Acta Technica *62* (2017), No. 5A, 597–616

Flexible nfa window management over interval-based event pattern matching

Weiliang Tao[2,5], Wenhai Li[3], Yan Liu[4]

Abstract. In this research, we provide a formalization framework to integrate the streaming window into the NFA-based event pattern matching system. We identify all the events with their duration interval and provide the related strategy to assure their partial order. This feature both offers the systematic support under the complex sub-query application and guarantee the scalability for the distributed real-time environment. By providing the event insert operations, the window management can flexibly settle the two sliding constricts in all four semantic context circumstances. The window confines each output of the low-level event to a certain range, and hence provide the high-level pattern with both temporal range and event counting semantics. Experimental results on various window parameters demonstrated effectiveness of the proposed window sliding semantics and the efficiency of the optimized management techniques.

Key words. Complex event process, pattern matching, temporal data, database.

1. Introduction

Due to the tremendous potentiality applied to business intelligence on workflows, supply chain management for RFID, large-scale attack monitoring and financial analysis, continuous pattern matching over durative atomic events along with their high-level composite processing has aroused considerable attention in both academic and industrial worlds. Pattern matching process over the event streaming, especially in the interval semantic environment, has increasingly received concerns in several

[1]Acknowledgement - This work was sponsored by the National Science Foundation, grant 60903035 and 41001296, and Wuhan University Youth Foundation grant 6082012. The authors would like to thank Huang Jianzhong and Cheng Yuan for many valuable comments to this paper. The authors would like also thank to The High Performance Computing Center at Computer School of Wuhan University for the platform provision.

[2]Workshop 1 - School of Electronic information, Wuhan University, Wuhan, Hubei, 430072, China

[3]Workshop 2 - School of Computer, Wuhan University, Wuhan, Hubei, 430072, China

[4]Workshop 3 - High Voltage Department, China Electric Power Research Institute, Wuhan, Hubei, 430074, China

[5]Corresponding author: Weiliang Tao; e-mail: taowl2003@sina.com

pivotal yet controversial aspects such as temporal semantics definition, event strategy selection, consumption mode construction, accomplished method foundation. Upon that, some framework prototypes accompanied by a fair amount of implemental and optimization innovations emerged in the correlative application area.

Different from the data stream management system (DSMS), event pattern matching system focuses more on the expressivity, effectivity, flexibility and scalability. For depicting the real application scenario, the queries in event pattern must possess abundant flow semantics, particularly the business expression in the sequence scenes. Furthermore, faced by the massive real-time data stream, the event pattern matching pays close attention to the resource utilization and the processing efficiency for providing both high throughput and low latency. At the same time, implemental strategy must have the flexible and scalable ability for the increasingly emerging requirement. We notice here that, as one of the most prominent characteristics of the complex event pattern, the temporal restriction and event feedback settlement is critical for either event composite or embedded sub-query circumstance. For example, in continuous (most recently arisen) three days, pick the stock that starts at a volume larger than 1000, and then has been increasing monotonically in price for at least 30 minutes and does not end off its initial value in each remaining day. These two involved semantic contexts need both embedded pattern definition and count/ temporal sliding window supports, and cannot be properly handled with the existing event processing systems.

In this paper, for flexibly providing semantic context scalability, we provide a formalization framework to integrate the streaming window into the NFA-based event pattern matching system. We are not presenting a complete system implementation in this paper due to the matching optimization challenges, but the proposed formalization framework can exhibit several contributions including:

1. **Pure interval semantic support**. Different from the detection-based primitive event expression, we identify all the events with their duration interval and provide the related strategy to assure their partial order. This feature both offers the systematic support under the complex sub-query application and guarantee the scalability for the distributed real-time environment.

2. **Partial semantic context.** Instead of the instance selection and consumption mode employed in the existing propositions, the semantic context in this paper exhibits a series of interval oriented context definition with the partial relationship. Towards the two basic operators: sequence and Kleene closure, the compact context compresses the initiator by abandoning the existing initial events. There against, the extended context rejects the newcomer matching the initiator's pattern, while the complete and traceable contexts output the complex events by recording all the arisen initiators. Similar with the initiator, the detector is also handled differently in the four semantic contexts, and the corresponding outputs respectively hold a successively temporal inclusion relation within the sliding window backgrounds.

3. **Flexible window management.** For the event pattern matching, one of the two sliding windows can be founded on the instances lists to resolve the

temporal and event counting constricts respectively. By providing the event insert operations, the window management can flexibly settle the two sliding constricts in all four semantic context circumstances. Fundamentally, the window confines each output of the low-level event to a certain range, and hence provide the high-level pattern with both temporal range and event counting semantics.

The remainder of the paper is organized as follows. Experimental part gives the formalization framework with its basic definitions, presents the state window management along with its semantic context related maintenance algorithms. Results and Discussion part details the relative experimental analysis comparison followed by the related background work. Finally, Conclusions part states conclusions and our future work.

2. Experimental Formalization Framework

In the event pattern matching systems, an event was originally defined as an instantaneous occurrence and the last event's occurrence time was used as the time of the entire event expression. The early event specification languages, especially in the active DBMSs, employ this detection semantic for compositing the instantaneous events.

Without differentiating between the event occurrence and the event detection, detection semantic cannot post its output to the event stream, and then cannot use the composite events in the embedded patterns. More importantly, the practical atomic events generally take on a durative manner, and the interval based pattern matching can make the operators easier to handle the orders of the overlap events than the detection based one. For the sake of simplification, in this section we will propose the four interval semantic contexts along with their partial properties.

(1) Expression on interval

Each event instance $e_i^j \Big| 1 < j < \infty$ matching an event pattern $E_i | 1 < i < \infty$ with its duration time $[t_s(e_i^j),\ t_e(e_i^j)]$ can be a primitive event or a composite event, and the whole event information is denoted by (e_i^j). Without loss of generality, we give a temporal space T composed of the infinitely nonnegative timestamps, and assume the time interval $I(e) = [t_s(e),\ t_e(e)]$ of any event instance e takes the timestamps $t_s(e),\ t_e(e)$ as its start and end. The fundamental relationships are expressed with a series of definitions as follows.

Definition 1 (Partial order). A temporal partial order $\prec$ on T satisfies the conditions:

1. If $t_0,\ t_1,\ t_2 \in \mathrm{T}$ and $t_0 \prec t_1,\ t_1 \prec t_2$, then $t_0 \prec t_2$.
2. For any $t \in \mathrm{T}$, tt.

The partial order on timestamps is the basis of the other interval operators. Among all the thirteen temporal relationships, during/contains and after/before are the most used binary relationships in the interval event matching circumstances.

Definition 2 (During/contains). A during relationship DUR and its counterpart contains CTN between two intervals I^1 and I^2 can be defined as $DUR(I^1,\ I^2)$: $t_s^2 \prec t_s^1 \prec t_e^1 \prec t_e^2$ and $CTN(I^1,\ I^2)$: $t_s^1 \prec t_s^2 \prec t_e^2 \prec t_e^1$ separately.

The during/contains relationships as well as the other eleven ones constitute Allen's interval algebra, which is one of the best established formalisms for temporal reasoning. Omitting the related reasoning properties, we use these two relationships here for demonstrating the temporal relationships among the four proposed semantic contexts. Somewhat differently, the before/after in Allen's algebra are utilized to extract a successor operator in the event pattern context.

Definition 3 (Successor). An instance e^j matching pattern E is the successor of another event e^k if and only if the former's duration time I^j starts after the latter's end and no extra instance of E after e^k ends before e^j, i.e.

For a given suffix pattern, this definition can dynamically extract the unique successor instance of a prefix even if the two rival suffixes end at the same time. Fig.1 gives a sample of the successor determination among four overlap events.

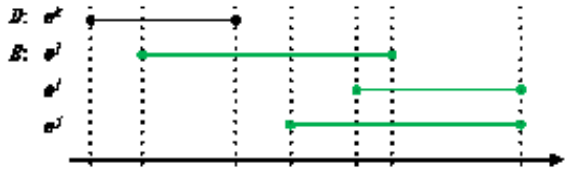

Fig. 1. Intervals of four overlap events

(2) Semantic context definition

Conventionally, event pattern matching system can be regarded as a combination of data streams and complex event processing systems. Nevertheless, the pattern matching could hardly be processed efficiently in a real-time stream manner.

Many researches omit this feature[1,2], while others compromise between the efficiency and the accuracy by tailoring the complete history semantic. It is worth noticed that selection strategy and consumption context are two of the most representative propositions. However, the former is essentially an alternative definition of sequence and Kleene closure founded on event filter, while the former can be seen as a variation of both tuple and temporal sliding window semantics in CQL[3]. Flexible window integration along with its sliding strategy is critical for the practicability of the pattern matching systems. Here we formally give the four interval based semantic contexts for sake of completeness, and will propose the related state window management strategies in the next chapter.

In terms of both the scope of output and the space utilization, we are dedicated to constructing a context hierarchy mainly target on the scalability of the most expensive operators, i.e. sequence and Kleene closure. In the sequel, the pattern of the initiator, detector and terminator is orderly denoted with E_i, E_d and E_t. More concretely, due to the uniqueness of the terminator in Kleene closure, the above two operators can be both regarded as a binary operator BOP with just two input streams. Briefly, we take the initiator and detector as the two restrictive patterns

on the two inputs respectively. We also propose the context definitions focused on the initiators followed with their restrictive demonstrations for the detectors of both sequence and Kleene closure.

Definition 4 (Compact). For a $BOP(E_i,\ E_d)$ on two inputs P, S, each output $(e_i^j,\ e_d^k)[o_s,\ o_e]$ must satisfy

Where $e_i^j \in [\![e_i^j]\!]_{\mathrm{P}}$ denotes the event e_i^j derived from stream P can match pattern E_i, and $(e_i^j,\ e_d^k)[o_s,\ o_e]$ takes e_i^j as the composite initiator. This definition picks the instance e_i^j of initial pattern E_i as the start of the output, if and only if e_i^j has the latest end time among the contiguous instances of E_i with no instances of E_d occurs between these contiguous initial instances. As shown in Fig.2, b^4 is the successor of both a^3 and a^4 for pattern operator $BOP(A,\ B)$, but only a^4 is selected as the start due its later end time relative to a^3.

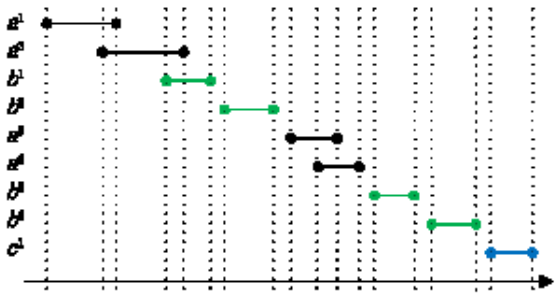

Fig. 2. Arrival instances of three primitive patterns

Definition 5 (Expended). For a $BOP(E_i,\ E_d)$ on two inputs P, S, each output $(e_i^j,\ e_d^k)[o_s,\ o_e]$ must satisfy:

$$G = \left(\frac{1}{a^2}\frac{\partial^2 w}{\partial \xi^2} + \frac{1}{b^2}\frac{\partial^2 w}{\partial \eta^2}\right)^2, \quad H = \frac{1}{a^2 b^2}\frac{\partial^2 w}{\partial \xi^2}\frac{\partial^2 w}{\partial \eta^2} - \left(\frac{1}{ab}\frac{\partial^2 w}{\partial \xi\, \partial \eta}\right)^2.$$

In contradiction to compact context, expended context chooses the first one of the contiguous initiators as the start of the output. For the same example above, expended context select a^3 instead of a^4.

It is obvious that the above two definitions share a same exclusive semantic, i.e. each detected event belongs to at most one output and is abandoned or consumed afterwards.

Definition 6 (Complete). For a $BOP(E_i,\ E_d)$ on two inputs P, S, each detector e_d^k starts an output for any of its initiators $e_i^j \in [\![e_i^j]\!]_{\mathrm{P}}$ supposing $e_d^k = SUC(e_i^j,\ E_d)$ held.

Complete context relaxes the above two with the non-abandon semantic, i.e. the later arrival initiator does not overwrite the existing initiator. The explicit semantic can be seen in Fig.2. When b^2 arrives, it only takes a^2 as the initiator for $BOP(A,\ B)$ even though a^1 has been selected as the initiator of b^1. Similarly, c^1 wholly takes b^1, b^2, b^3 and b^4 as its initiators for $BOP(B,\ C)$, but compact context select b^4 as

the only initiator.

Definition 7 (Traceable). For a $BOP(E_i,\ E_d)$ on two inputs P, S, any detector e_d^k starts an output for each of its initiators $e_i^j \in [\![e_i^j]\!]_{\mathrm{P}}$ along with their intermediations

$$\left[\left[e_i^j,\ e_d^k\right]\right]_{(E_i,\ E_d)} = \{\forall\ e_i^l \in [\![e_i^j]\!]_{\mathrm{P}} \Big| (e_d^k = SUC(e_i^j,\ E_d)) \wedge (e_d^k = SUC(e_i^l,\ E_d)) \wedge (t_e(e_i^j) \prec t_e(e_i^l))\}$$

Traceable context implies that any of the contiguous initiators with no detector followed can generate an output, and the intermediate initiators between the start initiator and the detector are treated as part of the output. As shown in Fig.2, considering the query "the series of B between A and C", all four outputs shown in row 4 and column 1 of Table 1 are generated in the traceable context. Slightly different from the omitting implications adopt by complete context, traceable records all the intermediate detectors for the one-to-one correspondence of the output in complete context.

Table 1. Output of the four contexts

	$[\![e_i^j]\!]_{\mathrm{P}}^I$	$[\![e_i^j]\!]_{\mathrm{P}}^I$
Compact	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$
Expended	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$
Complete	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$
Traceable	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$	$[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$ $[\![e_i^j]\!]_{\mathrm{P}}^I$

(3) Partial properties and event operators

Definitions 4-7 demonstrate the semantic construction of our research, and mainly express the determination of the initiators of a given pattern. This section will elaborate the relative partial properties and the two event operators processing in the above four contexts.

As mentioned above, among the contiguous initiators, the temporally rightmost and leftmost one is respectively selected as the pattern initiator in compact and expended contexts. Nevertheless, complete and traceable contexts generate an output

for each of the contiguous initiators. As for the detectors, when considering the sequence operator, all the four semantic contexts select the "first" successor for the contiguous initiators, i.e. the initiators $[\![e_i^j]\!]_{\mathrm{P}}^I$ and the detector e_d^k must satisfy:

$$\forall e_i^l \in [\![e_i^j]\!]_{\mathrm{P}}^I \Big| \, e_d^k = SUC(e_i^l,\ E_d)$$

Similarly, the Kleene closure operator in the above four contexts select the whole series of the detectors until the terminator $e_t^n | \ \forall e_d^m \in [\![e_d^k]\!]_{\mathrm{S}}^D \Big| \, SUC(e_d^m,\ E_t) = e_t^n$ happens, i.e.

$$\forall e_i^l \in [\![e_i^j]\!]_{\mathrm{P}}^I,\ e_d^m \in [\![e_d^k]\!]_{\mathrm{bS}}^D \Big| \, (e_d^m = SUC(e_i^l,\ E_d)) \\ \vee (e_d^m = SUC^*(e_i^l,\ [\![e_d^k]\!]_{\mathrm{S}}^D,\ E_d))$$

Where SUC^* is the transitive closure of the relationship SUC for depicting the transitively successive relationship among e_i^l e_d^k and e_t^n, and it can be expressed as follows.

$$e_d^m = SUC^*(e_i^l,\ [\![e_d^k]\!]_{\mathrm{S}}^D,\ E_d) \Leftrightarrow \exists e_d^1,\ e_d^2,\ ...,\ e_d^j \in [\![e_d^k]\!]_{\mathrm{S}}^D \Big| \\ (e_d^1 = SUC(e_i^l,\ E_d)) \wedge (e_d^2 = SUC(e_d^1,\ E_d)) ..... \\ \wedge (e_d^j = SUC(e_d^{j-1},\ E_d)) \wedge (e_d^m = SUC(e_d^j,\ E_d))$$

Obviously, under the four contexts, outputs of Fig.2 generated by the pattern sequence $SQ(A,\ B)$ and Kleene closure $KC(A,\ B,\ C)$ can be listed as follows.

Among the semantic contexts, compact and expended have the same quantity of outputs. The leftmost and the rightmost selection can both, however, result the temporal inclusion relationship between the corresponding outputs from the two contexts.

Property 8. Supposing for an operator $BOP(E_i,\ E_d)$ on any two inputs P and S, any detector $e_d^k \in [\![e_d^k]\!]_{\mathrm{S}}$ can generate an output $OS_{CPC}(e_i^j,\ e_d^k) \Big| \, e_i^j \in [\![e_i^j]\!]_{\mathrm{P}}$ in compact context or $OS_{EXP}(e_i^l,\ e_d^k) | \, e_i^l \in [\![e_i^l]\!]_{\mathrm{P}}$ in expended context, and $DUR(OS_{CPC})$ exists.

Two points must be noticed here. Firstly, as we mainly consider the selection strategy of the continuous initial intervals, the circumstance when $DUR(e_i^j,\ e_i^l)$ is assumed not happen in the input P. More precisely, $DUR(e_i^j,\ e_i^l)$ can result in a contrary temporal inclusion. Secondly, the relationship shown in property 8 is rather a "finished by" relationship between $OS_{CPC}(e_i^j,\ e_d^k)$ and $OS_{EXP}(e_i^l,\ e_d^k)$ due to the same detector and terminator of both contexts.

Similarly, as shown in definitions 4-7 and property 8, we give the following two properties in terms of the quantity inclusions without any demonstration.

Property 9. For any given event operator and input streams, the outputs from both compact and expended contexts are subsets of the output from complete context.

Property 10. For any given event operator and input streams, the outputs from complete and traceable context are the same except that the event set generated from the former is one-by-one the subset of the one from the later.

3. Window Management

In general, the development of the previous window semantics of the event processing is oriented towards the continuous query language, and thus can be divided into three abstract levels in terms of the granularity of the aggregate information extracted from the temporal raw tuples. Firstly, the statistical stream window extracts the aggregations over a sliding time period especially for several flat event operators[4]. Secondly, requiring the event detection as an antecedent condition, the point event window usually provides a temporal range for restricting the event matching with the given patterns. Finally, eminently different from the above two methods by merely providing the semantic framework of the event processing, the formalization window language proposes the definition construction of event pattern confined to both the interval and tuple along with their related sliding strategies[5].

(1) State-based window Semantic

To express the composite event semantic with intervals sliding, just as the example illustrated in the introduction, we integrate the sliding window strategy into the NFA state evaluating. We give the window semantic based on the NFA states and their corresponding instances, and later proposes the related evaluation issues especially for two sliding window strategies.

Similar with Cayuga, our language framework realizes the mapping of the event operators into a SQL-like syntax, which expresses the sliding window queries over event streams. The grammar is constituted as follows.

```
CONTEXT {CPC|EXP|CPL|TCL}
SELECT <Attributeset1 [, Attribute AS Alias]>
FROM [WINDOW {TEMPORAL|COUNT <Range, Slide>}]
        [RESTRICT {Predicts [AND|OR Predicts]}]
        <StreamIn1>
|SEQUENCE [WINDOW {TEMPORAL|COUNT <Range, Slide>}]
                [RESTRICT {Predicts3 [AND|OR Predicts]}]
                <StreamIn2>
|KLNCLOSE [WINDOW {TEMPORAL|COUNT <Range, Slide>}]
                {[RESTRICT {Predicts [AND|OR Predicts]}],
                 [RESTRICT {Predicts [AND|OR Predicts]}],
                 [RESTRICT {Predicts [AND|OR Predicts]}]}
```

This framework is mainly composed of three categories of element, i.e. Clauses, Operators and Restrictions. The clauses SELECT, FROM and PUBLISH can be well understood as its literal meaning similarly with Cayuga, while the clause CONTEXT designates the semantic type of the entire query among one of the above four context definitions. Restrictions by the parameters, appearing as either temporal windows or predict combinations, can act on a single stream or the stream compositions connected by Operators SEQUENCE and KLNCLOSE. Under this framework, the illustrative Stock example can be given as:

```
CONTEXT CPC
SELECT <Name, Price, Volume>
FROM WINDOW {COUNT 3, TEMPORAL 6hrs}
            <<Stock>
KLNCLOSE WINDOW {TEMPORAL 30min, COUNT 1}
            {RESTRICT {$pre.Volume > 1000},
             RESTRICT {$suf.Name = $cur.Name},

RESTRICT {$suf.Price < $cur.Price}}

            <Stock>>
KLNCLOSE WINDOW {TEMPORAL 6hrs–End($pre).end%6hrs,
                        COUNT 1}

{RESTRICT {Start($pre).Price < $suf.Price},

             RESTRICT {Start($pre).Name = $suf.Name},
             RESTRICT {Start($pre).Price > $suf.Price}}
            <Stock>
PUBLISH <StreamOut
```

As shown in the previous framework, KLNCLOSE has three components: initiator, detector, teminator. It also worth noting that the WINDOW restriciton acting on each output is composed of both range and sliding components complying with either temporal and count semantics.

(2) NFA State window evaluation

In the above SQL-like query for the example, Stock is sequentially cited as the inputs of the two KLNCLOSE operators along with their predict restrictions. By picking the following events of the initiator, the first KLNCLOSE operator keeps the states along with their instances until any terminator arises, and then provides these instantial events to the second one as long as the duration of its output satisfies the range requisition of the given window. The second KLNCLOSE, subsequently, takes the first's output as its prefix input and detects the price of the remaining events compared with their prefix. After these, the top WINDOW outputs all the event series satisfying the declarative continuous three days requirement, and shifts a time intveral of size 6 hours over its input stream.

As for the difference of the window restriction from the predictions, Fig.3 gives an abstract NFA structure along with its two restricting evaluation. For simplicity, in Fig.3A, the initiator and detector conditions of the both KLNCLOSE are denoted by the directed edges annotated with initiated and detected respectively, and the two types of restrictions guiding B and C are labeled at State 3 and Output. Furthermore, in both B and C, the events filtered by the initiator, detector and terminator are accordingly stated as a_i, b_j and c_k, and each completely arrowed path from State 1 to Output gives one output under the both restrictions. It must be noticed that, in Fig.3C, each suffix event can change the state of the instance generated by the initiator of the first CLNCLOSE, and the window locating in the Forward edge of each

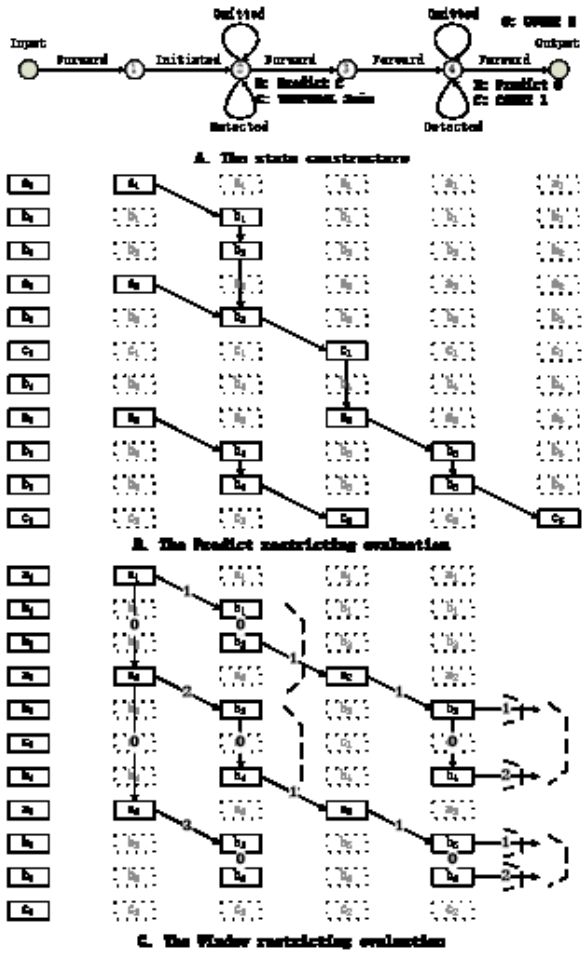

Fig. 3. The NFA window evaluation

prefixal operator controls the state transformation the suffixal operator. Moreover, this non-overlaped illustration can also handle the interval based primitive series by the initiator selected with one of the above four contexts.

(3) Sliding window management

The former sections demonstrate the window structure of the operator compared with the predict restrictions. Each path is composed of the qualified events linked by the coded edges, and the window restriction acting on the corresponding end state of each operator implements the transformation by inserting a new edge from the current event to its suffix event. Nevertheless, as the difference between State 3 and Output in Fig.3C, the COUNT and TEMPORAL restrictions trigger the transformation by the quantified count of the event iterations and the satisfied duration of the involved events respectively.

One import point of the above labeled edge evaluation is the coded strategy, which is closely related to the context semantic. Due to space limitation, we only give the coding procedure for the above illustration, and the four context semantics can be coded by directly adding edge into either the start or the intermediate state. Firstly, a linkage among the consequent start event, i.e. every event that begins a new instance, is constructed and each edge between every two of them is labeled with "0". Then, an increasingly labeled edge generates as long as each transformation arises from State 1. While in the composite event, each sequent isomorphic event generates an edge with increasing label for any transformation. As can be seen in Sate 5 of instances from a_1, event b_3 and b_4 generate the transformations to Output and mark their edge with different labels. Finally, the last temporary output can be organized with the temporally successive primitive events for aggregation. The result of Fig.3C is thus generated as $\{a_1,\ b_1,\ b_2,\ a_2,\ b_3,\ b_4,\ b_5,\ b_6\}$, or more thoroughly $\{\{a_1,\ b_1,\ b_2,\ a_2,\ b_3,\ b_4\},\ \{a_2,\ b_3,\ b_4,\ a_3,\ b_5,\ b_6\}\}$.

Besides the restricting requirement, WINDOW also involves the sliding processing management attaching importance to either the space efficiency or the statistical function. By default, WINDOW adopts a null-reference sliding strategy. On the one hand, the frontend events are pushed out from the cache as long as the temporal sliding is explicitly declared or the count sliding can wipe out all the citations of these events. On the other hand, especially in the count sliding circumstance, the cache must keep all the consecutive events of the referencing start event even though. As shown in Fig.4, these sliding procedures can be demonstrated by a slight modification on the state structure of Fig.3.

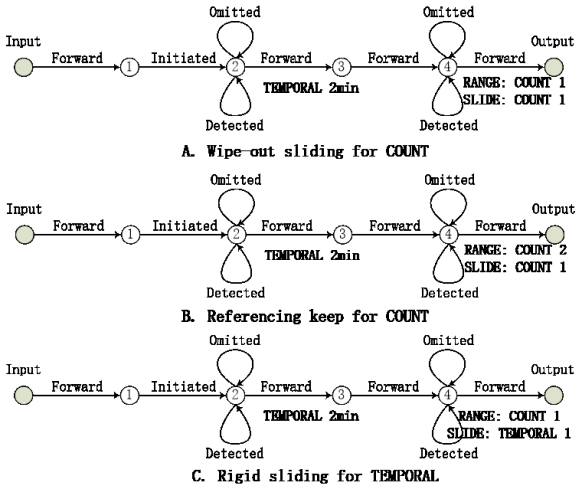

Fig. 4. Three sliding semantics in the context of window management

1. Wipe-out sliding for COUNT

The last WINDOW is abandoned so that both the range and the slide COUNT are marked as "1". Hence, the path 1-1-1-1 is treated as the final output, and

thereupon events $\{a_1,\ b_1,\ b_2,\ a_2,\ b_3\}$ is wiped out due to null-reference after output. Notice that, after this wiping, all the following processing will sharply differ from that of Fig.3C.

1. **Referencing keep for COUNT**

Similarly with Wipe-out sliding, as shown in Fig.4B, the third WINDOW is abandoned but the range COUNT is set by "2" and slide with "1". When 1-1-1-1 and 1-1-1-2 arises, these two series are simultaneously generated. Only the edge labeled with "1" from b_3 of State 5 to Output is deleted due to the referencing of the path 1-1-1-2.

1. **Rigid sliding for TEMPORAL**

Different from COUNT, the TEMPORAL sliding must erases the event from the cache and maintains the related linkages. As illustrated in Fig.4C, suppose the time window between event a_1 and b_3 is less than one unit time as given in Fig.4C, the path 1-1-1-2 will not be generated after we get the output 1-1-1-1; otherwise, we generate the path 1-1-1-2 and slide the window to abandon the event within the latest temporal unit window.

As shown in above, the window and sliding strategies are orthogonally divided in terms of both the count and temporal semantics. NFA maintains the instances of each state and generate the output once new instance is appended onto the cache. In the temporal sliding context, these instances are periodically deleted from the cache if their timestamps are more than a given temporal range old. In contrast, in the count-based sliding, the deletion will be triggered once each new instance is inserted. It's widely accepted that the time complexity of the NFA matching can be simplified to $O(n)$ to run a DFA-compiled regular expression against an n-length input series. Nevertheless, as the KNCLOSURE given in Section State-based window Semantic, the worst complexity would be $O(nm)$, where m is the number of edges of the NFA. After applying the window strategy, we can confine this computation within each window and abandon the out of date events by conducting the two proposed sliding strategies, which results in the time and space complexity of $O(wm)|\, w \ll n$.

4. Results and Discussion

We gave an open-sourcing implementation ICMDS by integrating the above orthogonal strategies into the Cayuga[6] to evaluate the output cardinalities and resources against different window settings. To verify the availability of the proposed window strategies, we conducted the experiments on the yahoo financial dataset to evaluate the efficiency of the proposed window management strategies. In our experiments, two sets of queries with different complexities were employed to validate the processing efficiency in terms of processing cost and memory usage. In detail, our experiments were divided into four categories, where the CPU and memory usages were plotted at the scale of input cardinalities. To further measure the effectiveness, the output cardinalities and processing time were taken as two dimensions to highlight the availability of the prosed window strategies.

(1) Comparisons simple event operator

The simple event operator took the real-time price of 20 stocks from yahoo as the test set. We feed ICDMS with the stock price, as the schema of <StockID, Price, Time>. The query is used to the denoted StockID that has an increasing price within a fix number of query window.

To measure the prosed window-based strategies in comparison with the traditional select-based methods, we confined the window size with COUNT 50. Since the traditional select-based cannot guarantee this statistical window, the framework needs to maintain all the historical dataset and enforce a self-join over a set of price series. We noted the join-involved atomic events as their OUTPUT and compared them with the cache size of the proposed framework. As shown in Fig.5, the proposed window-based strategy outperformed its competitor in all the four indexes since the substantial pruning power of the rolling window. Observe that the cache size can be linearly measured by the memory usage at the different input scales.

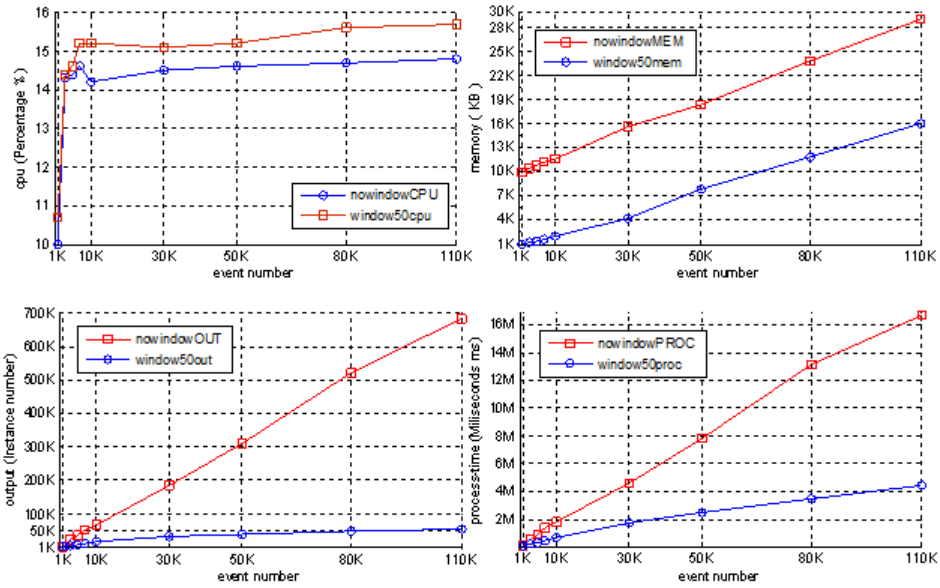

Fig. 5. Resource usage and efficiency comparison of the window-based framework against processing without window

We note here that the window range 50 denoted each stock were configured with window to accommodate the latest fifty atomic events. To this end, the initial CPU usage has slight difference in the first stage before the input cardinality was below three thousands events. Afterwards, the window-based strategy provided 20% CPU saving on average. Since this window is extremely small, our proposed strategy required only one tenth of memory of the non-window method, which results in the output of the window operator generate less than 10% of the non-window framework. In this regard, its process time showed a linear decrease to the later, exhibiting 12%˜20% of the non-window framework. In this simple query pattern, the NFA

inference has the linear complexity to the atomic events, which exhibited the linear corresponding relationships between the process time and the input cardinality.

(2) Comparisons with various window ranges

In this experiment, we conducted different window ranges to evaluate the efficiency and effectiveness of the proposed methods. The dataset is the same as given in section Comparisons simple event operator, where the various window ranges were given because of different time sensitive. In addition, we further introduced a select condition on the price to extract the stock that has the sharp changes. As compared to Fig.5, this results in at most 80% decreases of the memory usage when the event was extremely large (after the input is beyond 80K). Correspondingly, the output cardinalities in all four windows decreased linearly.

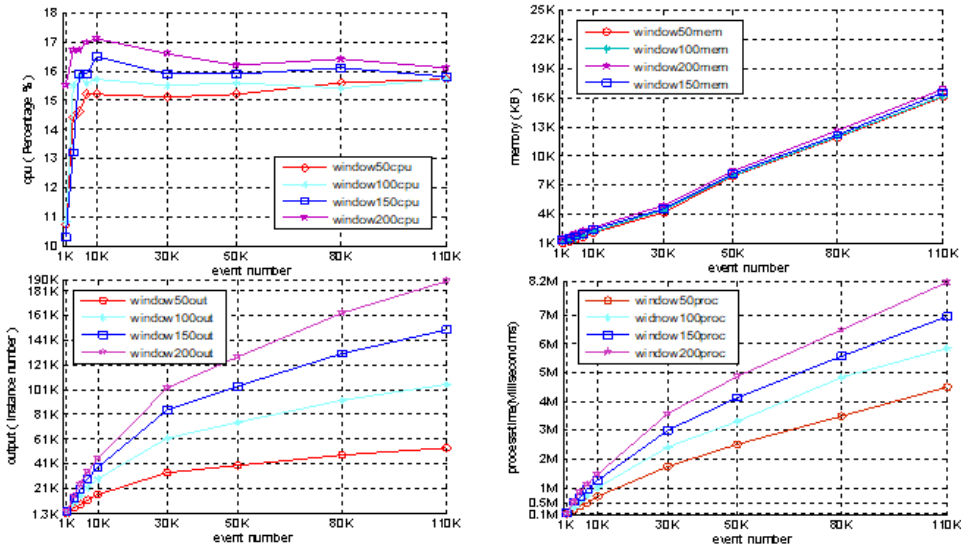

Fig. 6. Comparison towards complex queries with various window ranges

In Fig.6A, when we increasing the window range, the CPU usage increased less than 10% in the window 200 than that of window 50. Since our NFA computation is based on path coding as given in section NFA State window evaluation, the larger output as given in Fig.6C will not introduce significant increase of memory usage. As given in Fig.6B, the memory differences of all the settings are negligible even their window ranges linearly increase. While regarding the process time, the larger window range resulted in larger matching cardinality, and hence exhausted linear increase of computational time. As shown in Fig. 6C and 6D, the caching paths and process time in the window range 200 were almost 4 times of the window 50.

We note here that, as compared to the following results in different sliding settings, this experiment used the same sliding range as the window range. It means that, as given in the Wipe-out sliding strategy, the composite events that referencing the outdated atomic events will never generate the output when the temporal range between the outdated event and the oldest caching event are beyond a win-

dows range. As analyzed in the following sections, although the different sliding ranges will produce different composite events, the different sliding strategies will not significantly influence on the resources usages since the path coding as well as the incremental NFA matching techniques.

(3) Efficiency with various sliding ranges

To measure the cardinalities of composite events and resource usage in different window parameters, we use the same query condition as given in section Comparisons with various window ranges and vary the window sliding parameters in this experiment. The Wipe-out, Referencing keep and Rigid sliding strategies were measured as compared to the traditional rolling window. In addition, we also plotted the resources usage of the Referencing keep strategy similar to section Comparisons with various window ranges.

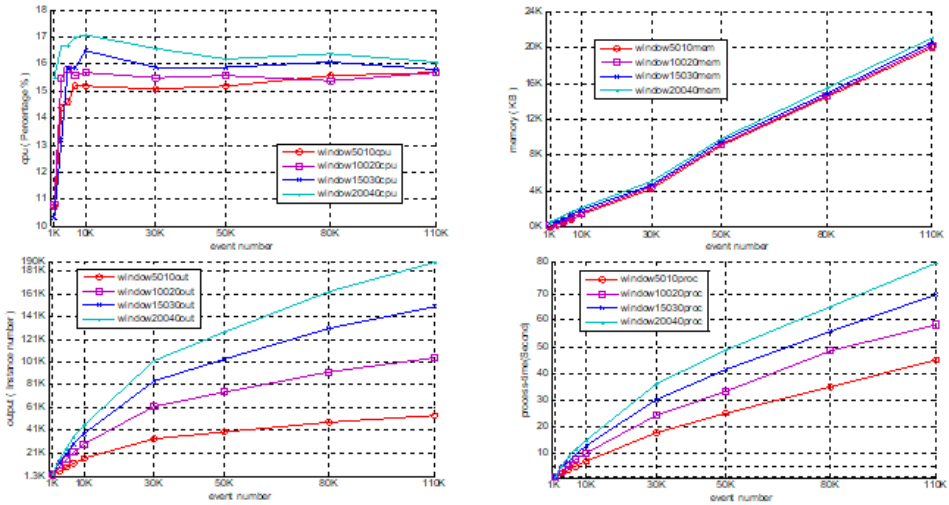

Fig. 7. Comparison towards complex queries with various window ranges and sliding parameters

As shown in section Sliding window management, the Reference keep strategy will abandon the atomic events of which the timestamp is outdated. It means that the event cache will keep a fix volume once the window range is given, regardless the sliding range. To this end, the NFA matching input will be constant for a given window size. Compare Fig.7 with Fig.6, there is slight difference in all the four indexes. With less than 2% jitter in CPU and process time, Fig.7 exhibited almost the same resource usage as shown in Fig.6. Recall that in each window range, Fig.6 from section Comparisons with various window ranges uses the same sliding size as the window range, while the sliding range of Fig.7 was proportional to the window range.

Table 2.Cardinality of composite events over 110K input with different sliding and window parameters

Ranges	Rolling	Wipe-out	Referencing keep	Rigid sliding
5010	39943	52342	61223	47823
10020	62232	105231	119823	89012
15030	92374	148724	167801	12999
20040	128714	189910	220005	16711

We further compared the different output cardinalities by varying the sliding strategies. Here, the traditional rolling strategy was enforced by fixing the sliding range as zero, which means that the composite event will never be produced once its oldest atomic event is outdated. With the atomic input cardinality 110K, the output cardinalities of all the four strategies are listed in Table 2.

It is clear that the rolling window omits the composite events across the window boundaries which results in the smallest output as compared to the three proposed sliding strategies. As for the later, the Referencing keep strategy maintained the atomic events among the cache regardless whether those atomic events have been used by the other generated events. In this regard, it generated 21% output more than the Wipe-out strategy. While in the Rigid sliding strategy, the outdated atomic event constricted by the sliding range will never been reused once the first composite event has been generated. As shown in Table 2, it reduced 23.1% output cardinality as compared to the Wipe-out strategy.

(4) Efficiency with couple window settings

Our last experiment employed an event sliding window couple with a predict-based sliding window to produce the composite events, where the stock price was relatively changing in a periodic time window. In general, this form of query has the smaller NFA matching input because of the selective filtering condition. However, as compared to the single sliding window, the higher matching complexity increases the running time in the sample input cardinality. We conducted the rolling strategy in the predict-based window while combining the temporal Wipe-out sliding strategy with the periodic window. In practice, we can configure this rolling predict window in the environment where an iterative sub-event is required. The path coding method given in section Sliding window management provide us an efficient way to share the prefix atomic events especially in the complex pattern matching environments. Similar to the above experiments, the input cardinalities are varying from 1K to 110K, we plotted the computational and memory resource in Fig.8, where the legends gave the form of temporal windows and the predict window parameters.

As compare the Fig.6A and 7A, the CPU usage of Fig.8A linearly decreases after the input reaching 10K. As demonstrated in Fig.3C, the predict events $\langle b_1, b_2\rangle$ and $\langle b_3, b_4\rangle$ are confined by the predict window, where the unsatisfied prefix paths will be removed by the predict filters. Meanwhile, once the predict windows are outdated, their succeed paths will be correspondingly deleted from the cache. It is highly desired that, the caching sub-events filtered by the predict condition should decrease after the cached event number reaching some peek value. As given in Fig.8A, the coupled sliding windows consumed the maximal CPU percentage at the scale of 10K.

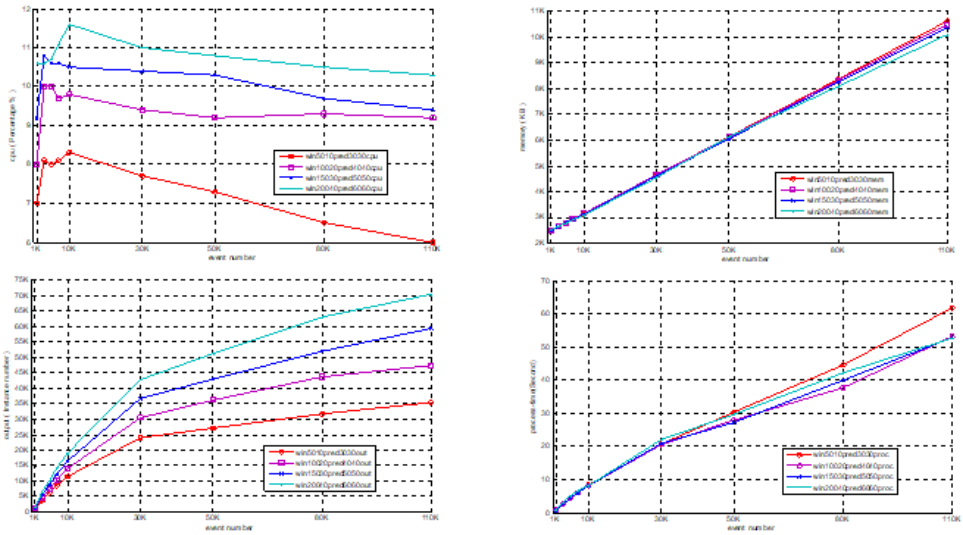

Fig. 8. Comparison towards complex queries with a couple of sliding window with various window and sliding ranges

Afterwards, the continuous deletion of the predict events will decrease at most 25% of the CPU usage. We also note here that, as compared the single sliding window given in section Efficiency with various sliding ranges, this coupled windows-based query is much more selective and its matching complexity is significantly larger. In contrast to Fig.7C, the composite events cardinalities of Fig.8C exhibited a similar relative changes yet decreased 12%˜68% in each sliding window configure. In addition, this higher matching complexity over a smaller matching cardinality made the process time insensitive to the window ranges. As shown in Fig.8D, the different window ranges only produced less than 12% differences process time. In comparison, Fig.7D in much more sensitive to the window ranges, where the window 20040 run 180% process time of window 50010.

In conclusion, the flexible sliding window and predict window gave us a two-level filtering method to extract the composite events towards the continuous atomic events, where the matching complexity increases in corresponding to the configured windows. Meanwhile, the memory usage is quite stable due to the path coding strategy regardless the caching composited events.

(5) Related works

We are witnessing a proliferation of business intelligence on workflows, supply chain management with RFID, large-scale attack monitoring and financial analysis. Due to the tremendous potentiality applied to these emerging fields, continuous pattern matching over the durative streaming events has aroused considerable attention in both academic and industrial worlds[7]. Pattern matching process over the event series, especially in the interval semantic environment, has increasingly received concerns in several pivotal yet controversial aspects such as temporal semantics definition, event strategy selection, consumption mod-e construction and

accomplished method foundation. Hence, some framework prototypes accompanied by a fair amount of implemental and optimized innovations emerged in the correlative application area.

Different from the data stream management system (DSMS), event pattern matching system as well as its regular expression methodology focus more on the expressivity, effectivity, flexibility and scalability. Furthermore, faced by the massive real-time data stream, the event pattern matching pays close attention to the resource utilization and the processing efficiency for providing both high throughput and low latency. Meanwhile, with the ever-expending fields of the complex event processing (CEP), the implemental strategy exhibits more flexible and scalable ability for the increasingly emerging requirement[3]. It should be noticed that, as one of the most prominent characteristics of the complex event pattern matching, the temporal restriction[1] and event feedback settlement is critical for either event composite or embedded sub-query circumstance.

In the event pattern matching systems, an event was originally defined as an instantaneous occurrence, and the last event's occurrence time was used as the time of the entire event expression[6]. The early event specification languages, especially in the active DBMSs, employ this detection semantic for compositing the instantaneous events[8]. In general, the development of the previous window semantics of the event processing is oriented towards the continuous query language, and thus can be divided into several abstract levels in terms of the granularity of the aggregate information extracted from the temporal raw tuples. The statistical stream window extracts the aggregations over a sliding time period especially for several at event operators[9]. By taking the event detection as an antecedent condition, the point event window usually provides a temporal range for restricting the event matching with the given patterns. Eminently different from the above two methods, by merely providing the semantic framework of the event processing, the formalization window language proposes the definition construction of event pattern confined to both the interval and tuple along with their related sliding strategies[2].

Apart from the constraint CEP, many researchers compromise between the efficiency and the accuracy by tailoring the complete history semantic[10]. It is worth noting that selection strategy and consumption context are two of the most representative propositions[11]. However, the former is essentially an alternative definition of Sequence-and-Kleene closure[12] founded on event filter, while the latter can be seen as a variation of both the tuple and temporal sliding window semantics in CQL[5]. As shown in 2, flexible window integration along with its sliding strategy is critical for the practicability of the pattern matching systems.

5. Conclusions

In this research, we provide a formalization framework to integrate the streaming window into the NFA-based event pattern matching system. In order to demonstrate the related properties, Cayuga is picked out from the existing mainstream prototypes due to its interval primitive event definition and its impressive sub-query support. We identify all the events with their duration interval and provide the

related strategy to assure their partial order. This feature both offers the systematic support under the complex sub-query application and guarantee the scalability for the distributed real-time environment. By providing the event insert operations, the window management can flexibly settle the two sliding constricts in all four semantic context circumstances. Fundamentally, the window confines each output of the low-level event to a certain range, and hence provide the high-level pattern with both temporal range and event counting semantics.

We identify the interval event duration and provide the related strategy to assure the temporal partial order. Instead of the instance selection and consumption mode employed in the existing propositions, the proposed semantic contexts exhibit a series of interval oriented event composition rules within the restricts of the partial relationship. The disorder and distributed streaming processing can be taken as our future directions.

References

[1] E. Wu, Y. L. Diao, S. Rizvi: *High performance complex event processing over streams.* In proceeding of the acm international conference on management of data (SIGMOD), Chicago *4* (2006) 407–418.

[2] D. Gyllstrom, J. Agarawal, Y. L. Diao, N. Immerman: *On supporting kleene closure over event streams.* In the 24th IEEE international conference on data engineering (ICDE), Cancun (2008) 1391–1393.

[3] J. Li, D. Maier, K. Tufle, V. Papadimos, P. A. Tucker: *Semantics and evaluation techniques for window aggregates in data stream. In Proceedings of the ACM international conference on management of data (SIGMOD).* Baltimore, Maryland, USA (2005) 311–322.

[4] F. S. Wang, S. R. Liu, P. Y. Liu: *Complex RFID event processing.* The international journal on very large database *18* (2009), No. 4, 913–931.

[5] A. Arasu, S. Babu, J. Widom: *The CQL continuous query language: semantic foundations and query execution.* The international journal on very large databases *15* (2006), No. 2, 121–142.

[6] A. J. Demers, J. Gehrke, B. Panda: *Cayuga: a general purpose event monitoring system.* In the 3rd Biennial conference on innovative data systems research (CEDR), Asilomar, CA (2007) 412–422.

[7] A. J. Demers, J. Gehrke, M. Hong: *Towards expressive publish/subscribe systems.* Sadhana (2006) 627–644.

[8] A. Majumder, R. Rastogi, S. Vanama: *Scalable regular expression matching on data streams.* In proceeding of the ACM international conference on management of data (SIGMOD) (2008) 161–172.

[9] J. Agrawal, Y. L. Diao, D. Gyllstrom, N. Immerman: *Efficient pattern matching over event streams.* Indian Journal of Pure and Applied Mathematics (2008) 147–160.

[10] R. Adaikkalavan, S. Chakravarthy: *SnoopIB: Interval-based event specification and detection for active databases.* Data & Knowledge Engineering *59*, (2006), No. 1, 139–165.

[11] R. S. Barga, J. Goldstein, M. H. Ali, M. Hong: *Consistent streaming through time: a vision for event stream processing.* IJ Applied Mechanics and Engineering (2007) 363–374.

Received November 16, 2017