On Construction of Holistic Synopses under the Duplicate Semantics of Streaming Queries*

David Toman

D. R. Cheriton School of Computer Science
University of Waterloo, Canada
E-mail: david@cs.uwaterloo.ca

Abstract

Transaction-time temporal databases and query languages provide a solid framework for analyzing properties of queries over data streams. In this paper we focus on issues connected with the construction of space-bounded synopses that enable answering of continuous queries over unbounded data streams while requiring only limited space. We link the problem to the problem of query-driven data expiration in append-only temporal databases and study space bounds on synopses that are sufficient and necessary for query answering under duplicate semantics.

1 Introduction

A considerable effort to understand many aspects of query processing over streaming data—data that is arriving in fragments over time—has been the topic of research in the last several years, e.g., [4, 5, 6, 10, 11, 12, 14, 15] and many others. These efforts have mainly focused on efficient processing of *continuous queries*—queries evaluated continuously as more data is arriving on a stream—over unbounded data streams. A *key* component of such solutions is the construction of *synopses*—data summaries that allow execution of continuous queries without the need to buffer or otherwise store the entire history of the data stream.

The main goal of this paper is to show that certain requirements, often rather desirable in such systems—such as the use of SQL-style duplicate semantics while maintaining reasonable bounds on the size of the summary data—are not possible to achieve. The paper later shows how acceptable results can be achieved by carefully limiting the expressive power of streaming query languages in which continuous queries are formulated.

*Preliminary version of this paper has appeared in [19].

As we desire to derive bounds as strong as possible, we adopt a *holistic* approach to the construction of synopses for continuous queries. Unlike other approaches that often construct synopses on a per-physical-operator basis (e.g., for the so called symmetric joins, etc., [4]), we develop techniques that tailor the synopsis to a complete continuous query—hence the use of the term *holistic synopsis*.

Many of the techniques presented in this paper can be traced to approaches designed to allow efficient data expiration in transaction-time temporal databases [17]. The novel contributions of this paper are mainly concerned with the use of *duplicate semantics* for queries and can be summarized as follows:

1. We show that adopting an SQL-style duplicate semantics for SQL-like streaming queries makes construction of bounded synopses impossible for many natural streaming query languages as, in general, the synopses may have to grow at least linearly with respect to the stream length. We contrast this with known upper bounds for the same languages when *set semantics* is used.

2. We show that for certain fragments of these languages, this growth can be tamed to a logarithmic factor; that factor, however, cannot be avoided while retaining any meaningful duplicate semantics.

Note that the negative results presented in the paper are based on information-theoretic properties of queries and thus cannot be improved upon by more sophisticated algorithms without resorting to approximations. The results are contrasted to similar results obtained for the set semantics of the same languages where constant bounds in the length of the stream can be obtained. In addition to the technical results, the paper establishes a strong parallel between techniques developed for transaction-time temporal databases

[9], data expiration [17], and approaches to space efficient evaluation of streaming queries. The results presented here also refine those presented in [2, 3], mainly by distinguishing between the space dependencies on the active data domain and on the current length of the data stream, respectively, and by improving on the lower bounds by establishing logarithmic and linear lower bounds as described above.

The rest of the paper is organized as follows: Section 2 provides the basic definitions and shows the links between streaming queries and temporal databases. Section 3 shows that in general, duplicate semantics leads at least to a logarithmic lower bound on the size of the synopses, measured in the length of the data stream; it also identifies cases in which allowing duplicates in the data model leads to linear lower bounds on the size of the synopses needed to answer a continuous query. Section 4 shows fragments of query languages for which a logarithmic bound can be achieved and discusses variants to the standard SQL-style duplicate semantics that could be more amenable in the streaming setting. We conclude by identifying a number of open issues topics for further research in Section 5.

2 Background

We begin with a review of the relevant definitions in the area of transaction-time temporal databases, and relate these definitions to querying data streams. The presentation in this section is based on a chapter on data expiration [17] with terminology suitably modified to data streams.

2.1 Streams and Temporal Databases

We first formalize the notion of a data stream as follows.

Definition 2.1 (Data Stream/History) A *data stream of k-tuples* is a sequence

$$S = \langle S_0, S_1, \ldots, S_t, \ldots \rangle$$

where each S_t is a multiset of k-tuples that have arrived in S at time t. We call the multisets S_t *states of S at t*. We assume discrete integer-like time with time instants drawn from a linearly ordered set and we allow multiple tuples to arrive at the same time. The data values forming the tuples belong to the domain of *uninterpreted constants* (the data domain) and are equipped with equality only. At any particular finite time t, we only have access to a *finite prefix* of S of the form

$$S(t) = \langle S_0, S_1, \ldots, S_t \rangle.$$

We use $\mathbf{T}(t)$ and $\mathbf{D}(t)$ to denote the *active temporal* and *data* domains of a stream prefix $S(t)$, respectively. Note that the active domains change with time as new data arrives on the stream. The active temporal domain $\mathbf{T}(t)$ is, in our setting, simply the set $\{0, \ldots, t\}$; the definition, however, allows to use timestamps from an arbitrary linearly ordered set. The active data domain consists of all data values that appear in $S(t)$.

For a multiset S, we denote by $\mathsf{dupl}(S)$ the maximal number of times any tuple is duplicated in S.

Without loss of generality, we present our results for a single data stream. However, the results immediately extend to multiple streams, e.g., by coding multiple streams by values of a distinguished attribute. This formalization shows that data stream is simply a variant name for an *append-only* temporal database (often called a *transaction-time temporal database*). This observation allows us eventually to use off-the shelf temporal query languages for querying data streams.

Definition 2.2 (Continuous Queries) Let Q be a computable function that maps a stream prefix $S(t)$ to a multiset S_t^Q. An *answer to a continuous query Q* is then defined as the stream

$$S^Q = \langle S_0^Q, S_1^Q, \ldots, S_i^Q, \ldots \rangle$$

where S_t^Q is the answer to Q over the stream prefix $S(t)$.

While the above definition does not fix the language for the queries Q, the link between data streams and temporal databases suggests that we can use off-the-shelf *temporal query languages* to query data streams. Indeed, it turns out that many of the languages proposed for querying data streams are variants of temporal query languages. Hence, in the rest of this paper we concentrate on temporal languages based on *two-sorted first-order logic* (2-FOL) and on a *fixpoint variant of past first-order temporal logic* (μTL), both assuming SQL-style duplicate semantics.

We also assume that duplicates are represented in *binary* (i.e., using counts to represent the numbers of duplicates). This representation is more compact than the explicit duplication of tuples (hence the synopses are more compact than if the explicit representation was used).

2.2 Holistic Synopses

For continuous queries, it is not desirable and often not practical or even feasible to store an entire data stream in computer storage. Therefore, streaming systems use summaries called *synopses* to remember parts of the data stream that are necessary to generate subsequent answers to continuous queries.

Definition 2.3 (Holistic Synopsis) Let Q be a query over S. A *holistic synopsis* $\mathcal{E}$ for Q is a triple $(\emptyset, \Delta, \Gamma)$ that satisfies the property

$$Q\langle S_0, \ldots, S_t\rangle = \Gamma(S_t, \mathcal{E}(S(t-1)))$$

for any prefix $S(t) = \langle S_0, \ldots, S_t\rangle$ of S; Moreover, the actual synopsis we retain in the streaming system is defined by:

$$\mathcal{E}(S(t)) = \Delta(S_t, \Delta(S_{t-1}, \Delta(\ldots \Delta(S_0, \emptyset)\ldots)))$$

where $\emptyset$ is a constant *initial* synopsis, and Δ is a map from synopses and states to synopses. In addition, we require that the triple $(\emptyset, \Delta, \Gamma)$ can be effectively constructed from Q.

The first two components define the actual *holistic synopsis* for Q as a self-maintainable materialized view of S: the $\emptyset$ component tells us what the contents of this view is in the beginning and the Δ component tells us how to update the view when more data arrives in S. The last component, Γ, reproduces the answers to Q while only accessing the information in the view. Note that the definition does not specify what data model the view uses nor what query languages are used for the three components of the synopsis.

This definition is essentially the same as the definition of a *data expiration operator* for transaction-time temporal databases [17].

How do we compare Holistic Synopses?

Intuitively, we have replaced the complete prefix of S with a materialized view $\mathcal{E}$ defined by the $\emptyset$ and Δ queries. Thus our aim is to minimize the size of the materialized view in terms of:

1. the length of the data stream S, $|\mathbf{T}(t)|$,
2. the number of distinct values in S, $|\mathbf{D}(t)|$,
3. the maximal number of duplicates of a single tuple in S, and
4. the size of Q.

The dependency on the length of the data stream is the most critical factor. Thus we call a synopsis *bounded* if it is bounded by a constant function in the length of the stream (it may, however, depend on the size of the active domain for the data elements, $|\mathbf{D}(t)|$). Similarly, we call a synopsis $\log$-*bounded* if it is bounded by a function logarithmic in $|\mathbf{T}(t)|$.

Results For set Semantics

For streaming query languages that use set semantics, results obtained for temporal databases can be applied:

Proposition 2.4 *A bounded holistic synopsis exists for any streaming query expressed as a*

- *two sorted first order (2-FOL) query [16], or*
- *past μ-calculus (μTL) query [18]*

under set semantics.

The above theorem shows that for queries under set semantics, bounded synopses exist for rather powerful query languages, in particular for the languages introduced in Definitions 3.1 and 3.3 below. The rest of the paper argues that bounded synopses cannot exist when duplicate semantics is used *for the same languages* and that $\log$-bounded synopses are the best we can hope for various fragments under *any reasonable duplicate semantics*.

3 Lower Bounds

In this section we establish the main results of the paper: we show that allowing SQL-style duplicate semantics for streaming queries *effectively prevents* the construction of bounded synopses. The lower bounds obtained here show that in these cases the best synopses are asymptotically comparable in size to the (length of the) complete prefix of the data stream.

3.1 Two-sorted First-order Logic

We start with two-sorted first-order logic: the natural extension of relational query languages to streams and the basis of many temporal and streaming query languages, such as TSQL2 [13] and CQL [4].

Definition 3.1 (First-order Queries: 2-FOL) Let S be a data stream of k-tuples. The syntax of *first-order streaming queries* over S is given by the following grammar.

$$\begin{array}{rcl} Q & ::= & S(t, x_1, \ldots, x_k) \\ & | & x_i = x_j \mid t_i < t_j \mid t_i = t_j \mid t_i > t_j \\ & | & Q \wedge Q \mid \exists x_i.Q \mid \exists t_i.Q \mid \varepsilon\, Q \\ & | & Q \vee Q \mid Q \wedge \neg Q \end{array}$$

The $\varepsilon\, Q$ subformula stands for *duplicate elimination* and can be considered an additional modality in the logic that governs the behavior of duplicates.

$$
\begin{aligned}
S(s), \theta, n &\models S(t, x_1, \ldots, x_k) && \text{if } \langle \theta(x_1), \ldots, \theta(x_k) \rangle \in S_{\theta(t)} \text{ duplicated } n \text{ times} \\
S(s), \theta, 1 &\models x_i = x_j && \text{if } \theta(x_i) = \theta(x_j) \\
S(s), \theta, 1 &\models x_i \neq x_j && \text{if } \theta(x_i) \neq \theta(x_j) \\
S(s), \theta, 1 &\models t_i < t_j && \text{if } \theta(t_i) < \theta(t_j) \\
S(s), \theta, m \cdot n &\models Q_1 \wedge Q_2 && \text{if } S(s), \theta, m \models Q_1 \text{ and } S(s), \theta, n \models Q_2 \\
S(s), \theta, \textstyle\sum_{v \in \mathbf{D}(t)} n_v &\models \exists x.Q && \text{if } S(s), \theta[v/x], n_v \models Q \\
S(s), \theta, \textstyle\sum_{u \in \mathbf{T}(t)} n_s &\models \exists t.Q && \text{if } S(s), \theta[u/t], n_u \models Q \\
S(s), \theta, 1 &\models \varepsilon\, Q && \text{if } S(s), \theta, n \models Q \\
S(s), \theta, n + m &\models Q_1 \vee Q_2 && \text{if } S(s), \theta, m \models Q_1 \text{ and } S(s), \theta, n \models Q_2 \\
S(s), \theta, \max(0, m - n) &\models Q_1 \wedge \neg Q_2 && \text{if } S(s), \theta, m \models Q_1 \text{ and } S(s), \theta, n \models Q_2
\end{aligned}
$$

Figure 1. SQL-style Duplicate Semantics for Streaming Queries.

We assume that the queries are *range-restricted* [1]; this is enforced by requiring variable-compatibility conditions for disjunctions ($\vee$) and negations ($\wedge\neg$) and by the usual restrictions on the occurrences of equalities and inequalities in queries.

The semantics of 2-FOL queries is defined using the usual Tarskian-style satisfaction relation extended to account for duplication; we write

$$S(t), \theta, n \models Q$$

to stand for "*the substitution (tuple) θ is an answer to Q duplicated n times, when evaluated over $S(t)$, a prefix of S*". The full semantics of the above language is given in Figure 1.

An *answer* to a query at time t is the multiset

$$S_t^Q := \{\underbrace{\theta, \ldots, \theta}_{n} \mid S(t), \theta, n \models Q\}.$$

Note that the value n must be unique for a given tuple θ, i.e., θ functionally determines n. To simplify the definition, we assume that substitutions not present in an answer have 0 duplicates.

The main difference from the standard definition is that it specifies how duplicates are handled in queries. We utilize an SQL-style definition of duplicates, i.e., we use a product for conjunctions, a sum for disjunction (union) and existential quantification (projection), and a difference for range-restricted negation (multiset difference).

Note, that the query language in Definition 3.1 is simply a temporal query language when the stream is regarded as finite prefix of a database history: the semantics of the language is defined with respect to the finite portion of the data stream; answering queries over *all potential extensions* of a stream has been shown undecidable for any reasonably powerful query language [8], for detailed discussion of this phenomenon see [9].

Lower Bound for 2-FOL

Now we are ready to provide lower bounds that show why the use of unrestricted duplicate semantics for streaming queries may be expensive and, in certain cases, not feasible at all. An $\Omega(\log |\mathbf{T}(t)|)$ lower bound has been observed for queries with the counting aggregate [16]. A similar query in 2-FOL, e.g.,

$$(\exists t.S(t, a)) \wedge \neg(\exists t.S(t, b)),$$

expressing that there were more a's than b's in the stream S, for a and b distinct constants, yields such a bound for queries with duplicates. It is easy to see using the pigeon-hole principle that any synopsis for this query needs $\Omega(\log |\mathbf{T}(t)|)$ bits.

However, it turns out that $\log$-bounded synopses are not sufficient in general for 2-FOL queries with duplicate semantics.

Theorem 3.2 *There is a first-order query for which the space required by any synopsis must be bounded from below by $\Omega(|\mathbf{T}(t)|)$.*

Proof (sketch): Consider the query

$$\begin{aligned}\varepsilon \exists t_1, t_2.t_1 < t_2 \wedge \neg((\exists x.S(t_1, x)) \wedge \neg(\exists x.S(t_2, x))) \\ \wedge \neg((\exists x.S(t_2, x)) \wedge \neg(\exists x.S(t_1, x)))\end{aligned}$$

The query expresses the condition that *two states of S, t_1 and t_2, contain the same number of tuples*. Now consider a

$$
\begin{aligned}
S(s), \theta, t, n &\models S(x_1, \ldots, x_k) && \text{if } \langle \theta(x_1), \ldots, \theta(x_k) \rangle \in S_t \text{ duplicated } n \text{ times} \\
S(s), \theta, t, 1 &\models x_i = x_j && \text{if } \theta(x_i) = \theta(x_j) \\
S(s), \theta, t, 1 &\models x_i \neq x_j && \text{if } \theta(x_i) \neq \theta(x_j) \\
S(s), \theta, t, m \cdot n &\models Q_1 \wedge Q_2 && \text{if } S(s), \theta, t, m \models Q_1 \text{ and } S(s), \theta, t, n \models Q_2 \\
S(s), \theta, t, \textstyle\sum_{v \in \mathbf{D}(t)} n_v &\models \exists x.Q && \text{if } S(s), \theta[v/x], t, n_v \models Q \\
S(s), \theta, t, 1 &\models \varepsilon\, Q && \text{if } S(s), \theta, t, n \models Q \\
S(s), \theta, t, n + m &\models Q_1 \vee Q_2 && \text{if } S(s), \theta, t, m \models Q_1 \text{ and } S(s), \theta, t, n \models Q_2 \\
S(s), \theta, t, \max(0, m - n) &\models Q_1 \wedge \neg Q_2 && \text{if } S(s), \theta, t, m \models Q_1 \text{ and } S(s), \theta, t, n \models Q_2 \\
S(s), \theta, t, n &\models \bullet Q && \text{if } S(s), \theta, t - 1, n \models Q \text{ and } t > 0 \\
S(s), \theta, t, n &\models \mu X.Q && \text{if } S(s), \theta, t, n \models Q[X/\mu X.Q]
\end{aligned}
$$

Figure 2. SQL-style Duplicate Semantics for Past μTL Streaming Queries.

prefix of a data stream $S(t)$ such that S_i contains the value a duplicated m_i times, $m_i \neq m_j$ for $0 \leq i \neq j \leq t$. To be able to answer the above query when the stream is extended by the state S_{n+1} we need at least a set of values $\{m_0, \ldots, m_t\}$. To represent this set we need at least

$$\sum_{i=0}^{t} \log(m_i) \geq t \cdot \log(\min\{m_0, \ldots, m_t\}) \in \Omega(|\mathbf{T}(t)|)$$

bits.

The reason for this lower bound can be traced to 2-FOL's ability to refer to individual past states of S using temporal variables, the same result thus applies to two-dimensional temporal logics when evaluated under duplicate semantics.

3.2 Fixpoint Temporal Logic Queries

The second language for which bounded synopses exist in the set semantics case is the past fragment of first-order fixpoint temporal logic (also called μ-calculus), a logic based on implicit access to the temporal attribute of tuples using *temporal operators*. This way the number of coexisting temporal contexts is limited while still commanding a sufficient expressive power. The syntax of the language is defined as follows:

Definition 3.3 (Fixpoint Temporal Logic μTL) Let S be a data stream of k-tuples. The syntax of *Past* μTL *queries* over S is given by the following grammar.

$$
\begin{aligned}
Q \quad ::= \quad & S(x_1, \ldots, x_k) \mid x_i = x_j \\
\mid \quad & Q \wedge Q \mid \exists x_i.Q \mid \varepsilon\, Q \\
\mid \quad & Q \vee Q \mid Q \wedge \neg Q \\
\mid \quad & \bullet Q \mid \mu X.Q \mid X
\end{aligned}
$$

Similarly to 2-FOL, we require the standard conditions that guarantee range-restrictedness for variables to hold in the queries. In addition, for the fixpoint variables X, we require for them to appear in the scope of an even number of negations and in the scope of at least one $\bullet$ (previous time) operator[1].

Note that formulas of μTL do not use variables ranging over the temporal domain; handling of this aspect of the queries is *encapsulated* in the *temporal operator* $\bullet$. Thus the semantics is defined *with respect to an evaluation point* $s \in \mathbf{T}(t)$ using a satisfaction relation

$$S(t), \theta, s, n \models Q$$

which, similarly to Definition 3.1, states that *the tuple θ is an answer to Q at time s with n duplicates in $S(t)$*. Note that the time point s may be different from t; this allows referring to past states of the data stream S in queries.

An answer to a μTL query Q is defined as

$$S_t^Q := \{\underbrace{\theta, \ldots, \theta}_{n} \mid S(t), \theta, t, n \models Q\}.$$

The duplicate semantics of μTL mimics that of 2-FOL introduced in Figure 1: first-order connectives and quantifiers are evaluated per state of S (using the so-called snapshot semantics [13]).

[1] Without the last restriction, duplicate semantics is not well defined in the presence of fixpoints. This arrangement also allows us to define the semantics of the least fixpoint operator by unfolding rather than by the equivalent, but more common fixpoint iteration. This is convenient for the development of the synopses for fixpoint queries.

Lower Bound for μTL

Unlike 2-FOL, μTL, by the virtue of using fixpoints, can express conjunction with number of conjuncts proportional to the length of the input stream. This, in the presence of duplicates and the *multiplicative nature* of how duplicates are handled by a conjunction, leads to the following result:

Theorem 3.4 *There is a* μTL *query for which the space required by any synopsis must be bounded from below by* $\Omega(|\mathbf{T}(t)|)$.

Proof (sketch): Consider the following *propositional* query

$$Q = \mu X.(\epsilon q) \vee \bullet((\text{true} \vee \text{true}) \wedge X)$$

over a data stream reporting on the truth value of the proposition q. Under duplicate semantics, the above query returns the empty tuple (standing for true) as long as q is reported true at least once in the stream. The number of *duplicates* of this value, however, record the complete history of the stream: at time t, the s-th bit in the binary representation of the number of duplicates corresponds to the truth value of q in S_{t-s}, $s \leq t$. Note the crucial use of the duplicate semantics for disjunction: the "$\text{true} \vee \text{true}$" formula allows us to define the value 2 which, in turn allows us to *shift* the bits in the binary representation of the duplicate count.

Example 3.5 Consider a prefix $S(6)$ of a data stream S:

$$S(6) = \langle\ q, \neg q, \neg q,\ q,\ q,\ q\ \rangle$$

at time 6. Then $S(6), \{\,\}, 6, 39 \models Q$ where $\{\,\}$ stands for the empty tuple as 39 in binary is 100111. Note that the result holds even when the input stream is duplicate-free.

Unlike the 2-FOL case, the reason for this lower bound can be traced to the way new duplicate values are *constructed* in queries from old values, in particular by conjunction.

An unfortunate consequence of these lower bounds is that, in general, the best synopses for queries expressed in common streaming languages when evaluated under duplicate semantics are essentially as big as the original data stream and thus we may be better off simply storing the stream itself, perhaps with the help of a standard lossless compression. These results also imply a linear lower space bound for streaming query languages that allow the use of the counting aggregate.

4 Upper Bounds

We now investigate restrictions in which logarithmic bounds on the size of holistic synopses can be obtained. To this end, we need to restrict the streaming query languages. We consider two different fragments of first-order logic:

1. restricted fixpoint temporal logic queries, and
2. positive first-order queries (unions of conjunctive queries).

In both cases our aim is to make the proof of Theorems 3.2 and 3.4 inapplicable. In the first case by disallowing negation and multiple temporal contexts to exist at the same time, in the second case by restricting the number of duplicates in answers to queries. The proposed synopsis maintains a sufficient amount of *temporal* information needed to evaluate a given restricted μTL query as an instance of an *extended database schema*; the approach is an extension of [7, 18] to duplicate semantics.

Definition 4.1 (Extended Database Schema) Let φ be a μTL query over a stream S. We define

$$\{\, R^{\psi}(n, x_1, \ldots, x_k) : \bullet\psi \text{ is a subformula of } \varphi \text{ with free variables } x_1, \ldots, x_k \,\}$$

to be the *extended schema* for φ, in which queries can refer to streams R^{ψ} in addition to S. We call the new predicates R^{ψ} *auxiliary*; its first argument is the explicit representation of the number of duplicates.

The synopsis is defined as a set of *materialized view* maintenance rules for R^{ψ}. The definition of the rules is based on the *unfolding property* of the fixpoint operator, $\mu X.Q \equiv Q(\mu X.Q)$. Before we define the operator itself, we need an auxiliary technical definition.

Definition 4.2 Let φ be a μTL formula of the form $\mu X.\psi$ such that ϑ is a subformula of ψ containing a free occurrence of the fixpoint variable X. We define the *X-closure of ϑ in φ* to be the formula $\vartheta[X/\mu X.\psi]$, i.e., in which all occurrences of X have been replaced by their fixpoint *definition*.

For ψ a subformula of φ with free fixpoint variables $X_1, \ldots, X_k$ we define the *closure (with respect to the fixpoint variables and φ)* to be the μTL formula

$$\psi[X_1/\mu X_1.\psi_1, \ldots, X_k/\mu X_k.\psi_k]$$

where $\mu X_i.\psi_i$ are subformulas of φ (assuming all fixpoint variables in φ are distinct).

This definition guarantees that, for every subformula ψ of a given μTL query φ, there is a μTL formula ψ' in which all fixpoint variables X_i are in the scope of an appropriate

fixpoint operator. Due to the unfolding rule, such a formula always exists and is equivalent to the original formula in the context of φ.

Definition 4.3 (Synopsis for Q) Let Q be a μTL query and $\bullet\psi_1, \ldots, \bullet\psi_l$ be all temporal subformulas of Q. We define a synopsis $\mathcal{E}$ for Q as follows:

- We define $\emptyset$ to be a constant instance of the extended schema defined by $R^{\psi_i} = \emptyset$.
- The Δ operator consists of *rematerialization rules* $\delta^{\bullet\psi_i}$ for $R^{\bullet\psi_i}$ defined by:

  $$\delta^{\psi_i}(S_t, \mathcal{E}(S(t-1))) = \psi_i^{\mathcal{E}}(S_t, R^{\psi_1}, \ldots, R^{\psi_l})$$

  where $\mathcal{E}(S(t-1))$ is an instance of the auxiliary relations $R^{\psi_1}, \ldots, R^{\psi_l}$ and $\psi_i^{\mathcal{E}}$ is the closure of ψ_i in which all occurrences of temporal subformulas $\bullet\vartheta$ were replaced by R^{ϑ}. The operator Δ is then defined by a simultaneous update of the instances of $(R^{\psi_1}, \ldots, R^{\psi_l})$ by the results of $(\delta^{\psi_1}, \ldots, \delta^{\psi_l})$.
- We define Γ to be the query Q in which all occurrences of temporal subformulas $\bullet\vartheta$ were replaced by R^{ϑ}.

$\mathcal{E}$ is then defined by the triple $(\emptyset, \Delta, \Gamma)$ as required.

Theorem 4.4 *$\mathcal{E}$ is a synopsis for PastμTL. Moreover, $\mathcal{E}$'s components Γ, $\emptyset$, and Δ are first-order queries.*

Proof (sketch): Correctness,

$$Q(S(t)) = \Gamma(S_t, \mathcal{E}(S(t-1)),$$

follows by induction on the length of $S(t)$ and the observation that the view maintenance rules $\psi_i^{\mathcal{E}}$ maintain the instance of R^{ψ} to be equivalent to $\bullet\psi$ in every state of S.

Restricting μTL

Unfortunately, as indicated by Theorem 3.4, the above construction alone is insufficient to guarantee that the constructed synopsis is log-bounded. Indeed, the above synopsis is linear in the length of the stream prefix due to the space needed to store the number of duplicates (even when represented in binary; cf. Example 3.5).

To prevent this from happening, we need to restrict the μTL queries allowed to those that limit the growth of the duplicate values when fixpoints are involved.

Definition 4.5 (Duplicate-bounded μTL) Let Q be a μTL query. We say that Q is *duplicate-bounded* if there is a constant c such that $\mathsf{dupl}(Q'[X/R])) \leq \mathsf{dupl}(R) + c$ for every subformula $\mu X.Q'$ of Q and R a relational instance with the same schema as Q'.

The duplicate-boundedness property of queries is undecidable (follows immediately from reduction from query emptiness). Hence, syntactic restriction on the appearances of fixpoint variables in the queries must be imposed. For the remainder of this paper we assume that the fixpoint variables may only appear in conjunctions in which all other conjuncts are duplicate free[2], under duplicate elimination operator, or in disjunctions with arbitrary other formulas.

Theorem 4.6 *Let Q be an arbitrary duplicate bounded μTL query. Then there is a* log*-bounded synopsis for Q.*

Proof (sketch): Follows from Theorem 4.4 by observing that, due to the restriction imposed by Definition 4.5, the values representing the number of duplicates in the auxiliary relations are bounded by $\log(|\mathbf{T}(t)|)$.

4.1 First-order Fragment of μTL: FOTL

We consider the first-order fragment of past μTL (denoted FOTL): a logic that uses the $\bullet$ and **since** connectives. The **since** connective is defined by a fixpoint unfolding. However, to guarantee duplicate-boundedness (and a reasonable interpretation of what the number of duplicates represents) the way duplicates are generated by the **since** connective has to be carefully controlled. This consideration yields the following definitions:

$$\begin{aligned} Q_1 \ \mathbf{since}_1 \ Q_2 &= \mu X.(\epsilon Q_1) \wedge (Q_2 \vee \bullet X) \\ Q_1 \ \mathbf{since}_2 \ Q_2 &= \mu X.(Q_1 \wedge (\epsilon Q_2)) \vee \\ &\qquad (Q_1 \wedge ((\epsilon \bullet X) \wedge \neg Q_2)) \vee \\ &\qquad ((\bullet X) \wedge ((\epsilon Q_1) \wedge \neg Q_2) \end{aligned}$$

The two variants, $\mathbf{since}_1$ and $\mathbf{since}_2$, differ in the way they handle duplicates: $\mathbf{since}_1$ counts the number of times the tuple θ satisfies Q_2 such that Q_1 was true since then; $\mathbf{since}_2$ counts the number of the answers θ satisfying Q_1 since Q_2 was true for the last time[3]. Note that both $\mathbf{since}_1$ and $\mathbf{since}_2$ have been chosen in such a way that the corresponding fixpoint formulae are duplicate bounded (this, however, would not be true if the classical fixpoint unfolding without the careful use of duplicate elimination were to be used).

[2] Answers to these subqueries must be sets; this can be achieved syntactically by requiring a top-level duplication elimination operator.

[3] Similar to the duplicate semantics for SQL, this is just a particular a way of choosing how many duplicates are in an answer to a particular query. The definitions above work correctly with the rest of the technical development in this paper. A comprehensive study of various possibilities to define duplication of tuples for temporal queries is beyond the scope of this paper.

Additional temporal connectives can be derived from the **since** and ● operators; again, the handling of duplicates is the only concern here.

Example 4.7 The *sometime in the past* (◆) connective can be defined as follows:

$$\blacklozenge_1 Q = \text{true } \mathbf{since}_1 \ Q$$
$$\blacklozenge_2 Q = \text{true } \mathbf{since}_2 \ Q$$

The two variants differ again in how duplication of results is defined: in the first case it indicates how many times Q has been true in the past and the second case how far in the past was the last Q has been[4]. Again additional connectives can be defined by combining the above two. For example, should we wish to know how far in the past the earliest Q appeared, we could write $(\neg Q)$ $\mathbf{since}_2$ Q.

Note that one might be tempted to define the duplicate semantics for, e.g., the *sometime in the past* (◆) operator as follows:

$$S(t), \theta, t, n \models \blacklozenge Q \text{ if } S(t), \theta, s, n \models \blacklozenge Q \text{ for a } s < t.$$

This definition would seemingly allow formulating the query "*were there two time instants with the same number of elements in the stream?*". However, it is important to see that such a definition is *incompatible* with the definition of duplicate semantics: it allows assigning two different duplicities to the same tuple—this is illegal under the duplicate semantics.

Example 4.8 The upper bound for duplicate bounded μTL also matches the lower bound from Section 3 as the query "*have there been more a values than b values in the stream S*" can be formulated in μTL using the formula

$$(\blacklozenge_1 S(a)) \wedge \neg(\blacklozenge_1 S(b)).$$

Thus the above technique is (worst-case) optimal up to a constant factor (w.r.t. $|\mathbf{T}(t)|$).

The auxiliary views for the query above are derived by translating the $\blacklozenge_1$ connectives to μTL as follows:

$$\begin{aligned}\blacklozenge_1 S(a) &= \text{true } \mathbf{since}_1 \ S(a) \\ &= \mu X.(\epsilon(\text{true})) \wedge (S(a) \vee \bullet X) \\ &= \mu X.(S(a) \vee \bullet X),\end{aligned}$$

similarly for $\blacklozenge_1 S(b)$. For readability, we label the two induced auxiliary relations by the original temporal subformulas: $R^{\blacklozenge_1 S(a)}(n)$ and $R^{\blacklozenge_1 S(b)}(n)$; note that both the views have single integer attribute since the original subqueries are closed formulas.

[4]Assuming the constant *true* is not duplicated.

The Δ operator is therefore defined by

$$R_t^{\blacklozenge_1 S(a)} := \{n+m \mid S(t), \{\}, m, t \models S(a), \\ R(t), \{\}, n, t-1 \models R^{\blacklozenge_1 S(a)}\}$$

where the stream $R^{\blacklozenge_1 S(a)}$ is the stream generated for the auxiliary relation by the above definition. A similar definition is used for $R^{\blacklozenge_1 S(b)}(n)$. Note that as streams, the auxiliary relations in this example are 0-ary—conceptually they contain the *true* tuple duplicated an appropriate number of times; the actual binary representations therefore contain a single integer value in each case.

The Γ part is then defined as $R^{\blacklozenge_1 S(a)} \wedge \neg R^{\blacklozenge_1 S(b)}$, which reduces to the numerical difference of the two counts held in the auxiliary relations.

4.2 Conjunctive and Positive Queries

Queries in μTL provide a powerful way of querying data streams. They also generalize the so called *windowed* queries in a natural way. However, there is a mismatch between streaming query languages that use SQL-like syntax (with explicit access to the time instant attributes [4]) and μTL.

In this section we therefore look on common sublanguages of 2-FOL, namely on conjunctive queries (CQ) and unions of conjunctive queries (UCQ). We show that, the technique developed for μTL can be adopted to both CQ and UCQ.

Definition 4.9 (Conjunctive Query) A *conjunctive query* is an expression of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x_1}), \wedge \ldots \wedge S(t_k, \bar{x_k}) \wedge \phi \wedge \psi$$

where $\bar{x_i}$ are vectors of data variables, ϕ is a conjunction of equalities over the data variables and ψ a ordering condition over the temporal variables. The query can be potentially prefixed by duplicate elimination.

Note that we require the answers to CQ not to contain any free temporal variables in order for them to serve as continuous queries over data streams.

The construction of synopses for CQ proceeds in two steps:

1. First a given CQ is rewritten to an equivalent union of CQs, such that the condition ψ in each of the constructed queries imposes a linear order among the variables $t_1, \ldots, t_k$.
2. Second, each of the above queries is translated to (the first-order fragment of) μTL and then the synopsis construction for μTL is used.

This approach is supported by the following two lemmas:

Lemma 4.10 Let Q be a CQ of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1), \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \psi.$$

Then there is a finite set of CQ such that

- the (duplicate preserving) disjunction of these CQ is equivalent to the original CQ and
- the subformulas ψ in these queries impose a linear order on valuations of the variables $t_1, \ldots, t_k$.

Proof (sketch): Let Ψ be the set of all formulas that express linear orders over $t_1, \ldots, t_k$ consistent with ψ. This set is finite and due to the law of excluded middle, no two elements of Ψ can be made true by the same valuation. Thus the set of CQ

$$\{\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1) \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \varphi \mid \varphi \in \Psi\}$$

fulfills the requirements of the Lemma.

Thus, for each pair of variables, we have $t_i < t_j$, $t_i = t_j$, or $t_i > t_j$. This allows us to construct a μTL formula by using the $\blacklozenge_1$ connective to simulate the inequalities. Hence the following Lemma:

Lemma 4.11 Let Q be a CQ of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1), \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \varphi.$$

in which φ imposes a linear order on the temporal variables $t_1, \ldots, t_k$. Then there is an equivalent formula in μTL.

Proof (sketch): We replace the CQ by a μTL query of the form

$$\begin{aligned}\exists \bar{y}.&\blacklozenge_1(S(\bar{x}_{i_1}) \wedge \ldots \wedge S(\bar{x}_{i_l}) \wedge \\ &\blacklozenge_1(S(\bar{x}_{j_1}) \wedge \ldots \wedge S(\bar{x}_{j_{l'}}) \wedge \ldots \wedge \\ &\blacklozenge_1(S(\bar{x}_{k_1}) \wedge \ldots \wedge S(\bar{x}_{k_{l''}}) \wedge) \ldots))\end{aligned}$$

where the subscripts $i_1, \ldots, i_l$ refer to those conjuncts in the original query that are first in the linear ordering of temporal variables, $j_1, \ldots, j_{l'}$ to the second, and $k_1, \ldots, k_{l''}$ to the last, i.e., the linear order of the temporal variables was

$$\begin{aligned}t_{i_1} = \ldots = t_{i_l} > t_{j_1} = \ldots = t_{j_{l'}} > \ldots \\ > t_{k_1} = \ldots = t_{k_{l''}}\end{aligned}$$

in φ.

We finish the construction by applying the approach to construction of bounded synopses introduces in Section 3.2. The use of this approach for UCQ is immediate.

5 Conclusion

In this paper we have shown that duplicate semantics causes severe difficulties in constructing bounded synopses for processing continuous queries. Indeed, even simple queries may require a synopsis (linearly) proportional to the length of the original stream—which defeats the usefulness of such synopsis. We have also developed restricted fragments of streaming queries that allow logarithmically-bounded synopses to be used and shown how such synopses can be constructed.

Future Directions of Research

There are many possible directions of future research, among which are the following:

Alternatives to standard duplicate semantics. In this paper we mainly considered the standard *SQL-style* approach of defining duplicate semantics of queries. However, beyond compatibility concerns, there is no principal reason why other numerical functions, such as the "min" and "max" functions, couldn't be used to define the duplicate semantic for conjunctions and disjunctions, respectively. Note that such an arrangement makes all μTL queries *duplicate bounded* and hence the lower bound derived for SQL-style semantics in Section 3.2 no longer applies to μTL. The main open question is, however, whether such a plausible duplicate semantics can exist for first-order queries (i.e., for which Theorem 3.2 no longer applies.)

2-FOL queries with $\log$-bounded synopses. We have shown two sub-languages of 2-FOL queries for which $\log$-bounded synopses can be constructed: FOTL and UCQ. However, it is not clear whether it is possible to syntactically characterize those 2-FOL queries (possibly up to query equivalence) for which $\log$-bounded synopses exist.

Aggregates. Another question relates to the possibility of introducing *aggregate functions* into the query language—again, the best lower bounds we know today are logarithmic in the length of the stream. However, the techniques proposed in this paper cannot cope with aggregates mainly since such functions introduce new *domain elements*. Moreover, unlike the duplicate counts, aggregates can associate multiple *count* values with a single tuple in the query.

Possible and certain answers. Yet another direction of research is to study fragments of query languages for which the possible result semantics is viable.

Also, the focus of this paper was on developing techniques that allow *precise* answers to continuous queries to be computed. Another direction of research is to consider appropriate ways to *approximate* the answers and trade-offs between quality of the approximations and space needed for storing synopses. While there has been a large amount of work in this area, surveying the issues connected with approximate query answers is beyond the scope of this paper.

References

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

[2] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing Memory Requirements for Queries over Continuous Data Streams. In *ACM Symposium on Principles of Database Systems*, pages 221–232, 2002.

[3] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Trans. Database Syst.*, 29(1):162–194, 2004.

[4] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, 2003-67, Stanford University, 2003.

[5] A. Ayad and J. F. Naughton. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In *ACM SIGMOD International Conference on Management of Data*, pages 419–430, 2004.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.

[7] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, June 1995.

[8] J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *J. Comput. Syst. Sci.*, 51(3):523–535, December 1995.

[9] J. Chomicki and D. Toman. Temporal Databases. In M. Fischer, D. Gabbay, and L. Villa, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 429–467. Elsevier *Foundations of Artificial Intelligence*, 2005.

[10] L. Golab and M. T. Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 500–511, 2003.

[11] J. Kang, J. F. Naughton, and S. Viglas. Evaluating Window Joins over Unbounded Streams. In *International Conference on Data Engineering (ICDE)*, pages 341–352, 2003.

[12] F. Korn, S. Muthukrishnan, and Y. Zhu. Checks and Balances: Monitoring Data Quality Problems in Network Traffic Databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 536–547, 2003.

[13] R. T. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. A. Sripada. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, Mar. 1994.

[14] U. Srivastava and J. Widom. Memory-Limited Execution of Windowed Stream Joins. In *International Conference on Very Large Data Bases (VLDB)*, pages 324–335, 2004.

[15] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load Shedding in a Data Stream Manager. In *International Conference on Very Large Data Bases (VLDB)*, pages 309–320, 2003.

[16] D. Toman. Expiration of Historical Databases. In *International Symposium on Temporal Representation and Reasoning*, pages 128–135. IEEE Press, 2001.

[17] D. Toman. Logical Data Expiration. In J. Chomicki, G. Saake, and R. van der Meyden, editors, *Logics for Emerging Applications of Databases*, chapter 7, pages 203–238. Springer, 2003.

[18] D. Toman. Logical Data Expiration for Fixpoint Extensions of Temporal Logics. In *International Symposium on Advances in Spatial and Temporal Databases SSTD 2003*, pages 380–393, 2003.

[19] D. Toman. On Construction of Holistic Synopses under the Duplicate Semantics of Streaming Queries. In *Proc. Int. Workshop on Spatio-Temporal Database Management STDBM 2006*, Seoul, Korea, 2006. CEUR-WS vol. 174.