A Formal Comparison between Datalog-based Languages for Stream Reasoning

Nicola Leone[1,†], Marco Manna[1,†], Maria Concetta Morelli[1,*,†] and Simona Perri[1,†]

[1]Department of Mathematics and Computer Science, University of Calabria, Rende, Italy

Abstract

The paper investigates the relative expressiveness of two logic-based languages for reasoning over streams, namely *LARS* Programs — the language of the Logic-based framework for Analytic Reasoning over Streams called *LARS* — and *LDSR* — the language of the recent extension of the *I-DLV* system for stream reasoning called *I-DLV-sr*. Although these two languages build over Datalog, they do differ both in syntax and semantics. To reconcile their expressive capabilities for stream reasoning, we define a comparison framework that allows us to show that, without any restrictions, the two languages are incomparable and to identify fragments of each language that can be expressed via the other one.

Keywords

Stream Reasoning, Datalog, Knowledge Representation and Reasoning, Relative Expressiveness

1. Introduction

Stream Reasoning (SR) [1] is a recently emerged research area that consists in the application of inference techniques to heterogeneous and highly dynamic streaming data. Stream reasoning capabilities are nowadays a key requirement for deploying effective applications in several real-world domains, such as IoT, Smart Cities, Emergency Management. Different SR approaches have been proposed in contexts such as Complex Event Processing, Semantic Web and Knowledge Representation and Reasoning (KRR) [2, 3, 4, 5]. In the KRR research field, the Answer Set Programming (ASP) declarative formalism [6, 7] has been acknowledged as a particularly attractive basis for SR [1] and a number of SR solutions relying on ASP have been recently proposed [8, 9, 10, 11, 12, 13, 4, 14, 15, 16]. Among these, *I-DLV-sr* [16] is a SR system that efficiently scale over real-world application domains thanks to a proper integration of the well-established stream processor *Apache Flink* [17] and the incremental ASP reasoner $\mathcal{I}^2$*-DLV* [18]. Its input language, called *LDSR* (the Language of *I-DLV* for Stream Reasoning) inherits the highly declarative nature and ease of use from ASP, while being extended with new constructs that are relevant for practical SR scenarios.

Although *LDSR* enjoys valuable knowledge modelling capabilities together with an efficient and effective reasoner, it would be desirable to formally investigate its expressive power. This is

Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy

*Corresponding author.

†These authors contributed equally.

nicola.leone@unical.it (N. Leone); marco.manna@unical.it (M. Manna); maria.morelli@unical.it (M. C. Morelli); simona.perri@unical.it (S. Perri)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3203/paper9.pdf

exactly the mission of the present paper. To this aim, the prime objective is to compare *LDSR* with one of the most famous and well-studied formalisms for reasoning over streams that goes under the name of *LARS Programs* (the language of *LARS*), where *LARS* is the Logic-based framework for Analytic Reasoning over Streams [8]. More precisely, the paper investigates the relative expressiveness of *LDSR* and *LARS* Programs. Despite the fact that these languages build over Datalog and represent the information via sets of ASP ground predicate atoms associated with different time points, unfortunately they overall differ both in syntax and semantics. In particular, *LDSR* associates information that is true at every time point with standard ASP facts, whereas *LARS* Programs involve background atoms whose truth is not directly associated with all the time points. Moreover, given an input stream, *LDSR* returns a single set of information related to the most recent time point (streaming model), whereas *LARS* Programs associate, for each different time point of the stream, another stream called answer stream at that time point.

To reconcile the expressive capabilities for stream reasoning of *LDSR* and *LARS* Programs, we first define a comparison framework that allows to understand in which cases, starting from the same input stream, both languages may produce the same output. Inside this framework, we identify three output profiles —called *atomic*, *bound*, and *full*— that fix the form of the output stream. Without any restriction on the two languages, the paper shows that they are incomparable under all the output profiles. Eventually, for each output profile, the paper isolates large fragments of each of the two languages that can be expressed via the other one.

2. Preliminaries

We assume to have finite sets $\mathcal{V}$, $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{U}$ consisting of *variables*, *constants*, *predicate names* and $time\ variables$ respectively; we constrain $\mathcal{V}$ and $\mathcal{C}$ to be disjoint. A *term* is either a variable in $\mathcal{V}$ or a constant in $\mathcal{C}$. A *predicate atom* has the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ is a predicate name, $t_1, \ldots, t_n$ are terms and $n \geq 0$ is the arity of the predicate atom; a predicate atom $p()$ of arity 0 can be also denoted by p. A predicate atom is *ground* if none of its terms is a variable. We denote by G the set of all ground predicate atoms constructible from predicate names in $\mathcal{P}$ and constants in $\mathcal{C}$. We divide the set of predicates $\mathcal{P}$ into two disjoint subsets, namely the extensional predicates $\mathcal{P}^{\varepsilon}$ and the intensional predicates $\mathcal{P}^{I}$. Extensional predicates are further partitioned into $\mathcal{P}^{\varepsilon}_{B}$ for background data and $\mathcal{P}^{\varepsilon}_{S}$ for data streams. The mentioned partitions are analogously defined for ground atoms G^{I},G^{ε}_{B} and G^{ε}_{S}. In what follows we will introduce different types of constructs peculiar to the two languages considered in this paper. In both, given a set of constructs C, we will denote by $pred(C)$ the set of predicates appearing in C.

A *stream* Σ is a sequence of sets of ground predicate atoms $\langle S_0, \ldots, S_n \rangle$ such that for $0 \leq i \leq n$, $S_i \subseteq G$. Each natural number i is called *time point*. A ground predicate atom $a \in S_i$ is true at the i-th time point. Given a value $m \in \mathbb{N}$ s.t. $0 \leq m \leq n$, we define the *restriction* of Σ to m the stream $\langle S_0, \ldots, S_m \rangle$ denoted with $\Sigma_{|_m}$. Moreover, let $F \subseteq \mathcal{P}$ we indicate with $\Sigma_{|_F}$ the stream $\langle S'_0, \ldots, S'_n \rangle$ with $S'_i = \bigcup_{\{a \in S_i | pred(a) \in F\}} a$ for each $i \in \{0, \ldots, n\}$. A subset of a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$ is a stream $\Sigma' = \langle S'_0, \ldots, S'_n \rangle$ such that $S'_i = \emptyset$ for each $i \notin t(\Sigma')$ where $t(\Sigma')$ is a subset of consecutive numbers of the set $\{0, \ldots, n\}$ and for each $i \in t(\Sigma')$ $S'_i \subseteq S_i$. For a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$, a *backward observation* identifies ground predicate atoms that are true at some time points preceding the n-th time point. More formally, given

a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$ and a set of numbers $D \subset \mathbb{N}$, we define the *backward observation of* Σ *w.r.t.* D as the family of sets $\{S_i \mid i = n - d \text{ with } d \in D \wedge i \geq 0\}$, and we denote it as $O(\Sigma, D)$. Given $w \in \mathbb{N}$, a backward observation of Σ w.r.t. $\{0, \ldots, w\}$ is called *window*.

2.1. *LARS* syntax and semantics

A *window function* is a function f_w that returns, given a stream Σ and a time point $t \in \{1, ..., n\}$, a substream of Σ. We consider only time-based window functions, which select all the atoms appearing in the last w time points, to which a window is trivially associated according to the definition above. Given a predicate atom a, a term $t \in N \cup \mathcal{U}$ and a window function f_w, formulas α are defined by the following grammar:

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \alpha \rightarrow \alpha \mid \diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus^{f_w} \alpha \mid \triangleright\alpha.$$

A *LARS* program P is a set of rules of the form $\alpha \leftarrow \beta_1, \ldots, \beta_n$, where $\alpha, \beta_1, \ldots, \beta_n$ are formulas. Given a rule r, we call α the *head* of r, denoted with $H(r)$, and we call the conjunction $\beta_1 \wedge \cdots \wedge \beta_n$ the *body* of r, denoted with $B(r)$. A ground formula can be satisfied at a time point t in a $structure$ that is a triple $M = (\Sigma, W, B)$ where Σ is a stream, W is a set of window functions and $B \subseteq G_B^\varepsilon$. Let $\Sigma' \subseteq \Sigma$, we start defining the entailment relation $\Vdash$ between (M, Σ', t) and formulas:

- $M, \Sigma', t \Vdash a$ iff $a \in S_t$ or $a \in B$
- $M, \Sigma', t \Vdash \neg\alpha$ iff $M, t \nVdash \alpha$
- $M, \Sigma', t \Vdash \alpha \wedge \beta$ iff $M, t \Vdash \alpha$ and $M, t \Vdash \beta$
- $M, \Sigma', t \Vdash \alpha \vee \beta$ iff $M, t \Vdash \alpha$ or $M, t \Vdash \beta$
- $M, \Sigma', t \Vdash \alpha \rightarrow \beta$ iff $M, t \nVdash \alpha$ or $M, t \Vdash \beta$
- $M, \Sigma', t \Vdash \diamond\alpha$ iff $M, \Sigma', t \Vdash \alpha$ for some $t' \in t(\Sigma')$
- $M, \Sigma', t \Vdash \Box\alpha$ iff $M, \Sigma', t \Vdash \alpha$ for all $t' \in t(\Sigma')$,
- $M, \Sigma', t \Vdash @_{t'}\alpha$ iff $M, \Sigma', t' \Vdash \alpha$ and $t' \in t(\Sigma')$,
- $M, \Sigma', t \Vdash \boxplus^{f_w}\alpha$, iff $M, \Sigma'', t \Vdash \alpha$ where $\Sigma'' = f_w(\Sigma', t)$,
- $M, \Sigma', t \Vdash \triangleright\alpha$ iff $M, \Sigma, t \Vdash \alpha$.

The structure $M = (\Sigma, W, B)$ satisfies α at time t ($M, t \models \alpha$) if $M, \Sigma', t \Vdash \alpha$. Given a ground *LARS* program P, a stream Σ and a structure M we say that: (i) M is a *model* of rule $r \in P$ for I at time t, denoted $M, t \models r$, if $M, t \models B(r) \rightarrow H(r)$; (ii) M is a *model* of P for I at time t, denoted $M, t \models P$, if $M, t \models r$ for all rules $r \in P$; (iii) M is a *minimal model*, if no model $M' = (\Sigma', W, B)$ of P for I at time t exists such that $\Sigma' \subset \Sigma$ and $t(\Sigma) = t(\Sigma')$; and (iv) The *reduct* of a program P with respect to M at time t is defined by $P^{M,t} = \{r \in P | M, t \models B(r)\}$. Fixed an input stream I, contains only atoms belong to G_S^ε, we call $interpretation\ stream\ for\ I$ any stream Σ such that all atoms that occur in Σ but not in I have intensional predicates. An interpretation stream Σ for a stream I is an $answer\ stream$ of a program $P\ for\ I\ at\ t$, if $M = (\Sigma, W, B)$ is a $\subseteq$-minimal model of the reduct $P^{M,t}$ for I at time t. The semantics of the non-ground programs is given by the answer streams of according groundings, obtained by replacing variables with constants from $\mathcal{C}$, respectively time points from $t(\Sigma)$, in all possible ways. We consider *LARS* programs with a single answer stream for each time point, denoted with *LARS*$_D$, and we indicate the single answer stream of P for I at t with $AS(P, I, t)$.

Table 1
Entailment of ground streaming literals.

α	$\Sigma \models \alpha$	$\Sigma \models \texttt{not}\ \alpha$
a **at least** c **in** $\{d_1, \ldots, d_m\}$	$\lvert\{A \in O(\Sigma, D) : a \in A\}\rvert \geq c$	$\lvert\{A \in O(\Sigma, D) : a \in A\}\rvert < c$
a **always in** $\{d_1, \ldots, d_m\}$	$\forall A \in O(\Sigma, D), a \in A$	$\exists A \in O(\Sigma, D) : a \notin A$
a **count** c **in** $\{d_1, \ldots, d_m\}$	$\lvert\{A \in O(\Sigma, D) : a \in A\}\rvert = c$	$\lvert\{A \in O(\Sigma, D) : a \in A\}\rvert \neq c$

2.2. *LDSR* syntax and semantics

Given a predicate atom a, a term $c \in \mathcal{C} \cap \mathbb{N}^+$, a *counting term* $t \in (\mathcal{C} \cap \mathbb{N}^+) \cup \mathcal{V}$, and a finite non-empty set $D = \{d_1, \ldots, d_m\} \subset \mathbb{N}$, we define three types of *streaming atoms*:

$$a\ \textbf{at least}\ c\ \textbf{in}\ \{d_1, \ldots, d_m\}\ \mid\ a\ \textbf{always in}\ \{d_1, \ldots, d_m\}\ \mid\ a\ \textbf{count}\ t\ \textbf{in}\ \{d_1, \ldots, d_m\}$$

In particular, if D is of the form $\{0, ..., w\}$, then this set can be alternatively written as $[w]$ inside streaming atoms; also we may write a in place of a **at least** 1 **in** $[0]$. A streaming atom α (resp., $\texttt{not}\ \alpha$) is said to be a *positive streaming literal* (resp., *negative streaming literal*), where not denotes *negation as failure*. A streaming literal is said to be *harmless* if it has form a **at least** c **in** D or a **always in** D; otherwise, it is said to be *non-harmless*. A streaming literal is said to be *ground* if none of its terms is a variable.

A *rule* is a formula of form $(1)\ \ a \texttt{:-}\ l_1, \ldots, l_b.$ or $(2)\ \ \#\textbf{temp}\ a \texttt{:-}\ l_1, \ldots, l_b.$, where a is a predicate atom, $b \geq 0$ and $l_1, \ldots, l_b$ represent a conjunction of literals (streaming literals or other literals defined in the ASP-Core-2 standard [19]).

For a rule r, we say that the *head* of r is the set $H(r) = \{a\}$, whereas the set $B(r) = \{l_1, \ldots, l_b\}$ is referred to as the *body* of r. A rule r is *safe* if all variables in $H(r)$ or in a negative streaming literal of $B(r)$ also appear in a positive streaming literal of $B(r)$.

A program P is a finite set of safe rules. We denote with $form_{(1)}(P)$ the set of rules of P of form (1) and with $form_{(2)}(P)$ the set of rules of P of form (2).

A program P is *stratified* if there is a partition of disjoint sets of rules $P = \Pi_1 \cup \cdots \cup \Pi_k$ (called strata) such that for $i \in \{1, \ldots, k\}$ both these conditions hold: (i) for each harmless literal in the body of a rule in Π_i with predicate p, $\{r \in P | H(r) = \{p(t_1, \ldots, t_n)\}\} \subseteq \bigcup_{j=1}^{i} \Pi_j$; (ii) for each non-harmless literal in the body of a rule in Π_i with predicate p, $\{r \in P | H(r) = \{p(t_1, \ldots, t_n)\}\} \subseteq \bigcup_{j=1}^{i-1} \Pi_j$. We call $\Pi_1, \ldots, \Pi_k$ a *stratification* for P and P is stratified by $\Pi_1, \ldots, \Pi_k$. An *LDSR* program is a program being also stratified.

A backward observation allows to define the truth of a ground streaming literal at a given time point. Given a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$, $D = \{d_1, \ldots, d_m\} \subset \mathbb{N}$, $c \in \mathcal{C} \setminus \{0\}$ and the backward observation $O(\Sigma, D)$, Table 1 reports when Σ *entails* a ground streaming atom α (denoted $\Sigma \models \alpha$) or its negation ($\Sigma \models \texttt{not}\ \alpha$). If $\Sigma \models \alpha$ ($\Sigma \models \texttt{not}\ \alpha$) we say that α is true (false) at time point n.

To make a comparison between *LARS*$_D$ and *LDSR*, we defined also for *LDSR* a model-theoretic semantics that can be shown to be equivalent to the operational semantics originally defined. Moreover, besides the concept of streaming model for *LDSR*, we defined the concept of answer stream.

Consider an *LDSR* program P. Given a rule $r \in P$, the *ground instantiation* $Gr(r)$ of r denotes the set of rules obtained by applying all possible substitutions σ from the variables in r to elements of $\mathcal{C}$. In particular, to the counting terms are applied only constants belong to $\mathcal{C} \cap \mathbb{N}^+$. Similarly, the *ground instantiation* $Gr(P)$ of P is the set $\bigcup_{r \in P} Gr(r)$. Given a ground rule $r \in P$, a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$ and a stream $\Sigma' = \langle S'_0, \ldots, S'_n \rangle$ such that $\Sigma \subseteq \Sigma'$, we say that Σ' is a model of r for Σ, denoted $\Sigma' \models r$, if $\Sigma' \models H(r)$ when $\Sigma' \models B(r)$. We say that Σ' is a *model* of P for Σ, denoted $\Sigma' \models P$, if $\Sigma' \models r$ for all rules $r \in Gr(P)$. Moreover, Σ' is a *minimal model*, if no model Ω of P exists such that $\Omega \subset \Sigma'$ and $t(\Omega) = t(\Sigma')$. Let Σ' be a model of P for Σ, an atom $a \in S'_n$ is *temporary* in Σ' if $a \notin S_n$ and there exists no rule $r \in form_{(1)}(P)$ such that $\Sigma' \models B(r)$ and $\Sigma' \models H(r)$. Accordingly, let $T(S'_n)$ be the set of all temporary atoms in S'_n, the stream $\langle S'_0, \ldots, S'_n \setminus T(S'_n) \rangle$ is called the *permanent part* of Σ'. Eventually, the *reduct* of P w.r.t. Σ', denoted by $P^{\Sigma'}$, consists of the rules $r \in Gr(P)$ such that $\Sigma' \models B(r)$.

Definition 1. *Given an LDSR program P, a stream $\Sigma = \langle S_0, \ldots, S_n \rangle$ and a stream $\Sigma' = \langle S'_0, \ldots, S'_n \rangle$ such that $\Sigma \subseteq \Sigma'$, Σ' is called* answer stream *and S'_n* streaming model *of P for Σ if:* (1) *if $n > 0$, $\langle S'_0 \rangle$ is the permanent part of the minimal model M of the reduct P^M for the stream $\langle S_0 \rangle$;* (2) *if $n > 0$, for all $i \in 1, \ldots, n-1$, $\Sigma'_{|i}$ is the permanent part of the minimal model M of the reduct P^M for the stream $\langle S'_0, \ldots, S'_{i-1}, S_i \rangle$; and* (3) *$\Sigma'$ is a minimal model of the reduct $P^{\Sigma'}$ for the stream $\langle S'_0, \ldots, S'_{n-1}, S_n \rangle$.*

Note that, differently from LARS, for which the information associated with each time point is entirely derived at the time point of evaluation, for *LDSR*, each time point t in the answer stream is associated with the information derived when the time point t has been evaluated. In other words, the answer stream for *LDSR* is obtained by collecting the results of the previous time points and integrating them with the result of the time point of evaluation.

3. Framework

In this section, we present the framework that has been designed for comparing the languages *LARS*$_D$ and *LDSR*. The comparison focuses on different parts of the answer stream. In particular, given an input stream $S = \langle S_0, \ldots, S_n \rangle$, when referring to an evaluation time point $t \leq n$, one could compare the answer streams only at t, or in all the time points up to t or also in all time points up to n. To this aim, we define three types of streams. Given $n \in \mathbb{N}$ and $t \in \{0, \ldots, n\}$, we say that a stream $S = \langle S_0, \ldots, S_n \rangle$ is of *type* t-*atomic* if $S_i = \emptyset$ for each $i \in \{0, \ldots, n\} \setminus \{t\}$; t-*bound* if $S_i = \emptyset$ for each $i \in \{t+1, \ldots, n\}$; and t-*full* if S_i may be nonempty for each $i \in \{0, \ldots, n\}$.

Consider a language $L \in \{$*LDSR*, *LARS*$_D\}$, an input stream $I = \langle I_0, \ldots, I_n \rangle$, a set of ground predicate atoms $B \subseteq G_B^{\epsilon}$ and a program $P \in L$, we call (I, B, P) an L-tuple. According to the three types of streams, for a L-tuple and for each time point $t \in \{0, \ldots, n\}$, we now define three types of output streams for each language L.

Given a *LDSR-tuple* (I, B, P) and a time point $t \in \{0, \ldots, n\}$, we define:

- $t\text{-}atomic(I, B, P) = \langle O_0, \ldots, O_n \rangle$ where $O_i = \emptyset$ for $i \neq t$ and O_t is the streaming model of $P \cup \{b. | b \in B\}$ on $I_{|_t} = \langle I_0, \ldots, I_t \rangle$.

- $t\text{-}bound(I, B, P) = \langle O_0, \ldots, O_n \rangle$, where $\langle O_0, \ldots, O_t \rangle$ is the answer stream of $P \cup \{b. | b \in B\}$ for $I_{|_t} = \langle I_0, \ldots, I_t \rangle$ and $O_i = \emptyset$ for $t < i \leq n$.
- $t\text{-}full(I, B, P) = \langle O_0, \ldots, O_n \rangle$ where $\langle O_0, \ldots, O_t \rangle$ is the answer stream of $P \cup \{b. | b \in B\}$ for $I_{|_t} = \langle I_0, \ldots, I_t \rangle$ and $O_i = I_i \cup B$ for $t < i \leq n$.

Analogously, given a *LARS$_D$*-$tuple$ (I, B, P) and a time point $t \in \{0, \ldots, n\}$, and the answer stream $AS(P, I, t) = \langle A_0, \ldots, A_n \rangle$, we define:

- $t\text{-}atomic(I, B, P) = \langle O_0, \ldots, O_n \rangle$ where $O_i = \emptyset$ for $i \neq t$ and $O_t = A_t \cup B$.
- $t\text{-}bound(I, B, P) = \langle O_0, \ldots, O_n \rangle$, where $O_i = A_i \cup B$ for $0 \leq i \leq t$ and $O_i = \emptyset$ for $t < i \leq n$.
- $t\text{-}full(I, B, P) = \langle O_0, \ldots, O_n \rangle$ where $O_i = A_i \cup B$ for $0 \leq i \leq n$.

We now define when a fragment of a language can be expressed in the other one in our framework. In particular, we differentiate expressible fragments from strictly expressible fragments. Given a stream form $\phi \in \{atomic, bound, full\}$,and $L_1, L_2 \in \{$*LDSR*, *LARS$_D$*$\}$ with $L_1 \neq L_2$, a fragment $F \subset L_1$ is $\phi\text{-}expressible$ via L_2 if there exists a mapping $\rho : F \to L_2$ such that, for each F-tuple (I, B, P) and for each time point of evaluation $t \in \{0, \ldots, n\}$, it holds that $t\text{-}\phi(I, B, P) = t\text{-}\phi(I, B, \rho(P))_{|_{pred(P \cup I \cup B)}}$; moreover, F is $strictly\ \phi\text{-}expressible$ via L_2 if $t\text{-}\phi(I, B, P) = t\text{-}\phi(I, B, \rho(P))$. Basically, for the non strict expressiveness, a translation into the other language is possible but it can involve the addition of auxiliary predicates, while for the strict one, there is a translation that does not require auxiliary predicates and thus for which the outputs coincide without the need of any filtering.

We are now ready to compare the two languages. The first result of the comparison is that, without any restrictions, the two languages are incomparable. The following two propositions describe the results. The ideas behind the formal demonstrations, which are instead reported in Appendix B [20], are also introduced.

Proposition 1. *LARS$_D$ is not $atomic\text{-}expressible$ via LDSR.*

To see this, consider the simple *LARS$_D$* program P_1=$\{@_{T-1}\ a \leftarrow\ @_T\ c.\}$. This program, that expresses that the presence of an atom c in a time point infers the presence of an atom a in the previous time point, is not expressible in *LDSR* since in its semantic the information associated at every time point are relative only to the information received and inferred up to it.

Proposition 2. *LDSR is not $atomic\text{-}expressible$ via LARS$_D$.*

To prove this result, consider the following *LDSR* program $P_2 = \{a(Y) :\text{-}\ a(X), b(X, Y).\}$ where the predicate a belongs to the input predicates $\mathcal{P}^{\varepsilon}_S$. The program P_2 is not expressible in *LARS$_D$* since its semantics avoids to infer ground atoms over input predicates.

Since $t\text{-}atomic(I, B, P) \neq t\text{-}atomic(I, B, \rho(P))$ implies $t\text{-}\phi(I, B, P) = t\text{-}\phi(I, B, \rho(P))$ holds for $\phi \in \{bound, full\}$, Proposition 1 and 2 imply the following result.

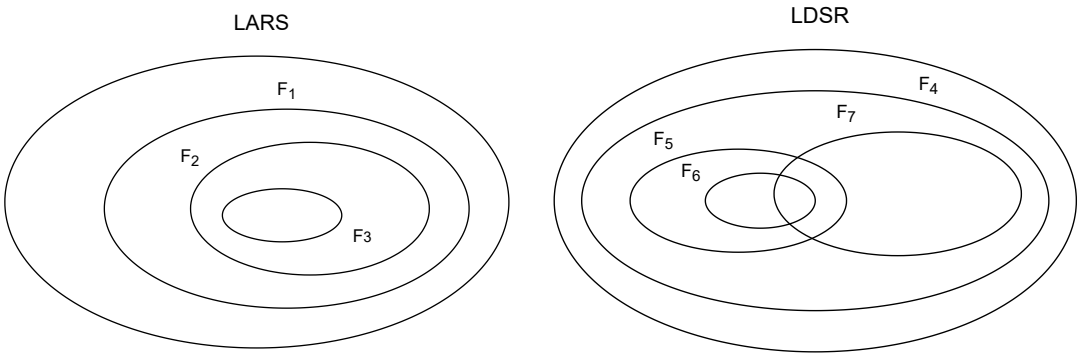
Theorem 1. *LARS$_D$ and LDSR are incomparable under each of the three stream forms.*

Table 2
LARS$_D$ to *LDSR*

ϕ	strictly	not strictly
atomic	F_1	F_1
bound	F_2	F_2
full	F_3	F_3

Table 3
LDSR to *LARS$_D$*

ϕ	strictly	not strictly
atomic	$F_6 \cup F_7$	F_4
bound	$F_6 \cup F_7$	F_4
full	F_6	F_5



(a) *LARS* fragments (b) *LDSR* fragments

Figure 1: Fragments and their relations

Given the incomparability of the languages, we introduced some restrictions to identify fragments of a language expressible in the other. Up to now, we identified seven fragments that are described in detail in sections 3.1 and 3.2. Here, we briefly discuss their relations and expressiveness (see Fig. 1 and Tables 2 and 3). In particular, Table 2 presents the fragments of *LARS$_D$* F_1, F_2 and F_3. All of them are strictly expressible via *LDSR*. F_1 is the largest identified fragment and it is atomic-expressible; F_2 is obtained from F_1 by imposing some restriction and it is bound-expressible, while F_3 is obtained by further restricting F_2, and allows for achieving full-expressivity.

As for *LDSR*, Table 3 summarizes its identified fragments F_4,F_5,F_6 and F_7. The largest fragment is F_4, which is bound-expressible via *LARS$_D$*; the fragment F_5, that restricts F_4, allows full expressiveness; the other two fragments allows for strictly expressiveness: F_6 further restricts F_5, and it is full-expressible, while F_7 is obtained by imposing restrictions on F_4 and is bound-expressible.

3.1. *LARS$_D$* to *LDSR*

Here we present the identified fragments of *LARS$_D$*. We consider two types of rules:

$$(I)\quad \Box(a \leftarrow \beta_1 \wedge \cdots \wedge \beta_b) \qquad \text{and} \qquad (II)\quad a \leftarrow \beta_1, \ldots, \beta_b$$

where a is an intensional predicate atom and $\beta_i \in \{\boxplus^m \Diamond p, \boxplus^m \Box p, p, \boxplus^0 @_T \top \wedge @_{T-K} p,$
$\neg \boxplus^m \Diamond p, \neg \boxplus^m \Box p, \neg p, \neg(\boxplus^0 @_T \top \wedge @_{T-K} p) | p$ is a predicate atom, $T \in \mathcal{U}$ and $m \in \mathbb{N}\}$

for $i \in \{1, \ldots, b\}$. For a rule of type (I), we denote as $cons(r)$ the consequent $\{a\}$ of the implication, and with $prem(r)$ the set of formulas $\{\beta_1, \ldots, \beta_b\}$ in the premise; moreover, for a rule of type (II), we denote with $head(r)$ the atom $\{a\}$ and with $body(r)$ the set of formulas $\{\beta_1, \ldots, \beta_b\}$. Let P be a *LARS*$_D$ program, we denote with $type_I(P)$ the set of rules of P of type (I) and with $type_{II}(P)$ the set of rules of P of type (II).
We say that a predicate p is *marked* if there are two rules $r \in type_I(P)$ and $r' \in type_{II}(P)$ with $p = pred(cons(r))$, $h \in pred(prem(r))$ and $h = pred(head(r'))$. The set of marked predicates of a program P is denoted with $M(P)$.

Consider a *LARS*$_D$ program P containing rules of type (I) and (II) only. Let $P_1 = type_I(P)$ and $P_2 = type_{II}(P)$. We define the graph $G(P) = \langle N, A \rangle$, where: (1) $N = (\cup_{r \in P_1} pred(cons(r))) \cup (\cup_{r \in P_2} pred(head(r)))$; (2) $(q, p, \text{“+”}) \in A$ if there exists a rule $r \in P_1$ with $pred(cons(r)) = p$ and $q \in pred(prem(r))$ occurring in a formula without negation or if there exists a rule $r' \in P_2$ with $pred(head(r')) = p$ and $q \in pred(body(r'))$ occurring in a formula without negation; and (3) $(q, p, \text{“-”}) \in A$ if there exists a rule $r \in P_1$ with $pred(cons(r)) = p$ and $q \in prem(r)$ occurs in a formula with negation or if there exists a rule $r' \in P_2$ with $pred(head(r')) = p$ and $q \in pred(body(r'))$ occurring in a formula with negation.

Definition 2. *Fragment F_1 of LARS$_D$ collects all the programs P that meet the next conditions: (i) $P = type_I(P) \cup type_{II}(P)$; (ii) $\cup_{r \in type_I(P)} pred(prem(r)) \cap M(P) = \emptyset$; (iii) $\cup_{r \in type_{II}(P)} pred(body(r)) \cap M(P) = \emptyset$; (iv) no cycle in $G(P)$ contains an arc labeled with “-”.*

Roughly, F_1 contains only programs that are stratified w.r.t negation, featuring only rules of types (I) and (II), where no marked predicate appears in premises and in bodies.

Proposition 3. *F_1 is strictly atomic-expressible via LDSR.*

Indeed, it can be shown that there is a mapping $\rho_1 : F_1 \to$ *LDSR* such that for each F_1-$tuple$ (I, B, P) and for each time point of evaluation $t \in \{0, \ldots, n\}$, it holds that $t\text{-}atomic(I, B, P) = t\text{-}atomic(I, B, \rho_1(P))$. In particular, given a program $P \in F_1$, the *LDSR* program $\rho_1(P)$ is obtained by replacing:

- each rule $\Box(a \leftarrow \beta_1, \ldots, \beta_m)$ of type (I) with the *LDSR* rule a :- $f(\beta_1), \ldots, f(\beta_m)$ of form (1)
- each rule $a \leftarrow \beta_1, \ldots, \beta_m$ of type (II) with the *LDSR* rule $\#\mathbf{temp}\ a$:- $f(\beta_1), \ldots, f(\beta_m)$ of form (2)

where f associates each *LARS*$_D$ formula in the F_1 fragment with a *LDSR* streaming atom as reported below:

- $f(\boxplus^m \Diamond p) = p$ **in** $[m]$.
- $f(\boxplus^m \Box p) = p$ **always in** $[m]$.
- $f(p) = p$.
- $f(\boxplus^0 @_T \top \wedge @_{T-K} p) = p$ **in** $\{k\}$.
- $f(\neg\beta) =$ `not` $f(\beta)$ where β is a formula.

Intuitively, the idea of the mapping ρ_1 is that *LARS*$_D$ rules of type (I) that must be evaluated at each time point are associated with $LDRS$ rules of form (1) and *LARS*$_D$ rules of type (II) that infer information only at the evaluation time point t are associate with *LDSR* rules of form (2) which are evaluated at each time point but the derivations of the time points preceding t have been forgotten via the $\#\mathbf{temp}$ operator. Moreover, we impose the restrictions (ii) and (iii) for defining F_1 in order to achieve atomic-expressiveness; indeed, they ensure that, if in *LDSR* a permanent information is derived relying on a temporary information, this is not used to derive other information.

Now, we define the fragment F_2 that imposes an additional restriction w.r.t to F_1, and the fragment F_3 that further restricts F_2.

Definition 3. *The fragment F_2 of LARS$_D$ is the subset of the programs of F_1 that meet the condition $(\bigcup_{r\in type_{II}(P)} pred(head(r))) \cap ((\bigcup_{r\in type_I(P)} pred(prem(r)) = \emptyset$.*

Proposition 4. *F_2 is strictly bound-expressible via LDSR.*

It can be shown that for the mapping $\rho_2 = \rho_{1|_{F_2}}$, it holds that for each F_2-$tuple$ (I, B, P) and for each time point of evaluation $t \in \{0, \ldots, n\}$, t-$bound(I, B, P) = t$-$bound(I, B, \rho_2(P))$. Basically, the additional condition for fragment F_2 avoids that a temporary information associated to a time point can generate a permanent information.

We now show an example of a program belonging to fragment F_2 and its image with respect to the function ρ_2. The example is taken from one of the tasks of the "model and solve" Stream Reasoning Hackathon 2021 [21]. The task concerns urban traffic management. Traffic is observed from a top-down, third-person perspective, and vehicle movement flows in a given road network coded as Datalog facts are considered. We want to identify the vehicles that appear or disappear in the network. It is then necessary to note vehicles that were absent at the previous time point and are now present and vice versa. A rule of type (I) is used to evaluate the presence of vehicles at the current and previous time points; then the obtained information is used in a rule of type (II) that detects the appearance or disappearance of a vehicle at the evaluation time point. This task can be modelled via the following program P in F_2:

$$\Box(inNetwork(Veh) \leftarrow onLane(Veh, X, Y)).$$
$$appears(Veh) \leftarrow onLane(Veh, X, Y), \neg \boxplus^0 @_T \top \wedge @_{T-1} inNetwork(Veh).$$
$$disappears(Veh) \leftarrow \boxplus^0 @_T \top \wedge @_{T-1} inNetwork(Veh), \neg inNetwork(Veh).$$

The corresponding *LDSR* program, image of the ρ_2 function, is:

$$inNetwork(Veh) \text{ :- } onLane(Veh, X, Y).$$
$$\#\mathbf{temp}\ appears(Veh) \text{ :- } onLane(Veh, X, Y), \texttt{not}\ inNetwork(Veh)\ \mathbf{in}\ \{1\}.$$
$$\#\mathbf{temp}\ disappears(Veh) \text{ :- } onLane(Veh, X, Y)\ \mathbf{in}\ \{1\}, \texttt{not}\ inNetwork(Veh).$$

Definition 4. *The fragment F_3 of LARS$_D$ is the subset of the programs of F_2 that meet the condition $P = type_{II}(P)$.*

Proposition 5. *F_3 is strictly full-expressible via LDSR.*

Similarly to F_2, it can be shown that, considering the mapping $\rho_3 = \rho_{1|_{F_3}}$, it holds that that for each F_3-$tuple$ (I, B, P) and for each time point of evaluation $t \in \{0, \ldots, n\}$, t-$full(I, B, P) = t$-$full(I, B, \rho_3(P))$. Intuitively, since in *LDSR* the evaluation of a program with respect to a time point t can not add information in the output stream at time points that are subsequent to t, to achieve full expressiveness, we impose the restriction to rules of type type (II) in *LARS$_D$* as these are evaluated only at t and do not change the output in the subsequent time points.

3.2. *LDSR* to *LARS$_D$*

Here we present the identified fragments of *LDSR* along with their expressiveness. While, for the fragments of *LARS$_D$* we obtained strict expressiveness and each rule in *LARS$_D$* has been translated into exactly one rule in *LDSR*, for the fragments of *LDSR*, the translation, in general, needs auxiliary atoms and additional rules to simulate the behavior of rules of the form (2) and of some streaming and aggregates atoms.

The largest identified fragment is F_4 that is defined as follows.

Definition 5. *The fragment F_4 is the subset of the LDSR programs P that meet the condition $\bigcup_{r \in P} pred(H(r)) \subset \mathcal{P}^I$.*

Basically, F_4 is obtained from *LDSR* by simply imposing that no extensional predicate appears in the head of a rule.

Proposition 6. *F_4 is bound-expressible via LARS$_D$.*

It can be shown that there is a mapping $\rho_4 : F_4 \rightarrow$ *LARS$_D$* such that for each F_4-$tuple$ (I, B, P) and for each $t \in \{0, \ldots, n\}$, t-$bound(I, B, P) = t$-$bound(I, B, \rho_4(P))_{|_{pred(P \cup I \cup B)}}$. First, we note that, in general, each streaming atom in F_4 has to properly translated into a *LARS$_D$* formula; moreover a special rewriting, requiring additional rules, has to be performed for streaming atoms of the form a **count** v **in** $\{d_1, \ldots, d_m\}$, where v is a variable in $\mathcal{C}$ and for all the aggregate atoms. Thus, without going into details, the mapping ρ_4 relies on a function g that associates each streaming atom (but those of form a **count** v **in** $\{d_1, \ldots, d_m\}$) with a *LARS$_D$* formula that expresses the condition that must be satisfied in the stream for the streaming atom to be true; moreover, if α is an aggregate atom or a streaming atom of the form a **count** v **in** $\{d_1, \ldots, d_m\}$, g associates it with a formula containing auxiliary atoms defined via an additional set of rules C_α that are needed to simulate its semantics. For the sake of the presentation, the function g and the set of additional rules C_α are reported in Appendix A.1 [20].

Furthermore, given a program $P \in F_4$, the mapping ρ_4 has to replace each rule r in P with one or more *LARS$_D$* rules. In sum, the program $\rho_4(P)$ is obtained by:

- replacing each rule a :- $\beta_1, \ldots, \beta_m$ of form (1) with the *LARS$_D$* rule $\Box(a \leftarrow \boxplus^0 @_T \top \wedge g(\beta_1) \wedge \cdots \wedge g(\beta_m))$.
- for each rule $\#\mathbf{temp}\ a$:- $\beta_1, \ldots, \beta_m$ of form (2),
  - replacing it with the *LARS$_D$* rule $a \leftarrow \boxplus^0 @_T \top, g(\beta_1), \ldots, g(\beta_m)$

- adding the rule $\Box(atemp \leftarrow \boxplus^0@_T\top \wedge g(\beta_1) \wedge \cdots \wedge g(\beta_m))$.

- adding for each streaming atom α of the form a **count** v **in** $\{d_1, \ldots, d_m\}$, where v is a variable in $\mathcal{C}$ or aggregates atom, a set of rules C_α.

Basically, the mapping ρ_4 manages the interaction between the two forms of rules, simulating, at an evaluation time point t, the temporary derivations obtained in the previous time points and evaluating their effect on the permanent derivations that will be part of the output. This is mainly obtained by creating and handling a copy of each rule of the form (2) where the suffix "temp" is added to the head predicate. Moreover, since the streaming atoms in *LDSR* are evaluated according to the backward observation, we need to identify the reference time point in which the $LARS$ formulas representing the conditions expressed by the streaming atoms have to checked. To do this, we use the formula $\boxplus^0@_T\top$ that evaluates the tautology within a window of size 0 and thus, it holds in the rule instance where the variable T corresponds to the reference time point.

We are now ready to define the fragment F_5 that is obtained from F_4 by avoiding rules of form (1).

Definition 6. *The fragment F_5 of LDSR is the subset of the programs of F_4 that meet the condition $P = form_2(P)$.*

Proposition 7. *F_5 is full-expressible via LARS$_D$.*

To see this, consider, the mapping $\rho_5 : F_5 \rightarrow$ *LARS$_D$* such that, for each program $P \in F_5$, $\rho_5(P)$ is obtained by:

- replacing each rule $\#\mathbf{temp}\ a \text{ :- } \beta_1, \ldots, \beta_m$ of form (2) , with the *LARS$_D$* rule $a \leftarrow \boxplus^0@_T\top, g'(\beta_1), \ldots, g'(\beta_m)$.
- adding for each streaming atom α of the form a **count** v **in** $\{d_1, \ldots, d_m\}$, where v is a variable in $\mathcal{C}$ or aggregates atom, a set of rules C_α.

It can be shown that ρ_5 is such that for each F_5-$tuple$ (I, B, P) and for each $t \in \{0, \ldots, n\}$, t-*full*$(I, B, P) = t$-*full*$(I, B, \rho_5(P))_{|_{pred(P \cup I \cup B)}}$. Roughly, similarly to ρ_4, ρ_5 relies on a function g' for rewriting body atoms and adds auxiliary rules for handling aggregate atoms or a streaming atoms of the form a **count** v **in** $\{d_1, \ldots, d_m\}$ (more details on this are reported in Appendix A.2 [20]); however the translation for the fragment F_5 is simpler than the one for F_4, as it is sufficient to associate each rule of form (2) with a *LARS$_D$* rule of type (II). Indeed, since rules of form (1) are not allowed, there is no need to consider the information that can be derived in a permanent way through them. The condition defining the fragment F_5 ensures the full expressiveness: since a program in this fragment features only rules of form (2), and its translation only rules of type (II), their evaluation at each time point t can derive information only at t, while leaving unchanged the output in the other time points.

The fragment F_6 restrict F_5 to reach strict full expressiveness.

Definition 7. *The fragment F_6 of LDSR is the subset of the programs of F_5 in which streaming atoms of the form a **count** v **in** $\{d_1, \ldots, d_m\}$ where $v \in V$ and aggregate atoms are not allowed in rule bodies.*

Proposition 8. *F_6 is strictly full-expressible via LARS$_D$.*

It can be shown that for the mapping $\rho_6 = \rho_{5|_{F_6}}$, it holds that for each F_6-$tuple$ (I, B, P) and for each $t \in \{0, \ldots, n\}$, t-$full(I, B, P) = t$-$full(I, B, \rho_6(P))$. Since no atoms involving the addition of auxiliary predicates are considered, F_6 is strictly expressible.

The last considered fragment F_7 still features strict expressiveness, but of bound type, as, differently from F_6, it allows also rules of form (1) to some extent.

Definition 8. *The fragment F_7 of LDSR is the subset of the programs of F_4 that meet the following conditions: (i) $(\cup_{r \in form_2(P)}(pred(B(r))) \cap (\cup_{r \in form_2(P)}(pred(H(r))) = \emptyset$ (ii) streaming atoms of the form a **count** v **in** $\{d_1, \ldots, d_m\}$ where $v \in V$ and aggregate atoms are not allowed in rule bodies.*

Proposition 9. *F_7 is strictly bound-expressible via LARS$_D$.*

It can be shown that there is a mapping $\rho_7 : F_7 \rightarrow$ *LARS$_D$* such that for each F_7-$tuple$ (I, B, P) and for each $t \in \{0, \ldots, n\}$, t-$bound(I, B, P) = t$-$bound(I, B, \rho_7(P))$. To see this, consider, the mapping $\rho_7 : F_7 \rightarrow$ *LARS$_D$* such that, for each program $P \in F_7$, $\rho_7(P)$ is obtained by:

- replacing each rule $\#\mathbf{temp}\ a \text{ :- } \beta_1, \ldots, \beta_m$ of form (2) with the *LARS$_D$* rule $a \leftarrow \boxplus^0 @_T \top, g'(\beta_1), \ldots, g'(\beta_m)$.
- replacing each rule $a \text{ :- } \beta_1, \ldots, \beta_m$ of form (1) with the *LARS$_D$* rule $\Box(a \leftarrow \boxplus^0 @_T \top \wedge g''(\beta_1) \wedge \cdots \wedge g''(\beta_m))$

The mapping relies on the same function g' as ρ_5 for the rules of form (2). In addition, for the rule of form (1), a different function g'' is used to associate the body streaming atoms with $LARS$ formulas based on the following definition.

Definition 9. *Given an atom $a(t_1, \ldots, t_n)$ and a LDSR rule r with $H(r) = a(t'_1, \ldots, t'_n)$ we call definition of $a(t_1, \ldots, t_n)$ in r, denoted with $d_r(a(t_1, \ldots, t_n))$, the conjunction $\bigwedge_{\{\beta \in B(r)\}} \beta \bigwedge_{1 \leq i \leq n} t_i = t'_i$. Given an LDSR program P the definition of $a(t_1, \ldots, t_n)$ in P is $d_P(a(t_1, \ldots, t_n)) = a(t_1, \ldots, t_n) \vee (\bigvee_{\{r \in P | pred(H(r)) = a\}} d_r(a(t_1, \ldots, t_n)))$.*

This definition identifies, for each predicate a in the head of a rule of form (2), a $LARS$ formula relying only on permanent information that can be used in the $LARS$ translation in place of a. Further details on the translation and the g'' functions are reported in Appendix A.3 [20]. We note here that the strict expressiveness of this fragment is obtained since the translation of the allowed streaming atoms does not require the use of additional atoms, and condition (ii) simplifies the rewriting of the rules of form (2) w.r.t what fragment F_4. Indeed, in this case, it is not necessary to add the rules used by ρ_4 such as $\Box(atemp \leftarrow \boxplus^0 @_T \top \wedge g(\beta_1) \wedge \cdots \wedge g(\beta_m))$ that required additional auxiliary atoms.

Consider, for example, the *LDSR* program P' that could be used for monitoring irregularity in a subway station monitoring system. Three minutes are expected to elapse between the arrival of one train and the next, so the program records an irregularity when one train passes and another has already passed in one of the previous two minutes:

$$P' = \{irregular \text{ :- } train\_pass, train\_pass\ \mathbf{at\ least}\ 1\ \mathbf{in}\ \{1, 2\}.\}$$

The program belongs to the F_7 fragment, and the corresponding $LARS_D$ program with respect to the ρ_7 function is as follows:

$$\rho_7(P') = \{\Box(irregular \leftarrow \boxplus^0 @_T \top \wedge (@_{T_1} train\_pass \wedge T_1 = T - 0) \\ \wedge (@_{T_2} train\_pass \wedge ((T_2 = T - 1) \vee (T_2 = T - 2))).\}$$

4. Conclusion

This work presents a formal comparison about the relative expressiveness of the two languages *LDSR* and *LARS*. The main contribution of the work is twofold: (i) we propose a suitable framework to compare the two languages, which exhibit different syntax and semantics. and (ii) for each language, we identify a number of fragments that can be expressed by the other one, showing possible rewritings. In order to compare the semantics of the two languages, we first provided an alternative equivalent model-theoretic definition of the semantics of *LDSR*, instead of the operational one originally provided. Moreover, we defined the concept of answer stream also for *LDSR*, as an extension of the streaming model. The framework allowed us for focusing the comparison on different forms of the output stream (atomic, bound, full) and on the nature of the rewriting that could forbid or admit the addition of auxiliary predicates (strict or not strict expressiveness, respectively). For each given form of output and type of rewriting, we studied how to build fragments of a language that could meet the desired expressiveness. To do this, we considered the semantics behind each construct or combination of constructs that can occur in the rules and the effect of interactions between the different rules. The fragments $F_1, ..., F_7$ are the largest we identified so far, but, of course, these could be further enlarged and new ones could be possibly found; this will be the subject of future works.

Acknowledgments

This work has been partially supported by the project “MAP4ID - Multipurpose Analytics Platform 4 Industrial Data”, N. F/190138/01-03/X44 and by the Italian MIUR Ministry and the Presidency of the Council of Ministers under the project “Declarative Reasoning over Streams" under the “PRIN" 2017 call (CUP $H24I17000080001$, project 2017M9C25L_001).

References

[1] D. Dell’Aglio, E. D. Valle, F. van Harmelen, A. Bernstein, Stream reasoning: A survey and outlook, Data Sci. 1 (2017) 59–83. URL: https://doi.org/10.3233/DS-170006. doi:10.3233/DS-170006.

[2] D. F. Barbieri, D. Braga, S. Ceri, E. D. Valle, M. Grossniklaus, C-SPARQL: a continuous query language for RDF data streams, Int. J. Semantic Comput. 4 (2010) 3–25. URL: https://doi.org/10.1142/S1793351X10000936. doi:10.1142/S1793351X10000936.

[3] J. Hoeksema, S. Kotoulas, High-performance distributed stream reasoning using s4, in: Ordring Workshop at ISWC, 2011.

[4] T. Pham, M. I. Ali, A. Mileo, C-ASP: continuous asp-based reasoning over RDF streams, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings, volume 11481 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 45–50. doi:10.1007/978-3-030-20528-7_4.

[5] D. L. Phuoc, M. Dao-Tran, J. X. Parreira, M. Hauswirth, A native and adaptive approach for unified processing of linked streams and linked data, in: L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. F. Noy, E. Blomqvist (Eds.), The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I, volume 7031 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 370–388. URL: https://doi.org/10.1007/978-3-642-25073-6_24. doi:10.1007/978-3-642-25073-6_24.

[6] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. doi:10.1145/2043174.2043195.

[7] M. Gebser, N. Leone, M. Maratea, S. Perri, F. Ricca, T. Schaub, Evaluation techniques and systems for answer set programming: a survey, in: J. Lang (Ed.), Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden, ijcai.org, 2018, pp. 5450–5456. URL: https://doi.org/10.24963/ijcai.2018/769. doi:10.24963/ijcai.2018/769.

[8] H. Beck, M. Dao-Tran, T. Eiter, LARS: A logic-based framework for analytic reasoning over streams, Artif. Intell. 261 (2018) 16–70. URL: https://doi.org/10.1016/j.artint.2018.04.003. doi:10.1016/j.artint.2018.04.003.

[9] H. R. Bazoobandi, H. Beck, J. Urbani, Expressive stream reasoning with laser, in: C. d'Amato, M. Fernández, V. A. M. Tamma, F. Lécué, P. Cudré-Mauroux, J. F. Sequeda, C. Lange, J. Heflin (Eds.), The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I, volume 10587 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 87–103. URL: https://doi.org/10.1007/978-3-319-68288-4_6. doi:10.1007/978-3-319-68288-4_6.

[10] H. Beck, M. Dao-Tran, T. Eiter, C. Folie, Stream reasoning with LARS, Künstliche Intell. 32 (2018) 193–195. URL: https://doi.org/10.1007/s13218-018-0537-9. doi:10.1007/s13218-018-0537-9.

[11] T. Eiter, P. Ogris, K. Schekotihin, A distributed approach to LARS stream reasoning (system paper), Theory Pract. Log. Program. 19 (2019) 974–989. URL: https://doi.org/10.1017/S1471068419000309. doi:10.1017/S1471068419000309.

[12] X. Ren, O. Curé, H. Naacke, G. Xiao, Bigsr: real-time expressive RDF stream reasoning on modern big data platforms, in: N. Abe, H. Liu, C. Pu, X. Hu, N. K. Ahmed, M. Qiao, Y. Song, D. Kossmann, B. Liu, K. Lee, J. Tang, J. He, J. S. Saltz (Eds.), IEEE International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, December 10-13, 2018, IEEE, 2018, pp. 811–820. URL: https://doi.org/10.1109/BigData.2018.8621947. doi:10.1109/BigData.2018.8621947.

[13] A. Mileo, A. Abdelrahman, S. Policarpio, M. Hauswirth, Streamrule: A nonmonotonic stream reasoning system for the semantic web, in: W. Faber, D. Lembo (Eds.), Web Reasoning and Rule Systems - 7th International Conference, RR 2013, Mannheim, Germany, July 27-29, 2013. Proceedings, volume 7994 of *Lecture Notes in Computer Sci-*

ence, Springer, 2013, pp. 247–252. URL: https://doi.org/10.1007/978-3-642-39666-3_23. doi:10.1007/978-3-642-39666-3_23.

[14] T. M. Do, S. W. Loke, F. Liu, Answer set programming for stream reasoning, in: C. J. Butz, P. Lingras (Eds.), Advances in Artificial Intelligence - 24th Canadian Conference on Artificial Intelligence, Canadian AI 2011, St. John's, Canada, May 25-27, 2011. Proceedings, volume 6657 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 104–109. URL: https://doi.org/10.1007/978-3-642-21043-3_13. doi:10.1007/978-3-642-21043-3_13.

[15] M. Gebser, T. Grote, R. Kaminski, T. Schaub, Reactive answer set programming, in: J. P. Delgrande, W. Faber (Eds.), Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings, volume 6645 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 54–66. URL: https://doi.org/10.1007/978-3-642-20895-9_7. doi:10.1007/978-3-642-20895-9_7.

[16] F. Calimeri, M. Manna, E. Mastria, M. C. Morelli, S. Perri, J. Zangari, I-dlv-sr: A stream reasoning system based on I-DLV, Theory Pract. Log. Program. 21 (2021) 610–628. URL: https://doi.org/10.1017/S147106842100034X. doi:10.1017/S147106842100034X.

[17] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink™: Stream and batch processing in a single engine, IEEE Data Eng. Bull. 38 (2015) 28–38. URL: http://sites.computer.org/debull/A15dec/p28.pdf.

[18] G. Ianni, F. Pacenza, J. Zangari, Incremental maintenance of overgrounded logic programs with tailored simplifications, Theory Pract. Log. Program. 20 (2020) 719–734. URL: https://doi.org/10.1017/S147106842000040X. doi:10.1017/S147106842000040X.

[19] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, Asp-core-2 input language format, TPLP 20 (2020) 294–309. URL: https://doi.org/10.1017/S1471068419000450. doi:10.1017/S1471068419000450.

[20] N. Leone, M. Manna, M. C. Morelli, S. Perri, A formal comparison between datalog-based languages for stream reasoning (extended version), 2022. URL: https://arxiv.org/abs/2208.12726. doi:10.48550/ARXIV.2208.12726.

[21] P. Schneider, D. Alvarez-Coello, A. Le-Tuan, M. N. Duc, D. L. Phuoc, Stream reasoning playground, in: P. Groth, M. Vidal, F. M. Suchanek, P. A. Szekely, P. Kapanipathi, C. Pesquita, H. Skaf-Molli, M. Tamper (Eds.), The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings, volume 13261 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 406–424. URL: https://doi.org/10.1007/978-3-031-06981-9_24. doi:10.1007/978-3-031-06981-9_24.