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 $r_1: \langle \exists \triangle \text{beltTmp}(X, Y) \land \text{high}(Y) \rightarrow \text{warn}(X) \rangle$ issues a warning if the temperature on a conveyor belt has been high for all ($\langle \triangledown \rangle$) last three time points ($\triangledown^3$).

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\(^*\) (Calì et al. 2010). As a simple example, the rule $r_2: \text{belt}(X) \rightarrow \exists Y.\text{beltOperator}(X, Y)$ 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.

Source code is at https://github.com/karmaresearch/elars; for a longer version of this paper see (Urbani, Krötzsch, and Eiter 2022).
2 LARS+

Currently, LARS and DatalogMTL (Brandt et al. 2017; Walega, 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 $\mathbb{E}$. 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 Urbaní 2017).

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

Predicates $p$ are from a set $P$ of predicates and have arity $ar(p) \geq 0$, with each position typed (abstract or time sort). A normal atom is an expression $p(t)$, $t = t_1, \ldots, t_{ar(p)}$, 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 $\mathbb{A}$. For an atom $\alpha$ (or any other logical expression introduced below), the domain $dom(\alpha)$ of $\alpha$ is the set of all terms in $\alpha$; we write $\alpha[x]$ to state that $x = dom(\alpha) \cap (V_A \cup V_T)$; and we say that $\alpha$ is ground if it contains no variables.

A predicate $p \in P$ 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 $\alpha$ has the form

$$\alpha := a | b | @(t) -b | \mathbb{E} p @(t) -b | \mathbb{E} p \diamond b | \mathbb{E} p \Box b$$

(1)

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

Arithmetic atoms do not depend on time, whereas atoms $\mathbb{E} p$ 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 $\mathbb{E} p b$ or as $\mathbb{E} p b$.

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

$$r = \forall x,y. (B[x, y] \rightarrow \exists v.H[y, v])$$

(2)

where $x$, $y$, and $v$ are mutually disjoint sets of variables, and $v$ contains only abstract variables; the body $B[x, y]$ is a conjunction of LARS$^*$ atoms; and the head $H[y, v]$ is a conjunction of atoms of the form $b$ or $\mathbb{E} p -b$ in (1). We set $b(r) := B$ and $h(r) := H$, and we usually omit the leading $\square$ 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 $L^+$.

Semantics Like for LARS, the semantics of LARS$^+$ is based on streams. Formally, a stream $S = (T, v)$ consists of a timeline $T = [0, h] \subset \mathbb{N}$ and an evaluation function $v : \mathbb{N} \rightarrow 2^A$ such that, for all $t \in \mathbb{N}$, $v(t)$ is a set of ground normal atoms and $v(t) = \emptyset$ if $t \notin T$. 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 $t \in T$, we write $w_n(S, t)$ for the stream $([0, t], v(t))$ 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 $S = (T, v)$, a simple ground atom $b$, and $t, t', n \in \mathbb{N}$, we write:

$$S, t \models b \iff b \in v(t), \quad S, t \models \mathbb{E} p -b \iff S, t' \models b, \quad S, t \models \mathbb{E} p b \iff S, t' \models b$$

for all $t' \in \mathbb{T}$ and $v(t') \subseteq v(t)$.

Further, $S, t \models \mathbb{T}$ holds for all $t \in T$ and $S, t \models a$ as 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 $T$-match $\sigma$ 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 $T$ ($X \sigma \in T$) 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 $S = (T, v)$, written $S \models r$, if either (i) $h(r)$ contains some time point $t \notin T$ (i.e., ignore inference outside of scope), or (ii) for all $t \in T$, every $T$-match $\sigma$ of $b(r)$ on $S$ and $t$ is extendable to a $T$-match $\sigma' \supseteq \sigma$ of $b(r)$ on $S$ and $t$.

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

Example 1. Consider the data stream $D = ([0, 9], v)$, where

$$v(t) = \{\text{belt}(b_1), \text{high}(90), \text{beltTemp}(b_1, \text{tmp}(t))\}$$

for each $t \in [0, 9]$, where $\text{tmp}(t) = 90$ if $t \leq 4$ and $\text{tmp}(t) = 70$ otherwise. Then any model $S$ of the rules $r_1, r_2$ in Section 1 and $D$ fulfills $S, 4 \models \text{warn}(b_1) \land \text{beltOperator}(b_1, v')$ for some constant or null $v$. Similarly $S, 5 \models \text{beltOperator}(b_1, v')$ for some constant or null $v'$ while $S, 5 \models \text{warn}(b_1)$ 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 $\exists x. Q[x]$, where $Q$ is a conjunction of LARS$^*$ atoms. A stream $S = (T, v)$ satisfies $q$ at time $t$, written $S, t \models q$, if some $T$-match $\sigma$ of $Q$ on $S$ and $t$ exists. A program $P \in L^+$ 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 $\exists X. \mathbb{E} p \diamond \text{warn}(X)$, 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 rewriting 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 $\text{rew}(P)$ has 5 steps:
(1) Each atom $\mathbb{P}^n \circ p(t)$ is replaced by $\mathbb{P}^n @ T p(t)$, where $T$ is a fresh variable used only in one atom.
(2) For any simple predicate $p$, we add auxiliary predicates $\mathbb{P}^n [p]$ and $\mathbb{P}^n [p]$ of arity $ar(p)+2$ resp. $ar(p)+3$. Intuitively, $\mathbb{P}^n [p] (t, n, C)$ and $\mathbb{P}^n [p] (t, n, T, C)$ mean that $\mathbb{P}^n \circ p(t)$ and $\mathbb{P}^n @ T p(t)$ hold at time $C$, respectively.
(3) Using a fresh variable $C$ to represent the current time, we rewrite non-arithmetic atoms $\alpha$ in $P$ (where $T = \mathbb{T}())$ to
\[
\text{rew}(\alpha) = \begin{cases}
\mathbb{P}^n [p](t, 0, C) & \text{if } \alpha = p(t), \\
\mathbb{P}^n [p](t, 0, T) & \text{if } \alpha = @ T p(t), \\
\mathbb{P}^n [p](t, n, C) & \text{if } \alpha = \mathbb{P}^n \circ p(t), \\
\mathbb{P}^n [p](t, n, T, C) & \text{if } \alpha = \mathbb{P}^n @ T p(t)
\end{cases}
\]
(4) We add $\mathbb{P}^n [\mathbb{T}] (0, C)$ in rule bodies not containing $C$.
(5) For every predicate $p$ (including $\mathbb{T}$), we add the following rules to $P$, where $X$ is a list of variables of length $ar(p)$ and $m = \max(0, n | \mathbb{P}^n$ occurs in $P)$:
\[
0 \leq C \rightarrow \mathbb{P}^n [\mathbb{T}](0, C) \quad (3)
\]
\[
\mathbb{P}^n [p](X, 0, 0) \rightarrow \mathbb{P}^n [p](X, m, 0) \quad (4)
\]
\[
\mathbb{P}^n [p](X, N', C) \land N' = N+1 \rightarrow \mathbb{P}^n [p](X, N, C) \quad (5)
\]
\[
\mathbb{P}^n [p](X, N, C) \land N' = N+1 \land N' \leq m \land C' = C+1 \land \mathbb{P}^n [p](X, 0, C') \rightarrow \mathbb{P}^n [p](X, N', C') \quad (6)
\]
\[
\mathbb{P}^n [p](X, 0, C) \rightarrow \mathbb{P}^n [p](X, N', C') \quad (7)
\]
\[
\mathbb{P}^n [p](X, N, T, C) \land N' \leq m \land N' = N+1 \land I \leq 1 \land C' = C+1 \rightarrow \mathbb{P}^n [p](X, N', T, C') \quad (8)
\]
We rewrite a LARS+ BCQ $\exists x. R$ and time point $t$ similarly to $\text{rew}(\exists x. R, t) = \exists x. \bigwedge_{n \in Q} \text{rew}(\alpha) \land C \leq t \leq C$ (treating atoms $\mathbb{P}^n \circ p(t)$ as before), and a stream $S = (T, v)$ to facts $\text{rew}(S) = \{ \mathbb{P}^n [p](t, 0, s) \mid p(t) \in v(t), t \in T \}$.

Example 2. We illustrate the rewriting on $r_1$. Step (2) creates predicates $\mathbb{P}^n \circ \text{tempM}(p)$, $\mathbb{P}^n \circ \text{high}$, and $\mathbb{P}^n \circ \text{warn}$ and Step (3) the rule $\mathbb{P}^n \circ \text{high}(X, Y, 3, C) \land \mathbb{P}^n \circ \text{high}(Y, 0, C) \rightarrow \mathbb{P}^n \circ \text{warn}(X, 0, C)$. Step (5) adds auxiliary rules to implement the semantics; e.g., rule (6) ensures that $\mathbb{P}^n$-facts survive across time points, say if $\mathbb{P}^n \circ \text{tempM}(a, b, 0, 6)$ and $\mathbb{P}^n \circ \text{tempM}(a, b, 2, 5)$ hold, then $\mathbb{P}^n \circ \text{tempM}(a, b, 3, 6)$ should hold as well.

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

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

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 $T$ can be simulated with regular atoms: simply add the set $\text{rew}(T)$ of all true instances of arithmetic atoms in $P$ over $T$ and view $\text{rew}(P) \cup \text{rew}(D) \cup \text{rew}(T)$ as a single-sorted theory.

4 Decidability

As BCQ entailment over existential rules is undecidable, we desire that the rewriting $\text{rew}(\cdot)$ 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 (see (Urbani, Krötzsch, and Eiter 2022)).

Conditions like the canonical weak acyclicity (WA) (Fagnin 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 $\text{rew}(P)$ (viewed as single-sorted theory) while ignoring $\text{rew}(D)$ and $\text{rew}(T)$. Universal termination may here be seen as an analysis that disregards time. To formalise this, let $\text{strip}(P)$ result from $P$ by deleting all arithmetic atoms, window operators, and temporal quantifiers, and let $CT$ and 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 $P \in L^+$, we have (i) $\text{strip}(P) \in CT$ iff $\text{rew}(P) \in CT$ and (ii) $\text{strip}(P) \in WA$ iff $\text{rew}(P) \in WA$.

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

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

**Example 3.** Consider $P = \{ @ T (X, Y) \land T' = T+1 \rightarrow \mathbb{P}^n @ T p(Y), (Y, V) \}$. The skolem chase on $\text{rew}(P) \cup \text{rew}(D) \cup \text{rew}(T)$ terminates on all $T$ and $D$, but not universally for non-standard timelines where e.g., $0 = 0 + 1$ holds. That is, reasoning with $P$ always terminates despite $P \notin L^+_{}\text{WA}$.

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

\[
p(t) \rightarrow @ N p(t) \quad (9)
\]
\[
\mathbb{P}^n @ T p(t) \rightarrow @ T p(t) \quad (11)
\]
\[
\mathbb{P}^n @ T p(t) \rightarrow @ T p(t) \quad (10)
\]
\[
\mathbb{P}^n @ T p(t) \rightarrow @ T p(t) \quad (12)
\]

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 $\text{wfree}(P)$ (“window-free”).

**Example 4.** Let $P$ consist of the following rules:

\[
\square[3] p(X) \rightarrow \exists Y q(X, Y) \quad (13)
\]
\[
@ T q(X, Y) \land U = T + 1 \rightarrow @ U p(Y) \quad (14)
\]

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

\[
@ N p(X) \rightarrow \exists Y @ N q(X, Y) \quad (15)
\]

Intuitively, in $\text{wfree}(P)$, $N$ is the time at which rules are evaluated and localises all simple atoms to it; windows are removed and their restrictions relaxed: $\square[3] p$ (“at all times in window up to now”) becomes $\square[3] p$ (“now”); $\mathbb{P}^n @ T$ (“at $T$ if in window”) becomes $\mathbb{P}^n @ T$; and $\mathbb{P}^n @ T$ (“at some time in window”) becomes $\mathbb{P}^n @ T$ (“at some time”). As this logically weakens rule bodies, $\text{wfree}(P)$ 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 \( \text{rew}(\text{wfree}(P)) \) and \( \text{rew}(D) \), then it also terminates on \( \text{rew}(P) \) and \( \text{rew}(D) \).

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

Definition 4. The partial grounding \( \text{grnd}_A(P) \) 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 \( \langle B, B_A \rightarrow \exists z. H \rangle \sigma \), where \( B_A \) are the atoms in \( B \) with predicate in \( \mathcal{P}_A \), s.t. a rule \( B \rightarrow \exists z. H \in P \) and a homomorphism \( \sigma \) between \( B_A \) and \( A \) exist, i.e., a sort- and constant-preserving mapping \( \sigma : \text{dom}(B_A) \rightarrow \text{dom}(A) \) s.t. \( B_A \sigma \subseteq A \).

As long as \( A \) comprises all facts over \( \mathcal{P}_A \), \( \text{grnd}_A(P) \) 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 \( \text{wfree}(P) \) for a timeline \( T \), denoted \( \text{tgnd}_T(P) \), is the partial grounding \( \text{grnd}_{a(T)}(P) \) where

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

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

\[
\begin{align*}
[\square p] \land \square q] (X, 0, 0) & \rightarrow \exists Y. [\square q] (Y, X, 0) \quad (0 \leq 1) \\
[\square p] (X, 0, 1) & \rightarrow \exists Y. [\square q] (Y, X, 0) \quad (1 \leq 1) \\
[\square q] (X, 0, 0) & \rightarrow [\square p] (Y, 1) \quad (1 = 0 + 1)
\end{align*}
\]

While universal termination on \( \text{tgnd}_T(P) \), which can be recognized in Example 5 using e.g. MFA (Cuenca Grau et al. 2013), ensures chase termination on \( \text{rew}(P) \) and \( \text{rew}(D) \) for all data streams \( D \) on \( T \), 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 \( [\square q] p \) \( (s, 0, t) \) only, where \( t \) is a time point. Then \( \text{tfree}(P) \) is obtained by replacing each \( [\square q] p \) \( (s, 0, t) \) with \( [p] s \) for a fresh predicate \( p \) of proper signature.

Let \( \text{tfgrnd}_T(P) := \text{tfree}(\text{tgnd}_T(P)) \). The following result shows that this is a good basis to check for acyclicity.

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

Example 6 (cont’d). As \( \text{tfgrnd}_T(P) \) is WA, by Theorem 4 the skolem chase on \( \text{rew}(P) \) and \( \text{rew}(D) \) always terminates.

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

Theorem 5. \( \mathbf{L}_{\text{TLWA}}^+ \subseteq \mathbf{L}_{\text{TLWA}}^+(T) \) holds for all \( T \) s.t. \( |T| \geq 2 \).

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

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

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 \( t \) 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 parametrised 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 at (Urbani, Krötzsch, and Eiter 2022).

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 \( \approx 13 \text{ms} \) 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 non-monotonic 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


