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
david@uwaterloo.ca

Abstract

Transaction-time temporal databases and query languages provide a solid framework for analyzing the 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. 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, see e.g., [2, 3, 4, 8, 9, 10, 12, 13], and many others. These efforts have mainly focused on efficient processing of *continuous queries*—queries evaluated continuously as more data is arriving on the 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 whole history of the data stream.

The 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 then shows how acceptable results can be achieved by carefully limiting the expressive power of streaming query languages in which continuous queries are formulated.

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 construct synopses on a per-physical-operator basis (e.g., for the so called symmetric joins, etc., [2]), we develop techniques that tailor the synopsis to the 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 [15]. 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; in the general case, the synopses may have to grow at least linearly with the stream length.

2. We show that for certain limited languages, this growth can be tamed to a logarithmic factor; that factor, however, cannot be avoided.

Note that the negative results presented in this paper are based on information-theoretic properties of queries and thus cannot be improved upon by more sophisticated algorithms without resorting to approximations. We also contrast these results with 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 provides a strong parallel between

Proceedings of the third Workshop on STDBM
Seoul, Korea, September 11, 2006

techniques developed for transaction-time temporal databases [7], data expiration [15], and approaches to efficient evaluation of streaming queries.

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. We conclude with identifying open issues in the area in Section 5.

2 Background

We first review the relevant definitions in the area of transaction-time temporal databases and link them to querying data streams. The presentation in this section is based on a chapter on data expiration [15] with terminology suitably modified to data streams.

2.1 Temporal Queries for Data Streams

We first formalize the notion of data stream as follows.

Definition 2.1 (Data Stream/History) Let σ be a relational schema. A *data stream* is a sequence

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

where each S_t is a multiset of tuples conforming to σ 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 allow multiple tuples to arrive at the same time instant. The data values forming the tuples belong to the domain of *uninterpreted constants*; the data domain is equipped with equality only. At any particular finite time t, we have access only 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), 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, just the set $\{0, \ldots, t\}$; the definition, however, allows to use timestamps from any linearly ordered set.

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 just a variant name for an *append-only* temporal database (often called a *transaction-time temporal database*). This observation allows us to use off-the shelf temporal query languages to query data streams. In this paper we use *two-sorted first-order logic* (2-FOL) to query such streams:

Definition 2.2 (Streaming/Temporal Queries) Let S be a data stream. 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*. In addition, we assume that the queries are *range restricted*; this is enforced by requiring the usual restrictions on the occurrences of equalities and inequalities and the variable-compatibility conditions for disjunctions ($\vee$) and negations ($\wedge\neg$). The semantics of the 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 *with* n *duplicates, when evaluated over* $S(t)$*, a prefix of* S". The full semantics of the above language is given in Figure 1. In particular, the semantics specifies how duplicates are handled in queries[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., $\theta(x)$ functionally determines n. To simplify the definition, we assume that substitutions not present in an answer have 0 duplicates.

Again, the query language in Definition 2.2 is just a temporal query language when the stream is regarded as finite prefix of a database history. Note also, that

[1] We utilize a SQL-style definition of duplicates, i.e., a product for conjunctions, sum for disjunction (union) and existential quantification (projection), and difference for range-restricted negation (set difference).

$$
\begin{aligned}
S(t), \theta, n &\models S(t, x_1, \ldots, x_k) && \text{if } \langle \theta(x_1), \ldots, \theta(x_k) \rangle \in S_{\theta(t)} \text{ duplicated } n \text{ times} \\
S(t), \theta, 1 &\models x_i = x_j && \text{if } \theta(x_i) = \theta(x_j) \\
S(t), \theta, 1 &\models t_i < t_j && \text{if } \theta(t_i) < \theta(t_j) \\
S(t), \theta_1 \circ \theta_2, m \cdot n &\models Q_1 \wedge Q_2 && \text{if } S(t), \theta_1, m \models Q_1 \text{ and } S(t), \theta_2, n \models Q_2 \\
S(t), \theta, \textstyle\sum_{v \in \mathbf{D}(t)} n_v &\models \exists x.Q && \text{if } S(t), \theta[v/x], n_v \models Q \\
S(t), \theta, \textstyle\sum_{s \in \mathbf{T}(t)} n_s &\models \exists t.Q && \text{if } S(t), \theta[s/t], n_s \models Q \\
S(t), \theta, 1 &\models \varepsilon\, Q && \text{if } S(t), \theta, n \models Q \\
S(t), \theta, n + m &\models Q_1 \vee Q_2 && \text{if } S(t), \theta, m \models Q_1 \text{ and } S(t), \theta, n \models Q_2 \\
S(t), \theta, \max(0, m - n) &\models Q_1 \wedge \neg Q_2 && \text{if } S(t), \theta, m \models Q_1 \text{ and } S(t), \theta, n \models Q_2
\end{aligned}
$$

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

the semantics 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 [6], for detailed discussion of this phenomenon see [7].

Definition 2.3 (Continuous Query Answer) Let Q be a query without free temporal variables. An *answer to a continuous query* specified by Q is defined as a 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 $S(t)$ as defined by the semantics in Figure 1.

The restriction to queries without free temporal variables is needed to ensure that results of a query form a proper data stream. This arrangement allows for compositionally, possibility of view definitions, etc. Note also that the representation of duplicates in *binary* (i.e., using counts to represent the numbers of duplicates) is a more compact representation than explicitly duplicating the tuples. Thus all lower bounds derived in this paper also hold for the SQL-style representation in unary (i.e., by explicit replication of the tuples in question).

2.2 Holistic Synopses and Data Expiration

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

Definition 2.4 (Holistic Synopsis) Let Q be a query over S. A *holistic synopsis* for Q is a triple $(\emptyset, \Delta, \Gamma)$ that satisfies the following property:

$$Q\langle S_0, \ldots, S_t \rangle = \Gamma(\Delta(S_t, \Delta(S_{t-1}, \Delta(\ldots \Delta(S_0, \emptyset)))))$$

for any prefix $\langle S_0, \ldots, S_t \rangle$ of S. 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 now generates the answers to Q 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. Note that this definition is essentially the same as the definition of a *data expiration operator* for transaction-time temporal databases [15].

How do we compare Holistic Synopses?

Intuitively, we have replaced the complete prefix of S with a materialized view 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, $|\mathbf{D}(t)|$, and
3. 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. We call a synopsis log-*bounded* if it is bounded by a function logarithmic in $|\mathbf{T}(t)|$.

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

Proposition 2.5 ([14]) *A bounded holistic synopsis exists for any two sorted first order streaming (2-FOL) query* 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 language introduced in Definition 2.2. The rest of the paper argues that bounded synopses do not exist when duplicate semantics is used *for the same language* and that log-bounded synopses are the best we can hope for in various fragments of the first-order language under duplicate semantics.

3 Lower Bounds

Now we are ready to provide simple 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 [14]. Similar query, e.g.,

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

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

However, it turns out that log-bounded synopses are not sufficient for first-order queries with duplicate semantics.

Theorem 3.1 *There is a first-order query for which any synopsis is bounded from below by* $\Omega(|\mathbf{T}(t)|)$.

Proof (sketch): Consider the query

$$\varepsilon\exists t_1,t_2.t_1<t_2 \wedge \neg((\exists x.S(t_1,x)) \wedge \neg(\exists x.S(t_2,x))) \\ \wedge\neg((\exists x.S(t_2,x)) \wedge \neg(\exists x.S(t_1,x)))$$

The query expresses the condition *two states of S contain the same number of tuples*. Now consider a 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 unfortunate consequence of this lower bound is that, in general, the best synopsis for first-order queries under duplicate semantics are essentially as big as the original data stream (and thus we may be better off just storing the stream itself).

4 Upper Bounds

We now investigate cases 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. positive first-order queries, and
2. temporal logic queries.

In both cases our aim is to make the proof of Theorem 3.1 inapplicable. In the first case by disallowing negation and in the second by disallowing multiple temporal contexts to exist at the same time.

4.1 First-order Temporal Logic Queries

We start with the case of the past fragment of the *first-order temporal logic* (FOTL), a logic based on implicit access to the temporal attribute of tuples using *modal operators*. This way the number of coexisting temporal contexts is limited while still commanding a sufficient expressive power[2]. The syntax of the language is defined as follows:

Definition 4.1 (First-order Temporal Logic) Let S be a data stream. The syntax of *FOTL queries* over S is given by the following grammar.

$$\begin{array}{rcl} Q & ::= & S(x_1,\ldots,x_k) \mid x_i = x_j \\ & \mid & Q \wedge Q \mid \exists x_i.Q \mid \varepsilon\, Q \\ & \mid & Q \vee Q \mid Q \wedge \neg Q \\ & \mid & Q \textbf{ since } Q \mid \bullet Q \end{array}$$

Note that formulas of FOTL do not use variables ranging over the temporal domain; handling of this aspect of the queries is *encapsulated* in the *temporal operators* **since** and $\bullet$ (previous time). Thus the semantics is now defined *with respect to an evaluation point* using a satisfaction relation

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

which, similarly to Definition 2.2, 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. The semantics of FOTL mimics that of 2-FOL introduced in Figure 1: all the standard first-order connectives and quantifiers are evaluated per state of S (using the so-called snapshot semantics [11]). The only addition are the rules for handling the temporal

[2]It has been shown, however, that FOTL is strictly less expressive fragment of 2-FOL, the language introduced in Definition 2.2 [1, 16].

operators; here we need to extend the standard definitions to handle *duplicates*. The semantics of the **since** operators (there are two variants to account for two ways to determine the number of duplicates) is defined as follows:

$S(t), \theta, s, \max_{s_2 \in \mathbf{T}} \sum_{s_1 \in \mathbf{T}(t)} m_{s_1} \models Q_1$ **since**$_1$ Q_2 if
$S(t), \theta, n_{s_2}, s_2 \models Q_2$ for some $s_2 < s, n_{s_2} > 0$ and
$S(t), \theta, m_{s_1}, s_1 \models Q_1$ for all $s_1 < s_2 \leq s, m_{s_1} > 0$

$S(t), \theta, s, \sum_{s_2 \in \mathbf{T}(t)} n_{s_2} \models Q_1$ **since**$_2$ Q_2 if
$S(t), \theta, n_{s_2}, s_2 \models Q_2$ for some $s_2 < s, n_{s_2} > 0$ and
$S(t), \theta, m_{s_1}, s_1 \models Q_1$ for all $s_1 < s_2 \leq s, m_{s_1} > 0$

For the previous time operator, $\bullet Q$, the duplicate semantics is defined as follows:

$$S(t), \theta, s, n \models \bullet Q \text{ if } S(t), \theta, s-1, n \models Q \text{ and } s > 0$$

An answer to a continuous query Q at time t is defined as the multiset

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

The two variants, **since**$_1$ and **since**$_2$, differ in the way they handle duplicates: **since**$_1$ counts the maximal number of the answers θ satisfying Q_1 since Q_2 was true; **since**$_2$ counts the number of times the tuple θ satisfies Q_2 such that Q_1 was true since then[3].

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

Example 4.2 The *sometime in the past* ($\blacklozenge$) 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 far in the past was the earliest Q has been and the second case how many times Q has been true in the past[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 latest Q was we can write $(\neg Q)$ **since**$_1$ Q.

[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.

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

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

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

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.

Synopses for FOTL

To define a synopsis for a given FOTL formula we use the following identities:

Lemma 4.3 Let S be a data stream and Q_1 and Q_2 FOTL queries. Then

$S(t), \theta, s, n+m \models Q_1$ **since**$_1$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_1$ Q_2 and
- $S(t), \theta, s, m \models Q_1$.

$S(t), \theta, s, m \models Q_1$ **since**$_1$ Q_2
- $S(t), \theta, s-1, n \models Q_2$,
- $S(t), \theta, s-1, l \not\models Q_1$ **since**$_1$ Q_2, and
- $S(t), \theta, s, m \models Q_1$.

$S(t), \theta, s, n+m \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

$S(t), \theta, s, m \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \not\models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

$S(t), \theta, s, n \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \not\models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

Proof (sketch): Immediate from the definitions.

With the help of these identities we can modify the approach to bounded checking of past FOTL constraints [5] as follows:

Definition 4.4 Let Q be a FOTL query and $\{\alpha_1, \ldots, \alpha_k\}$ all of its temporal subformulas (i.e., formulas rooted by a temporal operator). We define auxiliary views R^{α_i}, one for each of the subformulas α_i. The attributes of R^{α_i} correspond to the free variables in α_i with an additional distinguished *integer-valued* attribute n that accounts for the duplication of tuples.

The auxiliary views are initialized to an empty set each (this formally corresponds to the $\emptyset$ component of the synopsis) and then, whenever a new state of S arrives, the views are rematerialized using the rules in Lemma 4.3 (the Δ component). To evaluate the temporal subformulas in the preconditions of the rules we use the current and new instances of the auxiliary views (this imposes an ordering on the execution of the re-materializations). An answer to Q (the Γ component) is derived by executing Q after all its temporal subformulas have been replaced by the auxiliary views.

Theorem 4.5 *Let Q be a FOTL query. Then the instances of views constructed using Definition 4.4 form a $\log(|\mathbf{T}(t)|)$-bounded holistic synopsis for Q.*

Proof (sketch): Every time a new data state arrives on S, it is sufficient to access only the t-th and $(t-1)$st states of the auxiliary materialized views and the last state of S. The size of the auxiliary views depends on the size of the query (number of temporal subqueries), the size of the active data domain $|\mathbf{D}(t)|$ at time t, and $\log(|\mathbf{T}(t)|)$ due to the necessity to keep the counters counting the numbers of duplicates.

The above upper bound 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 FOTL using the formula

$$(\blacklozenge_2 S(a)) \wedge \neg(\blacklozenge_2 S(b)).$$

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

Example 4.6 The auxiliary views for the query above are $R^{\blacklozenge_2 S(a)}(n)$ and $R^{\blacklozenge_2 S(b)}(n)$; note that both the views have single integer attribute as the original subqueries are closed formulas.

The Δ operator is defined as

$$R_t^{\blacklozenge_2 S(a)} := \{n+m \mid S(t), m, t \models S(a), \\ R(t), n, t-1 \models R^{\blacklozenge_2 S(a)}\}$$

where the stream R is the stream generated for the auxiliary relations. Similar definition is used for $R^{\blacklozenge_2 S(b)}(n)$. 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 just one integer value each.

The Γ part is then defined as $R^{\blacklozenge_2 S(a)} \wedge \neg R^{\blacklozenge_2 S(b)}$.

4.2 Conjunctive and Positive Queries

Queries in FOTL 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 [2]) and FOTL. Indeed, [1, 16] show that the later are strictly less expressive. Also, the lower bound for 2-FOL in Section 3 compared to the upper bound for FOTL in Section 4 show, that this discrepancy cannot be repaired by adopting duplicate semantics.

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, in these cases the technique developed for FOTL can be adopted to CQ and UCQ.

Definition 4.7 (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 FOTL and then the synopsis construction for FOTL is used.

This approach is supported by the following two lemmas:

Lemma 4.8 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 two variables, we have $t_i < t_j$, $t_i = t_j$, or $t_i > t_j$. This allows us to construct a FOTL formula by using the $\blacklozenge_2$ connective to simulate the inequalities. Hence the following Lemma:

Lemma 4.9 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 FOTL.

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

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

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

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

in φ.

We finish the construction by applying the approach to construction of bounded synopses introduces in Section 4.1. The use of this approach for UCQ is immediate.

5 Conclusion

In this paper we have shown that duplicate semantics causes a 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.

5.1 Future Directions of Research

There are many directions of research to pursue in this direction. Among these are:

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. The main open question is whether a plausible duplicate semantics for first-order queries to which Theorem 3.1 doesn't apply exists.

2-FOL queries with log-bounded synopses. We have shown two sublanguages of 2-FOL queries for which log-bounded synopses can be constructed. 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 as aggregate functions introduce new *domain elements*.

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] Serge Abiteboul, Laurent Herr, and Jan Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 49–57, 1996.

[2] Arwind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution, 2003.

[3] Ahmed Ayada and Jeffrey 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.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.

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

[6] J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, December 1995.

[7] 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.

[8] Lukasz Golab and M. Tamer 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.

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

[10] Flip Korn, S. Muthukrishnan, and Yunyue 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.

[11] Richard 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, March 1994.

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

[13] Nesime Tatbul, Ugur Cetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load Shedding in a Data Stream Manager. In *International Conference on Very Large Data Bases (VLDB)*, pages 309–320, 2003.

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

[15] David Toman. Logical Data Expiration. In Jan Chomicki, Gunter Saake, and Ron van der Meyden, editors, *Logics for Emerging Applications of Databases*, chapter 7, pages 203–238. Springer, 2003.

[16] David Toman and Damian Niwinski. First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France, 1996.