. 2014. RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems. *Int. J. Semantic Web Inf. Syst.* 10(4):17–44.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2009. A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Auton. Agents Multi-Agent Syst.* 19(3):332–377.

Heintz, F., and Doherty, P. 2004. DyKnow: An approach to middleware for knowledge processing. *J. Intell. Fuzzy Syst.* 15(1):3–13.

Ho, H.-M.; Ouaknine, J.; and Worrell, J. 2014. Online monitoring of metric temporal logic. In *RV*, 178–192.

Kikot, S.; Ryzhikov, V.; Wałęga, P. A.; and Zakharyaschev, M. 2018. On the data complexity of ontology-mediated queries with MTL operators over timed words. In *DL*.

Lécué, F. 2017. Deep dive on smart cities by scaling reasoning and interpreting the semantics of IoT. In *EGC*, 7–8.

Margara, A.; Urbani, J.; van Harmelen, F.; and Bal, H. E. 2014. Streaming the web: Reasoning over dynamic data. *J. Web Semant.* 25:24–44.

Münz, G., and Carle, G. 2007. Real-time analysis of flow data for network attack detection. In *IM*, 100–108.

Ničković, D., and Piterman, N. 2010. From MTL to deterministic timed automata. In *FORMATS*, 152–167.

Nuti, G.; Mirghaemi, M.; Treleaven, P. C.; and Yingsaeree, C. 2011. Algorithmic trading. *IEEE Comput.* 44(11):61–69.

Özçep, Ö. L.; Möller, R.; and Neuenstadt, C. 2014. A stream-temporal query language for ontology based data access. In *KI*, 183–194.

Ronca, A.; Kaminski, M.; Cuenca Grau, B.; and Horrocks, I. 2018a. The window validity problem in rule-based stream reasoning. In *KR*, 571–580.

Ronca, A.; Kaminski, M.; Cuenca Grau, B.; Motik, B.; and Horrocks, I. 2018b. Stream reasoning in temporal datalog. In *AAAI*, 1941–1948.

Thati, P., and Roşu, G. 2005. Monitoring algorithms for metric temporal logic specifications. *Electron. Notes Theor. Comput. Sci.* 113:145–162.

Zaniolo, C. 2012. Logical foundations of continuous query languages for data streams. In *Datalog 2.0*, 177–189.