

パターンウィンドウに基づく シーケンスデータ処理と関係データ処理の統合

川島英之^{†1,†2} 北川博之^{†1,†2} Li Xin^{†1}

本論文はストリーム処理環境におけるパターンウィンドウなる新しいウィンドウ演算子を提案する。この演算子は時間順序を有するシーケンスから複合イベントを検出するために使われる。パターンウィンドウを関係データ処理と統合するため、演算木における演算子の状態を動的に変更する機構を導入する。

Pattern Window based Integration of Sequence Data Processing and Relational Data Processing

HIDEYUKI KAWASHIMA,^{†1,†2} HIROYUKI KITAGAWA^{†1,†2}
and XIN LI^{†1}

This paper proposes a novel window operator, pattern-based windows, for data stream processing. By using pattern-based windows, user-specified complex events could be extracted from data stream under the framework of Continuous Query Language (CQL), which integrates the current data stream processing research and event processing technologies. In the paper, pattern-based window is formally defined and the query language of pattern-based windows is designed. A simple pattern matching algorithm is introduced to show logical expression of the query language. We also give serial examples to show the usage of the query language by using pattern-based windows.

^{†1} 筑波大学大学院システム情報工学研究科

Graduate School of Systems and Information Engineering, University of Tsukuba

^{†2} 筑波大学計算科学研究センター

Center for Computational Sciences, University of Tsukuba

1. Introduction

Network and sensor device technologies are developing rapidly; and a variety of sensor devices such as network cameras, wireless sensor nodes and RFID readers are widely used in our daily lives. A variety of sensor devices such as network cameras, wireless sensor devices and RFID readers are widely used to catch and process the data from numerous primitive physical events which happen continually everyday. How to process this kind of primitive event streams has been focused increasingly by database community.

One of the most important problems is the processing of a sequence of primitive events to analyze raw data from sensor networks. Although stream processing techniques are discussed in depth, event processing (sequence pattern matching) has not yet been well developed under the framework of data stream processing systems which support by tuple semantics.

Therefore, the purpose of this paper is to design a novel event stream processing engine which can support pattern matching over sequence, and integration of sequence and relation. In this paper, we propose a novel window operator, pattern-based windows, which support event detection under the framework of data stream processing and continuous queries. We show that we can well integrated the existing data stream processing and event stream processing technologies. All the general operators in data stream processing including selection, projection, join, union, and duplicate-elimination, are supported in this research.

The rest of this paper is organized as follows: Section 2 describes related work. Section 3 illustrates basic data model on stream processing. Section 4 presents our proposal, pattern-based windows with definitions and examples. Section 5 concludes this paper and indicates future work.

2. Related Work

In this section, we describe a brief explanation to related works.

Stream processing engines such as StreamSpinner¹⁾, TelegraphCQ⁴⁾, STREAM⁷⁾ and Borealis¹⁾ have been well developed over the last several years both in research and industrial communities. One stream processing semantics and language, CQL²⁾ has been introduced by STREAM project team. It shows a strict model of stream processing and defines three kinds of window operators in their data model. However, all data processing systems above are awkward to express event sequences obtained by pattern matching. It is because usual stream processing engines sup-

port relation in which tuples are independent while sequence has a dependency of order among tuples.

Event processing system such as Cayuga⁵⁾, SASE^{8),12)} and Lahar⁹⁾ are close in spirit to this research. Cayuga^{3),5)} is a prototype event stream processing system, which enables high-speed processing of large sets of queries expressed in Cayuga algebra. SASE^{8),12)} describes events in a formalism related to regular expressions and uses some variants of a NFA model. One recent paper⁸⁾ of SASE presents a rich declarative event language referred to as SASE+. SASE+ can be used to define a wide variety of Kleene closure patterns to extract a finite yet unbounded number of events with a particular property from the input stream. Such patterns always appear in event streams in RFID and sensor network applications. Paper¹⁰⁾ and Lahar⁹⁾ are event processing systems over probabilistic event streams. But the event processing systems do not deal with general data stream processing operators such as join and duplicate-eliminating. Especially, the lack of join dramatically loses expressive power.

In this paper, we integrate both the technologies to increase the usability of a stream processing system.

3. Preliminary

In this paper, we utilize a data stream model similar to the data model used by CQL²⁾.

CQL is the short form of Continuous Query Language. In²⁾, a concrete query language and the abstract semantics were proposed. Tuples have timestamps, and there is no order between tuples with the same timestamps. The data model consists of streams and updatable relations for time-driven continuous queries.

In CQL model of defining stream and updatable relations, the authors assume a discrete and ordered time domain. Thus, the notion of timestamp is proposed to extend the conventional stream. The attribute of timestamp is not included in schema of a stream. And one timestamp can correspond to one element in a stream. A finite but unbounded number of elements with a concrete given timestamp is always required by users. A relation R defines multiset of tuples at any timestamp which is including in time domain. This definition for relation is quite ingenious to add timestamp attribute to traditional relation model.

A tuple t is conforming to the schema $\text{Stream}(a_1, \dots, a_n)$. A stream S is a finite (but unbounded) multiset of elements $\langle s, \tau \rangle$, where s is a tuple belonging to the schema of S and τ is the

timestamp. A relation R defines a multiset of tuples at any time τ , denoted instant $R(\tau)$.

Based on SQL, CQL formalized streams and updatable relations through defining three black-box classes of operators: (1)stream-to-relation operators that produce a relation from a stream; (2)relation-to-relation operators that produce a relation from one or more other relations; and (3)relation-to-stream operators that produce a stream from a relation. Stream-to-stream operators can be composed by the preceding three operators.

Stream-to-relation operators: all stream-to-relation operators in CQL are based on sliding window concept over a data stream. By utilizing slide window, the operators intercept and capture a finite portion of the stream at any timestamp. In CQL, there are three kinds of windows: time-based windows, tuple-based windows, and partitioned windows.

A time-based sliding window is specified by a time interval T and outputs relation over time by a sliding window to capture the last portion. A tuple-base sliding window is specified by a length N of number of tuples and capture the newest N tuples. Partitioned windows are quite different to the prior two operators. This window partitions stream referencing subset e.g. A_1, \dots, A_k . (A is the name of attribute of the steam.)

Relation-to-relation operators: On relation-to-relation, CQL refers all the operators from SQL by adding time variable to them.

Relation-to-stream operators: There are three relation-to-stream operators in CQL: ISTREAM, DSTREAM and RSTREAM. ISTREAM (for "insert stream") is the most commonly used operator, which indicates a insert stream to insert tuples according to the time τ by the difference of the relation results between τ and $\tau - 1$. DSTREAM (for "delete stream"), on the contrary, searches the deleted tuples and insert them into a stream. RSTREAM (for "relation stream") output all tuples satisfied by filter condition until now to construct an updated stream.

4. Pattern-based Windows

We first show the definition of pattern-based windows and sequence functions, then give some examples to show the usage of pattern windows and how they can process with normal relational operators in CQL model.

4.1 Definition

A pattern-based window on a stream S takes a specification of a sequence pattern P. Intuitively, a pattern-based window logically selects tuples on a stream which satisfy the user-defined

pattern. The timestamp of the last tuple of the window is time τ . More formally, we can write the definition of the output relation R of pattern P as:

$$R(\tau) = \{ \langle s_1, \tau_1 \rangle, \langle s_2, \tau_2 \rangle, \dots, \langle s_n, \tau \rangle \}$$
$$\tau_1 < \tau_2 < \dots < \tau_{n-1} < \tau$$

s_1, s_2, \dots, s_n satisfy pattern P

Tuples s_1, s_2, \dots, s_n should be in time order and satisfy the sequence pattern. The relation only has values when the last tuple is at the current timestamp τ . Thus, in this case pattern-based windows are similar to a special window, Now window “S [Now]” in CQL. But Now window only outputs a relation with tuples in the current timestamp, while a pattern-based window outputs a relation with tuples in different timestamps.

4.2 Syntax

We define the syntax of pattern-based windows as follows. The `stream` means a stream, and the following `[. . .]` means the pattern-based window.

```
stream[ PATTERN BY  
  ( START-CONDITION ;  
    TERMINATION-CONDITION ;  
    ITERATION-CONDITION ;  
    FILTER-CONDITION ) ]
```

Inspiration by FOLD Clause in Cayuga Event Language⁹⁾, four conditions as four parameters in a pattern-based window are defined. Here, the meaning of the conditions is introduced followed by a simple algorithm to indicate the logical processing of pattern matching using the four conditions.

The explanation of the parameters are as follows. (1) The first parameter, start-condition, is the condition to start a pattern-based window. Therefore, the first tuple in a pattern-based window should satisfy start-condition. (2) Termination-condition on the other side denotes the condition to end a pattern-based window. When a tuple in the window meets termination-condition, the pattern-based window is closed and should be output to the next operator. (3) Iteration-condition is the condition of matching iteration of a window and preserving the correctness of a pattern-based window. If iteration-condition is not met, the iteration is stopped and the window is deleted. We can set iteration-condition as FALSE to find two directly consecutive tuples on a data stream. (4) Filter-condition is the condition to select the tuples in the window. If we set the filter-condition as TRUE, all the tuples after the first tuple are added to the buffer of a pattern-based window. If we

set it as FALSE on the contrary, only the first tuple and the last tuple are contained in the window.

4.3 Sequence Functions

Sequence functions are similar to aggregation functions for a relation. But the general aggregation functions do not consider the time order for the tuples in a relation. Users, for example, usually want to find the first occurred tuple in a relation. Thus aggregation functions coping with timestamps should be introduced. In this paper, we propose five sequence functions. They are FIRST(), LAST(), PREV(), ITER() and NUM().

In pattern-based windows, attribute names are directly used to denote the attribute values of tuple in the current iteration. FIRST() denotes the attributes in the first tuple (the tuple with the minimum timestamp) in the relation and the matched sequence. LAST(), similarly, denotes the last tuple (the tuple with the maximum timestamp) in a relation. FIRST() can be used in the SELECT clause and in all conditions in pattern-based windows and LAST() can be only used in the SELECT clause. In SELECT clause, they work as the normal aggregation functions and the parameter of the functions is a list of attribute names. When the functions are used in pattern-based windows, the parameter has to be only one attribute name. The function return the attribute value in a matching buffer for a open window. PREV() can only be used in pattern-based windows, which represents the attribute in the previous tuple in a window. Please note that PREV() cannot be used in the start-condition because the first tuple does not have a previous tuple. We also define function ITER() as the current number of iteration in an open window and as the final number of iteration times of a window in SELECT clause. Similarly, Function NUM() is defined to return the number of tuples in a window.

4.4 Examples

We introduce the query language through several examples. We first assume a location stream for different persons from sensor networks. The schema of the stream is At(Person, Location), which simply describes someone is at some place at a certain time.

People usually want to find the next event after something happened. So, we can assume the simplest pattern with two tuples. The first tuple indicates that the some event happens at first, then the query find the next interesting tuple by user-defined pattern. We show a relatively simple example as follows.

Example 1. Suppose we want to find the next place where Tom go after in the laboratory. We can formulate this query in pattern-based window as follows.

```
SELECT ISTREAM(LAST(Location))
FROM At[PATTERN BY
  (Person = 'Tom' AND
   Location = 'Lab';
   Person = 'Tom';
   TRUE;
   FALSE)]
WHERE Location <> 'Lab'
```

Iteration-condition is set as TRUE and filter-condition is set as FALSE. So, tuples in iteration are not added to the window. This pattern help to find a window with only two consecutive tuples. The first state is that Tom is in the laboratory. The second tuple represents that Tom is at some place. When we find such a pattern-based window, we filter the tuple in the laboratory by selection in the WHERE clause, because we want to find some other places besides the laboratory. Finally, ISTREAM operator is used to translate the result into a stream.

Example 2. Suppose we want to find next place where someone is after in the laboratory. We can formulate this query in pattern-based window as follows.

```
SELECT ISTREAM(LAST(Location))
FROM At[PATTERN BY
  (Location = 'Lab';
   Person = FIRST(Person);
   TRUE;
   FALSE)]
WHERE Location <> 'Lab'
```

In this example, we change Tom to someone in the query. Therefore, a constraint condition is “Person = FIRST(Person)”. The attribute Person in the current iteration can be directly written as “Person”. First tuple is quoted in the termination-condition by using FIRST(). We can force the same person in a pattern-based window by checking equality in termination-condition and filter-condition.

Example 3. Suppose we want to find someone first in the laboratory and the directly next location is Room 101. We can formulate this query as follows.

```
SELECT ISTREAM(LAST(Location))
```

```
FROM At[PATTERN BY
  ( Location = 'Lab';
   Person = FIRST(Person) AND
   Location = 'Room 101';
   Person <> FIRST(Person);
   FALSE)]
WHERE Location <> 'Lab'
```

In this query, we use filter-condition to find the same person for iteration. But the iteration-condition is set as “Person \neq FIRST(Person)”, which means only direct next place “Room 101” after “Lab” for a certain person is allowed for the pattern. Otherwise the window will be dropped.

Example 4. Suppose we have another data stream indicating the states of rooms with a schema RoomState(Room, State). We can join pattern-based window results in Example 3 and Stream RoomState to find someone’s state after leaving the laboratory. We can formulate this query in pattern-based window as follows.

```
SELECT ISTREAM(Person, State)
FROM At[PATTERN BY
  (Location = 'Lab';
   Person = FIRST(Person);
   TRUE;
   FALSE)],
RoomState[Range 2min]
WHERE At.Location = RoomState.Room AND
At.Location <> 'Lab'
```

In this query, we use two kinds of window operators in the FROM clause. The first is a pattern-based window same to Example 2. The second is a time-based window. For the pattern-based window performs as Now window. We obtain a new relation from the join operator only when the current tuple meets the termination-condition of a pattern-based window. Please note that in our model we can rename the original timestamp of a data stream as a new attribute after window operators. The new timestamps are given by relation-to-stream operators by CQL model. This is important because the original timestamps contain the occurrence order of the tuples in a pattern-based window.

Example 5. Suppose we want to find that a person at first is in laboratory, finally returns home. The passes from laboratory to home are not considered (i.e. The person may pass some places and arrive at home indirectly or he go home from lab directly. We also want to calculate the duration time between the laboratory and the home. We can formulate the query in pattern-based window and sequence functions as follows.

```
SELECT ISTREAM(FIRST(Person) ,
              LAST(Timestamp)-FIRST(Timestamp))
FROM At[PATTERN BY
      (Location = 'Lab' ;
       Person = FIRST(Person) AND
       Location = 'Home' ;
       Person <> FIRST(Person) OR
       Location <> 'Lab' ;
       Person = FIRST(Person))]
```

This query can find a window for a certain person with the sequence from “Lab” to “Home”. We can see here that FIRST(), LAST() work as aggregation functions. So the relation after SELECT clause contains only one tuple. ISTREAM then converts the relation into stream for output.

Example 6. Suppose we want to find that a person is in laboratory at first, and sometime later he is at home. Similar to Example 5, the person might go home directly or pass some other places and finally arrive home. The query user wants to know all the passes from the laboratory to the home. We can formulate the query in pattern-based window and sequence functions as follows.

```
RSTREAM(
  SELECT *
  FROM At[PATTERN BY
        (Location = 'Lab' ;
         Person = FIRST(Person) AND
         Location = 'Home' ;
         Person <> FIRST(Person) OR
         Location <> 'Lab' ;
         Person = FIRST(Person))]
  WHERE Location <> 'Lab' AND
```

```
Location <> 'Home' )
```

In this query, different from Example 5 using aggregation, our system also can output the whole relation by specifying RSTREAM if users need it.

4.5 Dynamic State Management

In a SPE, stateful operators such as join or aggregation manage its state. Usually the length of state is based on either time duration (time-based window) or number of tuples (tuple-based window). Pattern window has a different feature. Until a pattern is obtained in the window, a pattern window does not generate any tuple. This behavior is different from usual windows. Therefore state management should be reconsidered for pattern window.

Since an output of pattern window has a time length, its counter part should be in the same time region. Therefore as for join operator, target stream length is dynamically controlled by each output of pattern window. A special mechanism for the control should be proposed in future work.

4.6 Consideration

Recall the Example 4 in the previous section. It is showing an instance that how to find someone's state according to the room in which the person is at that time. In this case, there are two updating data streams. One of them is a stream of person's location noted as **AT**; the other one is the state of room noted as **RoomState**. First, we process the **AT** by our pattern-based window and obtain one concrete relation. Similarly, we obtain a relation from **RoomState** by using time-based window. Then, we do a join operation between them to realize the purpose of survey some person's state. And then, after processing by selection operator, the tuples only with attributes of Person and State are ready to output into the purpose stream. At last, ISTREAM operator accomplishes the final work by outputting each tuple to form the updated stream.

About the last examples, pattern-based window can output the a relation after aggregation processing. However, Cayuga cannot do such operation because they do not consider a relation as their output.

The most important difference from FOLD clause in Cayuga is that as a window operator, it should be a one input construct and the pattern matching operators in Cayuga are binary constructs. So the constructs of the two operators are different and we can do aggregation through pattern-based windows easily because the pattern-based window supports a whole relation output as a kind of window operator.

Compared with CQL²⁾, our model of pattern-based window can be considered an extension to the window operators of CQL. We can present one kind of snapshots of query plan which satisfies the user's intending query through our model.

5. Conclusions and Future Work

In this paper, we proposed a novel window operator, pattern-based window, to integrate the current relational data stream processing research and event processing technologies. The definition of pattern-based windows is given and the query language and algorithm of pattern matching process for pattern-based windows are designed. The usage of pattern-based windows was also showed through several concrete examples.

As future work, we want to implement the system with pattern-based windows and consider optimization of the pattern matching algorithm, since the performance is also very important for stream processing systems. Probabilistic data model should be considered in future work to exploit the uncertain nature of data streams from sensor devices. Lineage will be a powerful tool for probability computation and data traceability.

謝辞

本研究の一部は科学研究費補助金基盤研究 (#18200005, #20700078), 筑波大学 VBL 研究プロジェクトによる。

参 考 文 献

- 1) D.Abadi, Y.Ahmad, M.Balazinska, U.Cetintemel, M.Cherniack, J.-H. Hwang, W.Lindner, A.Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and Stan Zdonik. The design of the borealis stream processing engine. In *Proc. of the Conference on Innovative Data Systems Research*, pp. 277–289, 2005.
- 2) Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, Vol.15, No.2, pp. 121–142, 2006.
- 3) Lars Brenna, Alan Demers, Johannes Gehrke, Mingsheng Hong, Joel Ossher, Biswanath Panda, Mirek Riedewald, Mohit Thatte, and Walker White. Cayuga: a high-performance event processing engine. In *Proc. of the 2007 ACM SIGMOD International Conference on Management of Data*, pp. 1100–1102, 2007.
- 4) Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred

- Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the Conference on Innovative Data Systems Research*, pp. 269–280, 2003.
- 5) A.J. Demers, J.Gehrke, and etal M.Hong. Towards expressive publish/subscribe systems. In *Proc. of International Conference on Extending Database Technology*, pp. 627–644, 2006.
- 6) Alan Demers, Johannes Gehrke, and Biswanath P. Cayuga: A general purpose event monitoring system. In *In CIDR*, pp. 412–422, 2007.
- 7) TheSTREAM Group. Stream: The stanford stream data manager. *IEEE Data Engineering Bulletin*, Vol.26, No.1, 2003.
- 8) Daniel Gyllstrom, Jagrati Agrawal, Yanlei Diao, and Neil Immerman. On supporting kleene closure over event streams. In *Proc. of IEEE International Conference on Data Engineering*, 2008.
- 9) Christopher Ré, Julie Letchner, Magdalena Balazinska, and Dan Suciu. Event queries on correlated probabilistic streams. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 715–728, 2008.
- 10) Zhitao Shen, Hideyuki Kawashima, and Hiroyuki Kitagawa. Lineage-based probabilistic event stream processing. In *Proc. International Workshop on Sensor Network Technologies for Information Explosion Era*, 2008.
- 11) Yousuke Watanabe, Shinichi Yamada, Hiroyuki Kitagawa, and Toshiyuki Amagasa. Integrating a stream processing engine and databases for persistent streaming data management. In *Proc. of 18th International Conference on Database and Expert Systems Applications (DEXA2007)*, LNCS 4653, pp. 414–423, 2007.
- 12) E. Wu, Y.Diao, and S.Rizvi. High-performance complex event processing over streams. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pp. 407–418, 2006.