An Algebric Window Model for Data Stream Management

Loïc Petit
Orange Labs
University of Grenoble
LIG Laboratory, France
loic.petit@orange-ftgroup.com

Cyril Labbé
University of Monash
University of Grenoble
LIG Laboratory, France
Cyril.Labbe@imag.fr

Claudia Lucia Roncancio
University of Grenoble
LIG Laboratory, France
Claudia.Roncancio@imag.fr

ABSTRACT

Querying streams of data from the sensors or other devices requires several operators. One of the most important operators is called Windowing. Creating windows consists in grouping of tuples from data streams at a specific rate according to a certain pattern. A large variety of window patterns exist and reflect different data management semantics that are useful for different purposes. Prior arts mainly focused on simple windows, like landmark and sliding windows, and only a few properties were considered in the case of query rewriting. This paper goes one step forward by proposing an algebraic model for generic windows. Our proposed model supports temporal, positional and cross-domain windows. Window's creation time can be specified by a complex function. This proposal subsumes most popular system formalizations and extends the possibilities of window management. This paper also demonstrates associativity and transposition properties useful for algebraic rewriting in query optimization. The implementation of this model is briefly presented.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics*

General Terms

Theory, Languages

Keywords

data stream, window, algebra, transposition, optimisation

1. INTRODUCTION

Data stream processing has now gained a certain level of maturity. Numerous efforts on this field have been motivated by the expansion of sensor-based applications. Several active research groups in university and institution have proposed different forms of declarative data stream processing. Some works introduced SQL extensions to stream querying [5, 4, 3, 8, 10] whereas others presented graphical interfaces to allow users to express their queries [6, 1]. The power of expression on such solutions are varied and hence they are difficult to compare. Moreover, the semantics of queries supported by different systems are not always well defined. A query Q_1 executed by two stream processing systems may lead to different results.

Our work contributes to the clarification of the semantics of operators involved in data stream queries. This work is particularly motivated by sensor data management, although it is general enough to be used in other data stream contexts. We propose Astral [2], a stream management algebra which provides a formal definition of the operators involved in the queries. Such formalization facilitates a better understanding of the queries, and as importantly for better comparison of the semantics implemented by different systems. The algebra isolates properties on the operators, which can be used for query rewriting and optimization purposes. In a broader context, such algebra can be helpful in mediation systems involving heterogeneous data stream processing such as real-time business intelligence or network monitoring.

This paper presents a portion of such formal work. It focuses on window operators which are very important in data stream management. Creating windows consists in grouping tuples from a stream at a specific rate according to a certain pattern. A large variety of window patterns exist and reflect different data management semantics that can be used for different purposes. The prior arts mainly focused on simple windows, like landmark and sliding windows, only a few properties were considered in the case of query rewriting. This paper goes one step forward by proposing an algebraic model for generic windows and some significant related properties. Our proposed model supports temporal, positional and cross-domain windows. This proposal subsumes most popular system formalization and extends the possibilities of window management. This paper also demonstrates associativity and transposition properties useful for algebraic rewriting in query optimization.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the theoretical basis of a stream algebra (Astral [2]) which are necessary to state our model. Section 4 describes the core definitions of windows over a stream. Section 5 shows properties (associativity and transposition) provided by our model. Section

MobiDE10, June 6, 2010, Indianapolis, Indiana, USA.

6 discusses validation and implementation issues. Finally, concluding remarks and future works are depicted in Section 7.

2. RELATED WORK

Data stream management systems are usually concerned with window operators for querying streams. Generally speaking, a window operator defines a finite set of items from an infinite data stream. The way such sets are created and evolve, leads to different semantics. For example, windows may reflect the items passed every 5 minutes or, the 20 last items independently of the evaluation period. Four main categories of windows have been identified in the literature [9],

- *Fixed*: the boundaries of the window are fixed. The window is evaluated only once.
- *Sliding*: the width of the window is fixed and its boundaries move linearly.
- *Tumbling*: is a particular sliding window where the boundaries move is equal to the window's width. Windows don't overlap.
- *Landmark*: the lower boundary is fixed and the upper one grows linearly.

Many works [8, 6, 3, 19, 4, 5] implement the *RANGE/ROW/SLIDE* based window definition. The *RANGE* statement indicates the time width of each window. The *ROW* statement relates with the size and *SLIDE* indicates the evaluation rate. Such definition is largely used as it allows the expression of the main categories of windows. Nevertheless, as its interpretation is ambiguous, its semantics has been clarified by the community [14]. The difference concerns, the moment when windows are evaluated and, the semantics given to tuples having the same timestamp (called *simultaneous*). A treatment based on the timestamps will lead to a different result when compared with a treatment that considers the data itself.

Other works have proposed more possibilities for windows evaluation. For instance, in SStreamWare [12], windows are defined by a 5-uplet which includes the coefficient of the linear bounds and a rate of evaluation. This allows to represent customized windows in a unified way. In the original version of TelegraphCQ [10], the SQL-like language included window definitions using a *for-loop*. Only temporal windows based on timestamp matching were supported. In the later version of the prototype, declarative descriptions based on *RANGE/ROW/SLIDE* semantics were adopted.

A first attempt of algebraic formalization of window was made in the core semantic of STREAM [7] and has been further analyzed in [16]. It was stated for the first time a clear semantics of the classical definitions. It also proposed the semantics for composite operators with windows, like the stream join by band. With this detailed formalization, some properties have been discovered, which are mainly related to the associativity between the unary operators. The first significant result was the associativity of selection and *logical window* (time based).

Despite such various efforts, further works on window descriptions are still needed. For instance, none of the previous window definitions is able to define a simple window that gathers the last 30 items each minute. The difficulty of the expression in such windows lies in the combination of a temporal rate and positional boundaries. Usual models do not provide such flexibility. This issue and the variety of semantics in window operators are the chief motivation of our work. As compared with the existing works, the proposed model in this paper covers more types of windows and mathematical properties that are useful for query rewriting.

3. ALGEBRA BASIS

This section introduces the core of Astral, our stream algebra. The notions presented here are the basis to correctly present our work on windows in Section 4.

3.1 Time and queries

Firstly, let's introduce the interpretation of time in Astral.

Definition 1 (Timespace): *The timespace $\mathcal{T}$ is a field isomorphic to $\mathbb{R}$. The timespace is naturally and totally ordered.*

A timestamp t is an element of $\mathcal{T}$.

The notion of time is the same for every sources. Therefore, we suppose that all clocks are *well-synchronized*. We are aware that this may be hard to ensure in practice in systems like sensor networks, but we wanted to focus on exact semantic. Continuous time appears as the best choice for generic stream treatment. This allows, among other, to make no hypothesis on the distance between two consecutive timestamps.

We also introduce the notion of batch (presented in [14]) which is necessary to handle simultaneous tuples (i.e., tuples having the same timestamp). The simultaneous tuples can arise when joining multiple streams, even in desynchronized systems.

Definition 2 (Batch): *A batch is a tuple-set which contains simultaneous tuples. Given a timestamp, batches form a partition of the set of all tuples having this timestamp. A tuple is part of one single batch, but two simultaneous tuples may belong to two different batches.*

Batch identifiers *are elements of the ordered set $\mathcal{T} \times \mathbb{N}$.*

For data stream management, we consider in particular continuous queries. Such queries are issued once and then logically run continuously over data streams. A continuous query starts at a certain timestamp. Section 5 shows how such timestamp can be manipulated.

Definition 3 (Query): *An Astral query, noted (Q, t_0), is defined as*

- *an algebraic expression Q involving one or multiple data streams or relations, and*
- *a start timestamp t_0.*

3.2 Streams and relations

Streams and relations are two concepts that were identified firstly in the abstract semantics of STREAM [8]. Here are the proper definitions we use.

Definition 4 (Stream): *A stream S is a possibly infinite set of tuples with a common schema containing two special attributes: t as timestamp and φ as physical identifier.*

Definition 5 (Temporal relation): *A temporal relation R is a step function that maps a batch identifier $(t, i) \in \mathcal{T} \times \mathbb{N}$ to a set of tuples $R(t, i)$ having a common schema and containing the attribute φ as physical identifier.*

By convention, $\forall t < t_0$, $\forall i \in \mathbb{N}$, $R(t, i) = \emptyset$

Classical relational operators can be easily extended to *temporal relations*. For example, selections $\sigma_c(R)$ are defined as follows $(\sigma_c R)(t,i) = \sigma_c(R(t,i))$. The set $R(t,i)$ is a (classical) relation at batch (t,i).

It is worth noting that the physical identifier φ is important in stream data management, and particularly in sensor data management. The reason is that duplicates may be meaningful for the applications. For example, sensors that are reporting the same measured value. The physical identifier also plays a central role in data ordering. It will rule the positional order in streams.

3.3 Batch functions

To define operators on streams, a precise definition of tuple's order is required. We consider a temporal order and a positional order inferred from physical identifiers. We use batches to partition data and enforce the fact that if according to the positional order a precedes b then, either a and b are simultaneous either a precedes b in the temporal order.

Definition 6 (Batch identifier on stream): *Let S be a stream. $\mathcal{B}_S : S \mapsto \mathcal{T} \times \mathbb{N}$ is the function that gives the batch identifier of the given tuple from S.*

Axiom 1 (Coherency position-timestamp): *The temporal order is coherent with the positional order:*

$$\forall s \in S, s' \in S, \quad s(\varphi) < s'(\varphi) \Rightarrow \mathcal{B}_S(s) \leq \mathcal{B}_S(s')$$

Recall, that tuples have a unique position in a stream. The following definition associates positions and timestamps. Given a tuple position, we can obtain the identifier of the batch containing that tuple.

Definition 7 (Position-timestamp mapping): *Let S be a stream, the function $\tau_S : \mathbb{N} \cup \{-1\} \rightarrow T \times \mathbb{N}$ is the function which gives the batch identifier of the only n-tuple having the given position. By convention, $\tau_S(-1) = (t_0, 0)$.*

Corollary 1: *Considering Axiom 1, the function: $\tau_S^{-1} : \mathcal{T} \times \mathbb{N} \rightarrow \mathbb{N} \cup \{-1\}$ is the pseudo inverse of τ_S. For a batch identifier (t,i) it returns the highest tuple position of that batch. If there is no batch identified by (t,i), the batch having the preceding identifier is used.*

More formally[1]*, $\forall (t,i) \in T \times \mathbb{N}$,*

$$\tau_S^{-1}(t,i) = \sum_{n=-1}^{+\infty} n \; \mathbf{1}_{[\tau_S(n), \tau_S(n+1)[}(t,i)$$

4. SUPPORTING GENERIC WINDOWS

In the previous section, the theoretical context to define operators on streams has been presented. This section focuses on the definitions to support generic windows. Our proposed method is built upon an earlier work [14]. Section 4.1 presents the basis of window sequences whereas Section 4.2 goes a step forward, with partitioned and aperiodic windows.

4.1 Window sequences and operator

In general, a fixed window is defined by a lower and an upper bound. Such bounds delimit the subset of the data stream *observed* in the window. In data stream management, sequences of windows are required to observe the data in the evolving stream. This section defines the operator to create sequences of windows (definition 10). It is based on a description of such a sequence (definition 8) and its correlation to data streams (definition 9). This correlation uses batches (introduced in section 3) that support positional and temporal windows in the presence of simultaneous tuples in the stream.

Definition 8 (Window Sequence Description): *Let $\mathcal{D}$ and $\mathcal{D}'$ be either $\mathcal{T}$ or $\mathbb{N}$, a Window Sequence Description is a triplet (α, β, r) where:*

- *$r \in \mathcal{D}$ is the boundaries evaluation rate*
- *α and β are two functions from $\mathbb{N} \rightarrow \mathcal{D}'$ representing the boundaries evolution.*

$\alpha(j)$ and $\beta(j)$ define the j^{th} values for the boundaries. The first values are given for $j = 0$.

These functions must verify the following properties (stated here for $\mathcal{D} = \mathcal{D}' = \mathcal{T}$):

$$\forall j \in \mathbb{N}, \begin{cases} \alpha(j) \leq \beta(j) & \text{the beginning is before the end} \\ \alpha(j) \geq t_0 & \text{the beginning exists} \\ \beta(j) \leq jr + \beta(0) & \text{the end is accessible} \end{cases}$$

By applying τ_S (or τ_S^{-1}), these conditions for other values of $\mathcal{D}$ and/or $\mathcal{D}'$ are easy to find.

Example 1: Let's consider some kind of monitoring which considers that per each 100 passed items, it extracts the last 10 items. For this case, we require a sequence of positional windows generated for every 100 items ($r = 100$) where each window contains the 10 last items. We handle full positional windows, $\alpha, \beta \in (\mathbb{N} \rightarrow \mathbb{N})^2$. The first window covers from the 91th item to 100th item: $\alpha(0) = 91$ and $\beta(0) = 100$. The boundaries evolve linearly as follows:

$$\begin{aligned} \alpha(j) &= 100j + 91 \\ \beta(j) &= 100(j+1) \\ r &= 100 \end{aligned}$$

Window creation requires mapping of the stream tuples into the corresponding window based on the given sequence description. For this, we use the following delaying function which includes batch identifiers. Based on the window sequence description, this function gives the "rank" of the last created window at the moment indicated by the given batch identifier.

Definition 9 (Delaying function): *The delaying function is a function from $\mathcal{T} \times \mathbb{N} \rightarrow \mathbb{Z}$ that maps the batch identifier with the id of last valid boundaries.*

- *If $r \in \mathcal{T}$, this function is $\gamma : t, i \mapsto \left\lfloor \frac{t - \beta(0)}{r} \right\rfloor$.*
- *If $r \in \mathbb{N}$, this function is $\gamma : t, i \mapsto \left\lfloor \frac{\tau_S^{-1}(t,i) - \beta(0)}{r} \right\rfloor$.*

The term delaying *is related to the fact that the j^{th} boundaries computation has to be delayed until $\gamma(t,i) = j$.*

Example 2: Considering the preceding example, after simplifications, the delaying function is $\gamma(t,i) = \left\lfloor \frac{\tau_S^{-1}(t,i)}{100} \right\rfloor - 1$. If we consider a stream sourced at a rate of one tuple per second (therefore, $\tau_S^{-1}(t,i) = \lfloor t/1s \rfloor$). $\gamma(1024s, 0) = \lfloor \frac{1024}{100} \rfloor -$

[1] As a recall, the function $\mathbf{1}_A$ is the indicator function that gives 1 if the input is part of A, 0 else

$1 = 9$. The 10th window ($j = 9$) is then the last created window at this batch.

Given this, it is now possible to define an operator that generates a temporal relation from a stream. This temporal relation has a sequence of states. Transitions between states are triggered by batches arrivals and changes of γ.

Definition 10 (Window Sequence Operator): *Let S be a stream, (α, β, r) be a Window Sequence Description and γ be the delaying function associated to this description,*
The Window Sequence Operator is defined as :

- $\forall(t,i)$, *such that* $\gamma(t,i) \geq 0$,

If the description has temporal bounds:

$$S[\alpha,\beta,r](t,i) = \{s \in S/\ (\alpha(\gamma(t,i)),0) \leq \mathcal{B}_S(s) \leq (\beta(\gamma(t,i)),i)\}$$

If the description has positional bounds:

$$E_{t,i} = \{s \in S/\ \tau_S(\alpha(\gamma(t,i))) \leq \mathcal{B}_S(s) \leq \tau_S(\beta(\gamma(t,i)))\}$$

$$S[\alpha,\beta,r](t,i) = \{s \in E_{t,i}/ \\ |\{s' \in E_{t,i}/s(\varphi) < s'(\varphi)\}| \leq \beta(\gamma(t,i)) - \alpha(\gamma(t,i))\}$$

- $\forall(t,i)$, *such that* $\gamma(t,i) < 0$, $S[\alpha,\beta,r](t,i) = \emptyset$

This definition distinguishes the case of temporal windows from positional windows.
For temporal windows, it considers the batches ranging:

- **from** the first batch providing tuples falling in the window scope (noted $(\alpha(\gamma(t,i)),0)$), i.e. their timestamp is $\geq$ than the lower bound of the window
- **until** the last batch providing tuples in the window scope (noted $(\beta(\gamma(t,i)),i)$). This is the ith batch having the maximum timestamp inferior than the higher bound of the window.

Note that the relation $S[\alpha,\beta,r]$ may change at batch arrival even if time doesn't change. Using the batch identifier to define a window is mandatory since the partitioning introduced by batches is needed when building a new stream from window. Disregarding batch identifiers would lead to the loss of batch grouping. A treatment exclusively based on timestamps would not be sufficient.

For positional windows, the case where batches contain exactly one tuple each (full spread stream) is simpler than the general case where batches contain more tuples.

- For full spread streams, $E_{t,i} = S[\alpha,\beta,r](t,i)$.
- For the general case, $E_{t,i}$ is composed of all batches including tuples falling in the current window scope. The instantaneous relation $S[\alpha,\beta,r](t,i)$ is a subset of $E_{t,i}$. As example, consider "1-tuple" width windows. $E_{t,i}$ contains the last batch, but as it may include several tuples, only one has to be selected. The selection over the more-recent tuples is done with regard to the positional order φ.
- The γ function in the positional case is driven by τ_S^{-1} which provides the maximum tuple position in case of conflict. This choice is important as other choices would introduce tuple loosing on simultaneous events.

In the following, the window rate r will be noted with the unit n for positional rates or will indicate the temporal unit, s, ms, m.

Example 3: **Sliding Windows** Figure 1 shows a window sequence where windows slide of two seconds every two seconds ($r = 2s$) with a constant width of 3 seconds. $t_0 = 0$ for simplicity. The first window boundaries are $\alpha(0) = 0$

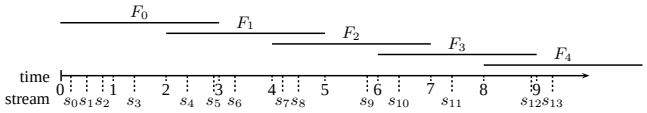

Figure 1: Window Sequence RANGE 3s SLIDE 2s

and $\beta(0) = 3$. The slide of boundaries is 2s.

$$\forall i \in \mathbb{N},\ \begin{cases} \alpha(i) & = i * 2s + t_0 \\ \beta(i) & = i * 2s + 3s + t_0 \end{cases}$$

The temporal relation generated from the stream S can be noted: $S[2is, 2is + 3s, 2s]$.

Given a timestamp $t = 5.5$ it is easy to compute $S[\alpha,\beta,r](5.5)$. The windows that can be computed at this time is the one numbered $\gamma(5.5) = \left\lfloor \frac{5,5-\beta(0)}{r} \right\rfloor = 1$. So $S[\alpha,\beta,r](5.5) = F_1 = \{s_4, s_5, s_6, s_7, s_8\}$

4.2 Partitioned and aperiodic windows

Definition 11 (Partitioned window): *A sequence of partitioned windows is built as the union of windows sequences on a stream partitioned by a set of attributes $a_1, ..., a_k$:*

$$S[a_1...a_k/\alpha,\beta,r] = \bigcup_{i \in Dom(a_1,...,a_k)} (\sigma_{(a_1,...,a_k)=i}S)[\alpha,\beta,r]$$

This operator is useful to create the same sequence of windows on sub-streams of a main stream.

Example 4: Given a stream S of measurements with the schema $(\mathrm{id}, \mathrm{loc}, m, t)$ in which *id* as a source identifier, *loc* as its location, m as a measurement and t as the timestamp. The last value sent by each location is given by $S[\mathrm{loc}/i, i, 1n]$ whereas the last value sent by each source is given by $S[id/i, i, 1n]$.

The considered window sequence descriptions use a rate r which will decide the exact form of γ. The fixed rate leads to a periodic windows creation. In the following, we'll introduce the possibility of using variable rates. This will allow the definition of aperiodic window sequences.

Definition 12 (Generic WSD): *Let $\mathcal{D}$ be either $\mathcal{T}$ or $\mathbb{N}$. A Generic Window Sequence Description is a triplet α, β, γ such that:*

- *α and β are two functions from $\mathcal{T} \times \mathbb{N} \to \mathcal{D} = \mathcal{T}$ representing the bounds.*
- *γ is a step function from $\mathcal{T} \times \mathbb{N} \to \mathcal{T} \times \mathbb{N}$ representing the delay between boundaries evolutions.*

These functions must verify the following properties to be valid (stated here for $\mathcal{D} = \mathcal{T}$):

$$\forall(t,i) \in \mathcal{T}\times\mathbb{N},\ \begin{cases} \alpha(t,i) \leq \beta(t,i) & \text{the beginning is before the end} \\ \alpha(t,i) \geq t_0 & \text{the beginning exists} \\ \beta(\gamma(t,i)) \leq t & \text{the end is accessible} \end{cases}$$

The conditions for $\mathcal{D} = \mathbb{N}$ are the following:

$$\forall(t,i) \in \mathcal{T}\times\mathbb{N},\ \begin{cases} \alpha(t,i) \leq \beta(t,i) & \\ \alpha(t,i) \geq 0 & \text{the beginning exists} \\ \tau_S(\beta(\gamma(t,i))) \leq (t,i) & \text{the end is accessible} \end{cases}$$

Example 5: Cases that need such definitions are not common. Let us consider a stream of traffic in network monitoring in a company. A close monitoring is required during working hours whereas overnights it can be relaxed. Such situation requires two different window creation rates (r_1 and r_2 hereafter) that reflect the frequency to be used during, and outside, working hours.

$$\gamma(t,i) = \begin{cases} \left(\left\lfloor \frac{t-\beta(0)}{r_1} \right\rfloor r_1 + \beta(0), 0\right) & \text{if } (t-t_d)\%24h \in [7,20[\\ \left(\left\lfloor \frac{t-\beta(0)}{r_2} \right\rfloor r_2 + \beta(0), 0\right) & \text{else} \end{cases}$$

γ plays an important role in the evaluation patterns of the window sequence operator. This is particularly concerned with the specification and the implementation of the process to determine when new boundary values are reached.

It is worth noting that another approach to express such generic window sequences could be based on the use of functions to calculate window boundaries without an explicit identification of γ. Nevertheless, that would be less explicit and the evaluation process would be more complicated.

5. PROPERTIES ON WINDOWS

This section discusses and presents the windows properties. Section 5.1 focuses on the associativity of the window operators and other operators. Section 5.2 presents a result on time transposition for window creation.

5.1 Associativity

The associativity is a core property that helps optimizers to push selections and projections down in the query tree. In this section, we confirm this property for some cases but we also show cases where it doesn't hold[2].

The first and easiest property is the associativity between the window sequence operator and projection or renaming operator.

Property 1 (Associativity with projection and renaming): *Let S be a stream and α, β, r be a window sequence description (WSD), then, the following properties state:*

$$\Pi_{a_1,\ldots,a_k} S[\alpha,\beta,r] = (\Pi_{a_1,\ldots,a_k} S)[\alpha,\beta,r]$$

$$\rho_{b/a} S[\alpha,\beta,r] = (\rho_{b/a} S)[\alpha,\beta,r]$$

These results hold because the projection and the renaming operators do not affect any of the elements (i.e., timestamps or tuple positions) used for window creation even with the use of τ_S. It is not always the case for the selection operator. Associativity for selection holds for full temporal windows but only for some specific cases of positional windows.

Property 2 (Associativity with selection for temporal windows): *Let S be a stream, α, β, r be a full temporal WSD and c be a selection condition, then, the following property states:*

$$\sigma_c S[\alpha,\beta,r] = (\sigma_c S)[\alpha,\beta,r]$$

This associativity holds because the selection eliminates tuples but does not change the criteria on time used for window creation. For positional windows, associativity is true only for a particular subset of cases. Let's consider an example where it doesn't hold.

Example 6: Let S be a stream of temperature sensor with schema $(temp, t)$. The following query

1. $\sigma_{temp>0}(S[i, i+9, 1n])$ means *the set of positive measurements from the last ten measurements each time a measurement is received*
2. $(\sigma_{temp>0}S)[i, i+9, 1n]$ means *the set of the last ten positive measurements each time a positive measurement is received.*

The difference between the preceding two queries can be seen by comparing the width of the resulting windows. In the second case, we are assured to have a window of 10 tuples with positive temperature whereas in the second one, the resulting windows may not contain 10 items. This is because the selection is made after the restriction to 10 tuples (with any temperature.)

The problem resides in the fact that $\tau_{\sigma_c S}$ and τ_S are not equal. This implies subtile, but meaningful differences in the semantics. So associativity is globally false for any positional windows. Yet, in the particular case of the classical landmark window sequence, the property is true.

Property 3 (Associativity with selection for unbounded accumulative windows): *The associativity of selection and positional windows holds for unbounded accumulative window sequence $[\infty] = [0, i, 1n]$.*

Property 3 relies on the other equivalent definition of the unbounded window sequence that is: $S[\infty](t,i) = \{s \in S/(t_0,0) \leq \mathcal{B}_S(s) \leq (t,i)\}$. This definition does not need the τ_S function.

The associativity between joins and window operators will not be discussed in this paper because of space limitation.

5.2 Time Transposing

Operator sharing [8, 13, 18] is an optimization technique known to be effective in data stream querying where the execution of a query may be infinite. For achieving an efficient sharing, the largest query equivalence is required. The exact match of two queries requires the equivalence of the query expression and their starting timestamps. The use of windows in the queries introduces different cases.

This section presents a contribution to compare two uniform[3] window descriptions namely, $[\alpha,\beta,r]$ and $[\alpha',\beta',r]$ with two different starting timestamp t_0 and t_1. The objective is to facilitate the sharing of window creation operators even if the two windows are not identical. Let:

- $Q_1 = (S[\alpha,\beta,r], t_0)$ and $Q_2 = (S[\alpha',\beta',r], t_1)$
- D is the implicit slide between two window sequence descriptions
  - For full temporal description: $D = 0$
  - For full positional description: $D = \tau^{-1}_{(S,t_0)}(t_1)$
- B_t is the minimal value of a lower bound of a description starting at t
  - If using temporal bounds: $B_t = t$
  - If using positional bounds: $B_t = 0$
- K is the synchronization factor: $K = \frac{\beta'(0)-\beta(0)+D}{r}$ It states the number of windows that were evaluated for Q_1 when the first window is evaluated for Q_2.

Property 4 (Linear Window Transposing): *Let S be*

[2] [8, 16] present results on this topic.

[3] e.g. bounds have the same type as the rate

a stream, if the description have the following form:

$$\alpha(i) = \max(ai + b, B_{t_0}) \quad \alpha'(i) = \max(ai + b', B_{t_1})$$
$$\beta(i) = ci + d \qquad\qquad \beta'(i) = ci + d'$$

If, the following properties state:

$$\begin{aligned} K &= \frac{d' - d + D}{r} \in \mathbb{N} \\ b' &= b + aK - D \text{ if } a \neq 0 \\ \max(B_{t_1}, b') &= \max(B_{t_1}, b - D) \text{ if } a = 0 \\ c &= r \text{ if } K \neq 0 \end{aligned}$$

Then, the following equivalence states:

$$(\sigma_{t \geq t_1} S[\alpha, \beta, r], t_0) = (S[\alpha', \beta', r], t_1)$$

Having K as an integer is rather intuitive as it describes the synchronization between the queries. In this case, it is possible to prove that

$$\gamma'(t, i) = \gamma(t, i) - K.$$

Considering this aspect, by replacing γ with γ', and by limiting the lower bound to $t \geq t_1$, the conditions over b' can be found.

The property about c states that the upper bound must follow the evaluation. This is necessary because the upper bound of the first window is always the moment of the first evaluation.

Example 7: Let's consider queries:

$$Q_1 = (S[2is + 3s, 5is + 3s, 5s], t_0)$$
$$Q_2 = (S[2is + 2s, 5is + 8s, 5s], t_1)$$

The objective is to reuse the results of Q1 for Q2. Here:

$$K = \frac{(8s + t_1) - (3s + t_0)}{5s} = 1 + \frac{t_1 - t_0}{5s}.$$

Then, we must synchronize them by having $t_1 = t_0 + n * 5s$.

As $K \neq 0$, c has to be equal to the rate. As $a \neq 0$, b' must be equal to $3s + t_0 + 2s * K = 5s + t_0 + n * 2s$. Therefore, we have, $b' - t_0 = 5s - n * 3s$. The problem is then reduced to one single equation to verify if the two windows can be coherent. For our example, $2 = 5 - 3n \Rightarrow n = 1$ is a valid matching. The figure 2 represents the behavior of this transposing.

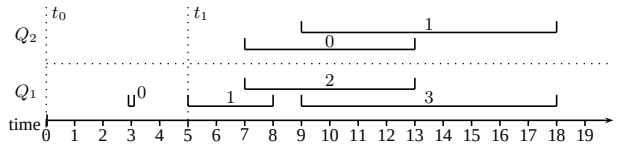

Figure 2: Transposition effects on window

Property 4 shows a result for windows with linear descriptions which are commonly used windows. This result can be integrated in stream query evaluators as the proposed equalities are easy to verify at runtime.

As another corollary, the following property points out the case of two queries using window sequences following the same pattern but having an initial offset.

Corollary 2 (Linear Natural Transposing): *Let S be a stream, if the descriptions have the following form:*

$$\alpha(i) - B_{t_0} = \alpha'(i) - B_{t_1} = \max(ai + b, 0)$$
$$\beta(i) - B_{t_0} = \beta'(i) - B_{t_1} = ci + d$$

If $c = r \wedge (a = 0 \wedge b \leq 0 \vee a = r)$ and

$$\begin{aligned} t_1 - t_0 &\in r\mathbb{N} \quad \textit{in temporal} \\ \tau^{-1}_{(S,t_0)}(t_1) &\in r\mathbb{N} \quad \textit{in positional} \end{aligned}$$

Then, the following equivalence states:

$$(\sigma_{t \geq t_1} S[\alpha, \beta, r], t_0) = (S[\alpha', \beta', r], t_1)$$

The conditions on the window synchronization are clearer as they are reduced to a condition over t_1 relatively to t_0. This result states that usual window sequences are naturally *transposable*.

Any sliding window sequence verifies this last property. This kind of window is similar to $[ir, ir + w, r]$ which verify the criterion. Therefore, a description $[5is, 5is + 2s, 5s]$ can be transposed *as it is* by an interval of any multiple of the rate r. It is interesting to note that it also works for landmark window sequences (case $a = 0$).

To our knowledge, a generic equivalence of windowing operators over time were never investigated this far. Further details on the transposability theory, as well as demonstrations of this section, are available on the Astral's wiki [2].

6. VALIDATION & PROTOTYPE

Thus far, we have seen how we formalize window operations on streams and some implications in practice. This section focuses on the validation of this theory. Section 6.1 discusses expressiveness w.r.t related work (introduced in section 2) whereas Section 6.2 introduces our prototype and discusses implementation issues.

6.1 Revisiting window definitions

The initial objective of this work was to clarify the semantics of window operator. In this section, we'll see how most popular window definitions proposed by related work can be formalized with our proposal. This allows to confirm that the expressive power of our algebra is enough for that. Windows that can be expressed by other systems can also be expressed with ours. This fact could be used later, to rely on such a unique formal description for mediation purposes.

Firstly, we look into the classic RANGE/SLIDE/ROW windows.

6.1.1 *RANGE x SLIDE y*

In the case of the sliding windows, there are two main window descriptions that can be made. The exact definition of this description as found in many works and as specified in [14].

$$\begin{aligned} r &= y \\ \beta(i) &= yi + t_0 \\ \alpha(i) &= \max(yi - x, 0) + t_0 \end{aligned}$$

As specified, the first evaluation is in $\beta(0) = t_0$ here which will contains only tuples that have a timestamp equals to t_0. Then, a first phase will contain a range under x. As soon as $i \geq x/y$, then the range of the timestamp in the window will be x.

The function $\alpha(i) = yi - x + t_0$ is not legal in our context. The second condition of a WSD requires $\alpha(i) \geq t_0$, which is not true here (for $i = 0$). Inserting the max function makes the description legal and describes the semantic of the first phases, which is not explicit in the textual description.

The following description is also valid, but it does not have an initial phase. Here, the relation is empty until $\beta(0) = t_0 + x$ to ensure that the window covers always a range of x.

$$\begin{aligned} \beta(i) &= yi + x + t_0 \\ \alpha(i) &= yi + t_0 \end{aligned}$$

In order to show the differences between the two possible models, the figure 3 represents the differences of evaluation for a sliding window as $y = 2$ and $x = 4$. In the first case (max), there are two other evaluations that are for $i = 0$ and $i = 1$. After that part, as demonstrated previously, the two models are identical.

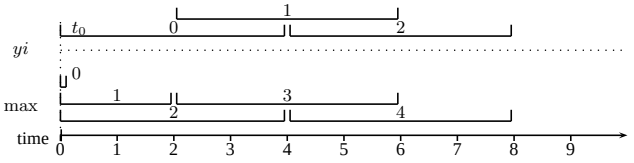

Figure 3: Difference between with and without max **in RANGE** 4 **SLIDE** 2

It has been noticed that there may be a lag of treatment (or delay) involved. This has already been formalized [16] and can be modelled here just by slightly modify the original γ function in an other one, γ', to slide all the evaluation steps. For instance, in a temporal description, a delaying function similar to $\gamma'(t,i) = \gamma(t-\delta,i) = \left\lfloor \frac{t-\beta(0)-\delta}{r} \right\rfloor$ expose that the evaluation is shifted by δ. In related works, this case is not easy to manipulate and is created from the beginning, here, we just need to go into a simple generic WSD.

6.1.2 RANGE x

In the usual definitions, RANGE x alone is possible and is similar to RANGE x SLIDE 1. In Astral, the time is not discreet and such definition is not making sense. A workaround would be to define the chronon of a system ε which corresponds to the minimal difference of timestamp that can happen, then RANGE x = RANGE x SLIDE ε. A better way of doing it is exploiting an extended WSD that bases its evaluation step whenever a tuple comes into a window, and whenever a tuple quits a window. But, a knowledge of α^{-1} and β^{-1} is necessary.

This last case makes a lot of sense in practice as it is possible to verify each timestamp (hence, each milliseconds) if the window verify the constraints, but it is a heavy way to do it. It is better to schedule when there will be changes.

The ROW descriptions are very similar to the ones presented here in the positional domain. Therefore, we will not provide further details on it.

6.1.3 Linear bounds description

In SStreamWare [12], windows are defined by a 5-uplet (start, end, rate, start_adv, end_adv)

- start (and end) describes the lower (and the upper) bound of the first window produced by the sequence.
- rate is the frequency of evaluation
- start_adv (and end_adv) denotes the sliding quantity of the lower (and the upper) at each evaluation.

In Astral, it is fairly easy to represent such window sequence:

$$\begin{aligned} r &= \text{rate} \\ \beta(i) &= \text{end\_adv} * i + \text{end} \\ \alpha(i) &= \text{start\_adv} * i + \text{start} \end{aligned}$$

6.1.4 Procedural descriptions

In the first versions of TelegraphCQ [10], window descriptions were made by a procedural for-loop on a temporal point of view like:

```
for (t = init ; continue(t) ; t = evolution(t) )
    WindowIs(S, begin(t), end(t));
```

This can be formalized by a generic WSD. If we consider the following list, u_0 =init, u_n =evolution(u_{n-1}) if continue(u_{n-1}) is true, $u_n = u_{n-1}$ else, then

$$\begin{aligned} \gamma(t) &= \sum_{i=0}^{+\infty} u_i \mathbf{1}_{[u_i, u_{i+1}[}(t) \\ \beta(t) &= \text{end(t)} \\ \alpha(t) &= \text{begin(t)} \end{aligned}$$

The γ function is defined by a list of points. This way of doing is usual in the definition of step functions. It can be noted that for window sequences with a rate and a correct initial position, a classic WSD is sufficient.

6.1.5 Cross-domain descriptions

In some systems [15] that are based on gathering data from sensors, it is common to retrieve data by bucket. Therefore, in a monitoring console, we can see frequent window sequences described by: "*Get the last n tuple each m seconds*". It is not necessary to go into a generic WSD, a classical one is sufficient as the evaluation is periodic. Here is the definition that corresponds to the last description:

$$\begin{aligned} r &= m \\ \beta(i) &= \tau_S^{-1}(mi + t_0) \\ \alpha(i) &= \max(\tau_S^{-1}(mi + t_0) - n, 0) \end{aligned}$$

The rate is temporal, and the bounds are positional. The gaps between the two domains are τ_S and τ_S^{-1}. In this case, the expression $\tau_S^{-1}(mi+t_0)$ gives the position at given timestamp. The remarks for α made on the section 6.1.1 are also stated here.

6.2 About the implementation

As a proof of concept, this theory has been implemented in the Astral prototype[4]. Its implementation has considered streams with associated functions as what have been presented here. This includes evaluation of the window operator exactly as shown in section 4. As for now, only classical WSD is implemented. The rest is on-going work.

For the implementation, we consider that α and β are rising (non strictly). This assertion is common and all the presented examples here have verified it. This point is important as it allows to destroy a tuple that goes out of the buffer and facilitates memory management.

Our prototype has a basic scheduler that understands when we need to evaluate the operator. Such question is hard to determine as we do not have assumption of the future. The source stream only regulates the time with its

[4]http://astral.ligforge.imag.fr

timestamp. Hence, the problem of network latency is not troublesome as we only consider the time indicated by the timestamp attribute. Nevertheless, it is necessary to wait for a tuple greater than the next evaluation to be sur to proceed without tuple loss. As specified in many sensor network projects and large-scale distributed systems, heartbeat messages would solve the problem of unbounded waiting.

In the scheduling process, another problem arises for positional windows that is the knowledge of τ_S. As we do not know when tuples will arrive, we must listen to every tuple coming event. An optimization approach would be to get an approximation of it based on statistics.

The prototype is coded in *Java* using *OSGi/iPOJO*[11] technologies over the *Apache Felix* framework. From an architectural point of view, each operator, and each stream (or relation) is a component. Scheduler, query runtime and builders are services that form the Astral core, which can execute the whole lifecycle of a query. The project's core requires more than 7 kSLOC (without commentaries), whilst the window logic takes about 700 SLOC.

7. CONCLUSION AND FUTURE WORK

In this paper, we have exposed the complexity of window semantics that was previously less considered. Our approach is based on data stream management algebra (named Astral), which contains a generic window operator. It allows expression of the existing window definitions and offers a clear semantic in conformity to the standard propositions. This is useful for mediation systems involving heterogeneous data stream.

This paper has also showed significant and non-trivial properties on windows creation. These properties are the associativity of window creation with some unary operators and an original first step to the transposability of queries on time. Such properties are particularly interesting for optimization purposes. Associativity properties are able to help in reducing data flows. Likewise, the transposition of queries on time addresses the sharing operators issue across concurrent queries. Our proposed approach allows reusing of windows created for queries that may not be identical. To the best of our knowledge no existing works on this are currently available.

A prototype of Astral is actively operational. It offers a subset of the presented operators and its implementation will be continued. Our future work involves further comprehensive study on query equivalences for streams including update patterns [17] and significant properties on the generic WSD. We are also interested in the static and dynamic algebraic optimization.

Acknowledgments: This research has been made under funding of the Transcap and ALAP projects.

8. REFERENCES

[1] Aleri Streaming Platform. `http://www.aleri.com/products/aleri-cep/aleri-streaming-platform`.

[2] AStrAL: Advanced Stream ALgebra, Wiki. `http://loic.doesntexist.com/mediawiki`.

[3] Coral8 Engine. `http://www.aleri.com/products/aleri-cep/coral8-engine`.

[4] SQLstream. `http://www.sqlstream.com`.

[5] StreamSQL online documentation. `http://streambase.com/developers/docs/latest/streamsql/index.html`.

[6] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB '03: Proceedings of the 29th international conference on Very large data bases*, 12(2), Aug 2003.

[7] A. Arasu. Continuous queries over data streams. *PhD Thesis*, 2006.

[8] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, and M. Datar. Stream: The stanford data stream management system. *Data Stream Management: Processing High-Speed Data Streams*, Jan 2004.

[9] D. Carney, U. Çetintemel, M. Cherniack, and C. Convey. Monitoring streams: a new class of data management applications. *VLDB '02: Proceedings of the 29th international conference on Very large data bases*, Jan 2002.

[10] S. Chandrasekaran, O. Cooper, and A. Deshpande. Telegraphcq: Continuous dataflow processing for an uncertain world. *CIDR '03: Proceedings of the First Biennial Conference on Innovative Data Systems Research*, Jan 2003.

[11] C. Escoffier, R. S. Hall, and P. Lalanda. ipojo: an extensible service-oriented component framework. *Services Computing, IEEE International Conference on*, 0:474–481, 2007.

[12] L. Gürgen. Gestion à grande échelle de données de capteurs hétérogènes. *PhD Thesis*, Jan 2007.

[13] M. Hong, M. Riedewald, C. Koch, J. Gehrke, and A. Demers. Rule-based multi-query optimization. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 120–131, New York, NY, USA, 2009. ACM.

[14] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik. Towards a streaming sql standard. *VLDB '08: Proceedings of the 34th international conference on Very Large Data bases*, 1(2):1379–1390, 2008.

[15] R. Jurdak, A. Nafaa, and A. Barbirato. Large scale environmental monitoring through integration of sensor and mesh networks. *MDPI Sensors: Molecular Diversity Preservation International Sensors*, 2008.

[16] K. Patroumpas and T. Sellis. Window specification over data streams. *ICSNW'06: Proceedings of the International Conference on Semantics of a Networked World*, 4254:445–464, 2006.

[17] K. Patroumpas and T. Sellis. Window update patterns in stream operators. *ADBIS'09: Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems*, 5739:118–132, 2009.

[18] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[19] A. Witkowski, S. Bellamkonda, H.-G. Li, and V. Liang. Continuous queries in oracle. *VLDB '07: Proceedings of the 33rd international conference on Very Large Data bases*, Jan 2007.