Hypothetical Answers to Continuous Queries over Data Streams

Luís Cruz-Filipe
Dept. of Mathematics and Computer Science,
University of Southern Denmark

Graça Gaspar and **Isabel Nunes**
University of Lisbon, Faculty of Sciences,
BioISI - Biosystems & Integrative Sciences Institute,
Portugal

Abstract

Continuous queries over data streams often delay answers until some relevant input arrives through the data stream. These delays may turn answers, when they arrive, obsolete to users who sometimes have to make decisions with no help whatsoever. Therefore, it can be useful to provide hypothetical answers – "given the current information, it is possible that X will become true at time t" – instead of no information at all.

In this paper we present a semantics for queries and corresponding answers that covers such hypothetical answers, together with an online algorithm for updating the set of facts that are consistent with the currently available information.

1 Introduction

Modern-day reasoning systems often have to react to real-time information about the real world provided by e.g. sensors. This information is typically conceptualized as a data stream, which is accessed by the reasoning system. The reasoning tasks associated to data streams – usually called *continuous queries* – are expected to run continuously and produce results through another data stream in an online fashion, as new elements arrive.

A data stream is a potentially unbounded sequence of data items generated by an active, uncontrolled data source. Elements arrive continuously at the system, potentially unordered, and at unpredictable rates. Thus, reasoning over data streams requires dealing with incomplete or missing data, potentially storing large amounts of data (in case it might be needed to answer future queries), and providing answers in timely fashion – among other problems, see e.g. (Babcock et al. 2002; Stonebraker, Çetintemel, and Zdonik 2005; Dell'Aglio et al. 2017).

The output stream is normally ordered by time, which implies that the system may have to delay appending some answer because of uncertainty in possible answers relating to earlier time points. The length of this delay may be unpredictable (*unbound wait*) or infinite, for example if the query uses operators that range over the whole input data stream (*blocking operations*). In these cases, answers that have been computed may never be output. An approach to avoid this problem is to restrict the language by forbidding blocking operations (Zaniolo 2012; Ronca et al. 2018a). Another approach uses the concept of reasoning window (Beck et al. 2015; Özçep, Möller, and Neuenstadt 2014), which bounds the size of the input that can be used for computing each output (either in time units or in number of events).

In several applications, it is useful to know that some answers are likely to be produced in the future, since there is already some information that might lead to their generation. This is the case namely in prognosis systems (e.g., medical diagnosis, stock market prediction), where one can prepare for the possibility of something happening. To this goal, we propose *hypothetical answers*: answers that are supported by information provided by the input stream, but that still depend on other facts being true in the future. Knowledge about both the facts that support the answer and possible future facts that may make it true gives users the possibility to make timely, informed decisions in contexts where preemptive measures may have to be taken.

Moreover, by giving such hypothetical answers to the user we cope with unbound wait in a constructive way, since the system is no longer "mute" while waiting for an answer to become definitive.

Many existing approaches to reasoning with data streams adapt and extend models, languages and techniques used for querying databases and the semantic web (Arasu, Babu, and Widom 2006; Barbieri et al. 2009). We develop our theory in line with the works of (Zaniolo 2012; Beck et al. 2015; Dao-Tran and Eiter 2017; Özçep, Möller, and Neuenstadt 2014; Ronca et al. 2018b), where continuous queries are treated as rules of a logic program that reasons over facts arriving through a data stream.

Contribution. We present a declarative semantics for queries in Temporal Datalog, where we define the notions of hypothetical and supported answers. We define an operational semantics based on SLD-resolution, and show that there is a natural connection between the answers computed by this semantics and hypothetical and supported answers. By refining SLD-resolution, we obtain an online algorithm for maintaining and updating the set of answers that are consistent with the currently available information. Finally, we show that our results extend to a language with negation.

Structure. Section 2 revisits some fundamental background notions and introduces the running example that we use throughout this article. Section 3 introduces our declarative semantics for continuous queries, defining hypothetical and supported answers, and relates these concepts with the standard definitions of answers. Section 4 presents our operational semantics for continuous queries and relates it to the declarative semantics. Section 5 details our online algorithm to compute supported answers incrementally, as input facts arrive through the data stream, and proves it sound and complete. Section 6 extends our framework to negation by failure. Section 7 compares our proposal to similar ones in the literature, discussing its advantages. Missing proofs and additional examples can be found in (Cruz-Filipe, Gaspar, and Nunes 2019).

2 Background

Continuous queries in Temporal Datalog

We use the framework from (Ronca et al. 2018b) to write continuous queries over datastreams, slightly adapting some definitions. We work in *Temporal Datalog*, the fragment of negation-free Datalog extended with the special temporal sort from (Chomicki and Imielinski 1988), which is isomorphic to the set of natural numbers equipped with addition with arbitrary constants. In Section 6 we extend this language with negation.

Syntax of Temporal Datalog. A vocabulary consists of constants (numbers or identifiers in lowercase), variables (single uppercase letters) and predicate symbols (identifiers beginning with an uppercase letter). All these may be indexed if necessary; occurrences of predicates and variables are distinguished by context. In examples, we use words in sans serif for concrete constants and predicates.

Constants and variables have one of two sorts: *object* or *temporal*. An *object term* is either an object (constant) or an object variable. A *time term* is either a natural number (called a *time point* or *temporal constant*), a time variable, or an expression of the form $T+\mathsf{k}$ where T is a time variable and k is an integer.

Predicates can take one temporal parameter, which we assume to be the last one (if present). A predicate with no temporal parameters is called *rigid*, otherwise it is called *temporal*. An atom is an expression $P(t_1, \ldots, t_n)$ where P is a predicate and each t_i is a term of the expected sort.

A rule has the form $\wedge_i \alpha_i \rightarrow \alpha$, where α and each α_i are rigid or temporal atoms. Atom α is called the *head* of the rule, and $\wedge_i \alpha_i$ the *body*. Rules are assumed to be *safe*: each variable in the head must occur in the body. A *program* is a set of rules.

A predicate symbol that occurs in an atom in the head of a rule with non-empty body is called *intensional* (IDB predicate). Predicates that are defined only through rules with empty body are called extensional (EDB predicates). An atom is extensional (EDB atom) or intensional (IDB atom) according to whether P is extensional or intensional.

A term, atom, rule, or program is *ground* if it contains no variables. We write $\mathsf{var}(\alpha)$ for the set of variables occurring in an atom, and extend this function homomorphically to rules and sets. A *fact* is a function-free ground atom; since Temporal Datalog does not allow function symbols except in temporal terms, every ground rigid atom is a fact.

Rules are instantiated by means of *substitutions*, which are functions mapping variables to terms of the expected sort. The *support* of a substitution θ is the set $\mathsf{supp}(\theta) = \{X \mid \theta(X) \neq X\}$. We consider only substitutions with finite support, and write $\theta = [X_1 := t_1, \ldots, X_n := t_n]$ for the substition mapping each variable X_i to the term t_i, and leaving all remaining variables unchanged. A substitution is *ground* if every variable in its support is mapped to a constant. An *instance* $r' = r\theta$ of a rule r is obtained by simultaneously replacing every variable X in r by $\theta(X)$ and computing any additions of temporal constants.

A *query* is a pair $Q = \langle P, \Pi \rangle$ where Π is a program and P is an IDB atom in the language underlying Π. (Note that we do not require P to be ground.)

A *dataset* is a set of EDB facts (*input facts*), intuitively produced by a data stream. For each dataset D and time point τ, we consider D's *τ-history*: the dataset D_τ of the facts produced by D whose temporal argument is at most τ. By convention, $D_{-1} = \emptyset$.

Semantics. The semantics of Temporal Datalog is a variant of the standard semantics based on Herbrand models. A Herbrand interpretation I for Temporal Datalog is a set of facts. If α is an atom with no variables, then we define $\bar{\alpha}$ as the fact obtained from α by evaluating each temporal term. In particular, if α is rigid, then $\bar{\alpha} = \alpha$. We say that I satisfies α, $I \models \alpha$, if $\bar{\alpha} \in I$. The extension of the notion of satisfaction to the whole language follows the standard construction, and the definition of entailment is the standard one.

An *answer* to a query $Q = \langle P, \Pi \rangle$ over a dataset D is a ground substitution θ whose domain is the set of variables in P, satisfying $\Pi \cup D \models P\theta$. In the context of continuous query answering, we are interested in the case where D is a τ-history of some data stream, which changes with time. We denote the set of all answers to Q over D_τ as $\mathcal{A}(Q, D, \tau)$.

We use a subset of Example 1 in (Ronca et al. 2018b) as running example throughout our paper.

Example 1 *A set of wind turbines are scattered throughout the North Sea. Each turbine has a sensor that sends temperature readings* $\mathsf{Temp}(Device, Level, Time)$ *to a data centre. The data centre tracks activation of cooling measures in each turbine, recording malfunctions and shutdowns by means of the following program* Π_E.

$$\begin{aligned} \mathsf{Temp}(X, \mathsf{high}, T) &\rightarrow \mathsf{Flag}(X, T) \\ \mathsf{Flag}(X, T) \wedge \mathsf{Flag}(X, T+1) &\rightarrow \mathsf{Cool}(X, T+1) \\ \mathsf{Cool}(X, T) \wedge \mathsf{Flag}(X, T+1) &\rightarrow \mathsf{Shdn}(X, T+1) \\ \mathsf{Shdn}(X, T) &\rightarrow \mathsf{Malf}(X, T-2) \end{aligned}$$

Consider the query $Q_E = \langle \mathsf{Malf}(X, T), \Pi_E \rangle$. *If the history* D_0 *consists of the single fact* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)$, *then at time instant* 0 *there is no output for* Q_E. *If* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)$ *arrives to* D, *then* $D_1 = D_0 \cup \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)\}$, *and there still is no answer to* Q_E.

Finally, the arrival of $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)$ *to* D *yields* $D_2 = D_1 \cup \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)\}$*, allowing us to infer* $\mathsf{Malf}(\mathsf{wt25}, 0)$*. Then* $\{X := \mathsf{wt25}, T := 0\} \in \mathcal{A}(Q_E, D, 2)$. ◁

Throughout this work, we do not distinguish between the temporal argument in a fact (the timepoint where it is produced) and the instant when it arrives in D. In other words, we assume that at each time point τ, the τ-history D_τ contains all EDB facts about time instants $\tau' < \tau$.

SLD-resolution

A *literal* is an atom or its negation. Atoms are also called *positive* literals, and a negated atom is a *negative* literal. A *definite clause* is a disjunction of literals containing at most one positive literal. In the case where all literals are negative, the clause is a *goal*. We use the standard rule notation for writing definite clauses.

The notions of substitution, unification and most general unifier (mgu) are standard. It is well known that there always exist several mgus of any two unifiable atoms, and that they are unique up to renaming of variables.

Recall that a *goal* is a clause of the form $\neg \wedge_j \beta_j$. If C is a rule $\wedge_i \alpha_i \rightarrow \alpha$, G is a goal $\neg \wedge_j \beta_j$ with $\mathsf{var}(G) \cap \mathsf{var}(C) = \emptyset$, and θ is an mgu of α and β_k, then the *resolvent* of G and C is the goal $\neg \left(\bigwedge_{j<k} \beta_j \wedge \bigwedge_i \alpha_i \wedge \bigwedge_{j>k} \beta_j \right) \theta$.

If P is a program and G is a goal, an *SLD-derivation* of $P \cup \{G\}$ is a (finite or infinite) sequence $G_0, G_1, \ldots$ of goals with $G = G_0$, a sequence $C_1, C_2, \ldots$ of α-renamings of program clauses of P and a sequence $\theta_1, \theta_2, \ldots$ of substitutions such that G_{i+1} is the resolvent of G_i and C_{i+1} using θ_{i+1}. A finite SLD-derivation of $P \cup \{G\}$ where the last goal is a contradiction ($\square$) is called an *SLD-refutation* of $P \cup \{G\}$ of length n, and the substitution obtained by restricting the composition of $\theta_1, \ldots, \theta_n$ to the variables occurring in G is called a *computed answer* of $P \cup \{G\}$.

3 Hypothetical answers

In our running example, $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)$ being produced at time instant 0 yields some evidence that $\mathsf{Malf}(\mathsf{wt25}, 0)$ may turn out to be true. At time instant 1, we may receive further evidence as in the example (the arrival of $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)$), or we might find out that this fact will not be true (if $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)$ does not arrive).

We propose a theory where such *hypothetical answers* to a continuous query are output: if some substitution can become an answer as long as some facts in the future are true, then we output this information. In this way we can lessen the negative effects of unbound wait. Hypothetical answers can also refer to future time points: in our example, $[X := \mathsf{wt25}, T := 2]$ would also be output at time point 0 as a substitution that may prove to be an answer to the query $\langle \mathsf{Shdn}(X, T), \Pi_E \rangle$ when further information arrives.

Our formalism uses ideas from multi-valued logic, where some substitutions correspond to answers (true), others are known not to be answers (false), and others are consistent with the available data, but can not yet be shown to be true or false. In our example, $\mathsf{Malf}(\mathsf{wt25}, 0)$ is consistent with the data at time point 0, and thus "possible"; it is also consistent with the data at time point 1, and thus "more possible"; and it finally becomes (known to be) true at time point 2.

As already motivated, we want answers to give us not only the substitutions that make the query goal true, but also ones that make the query goal possible in the following sense: they depend both on past and future facts, and the past facts are already known.

For the remainder of the article, we assume fixed a query $Q = \langle P, \Pi \rangle$, a data stream D and a time instant τ.

Definition 1 *A* hypothetical answer *to query* Q *over* D_τ *is a pair* $\langle \theta, H \rangle$*, where* θ *is a substitution and* H *is a finite set of ground EDB temporal atoms (the hypotheses) such that:*

- $\mathsf{supp}(\theta) = \mathsf{var}(P)$*;*
- H *only contains atoms with time stamp* $\tau' > \tau$*;*
- $\Pi \cup D_\tau \cup H \models P\theta$*;*
- H *is minimal with respect to set inclusion.*

$\mathcal{H}(Q, D, \tau)$ *is the set of hypothetical answers to* Q *over* D_τ*.*

Intuitively, a hypothetical answer $\langle \theta, H \rangle$ states that $P\theta$ holds if all facts in H are ever produced by the data stream. Thus, $P\theta$ is currently backed up by the information available. In particular, if $H = \emptyset$ then $P\theta$ is an answer in the standard sense (it is a known fact).

Proposition 1 *If* $\langle \theta, \emptyset \rangle \in \mathcal{H}(Q, D, \tau)$*, then* $\theta \in \mathcal{A}(Q, D, \tau)$*.*

We can generalize this proposition, formalizing the intuition we gave for the definition of hypothetical answer.

Proposition 2 *If* $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$*, then there exist a time point* $\tau' \geq \tau$ *and a data stream* D' *such that* $D_\tau = D'_\tau$ *and* $\theta \in \mathcal{A}(Q, D', \tau')$*.*

Example 2 *We illustrate these concepts in the context of Example 1. Consider* $\theta = [X := \mathsf{wt25}, T := 0]$*. Then* $\langle \theta, \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1), \mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)\} \rangle \in \mathcal{H}(Q_E, D, 0)$*. Since* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1) \in D_1$*, we get* $\langle \theta, \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)\} \rangle \in \mathcal{H}(Q_E, D, 1)$*. Finally,* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2) \in D_2$*, so* $\langle \theta, \emptyset \rangle \in \mathcal{H}(Q_E, D, 2)$*. This answer has no hypotheses, and indeed* $\theta \in \mathcal{A}(Q_E, D, 2)$*.*

Take $\theta' = [X := \mathsf{wt42}, T := 1]$ *for another constant* $\mathsf{wt42}$*. Then also e.g.* $\langle \theta', H'_0 \rangle \in \mathcal{H}(Q_E, D, 0)$ *with* $H'_0 = \{\mathsf{Temp}(\mathsf{wt42}, \mathsf{high}, k) \mid 1 \leq k \leq 3\}$*, but since* $\mathsf{Temp}(\mathsf{wt42}, \mathsf{high}, 1) \notin D_1$ *there is no element* $\langle \theta', H' \rangle \in \mathcal{H}(Q_E, D, \tau)$ *for* $\tau \geq 1$. ◁

Hypothetical answers $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$ where $H \neq \emptyset$ can be further split into two kinds: those that are supported by some present or past true fact(s), and those for which there is no evidence whatsover – they only depend on future, unknown facts. For the former, $\Pi \cup H \not\models P\theta$: they rely on some fact from D_τ. This is the class of answers that interests us, as there is non-trivial information in saying that they may become true.

Definition 2 *A non-empty set of facts* $E \subseteq D_\tau$ *is* evidence *supporting* $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$ *if* E *is a minimal set satisfying* $\Pi \cup E \cup H \models P\theta$*. A* supported answer *to* Q *over* D_τ *is a triple* $\langle \theta, H, E \rangle$ *where* E *is evidence supporting* $\langle \theta, H \rangle$*.*

$\mathcal{E}(Q, D, \tau)$ *is the set of supported answers to* Q *over* D_τ*.*

Since set inclusion is well-founded, if $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$ and $\Pi \cup E \cup H \models P\theta$, then there exists a set E' such that $\langle \theta, H, E' \rangle$ is a supported answer to Q over D_τ. However, in general, several such sets E' may exist. As a consequence, Propositions 1 and 2 generalize to supported answers in the obvious way.

Example 3 *In Example 2, the hypothetical answer $\langle \theta, H_0 \rangle$ is supported by $E_0 = \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)\}$, while $\langle \theta, H_1 \rangle$ is supported by $E_1 = \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0), \mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)\}$. Since there is no evidence for $\langle \theta', H'_0 \rangle$, this answer is not supported.* ◁

This example illustrates that unsupported hypothetical answers are not very informative: it is the existence of supporting evidence that distinguishes interesting hypothetical answers from any arbitrary future fact.

However, it is useful to consider even unsupported hypothetical answers in order to develop incremental algorithms to compute supported answers: the sequence of sets $\Theta^E_\tau = \{\theta \mid \langle \theta, H, E \rangle \in \mathcal{E}(Q, D, \tau)$ for some $H, E\}$ is non-monotonic, as at every time point new unsupported hypothetical answers may get evidence and supported hypothetical answers may get rejected. The sequence $\Theta^H_\tau = \{\theta \mid \langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$ for some $H\}$, on the other hand, is anti-monotonic, as the following results state.

Proposition 3 *If $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$, then there exists H^0 such that $\langle \theta, H^0 \rangle \in \mathcal{H}(Q, D, -1)$ and $H = H^0 \setminus D_\tau$. Furthermore, if $H \neq H^0$, then $\langle \theta, H, H^0 \setminus H \rangle \in \mathcal{E}(Q, D, \tau)$.*

Proposition 4 *If $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$ and $\tau' < \tau$, then there exists $\langle \theta, H' \rangle \in \mathcal{H}(Q, D, \tau')$ such that $H = H' \setminus (D_\tau \setminus D_{\tau'})$.*

Examples 2 and 3 also illustrate this property, with hypotheses turning into evidence as time progresses. Since $D_{-1} = \emptyset$, Proposition 3 is a particular case of Proposition 4.

In the next sections we show how to compute hypothetical answers and the corresponding sets of evidence for a given continuous query.

4 Operational semantics via SLD-resolution

The definitions of hypothetical and supported answers are declarative. We now show how SLD-resolution can be adapted to algorithms that compute these answers. We use standard results about SLD-resolution, see for example (Lloyd 1984).

We begin with a simple observation: since the only function symbol in our language is addition of temporal parameters (which is invertible), we can always choose mgus that do not replace variables in the goal with new ones.

Lemma 5 *Let $\neg \wedge_i \alpha_i$ be a goal and $\wedge_j \beta_j \rightarrow \beta$ be a rule such that β is unifiable with α_k for some k. Then there is an mgu $\theta = [X_1 := t_1, \ldots, X_n := t_n]$ of α_k and β such that all variables occurring in $t_1, \ldots, t_n$ also occur in α_k.*

Without loss of generality, we assume all mgus in SLD-derivations to have the property in Lemma 5.

In classical SLD-resolution, derivations must end in the empty clause. We relax this by allowing derivations to end with a goal if: this goal only refers to EDB predicates and all the temporal terms in it refer to future instants (possibly after further instantiation). This makes the notion of derivation also dependent on a time parameter.

Definition 3 *An atom $P(t_1, \ldots, t_n)$ is a* future atom wrt τ *if P is a temporal predicate and the time term t_n either contains a temporal variable or is a time instant $t_n > \tau$.*

Definition 4 *An* SLD-refutation with future premises *of Q over D_τ is a finite SLD-derivation of $\Pi \cup D_\tau \cup \{\neg P\}$ whose last goal only contains future EDB atoms wrt τ.*

If $\mathcal{D}$ is an SLD-refutation with future premises of Q over D_τ with last goal $G = \neg \wedge_i \alpha_i$ and θ is the substitution obtained by restricting the composition of the mgus in $\mathcal{D}$ to $\mathsf{var}(P)$, then $\langle \theta, \wedge_i \alpha_i \rangle$ is a computed answer with premises *to Q over D_τ, denoted $\langle Q, D_\tau \rangle \vdash_{\mathsf{SLD}} \langle \theta, \wedge_i \alpha_i \rangle$.*

Example 4 *Consider the query Q_E from Example 1 and let $\tau = 1$. There is an SLD-derivation of $\Pi \cup D_1 \cup \{\neg \mathsf{Malf}(X, T)\}$ ending with the goal $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)$, which is a future EDB atom with respect to 1. Thus, $\langle Q_E, D_1 \rangle \vdash_{\mathsf{SLD}} \langle \theta, \mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2) \rangle$ with $\theta = [X := \mathsf{wt25}, T := 0]$.* ◁

Computed answers with premises are the operational counterpart to hypothetical answers, with two caveats. First, a computed answer with premises need not be ground: there may be some universally quantified variables in the last goal. Second, $\wedge_i \alpha_i$ may contain redundant conjuncts, in the sense that they might not be needed to establish the goal. We briefly illustrate these two features.

Example 5 *Continuing with our running example, there is also an SLD-derivation of $\Pi \cup D_1 \cup \{\neg \mathsf{Malf}(X, T)\}$ ending with the goal $\neg \bigwedge_{i=0}^{2} \mathsf{Temp}(X, \mathsf{high}, T + i)$, which only contains future EDB atoms wrt 1. Thus also $\langle Q_E, D_1 \rangle \vdash_{\mathsf{SLD}} \langle \emptyset, \bigwedge_{i=0}^{2} \mathsf{Temp}(X, \mathsf{high}, T + i) \rangle$.* ◁

Example 6 *Consider the program Π' containing rules $\mathsf{P}(\mathsf{a}, T) \rightarrow \mathsf{R}(\mathsf{a}, T)$ and $\mathsf{P}(\mathsf{a}, T) \wedge \mathsf{Q}(\mathsf{a}, T) \rightarrow \mathsf{R}(\mathsf{a}, T)$ and the query $Q' = \langle \mathsf{R}(X, T), \Pi' \rangle$.*

Let $D'_0 = \emptyset$. There is an SLD-derivation of $\Pi' \cup D'_0 \cup \{\neg \mathsf{R}(X, T)\}$ ending with the goal $\neg (\mathsf{P}(\mathsf{a}, T) \wedge \mathsf{Q}(\mathsf{a}, T))$, which only contains future EDB atoms wrt τ. Thus $\langle Q', D'_0 \rangle \vdash_{\mathsf{SLD}} \langle [X := \mathsf{a}], \mathsf{P}(\mathsf{a}, T) \wedge \mathsf{Q}(\mathsf{a}, T) \rangle$. However, atom $\mathsf{Q}(\mathsf{a}, T)$ is redundant, since $\mathsf{P}(\mathsf{a}, T)$ alone suffices to make $[X := \mathsf{a}]$ an answer to Q for any T.

(Observe that also $\langle Q', D'_0 \rangle \vdash_{\mathsf{SLD}} \langle [X := \mathsf{a}], \mathsf{P}(\mathsf{a}, T) \rangle$, but produced by a different SLD-derivation.) ◁

We now look at the relationship between the operational definition of computed answer with premises and the notion of hypothetical answer. The examples above show that these notions do not precisely correspond. However, we can show that computed answers with premises approximate hypothetical answers and, conversely, every hypothetical answer is a grounded instance of a computed answer with premises.

Proposition 6 (Soundness) *If $\langle Q, D_\tau \rangle \vdash_{\mathsf{SLD}} \langle \theta, \wedge_i \alpha_i \rangle$ and σ is a ground substitution such that $\mathsf{supp}(\sigma) = \mathsf{var}(\wedge_i \alpha_i) \cup (\mathsf{var}(P) \setminus \mathsf{supp}(\theta))$ and $t\sigma > \tau$ for every temporal term t occurring in $\wedge_i \alpha_i$, then there is a set $H \subseteq \{\alpha_i \sigma\}_i$ such that $\langle (\theta\sigma)|_{\mathsf{var}(P)}, H \rangle \in \mathcal{H}(Q, D, \tau)$.*

Proposition 7 (Completeness) *If $\langle \theta, H \rangle \in \mathcal{H}(Q, D, \tau)$, then there exist substitutions ρ and σ and a finite set of atoms $\{\alpha_i\}_i$ such that $\theta = \rho\sigma$, $H = \{\alpha_i\sigma\}_i$ and $\langle Q, D_\tau \rangle \vdash_{\mathsf{SLD}} \langle \rho, \wedge_i \alpha_i \rangle$.*

All notions introduced in this section depend on the time parameter τ, and in particular on the history dataset D_τ. Next, we explore the idea of "organizing" the SLD-derivation adequately to pre-process Π independently of D_τ, so that the computation of (hypothetical) answers can be split into an offline part and a less expensive online part.

5 Incremental computation of answers

Proposition 4 states that the set of hypothetical answers evolves as time passes, with hypothetical answers either gaining evidence and becoming query answers or being put aside due to depending on facts that turn out not to be true.

In this section, we show how we can use this temporal evolution to compute supported answers incrementally. We start by revisiting SLD-derivations and showing how they can reflect this temporal structure.

Proposition 8 *If $\langle Q, D_\tau \rangle \vdash_{\mathsf{SLD}} \langle \theta, \wedge_i \alpha_i \rangle$, then there exist an SLD-refutation with future premises of Q over D_τ computing $\langle \theta, \wedge_i \alpha_i \rangle$ and a sequence $k_{-1} \leq k_0 \leq \ldots \leq k_\tau$ such that: (i) goals $G_1, \ldots, G_{k_{-1}}$ are obtained by resolving with clauses from Π; (ii) for $0 \leq i \leq \tau$, goals $G_{k_{i-1}+1}, \ldots, G_{k_i}$ are obtained by resolving with clauses from $D_i \setminus D_{i-1}$.*

An SLD-refutation with future premises with the property guaranteed by Proposition 8 is called a *stratified* SLD-refutation with future premises. Since data stream D only contains EDB atoms, it also follows that in a stratified SLD-refutation all goals after $G_{k_{-1}}$ are always resolved with EDB atoms. Furthermore, each G_{k_i} contains only future EDB atoms with respect to i. Let θ_i be the restriction of the composition of all substitutions in the SLD-derivation up to step k_i to $\mathsf{var}(P)$. Then $G_{k_i} = \neg \wedge_j \alpha_j$ represents all hypothetical answers to Q over D_i of the form $\langle (\theta_i\sigma)|_{\mathsf{var}(P)}, \wedge_j \alpha_j \rangle$ for some ground substitution σ (cf. Proposition 6).

This yields an online procedure to compute supported answers. In a pre-processing step, we calculate all computed answers with premises to Q over D_{-1}, and keep the ones with minimal set of formulas. (Note that Proposition 7 guarantees that all minimal sets are generated by this procedure, although some non-minimal sets may also appear as in Example 5.) The online part of the procedure then performs SLD-resolution between each of these sets and the facts produced by the data stream, adding the resulting resolvents to a set of schemata of supported answers (i.e. where variables may still occur). By Proposition 8, if there is at least one resolution step at this stage, then the hypothetical answers represented by these schemata all have evidence, so they are indeed supported.

In general, the pre-processing step of this procedure may not terminate, as the following example illustrates.

Example 7 *Consider the following program Π'', where R is an extensional predicate and S is an intensional predicate.*

$$S(X,T) \to S(X,T+1) \qquad R(X,T) \to S(X,T)$$

If $R(a, t_0)$ is produced by the datastream, then $S(a,t)$ is true for every $t \geq t_0$. Thus, $\langle [X := a], \{R(a, T-k)\} \rangle \in \mathcal{H}(\langle S(X,T), \Pi'' \rangle, D, 0)$ for all k. The pre-processing step needs to output this infinite set, so it cannot terminate. ◁

We establish termination of the pre-processing step for two different classes of queries. A query $Q = \langle P, \Pi \rangle$ is *connected* if each rule in P contains at most one temporal variable, which occurs in the head whenever it occurs in the body; and it is *nonrecursive* if the directed graph induced by its dependencies is acyclic, cf. (Ronca et al. 2018b).

Proposition 9 *Let $Q = \langle P, \Pi \rangle$ be a nonrecursive and connected query. Then the set of all computed answers with premises to Q over D_{-1} can be computed in finite time.*

(Ronca et al. 2018b) also formally define delay and window size as follows. Q is a query with temporal variable T.

Definition 5 *A* delay *for Q is a natural number d such that: for every substitution θ and every $\tau \geq T\theta + d$, $\theta \in \mathcal{A}(Q, D, \tau)$ iff $\theta \in \mathcal{A}(Q, D, T\theta + d)$.*

A natural number w is a window size *for Q if: each θ is an answer to Q over D_τ iff θ is an answer to Q over $D_\tau \setminus D_{\tau - w}$.*

Proposition 10 *If Q has a delay d and a window size w, then the set of all computed answers with premises to Q over D_{-1} can be computed in finite time.*

The (offline) pre-processing step gives us a finite set $\mathcal{P}_Q$ of *preconditions* for Q that represents $\mathcal{H}(Q, D, -1)$: for each computed answer $\langle \theta, \wedge_i \alpha_i \rangle$ with premises to Q over D_{-1} where $\{\alpha_i\}_i$ is minimal, $\mathcal{P}_Q$ contains an entry $\langle \theta, M, \{\alpha_i\}_i \setminus M \rangle$ where M is the subset of the α_i with minimal timestamp (those whose temporal variable is $T + k$ with minimal k).

Each tuple $\langle \theta, M, F \rangle \in \mathcal{P}_Q$ represents the set of all hypothetical answers $\langle \theta\sigma, (M \cup F)\sigma \rangle$ as in Proposition 6.

We now show that computing and updating the set $\mathcal{E}(Q, D, \tau)$ can be done efficiently. This set is maintained again as a set $\mathcal{S}_\tau$ of schematic supported answers. We assume that Q is a connected query; we discuss how to remove this restriction later.

Proposition 11 *The following algorithm computes $\mathcal{S}_{\tau+1}$ from $\mathcal{P}_Q$ and $\mathcal{S}_\tau$ in time polynomial in the size of $\mathcal{P}_Q$, $\mathcal{S}_\tau$ and $D_{\tau+1} \setminus D_\tau$.*

1. *For each $\langle \theta, M, F \rangle \in \mathcal{P}_Q$ and each computed answer σ to $(D_{\tau+1} \setminus D_\tau) \cup \{\neg \bigwedge M\}$, add $\langle \theta\sigma, M\sigma, F\sigma \rangle$ to $\mathcal{S}_{\tau+1}$. (By connectedness, all time variables in $M\sigma \cup F\sigma$ are instantiated in $\theta\sigma$.)*
2. *For each $\langle \theta, E, H \rangle \in \mathcal{S}_\tau$, compute the set $M \subseteq H$ of atoms with timestamp $\tau + 1$. For each computed answer σ to $(D_{\tau+1} \setminus D_\tau) \cup \{\neg \bigwedge M\}$, add $\langle \theta\sigma, (E \cup M)\sigma, (H \setminus M)\sigma \rangle$ to $\mathcal{S}_{\tau+1}$.*

The following example also illustrates that, by outputting hypothetical answers, we can answer queries earlier than in other formalisms.

Example 8 *Suppose that we extend the program Π_E in our running example as in Example 2 from (Ronca et al. 2018b), i.e. with the rule* $\mathsf{Temp}(X, \mathsf{n/a}, T) \to \mathsf{Malf}(X, T)$.

If D_1 *contains* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)$, $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 1)$ *and* $\mathsf{Temp}(\mathsf{wt42}, \mathsf{n/a}, 1)$, *then*

$$\begin{aligned}\mathcal{S}_1 = \{&\langle [T := 0, X := \mathsf{wt25}],\\ &\{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, i) \mid i = 0, 1\},\\ &\{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 2)\}\rangle,\\ &\langle [T := 1, X := \mathsf{wt42}], \{\mathsf{Temp}(\mathsf{wt42}, \mathsf{n/a}, 1)\}, \emptyset\rangle\}.\end{aligned}$$

Thus, the answer $[T := 1, X := \mathsf{wt42}]$ *is produced at time-point 1, rather than being delayed until it is known whether* $[T := 0, X := \mathsf{wt25}]$ *is really an answer.*

Proposition 12 (Soundness) *If* $\langle \theta, E, H\rangle \in \mathcal{S}_\tau$ *and* σ *instantiates all free variables in* $E \cup H$, *then* $\langle \theta\sigma, H\sigma, E\sigma\rangle \in \mathcal{E}(Q, D, \tau)$.

Proposition 13 (Completeness) *If* $\langle \sigma, H, E\rangle \in \mathcal{E}(Q, D, \tau)$, *then there exist a substitution* ρ *and a triple* $\langle \theta, E', H'\rangle \in \mathcal{S}_\tau$ *such that* $\sigma = \theta\rho$, $H = H'\rho$ *and* $E = E'\rho$.

It also follows from our construction that, if S_τ contains a triple $\langle \theta, E, H\rangle$ with $\theta(T) \leq \tau$ and $H \neq \emptyset$ and d is a delay for Q, then the time stamp of each element of H is at most $\tau + d$. Likewise, if w is a window size for Q, then all elements in E must have time stamp at least $\tau - w$.

Generalization. The hypotheses in Propositions 9 and 10 are not necessary to guarantee termination of the algorithm presented for the pre-processing step. Indeed, consider the following example.

Example 9 *In the context of our running example, we say that a turbine has a manufacturing defect if it exhibits two specific failures during its lifetime: at some time it overheats, and at some (different) time it does not send a temperature reading. Since this is a manufacturing defect, it holds at timepoint* 0, *regardless of when the failures actually occur. We can model this property by the rule*

$$\mathsf{Temp}(X, \mathsf{high}, T_1), \mathsf{Temp}(X, \mathsf{n/a}, T_2) \rightarrow \mathsf{Defective}(X, 0).$$

Let Π'_E *be the program obtained from* Π_E *by adding this rule and consider the query* $Q' = \langle \mathsf{Defective}(X, T), \Pi'_E\rangle$.

Performing SLD-resolution with Π'_E *and* $\mathsf{Defective}(X, 0)$ *yields (in one step) the goal* $\neg(\mathsf{Temp}(X, \mathsf{high}, T_1) \wedge \mathsf{Temp}(X, \mathsf{n/a}, T_2))$, *which only contains future atoms with respect to* -1. *The set of computed answers with premises to* Q' *over* D_{-1} *is indeed*

$$\{\langle [T := 0], \mathsf{Temp}(X, \mathsf{high}, T_1) \wedge \mathsf{Temp}(X, \mathsf{n/a}, T_2)\rangle\}. \quad \triangleleft$$

As this example shows, if a rule in the program includes different time variables, the query cannot have a delay or window size (since no predicate can use both T_1 and T_2).

We can also adapt our algorithm to work in this situation, removing the hypothesis of connectedness in Proposition 11 but sacrificing polynomial complexity. This allows us to deal with some situations of unbound wait, as we illustrate.

Example 10 *Continuing with Example 9, since* D_0 *contains* $\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)$, *the set* $\mathcal{S}_0$ *includes*

$$\begin{aligned}\langle \theta' = [&X := \mathsf{wt25}, T := 0],\\ &\{\mathsf{Temp}(\mathsf{wt25}, \mathsf{high}, 0)\}, \{\mathsf{Temp}(\mathsf{wt25}, \mathsf{n/a}, T_2)\}\rangle.\end{aligned}$$

Note that we do not know when (if ever) θ' *will become an answer to the original query, but there is relevant information output to the user.* $\triangleleft$

6 Adding negation

We now show how our framework extends naturally to programs including negated atoms in bodies of rules. We make the usual assumptions that negation is safe (each variable in a negated atom in the body of a rule occurs non-negated elsewhere in the rule).

In the pre-processing step, we compute $\mathcal{P}_Q$ as before. However, the leaves in the SLD-derivations constructed may now contain negated (intensional) atoms as well as extensional atoms. For each such negated atom we generate a fresh query Q', replacing the time parameter with a variable, and repeat the pre-processing step to compute $\mathcal{P}_{Q'}$. We iterate this construction until no fresh queries are generated. Since the number of queries that can be generated is finite, Propositions 9 and 10 still hold.

The online step of the algorithm is now more complicated, as it needs to keep track of all answers to the auxiliary queries generated by negated atoms. The following algorithm computes $\mathcal{S}_{\tau+1}$ from $\mathcal{P}$ and $\mathcal{S}_\tau$.

1. For each Q and each $\langle \theta, M, F\rangle \in \mathcal{P}_Q$: if there is a negated atom in M with time parameter t, let σ be the substitution such that $t\sigma = \tau + 1$; otherwise let $\sigma = \emptyset$. Let M' be the set of positive literals in M.

   For each computed answer σ' to $(D_{\tau+1} \setminus D_\tau) \cup \{\neg \bigwedge M'\sigma\}$ add $\langle \theta\sigma\sigma', M'\sigma\sigma', ((M \setminus M') \cup F)\sigma\sigma'\rangle$ to $\mathcal{S}_{\tau+1}(Q)$.

   (Observe that all time variables in $M\sigma\sigma' \cup F\sigma\sigma'$ are instantiated in $\theta\sigma\sigma'$.)

2. For each query Q and each $\langle \theta, E, H\rangle \in \mathcal{S}_\tau(Q)$, compute the set $M \subseteq H$ of positive literals with timestamp $\tau + 1$. For each computed answer σ to $(D_{\tau+1} \setminus D_\tau) \cup \{\neg \bigwedge M\}$, add $\langle \theta\sigma, (E \cup M)\sigma, (H \setminus M)\sigma\rangle$ to $\mathcal{S}_{\tau+1}(Q)$.

3. For each query Q and each $\langle \theta, E, H\rangle \in \mathcal{S}_{\tau+1}(Q)$, compute the set $M \subseteq H$ of negative literals. For each literal $\mathsf{not}\ \ell \in M$ with timestamp $t \leq \tau + 1$, let ℓ' be the query on the same predicate symbol as ℓ.

   If there is no tuple $\langle \theta', E', H'\rangle \in \mathcal{S}_{\tau+1}(\ell')$ where ℓ and $\ell'\theta'$ are unifiable, remove $\mathsf{not}\ \ell$ from H and add it to E.

   If there is a tuple $\langle \theta', E', \emptyset\rangle \in \mathcal{S}_{\tau+1}(\ell')$ such that ℓ and $\ell'\theta'$ are unifiable, then: (i) remove $\langle \theta, E, H\rangle$ from $S_{\tau+1}(Q)$, (ii) for each substitution σ ranging over the free variables in ℓ, if $\ell\sigma$ does not unify with $\ell'\theta'$, then add $\langle \theta\sigma, E\sigma, H\sigma\rangle$ to $S_{\tau+1}(Q)$.

Step 3 makes the running time of this algorithm exponential, since it requires iterating over a set of substitutions. However, if negation is T-stratified (which we define below), we can still establish a polynomial bound.

Definition 6 *The* temporal closure *of a program* Π *is the program* $\Pi^\downarrow$ *defined as follows. For each* $(n+1)$*-ary predicate symbol* p *with a temporal argument in the signature underlying* Π, *the signature for* $\Pi^\downarrow$ *contains a family of* n*-ary predicate symbols* $\{p_t\}_{t\in\mathbb{N}}$. *For each rule in* Π, $\Pi^\downarrow$ *con-*

tains all rules obtained by instantiating its temporal parameter in all possible ways and replacing $p(x_1, \ldots, x_n, t)$ by $p_t(x_1, \ldots, x_n)$. A program Π is T-stratified if $\Pi^{\downarrow}$ is stratified in the usual sense.

Since the number of predicate symbols in $\Pi^{\downarrow}$ is infinite, the usual procedure for deciding whether a program is T-stratified does not necessarily terminate. However, this procedure can be adapted to our framework.

Proposition 14 *There is an algorithm that decides whether a program Π is T-stratified, and in the affirmative case returns a finite representation of the strata.*

Furthermore, for T-stratified programs we can also give a complexity bound for the online step.

Proposition 15 *Let k be the highest arity of any predicate that occurs negated in Π. If Π is T-stratified, then the online algorithm runs in time polynomial in k, the size of $\mathcal{P}_Q$, $\mathcal{S}_\tau$, $D_{\tau+1} \setminus D_\tau$ and the total number of queries.*

7 Related work and discussion

Incremental evaluation. Computing answers to a query over a data source that is continuously producing information, be it at slow or very fast rates, asks for techniques that allow for some kind of *incremental evaluation*, in order to avoid reevaluating the query from scratch each time a new tuple of information arrives. Several efforts have been made in that direction, capitalising on incremental algorithms based on seminaive evaluation (Gupta, Mumick, and Subrahmanian 1993; Abiteboul, Hull, and Vianu 1995; Barbieri et al. 2010; Motik et al. 2015; Hu, Motik, and Horrocks 2018), on truth maintenance systems (Beck, Dao-Tran, and Eiter 2015), window oriented (Ghanem et al. 2007) among others. Our algorithm fits naturally in the first class, as it is an incremental variant of SLD-resolution.

Unbound wait and blocking queries. The problems of unbound wait and blocking queries have deserved much attention in the area of query answering over data streams. There have been efforts to identify the problematic issues (Law, Wang, and Zaniolo 2011) and varied proposals to cope with their negative effects, as in (Zaniolo 2012; Özçep, Möller, and Neuenstadt 2014; Beck et al. 2015; Ronca et al. 2018a) among others. Our framework deals with unbound wait by outputting at each time point all supported answers (including some that later may prove to be false), as illustrated in Examples 8 and 10. Furthermore, if we receive a supported answer whose time parameters are all instantiated, then we immediately have a bound on how long we have to wait until the answer is definite (or rejected).

Blocking queries may still be a problem in our framework, though: as seen in Example 7, blocking operations (in the form of infinitely recursive predicates) may lead to infinite SLD-derivations, which cause the pre-processing step of our algorithm to diverge. We showed that syntactic restrictions of the kind already considered by (Ronca et al. 2018b) are guaranteed to avoid blocking queries.

Our algorithm also implicitly embodies a forgetting algorithm, as the only elements of D_τ kept in the E sets are those that are still relevant to compute future answers to Q.

(Ronca et al. 2018a) proposes the *language of forward-propagating queries*, a variant of Temporal Datalog that allows queries to be answered in polynomial time in the size of the input data. This is achieved at the cost of disallowing propagation of derived facts towards past time points – precluding, e.g., rules like $\mathsf{Shdn}(X, T) \rightarrow \mathsf{Malf}(X, T-2)$ in Example 1. The authors present a generic algorithm to compute answers to a query, given a window size, and investigate methods for calculating a minimal window size.

(Zaniolo 2012), working in another variant of Datalog called Streamlog, characterise *sequential rules and programs* with the purpose of avoiding blocking behaviour. Again, rule $\mathsf{Shdn}(X, T) \rightarrow \mathsf{Malf}(X, T-2)$ from Example 1 is disallowed in this framework, since the timestamp in the head of a rule may never be smaller than the timestamps of the atoms in the body.

Incomplete information. When reasoning over sources with incomplete information, the concepts of certain and possible answers, and granularities thereof, inevitably arise (Gray, Nutt, and Williams 2007; Lang et al. 2014; Razniewski et al. 2015) as a way to assign confidence levels to the information output to the user. These approaches, like ours, also compute answers with incomplete information.

However, our proposal is substantially different from those works, since they focus on answers over past incomplete information. First, as mentioned in Section 2, we assume that our time stream is complete, in the sense that whenever it produces a fact about a time instant τ, all EDB facts about time instants $\tau' < \tau$ are already there (in line with the progressive closing world assumption of (Zaniolo 2012)). Secondly, our hypothetical and supported answers are built over present facts and future, still unknown, hypotheses, and eventually either become effective answers or are discarded; in the scenario of past incomplete information, the confidence level of answers may never change.

Negation. Other authors have considered languages using negation. Our definition of stratification is similar to the concept of temporal stratification from (Zaniolo 2015). However, temporal stratification requires the strata to be also ordered according to time (i.e., if the definition of p_t is in Π_i and the definition of p_{t+1} is in Π_j, then $j \geq i$). We make no such assumption in this work.

(Beck, Dao-Tran, and Eiter 2015) also consider a notion of temporal stratification for stream reasoning. However, their framework also includes explicit temporal operators, making the whole formalization more complex.

Acknowledgements. This work was partially supported by the Independent Research Fund Denmark, grant DFF-7014-00041, and by UID/MULTI/04046/2019 Research Unit grant from FCT, Portugal (to BioISI).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

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

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In Popa, L.; Abiteboul, S.; and Kolaitis, P. G., eds., *Proc. PODS*, 1–16. ACM.

Barbieri, D. F.; Braga, D.; Ceri, S.; Valle, E. D.; and Grossniklaus, M. 2009. C-SPARQL: SPARQL for continuous querying. In Quemada, J.; León, G.; Maarek, Y. S.; and Nejdl, W., eds., *Proc. WWW*, 1061–1062. ACM.

Barbieri, D. F.; Braga, D.; Ceri, S.; Valle, E. D.; and Grossniklaus, M. 2010. Incremental reasoning on streams and rich background knowledge. In *Proc. ESWC*, volume 6088 of *LNCS*, 1–15. Springer.

Beck, H.; Dao-Tran, M.; Eiter, T.; and Fink, M. 2015. LARS: A logic-based framework for analyzing reasoning over streams. In Bonet and Koenig (2015), 1431–1438.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2015. Answer update for rule-based stream reasoning. In Yang, Q., and Wooldridge, M. J., eds., *Proc. IJCAI 2015*, 2741–2747. AAAI Press.

Bonet, B., and Koenig, S., eds. 2015. *Proc. 29th AAAI Conference on Artificial Intelligence*. AAAI Press.

Chomicki, J., and Imielinski, T. 1988. Temporal deductive databases and infinite objects. In Edmondson-Yurkanan, C., and Yannakakis, M., eds., *Proc. PODS*, 61–73. ACM.

Cruz-Filipe, L.; Gaspar, G.; and Nunes, I. 2019. Hypothetical answers to continuous queries over data streams. *CoRR* abs/1905.09610.

Dao-Tran, M., and Eiter, T. 2017. Streaming multi-context systems. In Sierra, C., ed., *Proc. IJCAI*, 1000–1007. ijcai.org.

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

Ghanem, T. M.; Hammad, M. A.; Mokbel, M. F.; Aref, W. G.; and Elmagarmid, A. K. 2007. Incremental evaluation of sliding-window queries over data streams. *IEEE Trans. Knowl. Data Eng.* 19(1):57–72.

Gray, A. J.; Nutt, W.; and Williams, M. H. 2007. Answering queries over incomplete data stream histories. *IJWIS* 3(1/2):41–60.

Gupta, A.; Mumick, I. S.; and Subrahmanian, V. 1993. Maintaining views incrementally. In Buneman, P., and Jajodia, S., eds., *Proc. SIGMOD Conference*, 157–166. ACM Press.

Hu, P.; Motik, B.; and Horrocks, I. 2018. Optimised maintenance of datalog materialisations. In McIlraith and Weinberger (2018), 1871–1879.

Lang, W.; Nehme, R. V.; Robinson, E.; and Naughton, J. F. 2014. Partial results in database systems. In Dyreson, C. E.; Li, F.; and Özsu, M. T., eds., *Proc. SIGMOD Conference*, 1275–1286. ACM.

Law, Y.; Wang, H.; and Zaniolo, C. 2011. Relational languages and data models for continuous queries on sequences and data streams. *ACM Trans. Database Syst.* 36(2):8:1–8:32.

Lloyd, J. W. 1984. *Foundations of Logic Programming*. Springer.

McIlraith, S. A., and Weinberger, K. Q., eds. 2018. *Proc. 32nd AAAI Conference on Artificial Intelligence*. AAAI Press.

Motik, B.; Nenov, Y.; Piro, R. E. F.; and Horrocks, I. 2015. Incremental update of datalog materialisation: the backward/forward algorithm. In Bonet and Koenig (2015), 1560–1568.

Özçep, Ö. L.; Möller, R.; and Neuenstadt, C. 2014. A stream-temporal query language for ontology based data access. In Lutz, C., and Thielscher, M., eds., *Proc. KI*, volume 8736 of *LNCS*, 183–194. Springer.

Razniewski, S.; Korn, F.; Nutt, W.; and Srivastava, D. 2015. Identifying the extent of completeness of query answers over partially complete databases. In Sellis, T. K.; Davidson, S. B.; and Ives, Z. G., eds., *Proc. SIGMOD Conference*, 561–576. ACM.

Ronca, A.; Kaminski, M.; Grau, B. C.; and Horrocks, I. 2018a. The window validity problem in rule-based stream reasoning. In Thielscher, M.; Toni, F.; and Wolter, F., eds., *Proc. KR*, 571–581. AAAI Press.

Ronca, A.; Kaminski, M.; Grau, B. C.; Motik, B.; and Horrocks, I. 2018b. Stream reasoning in temporal datalog. In McIlraith and Weinberger (2018), 1941–1948.

Stonebraker, M.; Çetintemel, U.; and Zdonik, S. B. 2005. The 8 requirements of real-time stream processing. *SIGMOD Record* 34(4):42–47.

Zaniolo, C. 2012. Logical foundations of continuous query languages for data streams. In Barceló, P., and Pichler, R., eds., *Proc. Datalog*, volume 7494 of *LNCS*, 177–189. Springer.

Zaniolo, C. 2015. Expressing and supporting efficiently greedy algorithms as locally stratified logic programs. In Vos, M. D.; Eiter, T.; Lierler, Y.; and Toni, F., eds., *Proc. ICLP 2015 (Technical Communications)*, volume 1433 of *CEUR Workshop Proceedings*. CEUR-WS.org.