Chasing Streams with Existential Rules

Jacopo Urbani[1] , **Markus Krötzsch**[2] , **Thomas Eiter**[3]
[1]Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
[2]Knowledge-Based Systems Group, TU Dresden, Germany
[3]Institute of Logic and Computation, Technische Universität Wien, Austria
jacopo@cs.vu.nl,markus.kroetzsch@tu-dresden.de,eiter@kr.tuwien.ac.at

Abstract

We study reasoning with existential rules to perform query answering over streams of data. On static databases, this problem has been widely studied, but its extension to rapidly changing data has not yet been considered. To bridge this gap, we extend LARS, a well-known framework for rule-based stream reasoning, to support existential rules. For that, we show how to translate LARS with existentials into a semantics-preserving set of existential rules. As query answering with such rules is undecidable in general, we describe how to leverage the temporal nature of streams and present suitable notions of acyclicity that ensure decidability.

1 Introduction

Streaming data arises in many applications, fostered by the need of deriving timely insights from emerging information and the inherent impossibility of storing all available data (Margara et al. 2014). Stream reasoning has become a productive area of KR with many formalisms (Anicic et al. 2011; Le-Phuoc et al. 2011; Barbieri et al. 2010; Tiger and Heintz 2016; Dell'Aglio et al. 2017; Kharlamov et al. 2019; Wałega, Kaminski, and Cuenca Grau 2019). This multiplicity is justified by the breadth of scenarios where stream processing is useful. Many of the approaches are distinguished from classical temporal reasoning, e.g., since data snapshots (*windows*) play an important role to reduce data volumes.

A well-known formalism in this space is LARS (Beck, Dao-Tran, and Eiter 2018), which is a rule-based language for stream reasoning that combines concepts from logic programming with dedicated stream operators to express windows and temporal quantifiers. For example, the LARS rule r_1: $\boxplus^3 \Box \mathsf{beltTmp}(X, Y) \wedge \mathsf{high}(Y) \rightarrow \mathsf{warn}(X)$ issues a warning if the temperature on a conveyor belt has been high for all ($\Box$) last three time points ($\boxplus^3$).

Another prominent field in KR are *existential rules*, which are also used as a basis for ontological models, especially in applications with large amounts of data (Baget et al. 2011; Cuenca Grau et al. 2013; Gottlob, Lukasiewicz, and Pieris 2014). Other common names for these rules include *tuple-generating dependencies* (Abiteboul, Hull, and Vianu 1994) and *Datalog*$^\pm$ (Calì et al. 2010). As a simple example, the rule r_2: $\mathsf{belt}(X) \rightarrow \exists Y.\mathsf{beltOperator}(X, Y)$ expresses that every belt has an operator (even if unknown). Existential quantification is central for ontologies and provides high expressivity beyond plain Datalog (Krötzsch, Marx, and Rudolph 2019). While reasoning with existential rules is known to be undecidable in general (Beeri and Vardi 1981), many well-behaved language fragments and practical implementations exist (Benedikt et al. 2017; Urbani et al. 2018; Bellomarini, Sallinger, and Gottlob 2018).

Until now, however, these areas have not been combined, and stream reasoning approaches do not support existential rules. Even for logic-based ontology languages in general, solutions only seem to exist for specific cases where queries are rewritable (Kharlamov et al. 2019; Kalaycı et al. 2019). As a consequence, it is often unclear how existing ontological background knowledge can be used in stream reasoning.

Additionally, the lack of existential quantification prevents useful modelling techniques for stream analysis. In particular, existential quantification can be used to represent temporal events, possibly spanning multiple time points, or to track unknown individuals. For instance, it can be used to create a new incident ID if the temperature on a belt is high for too long, or to track a not-yet-recognized object within a bounding box in a video stream. Notice that while in principle events could be modeled without value invention, i.e., using ad-hoc relations, doing so would put an upper bound to the number of possible events which might be undesirable as the future stream is typically unknown. A similar argument applies to the example above about objects within bounding boxes: it is arguably more natural to introduce new values and treat them as first-class individuals.

With this motivation in mind, we developed an extension of existential rules with LARS-based temporal quantifiers called LARS$^+$. Due to the undecidability of query answering with existential rules, our objective are decidable fragments, with the following contributions:

- We introduce LARS$^+$ as an existential stream reasoning language with a model-theoretic semantics.
- We give a semantics-preserving transformation from LARS$^+$ to existential rules to allow query answering. Doing so allows us to exploit existing decidability results, but these are limited in their use of time. We thus present *time-aware extensions of acyclicity notions* for LARS$^+$ programs.
- Initial experiments suggest that our method is promising.[1]

[1]Source code is at https://github.com/karmaresearch/elars; this is an extended version of the eponymous KR'22 paper.

2 LARS$^+$

Currently, LARS and DatalogMTL (Brandt et al. 2017; Wałega, Kaminski, and Cuenca Grau 2019) are popular for rule-based reasoning on data streams. While we focus on LARS, some of our work may be adapted to DatalogMTL.

To cope with big data volumes, LARS allows one to restrict streams to data snapshots (i.e., substreams) taken by generic window operators $\boxplus$. Typically, windows are used to consider only the knowledge in the most recent past, but this is not enough to avoid a complexity explosion or even undecidability that could arise from reasoning over an indefinite future. To overcome this problem, it is common in this domain to restrict future predictions up to a horizon of interest h, which is moved forward indefinitely.

Our language LARS$^+$ can be viewed as an extension of existential rules with temporal features of LARS. In the choice of data-snapshot operators, we take inspiration from *plain LARS*, which is a LARS fragment that is apt for efficient implementation (Bazoobandi, Beck, and Urbani 2017).

Syntax We consider a two-sorted logic with abstract elements and the natural numbers $\mathbb{N}$ as time points. We assume infinite sets $\mathcal{V}_A$ of *abstract variables*, $\mathcal{V}_T$ of *time variables*, $\mathcal{N}$ of *labelled nulls*, and $\mathcal{C}$ of *constants* that are mutually disjoint and disjoint from $\mathbb{N}$. *Abstract terms* (resp. *time terms*) are elements of $\mathcal{V}_A \cup \mathcal{N} \cup \mathcal{C}$ (resp., $\mathcal{V}_T \cup \mathbb{N}$).

Predicates p are from a set $\mathcal{P}$ of predicates and have arity $\mathsf{ar}(p) \geq 0$, with each position typed (abstract or time sort). A *normal atom* is an expression $p(\mathbf{t})$, $\mathbf{t} = t_1, \ldots, t_{\mathsf{ar}}(p)$, where t_i is a term of proper sort. An *arithmetic atom* has the form $t_1 \leq t_2$ or $t_1 = t_2 + t_3$ for time terms t_1, t_2, t_3. The set of all *atoms* (normal and arithmetic) is denoted $\mathcal{A}$. For an atom α (or any other logical expression introduced below), the *domain* $\mathsf{dom}(\alpha)$ of α is the set of all terms in α; we write $\alpha[\mathbf{x}]$ to state that $\mathbf{x} = \mathsf{dom}(\alpha) \cap (\mathcal{V}_A \cup \mathcal{V}_T)$; and we say that α is *ground* if it contains no variables.

A predicate $p \in \mathcal{P}$ is *simple* if it has no position of time sort, while an atom is *simple* if it normal and has a simple predicate. A *LARS$^+$ atom* α has the form

$$\alpha \coloneqq a \mid b \mid @_T b \mid \boxplus^n @_T b \mid \boxplus^n \Diamond b \mid \boxplus^n \Box b \qquad (1)$$

where a is an arithmetic atom, b is either a null-free simple atom or $\top$ (which holds true at all times), T is a time term, and $n \in \mathbb{N}$. *Window operators* $\boxplus^n$ restrict attention back to n time points in the past, and $@_T$ (resp. $\Box$, $\Diamond$) indicates that a formula holds at time T (resp., *every*, *some* time point).

Arithmetic atoms do not depend on time, whereas atoms $@_T b$ refer to a specific time T. All other LARS$^+$ atoms are interpreted relative to some current time point. Simple atoms b can equivalently be written as $\boxplus^0 \Diamond b$ or as $\boxplus^0 \Box b$.

Definition 1. A *LARS$^+$ rule* is an expression of the form

$$r = \Box \forall \mathbf{x}, \mathbf{y}.(B[\mathbf{x}, \mathbf{y}] \rightarrow \exists \mathbf{v}.H[\mathbf{y}, \mathbf{v}]) \qquad (2)$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{v}$ are mutually disjoint sets of variables, and $\mathbf{v}$ contains only abstract variables; the body $B[\mathbf{x}, \mathbf{y}]$ is a conjunction of LARS$^+$ atoms; and the head $H[\mathbf{y}, \mathbf{v}]$ is a conjunction of atoms of the form b or $@_T b$ in (1). We set $\mathsf{b}(r) \coloneqq B$ and $\mathsf{h}(r) \coloneqq H$, and we usually omit the leading $\Box$ and universal quantifiers when writing rules.

A *LARS$^+$ program* is a finite set of *LARS$^+$ rules*; we denote the set of all such programs by $\mathbf{L}^+$.

Semantics Like for LARS, the semantics of LARS$^+$ is based on streams. Formally, a *stream* $S = (\mathbf{T}, v)$ consists of a timeline $\mathbf{T} = [0, h] \subset \mathbb{N}$ and an *evaluation function* $v : \mathbb{N} \rightarrow 2^{\mathcal{A}}$ such that, for all $t \in \mathbb{N}$, $v(t)$ is a set of ground normal atoms and $v(t) = \emptyset$ if $t \notin \mathbf{T}$. We call S a *data stream* if only extensional atoms occur in S, i.e., atoms with designated predicates not occurring in rule heads. Given $n \in \mathbb{N}$ and $t \in \mathbf{T}$, we write $w_n(S, t)$ for the stream $([0, t], v')$ where for any $t' \in \mathbb{N}$, $v'(t') = v(t')$ if $t - n \leq t' \leq t$, and $v'(t) = \emptyset$ otherwise; we call $w_n(S, t)$ a *window* of size n on S at t.

Models of LARS$^+$ are special streams. For a stream $S = (\mathbf{T}, v)$, a simple ground atom b, and $t, t', n \in \mathbb{N}$, we write:

$$S, t \models b \quad \text{if } b \in v(t), \qquad S, t \models @_{t'} b \quad \text{if } S, t' \models b,$$
$$S, t \models \Diamond b \,/\, \Box b \quad \text{if } S, t'' \models b \text{ for some / all } t'' \in \mathbf{T},$$
$$S, t \models \boxplus^n \beta \qquad \text{if } w_n(S, t), t \models \beta.$$

Further, $S, t \models \top$ holds for all $t \in \mathbf{T}$ and $S, t \models a$ for all ground arithmetic atoms a that express a true relation on $\mathbb{N}$. To define satisfaction of rules on a stream S at time point t, we introduce the auxiliary notion of $\mathbf{T}$-*match* σ for a set C of atoms on S and t as a sort-preserving mapping from the variables of C to terms, s.t. (i) each time variable X is mapped to $\mathbf{T}$ ($X\sigma \in \mathbf{T}$) and (ii) $S, t \models \alpha\sigma$ for each $\alpha \in C$.

Definition 2. A LARS$^+$ rule r as in (2) is *satisfied* by a stream $S = (\mathbf{T}, v)$, written $S \models r$, if either (i) $\mathsf{h}(r)$ contains some time point $t \notin \mathbf{T}$ (i.e., ignore inference out of scope), or (ii) for all $t \in \mathbf{T}$, every $\mathbf{T}$-match σ of $\mathsf{b}(r)$ on S and t is extendible to a $\mathbf{T}$-match $\sigma' \supseteq \sigma$ of $\mathsf{b}(r) \cup \mathsf{h}(r)$ on S and t.

A program $P \in \mathbf{L}^+$ is satisfied by S, written $S \models P$, if $S \models r$ for all $r \in P$. A data stream $D = (\mathbf{T}', v')$ is satisfied by S, written $S \models D$, if $\mathbf{T}' \subseteq \mathbf{T}$ and $v'(t) \subseteq v(t)$ for all $t \in \mathbf{T}'$. We then call S a *model* of P resp. D.

Example 1. Consider the data stream $D = ([0, 9], v)$, where $v(t) = \{\mathsf{belt}(b_1), \mathsf{high}(90), \mathsf{beltTmp}(b_1, tmp(t))\}$ for each $t \in [0, 9]$, where $tmp(t) = 90$ if $t \leq 4$ and $tmp(t) = 70$ otherwise. Then any model S of the rules r_1, r_2 in Section 1 and D fulfills $S, 4 \models \mathsf{warn}(b_1) \land \mathsf{beltOperator}(b_1, v)$ for some constant or null v. Similarly $S, 5 \models \mathsf{beltOperator}(b_1, v')$ for some constant or null v' while $S, 5 \models \mathsf{warn}(b_1)$ may fail.

3 Query Answering with LARS$^+$

The query answering problem in LARS$^+$ is as follows.

Definition 3. A *LARS$^+$ Boolean Conjunctive Query (BCQ)* q has the form $\exists \mathbf{x}.Q[\mathbf{x}]$, where Q is a conjunction of LARS$^+$ atoms. A stream $S = (\mathbf{T}, v)$ satisfies q at time t, written $S, t \models q$, if some $\mathbf{T}$-match σ of Q on S and t exists. A program $P \in \mathbf{L}^+$ and data stream D *entail* q at time t, written $P, D, t \models q$, if $S, t \models q$ for every model S of P and D.

For instance, a BCQ could be $\exists X. \boxplus^5 \Box \mathsf{warn}(X)$, which asks if there has been a warning over the same belt in the last 5 time points. To solve BCQ answering with LARS$^+$, we propose a consequence-preserving rewriting $\mathsf{rew}(\cdot)$ to existential rules with a time sort. This rewriting is useful because it will allow us to exploit known results for existential rules, e.g., acyclicity notions (Cuenca Grau et al. 2013).

Our proposed rewriting of P into $\mathsf{rew}(P)$ has 5 steps:
(1) Each atom $\boxplus^n \Diamond p(\mathbf{t})$ is replaced by $\boxplus^n @_T\, p(\mathbf{t})$, where T is a fresh variable used only in one atom.
(2) For any simple predicate p, we add auxiliary predicates $[\![\boxplus\Box\, p]\!]$ and $[\![\boxplus @\, p]\!]$ of arity $\mathsf{ar}(p){+}2$ resp. $\mathsf{ar}(p){+}3$. Intuitively, $[\![\boxplus\Box\, p]\!](\mathbf{t}, n, C)$ and $[\![\boxplus @\, p]\!](\mathbf{t}, n, T, C)$ mean that $\boxplus^n \Box p(\mathbf{t})$ and $\boxplus^n @_T\, p(\mathbf{t})$ hold at time C, respectively.
(3) Using a fresh variable C to represent the current time, we rewrite non-arithmetic atoms α in P (where $\top$ is $\top()$) to

$$\mathsf{rew}(\alpha) = \begin{cases} [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, C) & \text{if } \alpha = p(\mathbf{t}), \\ [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, T) & \text{if } \alpha = @_T\, p(\mathbf{t}), \\ [\![\boxplus\Box\, p]\!](\mathbf{t}, n, C) & \text{if } \alpha = \boxplus^n \Box p(\mathbf{t}), \\ [\![\boxplus @\, p]\!](\mathbf{t}, n, T, C) & \text{if } \alpha = \boxplus^n @_T\, p(\mathbf{t}) \end{cases}$$

(4) We add $[\![\boxplus\Box\, \top]\!](0, C)$ in rule bodies not containing C.
(5) For every predicate p (including $\top$), we add the following rules to P, where $\mathbf{X}$ is a list of variables of length $\mathsf{ar}(p)$ and $m = \max(0, n \mid \boxplus^n \text{ occurs in } P)$:

$$0 \le C \to [\![\boxplus\Box\, \top]\!](0, C) \tag{3}$$

$$[\![\boxplus\Box\, p]\!](\mathbf{X}, 0, 0) \to [\![\boxplus\Box\, p]\!](\mathbf{X}, m, 0) \tag{4}$$

$$[\![\boxplus\Box\, p]\!](\mathbf{X}, N', C) \wedge N'{=}N{+}1 \to [\![\boxplus\Box\, p]\!](\mathbf{X}, N, C) \tag{5}$$

$$\begin{aligned} &[\![\boxplus\Box\, p]\!](\mathbf{X}, N, C) \wedge N'{=}N{+}1 \wedge N'{\le}m \wedge C'{=}C{+}1 \\ &\quad \wedge [\![\boxplus\Box\, p]\!](\mathbf{X}, 0, C') \to [\![\boxplus\Box\, p]\!](\mathbf{X}, N', C') \end{aligned} \tag{6}$$

$$[\![\boxplus\Box\, p]\!](\mathbf{X}, 0, C) \to [\![\boxplus @\, p]\!](\mathbf{X}, 0, C, C) \tag{7}$$

$$\begin{aligned} &[\![\boxplus @\, p]\!](\mathbf{X}, N, T, C) \wedge N'{\le}m \wedge N'{=}N{+}1 \\ &\quad \wedge I \le 1 \wedge C'{=}C{+}I \to [\![\boxplus @\, p]\!](\mathbf{X}, N', T, C') \end{aligned} \tag{8}$$

We rewrite a LARS$^+$ BCQ $\exists\mathbf{x}.Q$ and time point t similarly to $\mathsf{rew}(\exists\mathbf{x}.Q, t) = \exists\mathbf{x}. \bigwedge_{\alpha\in Q} \mathsf{rew}(\alpha) \wedge C{\le}t \wedge t{\le}C$ (treating atoms $\boxplus^n \Diamond p(\mathbf{t})$ as before), and a stream $S = (\mathbf{T}, v)$ to facts $\mathsf{rew}(S) = \{[\![\boxplus\Box\, p]\!](\mathbf{t}, 0, s) \mid p(\mathbf{t}) \in v(t), t \in \mathbf{T}\}$.

Example 2. We illustrate the rewriting on r_1. Step (2) creates predicates $[\![\boxplus\Box\mathsf{beltTmp}]\!]$, $[\![\boxplus\Box\mathsf{high}]\!]$, and $[\![\boxplus\Box\mathsf{warn}]\!]$ and Step (3) the rule $[\![\boxplus\Box\mathsf{high}]\!](X, Y, 3, C) \wedge [\![\boxplus\Box\mathsf{high}]\!](Y, 0, C) \to [\![\boxplus\Box\mathsf{warn}]\!](X, 0, C)$. Step (5) adds auxiliary rules to implement the semantics; e.g., rule (6) ensures that "$\boxplus\Box$"-facts survive across time points, say if $[\![\boxplus\Box\mathsf{beltTmp}]\!](a, b, 0, 6)$ and $[\![\boxplus\Box\mathsf{beltTmp}]\!](a, b, 2, 5)$ hold, then $[\![\boxplus\Box\mathsf{beltTmp}]\!](a, b, 3, 6)$ should hold as well.

Let us denote by $P' \models_{\mathbf{T}} q'$ entailment of a BCQ q' from existential rules P' with timeline $\mathbf{T}$, which is defined using $\mathbf{T}$-matches as $P, D, t \models q$ but disregarding D and t. Then:

Theorem 1. *For any $P \in \mathbf{L}^+$, BCQ q, data stream D on $\mathbf{T}$, and $t{\in}\mathbf{T}$ holds $P, D, t \models q$ iff $\mathsf{rew}(P) \cup \mathsf{rew}(D) \models_{\mathbf{T}} \mathsf{rew}(q, t)$.*

Theorem 1 is important as it allows us to implement BCQ answering in LARS$^+$ using existential rule engines, e.g., GLog (Tsamoura et al. 2021); arithmetic atoms over $\mathbf{T}$ can be simulated with regular atoms: simply add the set $\mathsf{rew}(\mathbf{T})$ of all true instances of arithmetic atoms in P over $\mathbf{T}$ and view $\mathsf{rew}(P) \cup \mathsf{rew}(D) \cup \mathsf{rew}(\mathbf{T})$ as a single-sorted theory.

4 Decidability

As BCQ entailment over existential rules is undecidable, we desire that the rewriting $\mathsf{rew}(\cdot)$ falls into a known decidable fragment. Such may be defined by *acyclicity conditions* (Cuenca Grau et al. 2013), which ensure that a suitable *chase*, which is a versatile class of reasoning algorithms for existential rules (Benedikt et al. 2017) based on "applying" rules iteratively, will terminate over a given input. We use a variant of the *skolem chase* (Marnette 2009), using nulls instead of skolem terms (aka *semi-oblivious chase*), extended to the time sort .

Conditions like the canonical *weak acyclicity (WA)* (Fagin et al. 2005) ensure in fact *universal termination*, i.e., chase termination for a given rule set over all sets of input facts. We can thus apply such criteria to $\mathsf{rew}(P)$ (viewed as single-sorted theory) while ignoring $\mathsf{rew}(D)$ and $\mathsf{rew}(\mathbf{T})$. Universal termination may here be seen as an analysis that disregards time. To formalise this, let $\mathsf{strip}(P)$ result from P by deleting all arithmetic atoms, window operators, and temporal quantifiers, and let $\mathbf{CT}$ and $\mathbf{WA}$ be the classes of all rule sets on which the skolem chase universally terminates and of all weakly acyclic rule sets, respectively. Then:

Theorem 2. *For any $P \in \mathbf{L}^+$, we have (i) $\mathsf{strip}(P) \in \mathbf{CT}$ iff $\mathsf{rew}(P) \in \mathbf{CT}$ and (ii) $\mathsf{strip}(P) \in \mathbf{WA}$ iff $\mathsf{rew}(P) \in \mathbf{WA}$.*

Analogous results hold for elaborated acyclicity notions (Cuenca Grau et al. 2013). Notably, we can check acyclicity on the simpler rule set $\mathsf{strip}(P)$. With WA as a representative notion, we let $\mathbf{L}^+_{\mathsf{LWA}} = \{P \in \mathbf{L}^+ \mid \mathsf{strip}(P) \in \mathbf{WA}\}$.

While easy to check, universal termination also considers situations that are impossible on properly encoded streams.

Example 3. Consider $P = \{@_T\, p(X, Y) \wedge T'{=}T{+}1 \to \exists V. @_{T'}\, p(Y, V)\}$. The skolem chase on $\mathsf{rew}(P) \cup \mathsf{rew}(D) \cup \mathsf{rew}(\mathbf{T})$ terminates on all $\mathbf{T}$ and D, but not universally for non-standard timelines where e.g., $0 = 0 + 1$ holds. That is, reasoning with P always terminates despite $P \notin \mathbf{L}^+_{\mathsf{LWA}}$.

We thus introduce *time-aware acyclicity*, which retains relevant temporal information instead of working with $\mathsf{strip}(P)$ only. First, to simplify P, we fix a fresh time variable N and replace all LARS$^+$ atoms in all rules as follows:

$$p(\mathbf{t}) \mapsto @_N\, p(\mathbf{t}) \quad (9) \qquad \boxplus^n @_T\, p(\mathbf{t}) \mapsto @_T\, p(\mathbf{t}) \quad (11)$$

$$\boxplus^n \Box p(\mathbf{t}) \mapsto @_N\, p(\mathbf{t}) \quad (10) \qquad \boxplus^n \Diamond p(\mathbf{t}) \mapsto @_U\, p(\mathbf{t}) \quad (12)$$

where (9) refers to atoms with no surrounding LARS$^+$ operators and U in (12) is a fresh time variable unique for each replacement; arithmetic atoms are kept unchanged. The resulting program is denoted by $\mathsf{wfree}(P)$ ("*window-free*").

Example 4. Let P consist of the following rules:

$$\boxplus^3 \Box\, p(X) \to \exists Y. q(X, Y) \tag{13}$$

$$@_T\, q(X, Y) \wedge U = T + 1 \to @_U\, p(Y) \tag{14}$$

As in Example 3, the skolem chase on $\mathsf{rew}(P)$ terminates if the given input data encodes a valid timeline, else it may not (indeed, $P \notin \mathbf{L}^+_{\mathsf{LWA}}$). In $\mathsf{wfree}(P)$, (13) is changed to

$$@_N\, p(X) \to \exists Y. @_N\, q(X, Y) \tag{15}$$

Intuitively, in $\mathsf{wfree}(P)$, N is the time at which rules are evaluated and localises all simple atoms to it; windows are removed and their restrictions relaxed: $\boxplus^n \Box$ ("at all times in window up to now") becomes $@_N$ ("now"); $\boxplus^n @_T$ ("at T if in window") becomes $@_T$; and $\boxplus^n \Diamond$ ("at some time in window") becomes $@_U$ ("at some time"). As this logically weakens rule bodies, $\mathsf{wfree}(P)$ has more logical consequences than P. We obtain the following useful insight:

Theorem 3. *For every $P \in \mathbf{L}^+$ and data stream D, if the skolem chase terminates on* $\mathsf{rew}(\mathsf{wfree}(P))$ *and* $\mathsf{rew}(D)$, *then it also terminates on* $\mathsf{rew}(P)$ *and* $\mathsf{rew}(D)$.

To exploit Theorem 3, we study the chase termination over $\mathsf{rew}(\mathsf{wfree}(P))$ while restricting to actual timelines, which are incorporated by partial grounding.

Definition 4. The *partial grounding* $\mathsf{grnd}_A(P)$ of a program P for a set A of null-free facts over a set $\mathcal{P}_A$ of predicates not occurring in rule heads of P, is the set of all rules $(B \backslash B_A \rightarrow \exists \mathbf{z}.H)\sigma$, where B_A are the atoms in B with predicate in $\mathcal{P}_A$, s.t. a rule $B \rightarrow \exists \mathbf{z}.H \in P$ and a homomorphism σ between B_A and A exist, i.e., a sort- and constant- preserving mapping $\sigma : \mathsf{dom}(B_A) \rightarrow \mathsf{dom}(A)$ s.t. $B_A\sigma \subseteq A$.

As long as A comprises all facts over $\mathcal{P}_A$, $\mathsf{grnd}_A(P)$ has the same models as P and the chase is also preserved. We use this to ground the time sort in LARS$^+$:

Definition 5. Given a program P, the *temporal grounding* of $\mathsf{wfree}(P)$ for a timeline $\mathbf{T}$, denoted $\mathsf{tgrnd}_{\mathbf{T}}(P)$, is the partial grounding $\mathsf{grnd}_{a(\mathbf{T},P)}(P')$ where

- P' results from $\mathsf{rew}(\mathsf{wfree}(P))$ by adding, for each $T \in \mathcal{V}_T$ in each rule body B, an atom $T \leq T$ to B and
- $a(\mathbf{T}, P)$ is the set of all ground instances of arithmetic atoms in P with values from $\mathbf{T}$ that are true over $\mathbb{N}$.

Example 5. For $\mathsf{wfree}(P)$ from Example 4 and timeline $\mathbf{T} = [0, 1]$, the temporal grounding is as follows (the deleted ground instances of B_A are shown in parentheses):

$$[\![\boxplus\Box\, p]\!](X,0,0) \rightarrow \exists Y.[\![\boxplus\Box\, q]\!](X,Y,0,0) \qquad (0 \leq 0)$$
$$[\![\boxplus\Box\, p]\!](X,0,1) \rightarrow \exists Y.[\![\boxplus\Box\, q]\!](X,Y,0,1) \qquad (1 \leq 1)$$
$$[\![\boxplus\Box\, q]\!](X,Y,0,0) \rightarrow [\![\boxplus\Box\, p]\!](Y,0,1) \qquad (1 = 0+1)$$

While universal termination on $\mathsf{tgrnd}_{\mathbf{T}}(P)$, which can be recognized in Example 5 using e.g. MFA (Cuenca Grau et al. 2013), ensures chase termination on $\mathsf{rew}(P)$ and $\mathsf{rew}(D)$ for all data streams D on $\mathbf{T}$, simpler, position-based notions like WA still fail. We thus encode time into predicate names:

Definition 6. Let P be an existential rules program with atoms of form $[\![\boxplus\Box\, p]\!](\mathbf{s}, 0, t)$ only, where t is a time point. Then $\mathsf{tfree}(P)$ is obtained by replacing each $[\![\boxplus\Box\, p]\!](\mathbf{s}, 0, t)$ with $[\![p]\!]_t(\mathbf{s})$ for a fresh predicate $[\![p]\!]_t$ of proper signature.

Let $\mathsf{tfgrnd}_{\mathbf{T}}(P) := \mathsf{tfree}(\mathsf{tgrnd}_{\mathbf{T}}(P))$. The following result shows that this is a good basis to check for acyclicity.

Theorem 4. *If* $\mathsf{tfgrnd}_{\mathbf{T}}(P)$ *is weakly acyclic for* $P \in \mathbf{L}^+$ *and timeline* $\mathbf{T}$, *then the skolem chase terminates on* $\mathsf{rew}(P)$ *and* $\mathsf{rew}(D)$ *for all data streams* D *on* $\mathbf{T}$.

Example 6 (cont'd)**.** As $\mathsf{tfgrnd}_{\mathbf{T}}(P)$ is WA, by Theorem 4 the skolem chase on $\mathsf{rew}(P)$ and $\mathsf{rew}(D)$ always terminates.

In view of Theorem 4, we call $P \in \mathbf{L}^+$ *temporally weakly acyclic* (TLWA) over $\mathbf{T}$ if $\mathsf{tfgrnd}_{\mathbf{T}}(P)$ is WA, and denote by $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ the class of all such programs P. We then have:

Theorem 5. $\mathbf{L}^+_{\mathsf{LWA}} \subset \mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ *holds for all* $\mathbf{T}$ *s.t.* $|\mathbf{T}| \geq 2$.

Regarding complexity, as $\mathsf{rew}(\mathbf{T})$ is polynomial *in the length of* $\mathbf{T}$, it is exponential if $\mathbf{T}$ is encoded in binary. However, a polynomial axiomatisation of time is feasible, following the idea to encode numbers $0,1,\ldots,m$ using sequences of $\lceil \log_2 m \rceil$ bits and to define predicates on them, cf. (Dantsin et al. 2001), such that for the resulting rewriting $\mathsf{rew}_{\mathbf{T}}(\cdot)$ instead of $\mathsf{rew}(\cdot)$, Theorems 1 and 2 hold analogously.

BCQ answering for $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ is as for WA rules 2EXPTIME-complete in general (on extensional streams, i.e, all $v(t)$, $t \in \mathbf{T}$, are listed). The P-complete data complexity for WA rules carries over to $\mathbf{L}^+_{\mathsf{LWA}}$ but gets 2EXPTIME-hard for $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$, as hardest WA programs with bounded predicate arities (Calì, Gottlob, and Pieris 2010) can be emulated.

Table 1: Preliminary experiments for scenario S_A and S_B

S_A: $p_1/p_2/p_3$	*Run*	*Mem*	*Out*	S_B: n	*Run*	*Mem*	*Out*
0.0/0.0/0.0	13.12ms	21.9	10.5k	0	0.6s	45.0	36k
0.3/0.3/0.5	13.34ms	22.7	10.7k	2	1.3s	81.8	64k
0.7/0.7/1.0	13.67ms	22.9	10.7k	4	2.6s	114.4	82k

5 Preliminary Evaluation and Conclusion

We implemented an experimental prototype in Python, which is fed with the stream pointwise. At each time point, it computes the LARS$^+$ model with the stream collected up to the last ℓ time points, using the rewriting in Section 3 and the chase implementation of GLog (Tsamoura et al. 2021).

We considered two scenarios S_A and S_B. The first, S_A, is a toy example with conveyor belts and sensors that measure speed and temperature. The program contains 5 simple rules and the stream is parametrized by probability values p_1, p_2, and p_3 that regulate the number of rule executions (higher values lead to more reasoning). Scenario S_B is much more complex than S_A. We considered the dataset *Deep100* from the *ChaseBench* suite (Benedikt et al. 2017), which is a stress test of chase engines. We created a stream by copying all facts on each time point and rewrote the original rules using LARS$^+$ operators and different window sizes n. More details are available in Appendix D.

Table 1 reports multiple metrics obtained using a laptop, viz. avg. runtime (*Run*), avg. peak use of RAM (in MB, *Mem*), and avg. model size (# facts, *Out*). Notably, a LARS$^+$ model can be computed rather quickly, viz. in ≈13ms with an hypothetical input like S_A. This suggests that our approach can be used in scenarios that need fast response times. For "heavier" scenarios like S_B, the runtime increases but still stays within few seconds. Moreover, reasoning used at most 114MB of RAM; thus it may be done on limited hardware, e.g., sensors or edge devices.

Conclusion. Our work shows that combining existential rules with LARS can give rise to a versatile stream reasoning formalism with expressive features which is still decidable. A worthwhile future objective is to develop more efficient algorithms to compute the models. Our translation to existential rules is a good basis, but many optimisations are conceivable. On the theoretical side, a study of further decidability paradigms, especially related to guarded logics, is suggestive. Finally, further extensions towards nonmonotonic reasoning or other issues, like window validity (Ronca et al. 2018), are challenging for existential rules, but would be very useful for stream reasoning.

Acknowledgments. The authors would like to thank Mike Spadaru for his work on earlier versions of the prototype used in this work.

This work was partly supported by *Deutsche Forschungsgemeinschaft* (DFG) in project 389792660 (TRR 248, Center for Perspicuous Systems), by the *Bundesministerium für Bildung und Forschung* (BMBF) in project ScaDS.AI (Center for Scalable Data Analytics and Artificial Intelligence), and by the Center for Advancing Electronics Dresden (cfaed).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1994. *Foundations of Databases*. Addison Wesley.

Anicic, D.; Fodor, P.; Rudolph, S.; and Stojanovic, N. 2011. EP-SPARQL: A Unified Language for Event Processing and Stream Reasoning. In *WWW*, 635–644.

Baget, J.-F.; Leclère, M.; Mugnier, M.-L.; and Salvat, E. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175(9–10):1620–1654.

Barbieri, D. F.; Braga, D.; Ceri, S.; Valle, E. D.; and Grossniklaus, M. 2010. C-SPARQL: a continuous query language for rdf data streams. *International Journal of Semantic Computing* 04(01):3–25.

Bazoobandi, H. R.; Beck, H.; and Urbani, J. 2017. Expressive Stream Reasoning with Laser. In *ISWC*, 87–103.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2018. LARS: A Logic-based framework for Analytic Reasoning over Streams. *Artificial Intelligence* 261:16–70.

Beeri, C., and Vardi, M. Y. 1981. The Implication Problem for Data Dependencies. In *ICALP*, 73–85.

Bellomarini, L.; Sallinger, E.; and Gottlob, G. 2018. The Vadalog System: Datalog-based Reasoning for Knowledge Graphs. *PVLDB* 11(9):975–987.

Benedikt, M.; Konstantinidis, G.; Mecca, G.; Motik, B.; Papotti, P.; Santoro, D.; and Tsamoura, E. 2017. Benchmarking the Chase. In *PODS*, 37–52.

Brandt, S.; Kalaycı, E. G.; Kontchakov, R.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2017. Ontology-Based Data Access with a Horn Fragment of Metric Temporal Logic. In *AAAI*, 1070–1076.

Calì, A.; Gottlob, G.; Lukasiewicz, T.; Marnette, B.; and Pieris, A. 2010. Datalog+/-: A Family of Logical Knowledge Representation and Query Languages for New Applications. In *LICS*, 228–242.

Calì, A.; Gottlob, G.; and Pieris, A. 2010. Query Answering under Non-guarded Rules in Datalog+/-. In *RR*, 1–17.

Cuenca Grau, B.; Horrocks, I.; Krötzsch, M.; Kupke, C.; Magka, D.; Motik, B.; and Wang, Z. 2013. Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res.* 47:741–808.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

Dell'Aglio, D.; Della Valle, E.; van Harmelen, F.; and Bernstein, A. 2017. Stream reasoning: A survey and outlook. *Data Science* 1(1-2):59–83.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336(1):89 – 124.

Gottlob, G.; Lukasiewicz, T.; and Pieris, A. 2014. Datalog+/-: Questions and Answers. In *KR*, 682–685.

Kalaycı, E. G.; Brandt, S.; Calvanese, D.; Ryzhikov, V.; Xiao, G.; and Zakharyaschev, M. 2019. Ontology-based access to temporal data with ontop: A framework proposal. *Applied Mathematics and Computer Science* 29(1):17–30.

Kharlamov, E.; Kotidis, Y.; Mailis, T.; Neuenstadt, C.; Nikolaou, C.; Özçep, Ö. L.; Svingos, C.; Zheleznyakov, D.; Ioannidis, Y. E.; Lamparter, S.; Möller, R.; and Waaler, A. 2019. An ontology-mediated analytics-aware approach to support monitoring and diagnostics of static and streaming data. *Journal of Web Semantics* 56:30–55.

Krötzsch, M.; Marx, M.; and Rudolph, S. 2019. The Power of the Terminating Chase. In *ICDT*, 3:1–3:17.

Le-Phuoc, D.; Dao-Tran, M.; Xavier Parreira, J.; and Hauswirth, M. 2011. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *ISWC*, 370–388.

Margara, A.; Urbani, J.; Van Harmelen, F.; and Bal, H. 2014. Streaming the web: Reasoning over dynamic data. *Journal of Web Semantics* 25:24–44.

Marnette, B. 2009. Generalized Schema-Mappings: from Termination to Tractability. In *PODS*, 13–22.

Ronca, A.; Kaminski, M.; Cuenca Grau, B.; and Horrocks, I. 2018. The Window Validity Problem in Rule-Based Stream Reasoning. In *KR*, 571–581.

Tiger, M., and Heintz, F. 2016. Stream Reasoning Using Temporal Logic and Predictive Probabilistic State Models. In *TIME*, 196–205.

Tsamoura, E.; Carral, D.; Malizia, E.; and Urbani, J. 2021. Materializing knowledge bases via trigger graphs. *PVLDB* 14(6):943–956.

Urbani, J.; Krötzsch, M.; Jacobs, C. J. H.; Dragoste, I.; and Carral, D. 2018. Efficient Model Construction for Horn Logic with VLog - System Description. In *IJCAR*, 680–688.

Urbani, J.; Krötzsch, M.; and Eiter, T. 2022. Chasing Streams with Existential Rules. *arXiv:2205.02220 [cs]*. http://arxiv.org/abs/2205.02220.

Wałęga, P. A.; Kaminski, M.; and Cuenca Grau, B. 2019. Reasoning over Streaming Data in Metric Temporal Datalog. In *AAAI*, 3092–3099.

A Proofs Section 3

Theorem 1. *For any* $P \in \mathbf{L}^+$*, BCQ* q*, data stream* D *on* $\mathbf{T}$*, and* $t \in \mathbf{T}$ *holds* $P, D, t \models q$ *iff* $\mathsf{rew}(P) \cup \mathsf{rew}(D) \models_{\mathbf{T}} \mathsf{rew}(q, t)$.

Proof. Let η be the function that maps a stream S to the deductive closure of $\mathsf{rew}(S)$ under the rules (3)–(8). We show that, for any $t \in \mathbf{T}$, we have $S, t \models q$ iff $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(q, t)$, where $q = \exists \mathbf{x}.Q$. Analogous results will also be established for P and D.

Therefore, let $S = (\mathbf{T}, v)$ be such that $S, t \models q$ as in Definition 2. We show that $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(q, t)$. By $S, t \models q$, there is a $\mathbf{T}$-match σ of Q on S and t. Therefore, for every atom α in Q, $S, t \models \alpha\sigma$. Our rewriting replaces atoms $\alpha = \boxplus^n \Diamond p(\mathbf{t})$ by $\alpha' = \boxplus^n @_T\, p(\mathbf{t})$ for a fresh T. Clearly, whenever $S, t \models \alpha\sigma$, there is a suitable $s \in \mathbf{T}$ such that $S, t \models \alpha'\sigma\{T \mapsto s\}$. We can therefore construct a query without $\Diamond$ and a suitable $\mathbf{T}$-match over S and assume without loss of generality that $\Diamond$ does not occur in the query.

We claim that $\sigma_t := \sigma \cup \{C \mapsto t\}$ is a match for $\mathsf{rew}(q, t) = \exists \mathbf{x}. \bigwedge_{\alpha \in Q} \mathsf{rew}(\alpha) \wedge C \leq t \wedge t \leq C$ over $\eta(S)$. Clearly, $\eta(S), t \models_{\mathbf{T}} (C \leq t)\sigma_t$ and likewise for $t \leq C$. For the other atoms $\mathsf{rew}(\alpha)$ with $\alpha \in Q$, we can show $\eta(S), t \models \alpha\sigma_t$ by considering each possible form of atom:

- For $\alpha = p(\mathbf{t})$, we obtain $p(\mathbf{t})\sigma \in v(t)$. Since $\mathsf{rew}(\alpha) = [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, C)$ and $\mathsf{rew}(\alpha)\sigma_t = [\![\boxplus\Box\, p]\!](\mathbf{t}\sigma, 0, t)$, we get $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(\alpha)\sigma_t$ as required.
- For $\alpha = @_{t'}\, p(\mathbf{t})$, we obtain $p(\mathbf{t})\sigma \in v(t')$, and the claim follows with a similar argument as in the previous case.
- For $\alpha = \boxplus^n \Box p(\mathbf{t})$, we obtain $p(\mathbf{t})\sigma \in v(t')$ for all $t' \in \mathbf{T}$ with $t - n \leq t' \leq t$; hence, we get $\eta(S) \models_{\mathbf{T}} [\![\boxplus\Box\, p]\!](\mathbf{t}\sigma, 0, t')$ for every such t' by a similar argument as before. We have $\mathsf{rew}(\alpha) = [\![\boxplus\Box\, p]\!](\mathbf{t}, n, C)$. Since $\eta(S)$ satisfies rules (4)–(6), we find that $\eta(S) \models_{\mathbf{T}} [\![\boxplus\Box\, p]\!](\mathbf{t}, n, C)\sigma_t$: we can apply rule (6) on true atoms of the form $[\![\boxplus\Box\, p]\!](\mathbf{t}\sigma, 0, t')$ to infer windows of increasing sizes up until n; if $t - n < 0$, then rule (4) is used to start with a maximal window at time 0, which can be reduced in size by rule (5), before we again apply (6) to infer the required $[\![\boxplus\Box\, p]\!](\mathbf{t}\sigma, n, t)$.
- For $\alpha = \boxplus^n @_{t'}\, p(\mathbf{t})$, we obtain $p(\mathbf{t})\sigma \in v(t')$ and $t - n \leq t' \leq t$. Using a similar argument as before, we can use rule (7) to derive facts $[\![\boxplus @\, p]\!](\mathbf{t}, 0, t', t')$, and rule (8) to modify the window size and position to obtain $[\![\boxplus @\, p]\!](\mathbf{t}, n, t', t)$.
- The cases of atoms α that use $\top$ instead of $p(\mathbf{t})$ are shown in the same way, with the only difference that facts of the form $[\![\boxplus\Box\, p]\!](\mathbf{t}\sigma, 0, s)$ are now replaced by facts of the form $[\![\boxplus\Box\, \top]\!](0, s)$, which are provided by rule (3).
- For arithmetic atoms α, the rewriting does not change the atom, and the claim is immediate.

This completes the argument that $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(q, t)$.

Conversely, assume that there is a model $S' \models_{\mathbf{T}} \mathsf{rew}(q, t)$ that satisfies (3)–(8), and such that $S' = \eta(S)$ for a suitable S. We show that $S, t \models q$. The argument proceeds as before, but now using that $\eta(S)$ contains only facts of form $[\![\boxplus @\, p]\!](\mathbf{t}, n, s, t')$ or $[\![\boxplus\Box\, p]\!](\mathbf{t}, n', t')$ with $n' > 0$ that are needed to satisfy some rule (3)–(8).

We now also find that, for any rule $r \in P$, it holds that $S \models r$ iff $\eta(S) \models_T \mathsf{rew}(r)$, where $\mathsf{rew}(r)$ denotes the result of rewriting a single rule r as described before. This is an easy consequence from the previous statement for queries, since $\mathbf{T}$-matches for rule heads and bodies behave like query matches. Note that $\mathsf{b}(\mathsf{rew}(r))$ may contain not only the atoms in $\mathsf{rew}(\mathsf{b}(r))$ but also an additional atom $[\![\boxplus\Box\, \top]\!](0, C)$. However, the previous argument for queries still applies, since we can assume w.l.o.g. that $\mathsf{b}(r)$ contains the atom $\top$, in which case $\mathsf{b}(\mathsf{rew}(r)) = \mathsf{rew}(\mathsf{b}(r))$ does again hold. Finally, we also note that $S \models D$ iff $\eta(S) \models_T \mathsf{rew}(D)$ for data streams D.

These correspondences already show that, whenever there is a stream S with $S \models P$ and $S \models D$ but $S, t \not\models q$, we find that $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(P)$ and $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(D)$ but $\eta(S) \not\models_{\mathbf{T}} \mathsf{rew}(q, t)$. For the converse direction, we note that, for any model S' with $S' \models_{\mathbf{T}} \mathsf{rew}(P)$ and $S' \models_{\mathbf{T}} \mathsf{rew}(D)$, there is a model of the form $\eta(S) \subseteq S'$ for some stream S for which $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(P)$ and $\eta(S) \models_{\mathbf{T}} \mathsf{rew}(D)$, i.e., we can restrict attention to models of the form $\eta(S)$, which provide the semantic correspondences shown above. Indeed, a suitable $\eta(S)$ can be obtained by removing from S' all facts of the form $[\![\boxplus @\, p]\!](\mathbf{t}, n, s, t')$ or $[\![\boxplus\Box\, p]\!](\mathbf{t}, n', t')$ with $n' > 0$, and deductively closing the result under the rules (3)–(8). $\square$

B Proofs Section 4

First, we provide a more description of the (skolem) chase and of the standard notion of weak acyclicity.

The *chase* is a versatile class of reasoning algorithms for existential rules (Benedikt et al. 2017), which is based on "applying" rules iteratively until saturation (or, possibly, forever). We present a variant of the *skolem chase* (Marnette 2009), using nulls instead of skolem terms (this version is sometimes called the *semi-oblivious chase*), and extended to the time sort.

Let r be an existential rule of the form:

$$r = \forall \mathbf{x}, \mathbf{y}.\ B[\mathbf{x}, \mathbf{y}] \to \exists \mathbf{z}.\ H[\mathbf{y}, \mathbf{z}] \qquad (16)$$

where B and H are conjunctions of normal atoms, and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are mutually disjoint lists of variables. B is the *body* (denoted $\mathsf{b}(r)$), H the *head* (denoted $\mathsf{h}(r)$), and $\mathbf{y}$ the *frontier* of r. Notice that below we may treat conjunctions of atoms as sets, and we omit universal quantifiers in rules.

Moreover, let A a set of facts. A $\mathbf{T}$-match σ for $\mathsf{b}(r)$ (defined on $\mathbf{x}$ and $\mathbf{y}$) is extended to a term mapping σ^+ by setting, for each $Z \in \mathbf{z}$, $v\sigma^+ = n^{r,Z}_{\mathbf{y}\sigma}$, which is a fixed named null specific to r, Z, and $\mathbf{y}\sigma$. The $\mathbf{T}$-match σ is *active* for A if $\mathsf{h}(r)\sigma^+ \not\subseteq A$.

The *skolem chase sequence* $F_0, F_1, \ldots$ over a program P and a set of null-free facts A is specified as follows: (1) $F_0 = A$ and (2) F_{i+1} is obtained from F_i by adding $\mathsf{h}(r)\sigma^+$ for every rule $r \in P$ and active $\mathbf{T}$-match σ over $\mathsf{b}(r)$ and F_i. The result of the skolem chase is $\bigcup_{i \geq 0} F_0$ in this case. The chase *terminates* if $F_{i+1} = F_i$ for some $i \geq 0$. As usual, the *(skolem) chase* over R and A refers to this computation process or to its result, depending on context. Our definitions also apply to single-sorted existential rules without time.

For some (finite) I, a chase procedure might not terminate and determining this is undecidable in the most general case (?). Fortunately, many decidable conditions that guarantee chase termination were proposed (Cuenca Grau et al. (2013) give an overview and comparison). Among them, *weak acyclicity* can be seen as a simple representative of these approaches (Fagin et al. 2005). Intuitively, the idea is to construct a graph that we can use to track how variables "propagate" across the rules. If such propagations do not generate any cycle that involves existentially quantified variables, then we are sure the chase will always terminate. We describe the procedure more formally below.

Definition 7. For a program P, we define a directed graph G whose nodes are *predicate positions* $\langle p, i\rangle$, where $p \in \mathcal{P}$ and $1 \leq i \leq \mathsf{ar}(p)$. For a variable X and set A of atoms, let $\mathsf{pos}(X, A) := \{\langle p, i\rangle \mid p(\mathbf{t}) \in A \text{ and } t_i = X\}$ be the set of all positions where X occurs in A. For every rule r as in (16), frontier variable $Y \in \mathbf{y}$, position $\pi \in \mathsf{pos}(Y, \mathsf{b}(r))$, and existential variable $Z \in \mathbf{z}$, we add two kinds of edges to G:

- a *normal edge* $\pi \to \pi'$ for all $\pi' \in \mathsf{pos}(Y, \mathsf{h}(r))$
- a *special edge* $\pi \xrightarrow{*} \pi'$ for all $\pi' \in \mathsf{pos}(Z, \mathsf{h}(r))$

Then P is *weakly acyclic* (WA) if G does not have a cycle through a special edge.

Recall that we denote the class of all weakly acyclic programs with $\mathbf{WA}$. We are now ready to discuss Theorem 2.

Theorem 2. *For any* $P \in \mathbf{L}^+$*, we have (i)* $\mathsf{strip}(P) \in \mathbf{CT}$ *iff* $\mathsf{rew}(P) \in \mathbf{CT}$ *and (ii)* $\mathsf{strip}(P) \in \mathbf{WA}$ *iff* $\mathsf{rew}(P) \in \mathbf{WA}$.

Proof. 1) We begin with the first claim, which refers to chase termination. Let C be the set of all possible ground instances of arithmetic atoms, including "nonsensical" ones like, e.g., 0=1+0, in $\mathsf{rew}(P)$ using values from $\mathbf{T}$, and let $C_{\mathbf{T}}$ be the analogous set of all ground instances of rewritten arithmetic atoms in $\mathsf{rew}_{\mathbf{T}}(P)$ (where numbers are encoded in binary as explained before).

Let F be an arbitrary set of input facts for $\mathsf{rew}(P)$ such that $C \subseteq F$ and F does not contain facts for predicates of the form $[\![\boxplus @\, p]\!]$. Then the skolem chase on $\mathsf{rew}(P) \cup F$ contains a fact $[\![\boxplus\Box\, p]\!](\mathbf{t}, 0, t)$ iff it contains every fact of the form $[\![\boxplus\Box\, p]\!](\mathbf{t}, 0, s)$ for $s \in \mathbf{T}$. This follows from rules (5) and (6) using the atoms of C. Similarly, facts of the form $[\![\boxplus @\, p]\!](\mathbf{X}, N, T, C)$ hold at all times and for all window sizes if $[\![\boxplus\Box\, p]\!](\mathbf{t}, 0, t)$ is true for any $t \in \mathbf{T}$. In other words, the skolem chase for $\mathsf{rew}(P) \cup F$ effectively merges deductions for all time points.

The skolem chase on $\mathsf{rew}(P) \cup F$ therefore corresponds to the skolem chase on $\mathsf{strip}(P) \cup F'$, where $F' = \{p(\mathbf{t}) \mid [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, t) \in F\}$. Indeed, arithmetic atoms are always true on F since $C \subseteq F$ and can therefore be ignored, and all temporal operators can be omitted when all time points are merged. In particular, the skolem chase on $\mathsf{rew}(P) \cup F$ terminates iff the skolem chase on $\mathsf{strip}(P) \cup F'$ terminates. An analogous result holds for $\mathsf{rew}_{\mathbf{T}}(P)$ with inputs that contain $C_{\mathbf{T}}$.

Now to finish the proof of the first claim, consider $\mathsf{strip}(P) \in \mathbf{CT}$ iff $\mathsf{rew}(P) \in \mathbf{CT}$. First assume that there is a set of facts F' such that $\mathsf{strip}(P) \cup F'$ does not terminate. Every F' is of the form $F' = \{p(\mathbf{t}) \mid [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, t) \in F\}$ for some F with $C \subseteq F$ that contains no facts for predicates $[\![\boxplus @\, p]\!]$. Hence we find that $\mathsf{rew}(P) \cup F$ has no terminating skolem chase.

Conversely, assume that the skolem chase does not terminate on $\mathsf{rew}(P) \cup G$ for some set of input facts G that may not satisfy the previous conditions on F. We extend G by adding, for every fact $\alpha_@ = [\![\boxplus @\, p]\!](\mathbf{t}, n, s, t)$ a new fact $\alpha_\Box = [\![\boxplus\Box\, p]\!](\mathbf{t}, 0, t)$. This addition preserves non-termination of the chase, as every addition of input facts does for the skolem chase. As argued above, $\alpha_@$ follows from $\alpha_\Box$ using rules (4)–(8), hence we can delete $\alpha_@$ from G while preserving non-termination. This leads to a non-terminating set G without predicates $[\![\boxplus @\, p]\!]$. To satisfy the other condition on F, we can simply add C to G, which again preserves non-termination. The skolem chase on the resulting set G then again corresponds to a skolem chase on $\mathsf{strip}(P)$, which establishes non-termination. The case for $\mathsf{strip}(P) \in \mathbf{CT}$ iff $\mathsf{rew}_{\mathbf{T}}(P) \in \mathbf{CT}$ is analogous.

2) For the second claim, we first address $\mathsf{strip}(P) \in \mathbf{WA}$ iff $\mathsf{rew}(P) \in \mathbf{WA}$. Consider the graphs G_r and G_s as in Definition 7 for $\mathsf{rew}(P)$ and $\mathsf{strip}(P)$, respectively. The forward direction can be shown by establishing the following: (a) for every normal edge $\langle p, i\rangle \to \langle q, j\rangle$ in G_s, there is a path $\langle [\![\boxplus\Box\, p]\!], i\rangle \to \cdots \to \langle [\![\boxplus\Box\, q]\!], j\rangle$ in G_r; and (b) for every special edge $\langle p, i\rangle \xrightarrow{*} \langle q, j\rangle$ in G_s, there is a path $\langle [\![\boxplus\Box\, p]\!], i\rangle \to \cdots \xrightarrow{*} \langle [\![\boxplus\Box\, q]\!], j\rangle$ in G_r. Together, (a) and (b) imply that every cycle in G_s that involves a special edge also leads to such a cycle in G_r, showing the first part of the claim.

There are two kinds of rules in $\mathsf{rew}(P)$: rewritten versions of rules in P and auxiliary rules to axiomatise temporal operators. To show (a) and (b), note that the heads of rewritten rules in $\mathsf{rew}(P)$ only contain atoms of the form $[\![\boxplus\Box\, p]\!](\mathbf{t}, N, T)$, and that normal and special edges in rewritten rules are analogous to those in G_s. However, rewritten rules may also contain body predicates $[\![\boxplus @\, p]\!]$. The claim follows by noting that, for every predicate position $\langle p, i\rangle$, G_r contains a normal edge $\langle [\![\boxplus\Box\, p]\!], i\rangle \to \langle [\![\boxplus @\, p]\!], i\rangle$ due to rule (7).

For the converse direction, we can use a similar correspondence between paths in G_r and paths in G_s. However, we additionally need to observe that, for any predicate p of arity a, the additional argument positions $\langle [\![\boxplus\Box\, p]\!], a+1\rangle$, $\langle [\![\boxplus @\, p]\!], a+1\rangle$, and $\langle [\![\boxplus @\, p]\!], a+2\rangle$ do not occur in any cycle that involves a special edge. This is an easy consequence of the fact that those positions represent arguments of the time sort. Therefore, we find that every cycle in G_r that has a special edge corresponds to such a cycle in G_s. The argument for $\mathsf{strip}(P) \in \mathbf{WA}$ iff $\mathsf{rew}_{\mathbf{T}}(P) \in \mathbf{WA}$ is again similar. □

Remark. For the following Theorem 3 and the definitions of $\mathsf{grnd}_{\mathbf{T}}(\mathsf{wfree}(P))$ (Definition 5) and $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$, we assume that $\mathsf{rew}(\mathsf{wfree}(P))$ does not contain any of the auxiliary rules (6)–(8). Indeed, these rules are not relevant for chase termination in an existential rule set where atoms of the form $[\![\boxplus\Box p]\!](\mathbf{t}, n, c)$ only occur with $n = 0$ and predicates $[\![\boxplus@\, p]\!]$ do not occur at all.

Theorem 3. *For every $P \in \mathbf{L}^+$ and data stream D, if the skolem chase terminates on* $\mathsf{rew}(\mathsf{wfree}(P))$ *and* $\mathsf{rew}(D)$*, then it also terminates on* $\mathsf{rew}(P)$ *and* $\mathsf{rew}(D)$.

Proof. The claim follows from our previous observation that the rules in $\mathsf{wfree}(P)$ have more consequences than those in P. Indeed, the skolem chase is monotonic with respect to the amount of entailments, hence the result of a skolem chase on $\mathsf{rew}(\mathsf{wfree}(P))$ is a superset of the result of a skolem chase on $\mathsf{rew}(P)$. □

Consider the partial grounding introduced in Definition 4. We stated that as long as A comprises all facts over $\mathcal{P}_A$, $\mathsf{grnd}_A(P)$ has the same models as P and the chase is also preserved. This statement can be restated as follows.

Lemma 1. *Consider a program P and a set A of null-free facts over $\mathcal{P}_A$ as in Definition 4. If B is a fact set such that $A = \{p(\mathbf{t}) \in B \mid p \in \mathcal{P}_A\}$, then the skolem chase on P and B is the same as the skolem chase on* $\mathsf{grnd}_A(P)$ *and B.*

Proof. The claim follows because every $\mathbf{T}$-match on a rule of P must, by definition, instantiate all body atoms for a predicate in $\mathcal{P}_A$ to a fact in A. Since the program $\mathsf{grnd}_A(P)$ contains a rule for every possible choice of fact from A, an analogous $\mathbf{T}$-match is applicable on $\mathsf{grnd}_A(P)$, and the chases are based on applications of the same ground rule instances. □

We now consider that universal termination on $\mathsf{tgrnd}_{\mathbf{T}}(P)$ ensures chase termination on $\mathsf{rew}(P)$ and $\mathsf{rew}(D)$.

Lemma 2. *If the skolem chase universally terminates on* $\mathsf{tgrnd}_{\mathbf{T}}(P)$ *for $P \in \mathbf{L}^+$ and timeline $\mathbf{T}$, then it terminates on* $\mathsf{rew}(P)$ *and* $\mathsf{rew}(D)$ *for all data streams D on $\mathbf{T}$.*

Proof. Combine Theorem 3 and Lemma 1. □

Finally, the following lemma is easy to show.

Lemma 3. *For P as in Defn. 6 and a fact set A over predicates in P, the results of the skolem chase on P and A resp. on* $\mathsf{tfree}(P)$ *and* $\mathsf{tfree}(A)$ *are in a bijective correspondence.*

We are now ready to discuss Theorem 4.

Theorem 4. *If* $\mathsf{tfgrnd}_{\mathbf{T}}(P)$ *is weakly acyclic for $P \in \mathbf{L}^+$ and timeline $\mathbf{T}$, then the skolem chase terminates on* $\mathsf{rew}(P)$ *and* $\mathsf{rew}(D)$ *for all data streams D on $\mathbf{T}$.*

Proof. Suppose that the set $\mathsf{tfree}(\mathsf{tgrnd}_{\mathbf{T}}(P))$ of existential rules is weakly acyclic. Then the skolem chase universally terminates for it. Then, Lemma 3 ensures that the skolem chase universally terminates for $\mathsf{tfgrnd}_{\mathbf{T}}(P)$. From Lemma 2, it then follows that the skolem chase terminates on $\mathsf{rew}(P)$ and $\mathsf{rew}(D)$ for every data stream D over $\mathbf{T}$. □

Theorem 5. $\mathbf{L}^+_{\mathsf{LWA}} \subset \mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ *holds for all $\mathbf{T}$ s.t. $|\mathbf{T}| \geq 2$.*

Proof. From $\mathsf{tfree}(\mathsf{tgrnd}_{\mathbf{T}}(P))$ we can obtain $\mathsf{strip}(P)$ by a surjective renaming of predicates $[\![p]\!]_t \mapsto p$, and likewise the graph for WA by collapsing vertices, which preserves cycles. By Example 6, $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T}) \not\subseteq \mathbf{L}^+_{\mathsf{LWA}}$ for a timeline of size 2. □

C Notes on complexity

Below, we provide a more detailed description of the complexity of the proposed procedures to ensure decidability and to perform BCQ answering with LARS+ programs.

C.1 Complexity of deciding $\mathbf{L}^+_{\mathsf{LWA}}$ and $\mathbf{L}^+_{\mathsf{TLWA}}$

Theorem 6. *Given a LARS$^+$ program P, deciding whether $P \in \mathbf{L}^+_{\mathsf{LWA}}$ is* NL*-complete.*

Proof (Sketch). To decide whether $P \in \mathbf{L}^+_{\mathsf{LWA}}$ holds, we need to check whether the dependency graph G for $\mathsf{strip}(P)$ has no cycle containing a special edge. Each node of G (consisting of a pair $\langle p, i\rangle$) can be stored in logarithmic space, and deciding whether between two given nodes a normal resp. special edge exists is feasible in logarithmic space. Therefore, a cycle that contains a special edge can be nondeterministically guessed and checked stepwise in logarithmic space. Since NL = co-NL, this establishes NL membership of the problem.

The NL-hardness is inherited from the NL-hardness of WA checking of existential rules, which can be proved by a simple reduction from the graph reachability problem. Indeed, given a directed graph $G = (V, E)$ and a starting/end node *s/t* from V, we introduce a unary predicate p_v for each $v \in V$, and a binary predicate q. We then set up rules $p_v(X) \to p_{v'}(X)$ for all edges $v \to v'$ in E and for the start resp. terminal node the rules $p_t(X) \to \exists Y.q(X, Y)$ and $q(X, Y) \to p_s(Y)$, respectively. Then G has a cycle with a special edge iff there is a path from s to t. □

Turning to temporal acyclicity, we first note that already computing the temporal grounding of rules is intractable.

Proposition 1. *Deciding, given a rule r and a set A of null-free facts over predicates not occurring in* $\mathsf{b}(r)$*, whether* $\mathsf{ground}_A(\{r\})$ *is non-empty is* NP*-complete, and* NP*-hard even if* $\mathsf{ground}_A(\{r\})$ *is a temporal grounding as in Definition 5 over any timeline with at least two elements.*

Proof. Partial grounding requires to find a homomorphism from the conjunction $\mathsf{b}(r)_A$ (B_A in Definition 5) to A, which is NP-complete in general. For the special case of temporal grounding, we can show that it is still NP-complete to find a homomorphism from a conjunction of arithmetic atoms to the fact encoding of a timeline. For example, to encode three-colourability of a graph, every vertex v is assigned three variables V_r, V_g, V_b. We use atoms to express $0 \leq V_x \leq 1$ and $V_x + V_y \leq 1$, for all $x, y \in \{r, g, b\}$ with $x \neq y$. Every edge $v \to w$ of the graph is encoded as $V_x + W_y \leq 1$ for all $x \in \{r, g, b\}$. □

Deciding temporal acyclicity has presumably higher complexity.

Theorem 7. *Given a LARS$^+$ program P and a timeline $\mathbf{T}$, deciding whether $P \in \mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ is* PSPACE*- complete.*

Proof sketch. Membership in PSPACE follows as the required check corresponds to a reachability check on a graph of exponential size whose edge relation can be validated in PSPACE. For hardness, we can simulate acceptance of a

	$\mathsf{BCQ}_{\mathbf{L}}$	
	Data complexity	*Combined complexity*
$\mathbf{L}^+_{\mathsf{LWA}}$	PTIME-c	2EXPTIME-c
$\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$	2EXPTIME-c	2EXPTIME-c

Table 2: Complexity of LARS$^+$ classes (c=complete).

polynomial space bounded Turing machine by using positions of the form $\langle p_t, 1\rangle$ to represent configurations of the TM, where the binary encoding of t represents the (polynomial size) tape contents and p one of the (polynomially many) combinations of state and head positions. TM transitions are encoded by normal edges that are based on rules that use (polynomially long) conjunctions of arithmetic atoms to extract bits from time points, and to relate bits of consecutive configurations. The key for bit extraction is to define variables X_n that must be one of $\{0, 2^n\}$ for any position n on the tape. For $n = 0$, we can encode $0 \leq X_n \leq 1$. For $n + 1$, we can (recursively) define an auxiliary variable X'_n with values in $\{0, 2^n\}$ and require $X_{n+1} = X'_n + X'_n$. Note that one more auxiliary variable is used on each level, so the encoding of the polynomially many X_n is still polynomial. We can then render a given timepoint as a sum $\sum_{i=0}^{\ell} X_i$ to obtain a binary decoding. It is then easy to define rules for each TM transition. To reduce TM acceptance to weak acyclicity, it remains to create special edges from each accepting configuration to each starting configuration. □

C.2 BCQ Answering

The decision problem that corresponds to BCQ answering, defined in Definition 3, is stated below.

Problem LARS$^+$ BCQ Answering (BCQ)
Input: LARS$^+$ program P, data stream $D = (\mathbf{T}, v_D)$, time point $t \in \mathbf{T}$, and LARS$^+$ BCQ q.
Question: Does $P, D, t \models q$ hold?

Below, we will add a subscript to BCQ to indicate which class P belongs. For instance, $\mathsf{BCQ}_{\mathbf{L}}$ means that P is supposed to be in $\mathbf{L}$.

Theorem 8. *The complexity of the problem* $\mathsf{BCQ}_{\mathbf{L}}$ *is for* $\mathbf{L} \in \{\mathbf{L}^+_{\mathsf{LWA}}, \mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})\}$ *as reported in Table 2.*

This result is obtained from several lemmas presented in the following subsections. We make some useful observations about instances of BCQ:

Proposition 2. *Given an instance P, D, t, q of* BCQ*, let P' result from P by replacing each window size n occurring in P such that $n \geq |\mathbf{T}|$ with $|\mathbf{T}| - 1$. Then P, D, T, q is a yes-instance of* BCQ *iff P', D, t, q is a yes-instance of* BCQ.

As P' in Proposition 2 is easily constructed from P and $\mathbf{T}$, we thus may assume assume without loss of generality that the largest window size m occurring in P of an BCQ instance satisfies $m \leq |\mathbf{T}| - 1$.

Proposition 3. *Given an instance P, D, t, q of* BCQ*, let P' result from P by adding the rule $r_q := q \wedge @_N \mathsf{time}_q \wedge \boxplus^0 @_N \top \rightarrow @_0 \mathsf{yes}$, where time_q and yes are fresh nullary predicates. Then P, D, t, q is a yes-instance of* BCQ *iff P', D', t, yes is a yes-instance of* BCQ*, where $D' = (\mathbf{T}, v'_d\})$ with $v'_d(t) = v_d(t) \cup \{\mathsf{time}_q\}$ and $v'_d(t') = v_d(t')$ for $t' \neq t$.*

That is, we can compile the query q into the program P such that the new query is a simple propositional atom, with little effort. The membership of P' in any of the classes $\mathbf{L} \in \{\mathbf{L}^+_{\mathsf{LWA}}, \mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})\}$ that we consider coincides with the membership of P in the respective class $\mathsf{BCQ}_{\mathbf{L}}$.

In the sequel, we exploit Propositions 2 and 3 and restrict without loss of generality our attention for deriving upper boundes to BCQ instances where the largest window size is at most $|\mathbf{T}| - 1$ and the query is a simple nullary (i.e., propositional) atom. Furthermore, we assume without loss of generality that for any atom $@_T\, b$ occurring in rules of P, the term T is a time variable (if not, we can replace the atom by $@'_T b$ and add $T' = T$ in the rule body,[2] where T' is a fresh time variable). We call instances of BCQ which satisfy these properties *trimmed*.

BCQ Answering with $\mathbf{L}^+_{\mathsf{LWA}}$

Lemma 4. *Problem* $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{LWA}}}$ *is (i) in* PTIME *under data complexity and (ii) in* 2EXPTIME *under combined complexity.*

Proof (Sketch). Without loss of generality, the instance is trimmed. *(i)* For $P \in \mathbf{L}^+_{\mathsf{LWA}}$, the number of abstract constants and nulls generated in a chase of $P' = \mathsf{grnd}_{\mathbf{T}}(\mathsf{wfree}(P))$ can bounded similarly as in Lemma 6 below, but with a smaller value of $s = n_p \times a$, since the time arguments can be ignored and each null value must be generated at stage s. Thus, in the naive bound $b = (k \times |P| \times |\mathbf{T}|^{\ell} \times \ell \times n_c)^{\ell^s}$ the exponents ℓ and ℓ^s are constant; furthermore also the bound for $n_r'^{\ell^s}$ in (18) is then polynomial, and so overall we obtain that the number of abstract constants and nulls generated is polynomially bounded.

Along a similar argumentation as in Lemma 7, it can be shown that only a polynomial number of atoms will be generated, and each step can be done in polynomial time; hence we obtain overall a polynomial time algorithm for BCQ answering.

In case (ii), the result follows from Theorem 5 and Lemma 7. □

Lemma 5. *Problem* $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{LWA}}}$ *is (i)* PTIME*-hard under data complexity and (ii)* 2EXPTIME*-hard under combined complexity.*

Proof. In case (i), the result follows from Theorem 5 and from the fact that the complexity of BCQ answering from datalog programs is PTIME-complete under data complexity.

In case (ii), the result is trivially inherited from the complexity of BCQ answering from WA existential rules, which is 2EXPTIME-complete under combined complexity (Calì,

[2]We use $t_1 = t_2$ for time terms t_1 and t_2 as a shorthand for $t_1 \leq t_2 \wedge t_2 \leq t_1$.

Gottlob, and Pieris 2010), taking into account that the problem can be simply reduced to an empty data stream D with timeline $\mathbf{T} = [0, 0]$. $\square$

BCQ Answering with $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$

The following lemma is the key for obtaining upper bounds for BCQ answering with programs in $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$.

Lemma 6. *Given a trimmed instance of* $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})}$*, in the skolem chase of* $P' = \mathsf{grnd}_{\mathbf{T}}(\mathsf{wfree}(P))$ *over* $\mathsf{rew}(D) \cup \mathsf{rew}(T)$*, at most double exponentially many abstract constants and nulls in the size of* P *and* D *occur. This number can be (naively) bounded by*

$$b = (k \times |P| \times |\mathbf{T}|^\ell \times \ell \times n_c)^{\ell^s} \tag{17}$$

where

- ℓ *is the maximal rule length in* P*;*
- n_c *is the number of abstract constants in* P *and* D*;*
- $s = |\mathbf{T}| \times n_p \times a$*, where* n_p *and* a *are the number of predicates and the maximal predicate arity in* P*, respectively.*

Proof. We consider the application of the existential rules in P' for null value generation as in the skolem chase. Starting from constants, the first null values are generated by applying rules where all frontier variables are substituted with constants; then the null values generated can take part in generating further null values etc. Each null value is generated at a *stage* $i \geq 0$, which corresponds to the step of the skolem chase sequence where it first appears.

As P is in $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$, the program P' is weakly acyclic. We claim that each null value will be generated up to at most stage $s = |\mathbf{T}| \times n_p \times a$.

Recall that, as remarked earlier, the rules (6)–(8) are not included in the rewriting for P'.

The program P' therefore is of the form in Lemma 3, and we have in $\mathsf{tfree}(P')$ predicates $[\![p]\!]_t$ where p is a simple predicate from P and t is a time point in $\mathbf{T}$.

In the dependency graph G for $\mathsf{tfree}(P')$, the number of nodes is thus bounded by $|\mathbf{T}| \times n_p \times a$ (note that the nullary predicates $[\![\top]\!]_t$ do not create nodes in G).

Each null value ω of stage i can flow by rule applications along different predicate argument positions, reflected by normal edges in G, until it is used in creating a null value ω' of stage $i+1$. As P' is weakly acyclic, also $\mathsf{tfree}(P')$ is weakly acyclic and so the newly created null value ω' can not flow to any predicate argument positions at which ω was present. Hence, the number of stages for null value generation is bounded by $|\mathbf{T}| \times n_p \times a$.

We will now argue that the bound b claimed in (17), i.e. $b = (k \times |P| \times |\mathbf{T}|^\ell \times \ell \times n_c)^{\ell^s}$ holds and that it is double exponential in the size of P and D.

Now let $n_r = |P'|$ denote the number of rules in $|P'|$. Let v_i denote the number of abstract constants and null values that we have up to the i^{th} stage of null value generation. If $i = 0$, then we have not have applied any rules to generate null values and we thus have $v_0 = n_c$.

If we apply the rules to generate null values for the first time, their number is bounded by $n_r \times \ell \times {v_0}^\ell$ as we have n_r rules, each rule has at most ℓ existential variables, and we have at most ℓ frontier variables that generate a null value.

So v_1 will satisfy

$$v_1 \leq v_0 + n_r \times \ell \times {v_0}^\ell \leq n'_r \times \ell \times {v_0}^\ell,$$

where $n'_r = n_r + 1$.

Now we just iterate to obtain v_2 in a similar way, and we get

$$v_2 \leq v_1 + n_r \times \ell \times {v_1}^\ell \leq (n'_r \times \ell)^{\ell+1} \times {v_0}^{\ell^2}.$$

If we continue this, we can get the form

$$v_i \leq (n'_r \times \ell)^{\ell^i} \times {v_0}^{\ell^i} = (n'_r \times \ell \times v_0)^{\ell^i}.$$

That is, we get a value that for $i = s$ fulfills

$$v_s \leq (n'_r \times \ell \times v_0)^{\ell^s} = ((n_r + 1) \times \ell \times v_0)^{\ell^s}.$$

Now ℓ and $v_0 = n_c$ are polynomial in the size of P and D; hence the terms ℓ^{ℓ^s} and $v_0^{\ell^s}$ are double exponential in the size of P and D. The number n_r of rules in P' obeys

$$n_r < k \times |P| \times |\mathbf{T}|^\ell, \quad \text{thus} \quad n'_r \leq k \times |P| \times |\mathbf{T}|^\ell$$

for some constant k (each rule in P induces one rule in $\mathsf{rew}(P)$ and assuming P is nonempty k accounts for the extra rules (3–8)), and thus is single exponential in the size of P and D. Consequently,

$${n'_r}^{\ell^s} \leq (k \times |P| \times |\mathbf{T}|^\ell)^{\ell^s} = (k \times |P|)^{\ell^s} \times |\mathbf{T}|^{\ell^{s+1}} \tag{18}$$

is double exponential in the size of P and D; this shows the claim. $\square$

We note that under data complexity, the bound in Lemma 6 is still double exponential in the length $|\mathbf{T}|$ of the timeline $\mathbf{T}$, and in fact instances where double exponentially many null values are created do exist (see the proof of Lemma 7). However, if in addition $|\mathbf{T}|$ is bounded by a constant, then the bound is polynomial. Hence, BCQ answering with programs in $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})}$ is tractable in this case.

Based on Lemma 6, we obtain the following result.

Lemma 7. *Problem* $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})}$ *is in* 2EXPTIME *under data and combined complexity.*

Proof. Without loss of generality, the instance is trimmed. We note that the program $\mathsf{wfree}(P)$ is a logical strengthening of P, as in each rule r in P the body of r is weakened; that is, more rule applications to derive null-free facts are possible for $\mathsf{wfree}(P)$ over D than for P. Hence, an upper bound for deriving the query atom q with $\mathsf{wfree}(P)$ over D will give us an upper bound for deriving q with P over D as well. In the sequel, we thus consider $\mathsf{wfree}(P)$ and use Theorem 2.

By Lemma 6, the number of abstract constants and nulls created by evaluating the program $P' = \mathsf{grnd}_{\mathbf{T}}(\mathsf{wfree}(P))$ over $\mathsf{rew}(D) \cup \mathsf{rew}(T)$ is bounded by a double exponential number $b = (k \times |P| \times |\mathbf{T}|^\ell \times \ell \times n_c)^{\ell^s}$. Recall again that rules (4)–(8) are omitted from P', as remarked before.

By Lemma 3, instead of P' we can equivalently consider $\mathsf{tfree}(P')$, in which the number of predicates is bounded by

$\|P\| \cdot |\mathbf{T}|$ and their arities are bounded by the maximal rule length ℓ in P. Hence, no more than

$$\begin{aligned} c_1\|P\| \cdot |\mathbf{T}|b^\ell &= c_1\|P\| \cdot |\mathbf{T}|((k \times |P| \times |\mathbf{T}|^\ell \times \ell \times n_c)^{\ell^s})^\ell \\ &= c_1\|P\| \cdot |\mathbf{T}|(k \times |P| \times |\mathbf{T}|^\ell \times \ell \times n_c))^{\ell \times \ell^s} \end{aligned}$$

many ground atoms, where c_1 is a constant, will be derived by the skolem chase to answer the query q, which is double exponential in the size of P and D. As each derivation step can be done, relative to the already derived atoms, in time exponential in the maximal number of variables in a rule (by simple rule matching) and thus in double exponential time, the overall time to run the skolem chase is bounded double exponentially. Furthermore, computing P' is feasible in exponential time in the size of P and D, and computing $\mathsf{rew}(D) \cup \mathsf{rew}(T)$ is feasible in polynomial time in the size of P and D. Summing up, this yields a double exponential time upper bound for problem $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})}$. □

Lemma 8. *Problem* $\mathsf{BCQ}_{\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})}$ *is* 2EXPTIME*-hard under data complexity and combined complexity.*

Proof. This result can be accomplished by adjusting a 2EXPTIME-hardness proof for BCQ Answering from a set of WA existential rules by Calì, Gottlob, and Pieris (2010). Their proof presents an encoding of the acceptance problem for a (deterministic) Turing machine M that operates on a given input I in double exponential time. At the core of the encoding are existential rules that generate double exponentially many null values with a linear order (given by a successor relation $succ$ and predicates min and max that single out the first and the last element, respectively). The rules are schematic and use indexed predicates $succ_i, min_i, max_i, r_i, s_i$ which are defined inductively for $i = 0, \dots m$.

The machine computation is then simulated using standard Datalog rules at m, which are fixed (independent of the machine M); further rules serve to describe the tape contents of the initial configuration. For a machine description M, a Boolean query $q = accept(X)$, where X refers to a time instant of the computation (represented by a null), evaluates to true iff M accepts the input.

We now describe the construction following (**?**), adapted to our needs for $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ programs. The key observation is that each indexed predicate p_i from above can be replaced by a predicate p, such that $p_i(\mathbf{x})$ is represented by $@_i p(\mathbf{x})$, where we use a timeline $\mathbf{T} = [0, \dots, m]$.

Let $M = \langle S, \Lambda, \llcorner\!\lrcorner, \delta, s_0, F\rangle$ be an (one-tape) deterministic Turing machine (DTM), where S is a finite (non-empty) set of states, Λ is the finite (non-empty) set of the tape symbols, $\llcorner\!\lrcorner \in \Lambda$ is the blank symbol, $\delta : (S \setminus F) \times \Lambda \to S \times \Lambda \times \{-1, 1, 0\}$ is the transition function, $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of accepting states. We assume that M is well-behaved and never tries to read beyond its tape boundaries.

Without loss of generality, we can always assume that M has exactly one accepting state, denoted as s_{acc}, and that s_0 and s_{acc} are always the same (i.e., fixed). Furthermore, we may assume that M operates on empty input (I is void; we could in polynomial time construct a machine M' that first writes I on the tape and then simulates M). This assumption is not made in (**?**), but simplifies the construction.

We construct a fixed $\mathbf{L}^+_{\mathsf{TLWA}}(\mathbf{T})$ program P, a data stream D, and a (fixed) BCQ q such that $P, D \models q$ iff M accepts the empty input I within time 2^{2^m}-1 where $m = n^k$, $k > 0$, and n is the size of M; here δ is represented by a table T_δ that holds tuples $\mathbf{t} = (s, a, s', a', d)$, with the meaning that if the machine reads a in state s at position k on the tape, then it replaces a with a', changes to state s', and moves the cursor to position $k + d$.

We use the following predicates:

- $symbol/3$ to hold the contents of a cell of the tape, where $symbol(\tau, \pi, a)$ means that at time instant τ, cell π holds symbol a;
- $cursor/2$ to hold the position of the cursor (read/write head), where $cursor(\tau, \pi)$ means that at time instant τ, the cursor is at position π;
- $state/2$ to hold the state, where $state(\tau, s)$ means that at time instant τ, the machine is in state s;
- $transition/5$ to store the transition function δ; for each tuple $\mathbf{t} \in T_\delta$, we have an atom $transition(\mathbf{t})$. This predicate is extensional, i.e., it is in the data stream.
- $accept/1$ to hold that M accepts, where $accept(\tau)$ means that it accepts at time instant τ;
- $succ/2$, $min/1$, $max/1$, $r/1$, $s/3$: these predicates are auxiliary predicates to generate double exponentially many symbols for the simulation of M;
- $\leq/2$: this is a linear order on the (double exponentially many) elements of r at the end of the stream;
- $end/0$: an atom to mark the end of the stream.

The idea is that in the stream, at time point 0 we will have $2^{2^0} = 2$ many elements, c_0 and c_1, which are distinct constants. Using r and s, these constants will create new nulls in a progressive fashion along the timeline, such that at time point m, we shall have 2^{2^m} many elements.

The data stream $D = (\mathbf{T}, v_D)$ has the timeline $\mathbf{T} = [0, m]$, and we put at time m the description of the transition function of M, i.e., all facts $transition(s, a, s', a', d)$, and the atom end; there are no further atoms in D. That is, $v_D = \{m \mapsto \{transition(\mathbf{t}) \mid \mathbf{t} \in T_\delta\} \cup \{end\}$.

The program P consists of facts and rules as follows.

- $@_0 min(c_0)$, $@_0 max(c_0)$, $@_0 succ(c_0, c_1)$, $@_0 r(c_0)$, $@_0 r(c_1)$.
- (initialization rules) the tape of M will be initialized to all blanks (owing to our assumption; recall that rules are implicitly universally quantified over time):

$$end, min(X), r(Y) \to symbol(X, Y, \llcorner\!\lrcorner);$$

the cursor is at the initial position:

$$end, min(X) \to cursor(X, X)$$

and the machine is in the initial state:

$$end, min(X) \to state(X, s_0)$$

- (transition rules) three rules describe the moves of M:
  - left move:

$$end, transition(S_1, A_1, S_2, A_2, -1),$$
$$symbol(T_1, C_2, A_1), state(T_1, S_1),$$
$$cursor(T_1, C_2), succ(T_1, T_2), succ(C_1, C_2) \rightarrow$$
$$symbol(T_2, C_2, A_2), state(T_2, S_2), cursor(T_2, C_1)$$

  - right move:

$$end, transition(S_1, A_1, S_2, A_2, 1),$$
$$symbol(T_1, C_1, A_1), state(T_1, S_1),$$
$$cursor(T_1, C_1), succ(T_1, T_2), succ(C_1, C_2) \rightarrow$$
$$symbol(T_2, C_1, A_2), state(T_2, S_2), cursor(T_2, C_2)$$

  - stay move:

$$end, transition(S_1, A_1, S_2, A_2, 0), symbol(T_1, C, A_1),$$
$$state(T_1, S_1), cursor(T_1, C), succ(T_1, T_2) \rightarrow$$
$$symbol(T_2, C, A_2), state(T_2, S_2), cursor(T_2, C)$$

- (inertia rules) the contents of the tape not at the cursor position has to be carried over:

$$end, cursor(T_1, C_2), succ(C, C_2), \leq(C_1, C),$$
$$symbol(T_1, C_1, A), succ(T_1, T_2) \rightarrow$$
$$symbol(T_2, C_1, A)$$

$$end, cursor(T_1, C_1), succ(C_1, C), \leq(C, C_2),$$
$$symbol(T_1, C_2, A), succ(T_1, T_2) \rightarrow$$
$$symbol(T_2, C_2, A)$$

- (acceptance rule)

$$end, state(T, s_{acc}) \rightarrow accept(T)$$

In addition to these rules, we have rules that define the auxiliary predicates. At the heart is the generation of nulls at the time point m, which represent exponentially long bit vectors. This is accomplished with the following rules:

$$r(X), r(Y) \rightarrow \exists Z.s(X, Y, Z)$$

$$s(X, Y, Z), T' = \mathsf{now} + 1 \rightarrow \exists Z.@_{T'} r(Z)$$

$$s(X, Y_1, Z_1), s(X, Y_2, Z_2),$$
$$succ(Y_1, Y_2), T' = \mathsf{now} + 1 \rightarrow @_{T'} succ(Z_1, Z_2)$$

$$s(X_1, Y_1, Z_1), s(X_2, Y_2, Z_2), max(Y_1), min(Y_2),$$
$$succ(X_1, X_2), T' = \mathsf{now} + 1 \rightarrow @_{T'} succ(Z_1, Z_2)$$

$$s(X, X, Z), min(X), T' = \mathsf{now} + 1 \rightarrow @_{T'} min(Z)$$

$$s(X, X, Z), max(X), T' = \mathsf{now} + 1 \rightarrow @_{T'} max(Z)$$

Intuitively, the effect of these rules is that, at each time point i, the elements in r are paired, such that tuples of length 2^i yield tuples of length 2^{i+1}, where nulls give names to these tuples.

Finally, the order $\leq$ is defined as follows:

$$end, r(X) \rightarrow \leq(X, X)$$
$$end, succ(X, Y), \leq(Y, Z) \rightarrow \leq(X, Z).$$

This completes the description of the program P. Notice that for varying inputs M, the rules of P are by our assumptions the same, thus fixed. Furthermore, only a single rule has an existential variable in the head; however, no cycles through special edges in the dependency graph of the temporal grounding of the program P over D are possible.

It can be shown that some atom $accept(t)$, where t is a ground term (in fact, a null) can be derived with P over the data stream D at time point m iff M accepts (the empty) input.

To complete the construction, we thus set the BCQ q to $accept(X)$ for evaluation at time point m. Alternatively, we could also introduce a rule

$$accept(X) \rightarrow q$$

and answer q at time point m. □

Please notice that in the construction in Lemma 8, we may change the accept rule to

$$accept(X) \rightarrow @_0 q$$

i.e., put the query atom at time point 0; thus asking whether q can be entailed with P over D at time point 0 is 2EXPTIME-hard, i.e., for BCQ at a fixed query time.

D Further experimental details

The experiments were conducted using a MacbookPro16,1 with Intel Core i7 2.6GHz and 32GB RAM. Scenario S_A is meant to simulate a stream with b conveyor belts and multiple sensors that measure the speed and the temperature. When the speed is too slow or the temperature is too high, the rules trigger warnings and errors. The rules used in this scenario are the following:

$$\mathsf{belt}(X) \rightarrow \exists Y.\mathsf{bOpr}(X, Y) \quad (19)$$

$$\boxplus^5 \Diamond \mathsf{bSpeed}(X, Y) \wedge \mathsf{slow}(Y) \rightarrow \exists Z.\mathsf{brkG}(X, Z) \quad (20)$$

$$\boxplus^3 \Box \mathsf{bTmp}(X, Y) \wedge \mathsf{high}(Y) \rightarrow \exists Z.\mathsf{incId}(Z, X) \quad (21)$$

$$\mathsf{incId}(Y, X) \wedge \mathsf{bOpr}(X, Z) \rightarrow \mathsf{assign}(Y, Z) \quad (22)$$

$$\boxplus^3 \Box \mathsf{incId}(Z, X) \rightarrow \mathsf{block}(X) \quad (23)$$

where bOpr, brkG, incId is short for beltOperator, brokenGear, incidentId, respectively. Here (19) and (20) use existentials to introduce new potentially unknown individuals while (21) introduces a new incident ID if the temperature is high; (22) assigns the incident to the current belt's operator, while (23) blocks the belt if the incident has been persisting since three time points in the past.

Scenario S_B is created as follows. *Deep100* from the *ChaseBench* suite (Benedikt et al. 2017) contains 1k facts and 1.1k existential rules with 1 body atom and 3-4 head atoms, and predicates having arity 3 or 4. We created a stream by copying all 1k facts on each time point, and prefixed in rules each body atom B in with either $\boxplus^n \Box$ (50%) or $\boxplus^n \Diamond$ (50%) for some n. In both scenarios, we created streams with 100 time points and set $\ell = 6$, which is large enough to fill all windows.

The data stream contains $b = 100$ belts. For each belt, the data stream contains 3 facts: the identifier of the belt (e.g.,

$\mathsf{belt}(b_1)$, the value of the speed, which can be either high or low (e.g., $\mathsf{bSpeed}(b_1, low)$), and the value of the temperature, which is an integer from 1 to 9 (e.g., $\mathsf{bTmp}(b_1, 3)$). At each time point, every belt has a slow speed with probability p_1 (hence triggering rule (20)), and high temperature with probability p_2, which lasts for at least four consecutive time points with probability p_3 to trigger rules (21-23). The computation of the LARS$^+$ model is invoked at each time point and the reported numbers contain the averages across all time points.