LARS: A Logic-based Framework for Analyzing Reasoning over Streams*

Harald Beck and **Minh Dao-Tran** and **Thomas Eiter** and **Michael Fink**
Institute of Information Systems, Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{beck,dao,eiter,fink}@kr.tuwien.ac.at

Abstract

The recent rise of smart applications has drawn interest to logical reasoning over data streams. Different query languages and stream processing/reasoning engines were proposed. However, due to a lack of theoretical foundations, the expressivity and semantics of these diverse approaches were only informally discussed. Towards clear specifications and means for analytic study, a formal framework is needed to characterize their semantics in precise terms. We present LARS, a **L**ogic-based framework for **A**nalyzing **R**easoning over **S**treams, i.e., a rule-based formalism with a novel window operator providing a flexible mechanism to represent views on streaming data. We establish complexity results for central reasoning tasks and show how the prominent Continuous Query Language (CQL) can be captured. Moreover, the relation between LARS and ETALIS, a system for complex event processing is discussed. We thus demonstrate the capability of LARS to serve as the desired formal foundation for expressing and analyzing different semantic approaches to stream processing/reasoning and engines.

Introduction

The emergence of sensors, networks, and mobile devices has generated a trend towards *pushing* rather than *pulling* of data in information processing. In *stream processing* (Babu and Widom 2001), studied by the database community, input tuples dynamically arrive at systems in form of possibly infinite streams. To deal with unboundedness of data, the systems typically apply *window operators* to obtain snapshots of recent data. The user runs *continuous queries* on the latter that are triggered either periodically or by events, e.g., by the arrival of new input. A prominent stream processing language is the Continuous Query Language (CQL) (Arasu, Babu, and Widom 2006), which has an SQL-like syntax and a clear operational semantics.

Recently, the rise of *smart applications* such as smart cities, smart home, smart grid, etc., has raised interest in the topic of *stream reasoning* (Della Valle et al. 2009), i.e., logical reasoning on streaming data. Consider the following example.

*This research has been supported by the Austrian Science Fund (FWF) projects P24090 and P26471.

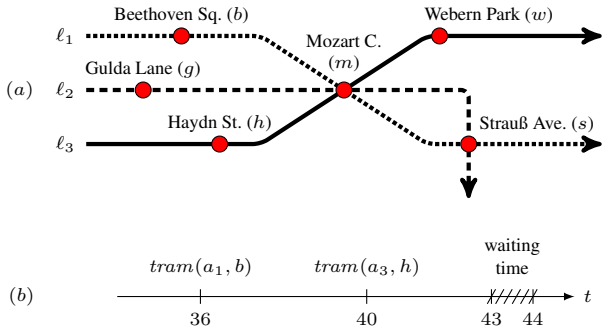

Figure 1: (a) Transportation map (b) Timeline (minutes)

Example 1 To monitor a city's public transportation, the city traffic center has a static background data set for the assignment of trams to lines of the form $line(Id, L)$, where Id is the tram and L the line identifier. The planned travelling time (duration Z) between stops X and Y with line L is stored by rows $plan(L, X, Y, Z)$. Facts of the form $old(Id)$ classify old trams which are inconvenient for travelling with baby strollers. Moreover, sensor data $tram(Id, X)$ and $jam(X)$ report the appearance of tram Id and traffic jams at stop X, respectively. Based on this, reports on the traffic status and suggested updates for travel routes shall be provided in real time.

Consider Bob travelling with his baby on line ℓ_3 (Fig. 1a). He is currently at Haydn Street (h) and wants to go to Strauß Avenue (s), so he has different options to change trams at Mozart Circus (m). Thus, he wants to know (i) the expected arrival time of the next tram that (ii) is convenient for the stroller. Fig. 1b depicts arrival times, e.g., $tram(a_1, b)$ at $t = 36$ represents that tram a_1 arrived at stop Beethoven Square at minute 36. Furthermore, consider the following background data tables, which specify the planned travel time between stops ($plan$), the association between lines and their trams ($line$) and which trams are old and thus not suitable for strollers (old).

$$plan = \{(\ell_1, b, m, 8), (\ell_2, g, m, 7), (\ell_3, h, m, 3), \ldots\}$$
$$line = \{(a_1, \ell_1), (a_2, \ell_2), (a_3, \ell_3), \ldots\} \quad old = \{(a_1), \ldots\}$$

Based on this input stream and the static background data, we expect the following reports (i) and (ii):

(i) Tram a_1 is expected to arrive at m at minute 44, and a_3 should arrive at m one minute earlier, i.e., at minute 43.

(ii) Switching from line ℓ_3 to ℓ_1 at m satisfies the short waiting time requirement. However, since tram a_1 is old, it is not a good connection with the stroller. ∎

Different research communities have contributed to various aspects of this topic, leaving several challenges to overcome. First, these predominantly practical approaches often define semantics only informally, which makes them hard to predict and hard to compare. Second, advanced reasoning features are missing, e.g., nonmonotonicity, nondeterminism or model generation. According techniques have been studied almost exclusively on static data.

Contributions. We present LARS, a **L**ogic-based framework for **A**nalyzing **R**easoning over **S**treams, providing (i) a rule-based formalism with (ii) different means to refer to or abstract from time, including (iii) a novel *window operator*, i.e., a flexible mechanism to change the view on streaming data. To date, no stream reasoning language with these features exists. Moreover, LARS features a model-based semantics, and it offers besides monotonic also nonmonotonic semantics that can be seen as an extension of Answer Set Programming (ASP) for stream reasoning.

We analyze the complexity of central reasoning tasks (model checking and satisfiability) in LARS, establishing that they do not get harder compared to ASP, provided that nesting of window operators is bounded (in particular, if no nesting occurs). Moreover, we demonstrate how the semantics of CQL can be expressed in LARS and study the relation of LARS and ETALIS (Anicic et al. 2010), a monotonic rule-based system for complex event processing.

The presented framework yields (a) a common ground to express various semantic concepts of different stream processing/reasoning formalisms and engines, which (b) can now be formally characterized in a common language, and thus (c) be compared analytically.

Streams

Streaming Data. We use mutually disjoint sets of *predicates* $\mathcal{P}$ and *constants* $\mathcal{C}$. The set $\mathcal{A}$ of *atoms* is defined as $\{p(c_1, \ldots, c_n) \mid p \in \mathcal{P}, c_1, \ldots, c_n \in \mathcal{C}\}$. If $i, j \in \mathbb{N}$, we call the set $[i, j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$ an *interval*. We divide $\mathcal{P}$ into two disjoint subsets, namely the *extensional predicates* $\mathcal{P}_\mathcal{E}$ and the *intensional predicates* $\mathcal{P}_\mathcal{I}$. The former is used for input streams and background data, while the latter serves for intermediate and output streams. Additionally, we assume basic arithmetic operations $(+, -, \times, \div)$ and comparisons $(=, \neq, <, >, \leq, \geq)$ are predefined by designated predicates $\mathcal{B} \subseteq \mathcal{P}_\mathcal{E}$, and used also in infix notation.

We now present the central notion of streams.

Definition 1 (Stream) Let T be an interval and $\upsilon \colon \mathbb{N} \to 2^\mathcal{A}$ an *evaluation function* such that $\upsilon(t) = \emptyset$ for all $t \in \mathbb{N} \setminus T$. Then, the pair $S = (T, \upsilon)$ is called a *stream*, T is the *timeline* of S, and the elements of T are *time points*.

Consider two streams $S = (T, \upsilon)$ and $S' = (T', \upsilon')$. We say S' is a *substream* or *window* of S, denoted $S' \subseteq S$, if $T' \subseteq T$ and $\upsilon'(t') \subseteq \upsilon(t')$ for all $t' \in T'$. We call S' a *proper substream* of S, denoted $S' \subset S$, if $S' \subseteq S$ and $S' \neq S$. Moreover, we define the *size* $\#S$ of S by $\Sigma_{t \in T} \max(|\upsilon(t)|, 1)$. The *restriction* $S|_{T'}$ *of* S *to* $T' \subseteq T$ is the stream $(T', \upsilon|_{T'})$, where $\upsilon|_{T'}$ restricts the domain of υ to T', i.e., $\upsilon|_{T'}(t) = \upsilon(t)$ for all $t \in T'$, else $\upsilon|_{T'}(t) = \emptyset$. A *data stream* contains only atoms with extensional predicates.

Example 2 (cont'd) Consider again the scenario of Example 1. We can model the input as the data stream $D = (T, \upsilon)$ with a timeline $T = [0, 50]$ and the evaluation $\upsilon(36) = \{tram(a_1, b)\}$, $\upsilon(40) = \{tram(a_3, h)\}$, and $\upsilon(t) = \emptyset$ for all $t \in T \setminus \{36, 40\}$. We will also represent the evaluation function υ by according mappings, i.e., by $\{36 \mapsto \{tram(a_1, b)\}, 40 \mapsto \{tram(a_3, h)\}\}$. ∎

Windows. An essential aspect of stream reasoning is to restrict data to so-called *windows*, i.e., recent substreams to limit the amount of data and forget outdated information.

Definition 2 (Window function) A *window function* w_ι of *type* ι takes as input a stream $S = (T, \upsilon)$, a time point $t \in T$, called the *reference time point*, and a vector of *window parameters* $\mathbf{x}$ for type ι and returns a substream S' of S.

The most common types of windows in practice are time-, tuple-, and partition-based windows. We associate them with three window functions w_τ, $w_\#$, and w_p, respectively. Traditionally (Arasu et al., 2006), these window functions take a fixed size ranging back in time from a reference time point t; we generalize this by allowing to look back and forth from t. Intuitively, these functions work as follows.

- *Time-based*: $\mathbf{x} = (\ell, u, d)$, where $\ell, u \in \mathbb{N} \cup \{\infty\}$ and $d \in \mathbb{N}$. The function $w_\tau(S, t, \mathbf{x})$ returns the substream of S that contains all tuples of the last ℓ time units and the next u time units relative to a *pivot* time point t' derived from t and the step size d (Fig. 2). We use $\ell = \infty$ (resp. $u = \infty$) to take all previous (resp. later) tuples.
- *Tuple-based*: $\mathbf{x} = (\ell, u)$, where $\ell, u \in \mathbb{N}$. The function $w_\#(S, t, \mathbf{x})$ selects a substream of S with the shortest interval $[t_\ell, t_u] \subseteq T$ as timeline, where $t_\ell \leq t \leq t_u$, such that ℓ tuples are in $[t_\ell, t]$ and u tuples are in $[t + 1, t_u]$. Exactly ℓ, resp. u tuples are returned. In case of multiple options due to multiple tuples at time points t_ℓ, resp. t_u, only tuples from there are removed at random.
- *Partition-based*: $\mathbf{x} = (\mathrm{idx}, n)$ where idx and n are two total functions $\mathrm{idx} \colon \mathcal{A} \to \emptyset \subset I \subset \mathbb{N}$ and $n \colon I \to \mathbb{N} \times \mathbb{N}$. Here, I is a finite index set. Applying $w_\mathrm{p}(S, t, \mathbf{x})$ first splits the input stream $S = (T, \upsilon)$ into $|I|$ substreams $S_i = (T, \upsilon_i)$ by taking $\upsilon_i(t) = \{a \in \upsilon(t) \mid \mathrm{idx}(a) = i\}$. Then, a tuple-based window $w_\#$ is applied on each S_i with parameters taken from $n(i) = (\ell_i, u_i)$. The output streams after $w_\#$ are then merged to the result window.

Here, we gave a slight generalization of window functions as presented (more formally) in (Beck et al. 2014), using the general parameter vector $\mathbf{x}$. This will be more useful when we discuss window applications with flexible sizes represented by variables.

Due to space reason, we present only the adaptation of time-based windows formally. The same idea can be applied to other types of windows straightforwardly.

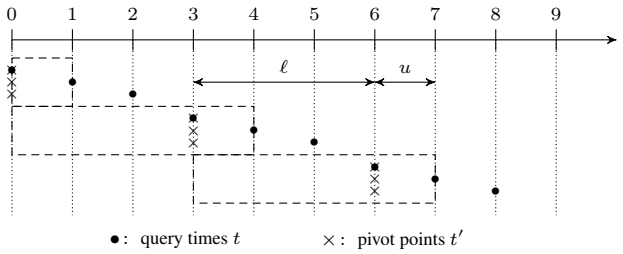

Figure 2: Time-based window w_τ with $\ell = 3, u = 1, d = 3$

Definition 3 (Time-based window func.) Let $S = (T, \upsilon)$ be a stream, $T = [t_{min}, t_{max}]$ and $t \in T$, $\mathbf{x} = (\ell, u, d)$ where $\ell, u \in \mathbb{N} \cup \{\infty\}$ and $d \in \mathbb{N}$ such that $d \leq \ell + u$. The *time-based window function with range* (ℓ, u) *and step size* d *of* S *at time* t is defined by $w_\tau(S, t, \mathbf{x}) = (T', \upsilon|_{T'})$, where $T'{=}[t_\ell, t_u]$, $t_\ell = \max\{t_{min}, t'{-}\ell\}$ with $t' = \lfloor \frac{t}{d} \rfloor \cdot d$, and $t_u = \min\{t'{+}u, t_{max}\}$.

The following example demonstrates the use of time-based windows.

Example 3 (cont'd) On the data stream D of Example 2, consider a monitoring use case where we want to know only the tram appearances reported within the last 4 minutes. To this end, we can use a time-based window function $w_\tau(D, t, (4, 0, 1))$ with a step size of 1. Applying it on D at $t = 42$ gives $w_\tau(D, 42, (4, 0, 1)) = ([38, 42], \upsilon')$, where $\upsilon' = \{40 \mapsto \{tram(a_3, h)\}\}$. ∎

The LARS Framework

We now define LARS, a logic-based framework for analyzing reasoning over streams. We present a logic with different means for time reference and time abstraction. On top of it, we will further introduce a rule language with a model-based, nonmonotonic semantics.

To account for the aspect of streaming, *window operators* of form $\boxplus^{\mathbf{x}}_{\iota,ch}$ are central to our approach. While a window *function* returns a stream S' based on an input stream S and a time point t, a window *operator* takes *two* streams as input. This allows us to distinguish between a fixed input stream $S^\star$ and the currently considered window S. Therefore, prior to the application of a window function, a *stream choice* ch selects a stream S' based on two streams. We use two straightforward stream choices $ch_1(S^\star, S){=}S^\star$ and $ch_2(S^\star, S){=}S$.

Definition 4 (Window operator) Let w_ι be a window function of type ι, ch a stream choice function, and $\mathbf{x}$ a vector of window parameters for type ι. Then, $\boxplus^{\mathbf{x}}_{\iota,ch}$ denotes a *window operator*.

We also omit ch_2 and simply write $\boxplus^{\mathbf{x}}_{\iota}$ for $\boxplus^{\mathbf{x}}_{\iota,ch_2}$. Furthermore, we use special syntax for typical parameters $\mathbf{x}$ for tuple- and time-based windows with step size $d = 1$ and write only ℓ for $\mathbf{x}$, if $u = 0$, and $+u$, if $\ell = 0$. For instance,

$$\boxplus^4_\tau = \boxplus^{4,0,1}_{\tau,ch_2} \qquad \boxplus^{+5}_\tau = \boxplus^{0,5,1}_{\tau,ch_2} \qquad \boxplus^1_\# = \boxplus^{1,0}_{\#,ch_2}$$

Here, all operators extract tuples from the second stream. The symbol $\boxplus^4_\tau$ abbreviates the time-based window operator that takes all tuples of the last 4 time points, while $\boxplus^{+5}_\tau$ takes all tuples of the next 5 time points. Moreover, $\boxplus^1_\#$ takes the latest tuple which arrived until the reference time point.

Syntax. In addition to window operators, we use further means to refer to or abstract from time. Similarly as in modal logic, we use operators $\Box$ and $\Diamond$ to represent that a tuple (atom) or formula holds at all times respectively some time in a window. Moreover, an *exact* operator @ is used to refer to specific time points.

Definition 5 (Formulas) Let $a \in \mathcal{A}$ be an atom and $t \in \mathbb{N}$. The set $\mathcal{F}$ of *formulas* is defined by the following grammar:

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \Diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus^{\mathbf{x}}_{\iota,ch}\alpha$$

Intuitively, given a stream $S^\star$ and a considered window S (which initially is $S^\star$), a formula α will be evaluated based on a reference time point t within S. An application of a window operator $\boxplus^{\mathbf{x}}_{\iota,ch}$ creates a new window S' that depends on $S^\star$ and S as specified by the stream choice. Within the current window S, $\Diamond\alpha$ (resp. $\Box\alpha$) holds, if α holds at some time point (resp. at all time points) in S. Relative to t, the formula α holds if α is true at t, and $@_{t'}\alpha$ holds if t' is in the timeline of S and α is true at t'. That is, the operator @ allows to 'jump' to a specific time point within the window.

Semantics. In addition to streams, we consider background knowledge in form of a static data set, i.e., a set $B \subseteq \mathcal{A}$ of atoms which does not change over time. From a semantic perspective, the difference to streams is that static data is always available, regardless of window applications.

Definition 6 (Structure) Let $S = (T, \upsilon)$ be a stream, W be a set of window functions and $B \subseteq \mathcal{A}$ a set of facts. Then, we call $M = \langle T, \upsilon, W, B\rangle$ a *structure*, S the *interpretation stream* and B the *data set* or *background data* of M.

We now define when a formula holds in a structure.

Definition 7 (Entailment) Let $M = \langle T^\star, \upsilon^\star, W, B\rangle$ be a structure, $S^\star = (T^\star, \upsilon^\star)$ and let $S = (T, \upsilon)$ be a substream of $S^\star$. Moreover, let $t \in T$. The *entailment* relation $\Vdash$ between (M, S, t) and formulas is defined as follows. Let $a \in \mathcal{A}$ be an atom, and let $\alpha, \beta \in \mathcal{F}$ be formulas. Then,

$$\begin{array}{lll}
M,S,t \Vdash a & \text{iff} & a \in \upsilon(t) \text{ or } a \in B,\\
M,S,t \Vdash \neg\alpha & \text{iff} & M,S,t \nVdash \alpha,\\
M,S,t \Vdash \alpha \wedge \beta & \text{iff} & M,S,t \Vdash \alpha \text{ and } M,S,t \Vdash \beta,\\
M,S,t \Vdash \alpha \vee \beta & \text{iff} & M,S,t \Vdash \alpha \text{ or } M,S,t \Vdash \beta,\\
M,S,t \Vdash \alpha \rightarrow \beta & \text{iff} & M,S,t \nVdash \alpha \text{ or } M,S,t \Vdash \beta,\\
M,S,t \Vdash \Diamond\alpha & \text{iff} & M,S,t' \Vdash \alpha \text{ for some } t' \in T,\\
M,S,t \Vdash \Box\alpha & \text{iff} & M,S,t' \Vdash \alpha \text{ for all } t' \in T,\\
M,S,t \Vdash @_{t'}\alpha & \text{iff} & M,S,t' \Vdash \alpha \text{ and } t' \in T,\\
M,S,t \Vdash \boxplus^{\mathbf{x}}_{\iota,ch}\alpha & \text{iff} & M,S',t \Vdash \alpha,\\
 & & \text{where } S' = w_\iota(ch(S^\star, S), t, \mathbf{x}).
\end{array}$$

If $M, S, t \Vdash \alpha$ holds, we say that (M, S, t) *entails* α. Moreover, M *satisfies* α *at time* t, if $(M, S^\star, t)$ entails α. In this case we write $M, t \models \alpha$ and call M a *model* of α *at time* t. Satisfaction and the notion of a model are extended to sets of formulas as usual.

Example 4 (cont'd) Let $D = (T, \upsilon)$ be the data stream of Ex. 3 and $S^\star{=}(T^\star, \upsilon^\star){\supseteq}D$ be a stream such that $T^\star{=}T$ and

$$\upsilon^\star{=}\left\{ \begin{array}{l} 36 \mapsto \{tram(a_1, b)\},\ 40 \mapsto \{tram(a_3, h),\\ 43 \mapsto \{exp(a_3, m)\},\ 44 \mapsto \{exp(a_1, m)\} \end{array} \right\}.$$

Let $M = \langle T^\star, \upsilon^\star, W, B\rangle$, where $W = \{w_\tau\}$, and B is the set of facts from the data tables in Example 1. Then it holds that $M, S^\star, 42 \Vdash \boxplus^{+5}_\tau \Diamond exp(a_3, m)$: The window operator $\boxplus^{+5}_\tau$ selects $S' = (T', \upsilon')$, with timeline $T' = [42, 47]$ and $\upsilon' = \{43 \mapsto \{exp(a_3, m)\}, 44 \mapsto \{exp(a_1, m)\}\}$, i.e., there is some $t' \in T'$ ($t' = 43$) s.t. $M, S', t' \Vdash exp(a_3, m)$. ∎

Programs. Now we define a rule language for stream reasoning with semantics similar to Answer Set Programming.

Definition 8 (Rule, Program) A *program* P is a set of *rules*, i.e., expressions of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_j, \text{not } \beta_{j+1}, \ldots, \text{not } \beta_n\ , \qquad (1)$$

where $\alpha, \beta_1, \ldots, \beta_n \in \mathcal{F}$ are formulas and α contains only intensional predicates.

Suppose we want to evaluate a program P on a data stream D. Let $I = (T, \upsilon)$ be a stream such that $D \subseteq I$. If at every time point in T, all atoms that occur in I but not in D have intensional predicates, then we call I an *interpretation stream for* D and a structure $M = \langle T, \upsilon, W, B\rangle$ an *interpretation (for D)*. Let for any rule r of form (1), be $\beta(r) = \beta_1 \wedge \ldots \wedge \beta_j \wedge \neg\beta_{j+1} \wedge \ldots \wedge \neg\beta_n$. We then say that M is a *model of P (for D at time t)*, denoted $M, t \models P$, if $M, t \models \beta(r) \rightarrow \alpha$ for all rules $r \in P$.[1] We call M a *minimal model*, if no model $M' = \langle T', \upsilon', W, B\rangle$ of P (for D at time t) exists such that $(T', \upsilon') \subset (T, \upsilon)$. The *reduct* of a program P w.r.t. M at time t is defined by $P^{M,t} = \{r \in P \mid M, t \models \beta(r)\}$, i.e., the subset of rules whose bodies are satisfied.

Definition 9 (Answer Stream) Let $M = \langle T, \upsilon, W, B\rangle$ be a structure, where $I = (T, \upsilon)$ is an interpretation stream for a data stream D, let P be a program and $t \in T$. Then, I is called an *answer stream* of P for D at time t (relative to W and B) iff M is a minimal model of the reduct $P^{M,t}$.

For ASP fragments of LARS, answer streams correspond to answer sets as defined by the FLP-reduct (Faber et al., 2004), which we formulated for LARS programs above. More precisely, consider an interpretation stream $I = (\{t\}, \upsilon')$ for a data stream $D = (\{t\}, \upsilon)$ and let P be a program where in each rule of form (1) all body formulas β_i are atoms and the head α is a disjunction of atoms with intensional predicates. Then, I is an answer stream of P at t relative to some W and B iff $\upsilon'(t)$ is an answer set of $P \cup \upsilon(t) \cup B$.

Towards more conciseness, we consider schematic programs with variables of two sorts, namely constant variables and time variables. The semantics of these *nonground programs* is given by the answer streams of according groundings, obtained by replacing variables with constants from $\mathcal{C}$, respectively time points from T, in all possible ways.

Example 5 (cont'd) The requests (i) and (ii) from Example 1 can be formulated by rules (2) and (3), respectively.

$$@_T\, exp(Id, Y) \leftarrow \boxplus^{\text{idx},n}_{\text{p}}\, @_{T_1}\, tram(Id, X), line(Id, L), \text{not } \boxplus^{20}_\tau \Diamond jam(X), plan(L, X, Y, Z), T = T_1 + Z. \qquad (2)$$

$$gc(Id_1, Id_2, X) \leftarrow @_T\, exp(Id_1, X), @_T\, \boxplus^{+5}_\tau \Diamond exp(Id_2, X), Id_1 \neq Id_2, \text{not } old(Id_2). \qquad (3)$$

[1] Thus, "not" and "¬" coincide, as well as "∧" and ",".

Rule (2) encodes when a tram is expected at later stops. For the partition-based window operator $\boxplus^{\text{idx},n}_{\text{p}}$, we use $\text{idx}(at) = i$ for an atom at of form $tram(a_i, X)$ and $\text{idx}(at) = 0$ else. By the tuple-based windows of sizes $n(i) = (1, 0)$ for $i > 0$ and $n(0) = (0, 0)$ applied on the $i + 1$ obtained substreams, we thus get for each tram a_i only its most recent appearance at some stop X. Usually, the expected arrival time on the next stop can be computed by the travelling duration according to the table $plan$. For the case of traffic jams within the last 20 minutes, we block such conclusions by means of default negation.

Next, rule (3) builds on the expected arrival times of rule (2) to identify good connections where the targeted tram is not old and the expected waiting time is at most 5 minutes. It uses a time-based window that looks 5 minutes ahead from the time when $exp(Id_1, X)$ is concluded and checks the existence (operator $\Diamond$) of an expected (different) tram Id_2.

We observe that the interpretation stream of the structure M of Example 4 is an answer stream of P for D at time t. Note that $gc(a_3, a_1, m)$ is not derived. Tram a_1 appears one minute after a_3 at Mozart Circus, but it is old. ∎

The next example demonstrates another advantage of our rule-based approach, namely the possibility to obtain different models for nondeterministic choices.

Example 6 (cont'd) Consider an extended scenario where a tram with identifier a_2 of line ℓ_2 is reported at Gulda Lane (g) at time point 38. This updates the data stream $D = (T, \upsilon)$ in Example 2 to $D' = (T, \upsilon')$, where $\upsilon' = \upsilon \cup \{38 \mapsto \{tram(a_2, g)\}\}$. By the entries $line(a_2, \ell_2)$ and $plan(\ell_2, g, m, 7)$ in B, rule (2) derives that tram a_2 is expected to arrive at Mozart Circus at $t = 45$. Furthermore, we now assume that tram a_1 is not old, i.e., $old(a_1) \notin B$. This gives Bob three good connections at stop m, when leaving tram a_3 at minute 43:

$$G = \{gc(a_3, a_1, m), gc(a_1, a_2, m), gc(a_3, a_2, m)\}$$

Bob is not interested in the connection from a_1 to a_2, since he is currently travelling with a_3. His smart phone streams an according tuple $on(a_3)$ at query time. This leaves him two options: He can either change to line ℓ_1 (and take tram a_1 after 1 minute at time point 44), or to line ℓ_2 (and take tram a_2 after 2 minutes at 45). The following two rules formalize the possibility to either change trams or skip a good connection:

$$change(Id_1, Id_2, X) \leftarrow on(Id_1), gc(Id_1, Id_2, X), \text{not } skip(Id_1, Id_2, X). \qquad (4)$$

$$skip(Id_1, Id_2, X) \leftarrow gc(Id_1, Id_2, X), change(Id_1, Id_3, X), Id_2 \neq Id_3. \qquad (5)$$

Consider the program P consisting of the rules (2)-(5). Moreover, let $D'' = (T, \upsilon'')$ be the data stream obtained from D' by adding $\{42 \mapsto \{on(a_3)\}\}$ to the evaluation and let $I_0 = (T, \upsilon_0)$, $I_1 = (T, \upsilon_1)$ and $I_2 = (T, \upsilon_2)$ be the following interpretation streams for D'': We take

$$\upsilon_0 = \upsilon \cup \left\{ \begin{array}{ll} 42 \mapsto G, & 43 \mapsto \{exp(a_3, m)\} \\ 44 \mapsto \{exp(a_1, m)\}, & 45 \mapsto \{exp(a_2, m)\} \end{array} \right\},$$

and for $i \in \{1, 2\}$, let $\upsilon_i = \upsilon_0 \cup \{42 \mapsto choice_i\}$, where

$$choice_1 = \{change(a_3, a_1, m), skip(a_3, a_2, m)\}, \text{ and}$$
$$choice_2 = \{change(a_3, a_2, m), skip(a_3, a_1, m)\}.$$

	α/α^-	P/P^-
MC	PSPACE/P	PSPACE/co-NP
SAT	PSPACE/NP	PSPACE/Σ_2^P

Table 1: Reasoning in ground LARS (completeness results)

Then, I_1 and I_2 are (the only) two answer streams for P at time 42 relative to $W = \{w_\tau, w_\mathrm{p}\}$ and B, i.e., we get the user choices as separate models. ∎

Note that in this example we did not constrain good connections by the actual destination Bob wants to reach. By means of the presented formalism, such reachability relations can be expressed elegantly through recursion as in Datalog.

Another benefit of our approach for advanced stream reasoning is the possibility to *retract* previous conclusions due to new input data. Combined with (minimal) model generation, i.e., alternatives that may be enumerated, compared under preference etc., such nonmonotonic reasoning allows for sophisticated AI applications in data stream settings.

Example 7 (cont'd) If the lines ℓ_1 and ℓ_2 have the same travelling time from Mozart Circus to Strauß Avenue, Bob will pick $choice_1$ (answer stream I_1), since at $t = 42$ tram a_1 is expected to arrive one minute earlier than tram a_2.

Suppose a few seconds later (still at $t = 42$) a traffic jam is reported for Beethoven Square. Thus, we now consider the data stream $D_j = (T, \upsilon_j)$, where $\upsilon_j = \upsilon \cup \{42 \mapsto \{on(a_3), jam(b)\}\}$. Thus, we have no expectation anymore when tram a_1 will arrive at Mozart Circus. Now $exp(a_1, m)$ cannot be concluded for $t = 44$, and as a consequence, $gc(a_3, a_1, m)$ will not hold anymore. Thus, the previous two answer streams are discarded and only $change(a_3, a_2, m)$ remains recommended in the resulting unique answer stream. ∎

Complexity of Reasoning in LARS

Let α be a formula, P a program, W a set of window functions evaluable in polynomial time, and let $B \subseteq \mathcal{A}$ be a set of atoms. We say that a stream $S = (T, \upsilon)$ *is over* $\mathcal{A}' \subseteq \mathcal{A}$, if $\upsilon(t) \setminus \mathcal{A}' = \emptyset$ for all $t \in T$.

We study the complexity of the following reasoning tasks:

(1) *Model checking* (MC). Given $M = \langle T, \upsilon, W, B\rangle$ and a time point t, check whether
- for a stream $S \subseteq (T, \upsilon)$ and formula α it holds that $M, S, t \Vdash \alpha$; resp.
- $I = (T, \upsilon)$ is an answer stream of a program P for $D \subseteq I$ at t.

(2) *Satisfiability* (SAT). For decidability, we assume that relevant atoms are confined to (polynomial) $\mathcal{A}' \subseteq \mathcal{A}$. The reasoning tasks are:
- Given W, B, a timeline T and a time point t, is there an evaluation function υ on T such that $M, S, t \Vdash \alpha$, where $M = \langle T, \upsilon, W, B\rangle$ and $S = (T, \upsilon)$ is over $\mathcal{A}'$?
- Given W, B and a data stream D, does there exist an answer stream of P for D over $\mathcal{A}'$ (relative to W and B) at t?

Table 1 shows the complexity of reasoning in ground LARS, where α^-, P^- are formulas resp. programs with nesting of window operators bounded by a constant. Note that the problems refer to the more general notion of entailment but (hardness) results carry over to satisfaction. The complexity of the general case is based on the following theorem.

Theorem 1 *Given a structure $M = \langle T, \upsilon, W, B\rangle$, a stream S, a time point t, and an arbitrary ground formula α, deciding $M, S, t \Vdash \alpha$ is* PSPACE*-complete, and* PSPACE*-hardness holds for $S = (T, \upsilon)$.*

Intuitively, PSPACE-membership can be shown by a depth-first-search evaluation of a formula along its tree representation. At each node of the tree, we need to store the content according to the window operators applied as in the path from the root, which requires only polynomial space.

The PSPACE-hardness can be shown by a reduction from evaluating QBFs $\exists x_1 \forall x_2 \ldots Q_n x_n \phi(\mathbf{x})$ to LARS MC. A LARS formula $\alpha = \boxplus^1 \Diamond \boxplus^2 \Box \ldots \phi(\mathbf{x})$ on the time line $T = [0, 1]$ is constructed where $\boxplus^i$ effects the possible truth assignments to x_i at the time points 0 or 1, and $\Diamond$, $\Box$ naturally encode the quantifiers $\exists$ and $\forall$.

The next result addresses the complexity of MC for ground LARS programs.

Theorem 2 *MC for LARS programs, i.e., given a structure $M = \langle T, \upsilon, W, B\rangle$, a data stream D, a program P, and a time point t, decide whether $I = (T, \upsilon)$ is an answer stream of P for D at time t, is* PSPACE*-complete.*

Proof. To decide the problem, we can (a) check that I is an interpretation stream for D, (b) compute $P^{M,t}$, and (c) check that M is a minimal model of $P^{M,t}$, i.e., that (c.1) $M, t \models P^{M,t}$ and (c.2) no $M' = \langle T', \upsilon', W, B\rangle$, with $(T', \upsilon') \subset (T, \upsilon)$ exists s.t. $M', t \Vdash P^{M,t}$. Now,

1. step (a) is trivially polynomial;
2. steps (b) and (c.1) are feasible in polynomial time using a PSPACE oracle; and
3. step (c.2) is feasible in nondeterministic polynomial time using a PSPACE oracle (guess (T', υ') and check $M', t \Vdash P^{M,t}$).

Overall, the computation is feasible in NPSPACE, thus in PSPACE (as NPSPACE = PSPACE).

PSPACE-hardness of the problem is immediate from Theorem 1: let $P = \{\alpha \leftarrow \top\}$, where $\top$ is an arbitrary tautology and exploit $S = (T, \upsilon)$. □

Under restrictions, however, MC may be tractable. This holds e.g. for the important case where only time-based windows are allowed. In case of α^-, the evaluation tree for MC has only polynomially many window contents to process, and we can use a standard labeling technique to evaluate formulas bottom up (from subformulas) in polynomial time.

SAT for α (resp. α^-) is in PSPACE (resp. NP) as guess and check establishes membership, and hardness is inherited from MC (resp. from propositional SAT). For monotone (e.g., time-based) window functions, the results apply setting $\mathcal{A}'$ to the atoms in α; also for tuple-based and partition-based windows reasonable assumptions (e.g., $\ell, u \ll \#S$ and idx monotone) yield only polynomially larger $\mathcal{A}'$.

For LARS programs, building the reduct $P^{M,t}$ is for P (resp. P^-) feasible in polynomial space (resp. time); the minimality check is feasible in polynomial space (resp. requires a polynomial guess to refute minimality). A more detailed complexity analysis including schematic programs (with $\mathcal{A}$ possibly infinite) is subject to ongoing work.

Capturing CQL

The Continuous Query Language (CQL) (Arasu, Babu, and Widom 2006) is an SQL based language for maintaining continuous queries over streaming data. It extends SQL with different operators. Two important ones are:

– *Stream-to-relation* (S2R) operators apply window functions to the input stream to create a mapping from execution times to bags of valid tuples (w.r.t. the window) without timestamps. This mapping is called a *relation*.

– *Relation-to-relation* (R2R) operators can manipulate relations similarly as in relational algebra, respectively SQL.

Example 8 The request (i) from Example 1 can be represented by the following CQL query.

```
SELECT ID, PLAN.Y, T2
FROM  TRAM[PARTITION BY ID ROWS 1],LINE,PLAN
WHERE TRAM.ID=LINE.ID AND LINE.L=PLAN.L AND
      TRAM.ST=PLAN.X AND T2=TRAM.T+PLAN.Z
      AND NOT EXISTS
              (SELECT * FROM JAM[RANGE 20]
               WHERE JAM.ST=TRAM.ST)
```

Note that these streams have designated timestamp fields. ∎

To capture CQL queries by LARS programs, we exploit two well-known translations: from SQL to relational algebra (Dadashzadeh and Stemple 1990) and from relational algebra to Datalog (Garcia-Molina, Ullman, and Widom 2009). Let us call the former Δ_ρ and the latter Δ_δ.

The idea is to have a 3-step process, given a CQL query q:

(1) apply a translation Δ_{SRC} (Table 2) to the input streams (with windows) and tables in the FROM and WHERE clauses of q. Let $\Delta_{\mathsf{SRC}}(q)$ denote the result of applying Δ_{SRC} on the input streams of q. Considering the formulas of $\Delta_{\mathsf{SRC}}(s)$ as table names, we get an SQL query;

(2) apply Δ_ρ on this query to get a relational algebra expression; and

(3) apply Δ_δ on the expression to get a program.

Considering the translated table names as LARS formulas, we get a LARS program. Formally speaking, the translation of a CQL query q is $\Delta(q) = \Delta_\delta(\Delta_\rho(\Delta_{\mathsf{SRC}}(q)))$, and that of a set Q of CQL queries is given by $\Delta(Q) = \bigcup_{q\in Q} \Delta(q)$.

Example 9 The translation of the CQL query in Ex. 8 is:

$$q_1(\mathbf{P}_1) \leftarrow \boxplus_{\mathrm{p}}^{\mathrm{idx},n} \Diamond tram(Id, ST, T_1), line(Id, L), \\ plan(L, X, Y, Z), ST = X, T_2 = T_1 + Z.$$

$$q_2(\mathbf{P}_2) \leftarrow \boxplus_{\tau}^{20} \Diamond jam(ST, T_3), \boxplus_{\mathrm{p}}^{\mathrm{idx},n} \Diamond tram(Id, ST, T_1).$$

$$q_{12}(\mathbf{P}_{12}) \leftarrow q_1(\mathbf{P}_1), q_2(\mathbf{P}_2).$$

$$q(\mathbf{P}) \leftarrow q_1(\mathbf{P}_1), \text{not } q_{12}(\mathbf{P}_{12}),$$

where $\mathbf{P}_1{=}Id, ST, X, Y, Z, T_1, T_2$; $\mathbf{P}_2{=}Id, ST, T_1, T_3$; $\mathbf{P}_{12}{=}\mathbf{P}_1, T_3$; $\mathbf{P}{=}Id, Y, T_2$; and idx, n are from Ex. 5. ∎

Input source s	$\Delta_{\mathsf{SRC}}(s)$
S	S
S[RANGE L]	$\boxplus_{\tau}^{L} \Diamond S$
S[RANGE L SLIDE D]	$\boxplus_{\tau}^{L,0,D} \Diamond S$
S[RANGE UNBOUNDED]	$\boxplus_{\tau}^{\infty} \Diamond S$
S[NOW]	S
S[ROWS N]	$\boxplus_{\#}^{N} \Diamond S$
S[PARTITION BY X1,...,Xk ROWS N]	$\boxplus_{\mathrm{p}}^{\mathrm{idx},n} \Diamond S$

Table 2: Translation function Δ_{SRC}

To establish the correspondence between the result of a set Q of CQL queries and its LARS translation $\Delta(Q)$, we first formally build a conversion of CQL streams to a LARS input stream. W.l.o.g., assume that Q is evaluated on a background data table B and input streams $\mathtt{S}_1, \ldots, \mathtt{S}_n$, and that any stream is only used in one place in the FROM clause in a single query (we can always duplicate streams and rename them). These input streams can be represented as the set $\mathtt{S} = \{\langle \mathtt{S}_i(\mathbf{a}_{ij}), t_{ij}\rangle \mid 1 \le i \le n, 1 \le j \le m_i\}$.

The corresponding representation of the input stream in LARS is defined by $\nabla(\mathtt{S}){=}(T_\mathtt{S}, \upsilon_\mathtt{S})$, where for $1 \le i \le n$, and $1 \le j \le m_i$:

$$T_\mathtt{S} = [t_{min}, t_{max}] \qquad t_{min} = \min\{t_{ij}\}$$
$$\upsilon_\mathtt{S}(t') = \{S_i(\mathbf{a}_{ij}) \mid t_{ij} = t'\}, t' \in T \qquad t_{max} = \max\{t_{ij}\}.$$

Let $res(q, t)$ denote the set of all answers to q and let $res(Q, t) = \bigcup_{q\in Q} res(q, t)$. The following theorem shows that the translation Δ faithfully captures CQL.

Theorem 3 *Let Q be a set of CQL queries to be evaluated on input streams $\mathtt{S} = \mathtt{S}_1, \ldots, \mathtt{S}_n$ and a background data table $\mathtt{B}$, $P = \Delta(Q)$, and t a time point. Then,*

(a) There exists an answer stream $I = (T, \upsilon)$ of P for $\nabla(\mathtt{S})$ at t, such that $\upsilon(t)|_Q = res(Q, t)$.

(b) If $I = (T, \upsilon)$ is an answer stream of P for $\nabla(\mathtt{S})$ at t, then $res(Q, t) = \upsilon(t)|_Q$.

Intuitively, (b) establishes the soundness and (a) the completeness of the translation Δ.

Proof (Sketch). Given a set of CQL queries Q and its translated LARS program $\Delta(Q)$, we establish the correspondence between their answers by a translation from Q to a Datalog program $\Delta_\mathcal{D}(Q)$. Briefly, $\Delta_\mathcal{D}(Q)$ is constructed in a similar way as $\Delta(Q)$, except that the first step translates the input streams (with windows) to plain table names instead of LARS formulas. Formally speaking, this is done by a renaming function ren instead of Δ_{SRC}. Then, we apply Δ_ρ and Δ_δ to get $\Delta_\mathcal{D}(Q) = \bigcup_{q\in Q} \Delta_\delta(\Delta_\rho(ren(q)))$. Note that both $\Delta(Q)$ and $\Delta_\mathcal{D}(Q)$ are acyclic programs, thus each of them has a unique minimal model.

By the correctness of Δ_ρ and Δ_δ, the unique answer set of $\Delta_\mathcal{D}(Q)$ and the result set of Q correspond. Moreover, one can guarantee that $\Delta(Q)$ and $\Delta_\mathcal{D}(Q)$ are evaluated essentially on the same input (despite slightly different representation) when computing answers for the same reference time

point. As moreover the programs are structurally the same, they correspond on their unique answer set/answer stream.

The two above observations yield the desired correspondence result between the results of Q and the answer stream of the respective LARS program $\Delta(Q)$. □

Relation to ETALIS

Related to stream processing is *complex event processing* (CEP), which is concerned with describing and detecting complex events (high-level information) based on atomic events (low-level information) of a stream. Complex events are typically expressed over time intervals. By briefly studying the well-known CEP language ETALIS (Anicic et al. 2010), we will draw a line between stream reasoning and complex event processing by means of our formalization.

In ETALIS, an *event stream* ϵ maps *atomic events* (ground atoms) to time points. Instead of non-negative rational numbers, we use natural numbers, which suffice for practical purposes. *Complex events* can be constructed by rules on *event patterns*, which are similar to interval relations by Allen (1983). An interpretation $\mathcal{I}$ maps atoms to sets of pairs $\langle t_1, t_2\rangle \in \mathbb{N} \times \mathbb{N}$, which represent intervals $[t_1, t_2]$. Intuitively, $\mathcal{I}$ satisfies a rule $a \leftarrow pt$, if the atom a holds at least in the set of intervals where the event pattern pt holds. For an event stream ϵ and a rule base $\mathcal{R}$, Anicic et al. define ETALIS semantics in terms of minimal models that (i) map each atomic event a to the interval $\langle t, t\rangle$ if a occurs in ϵ at time point t, and (ii) satisfy each rule $r \in \mathcal{R}$.

Intervals in LARS. Although LARS is based on time points, we can express ETALIS patterns that are based on intervals. Consider a window function w_{int} that selects the substream of (the greatest timeline within) a given interval $[\ell, u]$, and a window operator $\boxplus^{[\ell,u]}$ that employs w_{int} on the input stream. Furthermore, let $[\![\ell, u]\!]$ stand for $\boxplus^{[0,\infty]}@_u \boxplus^{[\ell,u]}\Box$, i.e., first create a view on the entire input stream, jump to reference time u, then select the substream of the timeline $[\ell, u]$ and apply $\Box$. Then, the formula $[\![\ell, u]\!]a$ holds iff a holds at every time point in the interval $[\ell, u]$, regardless of the query time. Similarly, we can define $[\![\ell, u]\!]^\star$ such that $[\![\ell, u]\!]^\star a$ holds iff $[\ell, u]$ is a *maximal* interval in which a always holds.

Example 10 Consider the events x and y which hold in the intervals given by the pairs $\langle t_1, t_2\rangle$ and $\langle t_3, t_4\rangle$, respectively, where $t_2 < t_3$. Then, the ETALIS rule $z \leftarrow x$ SEQ y assigns the pair $\langle t_1, t_4\rangle$ to z. It may be modelled in LARS by the rule $[\![t_1, t_4]\!]z \leftarrow [\![t_1, t_2]\!]^\star x, [\![t_3, t_4]\!]^\star y, t_2 < t_3$, i.e., if x holds in the entire interval $[t_1, t_2]$ (but not in any larger interval) and y holds throughout the later interval $[t_3, t_4]$ (also maximally), than z must hold throughout $[t_1, t_4]$. ■

However, we cannot fully express the ETALIS semantics in LARS by this straightforward encoding, since ETALIS allows atoms to be assigned to multiple intervals that need not be disjoint. In LARS, we assign atoms to a single timeline by the evaluation $\upsilon : T \to 2^{\mathcal{A}}$. Unless we explicitly use time points in atoms, we can encode intervals only by assigning atoms to consecutive time points. Overlapping or adjacent intervals for the same atom are indistinguishable from a merged view of them.

We call $\mathcal{I}$ *separable*, if such overlaps do not occur. If the minimal model of an event stream ϵ and a rule base $\mathcal{R}$ is separable, we also call the pair $\epsilon, \mathcal{R}$ separable. In this case, the approach of Ex. 10 allows us to capture ETALIS. In our subsequent results we confine to positive rule bases, i.e., without the NOT pattern. Our notion of minimality is based on set inclusion, whereas ETALIS defines minimality in terms of a different preference relation that ensures the minimal length and the supportedness of inferred intervals. By this, the minimal model is always unique, while a natural translation of negation in LARS would give multiple models in general. Capturing ETALIS' minimal model semantics of NOT patterns in LARS would require a more involved and less direct translation (which is beyond the scope of this work).

Theorem 4 *Let ϵ be an event stream, let $\mathcal{R}$ be a positive rule base (i.e. without negation) such that $\epsilon, \mathcal{R}$ is separable, and let $\mathcal{I}$ be an interpretation for $\epsilon, \mathcal{R}$. Then one can construct a LARS input stream Δ^ϵ, a program $\Delta^\mathcal{R}$, and an interpretation stream $\Delta^\mathcal{I} = (T, \upsilon)$, such that for each $t \in T$, $\mathcal{I}$ is the minimal model for $\epsilon, \mathcal{R}$ iff $\Delta^\mathcal{I}$ is the answer stream of $\Delta^\mathcal{R}$ for Δ^ϵ at time t relative to $W = \{w_{int}\}$ and $B = \emptyset$.*

Taking LARS and ETALIS as reference languages, separability can thus be regarded as the dividing line between stream reasoning and complex event processing. Under a stream reasoning view on ETALIS, focusing on truth at single time points, we get correspondence.

Corollary 1 *Let ϵ be an event stream, $\mathcal{R}$ be a positive rule base such that the minimal model $\mathcal{I}$ of $\epsilon, \mathcal{R}$ is separable and let $\Delta^\mathcal{I} = (T, \upsilon)$, i.e., the answer stream of $\Delta^\mathcal{R}$. Then, for all atoms $a \in \mathcal{A}$ and for all $t \in T$, $a \in \upsilon(t)$ iff there exists an interval $\langle t_1, t_2\rangle \in \mathcal{I}(a)$ such that $t \in [t_1, t_2]$.*

In summary, we have shown how LARS operators can be naturally used to reason over time intervals. However, the presented intuitive approach is less expressive than ETALIS, where an atom can be assigned to overlapping intervals. On the other hand, the minimal model of the monotonic ETALIS semantics can be constructed by computing fixed-points for intervals of increasing size. Hence, with an explicit encoding of intervals $\langle t_1, t_2\rangle$ into atoms that contain t_1 and t_2 as terms, one can mimic the bottom-up evaluation of such models with ASP and thus also with LARS. It is a research topic on its own to find a suitable extension of LARS for nonmonotonic complex event processing that builds upon an evaluation $\upsilon : T \times T \to 2^{\mathcal{A}}$ mapping *intervals* to atoms.

Related Work

In the Semantic Web area, the SPARQL language was extended to queries on streams of RDF triples; respective engines such as CQELS (Phuoc et al. 2011), C-SPARQL (Barbieri et al. 2010), and $\text{SPARQL}_{\text{Stream}}$ (Calbimonte, Corcho, and Gray 2010) follow the snapshot semantics approach of CQL. However, they face difficulties with extensions incorporating the Closed World Assumption, nonmonotonicity, or nondeterminism. Such features are important to deal with missing or incomplete data, which can, e.g., temporarily happen due to unstable network connections or hardware failure. In this case, these engines remain idle, while some output based on default reasoning might be useful.

In KR&R, first attempts towards expressive stream reasoning have been recently carried out and reveal many open problems. The plain approach of Do, Loke, and Liu (2011) periodically calls the dlvhex solver (Eiter et al. 2006) without incremental reasoning and thus cannot handle heavy data load. StreamLog (Zaniolo 2012) extends Datalog towards stream reasoning, based on stratification (which guarantees a single model), while OSMS (Ren and Pan 2011) considers streams of ontologies. Both StreamLog and OSMS have no window mechanisms. Time-decaying logic programs (Gebser et al. 2012) aim to implement time-based windows in reactive ASP (Gebser et al. 2008), whose relation to other stream processing/reasoning approaches is unexplored.

Moreover, as observed by Dindar et al. (2013), conceptually identical queries may produce different results on different engines. This may be due to differences (i.e., flaws) in implementations, but might also arise from (correct implementations of) different semantics. Comparisons between different approaches are confined to experimental analysis (Phuoc et al. 2012) or informal examination on specific examples. For the user it is important to know the exact capabilities and semantic behaviors of given approaches for systematic analysis and comparison.

Conclusion

We presented LARS, an expressive rule-based modelling language to formalize and analyze stream reasoning semantics. It provides an idealized model-based, nonmonotonic semantics as timestamps are not dropped but can be abstracted away. LARS allows to distinguish truth of a formula at (i) specific time points, but also (ii) at some resp. every time point in a window; furthermore, it offers general window operators that may be nested, which enables reasoning over streams within the language. We then considered the computational complexity of LARS, its capability to express CQL, and how to encode a stream reasoning view on ETALIS' complex event processing in it.

LARS is a starting point for intriguing research issues. Informally or operationally specified semantics of various state-of-the-art stream processing/reasoning engines such as CQELS, C-SPARQL, and $\text{SPARQL}_{\text{Stream}}$ may now be formalized, studied and compared rigorously in a common language. For practical concerns, tractable and efficient fragments of LARS are of interest; related to this are operational characterizations of its semantics. Later, along the lines of (Brewka et al., 2014), we aim at a formalism for stream reasoning in distributed settings across heterogeneous nodes that have potentially different logical capabilities.

References

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26(11):832–843.

Anicic, D.; Fodor, P.; Rudolph, S.; Stühmer, R.; Stojanovic, N.; and Studer, R. 2010. A rule-based language for complex event processing and reasoning. In *RR 2010*, 42–57.

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

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

Barbieri, D. F.; Braga, D.; Ceri, S.; Della Valle, E.; and Grossniklaus, M. 2010. C-SPARQL: a continuous query language for rdf data streams. *Int. J. Semantic Computing* 4(1):3–25.

Beck, H.; Dao-Tran, M.; Eiter, T.; and Fink, M. 2014. Towards Ideal Semantics for Analyzing Stream Reasoning. In *ReactKnow*.

Brewka, G.; Ellmauthaler, S.; and Pührer, J. 2014. Multi-Context Systems for Reactive Reasoning in Dynamic Environments. In *ECAI*, 159–164.

Calbimonte, J.-P.; Corcho, Ó.; and Gray, A. J. G. 2010. Enabling ontology-based access to streaming data sources. In *ISWC (1)*, 96–111.

Dadashzadeh, M., and Stemple, D. W. 1990. Converting SQL queries into relational algebra. *Information & Management* 19(5):307–323.

Della Valle, E.; Ceri, S.; van Harmelen, F.; and Fensel, D. 2009. It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems* 24:83–89.

Dindar, N.; Tatbul, N.; Miller, R. J.; Haas, L. M.; and Botan, I. 2013. Modeling the execution semantics of stream processing engines with SECRET. *VLDB J.* 22(4):421–446.

Do, T. M.; Loke, S. W.; and Liu, F. 2011. Answer set programming for stream reasoning. In *AI*, 104–109.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. dlvhex: A prover for semantic-web reasoning under the answer-set semantics. In *Web Intelligence*, 1073–1074.

Faber, W.; Leone, N.; and Pfeifer, G. 2004. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In *JELIA*, 200–212.

Garcia-Molina, H.; Ullman, J. D.; and Widom, J. 2009. *Database systems - the complete book (2nd ed.)*. Pearson Education.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Thiele, S. 2008. Engineering an incremental ASP solver. In *ICLP*, 190–205.

Gebser, M.; Grote, T.; Kaminski, R.; Obermeier, P.; Sabuncu, O.; and Schaub, T. 2012. Stream Reasoning with Answer Set Programming. Preliminary Report. In *KR*, 613–617.

Phuoc, D. L.; 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 (1)*, 370–388.

Phuoc, D. L.; Dao-Tran, M.; Pham, M.-D.; Boncz, P.; Eiter, T.; and Fink, M. 2012. Linked stream data processing engines: Facts and figures. In *ISWC - ET*, 300–312.

Ren, Y., and Pan, J. Z. 2011. Optimising ontology stream reasoning with truth maintenance system. In *CIKM*, 831–836.

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