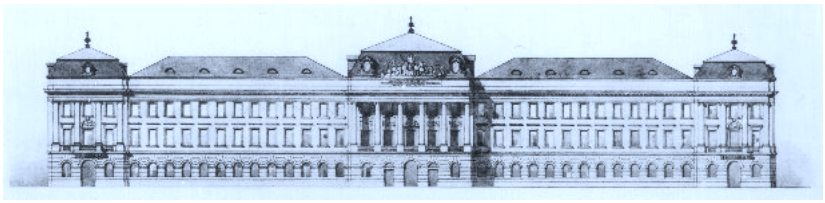
INFSYS
RESEARCH
REPORT



INSTITUT FÜR INFORMATIONSSYSTEME

ARBEITSBEREICH WISSENSBASIERTE SYSTEME

LARS: A LOGIC-BASED FRAMEWORK FOR ANALYTIC REASONING OVER STREAMS

HARALD BECK AND MINH DAO-TRAN AND THOMAS EITER

INFSYS RESEARCH REPORT 17-03

OCTOBER 2017

Institut für Informationssysteme
AB Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 17-03, OCTOBER 2017

LARS: A LOGIC-BASED FRAMEWORK FOR ANALYTIC REASONING OVER STREAMS

Harald Beck[1] Minh Dao-Tran[1] Thomas Eiter[1]

Abstract. The increasing availability of streaming data has accelerated advances in information processing tools that no longer store data for static querying but push information to consumers as soon as it becomes available. *Stream processing* aims at providing languages and tools for data that changes at a high rate. To cope with the volume of data, query languages often extend existing approaches for static data by means of *window operators* that return snapshots of recent data. However, the semantics of these languages are often given only informally or operationally, which makes their analysis and comparison difficult. A formal means to express the declarative semantics of such systems seems to be missing. This lack of theory is of particular relevance for the emerging research in *stream reasoning* which shifts the focus from throughput to higher expressiveness. To fill this gap, we present LARS, a Logic-based framework for Analytic Reasoning over Streams. At its core, LARS *formulas* extend propositional logic with generic window operators and additional controls to handle temporal information. On top of this, LARS *programs* extend Answer Set Programming (ASP) for rich stream reasoning capabilities; the latter can be exploited to target AI applications in a streaming context, such as diagnosis, configuration or planning. Specifically, we study in this article the computational complexity of LARS formulas and programs, their relationship to Linear Temporal Logic (LTL) and the well-known Continuous Query Language (CQL). Furthermore, we discuss the modelling capabilities of LARS in notes on the SPARQL extensions C-SPARQL and CQELS, and on the interval-based approach of the complex event processing language ETALIS. We finally briefly touch available implementations, in particular, the recent prototype engines Laser and Ticker that aim for high throughput and high expressiveness, respectively. Notably, both engines specify their semantics in LARS, indicating the desired flexibility of the framework and its potential as stream reasoning language itself, which is further explored in other works.

***Keywords* —** Answer Set Programming, Stream Reasoning, Dynamic Data

---

[1]Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria; email: {beck,dao,eiter}@kr.tuwien.ac.at.

Acknowledgements: This research has been supported by the Austrian Science Fund (FWF) projects P26471 and W1255-N23.

Some bof the concepts and results were presented in preliminary form at AAAI-2015 (Beck et al., 2015).

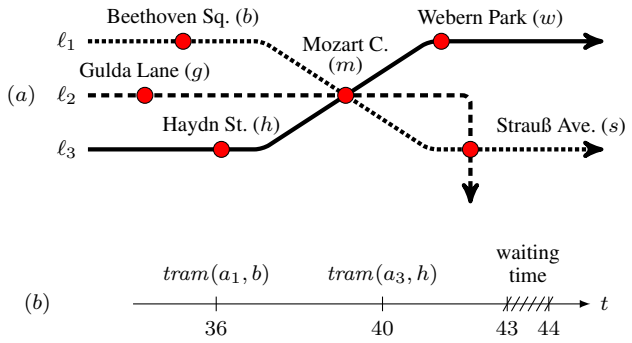


Figure 1: (a) Transportation map (b) Timeline (minutes)

1 Introduction

During the last decade, *stream reasoning* (Della Valle et al., 2009; Mileo et al., 2017) emerged as research topic from *stream processing* (Stephens, 1997; Babu & Widom, 2001) to address the semantic level of querying continuously changing data. The increasing amount and availability of data from sensors, networks, mobile devices, etc., has contributed to a shift in information processing from pulling static data on request to continuous pushing as soon as it is available. Many applications that deal with such potentially infinite streams of data make use of *window operators* to select recent snapshots of the unbounded input. Typical windows are obtained based on time or by counting tuples; they may progress instantly, or in larger steps. Window mechanisms have shown to be a useful ingredient in stream processing and reasoning for multiple reasons. First, some form of data limitation is often needed for pragmatic purposes, i.e., to cope with the amount of information. Second, viewing streams as sequence of relations, i.e., databases, allows for reusing available tools for processing static data. Third, many real-time use cases are inherently only concerned about the recent past, as the next example illustrates.

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 D) between stops X and Y with line L is stored by rows $plan(L, X, Y, D)$. 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 ill-suited for strollers (old).

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

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

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

(2) 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 aspects of reasoning over data streams, with different focus (see also Dell'Aglio, Della Valle, van Harmelen, & Bernstein, 2017):

– In the area of Data Stream Management Systems (DSMS), the focus is typically on "low-level" stream processing, i.e., queries on data that arrives at high input rate. Such queries center around cross-joins of data, elementary pattern matching, etc. Performance and scalability are important criteria, which has led to windows as a key means to deal with large data volumes, taking also physical implementation design (buffers, etc.) into account.

Particularly influential work in this area has been the Stanford Data Stream Management System (STREAM) (Arasu et al., 2003a) and its continuous query language (CQL) (Arasu et al., 2003b, 2006). It extends SQL with window operators and essentially reduces the semantics of stream processing to SQL-based queries over obtained snapshots.

– In Knowledge Representation and Reasoning (KR&R), the approaches aim at stream reasoning,[1] where implicit information is elicited from data in the input stream and a background knowledge base, represented for instance in a Description Logic or as a logic program. The focus is naturally put on higher-level (abstracted) information and lower-rate input streams. Often, stream reasoning is viewed as temporal reasoning over a sequence of knowledge bases extended with data. Windows or snapshots have not yet played a prominent role, or are limited to very restrictive forms, cf. (Do et al., 2011; Gebser et al., 2008; Zaniolo, 2012; Ren & Pan, 2011; Özcep et al., 2015).

– In the Semantic Web area, the Linked Data movement has been striving to lift stream data to a semantic level. To this end, linked stream data aims at coupling tuples with timestamps, in order to facilitate temporal data processing. Similarly as CQL extends SQL, several proposals have been made to add window mechanisms to SPARQL, the standard query language for RDF data. These works include, e.g., C-SPARQL (Barbieri et al., 2010), CQELS (Phuoc et al., 2011), and $\text{SPARQL}_{\text{Stream}}$ (Calbimonte et al., 2010). The semantics of according queries is typically given only informally or on an operational basis, hiding potentially significant differences in results behind similar syntax. In fact, comparisons of these languages have been geared towards high data frequency and mere output volume of queries (i.e., number of tuples) than concrete tuples (i.e., contents), not their semantics.

Motivation for a formal framework. The developments sketched above yield a landscape of different approaches for stream reasoning. While they often share conceptual ideas, their exact commonalities and differences remain unclear without a common theoretical underpinning, in which its declarative semantics may be expressed, analyzed, and compared. The lack of theoretical underpinnings for stream reasoning has been observed already in (Della Valle et al., 2009). In particular, the authors propose that a theoretical framework for stream reasoning must combine two aspects: first, it has to serve as basis for explicit formal semantics, and second, it must account for high-throughput, i.e., frequency and volume of data.

From a theoretical perspective, the trade-off between expressiveness and scalability is evident. In particular, some portions of the data might have a higher frequency and volume than others, and the potential

[1] There is no general consensus on how to discriminate stream reasoning from stream processing; opinions on this have diverged by large at a Stream Reasoning Workshop in Vienna, Nov. 2015, even whether stream reasoning is more general than stream processing or vice versa, or whether declarative (model-based) vs. operational (code-based) definition of semantics should play a role.

difficulty of reasoning does not imply that all operations are highly complex. A theoretical foundation for stream reasoning should thus aim to cover the entire spectrum, i.e., provide means to express both semantically trivial real-time computations as well as complex reasoning tasks that are necessarily slower; possibly within the same query or program. This can only be achieved by a modular system, where the mechanisms to handle streams (like window operators for deliberate information loss) can be used in a generic and compositional way.

Regarding the expressiveness, a rich framework should encompass advanced reasoning features as available in KR&R. In particular, this includes intensional data definitions, i.e., the ability to abstract from (extensional) input data. Moreover, nonmonotonicity is of special relevance in stream reasoning, since conclusions may have to be retracted due to later arrival of previously missing information (e.g., in case of defaults) or contrary evidence (in case of contradictions). Next, model generation as in SAT solving and Answer Set Programming (ASP) are useful when tackling domains which permit multiple solutions. Such features and according techniques have been studied almost exclusively on static data, where in particular extensions of Datalog (Ceri et al., 1990) play a prominent role. Zaniolo (2012) already noted a lack of logical foundation for Data Stream Management Systems. We aim here at providing one for stream reasoning with generic windows.

Contributions. Based on the above considerations, we have conceived LARS, a Logic-based Framework for Analytic Reasoning over Streams. Our contributions can be briefly summarized as follows.

– *Modeling streams and windows.* As a basis for a declarative semantics, we provide a formal model of streams and a generic notion of window function that can be applied to streams; our notion can be instantiated to a plethora of window functions that are used in practice, including time-based windows (where data is selected based on temporal constraints), tuple-based windows (selection on order constraints), and partition-based windows (selection by a mix of order and semantic information), and filter windows (semantics information). Notably, the result of applying a window to a stream yields a (sub)-stream, such that window functions can be composed (i.e., nested). This allows one to express complex data snapshots, based on a repertoire of selections that are available. In turn, composed data selections can be abstractly viewed as a single selection by a respective window function.

– *LARS language.* The LARS framework provides two languages for reasoning over streams, referred to as *LARS formulas* and *LARS programs* respectively. The former language is the monotone core of LARS, i.e., a first-order formalism to reason over extensional data only. Inspired by the way how data in stream processing is handled, the central entailment definition extends propositional logic by novel operators (i) to limit data by generic window operators and (ii) to specify a temporal modality of formulas within obtained snapshots. That is to say, LARS formulas express validity at some, all, or specific time points in a window. The propositional core is then considered also in a schematic way to allow for variables that serve as placeholders for domain values. In Example 1, the formula $\boxplus^{+5}\Diamond exp(a_3, m)$ could informally state that tram a_3 is expected to arrive at station m within the next five minutes; $\boxplus^{+5}\Diamond exp(X, Y)$ is a schematic version with variables X and Y.

LARS programs are an (in essence second-order) language for more involved applications that require reasoning over auxiliary (i.e., intensional) predicates. Programs are sets of rules similar as in Datalog but built upon LARS formulas instead of atoms as elementary expressions. In order to deal with incomplete information and negation, programs are equipped with a multiple model semantics as in Answer Set Programming (Gelfond & Lifschitz, 1991; Baral, 2003; Brewka, Eiter, & Truszczyński, 2011, 2016) to obtain preferable properties such as nonmonotonicity, model minimality and supportedness of inferences (Section 3.3). In

fact, LARS programs can be seen as an extension of Answer Set Programming for stream reasoning. In Example 1, the rule $poss(X, m) \leftarrow \boxplus^{+5}\Diamond exp(X, m), \neg old(X)$. could informally state that when a tram X is expected to arrive at station m (Mozart Circus) within the next five minutes and is not an old make, then it is a possible target for further consideration.

– *Computational complexity and expressiveness.* We investigate the computational complexity of model checking and satisfiability for both LARS formulas and programs, over an input data stream, assuming that window evaluation is tractable. Our analysis pays particular attention to nesting of windows and the use of different windows types that are common in practice. Our analysis reveals that both problems are PSpace-complete for propositional (ground) LARS formulas and programs in general, but have lower complexity if either the nesting depth of window operators is bounded by a constant, or only common window operators as those mentioned above are used, under some mild constraint. In particular, reasoning in LARS is then not harder than in ASP; notably, this includes the most practical programs which employ no window nesting. For non-ground LARS formulas and programs (i.e., the Datalog case), the combined complexity of satisfiability testing increases (up to $\mathsf{NExpTime}^{\mathsf{NP}}$) in the considered setting, while for model checking it does not drastically increase; for formulas, it remains unchanged. The data complexity is, as one might expect, in line with the complexity of the (bounded) ground case. Regarding expressiveness, LARS formulas can express only (and in general all) polynomial time recognizable languages. We show that in case of sliding time-based windows, propositional LARS formulas can be translated into linear-time temporal logic (LTL), and thus *a fortiori* can express only a strict fragment of the regular languages. While sliding time-based windows are easily described in LTL, expressing others (e.g., sliding tuple-based windows) is more involved, and even impossible for polynomial-time windows in general. Propositional LARS programs instead are shown to capture the class of regular languages, i.e., every regular language and only regular languages can be expressed by propositional LARS programs. As non-ground LARS program subsume disjunctive Datalog and are not harder to evaluate, these programs capture the class of Σ_2^p recognizable languages, and are thus a rather expressive formalism.

– *LARS use cases.* The use of LARS as framework allows for formally defining and comparing existing languages, in particular when they are defined only operationally or informally. By means of LARS one also may equip existing formalisms with a declarative semantics. We explore the modelling capabilities of LARS by drawing formal relations with different prominent languages: we capture the semantics of the (core of the) continuous query language (CQL) that extends SQL for streams, and we discuss C-SPARQL and CQELS that similarly extend SPARQL for RDF streams. Finally, we consider ETALIS (Anicic et al., 2010), which is also rule-based, but different from LARS aims at complex event processing and builds on time intervals rather than time points.

Furthermore, LARS can be used as a genuine language for stream reasoning itself. In particular, a fragment called *plain LARS* has been utilized in multiple works mentioned in Section 6.1. It extends normal logic programs essentially by so-called *extended atoms* for controlling the streaming aspects. Plain LARS programs have been used to model the decision unit in a simulation framework for Content-Centric Networking Management (Beck et al., 2016, 2017).

In summary, we have established with LARS a novel formalism to express and analyze stream reasoning, from both the theoretical and the practical perspective. Due to its linkage to ASP, it provides a uniform basis for developing AI applications in a streaming context, such as diagnosis, configuration, planning and many others, cf. (Erdem, Gelfond, & Leone, 2016). Further works complement the seminal results presented here and address equivalence issues for optimization (Beck et al., 2016), incremental answer update to queries (Beck et al., 2015), and prototype implementation (Beck et al., 2016, 2017; Bazoobandi et al., 2017).

Organization. The remainder of this article is organized as follows. The next section introduces the basic constituents of the model-theoretic semantics of our framework, viz. streams and window functions. Section 3 then presents the LARS language, where syntax and semantics of both LARS formulas and programs are defined. This is followed by a computational complexity analysis in Section 4. Section 5 investigates the relationship of LARS to selected other formalisms, among them CQL, LTL, C-SPARQL, and CQELS. while Section 6 provides a discussion of further work on LARS and puts LARS into the broader context of stream processing and reasoning. In the final Section 7 we draw conclusions and outline directions for future research. In order not to disrupt the flow of reading, proofs of technical results have been moved to the Appendix.

2 Streams and Windows

In line with the setting of many practical systems for stream processing, we adopt a discrete, linear time ontology and model windows as data snapshots that select data from a given input stream, based on different principles. Notably, the result of applying a window function to a stream is another (sub)-stream; this allows for nesting of windows and thus to express complex data selections. The generic notion of window is then instantiated to widely used concrete window functions of different type.

2.1 Streaming data

We use mutually disjoint finite sets of *predicates* $\mathcal{P}$, *constants* $\mathcal{C}$, *variables* $\mathcal{V}$ and *time variables* $\mathcal{U}$. The set $\mathcal{T}$ of *terms* is given by $\mathcal{C} \cup \mathcal{V}$ and the set $\mathcal{A}$ of *atoms* is defined as $\{p(t_1, \ldots, t_n) \mid p \in \mathcal{P}, t_1, \ldots, t_n \in \mathcal{T}\}$. The set $\mathcal{G}$ of *ground atoms* contains all atoms $p(t_1, \ldots, t_n) \in \mathcal{A}$ such that $\{t_1, \ldots, t_n\} \subseteq \mathcal{C}$. We also say a term is ground if it is a constant.

We divide $\mathcal{P}$ into two disjoint subsets, namely the *extensional predicates* $\mathcal{P}^{\mathcal{E}}$ and the *intensional predicates* $\mathcal{P}^{\mathcal{I}}$. Accordingly, we distinguish *extensional atoms* $\mathcal{A}^{\mathcal{E}}$ and *intensional atoms* $\mathcal{A}^{\mathcal{I}}$. Intensional predicates/atoms are used to express inferred information. On the other hand, extensional predicates (respectively atoms) are further partitioned into $\mathcal{P}^{\mathcal{E}}_B$ (resp. $\mathcal{A}^{\mathcal{E}}_B$) for *background data* and $\mathcal{P}^{\mathcal{E}}_S$ (resp. $\mathcal{A}^{\mathcal{E}}_S$) for *data streams*. The mentioned partitions are analogously defined for ground atoms $\mathcal{G}^{\mathcal{I}}$, $\mathcal{G}^{\mathcal{E}}_B$ and $\mathcal{G}^{\mathcal{E}}_S$. With slight abuse of notation, $\mathcal{G}$ usually refers to $\mathcal{G}^{\mathcal{I}} \cup \mathcal{G}^{\mathcal{E}}_S$, i.e., ground atoms that are not reserved for background data.

Additionally, we assume basic arithmetic operations ($+$, $-$, $\times$, $\div$) and comparisons ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$) are predefined by designated atoms $\mathcal{B} \subseteq \mathcal{A}^{\mathcal{E}}_B$, and used also in infix notation.

We now present the central notion of streams, which we base on the linear time ontology $\langle \mathbb{N}, \leq \rangle$; informally the increase by 1 is the passing of time in terms of a tick by a global system clock.

Definition 1 (Stream). Let T be a closed nonempty interval in $\mathbb{N}$ and $\upsilon \colon \mathbb{N} \to 2^{\mathcal{G}}$ 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*.

We write evaluation functions also as set of nonempty mappings. For instance, $\{17 \mapsto \{a, b\}\}$ assigns $\{a, b\}$ to time point 17, and $\emptyset$ else.

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 timeline $T = [0, 50]$ and evaluation $\upsilon(36) = \{tram(a_1, b)\}$, $\upsilon(40) = \{tram(a_3, h)\}$, and $\upsilon(t) = \emptyset$ for all $t \in T \setminus \{36, 40\}$. With the mapping notation, we simply write $\{36 \mapsto \{tram(a_1, b)\}$, $40 \mapsto \{tram(a_3, h)\}\}$. ∎

2.2 Window Functions

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). Any (computable) function w that returns, given a stream $S = (T, \upsilon)$, and a time point $t \in \mathbb{N}$, a window S' of S, is called a *window function*.

The most common types of windows in practice include time-, tuple-, and partition-based windows. We associate them with three window functions symbols τ, $\#$, and 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. Moreover, we introduce a *filter window* function f which only drops data but retains the current timeline.

Orthogonal to the selection mechanism of a window is the way it progresses. Figures 2-4 show three typical ways a standard time-based window of size $n = 3$ progresses. First, Figure 2 depicts the *sliding* window, where from the query time t (indicated by a bullet), always window $[t - n, t]$ is selected. The right end of the window does not necessarily have to equal the query time. If the window shifts in greater intervals (than 1), we get a *hopping* window as shown in Figure 3. Here, the *hop size* is 2, i.e., the window shifts to the right every 2 time points. If the hop size equals the window size, we get a *tumbling* window as shown in Figure 4. Sliding windows can be seen as a means for a fully continuous evaluation of recent events. Hopping windows additionally allow to specify a certain interval after which a re-evaluation is needed. This is of interest when we want to control the time when a result shall be refreshed, e.g., a condition over the last 60 seconds which shall be recomputed only every 15 seconds. Finally, the tumbling window is suitable when we want to partition the timeline, e.g., evaluate something in fixed intervals of 60 seconds.

Note that depending on available parameters, these principles of how a window progresses extends to other windows as well, in particular, for tuple-based windows. We will now introduce generalized versions of the main windows.

2.3 Time-based Window

Definition 3 (Time-based window). Let $S = (T, \upsilon)$ be a stream, $T = [t_{min}, t_{max}]$, $t \in \mathbb{N}$, $\ell, u \in \mathbb{N} \cup \{\infty\}$ and $d \in \mathbb{N}$ such that $d \leq \ell + u$. If $t \in T$, the *time-based window function* with *range* (ℓ, u) and *hop size* d of S at time t is defined by

$$\tau^{\ell,u,d}(S, t) = (T', \upsilon|_{T'}), \tag{1}$$

where T'=$[t_\ell, t_u]$, $t_\ell = \max\{t_{min}, t' - \ell\}$ with *pivot point* $t' = \lfloor \frac{t}{d} \rfloor \cdot d$, and $t_u = \min\{t' + u, t_{max}\}$. If $t \notin T$, we define $\tau^{\ell,u,d}(S, t) = S$.

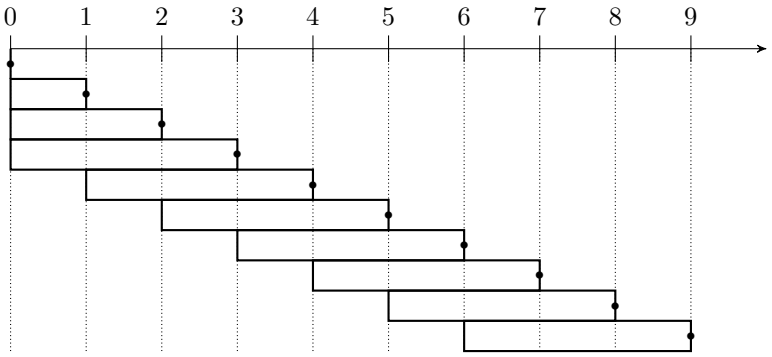


Figure 2: Sliding time-based window of size 3, bullets indicate query times

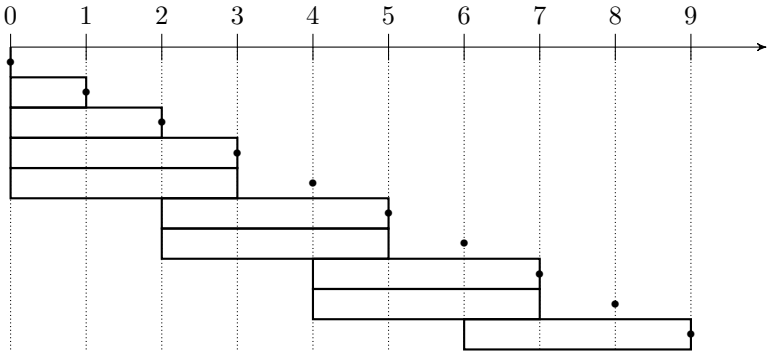


Figure 3: Hopping time-based window of size 3 and hop size 2, bullets indicate query times

The *size* of a time-based window function $\tau^{\ell,u,d}$ is given by $\ell + u$. Note that the case for $t \notin T$ is given only for formal reasons, i.e., compliance with Definition 2. Conceptually, the time-based window is only applicable if $t \in T$. The general approach is useful for a compositional approach as discussed later.

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, at every minute. To this end, we can use a time-based window function $\tau^{4,0,1}$. Applying it on D at $t = 42$ gives $\tau^{4,0,1}(D, 42) = ([38, 42], \upsilon')$, where $\upsilon' = \{40 \mapsto \{tram(a_3, h)\}\}$. ∎

A time-based window function $\tau^{\ell,u,d}$ is called *sliding*, if $d = 1$, and *tumbling*, if $d > 1$ and $d = \ell + u$, else *hopping*. We abbreviate $\tau^{\ell,0,1}$ by τ^{ℓ} and $\tau^{0,u,1}$ by τ^{+u}.

Example 4. Figure 5 shows the initial progress of a hopping time-based window function $\tau^{3,1,3}$ that also looks one time point into the future. Note that the pivot point stays the same for every three consecutive time points due to the hop size $d = 3$. ∎

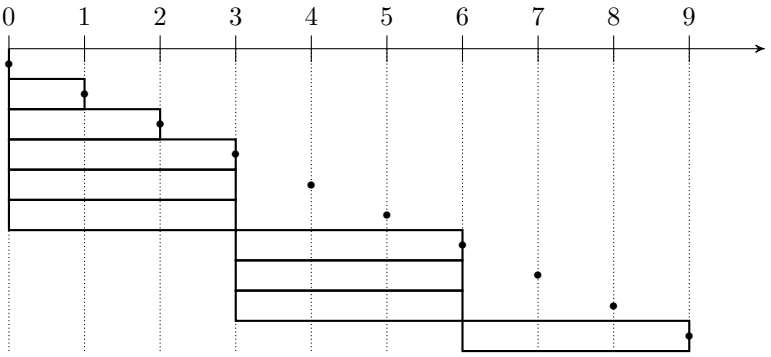


Figure 4: Tumbling time-based window of size 3, bullets indicate query times

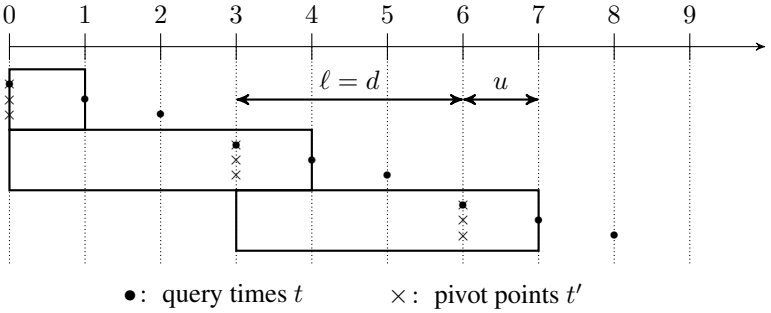


Figure 5: Generalized time-based window $\tau^{3,1,3}$

2.4 Tuple-based Window

Definition 4 (Tuple-based window). Let $S = (T, \upsilon)$ be a stream, $T = [t_{min}, t_{max}]$ and $t, \ell, u \in \mathbb{N}$. We define the *tuple time bounds* t_ℓ, t_u as

$$t_\ell = \max\{t_{min}\} \cup \{t' \mid t_{min} \leq t' \leq t \ \wedge \ \#(S|_{[t',t]}) \geq \ell\}, \text{ and}$$
$$t_u = \min\{t_{max}\} \cup \{t' \mid t \leq t' \leq t_{max} \ \wedge \ \#(S|_{[t+1,t']}) \geq u\}.$$

Let $T_\ell = [t_\ell, t]$ and $T_u = [t+1, t_u]$. If $t \in T$, the *tuple-based window with range* (ℓ, u) *of* S *at time* t is defined by

$$\#^{\ell,u}(S,t) = (T', \upsilon'|_{T'}), \text{ where } T' = [t_\ell, t_u], \text{ and}$$

$$\upsilon'(t') = \begin{cases} \upsilon(t') & \text{for all } t' \in T' \setminus \{t_\ell, t_u\} \\ \upsilon(t') & \text{if } t' = t_\ell \text{ and } \#(S|_{T_\ell}) \leq \ell \\ X_\ell & \text{if } t' = t_\ell \text{ and } \#(S|_{T_\ell}) > \ell \\ \upsilon(t') & \text{if } t' = t_u \text{ and } \#(S|_{T_u}) \leq u \\ X_u & \text{if } t' = t_u \text{ and } \#(S|_{T_u}) > u \end{cases}$$

where $X_q \subseteq \upsilon(t_q)$, $q \in \{\ell, u\}$, such that $\#(T_q, \upsilon'|_{T_q}) = q$. If $t \notin T$, we define $\#^{\ell,u}(S,t) = S$.

Note that the tuple-based window is unique only if for both $q \in \{\ell, u\}$, $\upsilon'(t_q) = \upsilon(t_q)$, i.e., if all atoms at the endpoints of the selected interval are retained. There are two natural possibilities to enforce the uniqueness of a tuple-based window. First, if there is a total order over all atoms, one can give a deterministic definition

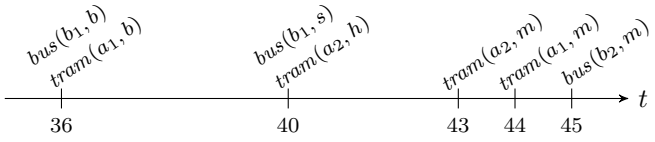


Figure 6: Scenario of Figure 1 (b) extended

of the sets X_q in Definition 4. Second, one may omit the requirement that *exactly* ℓ tuples of the past, resp. u tuples of the future are contained in the window, but instead demand the substream obtained by the smallest interval $[t_\ell, t_u]$ containing *at least* ℓ past and u future tuples. Note that this approach would simplify the definition to $\#^{\ell,u}(S,t) = (T', \upsilon|_{T'})$, requiring only to determine $T' = [t_\ell, t_u]$.

Example 5. To illustrate tuple-based window, we extend the input stream in Example 1 with facts of the form $bus(ID, X)$ to denote the appearance of buses at stops. Figure 6 depicts the new scenario. The data stream is $D = (T, \upsilon)$ where $T = [0, 50]$ and

$$\upsilon = \left\{ \begin{array}{l} 36 \mapsto \{tram(a_1, b), bus(b_1, b)\}, \\ 40 \mapsto \{tram(a_2, h), bus(b_1, s)\}, \\ 43 \mapsto \{tram(a_2, m)\}, \\ 44 \mapsto \{tram(a_1, m)\}, \\ 45 \mapsto \{bus(b_2, m)\} \end{array} \right\}$$

To get the last four and the next vehicle appearances w.r.t. a reference time point $t = 43$, we can use the tuple-based window function $\#^{4,1}$, which gives two possible substreams at t: $S_1 = (T_1, \upsilon_1)$ and $S_2 = (T_2, \upsilon_2)$, where for both $j \in \{1, 2\}$, $T_j = [36, 44]$,

$$\begin{array}{l} \upsilon_j(40) = \{tram(a_2, h), bus(b_1, s)\}, \\ \upsilon_j(43) = \{tram(a_2, m)\}, \\ \upsilon_j(44) = \{tram(a_1, m)\}; \end{array}$$

$\upsilon_1(36) = \{bus(b_1, b)\}$ and $\upsilon_2(36) = \{tram(a_1, b)\}$. That is to say, the two windows differ in the evaluation at time point 36, where a nondeterministic choice is made to pick exactly four elements from the input stream from time point 43 back to 36. ∎

2.5 Partition-based Window

We recall that $\mathcal{G}$ denotes the set of ground atoms, which is the basis for defining the index function idx of the partition-based window. Each index corresponds to a substream, on which a tuple-based window is applied. We obtain from a substream $S = (T, \upsilon)$ a substream $\mathrm{idx}_i(S) = (T, \upsilon_i)$ by taking $\upsilon_i(t) = \{a \in \upsilon(t) \mid \mathrm{idx}(a) = i\}$.

Definition 5 (Partition-based window). Let $S = (T, \upsilon)$ be a stream, $t \in \mathbb{N}$, and let $I \subset \mathbb{N}$ be a finite index set. Let $\mathrm{idx} : \mathcal{G} \to I$ and $n : I \to \mathbb{N} \times \mathbb{N}$ be total functions, $n(i) = (\ell_i, u_i)$. Moreover, for all $i \in I$, let $S_i = \mathrm{idx}_i(S)$ and $\#^{\ell_i,u_i}(S_i, t) = ([t_i^\ell, t_i^u], \upsilon_i')$ be the tuple-based window with range (ℓ_i, u_i) of S_i at time t. If $t \in T$, the *partition-based window of S at time t (relative to* idx, n*)* is defined by

$$\mathrm{p}^{\mathrm{idx},n}(S,t) = (T', \upsilon'), \text{ where } T' = [\min_{i \in I} t_i^\ell, \max_{i \in I} t_i^u]$$

and $\upsilon'(t') = \bigcup_{i \in I} \upsilon_i'(t')$ for all $t' \in T'$. If $t \notin T$, we define $\mathrm{p}^{\mathrm{idx},n}(S,t) = S$.

Note that, in contrast to schema-based streaming approaches, we assume multiple kinds of tuples (predicates) in a single stream. Whereas other approaches may use tuple-based windows of different counts on separate streams, we can have separate tuple-counts on the corresponding substreams of a partition-based window on a single stream.

Example 6 (cont'd). Continue with the extended scenario in Example 5, to get the last appearance of each tram/bus instance until time point $t = 45$, we use the partition-based window function $\mathrm{p}^{\mathrm{idx},n}$, where:

$$\begin{aligned}
&\mathrm{idx}(\mathit{tram}(a_i, X)) = i,\\
&\mathrm{idx}(\mathit{bus}(b_i, X)) = i + 2,\\
&\mathrm{idx}(c, X) = 0 \text{ for all } c \in \mathcal{G} \setminus \{\mathit{tram}(a_i, Y), \mathit{bus}(b_j, Z)\},\\
&n(i) = (1, 0) \text{ for } i > 0,\\
&n(0) = (0, 0).
\end{aligned}$$

This gives us four substreams for D at 45:

$$\begin{aligned}
S_1 &= ([36, 45], \{36 \mapsto \{\mathit{tram}(a_1, b)\}, 44 \mapsto \{\mathit{tram}(a_1, m)\}\})\\
S_2 &= ([40, 45], \{40 \mapsto \{\mathit{tram}(a_2, h)\}, 43 \mapsto \{\mathit{tram}(a_2, m)\}\})\\
S_3 &= ([36, 45], \{36 \mapsto \{\mathit{bus}(b_1, b)\}, 40 \mapsto \{\mathit{bus}(b_1, s)\}\})\\
S_4 &= ([45, 45], \{45 \mapsto \{\mathit{bus}(b_2, m)\}\})
\end{aligned}$$

Applying the tuple-based windows with $n(i) = (1, 0)$, $i > 0$, results in:

$$\begin{aligned}
S'_1 &= ([44, 45], \{44 \mapsto \{\mathit{tram}(a_1, m)\}\})\\
S'_2 &= ([43, 45], \{43 \mapsto \{\mathit{tram}(a_2, m)\}\})\\
S'_3 &= ([40, 45], \{40 \mapsto \{\mathit{bus}(b_1, s)\}\})\\
S'_4 &= ([45, 45], \{45 \mapsto \{\mathit{bus}(b_2, m)\}\})
\end{aligned}$$

Finally, the union of S'_1 to S'_4 gives us the result of the partition-based window:

$$\mathrm{p}^{\mathrm{idx},n}(D, 45) = ([40, 45], \upsilon),$$

where

$$\upsilon = \{40 \mapsto \{\mathit{bus}(b_1, s)\}, 43 \mapsto \{\mathit{tram}(a_2, m)\}, 44 \mapsto \{\mathit{tram}(a_1, m)\}, 45 \mapsto \{\mathit{bus}(b_2, m)\}\}.$$ ∎

The partition-based window works in three steps. First, in partitions the stream into substreams, second, it applies a tuple-based window on each of them, and third, it merges together the result. If the tuple-based window is needed only on a single substream, we may use a filter window instead of a designated index function.

Time-based:	$\boxplus^{\ell,u,d} := \boxplus^{\tau^{\ell,u,d}}$	$\boxplus^{\ell} := \boxplus^{\tau^{\ell,0,1}}$	$\boxplus^{+u} := \boxplus^{\tau^{0,u,1}}$
Tuple-based:	$\boxplus^{\#\ell,u} := \boxplus^{\#^{\ell,u}}$	$\boxplus^{\#\ell} := \boxplus^{\#^{\ell,0}}$	$\boxplus^{\#+u} := \boxplus^{\#^{0,u}}$
Partition-based:	$\boxplus^{\mathrm{idx},n} := \boxplus^{\mathrm{p}^{\mathrm{idx},n}}$		
Filter:	$\boxplus^{A} := \boxplus^{\mathrm{f}^{A}}$		

Figure 7: Definition of window operator shortcuts

2.6 Filter Window

For a set $A \subseteq \mathcal{G}$ of atoms, we define the *projection* of υ to A by $\upsilon|_A(t) = \upsilon(t) \cap A$ for all $t \in \mathbb{N}$.

Definition 6 (Filter window). Let $S = (T, \upsilon)$ be a stream, $t \in \mathbb{N}$, and $A \subseteq \mathcal{G}$ be a set of atoms. The *filter window function for* A (at time t) is defined by

$$\mathrm{f}^A(S, t) = (T, \upsilon|_A). \tag{2}$$

Note that the filter window function is essentially independent of time, i.e., it always retains the timeline and returns the same result for all $t \in \mathbb{N}$, in particular, for $t \notin T$. Thus, the filter windows selects atoms independent of time and the timeline can remain entirely. This is dual to the time-based window, where all atoms in the selected timeline can remain, and this timeline is selected independent of atoms. In this view, the partition-based window can be seen as the application of tuple-based windows on the result of filter windows, which are then merged together.

Example 7 (cont'd). Selecting the last appearance of tram a_1 amounts to first filtering for atoms of form $A = \{tram(a_1, st) \mid st \in \mathcal{C}\}$, and then selecting the last tuple from this intermediate stream. That is, by $\mathrm{f}^A(D, 45)$ we get $D' = (T, \upsilon')$, where $\upsilon' = \{36 \mapsto \{tram(a_1, b)\}, 44 \mapsto \{tram(a_1, m)\}\}$. The tuple-based window of size 1 of D' at 45 then returns $([44, 45], \{44 \mapsto \{tram(a_1, m)\}\})$. ∎

3 The LARS Framework

We now present an improved version of LARS (Beck et al., 2015), a Logic for Analytic Reasoning over Streams. LARS extends propositional logic by for streaming data by employing any window function w (Def. 2) in a window operator $\boxplus^w$. Within the resulting substream, one can then control the temporal modality of formulas, resp. access temporal information. Based on such formulas, LARS then provides a rule-based language with a model-based, nonmonotonic semantics, which can be seen an extension of Answer Set Programming for streaming data.

Before defining syntax and semantics of LARS below, we first present the central concepts informally.

Window operator & reset. If w is a window function, we call $\boxplus^w$ a *window operator*. Given a formula α, the expression $\boxplus^w\alpha$ has the effect that α will be evaluated on the window obtained by applying w in the current stream S at the current time t. Dually, the *reset operator* $\triangleright$ serves to re-access the original stream.

Temporal modalities. Regardless if formula evaluation is on the entire input stream or a window thereof, we provide explicit means to deal with the temporal information. Let $S = (T, \upsilon)$ be a stream, i.e., the input stream or a window, and $t \in T$ be a time point. There are different ways to evaluate a formula α in S at t. First, we express by $@_{t'}\alpha$, where $t' \in \mathbb{N} \cup \mathcal{U}$, that α has to hold when changing the evaluation time to t'. We call t' in $@_{t'}\alpha$ a *time pin*; which is *ground* if $t' \in \mathbb{N}$, else *non-ground*. Next, time might be abstracted away. That is to require that α holds at *some* time point $t' \in T$, denoted by $\Diamond\alpha$. Dually, $\Box\alpha$ shall hold iff α holds at *all* time points in T. Based on these modalities, we define our language.

3.1 LARS Formulas

Definition 7 (Formulas). Let $a \in \mathcal{A}$ be an atom, $t \in \mathbb{N} \cup \mathcal{U}$ and w be a window function. The set $\mathcal{F}$ of *formulas* is defined by the following grammar:

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \Diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus^w\alpha \mid \triangleright\alpha \tag{3}$$

The set $\mathcal{F}_\mathcal{G}$ of *ground formulas* contains all formulas where each term and each time pin is ground. In addition to streams, we consider background knowledge in form of static data, i.e., a set $B \subseteq \mathcal{G}_B^\mathcal{E}$ of ground 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.

The following definitions concern the semantics of ground formulas.

Definition 8 (Structure). Let $S = (T, \upsilon)$ be a stream, W be a set of window functions and $B \subseteq \mathcal{G}_B^\mathcal{E}$ a set of facts. Then, we call $M = \langle S, W, B\rangle$ a *structure*, S the *interpretation stream* and B the *background data* of M.

We now define when a ground formula holds in a structure.

Definition 9 (Entailment). Let $M = \langle S^\star, W, B\rangle$ be a structure, $S^\star = (T^\star, \upsilon^\star)$ and let $S = (T, \upsilon)$ be a substream of $S^\star$. Moreover, let $t \in T^\star$. The *entailment* relation $\Vdash$ between (M, S, t) and formulas is defined as follows. Let $a \in \mathcal{G}$ be an atom, let $\alpha, \beta \in \mathcal{F}_\mathcal{G}$ be ground formulas and $w \in W$. Then,

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

If $M, S, t \Vdash \alpha$ holds, we say that (M, S, t) *entails* α. Moreover, we say that 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 8 (cont'd). Let $D = (T, \upsilon)$ be the data stream of Example 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 $M = \langle S^\star, W, B\rangle$, where $W = \{\tau^{0,5,1}\}$, and B is the set of facts from the data tables in Example 1. Then $M, S^\star, 42 \Vdash \boxplus^{+5}\Diamond exp(a_3, m)$ holds: the window operator $\boxplus^{+5}$ 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 $t' \in T'$ ($t' = 43$) such that $M, S', t' \Vdash exp(a_3, m)$. ∎

We note that the original presentation of LARS (Beck et al., 2015) employed a so-called *stream choice* in window operators that allowed to direct the window function to be applied on the original stream $S^\star$ (stream choice 1) and the current stream S (stream choice 2). Definition 9 presents a cleaner approach, where a window operator is always applied on the current stream. In case the original stream needs to be re-accessed in nested windows, this can be done by an explicit reset step $\triangleright$, followed by a window operator $\boxplus^w$. We define this combination as *input window operator* $\boxtimes^w := \triangleright\boxplus^w$.

Example 9. Consider a monitoring use case where a signal s must always appear within 5 minutes. Testing whether this condition holds for the last hour amounts to the formula $\boxplus^{60}\Box\,\boxtimes^5\Diamond s$: We first select by a sliding time-based window the last 60 minutes. At every time point in this window, it must hold that if we consider the last 5 minutes there, signal s holds at some time point. Notably, by using $\boxtimes^5$ instead of $\boxplus^5$ we ensure that this inner window reaches beyond the limits of the first. For instance, consider a stream $S = ([0, 500], \upsilon)$. First, at $t = 500$, $\boxplus^{60}$ selects $S' = (T', \upsilon|_{T'})$, where $T' = [440, 500]$. During evaluation of $\Box$ at $t' = 440$, $\boxtimes^5$ now selects $S|_{[435,440]}$, while $\boxplus^5$ would select $S'|_{[440,440]}$, since the timeline in S' only starts at 440. ∎

Another subtle improvement over the previous version (Beck et al., 2015) concerns the fact that a window operator $\boxplus^w$ may return a substream that does not contain the evaluation time point, i.e., $w(S, t) = (T', \upsilon')$ does not imply $t \in T'$. However, given a (sub)formula $\boxplus^w\varphi$, one typically wants to evaluate φ in the obtained window regardless of the specific evaluation time and its position relative to the window. While this could be technically handled for relevant cases, we now consider time points $t \in T^\star$, and not $t \in T$. That is to say, the evaluation time point t needs to be contained only in the global timeline $T^\star$, not in the timeline T of the current substream S.

Example 10. Consider again Figure 4, which illustrates the progress of a tumbling time-based window of size 3, i.e., the function $\tau^{3,0,3}$. Assume further we are interested whether an atom x occurs in this window when evaluated at time 8. Accordingly, we evaluate $M, S, 8 \Vdash \boxplus^{3,0,3}\Diamond x$ and the substream returned by $\tau^{3,0,3}$ has timeline $[3, 6]$. We still expect that the entailment holds iff x appears within $[3, 6]$, regardless of the fact that $8 \notin [3, 6]$. ∎

Notably, allowing any $t \in T^\star$ in formula evaluation not only serves the applicability of windows such as hopping or tumbling windows. It also allows one to inspect whether the evaluation time is contained in the current timeline T. This possibility stems from the requirement that t' is contained in T for $@_{t'}\alpha$ to hold: the standard tautology $\top := a \vee \neg a$ holds (in all structures) at every time point $t \in T^\star$, where entailment is defined; however, $@_{t'}\top$ holds if and only if $t' \in T$. Since also time points $t \in T^\star \setminus T$ can be evaluated, formulas can express conditions based on T. For instance, $M, S, t \Vdash \boxplus^w(@_t\top \wedge \varphi)$ holds only if t is contained in the timeline of $w(S, t)$.

Queries & non-ground formulas. We now consider the use of variables, leading to open formulas and queries.

Definition 10 (Query). Let $M = \langle S, W, B\rangle$ be a structure, $\alpha \in \mathcal{F}$ be a formula and let $u \in \mathbb{N} \cup \mathcal{U}$. Then, the tuple $Q = \langle M, u, \alpha\rangle$ is called a *query*. We say Q is *ground* if α and u are ground, else *non-ground*.

Given a ground query $Q = \langle M, t, \alpha\rangle$, where $M = \langle S, W, B\rangle$, we define the *answer* $?Q$ to Q as yes, if $M, S, t \Vdash \alpha$ holds, else no.

To define the semantics of non-ground queries, we need the notions of a *substitution* σ, defined as mapping $\mathcal{V} \cup \mathcal{U} \to \mathcal{C} \cup \mathbb{N}$ that assigns (i) each variable $V \in \mathcal{V}$ a constant $\sigma(V) \in \mathcal{C}$, and (ii) each time

variable $U \in \mathcal{U}$ a natural number $\sigma(U) \in \mathbb{N}$. The *grounding* $\sigma(\alpha)$ (resp. $\sigma(u)$) of formula α (resp. time pin u) due to σ is obtained by applying the substitutions on variables/time variables as usual. Given a timeline T, we say a substitution σ is *over* $(\mathcal{C}, T)$, if the image of σ is contained in $\mathcal{C} \cup T$; we denote by $\sigma(\mathcal{C}, T)$ the set of all such substitutions. With this, we define the *answer* $?Q$ to a non-ground query $Q = \langle M, u, \alpha \rangle$ by

$$?Q = \{\sigma \in \sigma(\mathcal{C}, T) \mid M, S, \sigma(u) \Vdash \sigma(\alpha)\}. \tag{4}$$

This definition gives a general semantics to two important subclasses of non-ground queries $Q = \langle M, u, \alpha \rangle$. First, if α is ground and $u \in \mathcal{U}$ is is a time variable, then the answer to Q amounts to the time points when α holds. Dually, if $u \in \mathbb{N}$ and α is non-ground, we obtain a semantics for non-ground formula evaluation at a fixed time point.

For queries, the set of window functions W in a stated structure $M = \langle S, W, B \rangle$ is implicitly given by the window operators used in α. Unless stated otherwise, B is assumed to be empty.

Example 11. Consider the stream $S = (T, \upsilon)$ as shown in Figure 6, with $T = [30, 50]$. We ask:

Q_1 At $t = 45$, which trams arrived at which stations in the last 5 minutes?

Q_2 At $t = 45$, at which times and which stations did tram a_2 arrive in the last 5 minutes?

Q_3 At which times did we record a tram arrival at a station, where a bus arrived within the next 3 minutes?

We formalize these queries as $Q_1 = \langle M, 45, \alpha_1 \rangle$, $Q_2 = \langle M, 45, \alpha_2 \rangle$, and $Q_3 = \langle M, U, \alpha_3 \rangle$, where

$$\begin{aligned} \alpha_1 &= \boxplus^5 \Diamond \mathit{tram}(A, St), \\ \alpha_2 &= \boxplus^5 @_U \mathit{tram}(a_2, St), \text{ and} \\ \alpha_3 &= \mathit{tram}(A, St) \wedge \boxplus^{+3} \Diamond \mathit{bus}(B, St). \end{aligned}$$

We obtain the following answers:

$$\begin{aligned} ?Q_1 &= \{\, \{A \mapsto a_2, St \mapsto h\}, \{A \mapsto a_2, St \mapsto m\}, \{A \mapsto a_1, St \mapsto m\} \,\} \\ ?Q_2 &= \{\, \{St \mapsto h, U \mapsto 40\}, \{St \mapsto m, U \mapsto 43\} \,\} \\ ?Q_3 &= \{\, \{A \mapsto a_1, St \mapsto b, U \mapsto 36, B \mapsto b_1\}, \{A \mapsto a_2, St \mapsto m, U \mapsto 43, B \mapsto b_2\}, \\ &\qquad \{A \mapsto a_1, St \mapsto m, U \mapsto 44, B \mapsto b_2\} \,\} \end{aligned}$$

■

We observe that the operator @ allows for replaying a historic query. At any time $t' > t$, we can ask $\langle M, t', @_t \alpha \rangle$ to simulate a previous query $\langle M, t, \alpha \rangle$. In fact, this applies for any $t' \in \mathbb{N}$.

Example 12 (cont'd). Consider again Q_1 from Example 11, where α_1 is evaluated at $t = 45$, and an answer $\sigma \in ?Q_1$. Then, for any time point $t' \in \mathbb{N}$, σ is also an answer of $\langle M, t', @_{45} \alpha_1 \rangle$, since by definition, $M, S, 45 \Vdash \sigma(\alpha_1)$ iff $M, S, t' \Vdash @_{45} \sigma(\alpha_1)$ and $45 \in T$. ■

Nested windows. Typically, window functions are used exclusively to restrict the processing of streams to a recent subset of the input. In our view, window functions provide a flexible means to discard data.

Example 13. In Example 7, we selected the last appearance of tram a_1 in stream D of Figure 6 by first using a filter window for tram atoms $A = \{\mathit{tram}(a_1, st) \mid st \in \mathcal{C}\}$, and followed by a tuple-based window. We now also ask at which time U the tram was last recorded. The according LARS query $Q = \langle M, 45, \alpha \rangle$ is given by $M = \langle D, \{\mathrm{f}^A, \#^1\}, \emptyset \rangle$ and $\alpha = \boxplus^A \boxplus^{\#1} @_U \mathit{tram}(a_1, St)$, which has the single answer $\sigma = \{St \mapsto m, U \mapsto 44\}$. That is, tram a_1 last appeared at station m at time 44. ■

LARS formulas provide a powerful, flexible language to query streaming data. However, the formalism presented so far has no means of expressing auxiliary information, i.e., intensional atoms, and thus comes with limitations.

Example 14. Dual to query Q_3 in Example 11, we now want to ask for which tram appearances *no* bus arrived within 3 minutes at the same station. The intended answer to this query should only contain tram a_2 at station h at time 40. A naive translation simply adds negation to α_3, i.e., $\alpha_3' = tram(A, St) \wedge \neg \boxplus^{+3} \Diamond bus(B, St)$. However, the resulting query $Q_3' = \langle M, U, \alpha_3' \rangle$ expresses the following: At which time points U did some tram appear at station St, where within the next 3 minutes there was no bus recording at St for any B, i.e., for any constant that can be substituted for B. Thus, whenever a tram a_i is at station st at time t, we hypothetically consider any atom $bus(x, St)$, where $x \in \mathcal{C}$, at time points $n = t$, $t+1$, $t+2$, $t+3$, and get an answer of form $\{A \mapsto a_i, St \mapsto st, B \mapsto x, U \mapsto t\}$, whenever there $bus(x, st)$ is not in the evaluations from $\upsilon(t)$ to $\upsilon(t+3)$ in D. That results in answers like the following:

$$\begin{gathered}
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto a_1, U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto a_2, U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto b_2, U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto s,\; U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto h,\; U \mapsto 36\} \\
\vdots \\
\{A \mapsto a_2,\, St \mapsto h, B \mapsto b_1, U \mapsto 40\} \\
\vdots
\end{gathered}$$

Clearly, we can limit scope of B by considering only constants that have been observed as bus identifiers so far, using $\alpha_3'' = tram(A, St) \wedge \neg \boxplus^{+3} \Diamond bus(B, St) \wedge \Diamond bus(B, St')$. The additional subformula $\Diamond bus(B, St')$ now matches all bus appearances throughout the stream and will thus be joined with every tram appearance $tram(A, St)$ (at time U). Semantically, this cross product yields potential answers of form $\{A \mapsto a_i, St \mapsto st, B \mapsto x, St' = st', U \mapsto t\}$, which are reduced by those entries for which $bus(x, st)$ appeared between t and $t+3$. For instance, $\{A \mapsto a_1, St \mapsto b, B \mapsto b_1, St \mapsto y, U \mapsto 36\}$, where $y \in \{b, s, m\}$, is not returned since $bus(b_1, b)$ appeared at 36. We get, among others, the following answers:

$$\begin{gathered}
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto b_2, St' \mapsto b,\; U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto b_2, St' \mapsto s,\; U \mapsto 36\} \\
\{A \mapsto a_1,\, St \mapsto b,\; B \mapsto b_2, St' \mapsto m, U \mapsto 36\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_1, St' \mapsto b,\; U \mapsto 40\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_1, St' \mapsto s,\; U \mapsto 40\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_1, St' \mapsto m, U \mapsto 40\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_2, St' \mapsto b,\; U \mapsto 40\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_2, St' \mapsto s,\; U \mapsto 40\} \\
\{A \mapsto a_2,\, St \mapsto h,\; B \mapsto b_2, St' \mapsto m, U \mapsto 40\} \\
\vdots
\end{gathered}$$

The first question is how the result shall be interpreted. The non-essential St' does not capture anything of the conceptual query, which does not talk about stations of arbitrary bus stations. To remedy this, a simple post-processing may filter out such bindings and reduce resulting duplicates accordingly. However, this still leaves wrong results. For instance, the first three answers would reduce to $\{A \mapsto a_1, St \mapsto b, B \mapsto b_2, U \mapsto$

$36\}$. What this answer says is that bus b_2 did not arrive at station b within 3 minutes, but we intended to query for tram appearances after which *no* bus arrived. ∎

The fundamental problem in Example 14 is that we query for substitutions of a bus identifier B when we are interested in cases where none exists. That is, we have to abstract away from specific bus existences which can only be expressed by auxiliary atoms, i.e., intensional atoms. Thus, towards more expressive reasoning over data streams, we now introduce LARS programs.

3.2 LARS Programs

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

Definition 11 (Rule, Program). A *program* P is a set of *rules*, i.e., expressions of the form

$$\alpha \leftarrow \beta_1, \ldots, \beta_n, \tag{5}$$

where $\alpha, \beta_1, \ldots, \beta_n \in \mathcal{F}$ are formulas.

Given a rule r of form (5), $H(r)$ denotes the *head* α, and $\beta(r) = \beta_1 \wedge \ldots \wedge \beta_n$ the *body* of r; the commas in (5) are a syntactic variant of $\wedge$ as usual.

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. Any rule r amounts to the material implication

$$\beta(r) \rightarrow H(r). \tag{6}$$

Consider a structure $M = \langle I, W, B\rangle$, called an *interpretation (for D)*. We then say

- M is a *model of P for D at time t*, denoted $M, t \models P$, if $M, t \models \beta(r) \rightarrow \alpha(r)$ for all rules $r \in P$;
- M a *minimal model*, if no model $M' = \langle S', W, B\rangle$ of P for D at time t exists such that $S' = (T, \upsilon')$ and $S' \subset S$.

Note that minimality is defined w.r.t. to the same timeline T. We often omit "for D" and/or "at t" if this is clear from the context. The *reduct* of a program P with respect to 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 12 (Answer Stream). Let $M = \langle I, W, B\rangle$ 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), if M is a $\subseteq$-minimal model of the reduct $P^{M,t}$ for D at time 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_{ASP} be a program where in each rule of form (5) all body formulas β_i are literals, i.e., atoms or negated atoms, and the head α is a disjunction of atoms. Then, we have:

Proposition 1. For I and P_{ASP} as described, I is an answer stream of P for D at t relative to arbitrary W and B iff $\upsilon'(t)$ is an answer set of $P_{ASP} \cup \upsilon(t) \cup B$.

That is, ordinary answer set programs are subsumed by LARS programs.

Non-ground programs. As for formulas, we consider non-ground programs as schematic versions of ground programs with variables of two sorts, namely constant variables $\mathcal{V}$ and time variables $\mathcal{U}$. The semantics of these *non-ground 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 15 (cont'd). We now solve the problem of Example 14 as follows: the intention of formula α'_3 is formalized as rule r_1 which uses the intensional atom $aBus$, derived by rule r_2:

$$\begin{aligned} r_1: &\quad q(U,A,St) \leftarrow @_U\, tram(A,St), @_U \neg \boxplus^{+3} \Diamond aBus(St);\\ r_2: &\quad @_U\, aBus(St) \leftarrow @_U\, bus(B,St). \end{aligned}$$

Here, q is the output relation which may be used for post processing and $aBus$ is the intended abstraction for the appearance of any bus at the given station St. Rule r_2 assigns to any time point U an (intensional) atom $aBus(St)$ whenever there is an atom $bus(B,St)$ at U.

Apart from the expressiveness issue, intensional atoms may also be used to enhance readability. For instance, the complex formula $@_U \neg \boxplus^{+3} \Diamond aBus(St)$ in r_1 may be simplified, giving the subformula $\boxplus^{+3}\Diamond aBus(St)$ a name on its own. This leads to the following approach:

$$\begin{aligned} r'_1: &\quad q(U,A,St) \leftarrow @_U\, tram(A,St), @_U \neg busSoon(St),\\ r'_2: &\quad @_U\, busSoon(St) \leftarrow @_U \boxplus^{+3} \Diamond bus(B,St). \end{aligned}$$

Note further that rule r'_2 may be written without the use of a window as

$$@_U\, busSoon(St) \leftarrow @_{U'}\, bus(B,St),\ U' > U,\ U' - U \leq 3.$$

Using the program $P = \{r'_1, r'_2\}$, we now get the intended result in the single answer stream $I = (T, \upsilon^I)$ at $t = 45$: the evaluation function υ^I contains the assignments of intensional atoms

$$\begin{aligned} 33,34,35,36 &\mapsto \{busSoon(b)\},\\ 37,38,39,40 &\mapsto \{busSoon(s)\},\\ 42,43,44,45 &\mapsto \{busSoon(m)\},\ \text{and additionally}\\ 45 &\mapsto \{q(40,a_2,h)\}. \end{aligned}$$

Derivation $q(40,a_2,h)$ correctly reflects that only at station h no bus appeared within 5 minutes after a tram appearance (tram a_2 at minute 40). ∎

Particular semantic assets for LARS programs are inherited from Answer Set Programming, i.e., a multiple-model semantics permitting nonmonotonic reasoning.

Example 16 (cont'd). The requests (i) and (ii) from Example 1 can be formulated by rules (7) and (8), respectively.

$$\begin{aligned} @_T\, exp(ID,Y) \ \leftarrow\ & \boxplus^{\mathrm{idx},n}\, @_{T_1}\, tram(ID,X), line(ID,L), \neg \boxplus^{20} \Diamond jam(X),\\ & plan(L,X,Y,D), T = T_1 + D. \end{aligned} \tag{7}$$

$$\begin{aligned} gc(ID_1,ID_2,X) \ \leftarrow\ & @_T\, exp(ID_1,X), @_T \boxplus^{+5} \Diamond exp(ID_2,X),\\ & ID_1 \neq ID_2, \neg old(ID_2). \end{aligned} \tag{8}$$

Rule (7) encodes when a tram is expected at later stops. For the partition-based window operator $\boxplus^{\mathrm{idx},n}$, we use $\mathrm{idx}(at) = i$ for an atom at of form $tram(a_i, X)$ and $\mathrm{idx}(at) = 0$ 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 (8) builds on the expected arrival times of rule (7) to identify good connections where the targeted tram is not an old make 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 8 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 17 (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 (7) 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 time 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:

$$change(ID_1, ID_2, X) \leftarrow on(ID_1), gc(ID_1, ID_2, X), \neg skip(ID_1, ID_2, X). \tag{9}$$

$$skip(ID_1, ID_2, X) \leftarrow gc(ID_1, ID_2, X), change(ID_1, ID_3, X), ID_2 \neq ID_3. \tag{10}$$

Consider the program P consisting of rules (7)-(10). 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

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

Then, I_1 and I_2 are (the only) two answer streams for P at time 42 relative to $W = \{\tau, \mathrm{p}\}$ 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 18 (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. ∎

3.3 Semantic Properties of LARS Programs

In this subsection, we show that some basic properties of the answer semantics of logic programs carry over to the notion of answer stream defined above. These are minimality of answer streams, supportedness by rules and consistency, i.e., existence of an answer stream in the absence of negation provided that the windows functions involved are monotonic, i.e., return growing substreams if the stream data increases.

Let P be a program, D be a data stream and $t \in \mathbb{N}$. By $AS(P, D, t)$ we denote the set of answer streams of P for D at time t. The letter M always stands for the structure $M = \langle I, W, B\rangle$, where I is the considered answer stream, and W and B are implicit and fixed.

By Definition 12, the structure M (due to answer stream I) is a minimal model of the reduct $P^{M,t}$ for D at time t. Importantly, this implies that M is a model of the original program P, and in fact a minimal model.

Theorem 1 (Minimality of answer streams). Let P be a LARS program, D be a data stream, t be a time point and $I \in AS(P, D, t)$. Then, $M = \langle I, W, B\rangle$ is a minimal model of P for D at time t.

Thus, answer streams warrant the property of minimality that answer sets enjoy, in the spirit of logic programming semantics. A simple consequence of minimality of models is the following.

Corollary 1 (Incomparability). Answer streams are incomparable w.r.t. $\subseteq$. That is, if $I, I' \in AS(P, D, t)$, then $I \neq I'$ implies $I \not\subseteq I'$ and $I' \not\subseteq I$.

Our definition of answer streams follows the approach in (Faber et al., 2004), which requires a supporting rule for every derived atom. In other words, dropping any atom from an answer set would invalidate some rule. In our case, dropping an intensional atom a from an answer stream would lead to an unsatisfied rule that supports its derivation for some time point t'. To simplify notation, we consider $I = (T, \upsilon)$ also as set $\{t' \mapsto a \mid a \in \upsilon(t'), t' \in T\}$. Accordingly, $I \setminus \{t' \mapsto a\}$ amounts to removing in I atom a from $\upsilon(t')$, etc.

Theorem 2 (Supportedness). Let $I \in AS(P, D, t)$. Then, for every $t' \mapsto a \in I \setminus D$ there exists a rule $r \in P$ such that (i) $M, t \models \beta(r)$ and (ii) $M', t \not\models r$, where $M' = \langle I \setminus \{t' \mapsto a\}, W, B\rangle$.

Note that the conditions (i) and (ii) in Theorem 2 amount for ordinary logic programs to the usual notion of supportedness of answer sets; if the rule head $\alpha = a_1 \vee \cdots \vee a_k$ in (ii) is a disjunction of atoms, then M must satisfy a single atom a_i in α, and $a_i = a$.

Finally, let us consider LARS programs in which α and each formula β_i are positive, i.e., each atom occurs in the formula tree only under an even number of negations; we call such programs *positive*. As for windows, we naturally call a window function w *monotonic*, if for any streams S and S' such that $S \subseteq S'$ and for any time t' it holds that $w(S, t') \subseteq w(S', t')$. Then we obtain

Theorem 3 (Consistency). Let P be a positive LARS program such that all heads α of rules in P are satisfiable and all window operator $\boxplus^w$ occurring in P have monotonic window functions w. Then for any D and t, (i) $AS(P, D, t) \neq \emptyset$ and (ii) any $M = \langle I, W, B\rangle$ is a minimal model of $P^{M,t}$ at t iff $I \in AS(P, D, t)$.

For example, time-based sliding windows are monotonic and likewise the other time-based windows considered above; furthermore, also filter windows are monotonic. Tuple-based sliding windows (thus also partition-based windows) are not monotonic, and the statement in the theorem does not hold, even for very restricted rule syntax.

Example 19. Consider the program P consisting of the rules

$$\begin{aligned}
r_0 &: \quad c. \\
r_1 &: \quad d \leftarrow \boxplus^{\#1}\Diamond c. \\
r_2 &: a \wedge b \leftarrow \quad d. \\
r_3 &: \quad b \leftarrow \boxplus^{\#1}\Diamond a. \\
r_4 &: \quad a \leftarrow \boxplus^{\#1}\Diamond b.
\end{aligned}$$

and assume that the tie-break in the tuple selection is by lexicographic ordering, i.e., a before b before c before d. Informally, $\boxplus^{\#1}\Diamond x$ expresses that the single selected tuple is x. Then $M = \langle I, W, B\rangle$ where (in abuse of notation) $I = \{a, b, c\}$ is a model of P for the data stream $D = ([0, 0], \emptyset)$ at $t = 0$. Moreover, it is the single minimal model for D at t: for any other model $M' = \langle I', W, B\rangle$, we have that $d \in I'$ implies $I' = \{a, b, c, d\}$ (by r_0 and r_2), which is not minimal. Furthermore, $\{a, b\} \cap I' = \emptyset$ would lead by r_1 and r_2 to $\{a, b\} \subseteq I'$, which is contradictory; similarly $I' = \{c, a\}$ (resp. $I' = \{c, b\}$) would lead by r_3 to $b \in I'$ (resp. by r_4 to $a \in I'$), which is again a contradiction. Thus, M is the only answer stream candidate. However, the reduct $P^{M,t} = \{r_0;\ r_3\}$ has a model $M' = \langle I', W, B\rangle$ for D at t where $I' = \{c, b\}$. Thus, M is not an answer stream of P for D at $t = 0$ and $AS(P, D, t) = \emptyset$ follows. (The same holds if we replace r_2 with rules $a \leftarrow d$ and $b \leftarrow d$; the resulting program is in the *plain* LARS fragment; cf. Section 6.1.1.) ∎

However, refined versions of tuple-based windows, which e.g. count only extensional data (as occurs often in practice), are monotonic and thus admissible in Theorem 3; furthermore, monotone windows can be nested as monotonicity is preserved. We remark that the consistency result (part (i) of the theorem) can be extended to classes of programs with layered (stratified) negation and recursion through non-monotonic windows.

We note that Definition 11 is liberal in the sense that it permits extensional atoms also in rule heads. This is convenient in some scenarios with complex rule heads. Notably, any answer stream for a data stream D may only add intensional atoms to D. Thus, satisfaction of extensional atoms in rule heads anyway hinges on the input D; it is not possible to infer input data. If desired, one may rewrite a program in order to exclude extensional atoms from rule heads. To this end, one replaces every extensional predicate p (that is mentioned in a rule head) by a fresh intensional predicate p' and adds the rules $@_T\, p' \leftarrow @_T\, p$, $@_T \neg p' \leftarrow @_T \neg p$ (or

Problem	formula class α	formula class α^-	program class P	program class P^-
Model Checking (MC)	PSpace	P	PSpace	co-NP
Satisfiability (SAT)	PSpace	NP	PSpace	Σ_2^p

Table 1: Complexity of reasoning in ground LARS (completeness results)

for the latter rule, alternatively the constraint $\bot \leftarrow \Diamond(p' \wedge \neg p)$). In case a ground program is required that works for all inputs, one may alternatively use the following set of rules, where p'' is another fresh predicate:

$$\begin{aligned} \Box(p' \vee p'') &\leftarrow . \\ \bot &\leftarrow \Diamond(p \wedge p''). \\ \bot &\leftarrow \Diamond(p' \wedge p''). \end{aligned}$$

Clearly, using these encodings, the answer streams of the original program P and of the rewritten program P' are in one-to-one correspondence. Moreover, we note that for programs that do not use extensional predicates in rule heads, data streams can be reduced to programs without extensional data, by replacing any input atom $p \in \upsilon(t)$ in a data stream $D = (T, \upsilon)$ by the fact $@_t\, p \leftarrow$.

In conclusion, we find that LARS programs have basic semantic properties comparable to those of ordinary answer set programs under the FLP-reduct. They can thus be seen as an extension of ASP for use cases in streaming with flexible window functions.

4 Computational Complexity of Reasoning in LARS

In this section, we analyze the computational complexity of LARS, where we consider model checking and the satisfiability problem, for both LARS formulas and programs. In our analysis, we concentrate on the general case but pay attention to the effect of nested windows and particular classes of windows; a comprehensive study of the computational complexity for a rich taxonomy of classes of LARS formulas and LARS programs remains however for further study.

4.1 Problem Statements and Overview of Results

We say that a stream $S = (T, \upsilon)$ *is over* a subset $\mathcal{A}' \subseteq \mathcal{A}$ of atoms $\mathcal{A}$, if $\upsilon(t) \setminus \mathcal{A}' = \emptyset$ for all $t \in T$. We study the complexity of the following reasoning tasks, where in the sequel W is a set of window functions that are evaluable in polynomial time, $B \subseteq \mathcal{A}$ is a set of background atoms, and where α is a ground LARS formula and P a ground LARS program.

(1) *Model Checking* (MC). Given $M = \langle S^\star, W, B\rangle$, $S^\star = (T, \upsilon)$, and $t \in T$, check whether

- for a given stream $S \subseteq S^\star$ and formula α it holds that $M, S, t \Vdash \alpha$; respectively
- $I = (T, \upsilon)$ is an answer stream of a given program P for a data stream $D \subseteq I$ at t.

(2) *Satisfiability* (SAT). For decidability, we assume that relevant atoms are confined to a subset $\mathcal{A}' \subseteq \mathcal{A}$ of polynomial size in the input. The reasoning tasks are:

- Given W, B, a timeline T, a time point $t \in T$, and a formula α, does some stream $S = (T, \upsilon)$ over $\mathcal{A}'$ exist such that $M, S, t \Vdash \alpha$, where $M = \langle S, W, B \rangle$?
- Given W, B, a data stream D with timeline T, a time point $t \in T$, and a program P, does P have some answer stream for D at t that is over $\mathcal{A}'$?

Table 1 shows the computational complexity of reasoning in ground LARS, where α^- and P^- denotes the class of formulas respectively programs where the nesting depth of window operators in formulas and rules is bounded by a constant.

As we can see from the Table 1, in the general case model checking and satisfiability checking are both PSpace-complete, and thus beyond the Polynomial Hierarchy. Informally, the recursive evaluation of a formula creates an exponential size tree, but at each point in time, only a polynomial size portion of this tree needs to be in memory. The PSpace-hardness arises from the temporal operators $\Box$ and $\Diamond$ in combination with window operators. This allows for encoding quantified Boolean formulas (QBFs), whose evaluation is a canonical PSpace-complete problem.

The picture changes if we bound the window nesting depth in formulas and programs. Under a constant bound, the evaluation tree that is built has only polynomially many nodes (i.e., substreams). This allows us to solve the model checking problem for ground α^- formulas in polynomial time by using labeling techniques. The remaining results for satisfiability testing and for ground LARS programs are obtained from guess and check algorithms. The lower bounds (the hardness results) are in essence inherited from the complexity of answer set programs, except for model checking of LARS formulas. The P-hardness of the latter problem is due to the generic form of windows whose associated functions can be P-hard.

Note that from the results on Model Checking in Table 1, we immediately obtain complexity results for answering ground queries $Q = \langle M, t, \alpha \rangle$: the problem is PSpace-complete in general, but polynomial for bounded window-nesting; and as the discussion in Section 4.3 shows, this generalizes to a richer class of queries.

4.2 Derivation of the Complexity Results

4.2.1 LARS Formulas

The complexity results for LARS formulas in the general case are based on the following result for model checking.

Theorem 4. Given a structure $M = \langle S^\star, W, B \rangle$, a stream $S = (T, \upsilon)$ such that $S \subseteq S^\star$, a time point t, and an arbitrary ground formula α, deciding $M, S, t \Vdash \alpha$ is PSpace-complete, where the PSpace-hardness holds for $S = S^\star$.

Intuitively, PSpace-membership is shown by a depth-first-search evaluation of the input formula α along its tree representation. An example for the formula tree is shown in Figure 8.

At each node of the tree, we need to store the content according to the window operators that are applied as in the path from the root. This requires only polynomial space for that node and all nodes on the path to it as well.

The PSpace-hardness is shown by a reduction from evaluating QBFs $\exists x_1 \forall x_2 \cdots Q_n x_n \phi(\mathbf{x})$ to model checking. A LARS formula $\alpha = \Diamond \boxplus^{\mathrm{set}:x_1} \Box \boxplus^{\mathrm{set}:x_2} \cdots \phi(\mathbf{x})$ on the timeline $T = [0, 1]$ is constructed where the window operator $\boxplus^{\mathrm{set}:x_i}$ effects the possible truth assignments to x_i at the time points 0 or 1. To this end, the initial stream $S^\star$ has all atoms $x_1, \ldots, x_n$ at both 0 and 1. When $\boxplus^{\mathrm{set}:x_i}$ is evaluated at

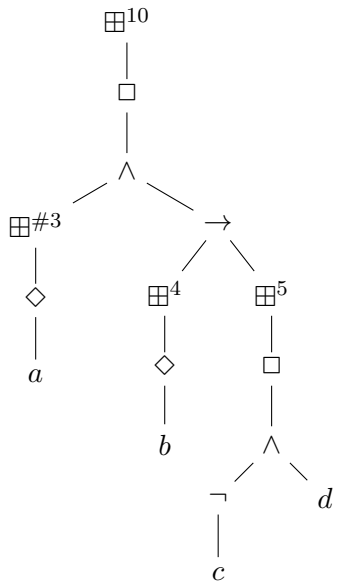


Figure 8: Tree representation of formula $\boxplus^{10}\Box(\boxplus^{\#3}\Diamond a \wedge (\boxplus^{4}\Diamond b \rightarrow \boxplus^{5}\Box(\neg c \wedge d)))$

time point 0 (resp. 1), it removes x_i from (resp. keeps x_i in) the stream. By branching to 0 or 1, all truth assignments to $x_1, \ldots, x_n$ are generated in an evaluation tree. On top, $\Diamond$, $\Box$ naturally encode the quantifiers $\exists$ and $\forall$. Figure 9 shows an example evaluation tree: the bold lines mark subtrees for which, given the assignment to $x_1, \ldots, x_i$ by the path, the subformula $Q_{i+1}x_{i+1} \ldots Q_n x_n \phi(\mathbf{x})$ evaluates to true. (More details are given in the Appendix.)

For the satisfiability problem of LARS formulas, we obtain a similar result.

Theorem 5. Problem SAT for LARS formulas, i.e., given W, B, T, and t, is there a stream $S = (T, \upsilon)$ is over $\mathcal{A}'$ such that $M, S, t \Vdash \alpha$, where $M = \langle S, W, B\rangle$, is PSpace-complete.

Informally, a suitable evaluation function can be guessed and checked in polynomial space, and from NPSpace = PSpace we obtain membership in PSpace. On the other hand, LARS model checking can be easily reduced to satisfiability testing (see Appendix).

4.2.2 LARS Programs

Based on Theorem 4, we show that model checking for ground LARS programs has the same complexity as for LARS formulas in general.

Theorem 6. Problem MC for LARS programs, i.e., given a structure $M = \langle I, W, B\rangle$, a data stream D, a program P, and a time point t, deciding whether $I = (T, \upsilon)$ is an answer stream of P for D at time t, is PSpace-complete.

Informally, this holds because it suffices to check that $M, t \models P$ and that no model M' smaller than M exists that satisfies the reduct $P^{M,t}$ (at time t); building the latter and testing all candidate M' is feasible in polynomial space. The PSpace-hardness is inherited from model checking for LARS formulas (see Appendix).

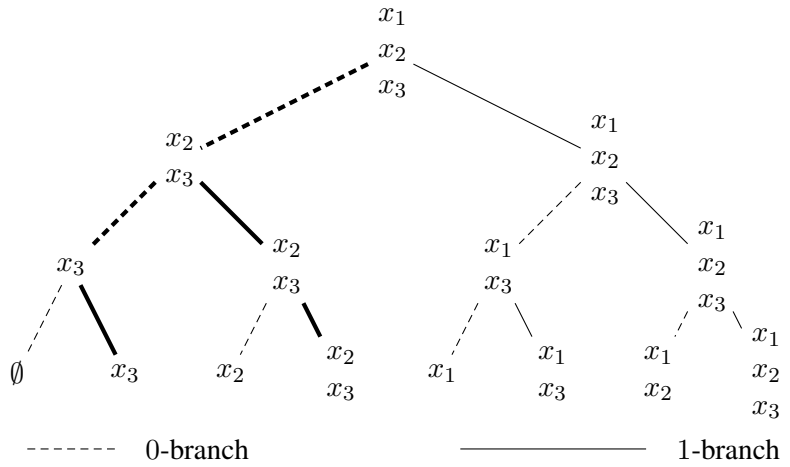


Figure 9: Evaluation tree for the formula $\Diamond \boxplus^{\mathrm{set}:x_1} \Box \boxplus^{\mathrm{set}:x_2} \Diamond \boxplus^{\mathrm{set}:x_3} (\neg x_1 \wedge (x_2 \vee x_3))$, encoding the QBF $\exists x_1 \forall x_2 \exists x_3 (\neg x_1 \wedge (x_2 \vee x_3))$. Nodes $x_1\, x_2\, x_3$, $x_2\, x_3$, etc. represent the streams with timeline $[0, 1]$ and interpretation $\upsilon(0) = \upsilon(1) = \{x_1, x_2, x_3\}$, $\upsilon(0) = \upsilon(1) = \{x_2, x_3\}$, etc., as obtained by window operators $\boxplus^{\mathrm{set}:x_i}$.

For checking satisfiability of LARS programs, we obtain based on the previous theorem also PSpace-completeness.

Theorem 7. Deciding SAT for LARS programs, i.e., given W, B, D and some LARS program P, does P have some answer stream I over $\mathcal{A}'$ for D at t, is PSpace-complete.

As for the membership part, a guess for an answer stream of P w.r.t. data stream D and a time point t has polynomial size and can be verified in polynomial space; the PSpace-hardness is again inherited from model checking for LARS formulas (see Appendix).

4.2.3 Bounded Window Nesting

Revisiting Figure 9, we see that an exponential size evaluation tree results from the evaluation of nested window operators $\boxplus^{\mathrm{set}:x_i}$, where each of them is evaluated at both time points 0 and 1. In this way, exponentially many different substreams are produced in the evaluation. Such an exponential explosion is avoided, if we bound the nesting of window operators in LARS formulas.

Definition 13 (Window nesting depth $wnd(\alpha)$). The *window nesting depth* of a LARS formula α, denoted $wnd(\alpha)$, is the maximal number of window operators encountered on any path from the root to a leaf in the formula tree of α.[2] Formally, $wnd(a) = 0$ for every atom a and inductively $wnd(\neg\alpha) = wnd(\Box\alpha) = wnd(\Diamond\alpha) = wnd(\triangleright\alpha) = wnd(\alpha)$; $wnd(\alpha \wedge \beta) = wnd(\alpha \vee \beta) = wnd(\alpha \rightarrow \beta) = \max(wnd(\alpha), wnd(\beta))$; and $wnd(\boxplus\alpha) = 1 + wnd(\alpha)$.

Note in particular that $wnd(\alpha) = 0$ means no window operators occur in α, and that $wnd(\alpha) = 1$ means that window operators occur but unnested.

[2]For simplicity, we omit a more fine-grained definition of wnd that respects $\triangleright$, for which "encountered" is replaced by "encountered subsequently ... with no $\triangleright$ in between." The tractability result carries over to the larger class of formulas.

If $\#w(\alpha)$ is the number of window operators occurring in a LARS formula α, then at most $(\#w(\alpha) \cdot |T|)^{wnd(\alpha)}$ many substreams of a stream $S = (T, \upsilon)$ (resp. $S^\star = (T^\star, \upsilon^\star)$) are created in a recursive evaluation of $M, S, t \Vdash \alpha$. If $wnd(\alpha)$ is bounded by a constant, then this number is polynomial in the size of S and α. We can thus use a labeling technique to evaluate formulas bottom up (from subformulas) over the possible substreams in polynomial time.

Theorem 8. Problem MC for LARS formulas α is in P, if $wnd(\alpha)$ is bounded by some constant $k \geq 0$, and is P-complete for arbitrary window operators.

The P membership part follows from a more general result in the next subsection (Theorem 10). We also have matching P-hardness (and thus P-completeness) in general due to the fact that evaluating window functions can be P-complete in general.

As a consequence of Theorem 8, also satisfiability of LARS formulas becomes easier to decide when the nesting depth is bounded.

Corollary 2. Problem SAT for LARS formulas α is NP-complete, if $wnd(\alpha)$ is bounded by some constant $k \geq 0$.

The membership is via a simple guess and check argument. Since LARS subsumes propositional logic, the problem is clearly also NP-hard.

Turning to LARS programs, let us define the window nesting depth for a program P naturally as follows.

Definition 14 (Window nesting depth $wnd(P)$). Given a LARS program P, its *window nesting depth* is defined as $wnd(P) = \max\{wnd(\beta(r) \rightarrow \alpha) \mid \alpha \leftarrow \beta(r) \in P\}$.

Our focus is here on finite LARS programs P, for which the nesting depth is always well-defined and finite. For model checking such programs, we obtain the following result.

Theorem 9. Problem MC for LARS programs P is co-NP-complete, if $wnd(P)$ is bounded by some constant $k \geq 0$.

Informally, an answer stream I of program P can be refuted by a guess and check algorithm in polynomial time, thanks to Theorem 8. The co-NP-hardness is inherited from the problem for ordinary answer set programs. From Theorem 9, the following corollary is not difficult to obtain.

Corollary 3. Problem SAT for LARS programs P is Σ_2^p-complete, if $wnd(P)$ is bounded by some constant $k \geq 0$.

The membership in Σ_2^p follows from Theorem 9, as a candidate answer stream for P w.r.t. a data stream D and time point t can be guessed and checked in polynomial time with an NP oracle. The Σ_2^p-hardness is inherited from propositional disjunctive logic programs, for which deciding answer set existence is Σ_2^p-complete (Eiter & Gottlob, 1995).

4.3 Semantic Restriction: Sparse Windows

Bounding the nesting depth of windows serves as a restriction that allows us to obtain tractability of model checking for LARS formulas. In addition to this syntactic criterion, we can obtain other important cases

for which solving this problem is feasible in polynomial time due to semantic properties of the window operators that occur in a LARS formula.

An important such property is that a window operator $\boxplus^w$ and a nested window operator $\boxplus^{w_1}\boxplus^{w_2}$, applied to a stream S from a small (polynomial size) set of streams, always will return a stream from that set. By sharing nodes in the substream evaluation tree, the resulting evaluation graph has polynomial size and the relevant subformula labeling for deciding satisfiability can be produced in polynomial time.

A prototypical example of such "sparse" windows are sliding time-based windows $\boxplus^{i,j}$ (with hop size $d = 1$) that cover the previous i and the next j time points. Applied on a stream $S = (T, \upsilon)$, the resulting window at time point t is the substream $S|_{T'}$, which restricts the timeline to $T' = T \cap [t-i, t+j]$. Notably, the result of evaluating nested sliding time-based windows $\boxplus^{i_1,j_1} \cdots \boxplus^{i_k,j_k}$ also is a substream S' obtained by simply restricting the timeline; overall, there are $O(|T|^2)$ many such S'.

We next describe evaluation graphs and results for sparse windows in more detail, and then discuss concrete classes of window operators that ensure the sparse window property. All time-based, tuple-based and filter windows considered in Section 2.3, 2.4, and 2.6, respectively, are among them, as well as a large class of partition-based windows in Section 2.5; furthermore, windows from these classes can be mixed arbitrarily.

For uniformity, we regard in the sequel the reset operator $\triangleright$ in abuse of the notion as a window operator that yields, in the context of a structure $M = \langle S^\star, W, B\rangle$, the original stream; i.e., $\triangleright$ is viewed as $\boxplus^{w^{S^\star}_{\triangleright}}$ where $w^{S^\star_{\triangleright}}(S, t) = S^\star$ (for all $t \in \mathbb{N}$).

Window graph. For any formula φ, we refer to the sequences of window operators $\boxplus^{w_1} \to \boxplus^{w_2} \to \cdots \to \boxplus^{w_k}$ of φ along the branches of the formula tree of φ, where $k = 1, 2, \ldots$, as the *window-paths* of φ. Consider now a structure $M = \langle S^\star, W, B\rangle$ and a substream $S \subseteq S^\star$. We call a set $\mathcal{S}$ of streams an *evaluation base* of (M, S, φ) and write $\mathcal{S}_B(M, S, \varphi)$ or simply $\mathcal{S}_B$, if it contains S and each stream S_{k+1} that results if we apply the window operators $\boxplus^{w_k}$, starting from $S_0 = S$, at each time point of the current stream S_k recursively along a window-path of φ. An evaluation base includes all streams that can be encountered in the recursive evaluation of $M, S, t \Vdash \varphi$ according to Definition 9, but it in general it includes further streams as well (we shall discuss this aspect later in this section). Given such a base $\mathcal{S}_B$, the *window graph* for (M, S, φ), denoted $WG_{\mathcal{S}_B}(M, S, \varphi)$ or simply $WG_{\mathcal{S}_B}$ is the graph $WG_{\mathcal{S}_B} = (N, E)$ with nodes $N = \mathcal{S}_B$ and edges E that are obtained inductively along window-paths as follows: add from the node S_{k-1} for each time point t in S_{k-1} an edge labeled $(\boxplus^{w_k}, t)$ to $S_k = w_k(S_{k-1}, t)$, where $S_0 = S$. Informally, paths in $WG_{\mathcal{S}_B}$ starting at S allow us to navigate between substreams of S as obtained by window operators as they occur in φ.

Example 20. Consider a structure $M = \langle S^\star, W, B\rangle$, where $S^\star = ([0, 8], \upsilon)$ and $\upsilon = \{5 \mapsto \{a\}, 7 \mapsto \{b\}, 8 \mapsto \{c\}\}$, and the formula $\varphi = \boxplus^{\#2}(\boxplus^3 \Diamond b \wedge \boxplus^{\#1} \Diamond c)$. We take $S = S^\star$. The window-paths of maximal length are $p_1 = \boxplus^{\#2} \to \boxplus^3$ and $p_2 = \boxplus^{\#2} \to \boxplus^{\#1}$, i.e., on the initial stream S we can apply $\boxplus^{\#2}$ (at potentially every time point), and in the resulting streams one can apply $\boxplus^3$, respectively $\boxplus^{\#1}$. We now establish an evaluation base $\mathcal{S}_B$ for (M, S, φ). The initial window operator $\boxplus^{\#2}$ yields potential windows $\#^2(S, 0), \ldots, \#^2(S, 8)$, i.e., by abbreviating the restriction $S|_{T'}$ to timeline $T' = [a, b]$ by S_{ab}, the streams S_{00}, S_{01}, S_{02}, $\ldots$, S_{06}, S_{57}, S_{78}. For path p_1, we may now apply on any of these streams at their respective time points the window function τ^3, similarly for p_2 function $\#^1$. This results in an evaluation base $\mathcal{S}_B$. Note that some of these additional applications return their input stream, e.g., $\tau^3(S_{78}, 8) = S_{78}$, since the timeline $[7, 8]$ has size 1 and is not shrunk further by a time-based window of size 3.

As for the edges of the window graph $WG_{\mathcal{S}_B}$, we add in the first step an edge $S \to S_{00}$ with label $(\boxplus^{\#2}, 0)$, an edge $S \to S_{01}$ with label $(\boxplus^{\#2}, 1)$, $\ldots$, and an edge $S \to S_{78}$ with label $(\boxplus^{\#2}, 8)$. Then,

in the second step for p_1, we add an edge $(S_{00} \to S_{00}, 0)$ with label $(\boxplus^3, 0)$, $S_{01} \to S_{00}$ with $(\boxplus^3, 0)$, $S_{01} \to S_{01}$ with $(\boxplus^3, 1)$, ..., $S_{06} \to S_{52}$ with $(\boxplus^3, 5)$, $S_{06} \to S_{36}$ with $(\boxplus^3, 6)$, ..., $S_{78} \to S_{77}$ with $(\boxplus^3, 7)$, and $S_{78} \to S_{78}$ with $(\boxplus^3, 8)$; and similarly with $\boxplus^{\#1}$ for p_2. ∎

For our purposes, the following lemma is useful.

Lemma 1. Given a structure $M = \langle S^\star, W, B\rangle$, a substream $S \subseteq S^\star$, a formula φ and an evaluation base $\mathcal{S}_B$ for (M, S, φ), the window graph $WG_{\mathcal{S}_B}$ for (M, S, φ) is computable in polynomial time.

The proof is given in the Appendix.

Stream labeling. Given a structure $M = \langle S^\star, W, B\rangle$, a substream $S \subseteq S^\star$ and a window graph $WG_{\mathcal{S}_B} = (N, E)$ for (M, S, φ), we label each pair (S, t) such that $S \in N$ and $t \in T^\star$, where $S = (T, \upsilon)$, with relevant formulas that hold in stream S at time t due to the evaluation base $\mathcal{S}_B$. We define the label set $L_{\mathcal{S}_B}(S, t)$ by the following steps:

1. take a subformula $\boxplus^{w_k}\alpha_k$ in φ such that $\boxplus^{w_1} \to \boxplus^{w_2} \to \cdots \to \boxplus^{w_k}$ is a maximal window-path in the formula tree of φ that has not yet been considered. We label each pair (S, t), such that $S = S_k$, with all subformulas of α_k that evaluate in the stream S at time t to true.

   More precisely, we add a subformula α' of α_k to $L_{\mathcal{S}_B}(S, t)$, where $S = (T, \upsilon)$, by a case distinction on the form of α' due to Definition 9 as follows. We add:

   - atom a, if $a \in \upsilon_k(t)$;
   - $\neg\alpha$, if α is not in $L_{\mathcal{S}_B}(S, t)$;
   - $\alpha \wedge \beta$, if α and β are in $L_{\mathcal{S}_B}(S, t)$; similarly for $\vee$ and $\to$;
   - $\Diamond\alpha/\Box\alpha$, if $\alpha \in L_{\mathcal{S}_B}(S, u)$ for some/all $u \in T$;
   - $@_u\alpha$, if $\alpha \in L_{\mathcal{S}_B}(S, u)$ and $u \in T$.

   This labeling can be carried out bottom up along the formula tree. Note that in the first application of Step 1 α_k does not contain any window operator.

2. We label (S, t), where $S = S_{k-1}$, with $\boxplus^{w_k}\alpha_k$ if (S_k, t) was labeled with α_k in Step 1.

3. Inductively, the window path $\boxplus^{w_1} \to \boxplus^{w_2} \to \cdots \to \boxplus^{w_i}$ is considered in Step 1 for $i < k$, i.e., one considers subformula $\boxplus^{w_i}\alpha_i$, after all window operators that occur in α_i have been considered. Any subformula $\boxplus^{w_j}\alpha$ of α_i starting with a window operator is like an atom, and presence of $\boxplus^{w_j}\alpha$ in $L_{\mathcal{S}_B}(S_i, t)$ reflects the entailment result for this subformula in S_i at t.

Example 21 (cont'd). Consider the window graph $WG_{\mathcal{S}_B}$ of Example 20. We are interested whether $M, S, 8 \Vdash \varphi$ holds ($S = S^\star$) and start illustrating the bottom up evaluation by considering window-path p_2 at $t = 8$, i.e., edges $S \to S_{78}$ and $S_{78} \to S_{88}$ with window graph labels $(\boxplus^{\#2}, 8)$ and $(\boxplus^{\#1}, 8)$, respectively. The (maximal) window-path p_2 ends before subformula $\Diamond c$ which, in Step 1, is evaluated in $S_k = S_{88} = ([8, 8], \{8 \mapsto \{c\}\})$. Thus, we obtain formula labels $L_{\mathcal{S}_B}(S_k, 8) = \{c, \Diamond c\}$. In Step 2, we thus get $L_{\mathcal{S}_B}(S_{k-1}, 8) = \{\boxplus^{\#1}\Diamond c\}$, where $S_{k-1} = S_{78} = ([7, 8], \{7 \mapsto \{b\}, 8 \mapsto \{c\}\})$, i.e., the previous stream in the considered path.

Likewise, we evaluate subformula $\Diamond b$ for path p_1 at $t = 8$, i.e., the edges $S \to S_{78}$ and $S_{78} \to S_{78}$ with window graph labels $(\boxplus^{\#2}, 8)$ and $(\boxplus^3, 8)$, respectively. In Step 1, we add label $\Diamond b$ to $L_{\mathcal{S}_B}(S_{78}, 8)$. Note that b does not hold at time 8 in S_{78} but $b \in L_{\mathcal{S}_B}(S_{78}, 7)$ (and $7 \in [7, 8]$) from similar evaluation, e.g., along path $S \to S_{78} \to S_{77}$ with window graph labels $(\boxplus^{\#2}, 8)$ and $(\boxplus^3, 7)$, respectively. Thus, we add in Step 2 to the formula $\boxplus^3 \Diamond b$ to $L_{\mathcal{S}_B}(S_{78}, 8)$ (note that in this path $S_k = S_{k-1}$).

Step 3 recognizes that all window operators of the conjunction $\varphi' = \boxplus^3 \Diamond b \wedge \boxplus^{\#1} \Diamond c$ have been considered. Hence, we go to Step 1 and find that for the formula $\alpha_k = \varphi'$ both conjuncts $\boxplus^3 \Diamond b$ and $\boxplus^{\#1} \Diamond c$ are in $L_{\mathcal{S}_B}(S_{78}, 8)$, i.e., φ' holds and is added. In the next Step 2, we consider $S_{k-1} = S$, i.e., the original stream before evaluating $\boxplus^{\#2}$. (That is, we navigate back the first edge $S \to S_{78}$ with label $(\boxplus^{\#2}, 8)$ for either path.) Since φ' has been added to $(S_{78}, 8)$ in Step 1, we now assign $L_{\mathcal{S}_B}(S, 8) = \{\boxplus^{\#2} \varphi'\}$. Finally, we recognize in Step 3 that no window operator remains to be considered along paths p_1 and p_2 at $t = 8$. We skip the stream labeling of further pairs (S, t), as we already obtained that $\varphi \in L_{\mathcal{S}_B}(S, 8)$ which means that $M, S, 8 \Vdash \varphi$ holds. ∎

Proposition 2. Let $\mathcal{S}_B$ be an evaluation base for (M, S, φ), where $M = \langle S^\star, W, B \rangle$ and $S \subseteq S^\star$, and let $t \in T^\star$. Then, it holds that $M, S, t \Vdash \varphi$ iff $\varphi \in L_{\mathcal{S}_B}(S, t)$.

Formally, this proposition can be proved by induction on the formula structure. We thus obtain an algorithm to decide $M, S, t \Vdash \varphi$ as follows:

1. given an evaluation base $\mathcal{S}_B$ for (M, S, φ), compute the window graph $WG_{\mathcal{S}_B}$;
2. compute the labeling $L_{\mathcal{S}_B}$ for φ;
3. return "yes" iff $\varphi \in L_{\mathcal{S}_B}(S, t)$.

The correctness of this algorithm follows from Proposition 2. Regarding its time complexity, it is not hard to see that the algorithm runs in time polynomial in the size of $\mathcal{S}_B$, M and φ (see Appendix). In particular, if the evaluation base $\mathcal{S}_B$ is small, we obtain tractability.

Theorem 10. Let $M = \langle S^\star, W, B \rangle$ be a structure, $S \subseteq S^\star$ and let φ be a formula. Suppose that (M, S, φ) has some evaluation base $\mathcal{S}_B$ of size polynomial in the size of M and φ. Then $M, S, t \Vdash \varphi$ is decidable in polynomial time.

The proof exploits that for an evaluation base $\mathcal{S}_B$ of polynomial size, the window graph $WG_{\mathcal{S}_B}$ and the labeling $L_{\mathcal{S}_B}(S, t)$ are constructed by the algorithm above in polynomial time. Notably, a suitable $\mathcal{S}_B$ need not be provided in the input. We can construct one on the fly along with the window graph $WG_{\mathcal{S}_B}$: we initialize $\mathcal{S}_B$ to S and add any stream S_k not yet in $\mathcal{S}_B$ to it, as in the Example 20. This in fact yields a unique evaluation base for (M, S, φ) that is contained in every evaluation base, and is thus guaranteed to have polynomial size.

As already mentioned, the presented stream labeling $L_{\mathcal{S}_B}$ for φ will in general contain more streams than necessary for the evaluation of $M, S, t \Vdash \varphi$. We considered in Example 21 the entailment relation $M, S, 8 \Vdash \varphi$, where S has the timeline $[0, 8]$. All pairs (S', t') with a proper substream S' of S that has a timeline overlapping with $[0, 6]$ are irrelevant, as the first window operator $\boxplus^{\#2}$ already restricts the relevant timeline to $[7, 8]$ and any further consideration affects only substreams of S_{78}. This would be different if a temporal modality $\circ \in \{\Diamond, \Box, @_{t'}\}$ was in front of φ. More generally, if we consider a formula $\circ\psi$ such that ψ does not start with a temporal modality at some time point t, we observe that $\circ$ changes which

time points have to be considered (i.e., all or some t'), while only a window operator in ψ will change the timeline. Moreover, given a sequence $\circ_1 \cdots \circ_n$ of modalities, we observe that the first $n-1$ are irrelevant. Accordingly, we can restrict both the evaluation base $\mathcal{S}_B$ and the window graph $WG_{\mathcal{S}_B}$ by considering the last modality $\circ$ for the current stream S_{k-1} to first determine the relevant time points t' based on which we step to $S_k = (S_{k-1}, t')$. Following this intuition, we in fact skipped the discussion of most pairs (S', t') in Example 21, by directly starting with the paths for $t = 8$ and focusing on the relevant streams in $\mathcal{S}_B$ and the relevant edges of the stream graph $WG_{\mathcal{S}_B}$. It is feasible to obtain these relevant subsets in polynomial time as well. While one can expect a significant speedup in a practical realization of this improvement, the worst case polynomial complexity of the stream labeling procedure does not change. In a more fine-grained view of the formula φ that looks besides window operators also at the occurrence of temporal operators, further tractable fragments of LARS formulas could be identified; we leave this for future work.

On the other hand, from the theoretical perspective, we note that an even more abstract approach is possible: we may alternatively define an evaluation base independently of a formula, i.e., purely based on a structure $M = \langle S^\star, W, B\rangle$: Any (potentially infinite) sequence of window operators $\boxplus^w$ with $w \in W$ will eventually not produce new streams. A given formula φ that only uses window functions from W only represents a subset of these sequences. Sparse windows ensure that, even in this high-level approach, the size of the evaluation base remains polynomial in the size of of the structure.

Sparse window classes. From Theorem 10 we immediately obtain the P-membership of model checking for LARS formulas with bounded window nesting in Theorem 8. Furthermore, we can conclude that model checking for LARS formulas with unbounded nesting is tractable for a broad class of window operators.

Concerning time-based window operators $\boxplus^{\ell,u,d}$, as already observed above, evaluating the window function $\tau^{\ell,u,d}(S,t)$ on the stream S at time point t always yields a substream $S' = (T', \upsilon') = (T', \upsilon|_{T'})$, i.e., S' restricts S to the timeline T'. If we apply a further time-based window on S', we obtain another stream of this form. Overall, there are $O(|T|^2)$ many such S'; if we take possible occurrence of the reset operator $\triangleright$ into account, there are $O(|T^\star|^2)$ many such streams. Thus, Theorem 10 holds for all LARS formulas that use only time-based window operators.

A similar consideration establishes the same result for tuple-based windows $\boxplus^{\#\ell,u}$: evaluating a tuple-based window function $\#^{\ell,u}(S,t)$ on the stream S at time point t yields a substream $S' = (T', \upsilon')$ of $S = (T, \upsilon)$ that diverges from $S = (T', \upsilon|_{T'})$ at most on the stream boundaries t_ℓ and t_u, where $T' = [t_\ell, t_u]$. In any case, S' is uniquely identified by the triple (ℓ, u, t). If we apply a further tuple-based window on S', we again obtain a substream of S of this form; overall, there are $O(|T^\star| \cdot A^2)$ many such streams, where $A = \sum_{t \in T^\star} |\upsilon^\star(t)|$ is the total number of atoms in the stream $S^\star$, thus polynomially many.

Each tuple-based window $\boxplus^{\#\ell,u}$ trivially amounts to a partition-based window $\boxplus^{\mathrm{idx}_{\mathcal{A}},n_{\mathcal{A}}}$ where all atoms are in one partition. The question is thus whether also partition-based windows are sparse. Unfortunately, the answer is negative.

Theorem 11. Problem MC for LARS formulas in which only partition-based windows occur is PSpace-complete.

The PSpace-hardness can be shown by adapting the QBF reduction in the proof for arbitrary LARS formulas in Theorem 4. An analysis of the proof shows that the result even holds if each partition-based window creates only two partitions (which is the minimum in order not to collapse with a tuple-based window); it is recursive nesting and the use of either changing partitions, or of changing tuple counts (or both) which leads to intractability.

The result for tuple-based windows generalizes to partition-based windows, provided that the index functions idx of the window operators $\boxplus^{\mathrm{idx},n}$ that occur in the formula partition the ground atoms $\mathcal{G}$ into

groups that are formed from constantly many base groups B_i. That is, each group $\mathrm{idx}^{-1}(i)$ is of the form $\mathrm{idx}^{-1}(i) = \bigcup \mathcal{B}$, where $\mathcal{B} \subseteq \{B_1, \ldots, B_k\}$ and $\mathcal{G} = \bigcup_{i=1}^{k} B_i$, where k is constant. Let us call such partitions *meager*. In this case, each (nested) result of evaluating a window function can be uniquely identified by a tuple $(\ell_1, u_1, \ldots, \ell_k, u_k, t)$, and there are polynomially many such tuples.

Finally, let us consider filter windows as introduced in Section 2.6. Recall that the function $f^A(S, t)$ associated with $\boxplus^A$ projects the input stream S to the atoms in A. We can extend the description for partition-based windows results above by adding a concrete filter A that is applied prior to the partition-based selection. While in general, semantically an exponential number of filters A are possible, for the concrete evaluation of a LARS formula φ only filters A that syntactically result from the formula matter, and there are only linearly many of them. Thus, the number of relevant substream descriptions $(\ell_1, u_1, \ldots, \ell_k, u_k, t, A)$ still polynomially bounded.

Clearly, the meager-partition representations include all tuple-based representations, which in turn include all time-based representations. Thus, we obtain the following result.

Theorem 12. For LARS formulas α (resp. LARS programs P), problem MC is in P (resp. co-NP-complete), if only time-based, tuple-based, meager partition-based, and filter windows occur in α (resp., in P).

This result can be further generalized by allowing in addition restricted occurrence of arbitrary windows in formulas. In particular, this holds if the nesting depth of such additional window operators in a formula is bounded by a constant. This is because if on a root-path in the formula tree the encountered such windows are $\boxplus^{w_1}, \ldots, \boxplus^{w_\ell}$, then the resulting substream S can be described by a sequence

$$sd_0, \boxplus^{w_1}, sd_1, \ldots, sd_{\ell-1}, \boxplus^{w_\ell}, sd_\ell,$$

where each $sd_i = (\ell_1^{(i)}, u_1^{(i)}, \ldots, \ell_k^{(i)}, u_k^{(i)}, t^{(i)}, A^{(i)})$ is an extended partition-based window description. In total, only polynomially many such descriptions will matter.

Thus in conclusion, for a wide range of formulas that are occur in practice the Model Checking problem for LARS formulas is solvable in polynomial time. Furthermore, in frequent use cases with time-based and tuple-based windows it will have low complexity inside P. Finally, based on the tractability of model checking, for satisfiability the same results as for α^- and P^- in Table 1 using analog arguments.

4.4 Non-ground LARS

For open LARS formulas and non-ground programs, infinite groundings and arithmetic are a source of undecidability. In analogy to database systems, and inspired by notions of safety in logic programming, we make here the following assumption.

The set $\mathcal{C}$ of constants includes, besides those mentioned in a formula α respectively the program P, at most polynomially many further constants, and the set $\mathcal{P}$ of predicates plus $\mathcal{C}$ are part of the input; the set of atoms $\mathcal{A}'$ to consider is $\mathcal{A}' = \mathcal{A}$, i.e., the set of all possible atoms over $\mathcal{P}$ and $\mathcal{C}$ (but not part of the input). We disregard here arithmetic, respectively assume that it is provided over the range of interest (i.e., the timeline of the current stream) in the background data B.

The results for LARS formulas and programs in this setting are shown in Table 2, where an open (schematic) LARS formula $\alpha(\mathbf{x})$ is viewed as a representative of all its instances. Besides the combined complexity, where both the structure and the LARS formula resp. program are part of the input, we also consider the data complexity, where the formula resp. program is fixed.

What we can observe is that in the general case, model checking is not more expensive than in the ground case. This is because a naive instantiation of the formula resp. program to the ground case, which

Problem		formula class		program class	
		α	α^-	P	P^-
combined complexity	Model Checking (MC)	PSpace	co-NP	PSpace	Π_2^p
	Satisfiability (SAT)	NExpTime		$\mathsf{NExpTime}^{\mathsf{NP}}$	
data complexity	Model Checking (MC)	P		co-NP	
	Satisfiability (SAT)	NP		Σ_2^P	

Table 2: Complexity of reasoning in non-ground (Datalog) LARS (completeness results)

would cause an exponential blowup, can be avoided. On the other hand, a model resp. answer stream S witnessing satisfiability in the general case, may have exponential size in the input. This blowup dominates the PSpace-complexity of model checking. Under data complexity, instantiation does not cause a blowup and we obtain for both LARS formulas and programs the same results as in the ground case for bounded window nesting, as in Table 1; the NP-, co-NP- and Σ_2^p-hardness parts are inherited from the complexity of deciding answer set existence respectively answer set checking for (disjunctive) Datalog programs that are subsumed by LARS formulas respectively programs.

Bounding the window nesting depth clearly does not affect the data complexity, nor the combined complexity of satisfiability where the exponential stream size is dominating. For model checking, the co-NP entry for LARS formulas α^- is explained by the tractability of the problem in the ground case: a model candidate can be refuted by guessing and checking an instance of the formula that violates the candidate. Finally, the Π_2^p-entry for LARS programs P^- is explained by an additional minimality check; the NP- resp. Π_2^p-hardness is inherited from conjunctive query evaluation in databases resp. from answer set checking for disjunctive Datalog programs, cf. (Eiter, Faber, Fink, & Woltran, 2007).

In order to avoid an inflation of formal statements, we confine here to a summarizing result.

Theorem 13. The complexity of Model Checking (MC) and Satisfiability (SAT) of an open LARS formula α, respectively non-ground LARS program P, in the general case and for bounded window nesting, is as listed in Table 2.

More proof details are given in the appendix. From the results in Table 2, we easily obtain the following results for answering non-ground LARS formula queries.

Corollary 4. Given a structure $M = \langle S, W, B\rangle$ and a query $Q = \langle M, u, \alpha\rangle$, deciding whether Q has some answer over M is (i) PSpace-complete for arbitrary α in combined complexity, (ii) NP-complete for α with bounded window nesting under combined complexity, and (iii) decidable in polynomial time (and P-complete in general) under data complexity.

These results derive from the fact that Q has no answer, if for every grounding $\sigma \in \sigma(\mathcal{C}, T)$, we have that $M, t, \sigma(u) \nVdash \sigma(\alpha)$, which is equivalent to $M, t, \sigma(u) \Vdash \sigma(\neg\alpha)$. From Theorem 13, we can thus infer the membership parts, where in (iii) α (as the formula is fixed) amounts to the α^- case and the number groundings σ is polynomial. The PSpace-hardness for (i) is inherited from the ground case, and the NP-hardness for (ii) from the classic NP-completeness of conjunctive query answering (Chandra & Merlin, 1977); finally, the P-hardness for (iii) is due the unrestricted windows.

Note that based on Theorem 12 and the discussion after it, the NP-completeness of case (ii) generalizes to queries with bounded window nesting if all time-based, tuple-based, meager partition-based and filter-windows occurring in them are disregarded. Thus for practical settings, answering LARS formula queries is NP-complete, and thus not harder than answering conjunctive queries.

5 Relation to other Languages and Formalisms

In this section we discuss the relationship of LARS with other formal languages for reasoning on data streams, starting with the prominent linear temporal logic (LTL). Then, we investigate the continuous query language (CQL), followed by a note on extensions of the SPARQL query language for streaming, i.e., C-SPARQL and CQELS. Finally, we consider ETALIS as an example of a complex event processing language that focuses on expressing of temporal intervals by rules. Further related work is discussed separately in Section 6.2.

5.1 Temporal Logic

In this section, we compare LARS to temporal logic, where we naturally focus on linear time logic (LTL) (Pnueli, 1977) extended with past time operators (PLTL) (Markey, 2003). Syntactically, these logics extend propositional logic with temporal operators, according to the following syntax

$$\phi ::= \bot \mid a \mid \neg\alpha \mid \alpha \wedge \beta \mid \alpha \vee \beta \mid \alpha \rightarrow \beta \mid \mathsf{X}\,\alpha \mid \alpha\,\mathsf{U}\,\beta \mid \mathsf{X}^{-1}\,\alpha \mid \alpha\,\mathsf{U}^{-1}\,\beta;$$

where $a \in \mathcal{G}$. The informal meaning of $\mathsf{X}\,\alpha$ is that α is true at the next point in time, and $\alpha\,\mathsf{U}\,\beta$ means that α is from now on true until β is true at some point; $\mathsf{X}^{-1}\,\alpha$ and $\alpha\,\mathsf{U}^{-1}\,\beta$ are the counterparts for the past (not available in LTL),[3] i.e., that α is true at the previous point of time respectively that α has always been true after some time point at which β was true. Important derived operators are $\mathsf{F}\,\alpha$ and $\mathsf{G}\,\alpha$ which are shorthand for $\top\,\mathsf{U}\,\alpha$, where $\top = \neg\bot$, and $\neg(\top\,\mathsf{U}\,\neg\alpha)$ and state that α is true now or at some respectively now and at every time point in the future; $\mathsf{F}^{-1}\,\alpha = \top\,\mathsf{U}^{-1}\,\alpha$ and $\mathsf{G}^{-1}\,\alpha = \neg(\top\,\mathsf{U}^{-1}\,\neg\alpha)$ express the counterparts for the past.

Semantically, PLTL-formulas are evaluated over paths, which are infinite sequences $\pi = \pi(0), \pi(1), \pi(2), \ldots$ of positions with an associated interpretation $\nu(\pi(i))$ of propositional atoms $\mathcal{A}$, for each $i \geq 0$; the latter is often tacitly omitted. The satisfaction relation $\pi, i \models \alpha$ is inductively defined as follows:

$$\begin{array}{lll}
\pi, i \models a & \text{iff} & a \in \nu(\pi(i)), \quad \text{for } a \in \mathcal{A}, \\
\pi, i \models \neg\alpha & \text{iff} & \pi, i \not\models \alpha, \\
\pi, i \models \alpha \wedge \beta & \text{iff} & \pi, i \models \alpha \text{ and } \pi, i \models \beta, \\
\pi, i \models \alpha \vee \beta & \text{iff} & \pi, i \models \alpha \text{ or } \pi, i \models \beta, \\
\pi, i \models \alpha \rightarrow \beta & \text{iff} & \pi, i \not\models \alpha \text{ or } \pi, i \models \beta, \\
\pi, i \models \mathsf{X}\,\alpha & \text{iff} & \pi, i+1 \models \alpha, \\
\pi, i \models \alpha\,\mathsf{U}\,\beta & \text{iff} & \pi, j \models \beta \text{ for some } j \geq i \text{ such that } \pi, k \models \alpha \text{ for all } i \leq k < j, \\
\pi, i \models \mathsf{X}^{-1}\,\alpha & \text{iff} & i > 0 \text{ and } \pi, i-1 \models \alpha, \\
\pi, i \models \alpha\,\mathsf{U}^{-1}\,\beta & \text{iff} & \pi, j \models \beta \text{ for some } j \leq i \text{ such that } \pi, k \models \alpha \text{ for all } j < k \leq j.
\end{array}$$

Note in particular that $\neg\,\mathsf{X}^{-1}\,\top$ allows us to recognize that we are at the beginning of the path, as $\pi, i \models \neg\,\mathsf{X}^{-1}\,\top$ holds iff $i = 0$. Two PLTL formulas α and β are *equivalent*, if for every path π and integer $i \geq 0$,

[3] To stress symmetry, we write U^{-1} instead of the usual S.

it holds that $\pi, i \models \alpha$ iff $\pi, i \models \beta$, and *initially equivalent*, if for every path π it holds that $\pi, 0 \models \alpha$ iff $\pi, 0 \models \beta$.

It is well-known that PLTL is not more expressive than LTL in the sense that every PLTL formula is initially equivalent to some LTL-formula (Gabbay, 1987), but the smallest such formula can be exponentially larger (Markey, 2003).

Comparison to LARS. Comparing LARS to linear temporal logic, we see that the temporal operators are clearly different, The temporal operators $\Box$ and $\Diamond$ in LARS have as counterparts the pairs $\mathsf{G}, \mathsf{G}^{-1}$ and $\mathsf{F}, \mathsf{F}^{-1}$ respectively, which allow one to address all positions in a path; the past time operators are indispensable for evaluation inside the path. The window operators in LARS have no counterpart in LTL and PLTL, and similarly the $@_{t'}$ operator which is known as nominal in hybrid logic and can be traced back to Prior's work (Prior, 1967). On the other hand, LARS has no next-time X nor until U or any of their past time counterparts.

The presence of temporal operators in linear time logic formulas affects the computational complexity of model checking and satisfiability testing in general (cf. (Demri & Schnoebelen, 2002)). In the general case, for both LTL and PLTL these problems are PSpace-complete, cf. (Sistla & Clarke, 1985), where satisfiability of a formula α means existence of a path π such that $\pi, 0 \models \alpha$, and model checking that $\pi, 0 \models \alpha$ for every path π in a given Kripke structure. Thus, at the surface LARS and LTL have the same computational complexity.

However, in LARS we consider only *single path Kripke structures* of a given length, for both satisfiability and model checking at some given time point t. For both LTL and PLTL, model checking single paths is feasible in polynomial time,[4] and satisfiability is thus easily seen to be NP-complete in this setting. Thus, there is a considerable complexity gap between LARS and linear time logic.

Nonetheless, it is possible to express (propositional) LARS in linear temporal logic, if we confine to particular window operators. We show in the next section that this is possible for sliding time-based windows; that is, we can view this instance of LARS as a fragment of linear time logic. As LTL expresses all and only the star-free regular languages, i.e., definable by regular expressions without Kleene star, we obtain that such LARS formulas can *a fortiori* express only a strict fragment of the regular languages, where a language consists of all input streams (representing a finite string) on which the formula evaluates to true. For example, the language defined by $(aa)^*$ (strings of a of even length) cannot be expressed. LARS formulas with arbitrary (polynomial-time evaluable) window operators trivially express by Corollary 4 all and only the polynomial time recognizable languages.[5]

5.1.1 Translation of LARS to Linear Temporal Logic

Formally, we represent any LARS structure $M = \langle S, W, \emptyset \rangle$, where $S = (T, \upsilon)$, as a PLTL interpretation $\pi(M) = \pi(0), \pi(1), \ldots$ where for each integer $i \geq 0$, $\nu(\pi(i)) = \{\mathbf{u}\}$ if $i \notin T$, and $\nu(\pi(i)) = \upsilon(i)$ otherwise, where $\mathbf{u}$ is a special atom which expresses that the position i is not in the stream.

Our translation of LARS formulas to PLTL-formulas for evaluation at a time point t in a stream $S = (T, \upsilon)$, where $T = [\ell', u']$, is shown in Algorithm 1. The parameters ℓ, u mark the interval $[\ell, u]$ of the substream that is currently considered, while ℓ', u' marks the original interval. The translation proceeds recursively, where the temporal modalities $\Box$, $\Diamond$ and $@_{t'}$ are effected using the X operator, where we use here $\mathsf{X}^k\,\alpha$ as a shorthand for the k-fold iteration of X on α; that is, $\mathsf{X}^0\,\alpha = \alpha$, for $k > 0$ we have $\mathsf{X}^k\,\alpha = \mathsf{X}\,\mathsf{X}^{k-1}\,\alpha$,

[4] It is known that the problem is NLogSpace-hard but unknown whether it is in NLogSpace nor P-hard in this setting, for both LTL and PLTL (Laroussinie, Markey, & Schnoebelen, 2002).

[5] Note that this does not refute Gurevich's conjecture that there is no query language that captures polynomial time (Gurevich, 1988), as the input stream comes with an ordering.

Algorithm 1 LARS to PLTL translation

Input: integers ℓ', ℓ, u, u' such that $0 \leq \ell' \leq \ell \leq u \leq u',\ t \in [\ell', u']$, ground LARS formula φ
Output: PLTL formula

function $\text{PLTL}(\ell', \ell, u, u', t, \varphi)$
match φ

case atom a	$\Longrightarrow$	a
case $\neg\alpha$	$\Longrightarrow$	$\neg\text{PLTL}(\ell', \ell, u, u', t, \alpha)$
case $\alpha \wedge \beta$	$\Longrightarrow$	$\text{PLTL}(\ell', \ell, u, u', t, \alpha) \wedge \text{PLTL}(\ell', \ell, u, u', t, \beta)$
case $\alpha \vee \beta$	$\Longrightarrow$	$\text{PLTL}(\ell', \ell, u, u', t, \alpha) \vee \text{PLTL}(\ell', \ell, u, u', t, \beta)$
case $\alpha \rightarrow \beta$	$\Longrightarrow$	$\text{PLTL}(\ell', \ell, u, u', t, \alpha) \rightarrow \text{PLTL}(\ell', \ell, u, u', t, \beta)$
case $\Box\alpha$	$\Longrightarrow$	$\bigwedge_{k=\ell-t}^{u-t} \mathsf{X}^k\, \text{PLTL}(\ell', \ell, u, u', t+k, \alpha)$
case $\Diamond\alpha$	$\Longrightarrow$	$\bigvee_{k=\ell-t}^{u-t} \mathsf{X}^k\, \text{PLTL}(\ell', \ell, u, u', t+k, \alpha)$
case $@_{t'}\,\alpha$	$\Longrightarrow$	**if** $\ell \leq t' \leq u$ **then** $\mathsf{X}^{t'-t}\,\text{PLTL}(\ell', \ell, u, u', t', \alpha)$ **else** $\bot$
case $\boxplus^{i,j}\alpha$	$\Longrightarrow$	$\text{PLTL}(\ell', \max(\ell, t-i), \min(t+j, u), u', t, \alpha)$
case $\triangleright\,\alpha$	$\Longrightarrow$	$\text{PLTL}(\ell', \ell', u', u', t, \alpha)$

end function

and for $k < 0$ we have $\mathsf{X}^k\,\alpha = \mathsf{X}^{-1}\,\mathsf{X}^{k+1}\,\alpha$. For window operators $\boxplus^{i,j}$ the current interval has to be adjusted to at most i steps before resp. j steps after t, while for the reset operator $\triangleright$ the original interval is selected.

Example 22. Consider the formula $\varphi = @_1 q \vee p \wedge \boxplus^{1,3}\Diamond r$, and let $[\ell', u'] = [\ell, u] = [2, 4]$ and $t = 3$. Then we have

$$\text{PLTL}(2, 2, 4, 4, 3, \varphi) = \bot \vee p \wedge \mathsf{X}^{-1}\, r \wedge r \wedge \mathsf{X}\, r.$$

The $\bot$ disjunct is due to the fact that position 1 is not in the timeline $T = [2, 4]$. The conjunction $\mathsf{X}^{-1}\, r \wedge r \wedge \mathsf{X}\, r$ stems from the translation of the window $\boxplus^{1,3}$. Note that $\mathsf{X}^2\, r$ and $\mathsf{X}^3\, r$ are missing since $t = 3$ is in distance 1 to the end of the bound $u = 4$. ∎

We then can show that the transformation in Algorithm 1 works properly.

Proposition 3. Let $M = \langle S, W, \emptyset\rangle$, where $S = (T, \upsilon)$ and $T = [t_\ell, t_u]$, and let $t \in T$ and φ be a LARS formula. Then

$$M, S, t \Vdash \varphi \quad \text{iff} \quad \pi(M), t \models \text{PLTL}(t_\ell, t_\ell, t_u, t_u, t, \varphi) \quad \text{iff} \quad \pi(M), 0 \models \mathsf{X}^t\, \text{PLTL}(t_\ell, t_\ell, t_u, t_u, t, \varphi).$$

By this proposition, we can reduce model checking of a LARS formula φ on an input stream $S = (T, \upsilon)$, to model checking an ad-hoc PLTL formula constructed from φ, M and the specific time point $t \in T$, on a single path $\pi(M)$. In particular, we can do this for $T = [0, t]$, i.e., at the end of the input stream. Furthermore, we can transform the PLTL formula easily to an LTL formula that is initially equivalent.

Unfortunately the transformation PLTL is not practical, as it is exponential in the formula size. However, it can be exploited by a nondeterministic algorithm for model checking $M, S, t \Vdash \varphi$ in $\mathsf{ALogSpace}$ (i.e., alternating logspace): in the recursive evaluation, conjunction and disjunction evaluated by universal and existential computation states, respectively, and no further storage than the interval bounds and few (constantly many) auxiliary variables is needed. Since $\mathsf{ALogSpace} = \mathsf{P}$, this yields an alternative proof that for LARS formulas with sliding time-based window operators, model checking is feasible in polynomial time.

Regarding the expressivity of LARS, we are interested in a transformation $\text{PLTL}(t', \varphi)$ (resp., $\text{PLTL}(\varphi)$) that depends only on t' and φ (resp., φ), but not the input stream S. It turns out that such a transformation exists, but is much more involved.

Theorem 14. For every LARS formula φ with sliding time-based windows over atoms $\mathcal{A}$, LTL formulas $\text{PLTL}(t', \varphi)$ and $\text{PLTL}(\varphi)$ over $\mathcal{A} \cup \{\mathbf{u}\}$ are constructible such that for every structure $M = \langle S, W, \emptyset \rangle$, where $S = (T, \upsilon)$ and $T = [0, t]$, it holds that

(i) $M, S, t' \Vdash \varphi$ iff $\pi(M), t' \models \text{PLTL}(t', \varphi)$ iff $\pi(M), 0 \models \mathsf{X}^{t'} \text{PLTL}(t', \varphi)$, respectively;

(ii) $M, S, t \Vdash \varphi$ iff $\pi(M), 0 \models \text{PLTL}(\varphi)$.

Thus in other words, LARS with sliding time-based windows is a fragment of PLTL (and in fact a strict fragment, as the until-operator can not be expressed). Furthermore, as PLTL and LTL have the same expressiveness and can express only regular languages, the considered LARS fragment is thus a strict fragment of the regular languages.

The proof of Theorem 14 is given in the Appendix. Informally, this result can be established as follows. If we let $dist(n) = \mathsf{X}^n \neg\mathbf{u} \wedge \mathsf{X}^{n+1} \mathbf{u}$ express that the distance to the end of the input stream S is $n \geq 0$, then the formula

$$\bigvee_{n=0}^{\infty} (dist(n) \wedge \text{PLTL}(0, 0, t'+n, t'+n, t', \varphi)) \tag{11}$$

would express model checking of φ at t'; however, this is not an admissible PLTL-formula.

If each occurrence of a temporal operator $\Box$, $\Diamond$, and $@_{t'}$ in φ is *windowed*, i.e., within the scope of some window $\boxplus^{i,j}$ not followed by any $\triangleright$, the transformation $\text{PLTL}(0, 0, t'+n, t'+n, t', \varphi)$ yields for fixed φ and t' only finitely many different formulas for all $n \geq 0$; this is because only a bounded number of distances $u - \ell$ occur, and for some large enough n_0, the result of the transformation is identical for all $n \geq n_0$. That is, we can obtain $\text{PLTL}(t', \varphi)$ as

$$\bigvee_{n=0}^{n_0-1} (dist(n) \wedge \text{PLTL}(0, 0, t'+n, t'+n, t', \varphi)) \vee (dist_{\geq}(n_0) \wedge \text{PLTL}(0, 0, t'+n_0, t'+n_0, t', \varphi)), \tag{12}$$

where $dist_{\geq}(n) = \mathsf{X}^n \neg\mathbf{u}$ expresses that the distance to the end of the input stream is at least n. Non-windowed occurrences of $\Box$, $\Diamond$, or $@_t$ in φ can be naively expressed using infinite formulas similarly as in (11), which again can be replaced by finite formulas (as only finitely many different constituents are necessary). Finally, the formula $\text{PLTL}(\varphi)$ can be obtained from $\text{PLTL}(t', \varphi)$ likewise using a formula $end(t') = \mathsf{X}^{t'} \neg\mathbf{u} \wedge \mathsf{X}^{t'+1} \mathbf{u}$ to recognize the end of the input stream.

5.1.2 LARS Programs

If we allow intensional atoms and consider LARS programs, then the expressivity increases and all regular languages can be expressed. Formally, let $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ denote the class of propositional LARS programs in which all windows are sliding time-based windows $\boxplus^{i,j}$. Furthermore, denote for each program $P \in \mathcal{P}_{\mathbb{N},\mathbb{N}}$ with extensional atoms $\mathcal{G}^{\mathcal{E}}(P)$ by $\mathcal{L}^{\mathcal{E}}(P)$ the set of all input (data) streams $S = (T, \upsilon)$ over $\mathcal{G}^{\mathcal{E}}$, where $T = [0, t]$, such that P has some answer stream for S at t. Naturally, any stream S over $\mathcal{G}^{\mathcal{E}}$ encodes a string $\upsilon(0)\upsilon(1)\cdots\upsilon(t)$ over the alphabet $2^{\mathcal{G}^{\mathcal{E}}}$. We then establish:

Theorem 15. $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ captures the class of regular languages modulo the empty string ϵ, that is

(i) for each $P \in \mathcal{P}_{\mathbb{N},\mathbb{N}}$ and extensional atoms $\mathcal{G}^{\mathcal{E}}(P)$, the set $\mathcal{L}^{\mathcal{E}}(P)$, viewed as set of strings over $\Sigma = 2^{\mathcal{G}^{\mathcal{E}}(P)}$, is a regular language;[6] and

(ii) for each regular language $\mathcal{L}$ over alphabet Σ such that $\epsilon \notin \mathcal{L}$, there exists some program $P \in \mathcal{P}_{\mathbb{N},\mathbb{N}}$ with extensional atoms $\mathcal{G}^{\mathcal{E}}(P)$ such that $\mathcal{L} = \mathcal{L}^{\mathcal{E}}(P)$.[7] Furthermore, this holds even if only the window $\boxplus^{1,0}$ is used and the @-operator and $\triangleright$ do not occur in $P_{\mathcal{L}}$.

Informally, (i) holds as we can express answer stream existence for a program in $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ over finite input streams S by a formula in monadic second-order logic (MSO); as the MSO definable languages coincide with the regular languages by the famous Büchi-Elgot-Trakhtenbrot Theorem (Büchi, 1960b, 1960a; Elgot, 1961; Trakhtenbrot, 1961), propositional LARS programs can express only regular languages. Conversely, (ii) holds as we can encode finite automata A in propositional LARS programs P_A such that answer stream existence for an input stream S amounts to acceptance of S by A. Informally, the answer streams encode accepting runs; for that, it is crucial that intensional (auxiliary) atoms are available, to store the states of the automaton during a run. Particularly worth noting is that P_A can not use an ordering on the input elements, as binary predicates are not available.

The availability of intensional atoms for capturing the regular languages is essential, as without such atoms the expressiveness of the LARS programs $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ coincides with the one of respective LARS formulas (i.e., with sliding time-based windows $\boxplus^{i,j}$). By the result of Theorem 14, such formulas are subsumed by the fragment of PLTL in which only the operators X, G and their past time versions X^{-1}, G^{-1} are available. That fragment amounts to the 2-variable fragment of FO-logic on strings (Etessami, Vardi, & Wilke, 2002) and is less expressive than full FO-logic on strings, which in turn is as expressive as full LTL and PLTL and captures the star-free regular languages.

If we abandon extensional atoms and consider *definability* in terms of models, i.e., answer streams of a LARS program represent strings over an alphabet $\Sigma = 2^{\mathcal{A}}$ (similarly as in part (i) of Theorem 15) where $\mathcal{A} = \mathcal{G}^{\mathcal{I}}(P) = \mathcal{G}(P)$, then $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ and the LARS $\boxplus^{i,j}$-formulas have incomparable expressiveness. However, if as in part (ii) of Theorem 15, the letters σ of Σ are singleton interpretations $\{\sigma\} \subseteq \mathcal{A}$, then $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ is strictly more expressive than the class of LARS $\boxplus^{i,j}$-formulas. In both settings, LARS programs can define languages that are not expressible in LTL. More details are given in the Appendix.

We remark that the translation $\text{PLTL}(\varphi)$ from above can be exploited to capture answer streams for restricted classes of LARS programs, in the sense that the existence of an answer stream for an input (data) stream S at its end t amounts to the existence of an equilibrium model of a formula $\text{PLTL}(\varphi, S, t)$, which extends $\text{PLTL}(\varphi)$ with the stream data. Equilibrium models of an LTL formula φ are particular models that satisfy a stability condition (Cabalar & Vega, 2007; Aguado, Cabalar, Diéguez, Pérez, & Vidal, 2013); they lift the respective notion for classical formulas, which captures the answer sets of ordinary logic programs (Pearce, 2006), to LTL. The restricted class of LARS programs can be singled out via the class LARS_{HT} of programs in (Beck et al., 2016), whose answer streams correspond as shown there to its equilibrium models, where the notion is naturally lifted from single interpretations to streams; we leave this for future work.

We close this section with noting that the expressiveness of LARS programs with sliding time-based windows increases if we move beyond propositional programs; by the results in Section 4 and well-known results for disjunctive Datalog (Eiter et al., 1997), it can be seen that they capture the class of languages with complexity in Σ_2^p; however, we omit here further consideration.

[6] This can be restricted to strings over $\mathcal{G}^{\mathcal{E}}(P)$, by neglecting others resp. requesting that P only accepts respective streams.

[7] Here $\sigma \in \Sigma$ is identified with the singleton $\{\sigma\}$. As streams are non-void, ϵ can not be accommodated in this string representation. If $S = \emptyset$ would be allowed, ϵ would be recognized e.g. by $\Box\bot$ and any regular $\mathcal{L}$ is expressible.

5.2 Continuous Query Language (CQL)

A particularly influential work in stream processing has been the Stanford Stream Data Manager (STREAM) (Arasu et al., 2003a) and its Continuous Query Language (CQL) (Arasu et al., 2003b, 2006). The central idea is to reuse existing features from SQL and extend it with streams as additional data sources. To this end, different window operators are used to obtain recent snapshots of data, which are then essentially viewed as database relations.

Example 23. Following up on request (i) from Example 1, we state a CQL query for expected arrival times of trams where no traffic jam has been reported at their last station within the last 20 minutes. Recall that the relation $plan(L, X, Y, D)$ records for line L the scheduled travel time D between station X and Y.

```
SELECT TRAM.ID, PLAN.Y, TY
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 TY=TRAM.T+PLAN.D AND
      NOT EXISTS
        (SELECT * FROM JAM[RANGE 20]
         WHERE JAM.ST=TRAM.ST)
```

Note that streams `TRAM` and `JAM` have designated timestamp fields "`T`", i.e., explicit attributes that state the time when the tuple occurred in the stream. ∎

In CQL, a *stream* is viewed as bag of *elements* of form $\langle \mathbf{c}, t \rangle$, where $\mathbf{c}$ is a tuple (which we can view as vector of constants) and t a timestamp; a *relation* maps timestamps to bags of tuples. To translate between these concepts, the operational semantics of CQL builds on three operators:

- *Stream-to-relation* (S2R) operators apply window functions to the input stream to create a relation for recent tuples, i.e., those in the selected window.
- *Relation-to-relation* (R2R) operators can manipulate relations similarly as in relational algebra, respectively SQL.
- *Relation-to-stream* (R2S) translates back a relation into a stream for the output of continuous queries.

Our focus here is on the first two operators, the R2S operator only concerns how output is generated but does not influence the query semantics as such. The S2R operator allows us to consider streaming tuples as sets of atoms. The semantics of CQL thus essentially reduces to the R2R operator, once recent snapshots of streaming data have been selected by S2R. Due to this, we show that LARS programs capture CQL by exploiting two well-known translations: from SQL to relational algebra (Dadashzadeh & Stemple, 1990) and from relational algebra to Datalog (Garcia-Molina, Ullman, & Widom, 2009). Let us call the former translation $RelAlg$ and the latter Dat.

The idea is to have a 3-step process to obtain a Datalog program for a CQL query q:

(1) replace in `FROM` clauses the input sources (i.e., streams with window expressions) by virtual table names due to the renaming function rel as defined in Table 3. By replacing in CQL query q each occurrence of an input stream s by a relation $rel(s)$, we obtain a SQL query $rel(q)$.

(2) Apply $RelAlg$ on this query to obtain a relational algebra expression.

(3) Apply Dat on the expression to obtain a Datalog program with a designated predicate $\hat{q}$ that reflects the resulting tuples.

Input source s in `FROM` clause	Relation $rel(s)$	LARS window fn. $w(s)$
`S[RANGE L]`	$s\_range\_\boldsymbol{L}$	τ^{L}
`S[RANGE L SLIDE D]`	$s\_range\_\boldsymbol{L}\_slide\_\boldsymbol{D}$	$\tau^{L,0,D}$
`S[RANGE UNBOUNDED]`	s_range_unb	τ^{∞}
`S[NOW]`	s_range_0	τ^{0}
`S[ROWS N]`	$s\_rows\_\boldsymbol{N}$	$\#^{N}$
`S[PARTITION BY X1,...,Xk ROWS N]`	$s\_part\_\boldsymbol{X1}\_\ldots\_\boldsymbol{Xk}\_rows\_\boldsymbol{N}$	$\mathrm{p}^{\mathrm{idx},n}$

Table 3: Translation function rel and LARS window $w(s)$

More formally, we get for a CQL query q a Datalog program $\Delta_D(q) = Dat(RelAlg(rel(q)))$. Any static relation (table) `B` can be naturally encoded as

$$\Delta(\mathtt{B}) := \{b(\mathbf{c}) \mid \mathbf{c} \text{ is a tuple in } \mathtt{B}\}, \tag{13}$$

where the lower case b version of relation name `B` serves as predicate name for atoms; tuples $\mathbf{c}$ can be seen as vectors of constants.

We observe that LARS allows us to model the S2R operator. A snapshot of a stream `S` amounts to (i) applying an according window operator and then (ii) abstracting away time. The second step amounts to existential quantification over time, i.e., formulas of form $\boxplus^{w}\Diamond\varphi$. Table 3 lists the LARS window functions corresponding to those in CQL. We thus can derive each snapshot relation $rel(s)$ for a CQL input source s (as listed in the table) using a *snapshot rule* of form

$$\Delta_L(s) := rel(s)(\mathbf{V}) \leftarrow \boxplus^{w(s)}\Diamond s(\mathbf{V}), \tag{14}$$

where the lower case s version of stream name `S` serves as predicate name, and $\mathbf{V}$ is the list of variables corresponding to the attributes of tuples in `S`. We refer to static relations and input streams (with window expressions) uniformly as *input sources*. We thus obtain a LARS program

$$\Delta_L(q) = \Delta_D(q) \cup \{\Delta_L(s) \mid s \text{ is an input source in } q\,\}.$$

For a set Q of queries, we simply take respective unions, i.e., $\Delta_x(Q) = \bigcup_{q\in Q} \Delta_x(q)$, $x \in \{L, D\}$.[8]

Example 24. We give a Datalog translation $\Delta_D(q)$ of the CQL query q in Example 23. (Note that due to the exact translation from SQL and potential optimizations the intermediate relational algebra representation might vary and thus the specific set of derived Datalog rules. The employed translation is detailed in the Appendix.) Let $\mathbf{T} = ID_1, ST_1, T_1$; $\mathbf{L} = ID_2, L_2$; $\mathbf{P} = L_3, X_3, Y_3, D_3$; $\mathbf{J} = ST_4, T_4$ (subscripts for variables serve to reflect their origin in the same schema in a readable way).

$$\begin{aligned}
q_0(\mathbf{T},\mathbf{L},\mathbf{P}) &\leftarrow tram_part_\boldsymbol{ID}_rows_\boldsymbol{1}(\mathbf{T}), line(\mathbf{L}), plan(\mathbf{P}).\\
q_1(\mathbf{T},\mathbf{L},\mathbf{P}) &\leftarrow q_0(\mathbf{T},\mathbf{L},\mathbf{P}), ST_1 = X_3, ID_1 = ID_2, L_2 = L_3.\\
q_2(\mathbf{T},\mathbf{J}) &\leftarrow tram_part_\boldsymbol{ID}_rows_\boldsymbol{1}(\mathbf{T}), jam_range_\boldsymbol{20}(\mathbf{J}).\\
q_{12}(\mathbf{T},\mathbf{L},\mathbf{P},\mathbf{J}) &\leftarrow q_1(\mathbf{T},\mathbf{L},\mathbf{P}), q_2(\mathbf{T},\mathbf{J}).\\
q'_{12}(\mathbf{T},\mathbf{L},\mathbf{P}) &\leftarrow q_{12}(\mathbf{T},\mathbf{L},\mathbf{P},\mathbf{J}).\\
q(ID_1, Y_3, TY) &\leftarrow q_1(\mathbf{T},\mathbf{L},\mathbf{P}), \neg q'_{12}(\mathbf{T},\mathbf{L},\mathbf{P}), TY = T_1 + D_3.
\end{aligned}$$

[8]Note that LARS formulas of form $\boxplus^{w}\Diamond\varphi$ could by themselves be viewed as relation names and interpreted as Datalog programs.

Informally, q_0 captures the cross product of relations `LINE` and `PLAN` as given in the `FROM`-clause, and the relation corresponding to the window on stream `TRAM`. The selection based on the statement `TRAM.ID=LINE.ID AND LINE.L=PLAN.L` in the `WHERE`-clause is captured in predicate q_1. The cross product of recent tram appearances at stations and traffic jams is then reflected in q_2 and the join with q_1 yields q_{12}, which thus captures tram appearances that shall not be considered. In order to remove these, jam information is projected away to obtain predicate q'_{12}. Finally, those variable groundings for q_1 are reported that are not groundings for q'_{12}, and in addition the calculated arrival time TY which adds the planned travel time D_3 to occurrence time T_1 of the last station. (Note that we explicitly model arrival times in tuples. Thus, they remain accessible after S2R, resp. in the Datalog and LARS encodings.)

Using snapshot rules of form (14), we obtain a LARS program $\Delta_L(q)$ by adding the following rules (idx, n are from Example 16):

$$\begin{aligned} \mathit{tram\_part\_}\boldsymbol{ID}\mathit{\_rows\_}\boldsymbol{1}(\mathbf{T}) &\leftarrow \boxplus^{\mathrm{idx},n}\Diamond \mathit{tram}(\mathbf{T}). \\ \mathit{jam\_range\_}\boldsymbol{20}(\mathbf{J}) &\leftarrow \boxplus^{20}\Diamond \mathit{jam}(\mathbf{J}). \end{aligned}$$

■

To establish the correspondence between the result of a set Q of CQL queries and its LARS translation $\Delta_L(Q)$, we first build a conversion of the input streams in Q to a LARS data stream. (Recall that LARS considers only a single stream which can be virtually split, e.g., by partition-based windows.) Without loss of generality, we assume that Q is evaluated on static relations $\mathtt{B}_1, \ldots, \mathtt{B}_m$ 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). We consider the union of these input streams, given by

$$\mathtt{S} = \{\langle \mathbf{c}_{ij}, t_{ij}\rangle \mid 1 \leq i \leq n, 1 \leq j \leq m_i\},$$

where the element $\langle \mathbf{c}_{ij}, t_{ij}\rangle$ represents the tuple $\mathbf{c}_{ij}$ that occurs at the j^{th} position at time t_{ij} in stream $\mathtt{S}_i$ (with m_i elements). We use the (lower case) name s_i of CQL stream $\mathtt{S}_i$ as predicate symbol of according atoms and thus obtain the LARS data stream by

$$\begin{aligned} \Delta(\mathtt{S}) &= (T, \upsilon) \text{ such that} \\ T &= [\min\{t_{ij}\}, \max\{t_{ij}\}] \text{ and} \\ \upsilon(t) &= \{s_i(\mathbf{c}_{ij}) \mid t_{ij} = t\} \text{ for all } t \in T; \end{aligned}$$

where $1 \leq i \leq n$ and $1 \leq j \leq m_i$. Similarly, we define for $\mathtt{B} = \mathtt{B}_1, \ldots, \mathtt{B}_m$ the atom set $\Delta(\mathtt{B}) = \bigcup_{i=1}^{m}\{b_i(\mathbf{c}) \mid \mathbf{c} \in \mathtt{B}_i\}$. Let $\mathit{cqlRes}(q, t)$ denote the set of resulting tuples of CQL query q at time t and let $\mathit{cqlRes}(Q, t) = \bigcup_{q\in Q} \mathit{cqlRes}(q, t)$. The following theorem shows that the translation Δ_L faithfully captures CQL. For a set A of atoms and a set Q of CQL queries let $A|_Q$ denote the set of all tuples $\mathbf{c}$ such that $\hat{q}(\mathbf{c}) \in A$ for some query $q \in Q$ (recall that $\hat{q}$ is the "output" predicate of the Datalog transformation of q).

Theorem 16. Let Q be a set of CQL queries to be evaluated on input streams $\mathtt{S} = \mathtt{S}_1, \ldots, \mathtt{S}_n$ and background relations $\mathtt{B} = \mathtt{B}_1, \ldots, \mathtt{B}_m$, $P = \Delta_L(Q)$, and t a time point. Moreover, let $M = \langle I, W, \Delta(\mathtt{B})\rangle$ such that W is implicit by Table 3 and windows mentioned in Q. Then,

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

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

Intuitively, (i) establishes the soundness and (ii) the completeness of the translation Δ_L.

Proof (Sketch). Consider a set Q of CQL queries and its translations $\Delta_D(Q)$ to Datalog and $\Delta_L(Q)$ to LARS, and moreover the data stream $\Delta(\mathtt{S})$ corresponding to CQL input stream S. First, we observe that the Datalog program $\Delta_D(Q)$ is an acyclic program and thus has as well-known a single answer set. Using the snapshot of the streaming data (i.e., the result of the S2R operator) as input, we thus get by the correctness from $RelAlg$ and Dat that the result of Q is captured by the answer set of $\Delta_D(Q)$. As noted above, the result of the S2R operator on an input source s amounts to abstracting away the temporal information. This step is carried out in LARS by existential temporal quantification with $\Diamond$ in the according window as listed in Table 3. We observe that snapshot rules, i.e. $\Delta_L(Q) \setminus \Delta_D(Q)$, add a stratified layer to $\Delta_D(Q)$ and faithfully derive the relations $rel(s)$ as follows. Provided encoding $\Delta(\mathtt{S})$ (and background data $\Delta(\mathtt{B})$ for static relations B), $rel(s)(\mathbf{c})$ will be derived iff $\mathbf{c}$ is a tuple in the snapshot of input source s in Q: any ground snapshot rule of form $rel(s)(\mathbf{c}) \leftarrow \boxplus^{w(s)}\Diamond s(\mathbf{c})$ must be satisfied when the formula $\boxplus^{w(s)}\Diamond s(\mathbf{c})$ is satisfied, thus the rule head $rel(s)(\mathbf{c})$ is concluded if the tuple $\mathbf{c}$ is contained in the snapshot; the only-if part is ensured by minimality of answer streams and the fact that relation names do not occur elsewhere as rule heads in the translation. From this also follows that the interpretation of these predicates must coincide in $\Delta_D(Q)$ and $\Delta_L(Q)$; the latter contains no further rules not contained in $\Delta_D(Q)$. It thus follows that the answer set of $\Delta_D(Q)$ corresponds to the answer stream of $\Delta_L(Q)$. That is to say, given the answer stream $I = (T, \upsilon)$ of $\Delta_L(Q)$ for $\Delta(\mathtt{S})$ at time t, $\hat{q}(\mathbf{c}) \in \upsilon(t)$ iff $\mathbf{c} \in cqlRes(Q, t)$. □

More details can be found in the Appendix.

5.3 Semantic Web: C-SPARQL and CQELS

Among research initiatives for the Semantic Web, RDF Stream Processing (RSP) emerged to address the question of querying heterogeneous streams. The RSP community is interested in extending SPARQL for streams in a similar way as CQL builds on SQL. In particular, C-SPARQL (Barbieri et al., 2010) and CQELS (Phuoc et al., 2011) employ an operational semantics that is based on the CQL approach of reducing stream reasoning to relational processing between a pre-processing of input streams and a post-processing towards output streams.

In (Dao-Tran et al., 2015a, 2015b) we investigated these two query languages for RDF data. We studied their difference which arises mainly due to the different execution modes. While C-SPARQL is *pull-based*, i.e., repeatedly returning a query result after a fixed temporal interval, CQELS is *push-based*, i.e., reporting results after every new input. We presented the comparative analysis by first formalizing these execution modes semantically for LARS programs. Then, we gave translations for the two RSP languages to LARS in a similar way as for CQL.

One difference between CQL and the SPARQL extensions are due to the fact that RDF graphs (i.e., sets of triples) do not come with a schema. While for CQL the integration of multiple input sources (tables and streams) is clear, there are different ways to integrate different input streams that arrive in RDF format. While C-SPARQL merges the relational snapshots into one graph (by stating them in the FROM-part of the query), CQELS provides explicit access to each input stream (in the WHERE clause of the query).

The central idea of capturing the push- and pull-based execution modes in LARS is to introduce an auxiliary atom c to rule bodies to control potential rule firing. The push-based mode will infer c whenever any atom holds in window $\boxplus^0$ that contains only the current time point. On the other hand, the encoding of pull-based execution amounts to testing for $\boxplus^0@_T\top$ whether current time T is a multiple of the query interval; only in this case, c shall hold.

Furthermore, window expressions of C-SPARQL and CQELS can easily be encoded as window operators in LARS. Note that LARS allows for using any kind of computable window function that does not

need further information than the input stream and the query time point. In fact, the time-based and tuple-based window functions as presented in Sections 2.3 and 2.4 correspond to those used in considered RSP languages.

Finally, the translation from RSP queries to LARS is based on an existing reduction from SPARQL to Datalog rules (Polleres, 2007). For C-SPARQL, it suffices to use a uniform representation of triples $t(s, p, o)$ that are available as input $i(s, p, o, x)$ in stream source x at some time point in the considered snapshot. Thus, the encoding essentially takes the form $t(s, p, o) \leftarrow \boxplus^w \Diamond i(s, p, o, x)$, where $\boxplus^w$ is the according window translation. For CQELS, we additionally have to disambiguate the stream source due to the so-called stream graph pattern as stated in the original WHERE clause.

Based on these translations, questions on the semantics of C-SPARQL and CQELS can then be made precise; in particular, when syntactically similar queries indeed produce the same results. In (Dao-Tran et al., 2015a) we formalized a notion of agreement and gave sufficient conditions for agreement for sliding time-based windows (under some restrictions). Our results formally reflect the drastic effect of execution modes on the query results from a semantic perspective.

5.4 Complex Event Processing: ETALIS

Related to stream processing is *complex event processing* (CEP), where one deals with the derivation of complex events (that typically span over temporal intervals) based on atomic events (that occur at time points). We briefly study the relation between LARS and the CEP language ETALIS (Anicic et al., 2010). This allows us to draw a line between stream reasoning and CEP by means of LARS.

In ETALIS, an *event stream* ϵ associates *atomic events* (represented as ground atoms) with time points, i.e., non-negative rational numbers. For comparability with LARS, we consider here only natural numbers. confine here to natural numbers. *Complex events* can be described by rules due to so-called *event patterns*, which resemble Allen's (1983) interval relations. An interpretation $\mathcal{I}$ is a function that maps atoms to sets of pairs $\langle t_1, t_2\rangle \in \mathbb{N} \times \mathbb{N}$ representing intervals $[t_1, t_2]$. Let $r = a \leftarrow pt$ be a rule, where a is an atom and pt an event pattern. Then, interpretation $\mathcal{I}$ *satisfies* r if all intervals assigned to pt are also assigned to a. Given an event stream ϵ and rule base $\mathcal{R}$, an interpretation $\mathcal{I}$ is model for $\epsilon, \mathcal{R}$, if $\mathcal{I}$ (i) maps each atomic event a to the interval $\langle t, t\rangle$ if a occurs in ϵ at time point t, and (ii) satisfies each rule $r \in \mathcal{R}$. The semantics of ETALIS is then defined in terms of minimal models,[9] which are always unique due to the definition of event patterns. More precisely, ETALIS employs a monotonic semantics computable by a straightforward fixed-point iteration.

Intervals in LARS. In contrast to ETALIS, the semantics of LARS is based on time points. Nevertheless, we can represent intervals in LARS and thus capture the ETALIS event patterns under some restrictions. Consider a window function $w_{[\ell,u]}$ that selects the (maximal) substream of the interval $[\ell, u]$. Given a formula α, we define the abbreviations

$$\begin{aligned} [\![\ell, u]\!]\alpha &:= \boxplus^{w_{[\ell,u]}} \Box\alpha\,, \\ \langle\!\langle \ell, u\rangle\!\rangle\alpha &:= [\![\ell, u]\!]\alpha \wedge @_{\ell-1}\neg\alpha \wedge @_{u+1}\neg\alpha\,. \end{aligned}$$

That is to say, formula $[\![\ell, u]\!]\alpha$ holds iff α holds at every time point in the interval $[\ell, u]$, regardless of the query time. Similarly, $\langle\!\langle \ell, u\rangle\!\rangle\alpha$ holds iff $[\ell, u]$ is a *maximal* interval in which α always holds.

[9] Where model comparability is defined as $\mathcal{I} \leq \mathcal{J} \Leftrightarrow \exists q \in \mathbb{R}_0^+ \, \forall a \, \forall \langle t, t'\rangle \in \mathcal{I}(a) : t' - t \leq q \Rightarrow \langle t, t'\rangle \in \mathcal{J}(a)$; that is, inclusion of intervals is relative to some arbitrary interval length q.

Example 25. Consider two events, x and y, which hold in the intervals $\langle t_1, t_2\rangle$ and $\langle t_3, t_4\rangle$, respectively, and assume $t_2 < t_3$. An ETALIS rule $r\ z \leftarrow x$ SEQ y thus assigns the interval $\langle t_1, t_4\rangle$ to z. With the above syntactic abbreviations, we may express r in LARS as

$$[\![t_1, t_4]\!]z \leftarrow \langle\!\langle t_1, t_2\rangle\!\rangle x,\ \langle\!\langle t_3, t_4\rangle\!\rangle y,\ t_2 < t_3.$$

That is, if $[t_1, t_2]$ is a maximal interval in which x holds, and $[t_3, t_4]$, where $t_2 < t_3$, is a maximal interval in which y holds, then z must hold at every time point in $[t_1, t_4]$. ∎

However, this straightforward encoding does not suffice to express ETALIS in LARS. The essential problem arises from overlapping intervals (for the same event/formula). LARS assigns atoms to a single timeline by an evaluation function $\upsilon : T \to 2^{\mathcal{A}}$. Unless an encoding makes use of time points in atoms, we can encode intervals only by assigning atoms to consecutive time points. Adjacent or overlapping intervals for the same atom cannot be distinguished, they all amount to a merged view of them. For instance, consider an event e that is assigned to $\langle 1, 4\rangle$ and $\langle 3, 5\rangle$ in ETALIS. By naively constructing a LARS interpretation υ, we would assign $\upsilon(t) = \{e\}$, for $t = 1, \ldots, 4$ and $t = 3, \ldots, 5$, and then only be able to read off a merged interval from 1 to 5. There is no way to distinguish the intervals $[t_1, t_4]$ and $[t_3, t_5]$ without explicitly encoding the boundaries of these atoms as terms in atoms such that they remain distinguishable.

For the sake of illustrating the capabilities of LARS regarding intervals, we now consider *separable* ETALIS interpretations, i.e., where 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 Example 25 allows us to faithfully translate positive ETALIS rule bases. The notion of minimality in LARS is based on set inclusion, whereas ETALIS defines minimality in terms of minimal length and the supportedness of inferred intervals (see Footnote 9). Notably, ETALIS defines a special form of negation (i.e., the NOT pattern) which ensures that a fixpoint iteration can be done. Using a natural translation for negation would give multiple LARS models in general. However, when confining to positive ETALIS programs with separable minimal models, a straightforward translation as indicated in Example 25 captures the ETALIS semantics.

The suggested intuitive approach is thus less expressive than ETALIS, where an atom can be assigned to overlapping intervals. On the other hand, the canonical minimal model of the ETALIS semantics can be computed by fixpoint-iteration for intervals of increasing size. Consequently, by explicitly encoding intervals $\langle t_1, t_2\rangle$ into atoms that contain t_1 and t_2 as terms, the bottom-up evaluation of such models can be emulated with LARS (as in Answer Set Programming). It remains as an interesting topic to find a suitable extension of LARS for nonmonotonic complex event processing (with multiple models) that has an interval-based evaluation function $\upsilon : T \times T \to 2^{\mathcal{A}}$.

6 Discussion

After having discussed relationships of LARS with selected formalisms, we now mention in Section 6.1 further research that is based directly on LARS. In Section 6.2 we will discuss the broader context of related work.

6.1 Further Work based on LARS

We now briefly discuss further work that is devoted to or based on LARS.

6.1.1 Theoretical Foundations

We first review theoretical work on LARS regarding program equivalence and techniques for incremental model update.

Equivalence notions. Towards optimizations of LARS programs, Beck et al. (2016) studied several notions of equivalence. They extended the notions of strong equivalence (SE) (Lifschitz et al., 2001), uniform equivalence (UE) (Eiter & Fink, 2003) and relativized uniform equivalence (RUE) (Woltran, 2004) from ASP to LARS and introduced data equivalence for streams. Based on a logic called Bi-LARS they captured the semantics of a large fragment of LARS, including plain LARS, by lifting the model-theoretic characterizations of SE/UE/RUE for ASP. Moreover, Beck et al. studied a special form of *monotone* windows, a class which includes time-based windows. Restricting to these allows for a variant of Bi-LARS that extends the logic of Here-and-There (Heyting, 1930), thus establishing a link to equilibrium logic (Pearce, 2006; Lifschitz et al., 2001) for the considered class of LARS programs. A final complexity analysis revealed that checking the considered equivalence relations is typically not worse than for ordinary ASP.

Incremental reasoning: answer update. Of special interest in stream reasoning is model update and incremental evaluation, in particular, when dealing with expressive languages such as LARS. To this end, we first extended in (Beck et al., 2015) Doyle's (1979) seminal justification-based truth maintenance system (JTMS) for a LARS fragment that was later called *plain LARS*. It extends normal logic rules of form

$$\alpha \leftarrow \beta_1, \ldots, \beta_n, \neg\beta_{n+1}, \ldots, \neg\beta_m$$

as follows: the rule head α can be an atom $a \in \mathcal{A}$ or a so-called *@-atom* of form $@_t a$ ($t \in \mathbb{N}$); and body elements β_i ($1 \leq i \leq m$) are *extended atoms*, given by the grammar

$$a \mid @_t a \mid \boxplus @_t a \mid \boxplus \Diamond a \mid \boxplus \Box a\,.$$

Doyle's JTMS deals with non-monotonic reasoning by updating a model in the light of new evidence. Technically, this change amounts to the addition of a new rule. Based on (partial) correspondence with stable model semantics (Elkan, 1990), we extended this maintenance technique for plain LARS. First, we extended the update mechanism for the time dimension which is included in the label of atoms. Secondly, according central JTMS data structures such as support and consequences have been adjusted accordingly.

In particular, the focus is on the concept of *stream-stratified* programs. In analogy to stratified negation, stream-stratified programs allow to split up a program into layers based on occurrences of window operators. This assumption aids efficient evaluation and is practical, since one can assume an acyclic flow of information in many applications. There, the output of any stream stratum serves as input for the next higher stratum. Notably, our approach accounts for a generic class of window operators as proposed by LARS as formal language.

6.1.2 Prototype implementations

Fragments of LARS have been implemented in several experimental prototypes (Beck et al., 2016, 2017; Bazoobandi et al., 2017) which are based on different realization principles.

LARS via ASP with external atoms. For a pilot application of LARS in multi-media data management discussed in Section 6.1.3, a specific fragment of LARS that amounts to plain LARS was implemented in dlvhex (Eiter et al., 2014, 2016), which is a solver for an extension of ASP with external atoms. The latter allow for an API style access to external information, which can be utilized to perform computations outside an ASP program. Specifically, sliding time-based windows have been realized as external atoms that access

the data stream viewed as an external object. While conceptually simple, this implementation is not geared towards performance and specifically lacks incremental reasoning techniques.

Ticker: incremental reasoning by program adjustment. More recently, Beck et al. (2017) developed another approach for practical, fully incremental reasoning, i.e., for sliding time- and tuple-based windows. While this work also exploits JTMS as specific update technique, it explores more generally the idea of updating a model by incrementally adjusting an encoding of a plain LARS program. Due to the correspondence with ASP, JTMS is a suitable mechanism for the show case. (In the absence of constraints and so-called odd loops JTMS computes and updates answer sets.)

Beck et al. considered two encodings: the first one is static and replaces window atoms by auxiliary atoms that are derived based on additional rules. For instance, the LARS rule

$$b(X) \leftarrow \boxplus^2 \Diamond a(X)$$

can be naturally translated into the following ASP rules, provided an additional predicate $now(t)$ is given to model the query time t.

$$\begin{array}{lll} r_0\colon\ b(X) & \leftarrow & w(X) \\ r_1\colon w(X) & \leftarrow & now(N), a_@(X,N) \\ r_2\colon w(X) & \leftarrow & now(N), a_@(X,N-1) \\ r_3\colon w(X) & \leftarrow & now(N), a_@(X,N-2) \end{array}$$

The idea similarly carries over for other operators, some of which are more subtle. Assuming that we remove the auxiliary predicate now and directly deal with partial groundings, we get, e.g., at time $t = 7$:

$$\begin{array}{lll} r'_1\colon w(X) & \leftarrow & a_@(X,7) \\ r'_2\colon w(X) & \leftarrow & a_@(X,6) \\ r'_3\colon w(X) & \leftarrow & a_@(X,5) \end{array}$$

We observe that the rules r'_1, r'_2, r'_3 cover the timeline $[5, 7]$. If we now move to time 8, i.e. the timeline $[6, 8]$, we can keep any groundings for time points 6 and 7; only groundings for rule r'_3 need to be removed, and suitable rules for time 8 have to be added. Thus, based on the window information and the development of the stream (in terms of time, respectively new atoms), new rules are added and some rules expire. For the latter, we can efficiently determine when a rule expires.

Based on this principle, we have built a reasoning engine called Ticker[10] that can be run in two evaluation modes. The first one exploits the static encoding and calls Clingo (Gebser et al., 2014) for ASP-based evaluation. From a semantic perspective, this mode is sufficient for use cases that always have a unique solution, i.e., a single model. The second mode uses the incremental technique and carries out the program update (and thus the model update) by our own implementation and extension of JTMS that also allows for removing rules. In case of multiple potential solutions (answer streams), the system will compute one model at random and then maintain it due to the operational semantics of JTMS. Our experimental evaluations indicate a clear performance benefit of incremental evaluation. For a detailed, formal review of the employed JTMS techniques we refer to (Beck, 2017).

Laser: high performance incremental reasoning. In (Bazoobandi et al., 2017) non-ground plain LARS with sliding windows was then considered with the aim of providing highly efficient model update. For applications that do not need multiple model reasoning, one can exploit additional structural information that

[10] Available at `https://github.com/hbeck/ticker`

arises from stratification. Hence, this work focused on positive and stratified programs, i.e., both stream-stratification and stratified negation. Existing semi-naive evaluation techniques as in Datalog (Abiteboul et al., 1995) have been extended to deal with the temporal dimension of LARS. In particular, for time-based windows, substitutions for non-ground formulas are annotated with two time markers that express the interval during which the according ground formula is guaranteed to hold. As long as the evaluation time is included in such an interval, the substitution can be retained. Thus, when time progresses, parts of the model may be carried over instead of being recomputed. Moreover, further annotations (i.e. guarantees) might be added incrementally for existing substitutions. In this way, the update process may partially reduce to updating annotations of existing derivations. On the other hand, substitutions may expire and are then removed efficiently due to the lookup of the respective time marker. The approach works similarly for tuple-based windows, under analogous annotations that refer to the global tuple count.

This research resulted in a prototype engine called Laser[11] that was evaluated empirically and compared against C-SPARQL (Barbieri et al., 2010), CQELS (Phuoc et al., 2011) and Ticker (using Clingo mode), all of which have been outperformed significantly. We note that Ticker and Laser have been developed in parallel and Ticker's incremental evaluation mode was not available for evaluation during the empirical study of Laser. However, the established results indicate that Laser will also be significantly faster than the incremental approach of Ticker.

We thus have two incremental reasoning engines for plain LARS using sliding (time-based and tuple-based) windows: Ticker may be used for problems with multiple models and Laser exploits the restriction to stratified negation to obtain high performance.

6.1.3 Applications of LARS

We now give examples how LARS may be used practically as modelling language, respectively as formal tool in theoretical work.

Cache Management in Content-Centric Networking. Beck et al. (2016, 2017) presented a show case application of stream reasoning with LARS. Research in the area of Content-Centric Networking (CCN) deals with future internet architectures for more efficient multimedia distribution. Clients issue interest packets, and routers return or need to retrieve data packets based on the content name. A router's caching strategy then decides which data item to store locally, and when to delete it. In our work, we investigated the use of a LARS-based decision component in a complex simulation environment for such an architecture. This implementation uses a specific fragment of LARS (which amounts to plain LARS) and was implemented in dlvhex (Eiter et al., 2014, 2016). This work tries to convey two key messages. First, switching caching strategies dynamically based on patterns of user demand leads to better cache hit ratios and a smaller average number of intermediate router forwards (hops) for content retrieval; and second, that the fully declarative control unit based on LARS is easier to maintain than imperative alternatives.

Streaming multi-context systems. Nonmonotonic Multi-Context Systems (MCS) (Brewka & Eiter, 2007) is a formal framework for interlinking knowledge bases, called contexts, via so called bridge rules for information exchange. In (Dao-Tran & Eiter, 2017), the latter have been generalized to streaming bridge rules, which utilize a fragment of LARS for processing of data streams that are dynamically generated by contexts. A semantics of streaming MCS was presented that lifts the key notion of (static) MCS equilibrium to an asynchronous execution model; it is the first semantics of this kind.

[11] Available at `https://github.com/karmaresearch/laser`

6.2 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) and C-SPARQL (Barbieri et al., 2010), as mentioned in Section 5.3, or SPARQL$_{\text{Stream}}$ (Calbimonte et al., 2010) follow up on the snapshot semantics of CQL (Arasu et al., 2003b, 2006) (cf. Section 5.2). Moreover, EP-SPARQL (Anicic et al., 2011), transfers event processing methods of ETALIS (Anicic et al., 2010) (cf. Section 5.4) to semantic web reasoning. It adds to the SPARQL syntax binary operators SEQ, EQUALS, OPTIONALSEQ and EQUALSOPTIONAL to combine query expressions (i.e., so-called graph patterns) similarly as UNION and OPTIONAL in SPARQL. These constructs introduce joins that depend on temporal information. For instance, given patterns P_1 and P_2, P_1 SEQ P_2 results in a join of P_1 and P_2 if they occur in a SEQ relation as in ETALIS, i.e., the instantiation of P_1 must occur strictly before that of P_2. Moreover, functions are provided for expressing conditions on the timestamps of the start time and the end time, and the duration of triples in the `FILTER` clause. The execution model exploits event-driven backward chaining rules as in ETALIS, couched in a Prolog implementation.

While the above SPARQL extensions provide a variety of ideas to lift static querying techniques for streams, they face difficulties with incorporating more expressive reasoning features as typically studied in Knowledge Representation and Reasoning (KR&R) like nonmonotonicity, default reasoning, or multiple possible solutions. Such features are important to deal with missing or incomplete data, respectively to enumerate alternative solutions and choices. Moreover, as observed by Dindar et al. (2013), conceptually identical queries may produce different results on different engines. This may be due to differences that either arise from potential flaws in implementations, but also due to (correctly implemented) different semantics. Comparisons between different approaches are confined to experimental analysis (Phuoc et al., 2012a) or informal examination on specific examples. For the user it is important to know the exact capabilities and semantic behaviors of given methods for systematic analysis and comparison.

In KR&R, first attempts towards expressive stream reasoning have been recently carried out and reveal many open problems. The plain approach of Do et al. (2011) periodically calls the dlvhex solver (Eiter et al., 2006) without incremental reasoning and thus cannot handle heavy data load. Another logic-based approach is Streamlog (Zaniolo, 2012) which extends Datalog for stream reasoning, motivated by a perceived lack of logical foundations of data stream management systems. By introducing so-called sequential programs that have syntactic restrictions on special temporal rules, Streamlog defines a non-blocking negation that can be used in recursive rules in a stream setting. Sequential programs are locally stratified and thus have unique models that can be computed efficiently by a fixpoint computation. The temporal predicates of Streamlog use the time point as first argument, i.e., an atom $p(t, x_1, \ldots, x_n)$ corresponds to $@_t\, p(x_1, \ldots, x_n)$ in LARS, where time is explicit; respectively orthogonal to logical truth.

Reasoning over streams has also been considered in ontology-based data access (OBDA). Ontology Stream Management Systems (OSMS), as introduced in (Ren & Pan, 2011), consider the use of Truth Maintenance Systems to deal with large volumes of data in EL++ reasoning. Streams of ontologies are also considered in the query language STARQL (Özcep et al., 2015) for query answering over streams. Neither Streamlog nor OSMS employ window mechanisms; STARQL on the other hand provides time-based windows.

Multiple works on the ASP solver Clingo have addressed the issue of data or program change. Incremental ASP (Gebser et al., 2008) introduced new techniques for incremental grounding and solving based on the module theory in (Oikarinen & Janhunen, 2006) which allows for composing programs with explicit (distinct) input and output atoms. An incremental program is a triple (B, P, Q) consisting of three program

parts: B describes static knowledge; P and Q are slices that depend on a parameter t. At each step t, the program grows by a new set $P[t]$, while $Q[t]$ is considered only temporarily at t. Relying on according composition of modules, model computation can then be carried out incrementally. The work resulted in the solver iClingo, which uses declarations `#base`, `#cumulative t`, and `#volatile t` to delineate program parts B, P, and Q from above, respectively. In a step k, the parameter (variable) `t` in a rule of program part P or Q is then instantiated with k. All other variables can be grounded only once, i.e., these instantiations must be derivable from static knowledge. While incremental ASP focuses on stepwise computation of models, reactive ASP (Gebser et al., 2011) targets real-time systems by providing additional means to add new data online. On top of incremental programs and its update mechanism, reactive programs support an asynchronous control via so-called *online progressions* of external events and inquiries which themselves are programs. In essence, at each step, external information can be incorporated to ground new rules dynamically. The resulting solver oClingo uses the additional declaration `#external` to introduce atoms that can be fed into the system in a streaming fashion. However, while reactive ASP provides incremental solving features for streaming data, it lacks a window mechanism. More precisely, the dynamic program parts P (which is cumulative) and Q (which concerns only the current step) do not fit the conceptual approach of windows which express the relevance of information relative to an interval of steps. With the aim of providing such a window mechanism for stream reasoning, *time-decaying logic programs* (Gebser et al., 2012) were defined as triples $(B, P, \{Q_1, \ldots, Q_m\})$, where the instantiation of each program part Q_i expires after a specified life span of n_i steps ($n_i \in \mathbb{N} \cup \{\infty\}$). Thus, each program part Q_i resembles functionality of a sliding window of length n_i. To incorporate this facility, oClingo's additional declaration of form `#volatile t : n` states that subsequent rules that are parameterized with variable `t` are discarded after `n` steps.

Ideas from incremental ASP, reactive ASP and time-decaying logic programs have been improved continuously and are now subsumed in the current version 5 of Clingo (Gebser et al., 2017). Its multi-shot solving capabilities to evaluate changing programs were presented earlier in (Gebser et al., 2015) which gave an introduction to multi-shot solving by modeling the board game Ricochet Robots. These works all provide additional control for grounding and solving via additional parameters that can be accessed by an external script. Such mechanisms can be used, e.g., to simulate the progress of time and to encode certain window operators. While Clingo's multi-shot features target the incremental and reactive control of the ASP solving process, LARS programs explicitly lift the ASP semantics for streams and provide novel language constructs than can be flexibly composed.

Another proposal for nonmonotonic stream reasoning is StreamRule (Mileo et al., 2013), which emphasizes the potential of ASP-based reasoning for the Semantic Web. The proposed architecture combines the Linked Sensor Middleware (LSM) (Phuoc et al., 2012b), CQELS for query processing and pattern matching, and oClingo for subsequent rule-based reasoning. In a similar way, the PrASP system (Nickles & Mileo, 2015) for probabilistic answer set programming is used as component of a probabilistic stream reasoning system architecture in (Nickles & Mileo, 2014). As LARS is not geared towards quantitative uncertainty, we can not directly model the semantics PrASP in our framework. Extending the framework in this direction is an interesting issue for future work.

A central challenge in incremental reasoning, in particular for materialization of Datalog programs, arises from fact deletion which requires to identify and retract inferred facts that are not derivable anymore. The Delete/Rederive (DRed) algorithm (Gupta, Mumick, & Subrahmanian, 1993) works by first overestimating deletions and then rederives them in case they still hold. This potential source of inefficiency was addressed by the Backward/Forward algorithm (Motik et al., 2015), which avoids the overdeletion phase. Instead, it determines by a combination of backward and forward chaining which of the facts that may have

to be removed have alternate proofs from remaining facts. The algorithm was shown to be significantly faster than DRed in some ontology query answering benchmarks. However, it is not applicable to programs with negation.

In Section 5.1, we already investigated the relation of LARS and Linear Temporal Logic (LTL). Metric Temporal Logic (MTL) (Koymans, 1990) is a notational variant of a fragment of Alur and Henzinger's (1993) timed propositional temporal logic, which has been conceived to model real-time systems. Informally, it generalizes LTL (and in presence of past time operators, PLTL), where operators $\mathsf{Op}_I\varphi$ allow for evaluating a formula φ on a set I of time points that can be any convex set (i.e., interval) or is defined by the congruence relation $\equiv_d c$. For example,

$$\boxplus_{[0,\infty)}(request \rightarrow \diamondplus_{[0,1]} grant)$$

expresses that each request is granted within one time step. Specifically, LTL resp. PLTL result for a uniform interval $I = [0, \infty)$ and the discrete time ontology $\langle \mathbb{N}, \leq \rangle$. Compared to LARS, MTL does not have general window operators (merely interval-bounded versions of the LTL operators G, F resp. G^{-1}, F^1 are available) but features all operators of LTL resp. PLTL; notably, time-based sliding windows of LARS with universal resp. existential formula evaluation correspond to MTL window operators; i.e.,

$$\boxplus^{a,b}\Box\varphi \text{ amounts to } \boxminus_{[0,a]}\ \varphi \wedge \boxplus_{[0,b]}\varphi \quad \text{and} \quad \boxplus^{a,b}\Diamond\varphi \text{ amounts to } \diamondminus_{[0,a]}\varphi \wedge \diamondplus_{[0,b]}\varphi.$$

Informally, $\boxminus_{[0,a]}\varphi$ (resp., $\diamondminus_{[0,a]}\varphi$) checks whether φ holds always (resp., at least once) within a steps back from the current time point resp. state; $\boxplus_{[0,b]}\varphi$ (resp., $\diamondplus_{[0,b]}\varphi$) is analogous for b steps forward. MTL has a timed state-sequence semantics with arbitrary time increase between successive states; this gives it an event-driven flavour, and comparability with LARS, where time increases by a single tick, is more difficult, in particular for LARS programs that resort to minimal models. Even for state-sequences with uniform time increase, nested windows may yield different results, as windows in LARS narrow down the input stream to a (finite) substream, while in MTL streams are at any evaluation stage infinite objects.[12] Furthermore, MTL is very expressive, as satisfiability and model checking are both ExpSpace-complete problems.[13]

While the use of MTL in stream processing has been advocated early on (Heintz & Doherty, 2004), the interest in it for stream reasoning has recently increased (Calvanese, Kalayci, Ryzhikov, & Xiao, 2016; Tiger & Heintz, 2016; Brandt, Güzel Kalaycı, Kontchakov, Ryzhikov, Xiao, & Zakharyaschev, 2017). Most recently, Brandt et al. (2017) have considered a Horn fragment of Metric Temporal Logic called Metric Time Datalog (datalogMTL). More precisely, it is a fragment of Alur et al.'s (1996) Metric Interval Temporal Logic (MITL), which uses a dense time ontology (the real numbers $\mathbb{R}$ resp. the rational numbers $\mathbb{Q}$) and allows to express truth of statements over a time interval rather than a single time point; as in MTL with dense time ontology, in MITL satisfiability and model checking are undecidable in general, but ExpSpace-complete if point intervals excluded. Brandt et al.'s datalogMTL programs contain rules of the form $A^+ \leftarrow A_1 \wedge \cdots \wedge A_k$ where each A_i is either (i) an inequality or (ii) a formula $\mathsf{Op}_{1\,I_1} \cdots \mathsf{Op}_{n\,I_n} B_i$, $n \geq 0$, where each Op_i is from $\{\boxplus, \boxminus, \diamondplus, \diamondminus\}$, I_i is an interval over $\mathbb{Q}$, and B_i is an atom, and where A^+ is a formula in (ii) without $\diamondplus$ and $\diamondminus$. They worked out a canonical model property for query answering, which is akin to the least model property of Horn logic programs, but technically more involved. Based on it, they showed that query answering is ExpSpace-complete already in the propositional case; notably, allowing $\diamondplus$ and $\diamondminus$ in rule head leads to undecidability. The canonical model property is then exploited to

[12] Remind, however, that the reset operator $\triangleright$ allows one to escape to the original stream; thus narrowing to substreams is not an essential restriction.

[13] Under the usual assumption that numbers are represented in binary. The results in Section 4 are invariant under tally (unary) representation.

develop a first-order rewriting for non-recursive (acyclic) Horn rules, referred to as $\mathrm{datalog}_{\mathrm{nr}}\mathrm{MTL}$; for this fragment, query answering is PSpace-complete. Modulo the divergence in window operators and different time ontologies, $\mathrm{datalogMTL}$ appears to be naturally related to LARS Horn rules resp. (negation-free) programs. It remains an issue for future work to study the relationship between $\mathrm{datalogMTL}$ and LARS in depth, and in particular to see whether notions and techniques for LARS (which is model-centric) can be exploited for $\mathrm{datalogMTL}$ (which is query-centric) and vice versa.

Finally, stream reasoning has also been considered from the perspective of spatio-temporal reasoning, e.g. in (Doherty, Kvarnström, & Heintz, 2009; de Leng & Heintz, 2016; Kontchakov, Pandolfo, Pulina, Ryzhikov, & Zakharyaschev, 2016).

7 Conclusion

We have presented LARS, a logic-based formalism to model and analyze stream processing respectively reasoning over data streams, where the data volume is reduced by using data snapshots called *windows*. LARS is equipped with a model-based semantics and offers besides "classical" semantics also an expressive rule-based language that adopts the answer set semantics for dealing with incomplete information.

Drawing from the observation what operations are typically performed on windows in stream processing, we have considered the temporal modalities $\Box$, $\Diamond$ and the nominal operator $@_t$; that is, LARS allows for distinguishing 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. After discussing some elementary properties of LARS programs, we have investigated the computational complexity of major reasoning tasks in LARS, viz. model checking and satisfiability testing, for both LARS formulas and programs, where we have put emphasis on specific classes of windows that are widely used in practice (time-based, tuple-based, and partition-based windows). In that, we have characterized the complexity of instance classes of the problems ranging from P-completeness in the propositional (ground) case to $\mathsf{NExpTime}^{\mathsf{NP}}$-completeness in the Datalog setting. Notably, the use of common windows in practice does not add to the worst-case complexity of the underlying fragment of temporal logic, nor does bounded nesting depth in general.

We have then related LARS to selected other languages and formalisms for reasoning over streams. We have shown that propositional LARS formulas with sliding time windows can be expressed in linear temporal logic, and thus amount to a (strict) fragment of the regular languages, while LARS program with such windows capture the regular languages. Furthermore, we have shown that queries in the well-known Continuous Query Language (CQL) (Arasu et al., 2006) can be captured by LARS, and we have briefly discussed that the operational semantics of the RDF Stream Processing (RSP) engines CQELS (Phuoc et al., 2011) and C-SPARQL (Barbieri et al., 2010) can modelled in LARS, while the prominent ETALIS language (Anicic et al., 2010), which is geared towards complex event processing, is beyond a natural representation.

The modelling of CQL, CQELS and C-SPARQL illustrates the usage of LARS as a formal representation and analysis framework, which was one of the motivations behind the formalism. As it turned out, the formalism and its affinity to Answer Set Programming make it attractive as a genuine reasoning language, which has been pursued in a pilot application in the context of content-centric network management (Beck et al., 2016, 2017); follow up research in cyber-physical systems are on the agenda. For these endeavors, recent experimental implementation prototypes of LARS fragments, that cater for incremental reasoning, will be instrumental.

7.1 Outlook and Future Work

There are several directions for future work. One is to enrich the LARS framework. Besides aggregates, towards event processing further temporal operators such as next time (X) and until (U), as well as their past time versions (X^{-1}, U^{-1}) can be considered. The entailment relation over streams can be naturally extended adhering to the usual semantics of these operators (where at the end of a stream, $\mathsf{X}\,\alpha$ evaluates to false). Accordingly, the evaluation algorithms discussed in Section 4 can be extended, without an increase of worst case complexity in the general case. However, the complexity of fragments of the language, as well their expressiveness might be affected.

Another direction is a refined picture of the complexity and expressivity of LARS fragments, both defined syntactically by restricting the rules and/or semantically the window operators resp. functions. The fragments introduced in (Beck et al., 2015, 2016) may serve here as a starting point. Based on the results, efficient implementations and optimization may be developed, where in particular incremental methods (as discussed in Section 6.1), data reduction via small window size (to be determined from the data) and approximation are of interest.

Related to the previous direction is to extend and deepen the investigation of the relationship to other formalisms and languages for stream reasoning, in particular to metric temporal logic and datalogMTL as the Horn fragment thereof. In connection with this, further reasoning tasks, such as prediction based on future evolutions of the stream might be considered; some initial results have been obtained in (Beck et al., 2016; Dao-Tran & Eiter, 2017).

Finally, further use of LARS to model stream reasoning languages, e.g. $\text{SPARQL}_{\text{Stream}}$ (Calbimonte et al., 2010), and to develop reasoning modules for applications remain to be continued.

A The LARS Framework

Proof of Theorem 1. Consider the structure $M = \langle I, W, B\rangle$, where $I \in AS(P, D, t)$. We first show that M is a model of P at time t, i.e., that $M, t \models r$ for all rules $r \in P$. There are two cases. If $r \in P^{M,t}$, satisfaction at t holds by definition. Else, let $r = \alpha \leftarrow \beta_1, \ldots, \beta_n$. We have $M, t \not\models \beta(r)$ and thus $M, t \models \beta(r) \rightarrow \alpha$.

As for minimality, suppose that $M' = \langle I, W, B\rangle$ where $M' \subset M$ is a model of P for D at time t. Then, $M', t \models r$ for each rule $r \in P$, and hence $M', t \models r$ for each rule $r \in P^{M,t} \subseteq P$. This means M' is a model of $P^{M,t}$ at time t; hence M is not a minimal model of $P^{M,t}$ at time t, which contradicts $I \in AS(P, D, t)$. ∎

Proof of Theorem 2. Let $I \in AS(P, D, t)$ and $t' \mapsto a \in I \setminus D$. By Definition 12, $M = \langle I, W, B\rangle$ is a minimal model of $P^{M,t}$ for D at t. Towards a contradiction, assume that for all $r \in P$ it holds that (i) $M, t \not\models \beta(r)$ or (ii) $M', t \models r$, where $M' = \langle I \setminus \{t' \mapsto a\}, W, B\rangle$. We first observe that item (i) cannot hold for all rules, as this would imply that $P^{M,t} = \emptyset$, which has the single minimal model $\langle D, W, B\rangle$ at t; this would imply $I = D$ and thus $t' \mapsto a \in I$ would be impossible, contradiction.

We now only consider those rules $r \in P$ where $M, t \models \beta(r)$, i.e., the reduct $P^{M,t} \neq \emptyset$. Since for all $r \in P^{M,t}$ we have that $M', t \models r$, where $M' = \langle I \setminus \{t' \mapsto a\}, W, B\rangle$, we conclude that M is not a minimal model of $P^{M,t}$. This yields the contradiction. ∎

Proof of Theorem 3. Under the asserted properties, clearly some model $M = \langle I, W, B\rangle$ of P exists for D at t; simply let in I all intensional atoms be true. Furthermore, any model $M' \subset M$ of $P^{M,t}$ for D at t satisfies $M', t \models P$, as under the monotonicity assertions $M', t \not\models \beta(r)$ for each (grounded) rule r in $P \setminus P^{M,t}$. By repeating this argument for M' etc., we can build a maximal, strictly decreasing chain of

models $M_0 = M, M_1, M_2, \ldots$ of P for D at t. The intersection N of all these models is another model of P for D at t, and hence no model $N' \subset N$ of P for D at t can exist. Consequently, N is also a minimal model of $P^{N,t}$ for D at t; in other words, N is an answer stream of P for D at t. This proves part (i). As for part (ii), by Theorem 1 each $I \in AS(P, D, t)$ is such that $M = \langle I, W, B\rangle$ is a minimal model of $P^{M,t}$ at t; conversely, the chain construction in part (i) starting with any minimal model $M = \langle I, W, B\rangle$ of P for D at t yields $N = M$, and thus $I \in AS(P, D, t)$ holds. ∎

B Computational Complexity

In this section, we provide proofs of the complexity results and further details on computation.

B.1 LARS Formulas α

Proof of Theorem 4. Let $M = \langle S^\star, W, B\rangle$, $S = (T^\star, \upsilon^\star)$, be a structure, let $S \subseteq S^\star$ be a substream of $S^\star$ and let α be a ground formula. Let N denote the size of M plus S.

PSpace *membership.* We show that the space used to determine $M, S, t \Vdash \alpha$ is bounded by $O(|\alpha| * N + N^k)$, where $|\alpha|$ is the size (length) of formula α and $k \geq 1$ is some constant.

Indeed, a ground formula α can be represented as a tree whose leaf nodes are atoms from $\mathcal{A}$ and whose intermediate nodes are operators from $\{\neg, \wedge, \vee, \rightarrow, \Diamond, \Box, @_t, \boxplus^w, \triangleright\}$, where $t \in T$. For example, the formula $\boxplus^{10}\Box(\boxplus^{\#3}\Diamond a \wedge (\boxplus^4\Diamond b \rightarrow \boxplus^5\Box(\neg c \wedge d)))$ can be represented by the tree in Figure 8.

The following strategy guarantees that evaluating a ground formula α remains in $O(|\alpha| * N + N^k)$ space. We travel the tree in a depth-first-search manner.

(1) When encountering a logical connective: evaluate the truth value of its sub-tree(s) and then combine the result using the semantics of the corresponding connective.

(2) When encountering a window operator $\boxplus^w$: extract a substream $S' = w(S, t)$ of the current stream S and store S' in a new place for evaluating the sub-tree of this operator.

(3) When encountering a $\Box$ or $\Diamond$ operator: iterate over the timeline T of the current window to determine the truth value of the sub-tree for each $t \in T$.

(4) When encountering an $@_t$ operator where t is a time point (and $t \in T$ where $S = (T, \upsilon)$): evaluate the sub-tree with reference to this time point.

(5) When encountering a leaf node: check the occurrence of the atom in the evaluation function at the current reference time point.

case (2) extends the space for setting up the environment for further checking of the sub-tree. The space for storing computed window $S' = w(S, t)$ is bounded by N, as a window is a substream of S. Furthermore, the computation of S' itself can be done in space N^k, for some constant $k \geq 1$, as it takes polynomial time in the size of S.

Furthermore, when visiting a node in the tree, we only need to consider the windows constructed by the window operators appearing on the path from the root to the current node. For other already visited branches, the space allocated for storing windows can be released. In case (3), we loop over all timepoints $t \in T$ and need only an iteration counter.

Therefore, at anytime, the space used is bounded by the depth of the tree times N, plus the space for window evaluation; this yields $O(|\alpha| * N + N^k)$.

PSpace *hardness.* Given a QBF of the form

$$\Phi = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \phi(x_1, x_2, \ldots, x_n), \tag{15}$$

where $Q_i \in \{\exists, \forall\}$, we translate it into a LARS formula

$$\alpha = W_1 \boxplus^{\mathrm{set}:x_1} W_2 \boxplus^{\mathrm{set}:x_2} \cdots W_n \boxplus^{\mathrm{set}:x_n} \phi(x_1, x_2, \ldots, x_n), \tag{16}$$

where for $1 \leq i \leq n$, $W_i = \Diamond$ if $Q_i = \exists$, $W_i = \Box$ if $Q_i = \forall$, and $\boxplus^{\mathrm{set}:x_i}$ is a window operator with an associated window function set^{x_i} defined as follows. Given a stream $S = (T, \upsilon)$ and a time point $t \in \{0, 1\}$:

$$\mathrm{set}^{x_i}(S, t) = (T', \upsilon'),$$

where $T' = T$ and for all $j \in T$:

$$\upsilon'(j) = \begin{cases} \upsilon(j) \setminus \{x_i\} & \text{if } t = 0, \\ \upsilon(j) & \text{if } t = 1. \end{cases}$$

That is, set^{x_i} removes x_i from the input stream S, if it is called at time $t = 0$, and it leaves S unchanged if it is called at $t = 1$; informally, this amounts to setting x_i to false (=0) and to true (=1), respectively.

Let now $S^\star = (T^\star, \upsilon^\star)$, where $T^\star = [0, 1]$ and $\upsilon^\star(0) = \upsilon^\star(1) = \{x_1, x_2, \ldots, x_n\}$ and let $M = \langle S^\star, W, \emptyset \rangle$, where $W = \{\boxplus^{\mathrm{set}:x_1}, \ldots, \boxplus^{\mathrm{set}:x_n}\}$.

Informally, $\Diamond \boxplus^{\mathrm{set}:x_1} \alpha'$ is entailed at $t = 0$ (likewise, at $t = 1$), if either at $t = 0$ or $t = 1$, after applying the window function $\mathrm{set}^{x_i}(S, t)$, the formula α' evaluates to true (=1) at t; that is, after either setting x_1 false (0) or to true (1), respectively. Dually, $\Box \boxplus^{\mathrm{set}:x_1} \alpha'$ is entailed at $t = 0$ (likewise, at $t = 1$) iff α' evaluates to true for both setting x_1 to false and to true. The nesting of the formula (16) thus mimics the QBF Φ in (15), as follows:

(i) the two time points 0 and 1 encode the truth values false and true, respectively.

(ii) By starting with the function $\upsilon^\star$ as the set $\{x_1, x_2, \ldots, x_n\}$ and by removing in $\boxplus^{\mathrm{set}:x_i}$ the atom x_i on the 0 branch and keeping x_i on the 1-branch, the evaluation of α can be seen as traversing a binary evaluation tree where the substream at each leaf node represents a complete truth assignment to $x_1, \ldots, x_n$. Figure 9 shows the tree with three variables.

(iii) The operator $\Diamond$ (resp. $\Box$) in front of $\boxplus^{\mathrm{set}:x_i}$ simulates the quantifier $\exists$ (resp., $\forall$): some (resp. every) of the subtrees, rooted at the 0 or 1 child, must evaluate to true.

A subtree of the tree starting at the root that fulfills the condition (iii) each satisfies the formula $\phi(x_1, \ldots, x_n)$ at each leaf witnesses then that Φ evaluates to true.

More formally, we show by induction on $i = 0, \ldots, n$ that if $S_{n-i} = (T^\star, \upsilon_{n-i})$ is a substream of $S^\star$ such that $\upsilon_{n-i}(0) = \upsilon_{n-i}(1) \supseteq \{x_{n-i+1}, \ldots, x_n\}$, then for the truth assignment σ to $x_1, \ldots, x_{n-i}$ such that $\sigma(x_j) \Leftrightarrow x_j \in \upsilon_{n-i}(0)$, it holds that

$$M, S_{n-i}, t \Vdash W_{n-i+1} \boxplus^{\mathrm{set}:x_{n-i+1}} \cdots W_n \boxplus^{\mathrm{set}:x_n} \phi(x_1, x_2, \ldots, x_n) \tag{17}$$

for any $t \in \{0,1\}$ iff

$$Q_{n-i+1}x_{n-i+1} \cdots Q_n x_n \phi(\sigma(x_1), \ldots, \sigma(x_{n-i}), x_{n-i+1}, \ldots, x_n), \tag{18}$$

evaluates to true; here $S_0 = S^\star$.

For $i = 0$, the stream S_n is a complete truth assignment to $x_1, \ldots, x_n$ and by construction the claim holds. For the inductive step, suppose the statement holds for i and consider $i+1$. By applying $\mathrm{set}^{x_{n-i}}$ on $S_{n-(i+1)+1} = S_{n-i}$ at $t = 0$, a stream of form S_{n-i+1} results, where x_{n-i} is set to false; thus by the induction hypothesis

$$\begin{aligned} & M, S_{n-i}, 0 \Vdash \boxplus^{\mathrm{set}:x_{n-i}} W_{n-i+1} \boxplus^{\mathrm{set}:x_{n-i+1}} \cdots W_n \boxplus^{\mathrm{set}:x_n} \phi(x_1, x_2, \ldots, x_n) \\ \text{iff} \quad & M, S_{n-i+1}, 0 \Vdash W_{n-i+1} \boxplus^{\mathrm{set}:x_{n-i+1}} \cdots W_n \boxplus^{\mathrm{set}:x_n} \phi(x_1, x_2, \ldots, x_n) \\ \text{iff} \quad & Q_{n-i+1}x_{n-i+1} \cdots Q_n x_n \phi(\sigma(x_1), \ldots, \sigma(x_{n-i}), x_{n-i+1}, \ldots, x_n) \text{ evaluates to true,} \end{aligned}$$

where $\sigma(x_{n-i}) = 0$; for $t = 1$ the argument is analogous, where "0" is replaced by "1". Hence by definition of $\Diamond$ resp. $\Box$, we obtain that (17) holds iff (18) holds. This proves the claim.

For $n = 0$, as $S_0 = S^\star$ we then obtain that $M, S^\star, t \Vdash \alpha$ for α in (16) and $t \in \{0,1\}$ iff Φ in (15) evaluates to true. As $M, S^\star$, α and t are computable in polynomial time from Φ, it follows that deciding $M, S, t \Vdash \alpha$, i.e., model checking for LARS formulas, is PSpace-hard. ∎

Proof of Theorem 5. PSpace membership follows from the fact that we can guess an evaluation function υ on T and then perform a model check $M, S, t \Vdash \alpha$ where $S = (T, \upsilon)$; relative to the set $\mathcal{A}'$ of atoms, the guess for υ has polynomial size, and thus the combined guess and check algorithm can run in NPSpace; as NPSpace = PSpace by Savitch's result (1970), it is thus in PSpace.

The PSpace-hardness follows from a simple reduction of model checking $M, S, t \Vdash \alpha$, where from the proof of Theorem 4 w.l.o.g. $S = S^\star = (T^\star, \upsilon^\star)$: we construct

$$\alpha_S = \alpha \wedge \bigwedge_{t \in T^\star, p \in \upsilon(t)} @_t p \wedge \bigwedge_{t \in T^\star, p \in \mathcal{A}' \setminus \upsilon(t)} @_t \neg p$$

i.e., fix the possible valuation to υ, and ask for an evaluation function υ' on $T^\star$ s.t. $M, (T^\star, \upsilon'), t \Vdash \alpha_S$. ∎

Proof of Theorem 6. 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 $M' = \langle I', W, B \rangle$, with $I' = (T, \upsilon') \subset (T, \upsilon)$ exists such that $M', t \models P^{M,t}$. 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 $M', t \models P^{M,t}$).

Overall, the computation is feasible in NPSpace, thus in PSpace (as NPSpace = PSpace).

The PSpace-hardness of the problem is easily obtained from Theorem 4: for given ground formula α and $M = \langle S, W, B \rangle$, let $P = \{\alpha \leftarrow \top\}$, where $\top$ is an arbitrary tautology. Note that no intensional data occur in α, and thus no interpretation M' that is smaller than M is possible, and thus $M = \langle S, W, B \rangle$ is an answer stream for P for D at t iff $M, S, t \Vdash P$ holds. ∎

Proof of Theorem 7. To show satisfiability of a ground LARS program P, we can guess a stream $I = (T, v)$ and check that I is an answer stream of P for D at t; the guess is polynomial in the size of $\mathcal{A}'$ and the check feasible in PSpace by Theorem 6; overall, the computation is feasible in NPSpace, thus in PSpace.

The PSpace-hardness of SAT for LARS programs P follows from the reduction of MC for LARS formulas to MC for LARS programs in the proof of Theorem 6. ∎

B.2 Bounded Window Nesting

Proof of Theorem 8. In the discussion preceding the results, a bound $N = (\#w(\alpha) * |T|)^{wnd(\alpha)}$ on the number of substreams that emerge in the recursive evaluation of α has been derived. Clearly the set of all these substreams forms an evaluation base $\mathcal{S}_B$ for M, S, α. If $wnd(\alpha)$ is bounded by a constant k, then N is polynomially bounded in the size of M and α. The result then follows immediately from Theorem 10. ∎

Proof of Theorem 9. Membership in co-NP can be seen as follows: the PSpace oracle in the algorithm considered in the proof of Theorem 6 can by Theorem 8 be replaced by a polynomial-time computation. It follows that we can refute nondeterministically in polynomial time that I is an answer stream of P for D at t. Consequently, problem MC is for LARS programs P^- in co-NP. On the other hand, co-NP-hardness is inherited from the co-NP-completeness of answer set checking for (disjunctive) propositional logic programs, cf. (Eiter & Gottlob, 1995), which is subsumed by model checking for LARS programs. ∎

B.3 Semantic Restriction: Sparse Windows

Proof of Lemma 1. Indeed, by traversing the recursive Definition 9, we can add the edges of $WG_{\mathcal{S}_B} = (N, E)$ as described, by calculating each $w_k(S_{k-1}, t)$, which takes polynomial time; there are $|T^\star|$ many such calculations to make, and in total thus at most $\#w(\varphi) * |T^\star|$ many, where $\#w(\varphi)$ is the total number of window occurrences in φ. In order to find $w_k(S_{k-1}, t)$ in $\mathcal{S}_B$, i.e., the stream $S' \in \mathcal{S}_B$ such that $S' = w_k(S_{k-1}, t)$ one can use hashing or, if the stream is to large, use a trie structure which makes this feasible in $O(\|S'\|)$ time, where $\|S'\|$ is the size of S' (which is $O(|\mathcal{A}| * |T|)$).

Thus in total the time to compute $WG_{\mathcal{S}_B}$ for (M, S, φ) is

$$O(\#w(\varphi) * |\mathcal{S}_B| * |T^\star| * (C_w + \|S^\star\|) + |\varphi|) \;=\; O(|\varphi| * |\mathcal{S}_B| * |T^\star|^{k+1} * |\mathcal{A}|^k), \tag{19}$$

where $C_w = O(\|S^\star\|^k) = O(|T^\star|^k * |\mathcal{A}|^k)$ is a polynomial in $\|S^\star\|$ that bounds the evaluation time of any window. Indeed, there are at most $\#w(\varphi) * |\mathcal{S}_B| * |T^\star|$ many edges to consider, and computing plus matching a window S' against $\mathcal{S}_B$ takes $O(C_w + \|S^\star\|)$ time. Overall, this is polynomial in the size of M, $\mathcal{S}_B$, and φ. ∎

Proof of Proposition 2. The statement is proved by induction on the structure of the formula φ. In the base case, φ is a single atom a, and by construction the windows graph $WG_{\mathcal{S}_B}$ has no edges. The node (S, t), $t \in T^\star$ is labeled with a iff $a \in v(t)$, where $S = (T, v)$, and thus $M, S, t \Vdash \varphi$ iff $\varphi \in L_{\mathcal{S}_B}(S, t)$ holds. In the induction step, assume that the statement holds for all subformulas of φ, and consider the different cases of the root connective *op* of φ. From the induction hypothesis that or each M, S, t and subformula α of φ, the entailment $M, S, t \Vdash \alpha$ is correctly reflected by $\alpha \in L_{\mathcal{S}_B}(S, t)$, it is not hard to verify that in each case, φ is added to the label of (S, t) if and only if $M, S, t \Vdash \varphi$ holds. Note in this context that for

$op = \boxplus^w$ and $op = \triangleright$, the window graph $WG_{\mathcal{S}_B}$ for φ contains the one for $M, S', t \Vdash \alpha$ resp. $M, S, t' \Vdash \alpha$ or $M, S^\star, t \Vdash \alpha$ in the recursive Definition 9 as a subgraph. ∎

Proof of Theorem 10. From a window graph $WG_{\mathcal{S}_B} = (N, E)$ for (M, S, φ), where $M = \langle S^\star, W, B\rangle$, we can drop each node $S' \neq S$ from $\mathcal{S}_B$ that does not occur in E; the remaining graph $WG_{\mathcal{S}^\star_B} = (N', E)$ is the smallest window graph possible, i.e., $N' \subseteq N$ holds for each window graph $WG_{\mathcal{S}'_B} = (N, E)$ for (M, S, φ). Notably we can build $\mathcal{S}^\star_B$ on the fly by initially setting $\mathcal{S}^\star_B = \{S\}$ and by then adding any $S' = w_k(S_{k-1}, t)$ along the window path that is not yet member of $\mathcal{S}^\star_B$. Following the analysis in Lemma 1, as inserting a stream S into $\mathcal{S}_B$ is like searching feasible in $O(\|S\|)$ time, building $WG_{\mathcal{S}^\star_B}$ take also time bounded by (19), i.e., by $O(\#w(\varphi) * |\mathcal{S}_B| * |T^\star| * (C_w + \|S^\star\|) + |\varphi|) = O(|\varphi| * |\mathcal{S}_B| * |T^\star|^{k+1} * |\mathcal{A}|^k)$.

The time to construct the bottom labeling is bounded by $O(|T^\star| * |\mathcal{S}^\star_B| * |\varphi|)$: for each subformula φ' of φ and pair (S', t'), where $S \in \mathcal{S}^\star_B$ and $t' \in T^*$ we have to decide whether φ' is put in $L_{\mathcal{S}_B}(S', t')$. This can be decided by constantly many lookups of already constructed labels for subformulas of φ'; for $\Diamond\alpha$, we can use a flag that is set true if (S', t') is labeled with α, where $t' \in T$. For $\Box\alpha$, we can proceed similarly.

The total runtime of the algorithm is thus bounded by

$$O(|\varphi| * |\mathcal{S}_B| * |T^\star|^{k+1} * |\mathcal{A}|^k + |T^\star| * |\mathcal{S}^\star_B| * |\varphi|) \;=\; O(|\varphi| * |\mathcal{S}_B| * |T^\star|^{k+1} * |\mathcal{A}|^k),$$

Given that the size of $\mathcal{S}_B$ is polynomial in the size of M and φ, it follows that the runtime is polynomial in the size of M and φ. ∎

Proof of Theorem 11. By Theorem 4, it remains to show PSpace-hardness. For this, we reconsider the reduction from evaluating a QBF $\Phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$ as in (15) to model checking for a LARS formula (16), and adapt the reduction as follows.

For each atom x_i, we introduce a fresh atom w_i, and we change the content of stream $S^\star = ([0, 1], \upsilon^\star)$ to $\upsilon^\star(0) = \{x_1, \ldots, x_n\}$, $\upsilon^\star(1) = \{w_1, x_1, \ldots, w_n, x_n\}$. Furthermore, we replace the window $\boxplus^{\text{set}:x_1}$ with the partition-based window $\boxplus^{\text{idx}^{(i)},n^{(i)}}$, where $\text{idx}^{(i)}$ creates two partitions $\text{idx}^{-1}(1) = \{w_i, x_i\}$ and $\text{idx}^{-1}(2) = \mathcal{A} \setminus \{w_i, x_i\}$, with the counts $n^{(i)}(1) = (1, 2)$ and $n^{(i)}(2) = (\infty, \infty)$, where ∞ can be replaced by any number $\geq 2 * (n - 1)$. For making the selection deterministic, we assume any total order $\leq$ (e.g., lexicographic order) such that $w_i \leq x_i$ for all $i = 1, \ldots n$.

Informally, these changes have the following effects:

- evaluated at time point 0, the partition based window function $\text{p}^{\text{idx}^{(i)},n^{(i)}}$ will for the partition $\{w_i, x_i\}$ remove at time0 one atom ($\ell^{(i)} = 1$); as w_i is not in $\upsilon^\star(0)$, it will remove x_1; at 1, it will remove two atoms ($u^{(i)} = 2$), and thus both w_i and x_i.
- evaluated at time point 1, $\text{p}^{\text{idx}^{(i)},n^{(i)}}$ will for the partition $\{w_i, x_i\}$ remove at time 1 one atom (as $\ell^{(i)} = 1$), and as $w_i < x_i$, it will remove w_i; the count $u^{(i)} = 2$ has no effect as 1 is the maximal time point in $T^\star = [0, 1]$.

Thus, the stream $\text{p}^{\text{idx}^{(i)},n^{(i)}}(S^\star, 0)$ has neither x_i nor w_i at $t = 0, 1$, and $\text{p}^{\text{idx}^{(i)},n^{(i)}}(S^\star, 0)$ has x_i at $t = 0, 1$ and w_i not at $t = 0, 1$. Thus, the evaluation of the formula

$$\alpha' = W_1 \boxplus^{\text{idx}^{(1)},n^{(1)}} W_2 \boxplus^{\text{idx}^{(2)},n^{(2)}} \cdots W_n \boxplus^{\text{idx}^{(n)},n^{(n)}} \phi(x_1, x_2, \ldots, x_n),$$

on the modified $S^\star$ generates at the innermost evaluation level the same streams

$$\text{p}^{\text{idx}^{(n)},n^{(n)}}(\text{p}^{\text{idx}^{(n-1)},n^{(n-1)}}(\cdots \text{p}^{\text{idx}^{(1)},n^{(1)}}(S^\star, t_1), \cdots), t_{n-1}), t_n),$$

where $t_1, \ldots, t_n \in \{0, 1\}$, as the evaluation of the formula α in (16) on the original $S^\star$ at the innermost level, which are given by $\mathrm{set}^{x_n}(\mathrm{set}^{x_{n-1}}(\cdots \mathrm{set}^{x_1}(S^\star, t_1), \cdots), t_{n-1}), t_n)$.

Consequently, α' is entailed at an arbitrary $t \in \{0, 1\}$ on the modified $S^\star$ iff α is entailed at an arbitrary $t \in \{0, 1\}$ on the original $S^\star$. This proves PSpace-hardness.

We note that the reduction proves the result where each partition-based window creates only two (individual) partitions, but all such windows use the same tuple counts $n^{(i)} = (1, 2)$. The reduction above can be easily adjusted to partition-based windows that use all the *same partitioning* but *different tuple counts*: just let $\mathrm{idx}^{(i')} = \mathrm{idx}$, where $\mathrm{idx}(j) = \{w_j, x_j\}$ and $n^{(i')}(j) = (\ell_j^i, u_j^i)$, for $i, j = 1, \ldots, n$, such that $\ell_j^i = u_j^i = \infty$ if $j \neq i$, and $\ell_j^i = 1$ and $u_j^i = 2$ if $j = i$; then each associated partition-based window function $\mathrm{p}^{\mathrm{idx}^{(i')}, n^{(i')}}$ clearly coincides with $\mathrm{p}^{\mathrm{idx}^{(i)}, n^{(i)}}$, which implies PSpace-hardness for this setting. ∎

Proof of Theorem 12. To begin with, each reset operator $\triangleright$ that occurs in the formula α returns the stream $S^\star$, and thus we need to deal with substreams of S and $S^\star$ in the evaluation of α. It is sufficient to consider just S (and substreams of it), as $S = S^\star$ is covered and the result then clearly follows.

As argued in the discussion, each result of a time-based window function $\tau^{l\ell,u,d}(S, t)$ can be expressed as the result of a tuple-based window function $\#^{\ell',u'}(S, t')$, where the counts ℓ' and u' are set such that exactly the atoms in $\tau^{l\ell,u,d}(S, t')$ will be selected. Furthermore, each result of a tuple-based window function $\#^{\ell,u}(S, t)$ can be viewed as the result of a dummy partition-based window function $\mathrm{p}^{\mathrm{idx}_\mathcal{A}, n_\mathcal{A}}(S, t)$ where $\mathrm{idx}_\mathcal{A} : \mathcal{A} \to \{1\}$ and $n_\mathcal{A}(1) = (\ell, u)$, i.e., only one partition exists that contains all tuples; clearly, this trivial partition is always the union $B_1 \cup \cdots \cup B_k$ of all base groups B_i for meager partitionings. Thus, it is sufficient to consider the latter case.

Consider a stream $S = (T, \upsilon)$, $T = [t_\ell, t_u]$, and some time point $t \in T$. We can describe with a tuple $d_i = (\ell_1, u_1, \ldots, \ell_k, u_k, t)$ stream S, where ℓ_i states how many atoms in S from $[t_\ell, t]$ that are in partition B_i should be included, and u_i similarly how many atoms in S from $[t + 1, t_u]$ are in B_i. Now if $\mathrm{p}^{\mathrm{idx},n}(S, t')$ is applied on S, we can single out, for each partition $\mathrm{idx}^{-1}(j) = \bigcup\{B_i \mid B_i \subseteq \mathrm{idx}^{-1}(j)\}$, how many atoms back (resp. forward) from t' have to be included, depending on $n(j) = (\ell_j^{\mathrm{idx}}, u_j^{\mathrm{idx}})$ and we can break this down to counts ℓ_i', u_i' for all base groups B_i, $i = 1, \ldots, k$. Thus we obtain a description $d_i' = (\ell_1', u_1', \ldots, \ell_k', u_k', t')$ that describes $\mathrm{p}^{\mathrm{idx},n}(S, t')$. Overall, there are polynomially many such descriptions in the size of the input.

Finally, we extend the description d_i for a substream with a filter A to $sd = (\ell_1, u_1, \ldots, \ell_k, u_k, t, A)$, where for describing the initial stream S we can use $A = \mathcal{A}$ (i.e., no atom is filtered). Clearly, if we apply a meager partition-based window function on S at t', the description $sd' = (\ell_1, u_1, \ldots, \ell_k, u_k, t, A)$ of S can be adjusted to represent $\mathrm{p}^{\mathrm{idx},n}(S, t')$ by a description $sd = (\ell_1', u_1', \ldots, \ell_k', u_k', t', A)$;[14] in case of a filter window application, we can represent $\boxplus^{A'}(S, t')$ by $sd' = (\ell_1', u_1', \ldots, \ell_k', u_k', t', A \cap A')$. Consequently, the possible descriptions that result are of the form $sd = (\ell_1, u_1, \ldots, \ell_k, u_k, t, A_1 \cap A_1 \cap \cdots \cap A_i)$ where $\boxplus^{A_1}, \boxplus^{A_2}, \ldots, \boxplus^{A_i}$ are the filter windows encountered on some path from the root of the formula tree of φ; overall, the number of paths (and thus such sequences) is bounded by the size of φ. Hence, overall the number of extended descriptions is polynomially bounded in the size of the input. ∎

B.4 Non-ground Case

Proof of Theorem 13. *Combined Complexity.* We first consider the combined complexity.

[14] We tacitly assume here that the order of atoms that is used to resolve non-determinstic selection is static and does not depend on the time point t.

(MC) Under the above assumptions, model checking for a LARS formula $\alpha(\mathbf{x})$ on a structure $M = \langle S^\star, W, B\rangle$ w.r.t. S and time point t is PSpace-complete; this is because an instance α' of $\alpha(\mathbf{x})$ where this fails can be guessed and verified in PSpace.

Likewise, MC for non-ground LARS programs P is in PSpace; to establish this, the algorithm in the proof of Theorem 6 has to be adapted, in that it is avoided that the reduct $P^{M,t}$ is built explicitly; rather, all ground instances of all rules $r \in P$ are considered one by one to check satisfaction of $P^{M,t}$, which is feasible in polynomial space.

The PSpace hardness for both α and P is inherited from the ground case. We note, however, that already for formulas $\alpha(\mathbf{x}) = \neg(p_1(\mathbf{x_1}) \wedge \cdots \wedge p_k(\mathbf{x_k}))$ i.e., negated conjunctive queries, the problem is co-NP-hard; this follows from the classic result that satisfiability of a conjunctive query is NP-complete (Chandra & Merlin, 1977).

Bounded window nesting. For the LARS formulas in the class α^-, the problem MC is co-NP-complete: a ground instance α' of α such that $M, S, t \not\Vdash \alpha'$ can be guessed and verified in polynomial time; the co-NP-hardness is inherited from the NP-completeness of conjunctive query evaluation.

Similarly, for LARS programs P^- the complexity increases from co-NP in the ground case to co-Σ_2^p, as the evaluation of LARS formula α^- is co-NP-complete: checking (c.1) that $M, t \models P^{M,t}$ is in co-NP and (c.2) that no $I' \subset I$ for D exists such that $M, I', t \Vdash P^{M,t}$ is in co-$\Sigma_2^p = \Pi_2^p$: a guess for such an I', which is of polynomial size, can be verified with an NP oracle in polynomial time. Combining (a)-(c), this leads to membership in co-$\Sigma_2^p = \Pi_2^p$. The matching Π_2^p-hardness is inherited from answer set checking of non-ground Datalog programs, see (Eiter et al., 2007).

(SAT) As regards SAT, arbitrary predicate arity causes an exponential size grounding for the valuation υ on T, i.e., for $S = (T, \upsilon)$, as the size of the set $\mathcal{A}$ of all atoms becomes exponential. However, once υ is available, we can for a ground formula α decide $M, S, t \Vdash \alpha$, where $M = \langle S^\star, W, B\rangle$, in time exponential in the size of the problem input: modulo window evaluation, the recursive evaluation procedure from Theorem 4 runs in polynomial space (iterations resp. guesses are made over the timeline $T^\star$); in total, it will encounter at most exponentially many points of substream creation (i.e., window evaluation) in the size of $T^\star$ and α. Each substream creation is polynomial in the size of S; as the latter is exponential in the size of the problem input, it follows that the total time spent to create substreams will also be exponential in the size of the problem input. Thus, deciding $M, S, t \Vdash \alpha$ is feasible in time exponential in the size of the problem input; if the size of $S^\star$ is exponential in the number $|\mathcal{P}| + |\mathcal{C}|$ of predicates and constants, it is feasible in time polynomial in the size of $S^\star$. Thus for open LARS formulas α, the complexity of SAT is in NExpTime, while for non-ground LARS programs P, it is in $\mathsf{NExpTime}^{\mathsf{NP}}$: an exponential size guess $I = (T, \upsilon)$ for an answer stream for P of D can be verified with an NP oracle in polynomial time. The matching hardness results follow from results for the complexity of disjunctive Datalog programs (Eiter et al., 1997): for LARS formulas, NExpTime-hardness is inherited from the NExpTime-completeness of deciding whether a disjunctive Datalog program P has some classical (Herbrand) model, and for LARS programs $\mathsf{NExpTime}^{\mathsf{NP}}$-hardness follows from the $\mathsf{NExpTime}^{\mathsf{NP}}$-completeness of deciding whether a disjunctive Datalog program P has some answer set. Both these results are easy corollaries to the proof of Theorem 8.5 in (Eiter et al., 1997). As P can be seen as an open LARS formula and no window operators occur in it, the hardness results hold already for α^- resp. P^-.

Data Complexity. The data complexity of LARS formulas and LARS programs (i.e., the formula α resp. the program P is fixed and just varying the data stream D and the time point t is varied) coincides with the one of ground LARS formulas α^- (resp. programs P^-), as shown in Table 1. Indeed, α (resp. P) can be reduced to an equivalent ground formula α' (resp. ground program P') by instantiation in polynomial time, as only polynomially many ground instances of α (resp. each rule in P) exist. The matching lower bounds, are

obtained from: (a) arbitrary polynomial-time window functions, which implies P-hardness of MC for fixed LARS formulas (b) the data complexity of disjunctive Datalog program; from the proofs in (Eiter et al., 1997), it follows that under data complexity, model checking is co-NP-complete; deciding classical model existence is NP-complete; and deciding answer set existence is Σ_2^p-complete. This proves the result. ∎

C Relation to other Languages and Formalisms

C.1 Temporal Logic

Proof of Proposition 3. The first equivalence is shown by induction on the structure of the formula; the second follows trivially. The base case of an atom is trivial; the other cases follow easily from the induction hypothesis. Indeed, the cases where φ is a Boolean combination are immediate; likewise, for $\Box\alpha$ and $\Diamond\alpha$ simple quantifier elimination works, and for $@_{t'}\alpha$ moving to the respective position. For the case of $\boxplus^{i,j}\alpha$, the window around t is properly calculated, where $T' = [\ell', u']$ and $\ell' = \max(\ell, t-i)$ and $u' = \min(t+j, u)$; note that $t \in T'$ holds. Finally, for $\triangleright\alpha$, as time-based windows do not remove content, all what needs to be done is to reset the bounds of the interval considered. ∎

Proof of Theorem 14. For a formula φ in which each occurrence of $\Box$, $\Diamond$ and $@_{t'}$ is windowed (call such φ windowed), by the argument given in the discussion after Theorem 14, if we set $\text{PLTL}(t', \varphi)$ to the formula (12), then $M, S, t' \Vdash \varphi$ iff $\pi(M), t' \models \text{PLTL}(t', \varphi)$ holds.

Exploiting this, we can transform a formula $\varphi = @_t\alpha$, where α is windowed, to

$$\text{PLTL}(t', \varphi) = \neg\mathbf{u} \wedge \mathsf{G}^{-1}(\neg\mathsf{X}^{-1}\top \rightarrow \mathsf{X}^t\,\text{PLTL}(t, \alpha)); \tag{20}$$

that is, move to the initial position 0 of the path and then check that at position t the formula $\text{PLTL}(t, \alpha)$ holds.

For a formula $\varphi = \Box\alpha$, where α is windowed, we can similarly as in (20) write naively

$$\neg\mathbf{u} \wedge \mathsf{G}^{-1}(\neg\mathsf{X}^{-1}\top \rightarrow \textstyle\bigwedge_{t=0}^{\infty} \mathsf{X}^t(\mathbf{u} \vee \text{PLTL}(t, \alpha)); \tag{21}$$

this is not an LTL-formula, but again we observe that only finitely many $\text{PLTL}(t, \alpha)$ will be produced: for some $t \geq t_0$ large enough, all formulas $\text{PLTL}(t, \alpha)$ are identical. Thus we can set

$$\text{PLTL}(t', \varphi) = \neg\mathbf{u} \wedge \mathsf{G}^{-1}(\neg\mathsf{X}^{-1}\top \rightarrow \Big(\textstyle\bigwedge_{t=0}^{t_0-1} \mathsf{X}^t(\mathbf{u} \vee \text{PLTL}(t, \alpha)) \wedge \mathsf{X}^{t_0}\,\mathsf{G}(\mathbf{u} \vee \text{PLTL}(t_0, \alpha))\Big). \tag{22}$$

For a formula $\Diamond\alpha$ where α is windowed, the transformation is analogous, viz.

$$\text{PLTL}(t', \varphi) = \neg\mathbf{u} \wedge \mathsf{G}^{-1}(\neg\mathsf{X}^{-1}\top \rightarrow \Big(\textstyle\bigvee_{t=0}^{t_0-1} \mathsf{X}^t(\neg\mathbf{u} \wedge \text{PLTL}(t, \alpha)) \vee \mathsf{X}^{t_0}\,\mathsf{F}(\neg\mathbf{u} \wedge \text{PLTL}(t_0, \alpha))\Big). \tag{23}$$

This argument can be extended to show by induction on the structure of an arbitrary LARS formula φ that some LTL-formula $\text{PLTL}(t', \varphi)$ exists such that $M, S, t' \Vdash \varphi$ iff $\pi(M), t' \models \text{PLTL}(t', \varphi)$; where the induction statement includes that some $t_0 \geq 0$ exists such that for all $t' \geq t_0$ we have $\text{PLTL}(t', \varphi) = \text{PLTL}(t_0, \varphi)$. This proves the first (left) equivalence in (i); the second, viz., $\pi(M), t' \models \text{PLTL}(t', \varphi)$ if and only $\pi(M), 0 \models \mathsf{X}^{t'}\,\text{PLTL}(t', \varphi)$, is obvious.

As for (ii), using the formula $end(t) = \mathsf{X}^t\,\neg\mathbf{u} \wedge \mathsf{X}^{t+1}\,\mathbf{u}$, we can naively transform φ to

$$\textstyle\bigvee_{t=0}^{\infty}(end(t) \wedge \text{PLTL}(t, \varphi)), \tag{24}$$

which is not an LTL-formula. As all $\text{PLTL}(t, \varphi)$ for $t \geq t_0$ are identical, we obtain

$$\text{PLTL}(\varphi) = \bigvee_{t=0}^{t_0-1}(end(t) \wedge \text{PLTL}(t, \varphi)) \vee \mathsf{X}^{t_0}\, \mathsf{F}(end(t_0) \wedge \text{PLTL}(t_0, \varphi)), \tag{25}$$

where $end = \neg\mathbf{u} \wedge \mathsf{X}^{\mathbf{u}}$. It then follows that $M, S, t \Vdash \varphi$ iff $\pi(M), 0 \models \text{PLTL}(\varphi)$; this proves the result. ∎

Proof of Theorem 15. To show that LARS programs can express only regular languages, we first note that any propositional LARS formula φ in which only windows $\boxplus^{i,j}$ occur is first-order expressible, i.e., that there is a monadic first-order formula $\Phi_\varphi(x)$ such that for any structure $M = \langle S, W, \emptyset\rangle$, $S = (T, \upsilon)$ and $t \in T$, we have $M, S, t \Vdash \varphi$ iff $S \models \Phi(t)$; this formula can be built inductively, using variables to access the window range, and by taking the distance of the time point t to the start and end of S, respectively, into account.

Furthermore, it is well-known that answer set existence for a logic program P can be expressed by a sentence $\Psi(P)$ in second-order logic, see e.g. (Eiter et al., 1997; Baral, 2003). Informally, that sentence says that there exists an interpretation I of the predicates in P such that (i) I is a model of P, and (ii) there is no interpretation $I' \subset I$ such that I' is a model of P^I, where P^I is the reduct of P w.r.t. I. Notably, $\Psi(P)$ is modular w.r.t. rules and extensional data, i.e., $\Psi(P) = \bigcup_{r\in P}(\Psi(\{r\}))$ and for facts D over external predicates not occurring in rule heads of P, $\Psi(P \cup D)$ holds iff $M(D) \models \Psi(P)$, where M is the rendering of D as a FO-structure. Moreover, if all predicates in P have arity at most one, then $\Psi(P)$ amounts to a monadic second-order (MSO) formula. One can easily extend the translation Ψ from ordinary logic programs P to programs where the rule constituents are arbitrary FO-formulas rather than atoms and literals, i.e., with rules of the form

$$\Phi_0(\mathbf{x}_0) \leftarrow \Phi_1(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_n),$$

where the $\Phi_i(\mathbf{x}_i)$ are FO-formulas, while preserving the monadic arity bound. It follows that over streams $S = (T, \upsilon)$, $T = [0, t]$, which represent strings $\upsilon(0)\upsilon(1)\cdots\upsilon(t)$ over the alphabet $\Sigma = 2^{\mathcal{G}^{\mathcal{E}}(P)}$, answer stream existence of P for S at t can be expressed in MSO; hence by the Büchi-Elgot-Trakhtenbrot Theorem (Büchi, 1960b, 1960a; Elgot, 1961; Trakhtenbrot, 1961) propositional LARS programs can express only regular languages.

Conversely, suppose $A = (\Sigma, Q, \delta, q_o, F)$ is a deterministic finite state automaton (FSA),[15] where Σ is the alphabet, Q is the set of states, $\delta \subseteq Q\times\Sigma\times Q$ is the (w.l.o.g. total) transition function, q_0 is the initial state, and F is the set of final states, that accepts a regular language $\mathcal{L}_A \subseteq \Sigma^*$ such that $\epsilon \notin \mathcal{L}_A$. We encode A in a propositional LARS program P_A with external atoms $\mathcal{G}^{\mathcal{E}}(P)$ such that A accepts a string $\sigma_0\sigma_1\cdots\sigma_t$ from Σ^* iff P_A has an answer set for $M = \langle S, W, \emptyset\rangle$ at position t, where $S = (T, \upsilon)$, $T = [0, t]$, and $\upsilon(t') = \{\sigma_{t'}\}$, for $t' = 0, \ldots, t$. Furthermore, by design P_A has some answer stream for S at t only if S is a valid string encoding (i.e., $|\upsilon(t)| = 1$ for all time points t in S).

To this end, we use as intensional atoms the states Q, a further atom p to distinguish, in combination with a window operator, neighbored positions in a string, and an atoms s to mark the beginning of the stream.

[15] The encoding can be readily extended to nondeterministic FSA without exponential blowup, but we refrain from this here.

We then set up the following rules for program P_A:

$$\bot \leftarrow \Box(\textstyle\bigwedge_{\sigma\in\mathcal{G}^{\mathcal{E}}(P)} \neg\sigma \vee \bigvee_{\sigma\neq\sigma'\in\mathcal{G}}(\sigma \wedge \sigma')) \tag{26}$$

$$@_0\,(q_0 \wedge p \wedge s) \leftarrow \top \tag{27}$$

$$@_1\neg s \wedge \Box(\boxplus^{1,0}\Diamond\neg s \rightarrow \neg s) \leftarrow @_1\top \tag{28}$$

$$\Box(\neg s \rightarrow (\boxplus^{1,0}\Diamond p \wedge \boxplus^{1,0}\Diamond\neg p)) \leftarrow @_1\top \tag{29}$$

$$\Box\begin{pmatrix}(p \wedge \boxplus^{1,0}\Diamond(\neg p \wedge q \wedge \sigma) \rightarrow q')\\ \wedge(\neg p \wedge \boxplus^{1,0}\Diamond(p \wedge q \wedge \sigma) \rightarrow q')\end{pmatrix} \leftarrow \top \qquad \text{for } (q,\sigma,q') \in \delta \tag{30}$$

$$\bot \leftarrow q,\ \sigma \quad \text{for } (q,\sigma,q') \in \delta, q' \notin F \tag{31}$$

Intuitively, the first rule is a constraint which checks that at each position exactly one letter from $\mathcal{G}$ occurs, i.e., that the input stream encodes a word over $\mathcal{G}$. We assign to the first position q_0, p, and s by (27). The rule (28) ensures that s can be only at the first position; it is effective if the input string has at least two letters (expressed by $@_1\top$, i.e., position 1 is within the stream). In that case, s can not be at second position 1 (ensured by the left conjunct), and then it can not be at any other position (right conjunct). The rule (29) propagates in alternation p and $\neg p$ from position 0 through the stream. The rule (30) checks the transitions of the automaton at the non-final positions, where p and $\neg p$ are used to distinguish the current and the predecessor position. Finally, the rules (31) check the transition at the last position of the input: if it does not lead to acceptance, no model (and thus answer stream) is possible.

For a non-void S, it can be shown that P_A has an answer stream for $M = \langle I, W, \emptyset\rangle$ for S at t iff S correctly encodes a string $\sigma_0\sigma_1\cdots\sigma_t$ and A accepts S. While (26) ensures the correct encoding, the capturing of acceptance can be argued as follows. By induction on the formula (27) for the base and (28), (29) for the induction step, we can see that in any model I of P for S at t, s occurs only at position 0 and p only at each even position 0, 2, etc. Furthermore, again by induction using (27) and (30) that up to position t', $0 \leq t' \leq t$, we have in I at position t' the state q in which the automaton A is after reading the prefix $\sigma_1\sigma_2\ldots\sigma_{t'-1}$, $0 \leq t' \leq t$ of the string encoded by S; notice that supportedness and minimality of answer streams (Theorems 1 and 2) imply that I has at each position t' a unique state q'. Rule (31) checks then whether reading the last letter σ_t leads to acceptance.

Thus, all regular languages $\mathcal{L}$ modulo the empty string ϵ can be expressed; the latter could be easily handled, if the empty steam would be allowed, using the formula $\Box(\bot)$ to recognize it (for $S = (T, \upsilon)$, we have $M, S, 0 \Vdash \Box(\bot)$ iff $T = \emptyset$).

On the other hand, the above encoding uses the @-operator. With a little more effort and exploiting answer set minimality, we can eliminate its use. Roughly speaking, the idea is to ensure the presence of s at position 0, which then by (28) can not occur elsewhere. We then can use s to recognize the first position, i.e., to emulate $@_0$, and replace (27) with

$$\Box(s \rightarrow q_0 \wedge p) \leftarrow \top; \tag{32}$$

furthermore, we can then replace $@_1\top$ everywhere by $\Diamond\neg s$ and remove $@_1\neg s$ from (28). Suppose we make these changes and set up the rule

$$\Diamond s \leftarrow \top\,. \tag{33}$$

Then in combination with the modified rule (28),

$$\Box(\boxplus^{1,0}\Diamond\neg s \rightarrow \neg s) \leftarrow \Diamond\neg s, \tag{34}$$

we obtain that in every model I for S at t, the occurrences of s form an initial segment of the stream. In order to make sure that this initial segment is minimal, i.e., consists only of position 0, we use a new atom s' and state that s' must also form an initial segment in every model I, and moreover one that is contained in the segment of s:

$$\Diamond s' \leftarrow \top \tag{35}$$

$$\Box(\boxplus^{1,0}\Diamond\neg s' \rightarrow \neg s') \leftarrow \Diamond\neg s \tag{36}$$

$$\Box(s' \rightarrow s) \leftarrow \top. \tag{37}$$

To ensure the minimality of the segment for s, we add a constraint which excludes that the segment for s' is a proper subsegment:

$$\bot \leftarrow \Diamond(s \wedge \neg s'). \tag{38}$$

As s' occurs only in the rules (35)–(38), minimality and supportedness of answer streams (Theorems 1 and 2) imply that in any answer stream I of the resulting program P'_A for S at t, the atom s' (and thus also s) must be present at position 0 and only there.

Formally, it can be shown that P'_A has an answer stream for $S = (T, \upsilon)$, $T = [0, t]$, at t iff S correctly encodes a string $\sigma_0\sigma_1 \cdots \sigma_t$ from Σ^* and A accepts $\sigma_0\sigma_1 \cdots \sigma_t$. This concludes the proof of the result. ∎

For an example of a regular language that is not expressible in LTL but definable by some LARS program in $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ that has only intensional atoms, consider $\mathcal{L} = a(aa)^*$ over $\Sigma = \{a\}$, i.e., strings of a's with odd length. The following program P defines $\mathcal{L}$:

$$\Box(\boxplus^{0,1}(\Diamond a \wedge \Diamond\neg a) \vee \boxplus^{1,0}(\Diamond a \wedge \Diamond\neg a) \vee a) \leftarrow \tag{39}$$

$$\Box a \leftarrow \Diamond\, \boxplus^{0,1}\, \Box\neg a \vee \Diamond\, \boxplus^{1,0}\, \Box a \tag{40}$$

$$a \leftarrow \tag{41}$$

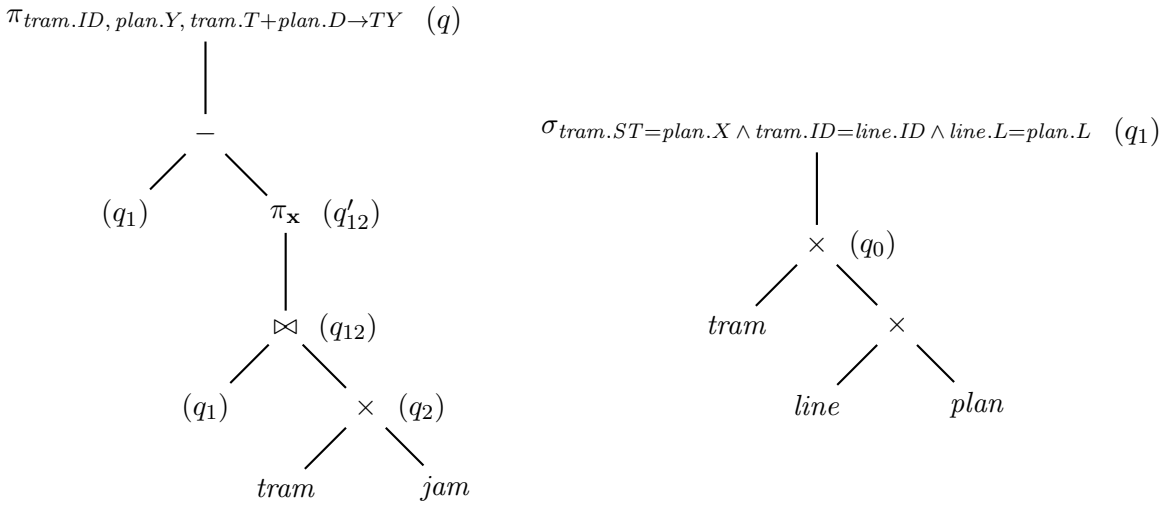
$$\bot \leftarrow \neg\Box a \tag{42}$$

Informally, for a timeline $T = [0, n]$, $n \geq 0$, the string $(\{a\})^{n+1}$ is an answer stream of P at $t = n$, if and only if n is even. To see this, clearly the corresponding stream $S = (T, \{i \mapsto \{a\} \mid 0 \leq i \leq n\})$ induces a model $M = \langle S, W, B\rangle$ of P; the constraint (42) ensures that this M is the only answer stream candidate for this timeline. The reduct $P^{M,t}$ contains all rules of P except (42). By the rule (41), a smaller model M' of $P^{M,t}$, requires $n > 0$, and by the rule (40) starting from position $i = n$, $n - 1$ down to 0 we must have at position i in alternation a, $\neg a$, a etc. and $\neg a$ at position 0 (which means $|T|$ is even, i.e.,n is odd); the rule (39) provides the necessary support for the alternating presence of a.

Note that the program P fits both alphabet settings (a) $\Sigma = 2^{\mathcal{A}}$ and (b) $\Sigma = \{\{\sigma\} \mid \sigma \in \mathcal{A}\}$. To establish that in the setting (b), $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ is strictly more expressive than the class of LARS $\boxplus^{i,j}$ formulas, it is then sufficient to observe that any language $\mathcal{L}$ over Σ defined by a LARS $\boxplus^{i,j}$ formula φ is also defined by the LARS program $P_\varphi = \{\bigvee_{\{\sigma\}\in\Sigma} \sigma \leftarrow;\ \ \bot \leftarrow \neg\varphi\}$. To establish that in setting (a), $\mathcal{P}_{\mathbb{N},\mathbb{N}}$ and the class of LARS $\boxplus^{i,j}$ formulas are incomparable, just consider the language $\mathcal{L} = \{\{a\}, \{\}\}$: by the minimality condition, not both $S = ([0, 0], \{0 \mapsto \{\}\})$ and $S = ([0, 0], \{0 \mapsto \{a\}\})$ can be answer streams of a LARS program defining $\mathcal{L}$, but $\mathcal{L}$ is definable by the LARS formula $\neg@_1(a \vee \neg a)$.

C.2 Continuous Query Language (CQL)

In this section, we give additional details on the translation from CQL to Datalog, resp. LARS, and the presented CQL query q of Example 23. First, applying rel on q gives us the following SQL query:



(a) Main tree (b) Sub tree for (q_1)

Figure 10: Relational algebra expression in tree representation. In projection node (q'_{12}) $\mathbf{x}$ is $tram.ID$, $tram.ST$, $tram.T$, $line.ID$, $line.L$, $plan.L$, $plan.X$, $plan.Y$, $plan.D$.

```
SELECT ID, plan.Y, TY
FROM  tram_part_ID_rows_1 , line , plan
WHERE tram_part_ID_rows_1.ID=line.ID AND
      line.L=plan.L AND
      tram_part_ID_rows_1.ST=plan.X AND
      TY=tram_part_ID_rows_1.T+plan.D AND
      NOT EXISTS
        (SELECT * FROM jam_range_20
         WHERE jam_range_20.ST=tram_part_ID_rows_1.ST)
```

Translating this CQL query to relational algebra according to (Dadashzadeh & Stemple, 1990), we get the following relational algebra expression $RelAlg(rel(q))$, where $tram$ and jam abbreviate relation names $tram\_part\_\boldsymbol{ID}\_rows\_\boldsymbol{1}$ and $jam\_range\_\boldsymbol{20}$, respectively.

$$q = \pi_{tram.ID,\, plan.Y,\, tram.T+plan.D \to TY}(q_1 - q'_{12}),$$

where

$$\begin{aligned}
q_1 &= \sigma_{tram.ST=plan.X \,\wedge\, tram.ID=line.ID \,\wedge\, line.L=plan.L}\; q_0\,,\\
q_0 &= tram \times line \times plan\,,\\
q'_{12} &= \pi_{tram.ID,tram.ST,tram.T,line.ID,line.L,plan.L,plan.X,plan.Y,plan.D}\; q_{12}\,,\\
q_{12} &= q_1 \bowtie q_2\,,\\
q_2 &= tram \times jam\,.
\end{aligned}$$

Figure 10 shows the syntactic expression tree. Note that different such translations might be considered, e.g., by considering other orders in cross products, or joining earlier, etc. However, no semantic differences arise from such optimizations and thus no further discussion is needed for our purposes.

The translated Datalog program $\Delta_D(q) = Dat(RelAlg(rel(q)))$, using the translation in (Garcia-Molina et al., 2009), is the one in Example 24.

Without loss of generality, i.e., due to possible renamings, we assume in the sequel that relation names $\mathtt{B}_i$ and stream names $\mathtt{S}_i$ are pairwise distinct.

Proof of Theorem 16. Before going into the details, let us note that the relational algebra expression is tree-shaped and thus $\Delta_D(Q)$ is an acyclic program (i.e., there is not cyclic recursion through rules). Furthermore, Dat only creates atomic rule heads; therefore $\Delta_D(Q)$ is also definite (i.e., each rule head consists of an atom). The translation $\Delta_L(Q)$ only adds a stratified layer to $\Delta_D(Q)$, i.e., the snapshot rules. Thus, both translations $\Delta_D(Q)$ and $\Delta_L(Q)$ amount to stratified theories and have a unique answer set resp. answer stream relative to given input data.

A correspondence between the CQL results $cqlRes(Q,t)$ and the answer set of $\Delta_L(Q)$ at time t obtained by the following steps.

(1) First, we construct the input for the translated Datalog program, i.e., the atoms reflecting the static relations and those obtained from snapshots. Correctness and completeness of the (compound) translation – from SQL to Relational Algebra and from the latter to Datalog – establishes a correspondence between the unique answer set of the Datalog program and the results of the CQL queries (Lemma 2).

(2) Lemma 3 shows that these atoms need not be provided as such, but can be derived by snapshot rules in LARS itself.

(3) Lemma 4 guarantees that the unique answer set of the Datalog encoding $\Delta_D(Q)$, given snapshot relations as input, corresponds with the unique answer stream of the LARS encoding $\Delta_L(Q)$, given the stream as input.

(4) By combining these lemmas, we obtain the desired correspondence between the results of CQL queries and the answer stream of respective LARS programs.

More formally, let Q be a set of CQL queries to be evaluated on static relations $\mathtt{B} = \mathtt{B}_1, \ldots, \mathtt{B}_m$ and input streams $\mathtt{S} = \mathtt{S}_1, \ldots, \mathtt{S}_n$ at a time point t. Without loss of generality, assume that to each input stream $\mathtt{S}_i$ only one of the CQL window functions in the first column of Table 3 (with window parameters replaced by values) is applied. We denote by $\mathtt{WINDOW}_i$ the CQL window function applied on stream $\mathtt{S}_i$ and by $\mathtt{WINDOW}_i(\mathtt{S}_i, t)$ the snapshot obtained for $\mathtt{S}_i$ at time t after the S2R operator. That is, $\mathtt{WINDOW}_i(\mathtt{S}_i, t)$ contains the selected tuples. (In case there is the need to apply two different windows on the same input stream, one can equivalently take a renamed copy of the stream.) The following defines an according set of input facts for the Datalog program $\Delta_D(Q)$ at t:

$$F(\mathtt{B}, \mathtt{S}, t) = \Delta(\mathtt{B}) \cup \{rel(\mathtt{S}_i)(\mathbf{c}) \mid \mathbf{c} \in \mathtt{WINDOW}_i(\mathtt{S}_i, t)\}$$

where $\Delta(\mathtt{B}) = \{b_i(\mathbf{c}) \mid \mathbf{c} \in \mathtt{B}_i\}$ as before. That is, $rel(\mathtt{S}_i)(\mathbf{c})$ is an atom corresponding to tuple $\mathbf{c}$ from the (snapshot of) stream $\mathtt{S}_i$. For any query $q \in Q$, let $\hat{q}$ denote the head predicate of the rule in $\Delta_D(q)$ corresponding to the root of the relational algebra expression for q. The following lemma establishes the correspondence between the answer set of $\Delta_D(Q)$ and CQL results $cqlRes(Q,t)$ for Q at t; it follows from the correctness and completeness of $RelAlg$ and Dat.

Lemma 2. Let A be the unique answer set of $\Delta_D(Q) \cup F(\mathtt{B}, \mathtt{S}, t)$, i.e., the translated Datalog program plus input facts as obtained from static relations and snapshot relations at time t. Then, for every query $q \in Q$, $\hat{q}(\mathbf{c}) \in A$ iff $\mathbf{c} \in cqlRes(Q,t)$.

The next lemma states that the snapshot semantics of CQL's S2R operator is faithfully captured by LARS formulas as given in Table 3.

Lemma 3. Assume static relations $\mathtt{B} = \mathtt{B}_1, \ldots, \mathtt{B}_m$ and streams $\mathtt{S} = \mathtt{S}_1, \ldots, \mathtt{S}_n$, LARS window functions W corresponding to those in CQL queries Q, and let $M = \langle S, W, \Delta(\mathtt{B})\rangle$ be a structure such that $\Delta(\mathtt{S}) \subseteq S$. Moreover, let w_i be the LARS window function corresponding to $\mathtt{WINDOW}_i$ due to Table 3. Then, $\mathbf{c} \in \mathtt{WINDOW}_i(\mathtt{S}_i, t)$ iff $M, \Delta(\mathtt{S}), t \Vdash \boxplus^{w_i} \Diamond s(\mathbf{c})$.

Proof. Consider an element $\langle \mathbf{c}, t'\rangle$ in stream $\mathtt{S}_i$ which corresponds to the inclusion $s(\mathbf{c}) \in \upsilon(t')$ in $\Delta(\mathtt{S}_i) = (T, \upsilon)$. That is, we can view LARS streams as notational variant of CQL streams, and vice versa. It follows directly from their definitions that window functions $\mathtt{WINDOW}_i$ and w_i (due to Table 3) select the same elements. Thus, $\langle \mathbf{c}, t'\rangle$ is in the CQL window iff it is in the LARS window. The appearance time t' is abstracted away in CQL by the S2R operator, which amounts to existential quantification with $\Diamond$ in LARS. Thus, $\mathbf{c} \in \mathtt{WINDOW}_i(\mathtt{S}_i, t)$ iff $\boxplus^{w_i} \Diamond s(\mathbf{c})$. □

We now establish correspondence between the encodings $\Delta_D(Q)$ and $\Delta_L(Q)$ at a time point t.

Lemma 4. Let Q be a set of CQL queries, $\mathtt{B} = \mathtt{B}_1, \ldots, \mathtt{B}_m$ be static relations, $\mathtt{S} = \mathtt{S}_1, \ldots, \mathtt{S}_n$ be input streams, and let t be a time point. Moreover, let A be the unique answer set of $\Delta_D(Q) \cup F(\mathtt{B}, \mathtt{S}, t)$ and $I = (T, \upsilon)$ the unique answer stream of $\Delta_L(Q)$ for $\Delta(\mathtt{S})$ at t (using structure $M = \langle I, W, \Delta(\mathtt{B})\rangle$). Then, for all $q \in Q$, $\hat{q}(\mathbf{c}) \in A$ iff $\hat{q}(\mathbf{c}) \in \upsilon(t)$.

Proof. From Lemma 3 we obtain that snapshot relations $rel(s)$ can be derived directly from the input stream $\Delta(\mathtt{S})$, using snapshot rules of form (14), instead of providing them explicitly. That is, the LARS subprogram $\Delta_L(Q) \setminus \Delta_D(Q)$ essentially computes $F(\mathtt{B}, \mathtt{S}, t)$ and associates these snapshot atoms of form $rel(s)(\mathbf{c})$ with time point t in the answer stream $I = (T, \upsilon)$: if an atom $s(\mathbf{c})$ is contained in the window, $\boxplus^{w_i} \Diamond s(\mathbf{c})$ holds and in order for the snapshot rule to be satisfied, $rel(s)(\mathbf{c})$ must be contained in $\upsilon(t)$. Due to the minimality and supportedness of I, $rel(s)(\mathbf{c})$ is contained in $\upsilon(t)$ *only* in this case. Moreover, no time point $t' \neq t$ will be assigned with any snapshot atom $rel(s)(\mathbf{c})$ (due to the form of snapshot rules). The snapshot atoms occur as rule heads only in snapshot rules and no rules other than snapshot rules distinguish $\Delta_D(Q)$ and $\Delta_L(Q)$. We thus conclude for any element $\langle \mathbf{c}, t\rangle$ that will be selected by $\mathtt{WINDOW}_i$, $\mathbf{c} \in F(\mathtt{S}, \mathtt{B}, t)$ by definition, and $s_i(\mathbf{c}) \in \upsilon(t)$ as argued. As the semantics after the S2R operator is captured by $\Delta_D(Q)$, we obtain the desired correspondence; in particular for output predicate $\hat{q}$ that $\hat{q}(\mathbf{c})$ is in the answer set of the Datalog encoding iff it is in $\upsilon(t)$ of the answer stream of the LARS encoding. □

References

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases*, Vol. 8. Addison-Wesley, Reading.

Aguado, F., Cabalar, P., Diéguez, M., Pérez, G., & Vidal, C. (2013). Temporal equilibrium logic: a survey. *Journal of Applied Non-Classical Logics*, *23*(1-2), 2–24.

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

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

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

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

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

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., & Widom, J. (2003a). STREAM: the stanford stream data manager. *IEEE Data Eng. Bull.*, *26*(1), 19–26.

Arasu, A., Babu, S., & Widom, J. (2003b). CQL: A language for continuous queries over streams and relations. In Lausen, G., & Suciu, D. (Eds.), *Database Programming Languages, 9th International Workshop, DBPL 2003, Potsdam, Germany, September 6-8, 2003, Revised Papers*, Vol. 2921 of *Lecture Notes in Computer Science*, pp. 1–19. Springer.

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

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

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

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

Bazoobandi, H. R., Beck, H., & Urbani, J. (2017). Expressive stream reasoning with laser. In *ISWC Oct. 21-25, 2017, Vienna, Austria*. To appear.

Beck, H. (2017). Reviewing Justification-based Truth Maintenance Systems from a Logic Programming Perspective. Tech. rep. INFSYS RR-1843-17-02, Institute of Information Systems, TU Vienna.

Beck, H., Bierbaumer, B., Dao-Tran, M., Eiter, T., Hellwagner, H., & Schekotihin, K. (2016). Rule-based Stream Reasoning for Intelligent Administration of Content-Centric Networks. In *15th European Conference on Logics in Artificial Intelligence (JELIA), November 9-11, 2016, Larnaca, Cyprus*.

Beck, H., Bierbaumer, B., Dao-Tran, M., Eiter, T., Hellwagner, H., & Schekotihin, K. (2017). Stream reasoning-based control of caching strategies in ccn routers. In *Proceedings of the IEEE International Conference on Communications, May 21-25, 2017, Paris, France*.

Beck, H., Dao-Tran, M., & Eiter, T. (2015). Answer Update for Rule-based Stream Reasoning. In *24th International Joint Conference on Artificial Intelligence (IJCAI), July 25-31, 2015, Buenos Aires, Argentinia*.

Beck, H., Dao-Tran, M., & Eiter, T. (2016). Equivalent stream reasoning programs. In Kambhampati, S., & Brewka, G. (Eds.), *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16), July 9-15, 2016, New York, USA*, pp. 929–935. AAAI Press/IJCAI.

Beck, H., Dao-Tran, M., Eiter, T., & Fink, M. (2015). LARS: A logic-based framework for analyzing reasoning over streams. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, January 25-29, 2015, Austin, Texas, USA*.

Beck, H., Eiter, T., & Folie, C. (2017). Ticker: A system for incremental ASP-based stream reasoning. *Theory and Practice of Logic Programming*, *XX*. Special issue on ICLP 2017. To appear.

Brandt, S., Güzel Kalaycı, E., Kontchakov, R., Ryzhikov, V., Xiao, G., & Zakharyaschev, M. (2017). Ontology-based data access with a horn fragment of metric temporal logic. In Singh, S. P., & Markovitch, S. (Eds.), *Proceedings 31st Conference on Artificial Intelligence (AAAI '17)*, pp. 1070–1076. AAAI Press.

Brewka, G., & Eiter, T. (2007). Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings 22nd Conference on Artificial Intelligence (AAAI '07), July 22-26, 2007, Vancouver*, pp. 385–390. AAAI Press.

Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Communications of the ACM*, *54*(12), 92–103. doi¿10.1145/2043174.2043195.

Brewka, G., Eiter, T., & Truszczyński, M. (Eds.). (2016). *AI Magazine: special issue on Answer Set Programming*. AAAI Press. Volume 37, number 3 (Fall issue).

Büchi, J. R. (1960a). On a Decision Method in Restriced Second-Order Arithmetic. In Nagel et al, E. (Ed.), *Proc. International Congress on Logic, Methodology and Philosophy of Science*, pp. 1–11, Stanford, CA. Stanford University Press.

Büchi, J. R. (1960b). Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, *6*, 66–92.

Cabalar, P., & Vega, G. P. (2007). Temporal equilibrium logic: A first approach. In Moreno-Díaz, R., Pichler, F., & Quesada-Arencibia, A. (Eds.), *Computer Aided Systems Theory - EUROCAST 2007, 11th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 12-16, 2007, Revised Selected Papers*, Vol. 4739 of *Lecture Notes in Computer Science*, pp. 241–248. Springer.

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

Calvanese, D., Kalayci, E. G., Ryzhikov, V., & Xiao, G. (2016). Towards practical OBDA with temporal ontologies - (position paper). In Ortiz, M., & Schlobach, S. (Eds.), *Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings*, Vol. 9898 of *Lecture Notes in Computer Science*, pp. 18–24. Springer.

Ceri, S., Gottlob, G., & Tanca, L. (1990). *Logic Programming and Databases*. Springer.

Chandra, A., & Merlin, P. (1977). Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth ACM Symposium on the Theory of Computing (STOC-77)*, pp. 77–89, Boulder, Colorado.

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

Dao-Tran, M., Beck, H., & Eiter, T. (2015a). Contrasting rdf stream processing semantics. In *5th Joint International Semantic Technology Conference (JIST), November 11-13, 2015, YiChang, China*.

Dao-Tran, M., Beck, H., & Eiter, T. (2015b). Towards Comparing RDF Stream Processing Semantics. In *HiDeSt*.

Dao-Tran, M., & Eiter, T. (2017). Streaming multi-context systems. In Sierra, C., & Bacchus, F. (Eds.), *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17), August 19-25, 2017, Melbourne, Australia*. AAAI Press/IJCAI. To appear.

de Leng, D., & Heintz, F. (2016). Qualitative spatio-temporal stream reasoning with unobservable intertemporal spatial relations using landmarks. In Schuurmans, D., & Wellman, M. P. (Eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 957–963. AAAI Press.

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

Dell'Aglio, D., Della Valle, E., van Harmelen, F., & Bernstein, A. (2017). Stream reasoning: A survey and outlook. *Data Science*, *x*(x). To appear. DOI: 10.3233/DS-170006.

Demri, S., & Schnoebelen, P. (2002). The complexity of propositional linear temporal logics in simple cases. *Inf. Comput.*, *174*(1), 84–103.

Dindar, N., Tatbul, N., Miller, R. J., Haas, L. M., & 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., & Liu, F. (2011). Answer set programming for stream reasoning. In *AI*, pp. 104–109.

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

Doyle, J. (1979). A Truth Maintenance System. *Artif. Intell.*, *12*(3), 231–272.

Eiter, T., Faber, W., Fink, M., & Woltran, S. (2007). Complexity results for answer set programming with bounded predicate arities. *Annals of Mathematics and Artificial Intelligence*, *51*(2-4), 123–165.

Eiter, T., & Gottlob, G. (1995). On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, *15*(3/4), 289–323.

Eiter, T., & Fink, M. (2003). Uniform equivalence of logic programs under the stable model semantics. In *ICLP*.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2014). Domain Expansion for ASP-Programs with External Sources. *Artif. Intell.*, *233*, 84–121.

Eiter, T., Gottlob, G., & Mannila, H. (1997). Disjunctive Datalog. *ACM Transactions on Database Systems*, *22*(3), 364–417.

Eiter, T., Ianni, G., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2016). A model building framework for ASP with external computations. *Theory and Practice of Logic Programming*, *16*(4), 418–464. Available on CJO 2015 doi:10.1017/S1471068415000113, http://arxiv.org/abs/1507.01451.

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

Elgot, C. C. (1961). Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, *98*, 21–51.

Elkan, C. (1990). A rational reconstruction of nonmonotonic truth maintenance systems. *Artif. Intell.*, *43*(2), 219–234.

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of answer set programming. *AI Magazine*, *37*(3), 53–68.

Etessami, K., Vardi, M. Y., & Wilke, T. (2002). First-order logic with two variables and unary temporal logic. *Inf. Comput.*, *179*(2), 279–295.

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

Gabbay, D. M. (1987). The declarative past and imperative future: Executable temporal logic for interactive systems. In *Temporal Logic in Specification*, pp. 409–448, London, UK, UK. Springer-Verlag.

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

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Thiele, S. (2008). Engineering an incremental ASP solver. In Garcia de la Banda, M., & Pontelli, E. (Eds.), *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, Vol. 5366 of *Lecture Notes in Computer Science*, pp. 190–205. Springer-Verlag.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2014). *Clingo* = ASP + control: Preliminary report. In Leuschel, M., & Schrijvers, T. (Eds.), *Technical Communications of the Thirtieth International Conference on Logic Programming (ICLP'14)*, Vol. 14(4-5). Theory and Practice of Logic Programming, Online Supplement.

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

Gebser, M., Grote, T., Kaminski, R., & Schaub, T. (2011). Reactive answer set programming. In *LPNMR*, pp. 54–66.

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2017). Multi-shot ASP solving with clingo. *CoRR*, *abs/1705.09811*.

Gebser, M., Kaminski, R., Obermeier, P., & Schaub, T. (2015). Ricochet robots reloaded: A case-study in multi-shot ASP solving. In Eiter, T., Strass, H., Truszczynski, M., & Woltran, S. (Eds.), *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday*, Vol. 9060 of *Lecture Notes in Computer Science*, pp. 17–32. Springer.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Gupta, A., Mumick, I. S., & Subrahmanian, V. S. (1993). Maintaining views incrementally. In Buneman, P., & Jajodia, S. (Eds.), *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pp. 157–166. ACM Press.

Gurevich, Y. (1988). Logic and the Challenge of Computer Science. In Börger, E. (Ed.), *Trends in Theoretical Computer Science*, chap. 1. Computer Science Press.

Heintz, F., & Doherty, P. (2004). Dyknow: An approach to middleware for knowledge processing. *Journal of Intelligent and Fuzzy Systems*, *15*(1), 3–13.

Heyting, A. (1930). Die formalen Regeln der intuitionistischen Logik. In *Sitzungsberichte der preußischen Akademie der Wissenschaften. phys.-math. Klasse*, pp. 4265, 57–71, 158–169.

Kontchakov, R., Pandolfo, L., Pulina, L., Ryzhikov, V., & Zakharyaschev, M. (2016). Temporal and spatial OBDA with many-dimensional Halpern-Shoham logic. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 1160–1166. IJCAI/AAAI Press.

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

Laroussinie, F., Markey, N., & Schnoebelen, P. (2002). Temporal logic with forgettable past. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pp. 383–392. IEEE Computer Society.

Lifschitz, V., Pearce, D., & Valverde, A. (2001). Strongly equivalent logic programs. *ACM Trans. Comput. Log.*, *2*(4), 526–541.

Markey, N. (2003). Temporal logic with past is exponentially more succinct, concurrency column. *Bulletin of the EATCS*, *79*, 122–128.

Mileo, A., Abdelrahman, A., Policarpio, S., & Hauswirth, M. (2013). Streamrule: A nonmonotonic stream reasoning system for the semantic web. In Faber, W., & Lembo, D. (Eds.), *Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings*, Vol. 7994 of *Lecture Notes in Computer Science*, pp. 247–252. Springer.

Mileo, A., Dao-Tran, M., Eiter, T., & Fink, M. (2017). Stream reasoning. In Liu, L., & Özsu, M. T. (Eds.), *Encyclopedia of Database Systems, 2nd edition*, p. 7 pp. Springer Science+Business Media. DOI10.1007/978-1-4899-7993-3_80715-1.

Motik, B., Nenov, Y., Piro, R., & Horrocks, I. (2015). Incremental Update of Datalog Materialisation: The Backward/Forward Algorithm. In *AAAI*.

Nickles, M., & Mileo, A. (2014). Web stream reasoning using probabilistic answer set programming. In Kontchakov, R., & Mugnier, M. (Eds.), *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*, Vol. 8741 of *Lecture Notes in Computer Science*, pp. 197–205. Springer.

Nickles, M., & Mileo, A. (2015). A hybrid approach to inference in probabilistic non-monotonic logic programming. In Riguzzi, F., & Vennekens, J. (Eds.), *Proceedings of the 2nd International Workshop on Probabilistic Logic Programming co-located with 31st International Conference on Logic Programming (ICLP 2015), Cork, Ireland, August 31st, 2015.*, Vol. 1413 of *CEUR Workshop Proceedings*, pp. 57–68. CEUR-WS.org.

Oikarinen, E., & Janhunen, T. (2006). Modular equivalence for normal logic programs. In Brewka, G., Coradeschi, S., Perini, A., & Traverso, P. (Eds.), *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*, Vol. 141 of *Frontiers in Artificial Intelligence and Applications*, pp. 412–416. IOS Press.

Özcep, Ö. L., Möller, R., & Neuenstadt, C. (2015). Stream-query compilation with ontologies. In Pfahringer, B., & Renz, J. (Eds.), *Proceedings of AI 2015: Advances in Artificial Intelligence. 28th Australasian Joint Conference, Canberra, ACT, Australia, 2015.*, Vol. 9457 of *Lecture Notes in Computer Science*, pp. 457–463. Springer.

Pearce, D. (2006). Equilibrium logic. *Annals of Mathematics and Artificial Intelligence*, *47*(1-2), 3–41.

Phuoc, D. L., Dao-Tran, M., Parreira, J. X., & Hauswirth, M. (2011). A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC (1)*, pp. 370–388.

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

Phuoc, D. L., Nguyen-Mau, H. Q., Parreira, J. X., & Hauswirth, M. (2012b). A middleware framework for scalable management of linked streams. *J. Web Sem.*, *16*, 42–51.

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57. IEEE Computer Society.

Polleres, A. (2007). From SPARQL to rules (and back). In *WWW 2007*, pp. 787–796.

Prior, A. (1967). *Past, Present and Future*. Oxford University Press.

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

Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, *4*(2), 177–192. ISSN 1439-2275.

Sistla, A. P., & Clarke, E. M. (1985). The complexity of propositional linear temporal logics. *J. ACM*, *32*(3), 733–749.

Stephens, R. (1997). A survey of stream processing. *Acta Inf.*, *34*(7), 491–541.

Tiger, M., & Heintz, F. (2016). Stream reasoning using temporal logic and predictive probabilistic state models. In Dyreson, C. E., Hansen, M. R., & Hunsberger, L. (Eds.), *23rd International Symposium on Temporal Representation and Reasoning, TIME 2016, Kongens Lyngby, Denmark, October 17-19, 2016*, pp. 196–205. IEEE Computer Society.

Trakhtenbrot, B. (1961). Finite Automata and the Logic of Monadic Predicates. *Dokl. Akad. Nauk SSSR*, *140*, 326–329.

Woltran, S. (2004). Characterizations for relativized notions of equivalence in answer set programming. In *JELIA*.

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