Web Semantics: Science, Services and Agents on the World Wide Web 76 (2023) 100776

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Stream reasoning with DatalogMTL

Przemysław A. Wałęga *, Mark Kaminski, Dingmin Wang, Bernardo Cuenca Grau

Department of Computer Science, University of Oxford, UK

ARTICLE INFO

Article history:
Received 23 May 2022
Received in revised form 18 December 2022
Accepted 5 February 2023
Available online 10 February 2023

Keywords:
Stream reasoning
DatalogMTL
Temporal reasoning
Monitoring

ABSTRACT

We study stream reasoning in DatalogMTL—an extension of Datalog with metric temporal operators. We propose a sound and complete stream reasoning algorithm that is applicable to forward-propagating DatalogMTL programs, in which propagation of derived information towards past time points is precluded. Memory consumption in our generic 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 may be undesirable in certain practical scenarios 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. We have implemented our approach as an extension of the DatalogMTL reasoner MeTeoR and tested it experimentally. The obtained results support the feasibility of our approach in practice.

1. 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 [1], oil companies monitor sensor readings to detect equipment malfunction and predict maintenance needs [2], network providers analyse flow data to identify traffic anomalies [3], and Internet of Things (IoT) applications typically require real-time analysis of data coming from multiple devices [4].

Data analysis in these applications relies on the ability to recognise relevant events as the input data stream is processed. For example, an analyst in a diagnostics centre may be interested in identifying signals received with high frequency over a period of time. Metric temporal logic (MTL) has been proposed as a suitable formalism for specifying and reasoning in real time over such complex events [5–11]. MTL provides complex formulas such as $\boxminus_{[k_1,k_2]}\varphi$ (or $\diamondminus_{[k_1,k_2]}\varphi$), with k_1 and k_2 rational numbers, which hold at time t if formula φ holds at each (respectively, some) timepoint within $[t-k_2, t-k_1]$. In particular, we can capture our previous example about high frequency signals using a formula $\boxminus_{[0,k_1]}\diamondminus_{[0,k_2]} \mathit{Signal}(s_1)$, which holds if signal s_1 has been received continuously over a time 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 temporal rule language such as DatalogMTL [12,13] 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 DatalogMTL rules:

$$\mathit{Flag}(x,z) \leftarrow \mathit{Monit}(x,z) \wedge \boxminus_{[0,4]} \diamondminus_{[0,2]} \mathit{Signal}(z), \tag{1}$$

$$\boxplus_{[0,3]}\mathit{Monit}(x,z) \leftarrow \mathit{Flag}(y,z) \wedge \mathit{Connect}(x,y). \tag{2}$$

Rule (1) states that a node monitoring a signal will flag it whenever the signal has been received continuously over an interval of length at least 4 with gaps between consecutive readings of length at most 2. In turn, Rule (2) ensures that each signal z, flagged by a node y, will be monitored in the future by all nodes x directly connected to y in the network for a period of length at least 3.

In contrast to standard query answering, where the input data is assumed to be static and finite, streaming data is seen as

* Corresponding author.
E-mail addresses: przemyslaw.walega@cs.ox.ac.uk (P.A. Wałęga), mark.kaminski@cs.ox.ac.uk (M. Kaminski), dingmin.wang@cs.ox.ac.uk (D. Wang), bernardo.cuenca.grau@cs.ox.ac.uk (B.C. Grau).

https://doi.org/10.1016/j.websem.2023.100776

an unbounded sequence of facts that flow through the system and must be processed incrementally. Streaming jobs are long-running: standing queries are deployed once and continue to produce results in near real-time, in the form of an output stream, using only limited memory resources.

Stream reasoning, the problem of query answering over data streams in the presence of background knowledge, has received a great deal of attention in the literature [14–22], and we refer the reader to Section 8 (Related Work) for a detailed discussion of the relevant literature. Stream reasoning in a recursive metric temporal language such as DatalogMTL, however, has not been studied to the best of our knowledge, and raises a number of technical challenges discussed next.

To ensure correctness, stream reasoning systems must compute results over the received segment of a data stream as if the entire (infinite) stream had been available. Furthermore, to achieve low latency and to 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 systems to keep in memory a large (or even unbounded) input history, and/or to delay the output of results (even indefinitely) to ensure correctness. Therefore, there exists a fundamental trade-off involving the basic properties that a stream reasoning 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. This work extends our conference paper [23], where we provided first results on the problems surrounding stream reasoning for DatalogMTL. In line with other works [9,11, 22,24], we focus on *forward-propagating* rules, which are syntactically restricted to preclude propagation of derived information towards past time points. Such rules are 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 is detected).

We propose a sound and complete stream reasoning algorithm for forward-propagating DatalogMTL programs 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 may be problematic 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. 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 in rules operators mentioning punctual intervals, such as $[1, 1]$, which contain a single time point.

We have implemented our approach as an extension of the MeTeoR system [25] and evaluated its performance on a temporal extension of the Lehigh University Benchmark and on one of the tasks from the Hackathon Challenge at the 2021 Stream Reasoning Workshop. Our experiments show that our stream reasoner is able to keep in memory only a very limited number of facts at each point in time. Initially, memory consumption increases rapidly as the first window is populated and new facts are derived; however, as soon as the window starts sliding and the algorithm starts forgetting old facts that are no longer relevant, memory consumption stabilises and remains essentially constant from then onwards.

2. Preliminaries

In this section we recapitulate the syntax and semantics of DatalogMTL, focusing on the standard continuous semantics over the rational timeline [13], as opposed to the integer timeline [26] and the pointwise semantics [27].

Time and Intervals. The *(rational) timeline* is the ordered set $\mathbb{Q}$ of rational numbers; each element of the timeline is called a *time point*. A set $\mathbb{T} \subseteq \mathbb{Q}$ of time points is discrete if each non-maximal $t \in \mathbb{T}$ has a *successor* $\mathrm{succ}_{\mathbb{T}}(t)$ in $\mathbb{T}$; that is, if $\mathrm{succ}_{\mathbb{T}}(t) \in \mathbb{T}$, $t < \mathrm{succ}_{\mathbb{T}}(t)$, and there is no $t' \in \mathrm{succ}_{\mathbb{T}}(t)$ such that $t < t' < t < \mathrm{succ}_{\mathbb{T}}(t)$. For definiteness, we let $\mathrm{succ}_{\mathbb{T}}(t) = \infty$ if t is the maximal element of $\mathbb{T}$. The *greatest common divisor* of a set $\mathbb{T} \subseteq \mathbb{Q}$ of rational numbers, written as $\gcd(\mathbb{T})$, is the greatest rational number which divides all the numbers in $\mathbb{T}$ to integer values (or it is not defined, if such divisor does not exist). The existence of a gcd for a set of time points ensures existence of a minimal distance between them. Indeed, in (infinite) sets without a gcd, such as $\{0, \frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \ldots\}$, there is no minimal distance between numbers.

An *interval* ϱ is a non-empty subset of $\mathbb{Q}$ satisfying two properties: *(i)* for all $t_1, t_2, t_3 \in \mathbb{Q}$ with $t_1 < t_2 < t_3$ and $t_1, t_3 \in \varrho$, it is the case that $t_2 \in \varrho$; and *(ii)* both the greatest lower bound ϱ^- and the least upper bound ϱ^+ of ϱ are in $\mathbb{Q} \cup \{-\infty, \infty\}$. The bounds ϱ^- and ϱ^+ are the *left* and *right endpoints* of ϱ, respectively, and $|\varrho| = \varrho^+ - \varrho^-$ is the *length* of ϱ. Interval ϱ is *punctual* if it contains a single time point, it is *positive* if it does not contain negative time points, and it is *bounded* if both of its endpoints are rational numbers. For intervals ϱ and ϱ' we define the following operations, which will be useful to concisely formulate our algorithms:

$$\varrho + \varrho' = \{t + t' \mid t \in \varrho \text{ and } t' \in \varrho'\},$$
$$\varrho - \varrho' = \{t - t' \mid t \in \varrho \text{ and } t' \in \varrho'\},$$
$$\varrho \oplus \varrho' = \{t \mid (\{t\} - \varrho') \subseteq \varrho\};$$

for example, $[0, 10] + [2, 4] = [0 + 2, 10 + 4] = [2, 14]$, $[0, 10] - [2, 4] = [0 - 4, 10 - 2] = [-4, 8]$, whereas $[0, 10] \oplus [2, 4] = [0 + 4, 10 + 2] = [4, 12]$. We use the standard representation $\langle \varrho^-, \varrho^+ \rangle$ for an interval ϱ, where the *left bracket* $\langle$ is either $[$ or $($, the *right bracket* $\rangle$ is either $]$ or $)$. Brackets $[$ and $]$ indicate that the corresponding endpoints are included in the interval, whereas $($ and $)$ indicate that they are not. We often abbreviate a punctual interval $[t, t]$ as t. Since different intervals cannot have the same representation, we often abuse notation and identify an interval representation with the interval it represents.

Syntax. Assume a function-free first-order signature. A *relational atom* is a first-order atom of the form $P(\mathbf{s})$, with P a predicate and $\mathbf{s}$ a tuple of terms whose length matches the arity of P. A *metric atom* is an expression given by the following grammar, where $P(\mathbf{s})$ is a relational atom and ϱ a positive and non-empty interval:

$$M ::= \top \mid \bot \mid P(\mathbf{s}) \mid \Diamond_{\varrho} M \mid \Diamond_{\varrho} M \mid \boxminus_{\varrho} M \mid \boxplus_{\varrho} M \mid M \mathcal{S}_{\varrho} M \mid M \mathcal{U}_{\varrho} M.$$

A *rule* is an expression of the form

$$M' \leftarrow M_1 \wedge \cdots \wedge M_n, \qquad \text{for } n \geq 1, \tag{3}$$

where each M_i is a metric atom; furthermore, M' is a metric atom not mentioning $\Diamond$, $\Diamond$, $\mathcal{S}$, and $\mathcal{U}$, and hence generated by the following grammar:

$$M' ::= \top \mid \bot \mid P(\mathbf{s}) \mid \boxminus_{\varrho} M' \mid \boxplus_{\varrho} M'.$$

The conjunction $M_1 \wedge \cdots \wedge M_n$ in Rule (3) is the rule's *body*; each M_i is a *body atom* and M' is the rule's *head*. A rule is *safe* if each variable mentioned in the head occurs also in its body, and this

Table 1
Semantics of ground metric atoms.

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

occurrence is not in a left operand of $\mathcal{S}$ or $\mathcal{U}$. A *program* is a finite set of safe rules. An expression (metric atom, rule, or program) is *ground* if it has no variables.

A *metric fact* over an interval ϱ is an expression $M@\varrho$, with M a ground metric atom; it is relational if so is M and bounded if so is ϱ. A *dataset* is a finite set of relational facts. The *grounding* $\mathsf{ground}(\Pi)$ of Π is the set of ground rules obtained by assigning constants to variables in Π. In turn, the grounding $\mathsf{ground}(\Pi, X)$ of Π with respect to a set X of expressions (e.g., facts and atoms) is the restriction of $\mathsf{ground}(\Pi)$ to ground rules mentioning only constants occurring in Π and X.

Semantics. An *interpretation* $\mathfrak{I}$ is a function which assigns to each time point t a set of ground relational atoms; if an atom $P(\mathbf{c})$ belongs to this set, we say that $P(\mathbf{c})$ is *satisfied* at t in $\mathfrak{I}$ and we write $\mathfrak{I}, t \models P(\mathbf{c})$. This notion extends to ground metric atoms with temporal operators as specified in Table 1. Interpretation $\mathfrak{I}$ satisfies a metric fact $M@\varrho$, written $\mathfrak{I} \models M@\varrho$, if $\mathfrak{I}, t \models M$ for all $t \in \varrho$. An interpretation $\mathfrak{I}$ satisfies a ground rule r if, whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $\mathfrak{I}$ also satisfies the head of r at t. An interpretation $\mathfrak{I}$ satisfies a program Π if it satisfies all rules in $\mathsf{ground}(\Pi)$. An interpretation $\mathfrak{I}$ is a *model* of a program Π if it satisfies Π, and it is a *model* of a set $\mathcal{M}$ of metric facts if it satisfies each element of $\mathcal{M}$. A set $\mathcal{M}$ of metric facts *entails* a metric fact $M@\varrho$ if in every interpretation $\mathfrak{I}$ such that $\mathfrak{I} \models M'@\varrho'$, for each $M'@\varrho' \in \mathcal{M}$, it holds that $\mathfrak{I} \models M@\varrho$. A program Π and a set $\mathcal{M}$ of metric facts (e.g., a dataset) *entail* a set $\mathcal{M}'$ of metric facts, written $(\Pi, \mathcal{M}) \models \mathcal{M}'$, if each model of both Π and $\mathcal{M}$ is also a model of $\mathcal{M}'$. We may write $\mathcal{M} \models \mathcal{M}'$ instead of $(\emptyset, \mathcal{M}) \models \mathcal{M}'$. Moreover, if $\mathcal{M}$ or $\mathcal{M}'$ is a singleton, say $\{M@\varrho\}$, then we may omit curly brackets and write $M@\varrho \models \mathcal{M}'$ and $\mathcal{M} \models M@\varrho$, respectively.

An interpretation $\mathfrak{I}$ *contains* an interpretation $\mathfrak{I}'$, written $\mathfrak{I}' \subseteq \mathfrak{I}$, if $\mathfrak{I}', t \models P(\mathbf{s})$ implies $\mathfrak{I}, t \models P(\mathbf{s})$, for each ground relational atom $P(\mathbf{s})$ and time point t. Then, $\mathfrak{I}$ is the *least* interpretation in a set S of interpretations if $\mathfrak{I} \subseteq \mathfrak{I}'$ for every $\mathfrak{I}' \in S$. Each set $\mathcal{M}$ of relational facts admits the least interpretation $\mathfrak{I}_\mathcal{M}$ amongst all models of $\mathcal{M}$ and we say that a set $\mathcal{M}$ *represents* an interpretation $\mathfrak{I}$ if $\mathfrak{I} = \mathfrak{I}_\mathcal{M}$.

Canonical Interpretation. The *immediate consequence operator* T_Π for a program Π is a function mapping an interpretation $\mathfrak{I}$ to the least interpretation containing $\mathfrak{I}$ and satisfying the following property for each $r \in \mathsf{ground}(\Pi)$: whenever $\mathfrak{I}$ satisfies each body atom of r at a time point t, then $T_\Pi(\mathfrak{I})$ satisfies the head of r at t. The successive application of T_Π to $\mathfrak{I}_\mathcal{M}$, where $\mathcal{M}$ is a (possibly infinite) set of relational facts, defines a transfinite sequence of interpretations $T_\Pi^\alpha(\mathfrak{I}_\mathcal{M})$ for ordinals α as follows:

$$T_\Pi^0(\mathfrak{I}_\mathcal{M}) = \mathfrak{I}_\mathcal{M},$$
$$T_\Pi^\alpha(\mathfrak{I}_\mathcal{M}) = T_\Pi(T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{M})), \quad \text{for a successor } \alpha,$$
$$T_\Pi^\alpha(\mathfrak{I}_\mathcal{M}) = \bigcup_{\beta<\alpha} T_\Pi^\beta(\mathfrak{I}_\mathcal{M}), \quad \text{for a limit } \alpha.$$

The *canonical interpretation* $\mathfrak{C}_{\Pi,\mathcal{M}}$ of Π and $\mathcal{M}$ is defined as $T_\Pi^{\omega_1}(\mathfrak{I}_\mathcal{M})$, where ω_1 is the first uncountable ordinal [28]. If Π and $\mathcal{M}$ have a model, then $\mathfrak{C}_{\Pi,\mathcal{M}}$ is the least model of Π and $\mathcal{M}$. The canonical interpretation $\mathfrak{C}_{\Pi,\mathcal{M}}$ can be divided into regularly distributed $(\Pi, \mathcal{M})$-*intervals* whose time points satisfy the same ground atoms [13]. In particular, the $(\Pi, \mathcal{M})$-*ruler* is the set of time points of the form $t + i \cdot \mathsf{div}(\Pi)$, for $t \in \mathbb{Q}$ mentioned in $\mathcal{M}$ and $i \in \mathbb{Z}$, and where $\mathsf{div}(\Pi) = \frac{1}{k}$, with k being the product of all denominators in the rational endpoints of the intervals mentioned in Π (for definiteness, if Π has no intervals with rational endpoints we let $k = 1$ and hence $\mathsf{div}(\Pi) = 1$). A $(\Pi, \mathcal{M})$-*interval* is either a punctual interval over a time point on the $(\Pi, \mathcal{M})$-ruler, or an interval (t_1, t_2) with t_1 and t_2 consecutive time points on the $(\Pi, \mathcal{M})$-ruler. Interpretation $\mathfrak{I}$ is a $(\Pi, \mathcal{M})$-interpretation if for every relational fact $M@t$ it holds that $\mathfrak{I} \models M@t$ implies $\mathfrak{I} \models M@\varrho$, where ϱ is the $(\Pi, \mathcal{M})$-interval containing t. It turns out that $\mathfrak{C}_{\Pi,\mathcal{M}}$ as well as $T_\Pi^\alpha(\mathfrak{I}_\mathcal{M})$, for any ordinal α, are $(\Pi, \mathcal{M})$-interpretations [13].

Reasoning. A key reasoning tasks in DatalogMTL is to check whether a relational fact is entailed by a program and a dataset. This problem is PSpace-complete in data complexity [13], that is, when the size of a program is considered fixed, and ExpSpace-complete in combined complexity [28]. These complexity bounds hold already in the case of bounded programs and datasets [29].

3. Streams and stream queries

Streaming data is often represented as an unbounded sequence of timestamped facts. Similarly to Brandt et al. [12], we 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 [20,21]. We also assume that the timeline of a stream is discrete—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 $\mathcal{S}$ assigning a (possibly empty) finite set of ground relational atoms to each $t \in \mathbb{Q}_{\geq 0}$. The *temporal domain* $\mathbb{T}_\mathcal{S}$ of $\mathcal{S}$ is the set of time points t satisfying $\mathcal{S}(t) \neq \emptyset$; we require that the temporal domain of every stream is a discrete set. Moreover, we say that $\mathcal{S}$ is *regular* if $\mathsf{gcd}(\mathbb{T}_\mathcal{S})$ is well-defined. The *object domain* $\mathbb{O}_\mathcal{S}$ of $\mathcal{S}$ is the set of all constants occurring in the values of $\mathcal{S}$. By slight abuse of notation, we identify a stream $\mathcal{S}$ with the (possibly infinite) set of facts $\{M@t \mid t \in \mathbb{T}_\mathcal{S} \text{ and } M \in \mathcal{S}(t)\}$.

A stream 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) consisting of a program Π and a predicate Q. The *answer* to (Π, Q) over a stream $\mathcal{S}$, relative to a discrete set $\mathbb{T} \subseteq \mathbb{Q}_{\geq 0}$, is the stream $\{Q(\mathbf{c})@t \mid t \in \mathbb{T} \text{ and } (\Pi, \mathcal{S}) \models Q(\mathbf{c})@t\}$.

Note that set $\mathbb{T}$ in the previous definition specifies the time points for which answers are required in the output; thus, $\mathbb{T}$ contains the temporal domain of the answer stream.

4. Forward-propagating programs

In the context of stream reasoning we will not consider arbitrary DatalogMTL programs, but instead we will restrict ourselves to programs that are *forward-propagating*.

On the one hand, forward-propagating programs are always consistent with a dataset, which allows us to focus on fact entailment as the key reasoning problem. On the other hand, rules in a forward-propagating program are syntactically restricted so that their application to a set of facts can derive information relevant to present and future time points, but not to past time points.

This 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); similar restrictions have been adopted in the stream reasoning [24], monitoring [11], and database view update literature [30]. From an algorithmic perspective, these restrictions will allow us to limit memory consumption by forgetting previously received facts without compromising completeness.

Definition 4. A rule is *forward-propagating* if it satisfies all of the following restrictions:

- it does not mention $\top$ or $\bot$,
- it does not mention in the body any metric operators except $\boxminus$ and $\diamondminus$, and
- it does not mention in the head any metric operators except $\boxplus$.

A DatalogMTL program is forward-propagating if so is each of the rules it contains. A stream query (Π, Q) is forward-propagating if so is Π.

For convenience, and without loss of generality, we will assume that forward-propagating programs are given in the normal form defined next, where there is no nesting of metric temporal operators.

Definition 5. A forward-propagating program is in *normal form* if it contains only rules of the following forms:

$$M' \leftarrow \diamondminus_{\varrho} M, \tag{4}$$

$$M' \leftarrow \boxminus_{\varrho} M, \tag{5}$$

$$M' \leftarrow M_1 \wedge \cdots \wedge M_n, \qquad \text{for } n \geq 1, \tag{6}$$

where $M, M', M_1, \ldots, M_n$ are relational atoms and ϱ is a positive non-empty interval. A forward-propagating stream query (Π, Q) is in normal form if so is Π.

As we show next, each forward-propagating program can be normalised in polynomial time.

Proposition 6. *Each forward-propagating program Π can be transformed in polynomial time into a program Π' in normal form such that, for every stream $\mathcal{S}$ and each relational fact $P(\mathbf{c})@\varrho$ with P in the signature of Π and $\mathcal{S}$, we have $(\Pi, \mathcal{S}) \models P(\mathbf{c})@\varrho$ if and only if $(\Pi', \mathcal{S}) \models P(\mathbf{c})@\varrho$.*

Proof. We construct Π' in two steps. First, we ensure that rule heads do not mention $\boxplus$. To this end, we use the correspondence between the box and diamond operators and replace each rule $\boxplus_{\varrho_1} \cdots \boxplus_{\varrho_k} P(\mathbf{s}) \leftarrow M_1 \wedge \cdots \wedge M_n$ with the following two rules: $P'(\mathbf{s}) \leftarrow M_1 \wedge \cdots \wedge M_n$ and $P(\mathbf{s}) \leftarrow \diamondminus_{\varrho_1 + \cdots + \varrho_k} P'(\mathbf{s})$, where P' is a fresh predicate of the same arity as P. In the second step, we introduce additional rules with fresh predicates in order to eliminate temporal operators from rules with conjunctions in bodies and to flatten nested operators from the remaining rules. For example, a rule $M' \leftarrow M_1 \wedge \cdots \wedge M_n$ with $M_1 = \boxminus_{\varrho_1} \diamondminus_{\varrho_2} P(\mathbf{s})$ is replaced with three rules $P'(\mathbf{s}) \leftarrow \diamondminus_{\varrho_2} P(\mathbf{s})$, $P''(\mathbf{s}) \leftarrow \boxminus_{\varrho_1} P'(\mathbf{s})$, and $M' \leftarrow P''(\mathbf{s}) \wedge \cdots \wedge M_n$, where P' and P'' are fresh predicates of the same arity as P. □

Example 7. Observe that the program in Example 1, which consists of Rules (1) and (2), is forward-propagating, but it is not in normal form. Indeed, after normalisation of Rule (1) we obtain the following rules:

$$P(z) \leftarrow \diamondminus_{[0,2]} \mathit{Signal}(z), \tag{7}$$

$$P'(z) \leftarrow \boxminus_{[0,4]} P(z), \tag{8}$$

$$\mathit{Flag}(x, z) \leftarrow \mathit{Monit}(x, z) \wedge P'(z), \tag{9}$$

and after the normalisation of Rule (2) we obtain:

$$P''(x, z) \leftarrow \mathit{Flag}(y, z) \wedge \mathit{Connect}(x, y),$$

$$\mathit{Monit}(x, z) \leftarrow \diamondminus_{[0,3]} P''(x, z),$$

where P, P', and P'' are fresh predicates.

In the remainder of this paper, we will focus on forward-propagating stream queries in normal form.

5. A generic stream reasoning algorithm

In this section, we present a generic stream reasoning algorithm applicable to forward-propagating stream queries. Our procedure (Algorithm 1) takes as input a stream $\mathcal{S}$ and outputs, also as a stream, the answers to a standing forward-propagating query (Π, Q). The query is assumed to be in normal form and fixed, as a parameter. The algorithm is also parametrised with a discrete set $\mathbb{T}$, which specifies the temporal domain of the answer stream, and a value step which controls the incrementality of query processing.

Algorithm 1: Generic stream reasoning algorithm

Parameters: A stream query (Π, Q) with a forward-propagating program Π in normal form, a discrete set $\mathbb{T} \subseteq \mathbb{Q}_{\geq 0}$, and $\mathsf{step} \in \mathbb{Q}_{>0}$
Input: A stream $\mathcal{S}$

1 $t := -\mathsf{step}$; // current time point
2 $\mathcal{W} := \emptyset$; // window contents
3 $\mathcal{H} := \emptyset$; // history contents
4 $t_\Pi :=$ maximal rational endpoint in intervals of Π (or 0 if Π does not mention rational numbers);
5 **loop**
6 **if** $\mathsf{succ}_{\mathcal{S}}(t) \leq t + \mathsf{step}$ **then**
7 $t_{\mathsf{next}} := \mathsf{succ}_{\mathcal{S}}(t)$;
8 $\mathcal{W} := \mathcal{W} \cup \mathcal{S}(t_{\mathsf{next}})$;
9 **else**
10 $t_{\mathsf{next}} := t + \mathsf{step}$;
11 $\mathcal{W} := \mathsf{ApplyRules}(\Pi, \mathcal{W}, \mathcal{H}, t, t_{\mathsf{next}})$;
12 **for** *each* $Q(\mathbf{c})@\varrho \in \mathcal{W}$ **do**
13 **for** *each* $t' \in \varrho \cap (t, t_{\mathsf{next}}] \cap \mathbb{T}$ **do**
14 **stream out** $Q(\mathbf{c})@t'$;
15 $t := t_{\mathsf{next}}$;
16 $(\mathcal{W}, \mathcal{H}) := \mathsf{Forget}(\mathcal{W}, \mathcal{H}, t, t_\Pi)$;

To simplify the presentation, and without loss of generality, we assume that the input stream $\mathcal{S}$ mentions only predicates occurring in Π (facts involving other predicates are irrelevant for answering the query) and that all constants from Π occur in $\mathcal{S}$ (this can be easily achieved by extending $\mathcal{S}$ with facts explicitly mentioning these constants).

Algorithm 1 is initialised in Lines 1–4; Line 1 initialises the current time point t in the input stream to a value smaller than all time points in $\mathbb{T}_{\mathcal{S}}$ (note that we initialise t as $-\mathsf{step}$, but we could use any value smaller than the first time point in the input stream), and Line 4 sets t_Π to the maximum rational number occurring as an interval endpoint in program Π. Moreover, Lines 2 and 3 initialise the memory contents, which consist of the following two parts.

- The *window* $\mathcal{W}$—a set of relational facts storing information about atoms holding in the most recent interval (in particular, in the interval $[t - t_{\Pi}, t]$).
- The *history* $\mathcal{H}$—a set of relational atoms storing information about atoms which hold further back into the past than the recent window covered by $\mathcal{W}$, 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 6–10, the algorithm processes the next fragment of the input stream by either moving to the next time point in the stream containing data and loading the relevant facts into the window memory, or moving forward by step (if there is no data point between t and t + step). In Line 11, the algorithm updates $\mathcal{W}$ by applying to them rules from Π which derive new information within the interval $(t, t_{\text{next}}]$. In Lines 12–14 the algorithm outputs all answers to the query in-between t and t_{next}. Finally, in Line 16 the algorithm trims the window $\mathcal{W}$ by forgetting information that is no longer relevant and updates the history $\mathcal{H}$.

Next, we describe in detail the procedures ApplyRules and Forget from Lines 11 and 16, respectively. Procedure ApplyRules is presented in Algorithm 2. It takes as input a program Π, a window $\mathcal{W}$, a history $\mathcal{H}$, and two time points t and t_{next}, to compute updated $\mathcal{W}$. In particular, ApplyRules successively applies rules of Π to $\mathcal{W}$ and $\mathcal{H}$ (Lines 2–3) to derive all the facts that hold within the interval $(t, t_{\text{next}}]$. If no more rules can be applied (Line 17), the procedure terminates and returns updated $\mathcal{W}$ (Line 18). The rules in Π of Form (4) are applied in Lines 3–7; rules of Form (5) in Lines 8–10; and rules of Form (6) are applied in Lines 11–13. Additionally, in Lines 14–16, facts in $\mathcal{W}$ mentioning the same atoms are coalesced by merging together facts over adjacent intervals.

Algorithm 2: ApplyRules

Input: A forward-propagating program Π in normal form, a set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_{next}
Output: A set of facts

1 **repeat**
2 **for** *each* $r \in \text{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$ **do**
3 **if** *r is of the form* $M' \leftarrow \diamondsuit_{\varrho_2} M$ **then**
4 **for** *each* $M@\varrho_1 \in \mathcal{W}$ *such that* $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$ *is non-empty* **do**
5 Add $M'@\varrho$ to $\mathcal{W}$;
6 **if** ϱ_2 *is unbounded and* $M \in \mathcal{H}$ **then**
7 Add $M'@(t, t_{\text{next}}]$ to $\mathcal{W}$;
8 **if** *r is of the form* $M' \leftarrow \boxminus_{\varrho_2} M$ **then**
9 **for** *each* $M@\varrho_1 \in \mathcal{W}$ *such that* $\varrho = (\varrho_1 \oplus \varrho_2) \cap (t, t_{next}]$ *is non-empty* **do**
10 Add $M'@\varrho$ to $\mathcal{W}$;
11 **if** *r is of the form* $M' \leftarrow M_1 \wedge \cdots \wedge M_n$ **then**
12 **for** *all* $M_1@\varrho_1, \ldots, M_n@\varrho_n \in \mathcal{W}$ *such that* $\varrho = \varrho_1 \cap \cdots \cap \varrho_n$ *is non-empty* **do**
13 Add $M'@\varrho$ to $\mathcal{W}$;
14 **for** *each* $M@\varrho_1, M@\varrho_2 \in \mathcal{W}$ **do**
15 **if** $\varrho = \varrho_1 \cup \varrho_2$ *is an interval* **then**
16 Add $M@\varrho$ to $\mathcal{W}$;
17 **until** *no more elements are added to* $\mathcal{W}$;
18 **return** $\mathcal{W}$;

Example 8. Consider Algorithm 2 running on a program Π consisting of Rules (7)–(9) presented in Example 7, empty history $\mathcal{H}$, window $\mathcal{W}$ with facts

$$\textit{Signal}(s_1)@96.3, \quad \textit{Signal}(s_1)@98, \quad \textit{Signal}(s_1)@100, \quad P(s_1)@[97, 100], \quad \textit{Monit}(n, s_1)@101,$$

and time points $t = 100$, $t_{\text{next}} = 101$, and $t_{\Pi} = 4$.

Algorithm 2 extends $\mathcal{W}$ with new facts. In Lines 3–7, Rule (7) is applied to the fact $\textit{Signal}(s_1)@100$, and so, $P(s_1)@(100, 101]$ is added to $\mathcal{W}$.

Facts $P(s_1)@[97, 100]$ and $P(s_1)@(100, 101]$ are coalesced in Lines 14–16, and so the fact $P(s_1)@[97, 101]$ is added to $\mathcal{W}$. In Lines 8–10, Rule (8) is applied to the fact $P(s_1)@[97, 101]$, adding $P'(s_1)@101$ to $\mathcal{W}$.

Next, Rule (9) is applied to facts $P'(s_1)@101$ and $\textit{Monit}(n, s_1)@101$ in Lines 11–13, and $\textit{Flag}(n, s_1)@101$ is added to $\mathcal{W}$, as a result.

We next describe the procedure Forget specified by Algorithm 3, which trims the window memory $\mathcal{W}$ by forgetting information that is no longer relevant and adds corresponding atoms to $\mathcal{H}$. In Lines 1–7, the algorithm 'slides' the window by deleting from $\mathcal{W}$ old information that falls outside $[t - t_{\Pi}, t]$ and by extending the history $\mathcal{H}$ with corresponding ground atoms (not mentioning intervals); intuitively, such atoms hold outside the sliding window, that is, somewhere within the range $(-\infty, t - t_{\Pi})$. Then, in Lines 8 and 9, the algorithm it eliminates redundancy by deleting from $\mathcal{W}$ all relational facts subsumed by others.

Example 9. Consider Algorithm 3 running on an input consisting of the window $\mathcal{W}$ computed by ApplyRules in Example 8 and the time points $t = 101$ and $t_{\Pi} = 4$. In Lines 1–7 the algorithm forgets all facts that hold to the left of the sliding window (i.e., to the left of $t - t_{\Pi} = 97$). In particular, the fact $\textit{Signal}(s_1)@96.3$ lies (at least partially) to the left of the sliding window (as $96.3 < 97$); hence, its ground atom $\textit{Signal}(s_1)$ is added to $\mathcal{H}$ in Line 3 and $\textit{Signal}(s_1)@96.3$ is removed from $\mathcal{W}$ in Line 5.

Furthermore, in Lines 8 and 9, the facts $P(s_1)@[97, 100]$ and $P(s_1)@(100, 101]$ are deleted from $\mathcal{W}$, as they are both subsumed by $P(s_1)@[97, 101]$, which is also in $\mathcal{W}$.

Algorithm 3: Forget

Input: A set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_{Π}
Output: A pair consisting of a set of facts and a set of atoms

1 **for** *each* $M@\varrho \in \mathcal{W}$ **do**
2 **if** $\varrho \not\subseteq [t - t_{\Pi}, t]$ **then**
3 Add M to $\mathcal{H}$;
4 **if** $\varrho \cap [t - t_{\Pi}, t] = \emptyset$ **then**
5 Delete $M@\varrho$ from $\mathcal{W}$;
6 **else**
7 Replace $M@\varrho$ by $M@\varrho \cap [t - t_{\Pi}, t]$ in $\mathcal{W}$;
8 **for** *each* $M@\varrho_1, M@\varrho_2 \in \mathcal{W}$ *with* $\varrho_1 \subsetneq \varrho_2$ **do**
9 Delete $M@\varrho_1$ from $\mathcal{W}$;
10 **return** $(\mathcal{W}, \mathcal{H})$;

The following theorems establish soundness and completeness of Algorithm 1.

Theorem 10 (*Soundness*). *Each fact streamed out by Algorithm 1 is in the answer to* (Π, Q) *over* $\mathcal{S}$ *relative to* $\mathbb{T}$.

Proof. Each fact $Q(\mathbf{c})@t'$ streamed out by Algorithm 1 is such that $t' \in \mathbb{T}$ and there exists $Q(\mathbf{c})@\varrho \in \mathcal{W}$ with $t' \in \varrho$ (see Lines 12–14). Thus, it suffices to show that if $Q(\mathbf{c})@\varrho \in \mathcal{W}$, then $(\Pi, \mathcal{S}) \models Q(\mathbf{c})@\varrho$. To this end, we will show that, in each step of Algorithm 1, the following statements hold for all relational atoms M and all intervals ϱ (where t is the current time point kept by Algorithm 1):

1. If $M@\varrho \in \mathcal{W}$, then $(\Pi, \mathcal{S}) \models M@\varrho$.
2. If $M \in \mathcal{H}$, then $(\Pi, \mathcal{S}) \models M@t'$ for some $t' < t - t_\Pi$.

To show these statements we proceed inductively on the number of computations adding new elements to $\mathcal{W}$ or $\mathcal{H}$.

Basis of induction: The basis for Statements 1 and 2 holds trivially, as $\mathcal{W}$ and $\mathcal{H}$ are initially empty (by Lines 2 and 3).

Inductive step for Statement 1: New facts can be added to $\mathcal{W}$ only in Line 8 of Algorithm 1, in Lines 5, 7, 10, 13, and 16 of Algorithm 2, and in Line 7 of Algorithm 3. We will argue that all the new facts are entailed by Π and $\mathcal{S}$.

If a fact is added to $\mathcal{W}$ in Line 8 of Algorithm 1, then this fact holds in $\mathcal{S}$. Thus, the fact is entailed by Π and $\mathcal{S}$.

If $M'@\varrho$ is added to $\mathcal{W}$ by Algorithm 2 in Line 5, there is $M@\varrho_1$ in $\mathcal{W}$ and $M' \leftarrow \diamondsuit_{\varrho_2} M$ in $\mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$ such that $\varrho \subseteq \varrho_1 + \varrho_2$. By the inductive assumption, we have $(\Pi, \mathcal{S}) \models M@\varrho_1$. Hence, by the semantics of $\diamondsuit_{\varrho_2}$, we obtain that $(\Pi, \mathcal{S}) \models M'@\varrho_1 + \varrho_2$, and so, $(\Pi, \mathcal{S}) \models M'@\varrho$.

If $M'@(t, t_{\mathsf{next}}]$ is added by Algorithm 2 in Line 5, there is M in $\mathcal{H}$ and $M' \leftarrow \diamondsuit_{\varrho_2} M$ in $\mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$, with unbounded ϱ_2. By Statement 2, $(\Pi, \mathcal{S}) \models M@t'$ for some $t' < t - t_\Pi$, so $(\Pi, \mathcal{S}) \models M'@t' + \varrho_2$. By the definition of t_Π, we have $\varrho_2^- \leq t_\Pi$, so $t' + \varrho_2^- \leq t$. Since ϱ_2 is unbounded, $(\Pi, \mathcal{S}) \models M'@(t, \infty)$ and thus $(\Pi, \mathcal{S}) \models M'@(t, t_{\mathsf{next}}]$.

If $M'@\varrho$ is added to $\mathcal{W}$ by Algorithm 2 in Line 10, then $(\Pi, \mathcal{S}) \models M'@\varrho$, by a similar argument to the one we used in the case of Line 5, but where we now use the correspondence between $\oplus$ and $\boxminus$ instead of that between $+$ and $\diamondsuit$.

If a fact is added to $\mathcal{W}$ by Algorithm 2 in Lines 13 or 16, it follows from the inductive assumption that this fact must be entailed by Π and $\mathcal{S}$.

Finally, if a fact $M@\varrho \cap [t - t_\Pi, t]$ is added to $\mathcal{W}$ in Line 7 of Algorithm 3, then there was already $M@\varrho$ in $\mathcal{W}$ so, by the inductive assumption, $(\Pi, \mathcal{S}) \models M@\varrho$, which implies that $(\Pi, \mathcal{S}) \models M@\varrho \cap [t - t_\Pi, t]$.

Inductive step for Statement 2: An atom M can be added to $\mathcal{H}$ only in Line 3 of Algorithm 3; therefore, there exists $M@\varrho \in \mathcal{W}$ with $\varrho^- < t - t_\Pi$. By Statement 1, $(\Pi, \mathcal{S}) \models M@\varrho$, so there is $t' \in \varrho'$ such that $t' < t - t_\Pi$ and $(\Pi, \mathcal{S}) \models M@t'$, as required. □

Theorem 11 (*Completeness*)**.** *Each fact in the answer to (Π, Q) over $\mathcal{S}$ relative to $\mathbb{T}$ is streamed out by Algorithm 1.*

Proof. If a fact $Q(\mathbf{c})@t'$ is in the answer, $(\Pi, \mathcal{S}) \models Q(\mathbf{c})@t'$. By Lines 12–14 in Algorithm 1, this fact will be streamed out provided that $Q(\mathbf{c})@\varrho \in \mathcal{W}$, for some interval ϱ containing t'. Thus, we show that, for each relational fact $M'@t'$, if $(\Pi, \mathcal{S}) \models M'@t'$ then, at some step during the execution of Algorithm 1 there exists $M'@\varrho \in \mathcal{W}$ such that $t' \in \varrho$.

For a time point $t' > -\mathsf{step}$ and $\ell \in \{1, \dots, 16\}$, we use $\mathcal{W}(t', \ell)$ and $\mathcal{H}(t', \ell)$ to respectively represent the contents of $\mathcal{W}$ and $\mathcal{H}$ after executing Line ℓ in the iteration of the main loop of Algorithm 1 in which $t' \in (t, t_{\mathsf{next}}]$, for the value of t_{next} obtained just after executing Lines 6–10. Since all facts entailed by Π and $\mathcal{S}$ are satisfied in $T_\Pi^{\omega_1}(\mathfrak{I}_\mathcal{S})$, it suffices to show that, for each ordinal α and each relational fact $M'@t'$, if $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) \models M'@t'$, then there exists $M'@\varrho$ in $\mathcal{W}(t', 11)$ such that $t' \in \varrho$. We will prove this implication by a transfinite induction on ordinals α.

Basis of induction: Clearly, we have $T_\Pi^0(\mathfrak{I}_\mathcal{S}) = \mathfrak{I}_\mathcal{S}$. If $T_\Pi^0(\mathfrak{I}_\mathcal{S}) \models M'@t'$, then $\mathcal{S} \models M'@t'$. Thus, by Lines 7 and 8, we obtain $M'@t' \in \mathcal{W}(t', 8)$. As executing ApplyRules in Line 11 does not delete any facts from $\mathcal{W}$, we have $M'@t' \in \mathcal{W}(t', 11)$.

Inductive step for a successor ordinal α: We assume that $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) \models M'@t'$, but $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \not\models M'@t'$. Hence, by the definition of T_Π, there exists $r \in \mathsf{ground}(\Pi)$ of one of Forms (4)–(6), such that the body of r holds at t' in $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S})$ and the head of r is M'. By the inductive assumption and the fact that all constants occurring in the entailed facts occur in $\mathcal{W}$ or $\mathcal{H}$, we obtain that $r \in \mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$.

Assume that r is of Form (4), that is $M' \leftarrow \diamondsuit_{\varrho_2} M$. Then, there exists t'' such that $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \models M@t''$ and $t' \in t'' + \varrho_2$. Thus, by the inductive assumption, $M@\varrho'' \in \mathcal{W}(t'', 11)$, for some ϱ'' with $t'' \in \varrho''$. The execution of Forget in Line 16 ensures that either there exists $M@\varrho_1 \in \mathcal{W}(t', 11)$ with $t'' \in \varrho_1$, or the interval ϱ_2 is unbounded and there exists $M \in \mathcal{H}(t', 11)$. In the first case, by Line 3–5 of Algorithm 2, we have $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$. In the latter case, by Lines 6 and 7 of Algorithm 2, we obtain that $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (t, t_{\mathsf{next}}]$. Since $t' \in (t, t_{\mathsf{next}}]$, we obtain that $t' \in \varrho$, as required.

Next, assume that r is of Form (5), that is $M' \leftarrow \boxminus_{\varrho_2} M$. Hence, $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \models \boxminus_{\varrho_2} M@\varrho'$, where ϱ' is such that $\varrho' \oplus \varrho_2 = t'$. Since Π is forward propagating, ϱ_2 needs to be bounded, so ϱ' is also bounded and $(\varrho')^+ < t_\Pi$. Thus, we can use the inductive assumption to show that, for each $t'' \in \varrho'$, there exists ϱ'' such that $M@\varrho'' \in \mathcal{W}(t', 11)$. Moreover, by Lines 14–16 of Algorithm 2, there exists ϱ_1 such that $\varrho' \subseteq \varrho_1$ and $M@\varrho_1 \in \mathcal{W}(t', 11)$. By Lines 8–11 of Algorithm 2, we obtain that $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (\varrho_1 \oplus \varrho_2) \cap (t, t_{\mathsf{next}}]$. Since $\varrho_1 \oplus \varrho_2 = t'$ and $t' \in (t, t_{\mathsf{next}}]$, we get $t' \in \varrho$.

Finally we assume that rule r is of Form (6), that is $M \leftarrow M_1 \wedge \dots \wedge M_n$. Thus, all of $M_1, \dots, M_n$ need to be satisfied at t' in $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S})$. By the inductive assumption, there exist intervals $\varrho_1, \dots, \varrho_n$ such that all of $M_1@\varrho_1, \dots, M_n@\varrho_n$ belong to $\mathcal{W}(t', 11)$ and $t' \in \varrho_1 \cap \dots \cap \varrho_n$. Thus, by Lines 11–13 of Algorithm 2, we get $M@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = \varrho_1 \cap \dots \cap \varrho_n$, and so, $t' \in \varrho$.

Inductive step for a limit ordinal α: By the definition, $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) = \bigcup_{\beta<\alpha} T_\Pi^\beta(\mathfrak{I}_\mathcal{S})$. Thus, if $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) \models M'@t'$, then there exists $\beta < \alpha$ such that $T_\Pi^\beta(\mathfrak{I}_\mathcal{S}) \models M'@t'$. Then, by the inductive assumption, we obtain that there exists $M'@\varrho$ in $\mathcal{W}(t', 11)$ such that $t' \in \varrho$. □

Finally, we analyse the memory requirements of Algorithm 1 by providing a bound on the number of facts stored in the window and the history at any point during its execution.

Theorem 12 (*Memory Bound*)**.** *At each point during the execution of Algorithm 1 on a regular (i.e., with well-defined* gcd*) input stream $\mathcal{S}$, the number of elements in the set $\mathcal{W} \cup \mathcal{H}$ is bounded by*

$$\left(4 \cdot \left(\frac{t_\Pi + \mathsf{step}}{\mathsf{gcd}(\mathbb{T}_\mathcal{S} \cup \mathsf{N} \cup \{\mathsf{step}\})} + 1\right)^2 + 1\right) \cdot \mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A},$$

where N *is the set of rationals mentioned as interval endpoints in Π,* P *is the number of predicates in Π, and* A *is the maximum arity of these predicates.*

Proof. We observe that each element of $\mathcal{H}$ is a ground atom which mentions one of P predicates from Π and at most A (possibly repeating) constants from $|\mathbb{O}_\mathcal{S}|$. Hence the cardinality of $\mathcal{H}$ is bounded by $\mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A}$. Similarly, for any interval ϱ, there are at most $\mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A}$ relational facts of the form $M@\varrho$ in $\mathcal{W}$. Thus,

it suffices to determine a bound on the number of intervals that may occur in $\mathcal{W}$. To this end, we observe that, by the construction of $\mathcal{W}$, the endpoints ϱ^- and ϱ^+ divided by $\gcd(\mathbb{T}_\mathcal{S} \cup \mathsf{N} \cup \{\mathsf{step}\})$ need to yield integer values and that $\varrho \subseteq [t-t_\Pi, t+\mathsf{step}]$. Indeed, the former follows from the definition of operations $+$ and $\oplus$ on intervals, whereas the latter follow from the fact that $\varrho \subseteq [t-t_\Pi, t_\mathsf{next}]$ and $t+\mathsf{step} \leq t_\mathsf{next}$ (due to Lines 6–10 of Algorithm 1). As a result, there are $\frac{t_\Pi+\mathsf{step}}{\gcd(\mathbb{T}_\mathcal{S} \cup \mathsf{N} \cup \{\mathsf{step}\})} + 1$ possible endpoints of ϱ, and thus ϱ is one of at most $4 \cdot \left(\frac{t_\Pi+\mathsf{step}}{\gcd(\mathbb{T}_\mathcal{S} \cup \mathsf{N} \cup \{\mathsf{step}\})} + 1\right)^2$ intervals, where factor 4 takes into account whether the interval is left/right open or closed. Then, the bound from the theorem follows. □

Since each application of a rule in Π introduces at least one new fact, the bound from Theorem 12 also provides a bound on the number of iterations of the main loop of ApplyRules, each time this procedure is called by Algorithm 1. In general, however, Theorem 12 does not provide a finite bound on the memory size. In particular, the algorithm may consume unbounded memory if the input stream $\mathcal{S}$ is such that the object domain $\mathbb{O}_\mathcal{S}$ has infinitely many elements, or the temporal domain $\mathbb{T}_\mathcal{S}$ has no gcd. In many practical scenarios we may assume that the object domain of all streams is finite (e.g., the set of nodes and monitored signals in our running example is finite over time) and that the timestamps 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 memory requirements; furthermore, gcd may not be known to the designer of the query, which makes it difficult to estimate memory efficiency 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 Algorithm 1, so that it will enjoy this desirable property for a particular class of programs.

6. Stream-independent memory guarantees

In this section we present a stream reasoning algorithm with bounded memory usage for all input streams. This guarantee, however, is only provided for stream queries not mentioning punctual intervals in rules. This is in contrast to Algorithm 1, where memory bounds apply to all forward-propagating queries, but only under certain assumptions on the input stream.

Let us define the *granularity* $\mathsf{gran}(\Pi)$ of a program Π as the minimal length of the intervals mentioned in Π (or ∞ if Π does not mention any interval). Hence, programs with non-zero granularity are exactly the programs which do not mention punctual intervals. Let Algorithm 6 be obtained from Algorithm 1 by replacing

- the call to ApplyRules in Line 11 with a call to the procedure ApplyRules* given in Algorithm 4, and
- the call to Forget in Line 16 with a call to the procedure Forget* given in Algorithm 5.

Our modifications exploit the following observations:

1. Facts in $\mathcal{W}$ holding at 'short' intervals cannot be used to derive new information using rules mentioning $\boxminus$ in the body. In particular, a rule $M' \leftarrow \boxminus_{\varrho_2} M$ does not apply to $M@\varrho_2$ in $\mathcal{W}$ if $|\varrho_1| < |\varrho_2|$.
2. Let M be an atom that holds in $\mathcal{W}$ in a set X of intervals and consider the application of a rule of the form $M' \leftarrow \diamondminus_{\varrho_2} M$ to $\mathcal{W}$. If the gaps between intervals in X are smaller than $\mathsf{gran}(\Pi)$, then we can equivalently apply the rule to a single fact $M@\varrho$ with ϱ the minimal interval containing all intervals in X.

Motivated by the second observation, we introduce a new type of fact in $\mathcal{W}$ of the form $M\#\varrho$. Intuitively, a fact $M\#\varrho$ holds if ground atom M is true 'sufficiently often' in the interval ϱ. In particular, if $M\#\varrho \in \mathcal{W}$, then for each $t' \in \varrho$ there is a closed interval $\varrho_1 \subseteq \varrho$ with $t' \in \varrho'$ and $|\varrho_1| < \mathsf{gran}(\Pi)$, such that $(\Pi, \mathcal{S}) \models M@\varrho_1^-$ and $(\Pi, \mathcal{S}) \models M@\varrho_1^+$. Clearly, if $M@\varrho \in \mathcal{W}$ and $\mathsf{gran}(\Pi) \neq 0$ then $M\#\varrho$ holds in $\mathcal{W}$, but the converse does not always hold.

Procedure ApplyRules* (c.f. Algorithm 4) extends Algorithm 2 with new derivation rules capturing the semantics of this new type of facts. In particular, Lines 3 and 4 generate a fact $M\#\varrho$ for each fact of the form $M@\varrho$ in the window memory; Lines 5–7 describe the application of rules $M' \leftarrow \diamondminus_{\varrho_2} M$ to facts $M\#\varrho$; whereas Lines 8–10 coalesce facts of the form $M\#\varrho$.

Algorithm 4: ApplyRules*

Input: A forward-propagating program Π in normal form, a set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_next
Output: A set of facts

1 **repeat**
2 Execute Lines 2–16 from Algorithm 2;
3 **for** *each* $M@\varrho \in \mathcal{W}$ **do**
4 Add $M\#\varrho$ to $\mathcal{W}$;
5 **for** *each* $M' \leftarrow \diamondminus_{\varrho_2} M$ *in* $\mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$ **do**
6 **for** *each* $M\#\varrho_1 \in \mathcal{W}$ *such that* $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$ *is non-empty* **do**
7 Add $M'@\varrho$ to $\mathcal{W}$;
8 **for** *each* $M\#\varrho_1, M\#\varrho_2 \in \mathcal{W}$ **do**
9 **if** $\varrho_1 \cup \varrho_2$ *is an interval or* $\varrho_1^+ \leq \varrho_2^-$ *and* $\varrho_2^- - \varrho_1^+ < \mathsf{gran}(\Pi)$ **then**
10 Add $M\#\varrho$ to $\mathcal{W}$, where ϱ is the minimal interval containing ϱ_1 and ϱ_2;
11 **until** *no more elements are added to* $\mathcal{W}$;
12 **return** $\mathcal{W}$;

Algorithm 5: Forget*

Input: A set $\mathcal{W}$ of facts, a set $\mathcal{H}$ of atoms, and time points t, t_Π
Output: A pair consisting of a set of facts and a set of atoms

1 Execute Lines 1–9 from Algorithm 3;
2 **for** *each* $M@\varrho \in \mathcal{W}$ **do**
3 **if** $\varrho^+ < t$ *and* $|\varrho| < \mathsf{gran}(\Pi)$ **then**
4 Delete $M@\varrho$ from $\mathcal{W}$;
5 **for** *each* $M\#\varrho \in \mathcal{W}$ **do**
6 **if** $\varrho \not\subseteq [t-t_\Pi, t]$ **then**
7 Add M to $\mathcal{H}$;
8 **if** $\varrho \cap [t-t_\Pi, t] = \emptyset$ **then**
9 Delete $M\#\varrho$ from $\mathcal{W}$;
10 **else**
11 Replace $M\#\varrho$ by $M\#\varrho \cap [t-t_\Pi, t]$ in $\mathcal{W}$;
12 **for** *each* $M\#\varrho_1, M\#\varrho_2 \in \mathcal{W}$ *with* $\varrho_1 \subsetneq \varrho_2$ **do**
13 Delete $M\#\varrho_1$ from $\mathcal{W}$;
14 **return** $(\mathcal{W}, \mathcal{H})$;

Analogously, procedure Forget* (c.f. Algorithm 5) extends Algorithm 3. In Lines 2–4, the algorithm deletes from $\mathcal{W}$ all facts

$M@\varrho$ where interval ϱ is 'too short', as these facts cannot be used to derive new information. Then Lines 5–12 forget facts of the form $M\#\varrho$, by adding an atom to $\mathcal{H}$ and either removing the fact or shortening its validity interval, in the same way as Lines 1–9 in Algorithm 3 forget facts of the form $M@\varrho$.

Correctness of our modified algorithm is established by Theorems 13 and 14, which are provided next.

Theorem 13 (*Soundness*)**.** *Each fact streamed out by Algorithm 6 is in the answer to* (Π, Q) *over* $\mathcal{S}$ *relative to* $\mathbb{T}$.

Proof. As in Theorem 10, we show that $Q(\mathbf{c})@\varrho \in \mathcal{W}$ implies $(\Pi, \mathcal{S}) \models Q(\mathbf{c})@\varrho$. For this, we show that the following statements hold for all relational atoms M and all intervals ϱ at each step in the execution of the algorithm, where t represents the current time point being processed.

1. If $M@\varrho \in \mathcal{W}$, then $(\Pi, \mathcal{S}) \models M@\varrho$.
2. If $M \in \mathcal{H}$, then $(\Pi, \mathcal{S}) \models M@t'$ for some $t' < t - t_\Pi$.
3. If $M\#\varrho \in \mathcal{W}$, then for each $t' \in \varrho$ there exist $t_1, t_2 \in \varrho$ such that $t_1 \leq t' \leq t_2$, $t_2 - t_1 < \mathsf{gran}(\Pi)$, $(\Pi, \mathcal{S}) \models M@t_1$, and $(\Pi, \mathcal{S}) \models M@t_2$.

As in the proof of Theorem 10, we proceed inductively on the number of computations adding new elements to $\mathcal{W}$ or $\mathcal{H}$, where the base case holds trivially as both $\mathcal{W}$ and $\mathcal{H}$ are initially empty. To show the inductive step, it suffices to show that the new computations from Algorithms 4 and 5 (which were not present in Algorithms 2 and 3, respectively) preserve Statements 1–3.

Inductive step for Statement 1: The only new computations adding facts $M'@\varrho$ to $\mathcal{W}$ are in Line 7 of Algorithm 4. The fact $M'@\varrho$ is added to $\mathcal{W}$ if $M\#\varrho_1 \in \mathcal{W}$ and if there exists $M' \leftarrow \diamondsuit_{\varrho_2} M \in \mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$ such that $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$ is non-empty. We will show that $(\Pi, \mathcal{S}) \models M'@t'$, for each $t' \in \varrho_1 + \varrho_2$. If $(\Pi, \mathcal{S}) \models M@t''$ for some t'' in $\varrho'' = \{t'' \mid t' \in t'' + \varrho_2\}$, then, by $M' \leftarrow \diamondsuit_{\varrho_2} M$, we have $(\Pi, \mathcal{S}) \models M'@t'$. Thus, it suffices to show $(\Pi, \mathcal{S}) \models M@t''$ for some $t'' \in \varrho''$. Observe that $|\varrho''| \geq \mathsf{gran}(\Pi)$. Hence, if $\varrho'' \subseteq \varrho_1$, by the inductive assumption for Statement 3 and $M\#\varrho_1 \in \mathcal{W}$, there exist $t_1, t_2, \in \varrho''$ such that $(\Pi, \mathcal{S}) \models M@t_1$ and $(\Pi, \mathcal{S}) \models M@t_2$. Assume that $\varrho'' \not\subseteq \varrho_1$. By construction, $\varrho'' \cap \varrho_1 \neq \emptyset$; thus, ϱ'' contains a left-most or a right-most fragment of ϱ_1. Hence, by the inductive assumption for Statement 3 and $M\#\varrho_1 \in \mathcal{W}$, there exists $t'' \in \varrho''$ such that $(\Pi, \mathcal{S}) \models M@t''$.

Inductive step for Statement 2: The only new computations extending $\mathcal{H}$ are in Line 7 of Algorithm 5, where M is added whenever $M\#\varrho \in \mathcal{W}$ for some ϱ with $\varrho^- < t - t_\Pi$. By the inductive assumption for Statement 3 and $M\#\varrho \in \mathcal{W}$, there exists $t' \in \varrho$ such that $t' < t - t_\Pi$ and $(\Pi, \mathcal{S}) \models M@t'$.

Inductive step for Statement 3: Facts of the form $M\#\varrho$ can be added to $\mathcal{W}$ in Lines 4 and 10 of Algorithm 4, or in Line 11 of Algorithm 5. If $M\#\varrho$ is added to $\mathcal{W}$ in Line 4 of Algorithm 4, then, by the inductive assumption for Statement 1, we have $(\Pi, \mathcal{S}) \models M@\varrho$, which implies the inductive step. If $M\#\varrho$ is added to $\mathcal{W}$ in Line 10 of Algorithm 4, then there are $M\#\varrho_1, M\#\varrho_2 \in \mathcal{W}$ such that ϱ is the minimal interval containing ϱ_1 and ϱ_2, and moreover, $\varrho_1 \cup \varrho_2$ is an interval, or $\varrho_1^+ \leq \varrho_2^-$ and $\varrho_2^- - \varrho_1^+ < \mathsf{gran}(\Pi)$. In both cases, by the inductive assumption for Statement 3 and the fact that $M\#\varrho_1, M\#\varrho_2 \in \mathcal{W}$, we obtain that the inductive step holds. Finally, if $M\#\varrho \cap [t - t_\Pi, t]$ is added to $\mathcal{W}$ in Line 11 of Algorithm 5, then $M\#\varrho \in \mathcal{W}$. By the inductive assumption for the Statement 3 and the fact that $\varrho \cap [t - t_\Pi, t] \subseteq \varrho$, the inductive step holds. □

Theorem 14 (*Completeness*)**.** *Each fact in the answer to* (Π, Q) *over* $\mathcal{S}$ *relative to* $\mathbb{T}$ *is streamed out by Algorithm 6.*

Proof. To represent contents of $\mathcal{W}$ and $\mathcal{H}$ in a run of Algorithm 6, we use notation $\mathcal{W}(t', \ell)$ and $\mathcal{H}(t', \ell)$ as in the proof of Theorem 11. We show by transfinite induction on ordinals α that, for each relational fact $M'@t'$, if $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) \models M'@t'$, there exists $M'@\varrho$ in $\mathcal{W}(t', 11)$ such that $t' \in \varrho$. The basis of induction and the inductive step for limit ordinals hold by the same arguments as in the proof of Theorem 11, so it suffices to show the inductive step for successor ordinals.

Inductive step for successor ordinal α*:* Let us assume that $T_\Pi^\alpha(\mathfrak{I}_\mathcal{S}) \models M'@t'$, but $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \not\models M'@t'$. Hence, there need to exist a ground rule $r \in \mathsf{ground}(\Pi, \mathcal{W} \cup \mathcal{H})$ of one of Forms (4)–(6), such that the body of r holds at t' in $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S})$ and the head of r is M'.

We assume that r is of the form $M' \leftarrow \diamondsuit_{\varrho_2} M$. Hence, there exists t'' such that $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \models M@t''$ and $t' \in t'' + \varrho_2$. Thus, by the inductive assumption, $M@\varrho'' \in \mathcal{W}(t'', 11)$, for some ϱ'' with $t'' \in \varrho''$. Moreover, by Line 4 of Algorithm 4, we have $M\#\varrho'' \in \mathcal{W}(t'', 11)$. The algorithm's execution ensures that either *(i)* $t'' \in [t - t_\Pi, t_{\mathsf{next}})$ and there exists $M\#\varrho_1 \in \mathcal{W}(t', 10)$ with $t'' \in \varrho_1$, or *(ii)* $t'' < t$, interval ϱ_2 is unbounded, and there exists $M \in \mathcal{H}(t', 10)$. In the first case, by Line 7 of Algorithm 4, we have $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (\varrho_1 + \varrho_2) \cap (t, t_{next}]$. In the latter case, by Line 7 of Algorithm 2, we obtain that $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (t, t_{\mathsf{next}}]$. Since $t' \in (t, t_{\mathsf{next}}]$, we have $t' \in \varrho$, as required.

Next, assume that r is of the form $M' \leftarrow \boxminus_{\varrho_2} M$. The proof for this case is similar to that for Theorem 11, but we argue that deleting from $\mathcal{W}$ facts over intervals shorter than $\mathsf{gran}(\Pi)$ (Lines 2–4 from Algorithm 5) does not invalidate the argumentation. We have $T_\Pi^{\alpha-1}(\mathfrak{I}_\mathcal{S}) \models \boxminus_{\varrho_2} M@\varrho'$, where ϱ' is such that $\varrho' \oplus \varrho_2 = t'$. Since Π is forward propagating, ϱ_2 needs to be bounded, so ϱ' is also bounded and $\varrho'^+ < t_\Pi$. Now, we use the inductive assumption to obtain that for each $t'' \in \varrho'$, there exists ϱ'', with $t'' \in \varrho''$ such that $M@\varrho'' \in \mathcal{W}(t'', 11)$. Furthermore, by coalescing the intervals ϱ'' in Lines 14–16 of Algorithm 2 and the fact that the right endpoint of the coalesced intervals is never smaller than the current time point t, we observe that facts with such coalesced intervals cannot be deleted in Lines 2–4 from Algorithm 5. Therefore, there exists ϱ_1 such that $\varrho' \subseteq \varrho_1$ and $M@\varrho_1 \in \mathcal{W}(t', 11)$. By Lines 8–10 of Algorithm 2, we obtain that $M'@\varrho \in \mathcal{W}(t', 11)$, for $\varrho = (\varrho_1 \oplus \varrho_2) \cap (t, t_{\mathsf{next}}]$. Since $\varrho_1 \oplus \varrho_2 = t'$ and $t' \in (t, t_{\mathsf{next}}]$, we get $t' \in \varrho$.

If r is of Form (6), then the argumentation is exactly the same as in the proof of Theorem 11. □

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

Theorem 15 (*Memory Bound*)**.** *At each step during the execution of Algorithm 6, on a normalised forward-propagating program* Π *such that* $\mathsf{gran}(\Pi) \neq 0$*, the number of elements in* $\mathcal{W} \cup \mathcal{H}$ *is bounded by*

$$9 \cdot \left(2\left(\frac{t_\Pi}{\mathsf{gran}(\Pi)} + 2\right) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}\right)^2 \cdot \mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A},$$

where R *is the number of rules in* Π*,* K *is the least non-zero endpoint of intervals mentioned in* Π*,* P *is the number of predicates in* Π*, and* A *is the maximum of their arities.*

Proof. Since $\mathcal{H}$ contains only ground atoms, it has at most $\mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A}$ elements. We next show a bound on the size of $\mathcal{W}$ which, in each iteration of Algorithm 6, has a maximal number of elements after executing ApplyRules*.

We start by considering the contents of $\mathcal{W}$ after executing Forget* in the end of the previous iteration of the algorithm. If $M@\varrho$ in $\mathcal{W}$, then $\varrho \subseteq [t - t_\Pi, t]$ by Lines 1–7 of Algorithm 3; moreover, by Lines 2–4 of Algorithm 5, we obtain that $|\varrho| \geq \mathsf{gran}(\Pi)$ unless $\varrho^+ = t$. Thus, there are at most $(t_\Pi / \mathsf{gran}(\Pi)) + 1$

such intervals ϱ. If $M\#\varrho \in \mathcal{W}$, then, by Lines 5–11 of Algorithm 5, we have $\varrho \subseteq [t - t_\Pi, t]$; furthermore, by Lines 8–10 of Algorithm 4 and Lines 12–13 of Algorithm 5, the minimal distance between ϱ and any $\varrho' \neq \varrho$ such that $M\#\varrho' \in \mathcal{W}$ is at least $\mathsf{gran}(\Pi)$, and so there are at most $(t_\Pi/\mathsf{gran}(\Pi)) + 1$ intervals ϱ. Hence, in total, $\mathcal{W}$ mentions at most $C = 2\,((t_\Pi/\mathsf{gran}(\Pi)) + 1)$ intervals after executing the procedure Forget*.

We next analyse the number of elements in $\mathcal{W}$ after executing ApplyRules*. Assume that $M@\varrho$ or $M\#\varrho$ was added to $\mathcal{W}$. Then, ϱ^- is of the form $x + k_1 \cdot a_1 + \cdots + k_n \cdot a_n$, where *(i)* x is t or t_{next}, or a left endpoint of an interval mentioned in $\mathcal{W}$ before executing ApplyRules*, *(ii)* $a_1, \ldots, a_n$ are the left endpoints of intervals ϱ_2 in rules $M' \leftarrow \diamondsuit_{\varrho_2} M$ and the right endpoints of intervals ϱ_2 mentioned in rules $M' \leftarrow \boxminus_{\varrho_2} M$, and *(iii)* $k_1, \ldots, k_n$ are any natural numbers. Since the number of intervals mentioned in $\mathcal{W}$ before executing ApplyRules* is at most C, we obtain that x can take at most $C + 2$ different values. We note that $n \leq \mathsf{R}$, each $a_i \geq \mathsf{K}$, and each $k_i \leq (t_\Pi + \mathsf{step})/\mathsf{K}$, since $\varrho^- \in [t - t_\Pi, t + \mathsf{step}]$. Therefore, the number of possible values of ϱ^- is bounded by $(C+2) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}$. By an analogous argument, we obtain the same bound on the number of possible values of ϱ^+. Since ϱ is determined by its endpoints and the types of its left and right brackets (there are 4 possible combinations of the brackets), the number of new intervals appearing in $\mathcal{W}$ after executing ApplyRules* is at most $4 \cdot \left((C+2) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}\right)^2$. Hence, the total number of intervals mentioned in $\mathcal{W}$ is at most $C + 4 \cdot \left((C+2) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}\right)^2$. Therefore the number of facts of the form $M@\varrho$ in $\mathcal{W}$ is bounded by

$$\left(C + 4 \cdot \left((C+2) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}\right)^2\right) \cdot \mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A}$$

and the same bound holds for facts of the form $M\#\varrho$ in $\mathcal{W}$. In total, the number of facts in $\mathcal{W} \cup \mathcal{H}$ is bounded by

$$2 \cdot \left(C + 4 \cdot \left((C+2) \cdot \mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}}\right)^2\right) \cdot \mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A} + \mathsf{P} \cdot |\mathbb{O}_\mathcal{S}|^\mathsf{A}.$$

By replacing C with its definition and given that $C > 2$ and $\mathsf{R}^{\frac{t_\Pi + \mathsf{step}}{\mathsf{K}}} \geq 1$ (Π is non-empty and $\mathsf{gran}(\Pi) \neq 0$) we can simplify this expression to the bound in the theorem. □

7. Experiments

We have implemented Algorithm 1 as an extension of the MeTeoR reasoner [25]. In this section, we refer to our stream reasoning extension as MeTeoR-Str.

We have conducted experiments on two benchmarks. The first one is based on the temporal extension of the Lehigh University Benchmark (LUBM) [31] introduced by Wang et al. [25], which provides a fixed DatalogMTL program and a generator of temporal datasets of various sizes. The second benchmark is based on a task provided during the Hackathon challenge at the 2021 Stream Reasoning Workshop.

All experiments were conducted on a Dell PowerEdge R730 server with 512 GB of RAM and two Intel Xeon E5-2640 2.6 GHz processors running Fedora 33, kernel 5.8.17. Note, however, that our implementation does not involve any multi-threading. The code and data are available online.[1]

7.1. Temporal LUBM benchmark

The extension of the LUBM benchmark by Wang et al. [25] provides a fixed DatalogMTL program and temporal datasets generator. The benchmark's program Π extends the 56 Datalog rules obtained from the OWL 2 RL fragment of the LUBM ontology with 29 temporal rules involving recursion and mentioning all metric operators in DatalogMTL.

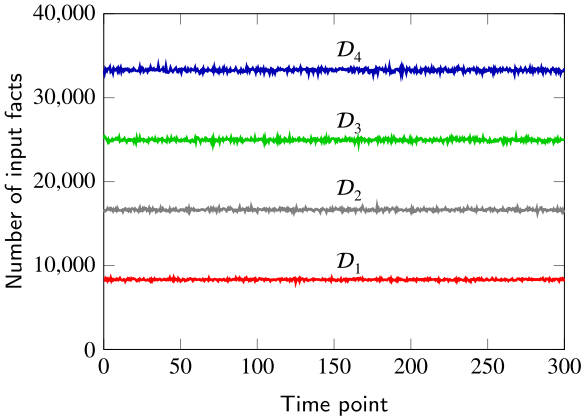

Fig. 1. Temporal distribution of facts in input streams.

Table 2
Maximal number of facts stored by MeTeoR and MeTeoR-Str.

		$\mathcal{D}_5$	$\mathcal{D}_{10}$	$\mathcal{D}_{15}$	$\mathcal{D}_{20}$
Π_{nr}	MeTeoR	41 612	83 196	124 717	165 812
	MeTeoR-Str	**1 006**	**2 015**	**2 841**	**3 738**
Π_{r}	MeTeoR	92 243	186 087	279 019	369 664
	MeTeoR-Str	**2 829**	**5 699**	**8 353**	**10 945**

Several rules in Π are not forward-propagating; hence, we chose two query predicates P_{nr} and P_r for which the relevant fragments of Π, namely Π_{nr} and Π_r, are non-recursive and recursive forward-propagating programs, respectively. We generated timestamped datasets using the data generator in the aforementioned temporal LUBM benchmark and assigned time points to facts from a range $[0, 300]$. In this way, we generated temporal datasets $\mathcal{D}_5$, $\mathcal{D}_{10}$, $\mathcal{D}_{15}$, and $\mathcal{D}_{20}$ consisting of 5, 10, 15, and 20 million facts, respectively. The distribution of facts in these datasets over various time points is depicted in Fig. 1.

For each dataset $\mathcal{D}_i$, with $i \in \{5, 10, 15, 20\}$, and each query (Π_j, P_j), with $j \in \{nr, r\}$, we computed the set of all query answers within the time interval $[0, 300]$. We did this in two different ways:

1. We passed program Π_j and the entire dataset $\mathcal{D}_i$ as an input to MeTeoR and computed all facts $P_j@t$, with $t \in [0, 300]$, which are entailed by Π_j and D_i.
2. We ran MeTeoR-Str, which implements Algorithm 1, on Π_j and dataset $\mathcal{D}_i$ 'sliced' as a stream, in which facts were passed on as input in 0.5 time increments.

In both cases, we reported memory usage as the maximal number of temporal facts stored in memory at any point during the execution of the algorithm.

The results are summarised in Table 2, and clearly suggest that MeTeoR-Str uses significantly less memory than MeTeoR (approximately a 40-fold reduction in memory consumption in most cases). This is thanks to the sliding window mechanism, which allows Algorithm 1 to keep in memory only a limited number of facts by forgetting 'old' facts that are no longer relevant.

The results from Table 2 also indicate that memory usage in Algorithm 1 increases linearly with the size of the input datasets $\mathcal{D}_5$–$\mathcal{D}_{20}$ passed on as streams. This is due to the fact that larger datasets were generated by increasing the density of the input

[1] https://github.com/wdimmy/StreamReasoningWithDatalogMTL.

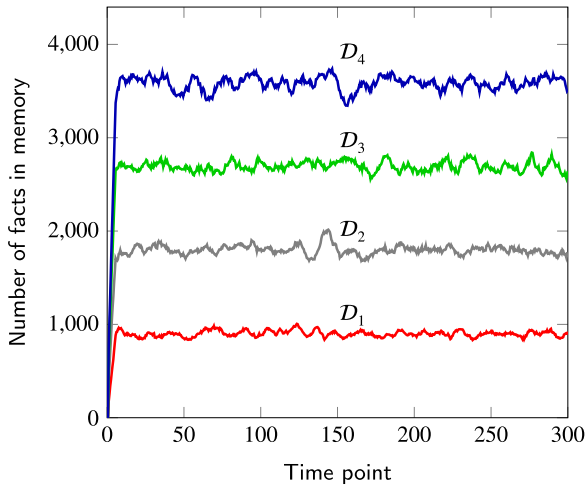

Fig. 2. Memory usage of MeTeoR-Str running on the non-recursive query (Π_{nr}, P_{nr}) and different input streams.

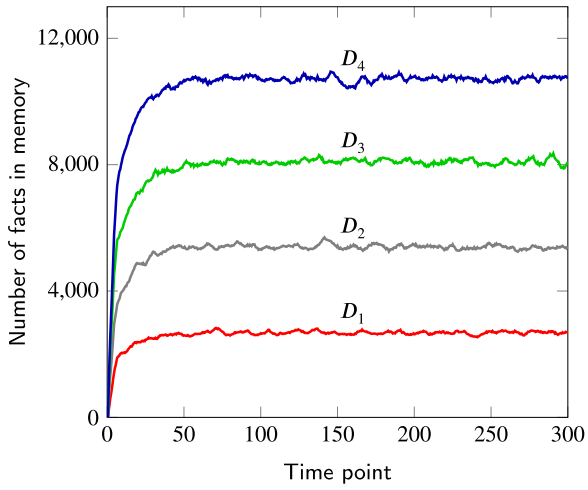

Fig. 3. Memory usage of MeTeoR-Str running on the recursive query (Π_r, P_r) and different input streams.

streams, rather than by expanding the overall temporal interval over which facts are generated. Indeed, we have monitored memory usage of Algorithm 1 as the stream is processed and the window slides accordingly; the results for the non-recursive and the recursive benchmark queries are given in Figs. 2 and 3, respectively. The figures show that, initially, memory consumption increases rapidly as the first window is populated and materialised and no facts have been forgotten yet; however, as soon as the algorithm starts forgetting old facts, memory consumption stabilises and remains roughly constant from then onwards. This is a crucial property of stream reasoning algorithms, and suggests that the algorithm is able to process datasets of *any size* in a streaming fashion.

7.2. Hackathon benchmark

The Hackathon Challenge, organised at the Stream Reasoning Workshop 2021[2] [32] provides a stream generator together with several reasoning tasks. We considered the scenario where input streams contain data from Eclipse Simulation of Urban MObility (SUMO)[3] describing road vehicles in a traffic jam, and the task is to detect vehicles that make a short stop (less than 5 s).

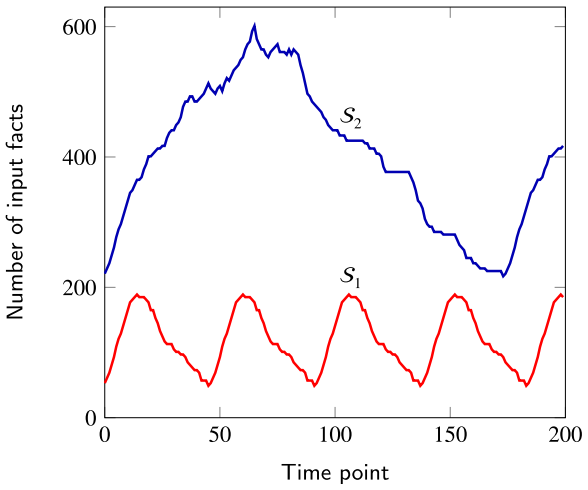

Fig. 4. Temporal distribution of facts in input streams.

We use input streams $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_2$ contains a significantly larger number of facts than $\mathcal{S}_1$. Both streams provide new data every second indicating vehicles' positions, speeds, accelerations, etc. We used the initial fragments of these streams for the first 200 s; in this fragment $\mathcal{S}_1$ contains 23,828 facts, whereas $\mathcal{S}_2$ contains 80,124 facts. The distribution of facts in $\mathcal{S}_1$ and $\mathcal{S}_2$ over various time points is depicted in Fig. 4.

To solve the task we have constructed a stream query $(\Pi, \mathit{ShortStop})$, with a DatalogMTL program Π consisting of the following rules, where $\mathit{Speed}_{=0}(x)$ and $\mathit{Speed}_{\neq 0}(x)$ mean that a vehicle x has currently a zero or non-zero speed, respectively, whereas $\mathit{NotOnMap}(x)$ indicates that a vehicle x is not on the map at the moment:

$$\begin{aligned}
\mathit{Speed}_{\neq 0}(x) &\leftarrow \mathit{NotOnMap}(x),\\
\mathit{ShortStop}(x) &\leftarrow \mathit{Speed}_{\neq 0}(x) \wedge \boxminus_1 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_2 \mathit{Speed}_{\neq 0}(x),\\
\mathit{ShortStop}(x) &\leftarrow \mathit{Speed}_{\neq 0}(x) \wedge \boxminus_1 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_2 \mathit{Speed}_{=0}(x) \wedge \boxminus_3 \mathit{Speed}_{\neq 0}(x),\\
\mathit{ShortStop}(x) &\leftarrow \mathit{Speed}_{\neq 0}(x) \wedge \boxminus_1 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_2 \mathit{Speed}_{=0}(x) \wedge \boxminus_3 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_4 \mathit{Speed}_{\neq 0}(x),\\
\mathit{ShortStop}(x) &\leftarrow \mathit{Speed}_{\neq 0}(x) \wedge \boxminus_1 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_2 \mathit{Speed}_{=0}(x) \wedge \boxminus_3 \mathit{Speed}_{=0}(x) \wedge \\
&\qquad \boxminus_4 \mathit{Speed}_{=0}(x) \wedge \boxminus_5 \mathit{Speed}_{\neq 0}(x).
\end{aligned}$$

By the first rule, vehicles not present on a map are treated as having non-zero speed (short stops are detected only for vehicles on the map). The remaining four rules define the concept of a short stop; in particular, they state that a vehicle makes a short stop if it has zero speed in exactly one, two, three, or four consecutive time points of the input stream.

We report reasoning times and memory consumption of MeTeoR-Str on Figs. 5 and 6, respectively. Our results on the Hackathon data are consistent with those on the temporal LUBM benchmark: memory consumption of MeTeoR-Str rapidly increases in the beginning until the first sliding window is populated with materialised facts, and then remains essentially stable

[2] https://streamreasoning.org/events/srw2021/.

[3] http://www.eclipse.org/sumo/.

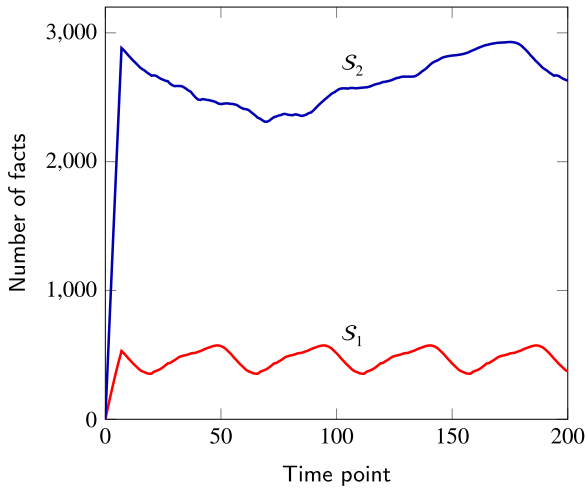

Fig. 5. Memory usage of MeTeoR-Str running on the query $(\Pi, \mathit{ShortStop})$ and the input streams $\mathcal{S}_1$ and $\mathcal{S}_2$.

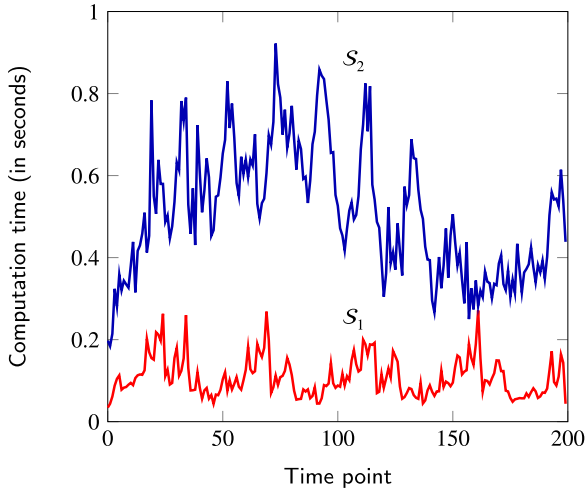

Fig. 6. Time consumption of MeTeoR-Str running on the query $(\Pi, \mathit{ShortStop})$ and the input streams $\mathcal{S}_1$ and $\mathcal{S}_2$.

with fluctuations depending on the number of facts arriving in the input streams (see Fig. 5). These fluctuations are much more visible than in the temporal LUBM benchmark since the distribution of facts in the input streams is much less uniform for the Hackathon benchmark (c.f. Fig. 4). For the same reasons, we can also observe fluctuations in reasoning times (Fig. 6). Note, however, that maximal reasoning time for each window remains below one second; since the input streams generate new facts each second, it means that MeTeoR-Str is able to perform reasoning in real time as data arrives.

8. Related work

In this section we discuss related work in the areas of stream query processing, temporal and stream reasoning, temporal deductive databases, and temporal model checking.

8.1. Query languages for stream processing

The foundations of stream query processing in databases were established by Arasu et al. [33], Babu and Widom [34] and Babcock et al. [35]. These works led to the development of CQL: an extension of SQL with specific window constructs for stream processing. CQL has since then become the basis for Semantic Web stream query languages [15,16,36–39].

8.2. Temporal extensions of datalog

There have been many proposals within the databases and KR research communities for extending Datalog with temporal constructs. In this context, the focus is typically on query answering over (static) temporal datasets, rather than on stream processing.

Datalog_{1S} [40] is an extension of Datalog, where predicates are equipped with an additional argument over terms of a special sort constructed using non-negative integer constants and a unary successor function symbol. In this setting, time points are represented as non-negative integers and the successor function symbol represents the 'next time point' relation. It turns out that Datalog_{1S} is equivalent in expressive power to TempLog—an extension of Datalog with temporal modal operators [41]. However, Datalog_{1S} is strictly less expressive than DatalogMTL; in particular, each Datalog_{1S} program can be equivalently rewritten as a DatalogMTL program using only diamond operators and punctual intervals and interpreted over the integer timeline [26]. Datalog_{1S} can also be seen as a fragment of *bi-directional ASP programs*, which extend Datalog with disjunction, negation-as-failure, and function symbols while preserving decidability of reasoning [42]. Despite its simplicity, however, fact entailment in Datalog_{1S} remains PSpace-complete in data complexity [40]. Datalog_{1S} is a core temporal KR language, which has been extended with various other temporal constructs, such as integer periodicity constraints [43]. We refer the reader to the work of Baudinet et al. [44] for an extensive survey on temporal deductive databases.

The complexity of query answering in DatalogMTL and its fragments under continuous semantics has been studied by Brandt et al. [12] and Wałęga et al. [13], who, respectively, showed that the problem is ExpSpace-complete for combined complexity and PSpace-complete for data complexity. Low complexity fragments of DatalogMTL have been studied by Wałęga et al. [45] and Brandt et al. [12] and alternative semantics with potentially more favourable computational properties have also been proposed by Wałęga et al. [26] and Ryzhikov et al. [27]. DatalogMTL has also been recently extended with stratified negation-as-failure [46] and subsequently with arbitrary negation under stable model semantics [47]. To the best of our knowledge, in addition to MeTeoR, there are two DatalogMTL reasoners available. The first one [12] is based on Ontop and supports only non-recursive rules; the second one is based on Vadalog, and does not ensure termination of reasoning [48]. These systems were not publicly available for testing at the time of writing.

8.3. Rule and ontology-based stream reasoning

Reasoning over streams in the presence of an OWL 2 ontology has been widely by studied [14–18]. OWL 2 ontologies are, however, non-temporal and hence, in contrast to DatalogMTL rules, ontological axioms are not well-suited for defining complex temporal events. Although temporal extensions of Description Logics have been proposed in the literature [49], to the best of our knowledge they have not been investigated in a streaming setting beyond the cases where the ontology can be compiled into the query using query rewriting techniques [17].

Stream reasoning has also been studied for Datalog_{1S} [19,20,22,24]. Similarly to our work, Ronca et al. [22,24] study a forward-propagating version of Datalog_{1S} and define an algorithm based on a sliding window that is conceptually similar to ours. Datalog_{1S}, however, does not allow for metric operators and

rules are evaluated over the natural numbers, which makes their technical results very different to ours.

The LARS framework [21] aims to unify various approaches to stream reasoning and defines an ASP language with (non-metric) modal temporal operators and windowing constructs. In contrast to our work, the LARS framework considers only reasoning over finite data (i.e., streams in LARS are intrinsically bounded), and windows are used to select finite fragments of streams. Reasoning over streams in LARS has been implemented in the Laser [50] and Ticker [51] systems; furthermore, distributed implementations have also been recently considered [52].

8.4. Related problems

The evaluation of MTL formulas over streams has received significant attention in the literature. Basin et al. [11] 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 [9] 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 [5–11]. 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. 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 [53], but become ExpSpace-complete if punctual intervals are disallowed [54].

Stream reasoning is also related to the problem of checking temporal integrity constraints in databases [30], where such constraints are typically expressed in an extension of first-order logic with past-only temporal operators; this is similar in spirit to our restriction to forward-propagating rules. Incremental update algorithms in the presence of temporal integrity constraints are also conceptually similar to reasoning stream reasoning algorithms since they also compute and store logical consequences for increasing time points while keeping in memory only a bounded number of temporal facts.

9. Conclusion and future work

In this paper, we have studied the problem of stream reasoning in the context of DatalogMTL and proposed and analysed two sound and complete algorithms which offer precise memory bound guarantees under different assumptions. We have implemented our techniques as an extension of the MeTeoR reasoner and obtained very encouraging empirical results, which supports the practical feasibility of our approach in streaming applications.

We see many avenues for future work. First, we will extend our algorithms to support the *since* operator $\mathcal{S}_\varrho$, and also consider extensions with stratified negation-as-failure. It would also be interesting to consider relaxing the syntactic restrictions of forward-propagating programs to allow for bounded (i.e., non-recursive) propagation of derived information towards past time points.

Second, our evaluation suggests the feasibility of our approach in practice. Therefore, we are planning to explore potential applications in close collaboration with our industrial partners at the SIRIUS Centre for Scalable Data Access and at Samsung Research UK.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Link to data is provided in the paper

Acknowledgements

This work was supported by the EPSRC, UK projects OASIS (EP/S032347/1), and UK FIRES (EP/S019111/1), as well as the SIRIUS Centre for Scalable Data Access, and Samsung Research UK. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission.

References

[1] G. Nuti, M. Mirghaemi, P.C. Treleaven, C. Yingsaeree, Algorithmic trading, IEEE Comput. 44 (11) (2011) 61–69.
[2] C. Cosad, K.J. Dufrene, K. Heidenreich, M. McMillon, A. Jermieson, M. O'Keefe, L. Simpson, Wellsite support from afar, Oilfield Rev. 21 (2) (2009) 48–58.
[3] G. Münz, G. Carle, Real-time analysis of flow data for network attack detection, in: Proc. of IM, 2007, pp. 100–108.
[4] F. Lécué, Deep dive on smart cities by scaling reasoning and interpreting the semantics of IoT, in: Proc. of EGC, 2017, pp. 7–8.
[5] F. Heintz, P. Doherty, DyKnow: An approach to middleware for knowledge processing, J. Intell. Fuzzy Systems 15 (1) (2004) 3–13.
[6] P. Thati, G. Roşu, Monitoring algorithms for metric temporal logic specifications, Electron. Notes Theor. Comput. Sci. 113 (2005) 145–162.
[7] P. Doherty, J. Kvarnström, F. Heintz, A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems, Auton. Agents Multi-Agent Syst. 19 (3) (2009) 332–377.
[8] D. Ničković, N. Piterman, From MTL to deterministic timed automata, in: Proc. of FORMATS, 2010, pp. 152–167.
[9] K. Baldor, J. Niu, Monitoring dense-time, continuous-semantics, metric temporal logic, in: Proc. of RV, 2012, pp. 245–259.
[10] H.-M. Ho, J. Ouaknine, J. Worrell, Online monitoring of metric temporal logic, in: Proc. of RV, 2014, pp. 178–192.
[11] D. Basin, F. Klaedtke, E. Zălinescu, Algorithms for monitoring real-time properties, Acta Inform. 55 (4) (2018) 309–338.
[12] S. Brandt, E.G. Kalaycı, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Querying log data with metric temporal logic, J. Artificial Intelligence Res. 62 (2018) 829–877.
[13] P.A. Wałęga, B. Cuenca Grau, M. Kaminski, E.V. Kostylev, DatalogMTL: Computational complexity and expressive power, in: Proc. of IJCAI, 2019, pp. 1886–1892.
[14] D.F. Barbieri, D. Braga, S. Ceri, E. Della Valle, M. Grossniklaus, C-SPARQL: SPARQL for continuous querying, in: Proc. of WWW, 2009, pp. 1061–1062.
[15] D. Anicic, P. Fodor, S. Rudolph, N. Stojanovic, EP-SPARQL: a unified language for event processing and stream reasoning, in: Proc. of WWW, 2011, pp. 635–644.
[16] D. Dell'Aglio, E.D. Valle, J. Calbimonte, Ó. Corcho, RSP-QL semantics: A unifying query model to explain heterogeneity of RDF stream processing systems, Int. J. Semantic Web Inf. Syst. 10 (4) (2014) 17–44.
[17] Ö.L. Özçep, R. Möller, C. Neuenstadt, A stream-temporal query language for ontology based data access, in: Proc. of KI, 2014, pp. 183–194.
[18] A. Margara, J. Urbani, F. van Harmelen, H.E. Bal, Streaming the Web: Reasoning over dynamic data, J. Web Semant. 25 (2014) 24–44.
[19] C. Zaniolo, Logical foundations of continuous query languages for data streams, in: Proc. of Datalog 2.0, 2012, pp. 177–189.
[20] A. Ronca, M. Kaminski, B. Cuenca Grau, B. Motik, I. Horrocks, Stream reasoning in temporal Datalog, in: Proc. of AAAI, 2018, pp. 1941–1948.
[21] H. Beck, M. Dao-Tran, T. Eiter, LARS: A logic-based framework for analytic reasoning over streams, Artificial Intelligence 261 (2018) 16–70.
[22] A. Ronca, M. Kaminski, B. Cuenca Grau, I. Horrocks, The delay and window size problems in rule-based stream reasoning, Artificial Intelligence 306 (2022).
[23] P.A. Wałęga, M. Kaminski, B. Cuenca Grau, Reasoning over streaming data in metric temporal Datalog, in: Proc. of AAAI, 2019, pp. 3092–3099.
[24] A. Ronca, M. Kaminski, B. Cuenca Grau, I. Horrocks, The window validity problem in rule-based stream reasoning, in: Proc. of KR, 2018, pp. 571–580.

[25] D. Wang, P. Hu, P.A. Wałęga, B.C. Grau, MeTeoR: Practical reasoning in datalog with metric temporal operators, in: Proc. of AAAI, 2022, pp. 5906–5913.
[26] P.A. Wałęga, B. Cuenca Grau, M. Kaminski, E.V. Kostylev, DatalogMTL over the integer timeline, in: Proc. of KR, 2020, pp. 768–777.
[27] V. Ryzhikov, P.A. Wałęga, M. Zakharyaschev, Data complexity and rewritability of ontology-mediated queries in metric temporal logic under the event-based semantics, in: Proc. of IJCAI, 2019, pp. 1851–1857.
[28] S. Brandt, R. Kontchakov, V. Ryzhikov, G. Xiao, M. Zakharyaschev, Ontology-based data access with a horn fragment of metric temporal logic, in: Proc. of AAAI, 2017, pp. 1070–1076.
[29] P.A. Wałęga, M. Zawidzki, B. Cuenca Grau, Finitely materialisable datalog programs with metric temporal operators, in: Proc. of KR, 2021, pp. 619–628.
[30] J. Chomicki, Efficient checking of temporal integrity constraints using bounded history encoding, ACM Trans. Database Syst. 20 (2) (1995) 149–186.
[31] Y. Guo, Z. Pan, J. Heflin, LUBM: a benchmark for OWL knowledge base systems, J. Web Semant. 3 (2–3) (2005) 158–182.
[32] P. Schneider, D. Alvarez-Coello, A. Le-Tuan, M. Nguyen-Duc, D. Le-Phuoc, Stream reasoning playground, in: Proc. of ESWC, 2022, pp. 406–424.
[33] A. Arasu, B. Babcock, S. Babu, J. McAlister, J. Widom, Characterizing memory requirements for queries over continuous data streams, ACM Trans. Database Syst. 29 (2004) 162–194.
[34] S. Babu, J. Widom, Continuous queries over data streams, SIGMOD Rec. 30 (3) (2001) 109–120.
[35] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proc. of PODS, 2002, pp. 1–16.
[36] J. Calbimonte, Ó. Corcho, A.J.G. Gray, Enabling ontology-based access to streaming data sources, in: Proc. of ISWC, Vol. 6496, 2010, pp. 96–111.
[37] A. Bolles, M. Grawunder, J. Jacobi, Streaming SPARQL: Extending SPARQL to process data streams, in: Proc. of ESWC, Vol. 5021, 2008, pp. 448–462.
[38] D.L. Phuoc, M. Dao-Tran, J.X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: Proc. of ISWC, Vol. 7031, 2011, pp. 370–388.
[39] D.F. Barbieri, D. Braga, S. Ceri, E.D. Valle, M. Grossniklaus, C-SPARQL: A continuous query language for RDF data streams, Int. J. Semant. Comput. 4 (1) (2010) 3–25.
[40] J. Chomicki, T. Imieliński, Temporal deductive databases and infinite objects, in: Proc. of PODS, 1988, pp. 61–73.
[41] M. Abadi, Z. Manna, Temporal logic programming, J. Symbolic Comput. 8 (3) (1989) 277–295.
[42] T. Eiter, M. Simkus, FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols, ACM Trans. Comput. Log. 11 (2) (2010) 14:1–14:50.
[43] D. Toman, J. Chomicki, Datalog with integer periodicity constraints, J. Log. Program. 35 (3) (1998) 263–290.
[44] M. Baudinet, J. Chomicki, P. Wolper, Temporal deductive databases, in: Temporal Databases: Theory, Design, and Implementation, 1993, pp. 294–320.
[45] P.A. Wałęga, B. Cuenca Grau, M. Kaminski, E.V. Kostylev, Tractable fragments of datalog with metric temporal operators, in: Proc. of IJCAI, 2020, pp. 1919–1925.
[46] D.J. Tena Cucala, P.A. Wałęga, B. Cuenca Grau, E.V. Kostylev, Stratified negation in datalog with metric temporal operators, in: Proc. of AAAI, 2021, pp. 6488–6495.
[47] P.A. Wałęga, D.J. Tena Cucala, E.V. Kostylev, B. Cuenca Grau, DatalogMTL with negation under stable models semantics, in: Proc. of KR, 2021, pp. 609–618.
[48] L. Bellomarini, L. Blasi, M. Nissl, E. Sallinger, The Temporal Vadalog System: Datalog-based reasoning for knowledge graphs, in: Proc. of RuleML + RR, 2022.
[49] A. Artale, R. Kontchakov, A. Kovtunova, V. Ryzhikov, F. Wolter, M. Zakharyaschev, Ontology-mediated query answering over temporal data: A survey (invited talk), in: Proc. of TIME, 2017, pp. 1:1–1:37.
[50] H.R. Bazoobandi, H. Beck, J. Urbani, Expressive stream reasoning with laser, in: Proc. of ISWC, Vol. 10587, 2017, pp. 87–103.
[51] H. Beck, T. Eiter, C. Folie, Ticker: A system for incremental ASP-based stream reasoning, Theory Pract. Log. Program. 17 (5–6) (2017) 744–763.
[52] T. Eiter, P. Ogris, K. Schekotihin, A distributed approach to LARS stream reasoning (system paper), Theory Pract. Log. Program. 19 (5–6) (2019) 974–989.
[53] R. Alur, T.A. Henzinger, Real-time logics: Complexity and expressiveness, Inform. and Comput. 104 (1) (1993) 35–77.
[54] R. Alur, T. Feder, T.A. Henzinger, The benefits of relaxing punctuality, J. ACM 43 (1) (1996) 116–146.