The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

# Stream Reasoning in Temporal Datalog

**Alessandro Ronca, Mark Kaminski,
Bernardo Cuenca Grau, Boris Motik, Ian Horrocks**
Department of Computer Science, University of Oxford, UK
{alessandro.ronca, mark.kaminski, bernardo.cuenca.grau, boris.motik, ian.horrocks}@cs.ox.ac.uk

**Abstract**

In recent years, there has been an increasing interest in extending traditional stream processing engines with logical, rule-based, reasoning capabilities. This poses significant theoretical and practical challenges since rules can derive new information and propagate it both towards past and future time points; as a result, streamed query answers can depend on data that has not yet been received, as well as on data that arrived far in the past. Stream reasoning algorithms, however, must be able to stream out query answers as soon as possible, and can only keep a limited number of previous input facts in memory. In this paper, we propose novel reasoning problems to deal with these challenges, and study their computational properties on Datalog extended with a temporal sort and the successor function—a core rule-based language for stream reasoning applications.

## 1 Introduction

Query processing over data streams is a key aspect of Big Data applications. For instance, algorithmic trading relies on real-time analysis of stock tickers and financial news items (Nuti et al. 2011); oil and gas companies continuously monitor and analyse data coming from their wellsites in order to detect equipment malfunction and predict maintenance needs (Cosad et al. 2009); network providers perform real-time analysis of network flow data to identify traffic anomalies and DoS attacks (Münz and Carle 2007).

In stream processing, an input data stream is seen as an unbounded, append-only, relation of timestamped tuples, where timestamps are either added by the external device that issued the tuple or by the stream management system receiving it (Babu and Widom 2001; Babcock et al. 2002). The analysis of the input stream is performed using a standing query, the answers to which are also issued as a stream. Most applications of stream processing require near real-time analysis using limited resources, which poses significant challenges to stream management systems. On the one hand, systems must be able to compute query answers over the partial data received so far as if the entire (infinite) stream had been available; furthermore, they must stream query answers out with the minimum possible delay. On the other hand, due to memory limitations, systems can only keep a limited *history* of previously received input facts in memory to perform computations. These challenges have been addressed by extending traditional database query languages with window constructs, which declaratively specify the finite part of the input stream relevant to the answers at the current time (Arasu, Babu, and Widom 2006).

In recent years, there has been an increasing interest in extending traditional stream management systems with logical, rule-based, reasoning capabilities (Barbieri et al. 2010; Calbimonte, Corcho, and Gray 2010; Anicic et al. 2011; Le-Phuoc et al. 2011; Zaniolo 2012; Özçep, Möller, and Neuenstadt 2014; Beck et al. 2015; Dao-Tran, Beck, and Eiter 2015). Rules can be very useful in stream processing applications for capturing complex analysis tasks in a declarative way, as well as for representing background knowledge about the application domain.

**Example 1.** *Consider a number of wind turbines scattered throughout the North Sea. Each turbine is equipped with a sensor, which continuously records temperature levels of key devices within the turbine and sends those readings to a data centre monitoring the functioning of the turbines. Temperature levels are streamed by sensors using a ternary predicate $Temp$, whose arguments identify the device, the temperature level, and the time of the reading. A monitoring task in the data centre is to track the activation of cooling measures in each turbine, record temperature-induced malfunctions and shutdowns, and identify parts at risk of future malfunction. This task is captured by the following set of rules:*

$$Temp(x, high, t) \rightarrow Flag(x, t) \quad (1)$$
$$Flag(x, t) \wedge Flag(x, t+1) \rightarrow Cool(x, t+1) \quad (2)$$
$$Cool(x, t) \wedge Flag(x, t+1) \rightarrow Shdn(x, t+1) \quad (3)$$
$$Shdn(x, t) \rightarrow Malfunc(x, t-2) \quad (4)$$
$$Shdn(x, t) \wedge Near(x, y) \rightarrow AtRisk(y, t) \quad (5)$$
$$AtRisk(x, t) \rightarrow AtRisk(x, t+1) \quad (6)$$

*Rule* (1) *'flags' a device whenever a high temperature reading is received. Rule* (2) *says that two consecutive flags on a device trigger cooling measures. Rule* (3) *says that an additional consecutive flag after activating cooling measures triggers a pre-emptive shutdown. By Rule* (4)*, a shutdown is due to a malfunction that occurred when the first flag leading to shutdown was detected. Finally, Rules* (5) *and* (6) *identify*

*devices located near a shutdown device as being at risk and propagate risk recursively into the future.*

The power and flexibility provided by rules poses additional challenges. As seen in our example, rules can derive information and propagate it both towards past and future time points. As a result, query answers can depend on data that has not yet been received (thus preventing the system from streaming out answers as soon as new input arrives), as well as on data that arrived far in the past (thus forcing the system to keep in memory a potentially large input history).

Towards developing a solid foundation for rule-based stream reasoning, we propose in Section 3 a suite of decision problems that can be exploited by a stream reasoning algorithm to deal with the aforementioned challenges.

- The *definitive time point* (DTP) problem is to check whether query answers to be issued at a given time $\tau_{out}$ will remain unaffected by any future input data given the current history; if so, $\tau_{out}$ is *definitive* and answers at $\tau_{out}$ can be safely output by the algorithm.
- The *forgetting* problem is to determine whether facts received at a given previous time point and recorded in the current history can be 'forgotten', in that they cannot affect future query answers. Forgetting allows the algorithm to maintain as small a history as possible.
- The *delay* problem is to check, given a time gap $d$, whether time point $\tau_{in} - d$ is definitive for each time point $\tau_{in}$ at which new input facts are received and each history up to $\tau_{in}$. Delay can thus be seen as a data-independent variant of DTP: the delay $d$ can be computed offline before receiving any data, and the algorithm can then safely output answers at $\tau_{in} - d$ as data at $\tau_{in}$ is being received.
- The *window size* problem is a data-independent variant of forgetting. The task is to determine, given a window size $s$, whether all history facts at time points up to $\tau_{in} - s$ can be forgotten for each time $\tau_{in}$ at which new input facts are received and each history up to $\tau_{in}$. A stream reasoning algorithm can compute $s$ in an offline phase and then, in the online phase, immediately delete all history facts older than $s$ time points as new data arrives.

In Section 4, we proceed to the study of the computational properties of the aforementioned problems. For this, we consider as query language *temporal Datalog*—negation-free Datalog with a special temporal sort to which the successor function (or, equivalently, addition by a constant) is applicable (Chomicki and Imieliński 1988). This is a core temporal rule-based language, which captures other prominent temporal languages (Abadi and Manna 1989; Baudinet, Chomicki, and Wolper 1993) and forms the basis of more expressive formalisms for stream reasoning recently proposed in the literature (Zaniolo 2012; Beck et al. 2015).

We show in Section 4.1 that DTP is PSPACE-complete in data complexity and becomes tractable for nonrecursive queries under very mild additional restrictions; thus, DTP is no harder than query evaluation (Chomicki and Imieliński 1988). In Section 4.2, we show that forgetting is undecidable; however, quite surprisingly, the aforementioned restrictions to nonrecursive queries allows us to regain not only decidability, but also tractability in data complexity. In Section 4.3, we turn our attention to data-independent problems. We show that both delay and window size are undecidable in general and become co-NEXP-complete for nonrecursive queries.

Our results show that, although stream reasoning problems are either intractable in data complexity or undecidable in general, they become feasible in practice for nonrecursive queries under very mild additional restrictions. On the one hand, the data-dependent problems (DTP and forgetting) become tractable in data complexity (a very important requirement for achieving near real-time computation in practice); on the other hand, although the data-independent problems (delay and window size) remain intractable, these are one-time problems which only need to be solved once prior to receiving any input data.

The proofs of all results are given in an extended version of this paper (Ronca et al. 2017).

## 2 Preliminaries

**Syntax** A vocabulary consists of *predicates*, *constants* and *variables*, where constants are partitioned into *objects* and integer *time points* and variables are partitioned into *object variables* and *time variables*. An *object term* is an object or an object variable. A *time term* is either a time point, a time variable, or an expression of the form $t + k$ where $t$ is a time variable, $k$ is an integer number, and $+$ is the standard *integer addition function*. The offset $\Delta(s)$ of a time term $s$ equals zero if $s$ is a time variable or a time point and it equals $k$ if $s$ is of the form $s = t + k$.

Predicates are partitioned into *extensional* (EDB) and *intensional* (IDB) and they come with a nonnegative integer *arity* $n$, where each position $1 \leq i \leq n$ is of either *object* or *time sort*. A predicate is *rigid* if all its positions are of object sort and it is *temporal* if the last position is of time sort and all other positions are of object sort. An *atom* is an expression $P(t_1, \ldots, t_n)$ where $P$ is a predicate and each $t_i$ is a term of the required sort. A *rigid* atom (respectively, temporal, IDB, EDB) is an atom involving a rigid predicate (respectively, temporal, IDB, EDB).

A *rule* $r$ is of the form $\bigwedge_i \alpha_i \rightarrow \alpha$, where $\alpha$ and each $\alpha_i$ are rigid or temporal atoms, and $\alpha$ is IDB whenever $\bigwedge_i \alpha_i$ is non-empty. Atom $\mathrm{head}(r) = \alpha$ is the *head* of $r$, and $\mathrm{body}(r) = \bigwedge_i \alpha_i$ is the *body* of $r$. Rules are assumed to be *safe*: each head variable must occur in the body. An *instance* $r'$ of $r$ is obtained by applying a substitution to $r$. A *program* $\Pi$ is a finite set of rules. Predicate $P$ is $\Pi$-*dependent* on a predicate $P'$ if there is a rule of $\Pi$ with $P$ in the head and $P'$ in the body. The rank $\mathrm{rank}(P, \Pi)$ of $P$ w.r.t. $\Pi$ is 0 if $P$ does not occur in head position in $\Pi$, and is the maximum of the values $\mathrm{rank}(P') + 1$ for $P'$ a predicate such that $P$ is $\Pi$-dependent on $P'$ otherwise. We write $\mathrm{rank}(P)$ for $\mathrm{rank}(P, \Pi)$ if $\Pi$ is clear from the context. The rank $\mathrm{rank}(\Pi)$ of $\Pi$ is the maximum rank of a predicate in $\Pi$.

A *query* is a pair $Q = \langle P_Q, \Pi_Q \rangle$ where $\Pi_Q$ is a program and $P_Q$ is an IDB predicate in $\Pi_Q$; query $Q$ is *temporal* (*rigid*) if $P_Q$ is a temporal (rigid) predicate. A term, atom, rule, or program is *ground* if it contains no variables. A *fact* $\alpha$ is a ground, function-free rigid or temporal atom; every

fact $\alpha$ corresponds to a rule of the form $\top \rightarrow \alpha$ where $\top$ denotes the empty conjunction, so we use $\alpha$ and $\top \rightarrow \alpha$ interchangeably. A *dataset* $D$ is a program consisting of EDB facts. The *$\tau$-segment* $D[\tau]$ of dataset $D$ is the subset of $D$ containing all rigid facts and all temporal facts with time argument $\tau' > \tau$.

A program $\Pi$ (respectively, query $Q$) is: *Datalog* if no temporal predicate occurs in $\Pi$ (in $\Pi_Q$); and *nonrecursive* if the directed graph induced by the $\Pi$-dependencies ($\Pi_Q$-dependencies) is acyclic.

**Semantics and standard reasoning** Rules are interpreted in the standard way as universally quantified first-order sentences. A Herbrand interpretation $\mathcal{H}$ is a (possibly infinite) set of facts. Interpretation $\mathcal{H}$ *satisfies* a rigid atom $\alpha$ if $\alpha \in \mathcal{H}$, and it satisfies a temporal atom $\alpha$ if evaluating the addition function in $\alpha$ yields a fact in $\mathcal{H}$. The notion of satisfaction is extended to conjunctions of ground atoms, rules and programs in the standard way. If $\mathcal{H} \models \Pi$, then $\mathcal{H}$ is a *model* of $\Pi$. Program $\Pi$ *entails* a fact $\alpha$, written $\Pi \models \alpha$, if $\mathcal{H} \models \Pi$ implies $\mathcal{H} \models \alpha$. The *answers* to a query $Q$ over a dataset $D$, written $Q(D)$, are the tuples $\mathbf{a}$ of constants such that $\Pi_Q \cup D \models P_Q(\mathbf{a})$. If $Q$ is a temporal query, we denote with $Q(D, \tau)$ the subset of answers in $Q(D)$ referring to time point $\tau$. Given an input query $Q$, dataset $D$ and tuple $\mathbf{a}$, the *query evaluation problem* is to check whether $\mathbf{a}$ is an answer to $Q$ over $D$; the *data complexity* of query evaluation is the complexity when $Q$ is considered fixed. Finally, a query $Q_1$ is *contained* in a query $Q_2$, written $Q_1 \sqsubseteq Q_2$, if $Q_1(D) \subseteq Q_2(D)$ for every dataset $D$. Given input queries $Q_1$ and $Q_2$, the *query containment problem* is to check whether $Q_1$ is contained in $Q_2$.

**Complexity** Query evaluation is PSPACE-complete in data complexity assuming that numbers are coded in unary (Chomicki and Imieliński 1988). Data complexity drops to the circuit class $AC^0$ for nonrecursive programs. By standard results in nontemporal Datalog, containment of temporal queries is undecidable (Shmueli 1993). Furthermore, it is co-NEXP-hard for nonrecursive queries (Benedikt and Gottlob 2010).

## 3 Stream Reasoning Problems

A stream reasoning algorithm receives as input a query $Q$ and a stream of temporal EDB facts, and produces as output a stream of answers to $Q$. Both input facts and query answers are processed by increasing value of their timestamps, where $\tau_{in}$ and $\tau_{out}$ represent the current times at which input facts are received and query answers are being streamed out, respectively. Answers at $\tau_{out}$ are only output when the algorithm can determine that they cannot be affected by future input facts. In turn, input facts received so far are kept in a *history dataset* $D$ since future query answers can be influenced by facts received at an earlier time; practical systems, however, have limited memory and hence the algorithm must also *forget* facts in the history as soon as it can determine that they will not influence future query answers.

Algorithms 1 and 2 provide two different realisations of such a stream reasoning algorithm, which we refer to as *online* and *offline*, respectively.

**Algorithm 1:** 'Online' Stream Reasoning Algorithm

```
   Parameters: Temporal query Q
1  D := ∅, τ_in := 0, τ_out := 0, τ_mem := 0
2  loop
3      receive facts U that hold at τ_in and set D := D ∪ U
4      while τ_out ≤ τ_in and τ_out is definitive for Q, D, τ_in
         do
5          stream output Q(D, τ_out)
6          τ_out := τ_out + 1
7      end
8      while τ_mem < τ_out and
         τ_mem is forgettable for Q, D, τ_in, τ_out do
9          forget all temporal facts in D holding at τ_mem
10         τ_mem := τ_mem + 1
11     end
12     τ_in := τ_in + 1
13 end
```

The *online algorithm* (see Algorithm 1) decides which answers to stream and which history facts to forget 'on the fly' as new input data arrives. The algorithm records the latest time point $\tau_{out}$ for which answers have not yet been streamed; as $\tau_{in}$ increases and new data arrives, the algorithm checks (lines 4-7) whether answers at $\tau_{out}$ can now be streamed and, if so, it continues incrementing $\tau_{out}$ until it finds a time point for which answers cannot be provided yet. This process relies on deciding whether the considered $\tau_{out}$ are *definitive*—that is, the answers to $Q$ at $\tau_{out}$ for the history $D$ will remain stable even if $D$ were extended with an unknown (and thus arbitrary) set $U$ of future input facts.

**Definition 1.** *A $\tau_{in}$-*history *$D$ is a dataset consisting of rigid facts and temporal facts with time argument at most $\tau_{in}$. A $\tau_{in}$-*update *$U$ is a dataset consisting of temporal facts with time argument strictly greater than $\tau_{in}$.*

**Definition 2.** *An instance $I$ of the* Definitive Time Point *(*DTP*) problem is a tuple $\langle Q, D, \tau_{in}, \tau_{out} \rangle$, with $Q$ a temporal query, $D$ a $\tau_{in}$-history and $\tau_{out} \leq \tau_{in}$.* DTP *holds for $I$ iff $Q(D, \tau_{out}) = Q(D \cup U, \tau_{out})$ for each $\tau_{in}$-update $U$.*

**Example 2.** *Consider Example 1, and suppose we are interested in determining the time points at which a turbine malfunctions. Thus, let the query $Q$ have output predicate $Malfunc$ and include rules* (1)–(4) *together with rule*

$$Temp(x, na, t) \rightarrow Malfunc(x, t)$$

*which defines an invalid reading as a malfunction. For a history $D$ consisting of the fact $Temp(a, high, 0)$, we have that* $\text{DTP}(Q, D, 0, 0)$ *is false, since $Q(D, 0)$ is empty and $Q(D \cup U, 0)$ is not if the update $U$ contains $Temp(a, high, 1)$ and $Temp(a, high, 2)$. For a history $D'$ consisting of the fact $Temp(a, na, 0)$, we have that* $\text{DTP}(Q, D', 0, 0)$ *is true, since $Q(D', 0)$ already includes the only possible answer.*

Algorithm 1 also records the latest time point $\tau_{mem}$ for which history facts have not yet been forgotten. As $\tau_{in}$ increases, the algorithm checks in lines 8-11 whether all history facts at time $\tau_{mem}$ can now be forgotten and, if so, it continues incrementing $\tau_{mem}$ until it finds a point where this is no longer possible. For this, the algorithm decides whether

**Algorithm 2:** 'Offline' Stream Reasoning Algorithm

**Parameters:** Temporal query $Q$

$D := \emptyset,\ \tau_{in} := 0$
compute minimal delay $d$ and minimal window size $s$ for $Q$
**loop**
  receive facts $U$ that hold at $\tau_{in}$ and set $D := D \cup U$
  **if** $\tau_{in} - d \geq 0$ **then** stream output $Q(D, \tau_{in} - d)$
  forget all temporal facts in $D$ holding at $\tau_{in} - s$
  $\tau_{in} := \tau_{in} + 1$
**end**

the relevant $\tau_{mem}$ are *forgettable*, in the sense that no future answer to $Q$ can be affected by the history facts at $\tau_{mem}$.

**Definition 3.** *An instance $I$ of* FORGET *is a tuple of the form $\langle Q, D, \tau_{in}, \tau_{out}, \tau_{mem}\rangle$, with $Q$ a temporal query, $D$ a $\tau_{in}$-history, and $\tau_{mem} \leq \tau_{out} \leq \tau_{in}$.* FORGET *holds for $I$ iff $Q(D \cup U, \tau) = Q(D[\tau_{mem}] \cup U, \tau)$ for each $\tau_{in}$-update $U$ and each time point $\tau \geq \tau_{out}$.*

**Example 3.** *Consider the query $Q$ with output predicate $Shdn$ and rules* (1)–(3) *from Example 1. For a history $D$ consisting of facts $Temp(a, high, 0)$ and $Temp(a, low, 1)$, we have that* FORGET$(Q, D, 1, 1, 1)$ *is true, since $Q(D[1] \cup U, 1)$ is empty for every 1-update $U$. For a history $D'$ containing facts $Temp(a, high, 0)$ and $Temp(a, high, 1)$, we have that* FORGET$(Q, D, 1, 1, 1)$ *is false, since $Q(D[1] \cup U, 1)$ is empty but $Q(D \cup U, 1)$ is not for the 1-update containing $Temp(a, high, 2)$.*

The *offline algorithm* (Algorithm 2) precomputes the minimum *delay* $d$ and *window size* $s$ for the standing query $Q$ in a way that is independent from the input data stream.

Intuitively, $d$ represents the smallest time gap needed to ensure that, for any input stream and any time point $\tau_{in}$, the time point $\tau_{out} = \tau_{in} - d$ is definitive; in other words, that it is always safe to stream answers with a delay $d$ relative to the currently processed input facts.

**Definition 4.** *An instance $I$ of* DELAY *is a pair $\langle Q, d\rangle$, with $Q$ a temporal query and $d$ a nonnegative integer.* DELAY *holds for $I$ iff $Q(D, \tau_{in} - d) = Q(D \cup U, \tau_{in} - d)$ for each time point $\tau_{in}$, each $\tau_{in}$-history $D$, and each $\tau_{in}$-update $U$.*

**Example 4.** *In Example 1, 0 is a valid delay for the $AtRisk$ and $Shdn$ queries, and so is 2 for the $Malfunc$ query.*

In turn, $s$ represents the size of the smallest time interval for which the history needs to be kept; in other words, for any input stream and any time point $\tau_{in}$, it is safe to forget all history facts with timestamp smaller than $\tau_{in} - s$.

**Definition 5.** *An instance $I$ of* WINDOW *is a triple $\langle Q, d, s\rangle$, with $Q$ a temporal query and $d$ and $s$ nonnegative integers.* WINDOW *holds for $I$ iff $Q(D \cup U, \tau_{out}) = Q(D[\tau_{in} - s] \cup U, \tau_{out})$ for all time points $\tau_{in}$ and $\tau_{out}$ with $\tau_{out} > \tau_{in} - d$, each $\tau_{in}$-history $D$, and each $\tau_{in}$-update $U$.*

**Example 5.** *Consider Example 1. Assuming that we want to evaluate queries with delay 0, a valid window size for the $Shdn$ query is 2; whereas the $AtRisk$ query has no valid window size (or, equivalently, the query requires a window of infinite size), since answers for that query can depend on facts arbitrarily far in the past.*

Once the delay $d$ and window size $s$ have been determined, they remain fixed during execution of the algorithm: indeed, as $\tau_{in}$ increases and new data arrives in each iteration of the main loop, Algorithm 2 simply streams query answers at $\tau_{in} - d$ and forgets all history facts at $\tau_{in} - s$. This is in contrast to the online approach, where the algorithm had to decide in each iteration of the main loop which answers to stream and which facts to forget.

# 4 Complexity of Stream Reasoning

We now start our investigation of the computational properties of the stream reasoning problems introduced in Section 3. For all problems, we consider both the general case applicable to arbitrary inputs and the restricted setting where the input queries are nonrecursive. All our results assume that numbers are coded in unary.

## 4.1 Definitive Time Point

Let $I = \langle Q, D, \tau_{in}, \tau_{out}\rangle$ be a fixed, but arbitrary, instance of DTP and denote with $\Omega_I$ the set consisting of all objects in $\Pi_Q \cup D$ and a fresh object $o_I$ unique to $I$.

As stated in Definition 2, DTP holds for $I$ if and only if the query answers at time $\tau_{out}$ over the history $D$ coincide with the answers at the same time point but over $D$ extended with an arbitrary $\tau_{in}$-update $U$. Note that, in addition to new facts over existing objects in $D$ and $Q$, the update $U$ may also include facts about new objects. The following proposition shows that, to decide DTP, it suffices to consider updates involving only objects from $\Omega_I$. Intuitively, updates containing fresh objects can be homomorphically embedded into updates over $\Omega_I$ by mapping all fresh objects to $o_I$.

**Proposition 1.** DTP *holds for $I$ iff $\mathbf{o} \in Q(D \cup U, \tau_{out})$ implies $\mathbf{o} \in Q(D, \tau_{out})$ for every tuple $\mathbf{o}$ over $\Omega_I$ and every $\tau_{in}$-update $U$ involving only objects in $\Omega_I$.*

**The general case.** We next show that DTP is decidable and provide tight complexity bounds. Our upper bounds are obtained by showing that, to decide DTP, it suffices to consider a single *critical update* and a slight modification of the query, which we refer to as the *critical query*. Intuitively, the critical update is a dataset that contains all possible facts at the next time point $\tau_{in} + 1$ involving EDB predicates from $Q$ and objects in $\Omega_I$. In turn, the critical query extends $Q$ with rules that propagate all facts in the critical update recursively into the future. The intention is that the answers to the critical query over $D$ extended with the critical update will capture the answers to $Q$ over $D$ extended with any arbitrary future update. In the following definition, we use $\psi$ to denote the renaming mapping each temporal EDB predicate to a fresh temporal IDB predicate of the same arity.

**Definition 6.** *Let $A$ be a fresh unary temporal EDB predicate. The* critical update $\Upsilon_I$ *for $I$ is the $\tau_{in}$-update containing the fact $A(\tau_{in} + 1)$, and all facts $P(\mathbf{o}, \tau_{in} + 1)$ for each temporal EDB predicate $P$ in $\Pi_Q$ and each tuple $\mathbf{o}$ over $\Omega_I$.*

*Let $V$ be a fresh unary temporal IDB predicate. The* critical query $\Theta_I$ *for $I$ is the query where $P_{\Theta_I} = P_Q$ and $\Pi_{\Theta_I}$ is obtained from $\psi(\Pi_Q)$ by adding rule $A(t) \rightarrow V(t)$, rule*

$V(t) \to V(t+1)$*, and the following rules for each temporal EDB predicate* $P$ *occurring in* $\Pi_Q$*, where* $P' = \psi(P)$*:*

$$P(\mathbf{x}, t) \to P'(\mathbf{x}, t)$$
$$V(t+1) \wedge P'(\mathbf{x}, t) \to P'(\mathbf{x}, t+1)$$

The construction of the critical query and update ensures, on the one hand, that $Q(D, \tau_{out}) = \Theta_I(D, \tau_{out})$ and, on the other hand, that $Q(D \cup U, \tau_{out}) \subseteq \Theta_I(D \cup \Upsilon_I, \tau_{out})$ for each $\tau_{in}$-update $U$ involving only objects in $\Omega_I$. We can exploit these properties, together with Proposition 1, to show that DTP can be decided by checking whether, at $\tau_{out}$, the answers to the critical query over $D$ remain the same if $D$ is extended with the critical update.

**Lemma 1.** DTP *holds for* $I$ *iff* $\mathbf{o} \in \Theta_I(D \cup \Upsilon_I, \tau_{out})$ *implies* $\mathbf{o} \in \Theta_I(D, \tau_{out})$ *for every tuple* $\mathbf{o}$ *over* $\Omega_I$*.*

It follows from Lemma 1 that, to decide DTP, we need to perform two temporal query evaluation tests for each candidate tuple $\mathbf{o}$. Since temporal query evaluation is feasible in PSPACE in data complexity, then so is DTP because the number of candidate tuples $\mathbf{o}$ is polynomial if $Q$ is fixed.

Furthermore, query evaluation is reducible to DTP, and hence the aforementioned PSPACE upper bound in data complexity is tight.

**Theorem 1.** DTP *is* PSPACE-complete *in data complexity.*

**Nonrecursive queries** We next show that DTP becomes tractable in data complexity for nonrecursive queries. In the remainder of this section, we fix an arbitrary instance $I = \langle Q, D, \tau_{in}, \tau_{out} \rangle$ of DTP, where $Q$ is nonrecursive. We assume w.l.o.g. that $Q$ does not contain rigid atoms: each such atom $P(\mathbf{c})$ can be replaced with a temporal atom of the form, e.g., $P'(\mathbf{c}, 0)$. We make the additional technical assumption that each rule in $Q$ is restricted as follows.

**Definition 7.** *A rule is* connected *if it contains at most one temporal variable, which occurs in the head whenever it occurs in the body. A query is connected if so are its rules.*

Restricting our arguments to connected queries allows us to considerably simplify definitions and proofs.

We start with the observation that the critical query of $I$ always includes recursive rules that propagate information arbitrarily far into the future (see Definition 6). As a result, our general algorithm for DTP does not immediately provide an improved upper bound in the nonrecursive case.

The need for such recursive rules, however, is motivated by the fact that the answers to a recursive query $Q$ at time $\tau_{out}$ may depend on facts at time points $\tau$ arbitrarily far from $\tau_{out}$; in other words, there is no bound $b$ for $I$ such that $|\tau - \tau_{out}| \leq b$ in every derivation of query answers at $\tau_{out}$ involving input facts at $\tau$. If $Q$ is nonrecursive, however, such a bound is guaranteed to exist and can be established based on the following notion of program *radius*. Intuitively, query answers at $\tau_{out} \leq \tau_{in}$ can only be influenced by future facts whose timestamp is located within the interval $[\tau_{in}, \tau_{out} + \mathrm{rad}(\Pi_Q)]$.

**Definition 8.** *Let* $r$ *be a connected rule mentioning a time variable. The* radius $\mathrm{rad}(r)$ *of* $r$ *is the maximum of the values* $|\Delta(s) - \Delta(s')|$ *for* $s$ *the time argument in the head of* $r$ *and* $s'$ *the time argument in a body atom of* $r$*. The* radius $\mathrm{rad}(\Pi)$ *of a connected program* $\Pi$ *is given by the number of rules in* $\Pi$ *multiplied by the maximum radius of a rule in* $\Pi$*.*

Thus, to show tractability of DTP, we identify a polynomially bounded number of *critical time points* using the radius of $\Pi_Q$ and argue that we can dispense with the aforementioned recursive rules by constructing a critical update for $I$ that includes all facts over these time points. Since $\Pi_Q$ may contain explicit time points in rules, these also need to be taken into account when defining the relevant critical time points and the corresponding critical update.

**Definition 9.** *Let* $\tau_0$ *be the maximum value between* $\tau_{out}$ *and the largest time point occurring in* $\Pi_Q$*.*

*A time point* $\tau$ *is* critical *for* $I$ *if* $\tau_{in} < \tau \leq \tau_0 + \mathrm{rad}(\Pi_Q)$*. The* bounded critical update $\Upsilon_I^{\mathrm{b}}$ *of* $I$ *consists of each fact* $P(\mathbf{o}, \tau)$ *with* $P$ *a temporal EDB predicate in* $\Pi_Q$*,* $\mathbf{o}$ *a tuple over* $\Omega_I$*, and* $\tau$ *a critical time point.*

The following lemma justifies the key property of the critical update $\Upsilon_I^{\mathrm{b}}$, namely that the answers to $Q$ over $D \cup \Upsilon_I^{\mathrm{b}}$ capture those over $D$ extended with any future update.

**Lemma 2.** *Let* $a$ *be the maximum radius of a rule in* $\Pi_Q$ *and let* $\mathbf{T}$ *consist of* $\tau_{out}$ *and the time points in* $\Pi_Q$*. If* $\Pi_Q \cup D \cup U \models P(\mathbf{o}, \tau)$ *for a* $\tau_{in}$*-update* $U$ *involving only objects in* $\Omega_I$ *and a predicate* $P$ *in* $\Pi_Q$*, and* $|\tau - \tau'| \leq a \cdot (\mathrm{rank}(\Pi_Q) - \mathrm{rank}(P))$ *for some* $\tau' \in \mathbf{T}$*, then* $\Pi_Q \cup D \cup \Upsilon_I^{\mathrm{b}} \models P(\mathbf{o}, \tau)$*.*

We are now ready to establish the analogue to Lemma 1 in the nonrecursive case.

**Lemma 3.** DTP *holds for* $I$ *iff* $\mathbf{o} \in Q(D \cup \Upsilon_I^{\mathrm{b}}, \tau_{out})$ *implies* $\mathbf{o} \in Q(D, \tau_{out})$ *for every tuple* $\mathbf{o}$ *over* $\Omega_I$*.*

Tractability of DTP then follows from Lemma 3 and the tractability of query evaluation for nonrecursive programs.

**Theorem 2.** DTP *is in* P *in data complexity if restricted to nonrecursive connected queries.*

## 4.2 Forgetting

We now move on to the forgetting problem as given in Definition 3. Unfortunately, in contrast to DTP, forgetting is undecidable. This follows by a reduction from containment of nontemporal Datalog queries—a well-known undecidable problem (Shmueli 1993).

**Theorem 3.** FORGET *is undecidable.*

In the remainder of this section we show that, by restricting ourselves to nonrecursive input queries, we can regain not only decidability of forgetting, but also tractability in data complexity. Let $I = \langle Q, D, \tau_{in}, \tau_{out}, \tau_{mem} \rangle$ be an arbitrary instance of FORGET where $Q$ is nonrecursive. We adopt the same technical assumptions as in Section 4.1 for DTP in the nonrecursive case. Additionally, we assume that $Q$ does not contain explicit time points in rules—note that the rules in our running example satisfy this restriction. We believe that dropping this assumption does not affect tractability, but we leave this question open for future work.

By Definition 3, to decide FORGET for $I$, we must check whether the answers $Q(D \cup U, \tau)$ are included in $Q(D[\tau_{mem}] \cup U, \tau)$ for every $\tau_{in}$-update $U$ and $\tau \geq \tau_{out}$.

Similarly to the case of DTP, we identify two time intervals of polynomial size in data, and show that it suffices to consider only updates $U$ over the first interval and only time points $\tau$ over the second interval. In contrast to DTP, however, we need to potentially consider all possible such updates and cannot restrict ourselves to a single critical one.

Note, however, that checking the aforementioned inclusion of query answers for all relevant updates and time points would lead to an exponential blowup in data complexity. To overcome this, we define instead nonrecursive queries $Q_1$ and $Q_2$ such that the desired condition holds if and only if $Q_1 \sqsubseteq Q_2$, where $Q_1$ and $Q_2$ contain a (fixed) rule set derived from $Q$ and a portion of the history $D$. Then, we show that checking such containment where only the data-dependent rules are considered part of the input is feasible in polynomial time.

We start by identifying the set of relevant time points for query answers and updates.

**Definition 10.** *A time point $\tau$ is* output-relevant *for $I$ if $\tau_{out} \leq \tau \leq \tau_{mem} + \mathrm{rad}(Q)$. In turn, it is* update-relevant *for $I$ if it satisfies $\tau_{in} < \tau \leq \tau_{mem} + 2 \cdot \mathrm{rad}(Q)$.*

Intuitively, answers at time points bigger than $\tau_{mem} + \mathrm{rad}(Q)$ cannot be affected by history facts that hold before $\tau_{mem}$; thus, we do not need to consider answers at time points that are not output-relevant. In turn, output-relevant time points cannot depend on facts in an update after $\tau_{mem} + 2 \cdot \mathrm{rad}(Q)$; thus, it suffices to consider updates containing only facts that hold at update-relevant time points. We next construct the aforementioned queries $Q_1$ and $Q_2$ using the identified update-relevant and output-relevant time points.

**Definition 11.** *Let $B$ be a fresh temporal IDB unary predicate and let $D_0$ consist of all facts $B(\tau)$ for each update-relevant time point $\tau$. For $\psi$ as in Definition 6, let $\Pi$ be the smallest program containing:* (i) *each rule in $\psi(\Pi_Q)$ having a predicate different from $P_Q$ in the head;* (ii) *each rule obtained by grounding the time argument to an output-relevant time point in a rule in $\psi(\Pi_Q)$ having $P_Q$ as head predicate; and* (iii) *the rule $P(\mathbf{x}, t) \wedge B(t) \rightarrow P'(\mathbf{x}, t)$ for each temporal EDB predicate $P$ occurring in $\Pi_Q$, where $P' = \psi(P)$.*

*We now let $Q_1 = \langle P_Q, \Pi_1 \rangle$ and $Q_2 = \langle P_Q, \Pi_2 \rangle$, where $\Pi_1 = \Pi \cup D_0 \cup \psi(D)$, and $\Pi_2 = \Pi \cup D_0 \cup \psi(D[\tau_{mem}])$.*

Intuitively, the facts about $B$ are used to 'tag' the update-relevant time points; rule $P(\mathbf{x}, t) \wedge B(t) \rightarrow P'(\mathbf{x}, t)$, when applied to the history $D$ and any update $U$, will 'project' $U$ to the relevant time points and filter out all facts in $D$. Finally, the rules in *(i)* and *(ii)* allow us to derive the same consequences (modulo predicate renaming) as $\Pi_Q$, but only over the output-relevant time points.

We can now establish correctness of our approach.

**Lemma 4.** FORGET *holds for $I$ iff $Q_1 \sqsubseteq Q_2$, where $Q_1$ and $Q_2$ are as given in Definition 11.*

**Theorem 4.** FORGET *is in* P *in data complexity if restricted to nonrecursive connected queries whose rules contain no time points.*

This concludes our discussion of the data-dependent problems motivated by our 'online' stream reasoning algorithm (recall Algorithm 1). In the following section, we turn our attention to the data-independent problems motivated by our 'offline' approach (Algorithm 2).

### 4.3 Data-Independent Problems

Query containment can be reduced to both DELAY and WINDOW using a variant of the reduction we used in Section 3 for the forgetting problem. As a result, we can show undecidability of both of our data-independent reasoning problems in the general case.

**Theorem 5.** DELAY *is undecidable.*

**Theorem 6.** WINDOW *is undecidable.*

Furthermore, our reductions from query containment preserve the shape of the queries and hence they also provide a co-NEXP lower bound for both problems in the nonrecursive case (Benedikt and Gottlob 2010). In the remainder of this section, we show that this bound is tight.

The co-NEXP upper bounds are obtained via reductions from DELAY and WINDOW into query containment for temporal Datalog, which we detail in the remainder of this section. Similarly to previous sections, we assume that queries are connected and also that they do not contain explicit time points or objects; the latter restriction is consistent with the 'purity' assumption in (Benedikt and Gottlob 2010).

Note, however, that the upper bound in (Benedikt and Gottlob 2010) for query containment only holds for standard nonrecursive Datalog; therefore, we first establish that this upper bound extends to the temporal case. Intuitively, temporal queries can be transformed into nontemporal ones by grounding them to a finite number of relevant time points based on their (finite) radius.

**Lemma 5.** *Query containment restricted to nonrecursive queries that are connected and constant-free is in* co-NEXP.

We now proceed to discussing our reductions from DELAY and WINDOW into temporal query containment.

Consider a fixed, but arbitrary, instance $I = \langle Q, d \rangle$ of DELAY. We construct queries $Q_1$ and $Q_2$ providing the basis for our reduction.

Let $A$ and $B$ be fresh fresh unary temporal predicates, where $A$ is EDB and $B$ is IDB. Furthermore, let $G$ be a fresh temporal IDB predicate of the same arity as $P_Q$. Let $\Pi_1$ extend $\Pi_Q$ with the following rule:

$$P_Q(\mathbf{x}, t) \wedge A(t) \rightarrow G(\mathbf{x}, t) \tag{7}$$

and let $Q_1 = \langle G, \Pi_1 \rangle$. Intuitively, $Q_1$ restricts the answers to $Q$ to time points where $A$ holds.

Let $Q_2 = \langle G, \Pi_2 \rangle$, where $\Pi_2$ is now the program obtained from $\psi(\Pi_Q)$ by adding the previous rule (7) and the following rules for each $k$ satisfying $-\mathrm{rad}(Q) \leq k \leq d$ and each temporal EDB predicate $P$ in $\Pi_Q$, where $P' = \psi(P)$:

$$A(t) \rightarrow B(t + k) \tag{8}$$

$$P(\mathbf{x}, t) \wedge B(t) \rightarrow P'(\mathbf{x}, t) \tag{9}$$

Intuitively, $Q_2$ further restricts the answers to $Q_1$ at any time $\tau_{out}$ to those that can be derived using facts in the interval $[\tau_{out} - \mathrm{rad}(Q), \tau_{out} + d]$.

It then follows that $Q_1 \sqsubseteq Q_2$ if and only if DELAY holds for $I$. Furthermore, the construction of $Q_1$ and $Q_2$ is feasible in LOGSPACE.

**Theorem 7.** DELAY *restricted to nonrecursive queries that are connected and constant-free is* co-NEXP-complete.

We conclude by providing the upper bound for WINDOW. For this, consider an arbitrary instance $I = \langle Q, d, s\rangle$. Similarly to the case of DELAY, we construct queries $Q_1$ and $Q_2$ for which containment holds iff WINDOW holds for $I$. Let $In$, $Out$ and $B$ be fresh unary temporal predicates, where $In$ is EDB and $Out$, $B$ are IDB. Furthermore, as before, let $G$ be a fresh temporal IDB predicate of the same arity as $P_Q$.

For $j$ a nonnegative integer, let $\Pi^j$ be the program extending $\psi(\Pi_Q)$ with the following rules for each $k$ satisfying $-d < k \leq -s + \mathrm{rad}(Q)$, each $\ell$ satisfying $-j < \ell \leq -s + 2 \cdot \mathrm{rad}(Q)$, and each temporal EDB predicate $P$ in $\Pi_Q$, where $P' = \psi(P)$:

$$In(t) \rightarrow Out(t + k) \tag{10}$$

$$P_Q(\mathbf{x}, t) \wedge Out(t) \rightarrow G(\mathbf{x}, t) \tag{11}$$

$$In(t) \rightarrow B(t + \ell) \tag{12}$$

$$P(\mathbf{x}, t) \wedge B(t) \rightarrow P'(\mathbf{x}, t) \tag{13}$$

We define $Q_1 = \langle G, \Pi^{d+\mathrm{rad}(Q)}\rangle$ and $Q_2 = \langle G, \Pi^s\rangle$. Intuitively, given a dataset for which $In$ holds for a set of time points $\mathbf{T}$, query $Q_1$ captures the answers to $Q$ at time points $\tau_{out}$ within the the interval $[\tau - d, \tau - s + \mathrm{rad}(Q)]$ for some $\tau \in \mathbf{T}$; in turn, for each such interval $[\tau - d, \tau - s + \mathrm{rad}(Q)]$, query $Q_2$ further restricts the answers to $Q_1$ to those that depend on input facts holding after $\tau - s$.

Since these queries can again be constructed in LOGSPACE, we obtain the desired upper bound.

**Theorem 8.** WINDOW *restricted to nonrecursive queries that are connected and constant-free is* co-NEXP-complete.

## 5 Related Work

The main challenges posed by stream processing and the basic architecture of a stream management system were first discussed in (Babu and Widom 2001; Babcock et al. 2002). Arasu, Babu, and Widom (2006) proposed the CQL query language, which extends SQL with a notion of *window*—a mechanism that allows one to reduce stream processing to traditional query evaluation. Since then, there have been numerous extensions and variants of CQL, which include a number of stream query languages for the Semantic Web (Barbieri et al. 2009; Le-Phuoc et al. 2011; 2013; Dell'Aglio et al. 2015).

In recent years, there have been several proposals for a general-purpose rule-based language in the context of stream reasoning. Streamlog (Zaniolo 2012) is a temporal Datalog language, which differs from the language considered in our paper in that it provides nonmonotonic negation and restricts the syntax so that only facts over time points explicitly present in the data can be derived. Furthermore, the focus in (Zaniolo 2012) is on dealing with so-called 'blocking queries', which are those whose answers may depend on input facts arbitrarily far in the future; for this, a syntactic fragment of the language is provided that precludes blocking queries. LARS is a temporal rule-based language featuring window constructs and negation interpreted according to the stable model semantics (Beck et al. 2015; Beck, Dao-Tran, and Eiter 2015; Beck, Dao-Tran, and Eiter 2016). The semantics of LARS is rather different from that of temporal Datalog; in particular, the number of time points in a model is considered as part of the input to query evaluation, and hence is restricted to be finite; furthermore, the notion of window is built-in in LARS.

Stream reasoning has been studied in the context of RDF-Schema (Barbieri et al. 2010), and ontology-based data access (Calbimonte, Corcho, and Gray 2010; Özçep, Möller, and Neuenstadt 2014). In these works, the input data is assumed to arrive as a stream, but the ontology language is assumed to be nontemporal. Stream reasoning has also been considered in the unrelated context of complex event processing (Anicic et al. 2011; Dao-Tran and Le-Phuoc 2015).

There have been a number of proposals for rule-based languages in the context of temporal reasoning; here, the focus is on query evaluation over static temporal data, rather than on reasoning problems that are specific to stream processing. Our temporal Datalog language is a notational variant of $\mathrm{Datalog}_{1\mathrm{S}}$—the core language for temporal deductive databases (Chomicki and Imieliński 1988; Chomicki and Imieliński 1989; Chomicki 1990). Templog is an extension of Datalog with modal temporal operators (Abadi and Manna 1989), which was shown to be captured by $\mathrm{Datalog}_{1\mathrm{S}}$ (Baudinet, Chomicki, and Wolper 1993). Datalog was extended with integer periodicity and gap-order constraints in (Toman and Chomicki 1998); such constraints allow for the representation of infinite periodic phenomena. Finally, DatalogMTL is a recent Datalog extension based on metric temporal logic (Brandt et al. 2017).

In the setting of database constraint checking, a problem related to our window problem was considered by Chomicki (1995), who obtained some positive results for queries formulated in temporal first-order logic.

## 6 Conclusion and Future Work

In this paper, we have proposed novel decision problems relevant to the design of stream reasoning algorithms, and have studied their computational properties for temporal Datalog. These problems capture the key challenges behind rule-based stream reasoning, where rules can propagate information both to past and future time points. Our results suggest that rule-based stream reasoning is feasible in practice for nonrecursive temporal Datalog queries. Our problems are, however, either intractable in data complexity or undecidable in the general case.

We have made several mild technical assumptions in our upper bounds for nonrecursive queries, which we plan to lift in future work. Furthermore, we have assumed throughout the paper that numbers in the input are encoded in unary; we are currently looking into the impact of binary encoding on the complexity of our problems. Finally, we are planning to study extensions of nonrecursive temporal Datalog for which decidability of all our problems can be ensured.

## Acknowledgments

This research was supported by the SIRIUS Centre for Scalable Data Access in the Oil and Gas Domain, the Royal Society, and the EPSRC projects DBOnto, MaSI$^3$, and ED$^3$.

## References

Abadi, M., and Manna, Z. 1989. Temporal logic programming. *J. Symb. Comput.* 8(3):277–295.

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

Arasu, A.; Babu, S.; and Widom, J. 2006. The CQL continuous query language: Semantic foundations and query execution. *VLDB J.* 15(2):121–142.

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In *PODS*, 1–16.

Babu, S., and Widom, J. 2001. Continuous queries over data streams. *SIGMOD Rec.* 30(3):109–120.

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

Barbieri, D. F.; Braga, D.; Ceri, S.; Valle, E. D.; and Grossniklaus, M. 2010. Incremental reasoning on streams and rich background knowledge. In *ESWC*, 1–15.

Baudinet, M.; Chomicki, J.; and Wolper, P. 1993. Temporal deductive databases. In Tansel, A. U.; Clifford, J.; Gadia, S.; Jajodia, S.; Segev, A.; and Snodgrass, R., eds., *Temporal Databases*. Benjamin Cummings. 294–320.

Beck, H.; Dao-Tran, M.; Eiter, T.; and Fink, M. 2015. LARS: A logic-based framework for analyzing reasoning over streams. In *AAAI*, 1431–1438.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2015. Answer update for rule-based stream reasoning. In *IJCAI*, 2741–2747.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2016. Equivalent stream reasoning programs. In *IJCAI*, 929–935.

Benedikt, M., and Gottlob, G. 2010. The impact of virtual views on containment. *PVLDB* 3(1-2):297–308.

Brandt, S.; Kalayci, E. G.; Kontchakov, R.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2017. Ontology-based data access with a horn fragment of metric temporal logic. In *AAAI*, 1070–1076.

Calbimonte, J.-P.; Corcho, O.; and Gray, A. J. 2010. Enabling ontology-based access to streaming data sources. In *ISWC*, 96–111.

Chomicki, J., and Imieliński, T. 1988. Temporal deductive databases and infinite objects. In *PODS*, 61–73.

Chomicki, J., and Imieliński, T. 1989. Relational specifications of infinite query answers. In *SIGMOD*, 174–183.

Chomicki, J. 1990. Polynomial time query processing in temporal deductive databases. In *PODS*, 379–391.

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

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

Dao-Tran, M., and Le-Phuoc, D. 2015. Towards enriching CQELS with complex event processing and path navigation. In *HiDeSt@KI*, 2–14.

Dao-Tran, M.; Beck, H.; and Eiter, T. 2015. Towards comparing RDF stream processing semantics. In *HiDeSt@KI*, 15–27.

Dell'Aglio, D.; Calbimonte, J.; Valle, E. D.; and Corcho, Ó. 2015. Towards a unified language for RDF stream query processing. In *ESWC (Satellite Events)*, 353–363.

Le-Phuoc, D.; Dao-Tran, M.; Parreira, J. X.; and Hauswirth, M. 2011. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC*, 370–388.

Le-Phuoc, D.; Quoc, H. N. M.; Le Van, C.; and Hauswirth, M. 2013. Elastic and scalable processing of linked stream data in the cloud. In *ISWC*, 280–297.

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

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

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

Ronca, A.; Kaminski, M.; Cuenca Grau, B.; Motik, B.; and Horrocks, I. 2017. Stream reasoning in temporal Datalog. *CoRR* abs/1711.04013.

Shmueli, O. 1993. Equivalence of datalog queries is undecidable. *J. Log. Program.* 15(3):231–241.

Toman, D., and Chomicki, J. 1998. Datalog with integer periodicity constraints. *J. Log. Program.* 35(3):263–290.

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