Conjunction, Sequence, and Interval Relations in Event Stream Processing

Samujjwal Bhandari
Department of Computer Science
Texas Tech University, Lubbock, U.S.A
samujjwal.bhandari@ttu.edu

Susan D. Urban
Department of Industrial Engineering
Texas Tech University, Lubbock, U.S.A
susan.urban@ttu.edu

***Abstract*—The conjunction operator can be augmented with temporal constraints to define an arbitrary pattern of events in event stream processing (ESP). However, using temporal constraints to specify patterns can be complex. This research has defined an operator hierarchy, where the top of the hierarchy defines the conjunction operator and the leaves of the hierarchy define more specific semantics associated with a sequence of events. The use of the specialized operators simplifies pattern expression and make the sequence semantics clear. Furthermore, in an experimental study, patterns using operators from the hierarchy outperform patterns expressed using the conjunction operator with temporal constraints in run time performance, further validating the usefulness of the operator hierarchy.**

***Keywords*-event operators; sequence; event processing language; operator semantics;**

I. Introduction

REAL world applications have become increasingly event-driven in nature, focusing on the occurrence or non-occurrence of several activities or their combinations to respond to a situation of interest using event processing systems such as [1], [2]. The situations of interest are encoded as complex event patterns using a specific ESP language, where complex event patterns are specified using event operators and other events. These complex event patterns are matched by the event processing system to detect complex events.

Encodings of event patterns should be able to define a situation in a unique manner. However, existing work [3], [4], [1], [2] defines patterns in an ambiguous way. For example, suppose in a health care application, situations of importance are detected if i) a high temperature is detected after nausea is detected and ii) a high temperature event is followed by a low temperature. Both *i* and *ii* can be defined as sequential occurrences of two events. In these situations, there are several possibilities that can be true of the patterns. For instance, in situation *i*, the nausea event occurs while there is a high temperature. However, in case of *ii*, a high temperature cannot occur at the same time as a low temperature. This example shows that the sequence pattern may have different interpretations. One of the possible solutions to this problem is to use the conjunction operator with relevant temporal constraints to restrict the detection of the event patterns. When an interval-based event is considered, this approach can specify all the possible patterns [5]. However, it is desirable to specify the intended semantics in an explicit way such as by using special operators. For example, rather than expressing a sequential pattern, E_1 followed by E_2 as $AND(E_1, E_2)$ **WHERE** $E_1.t_e < E_2.t_e$, where t_e represents an ending time of event occurrence, it would be intuitive to express the condition as $SEQ(E_1, E_2)$, where the **SEQ** operator explicitly defines the intended meaning.

To address the issues of specification complexity and ambiguous operator interpretation, this research defines an operator hierarchy based upon the conjunction and the sequence operators. The top of the hierarchy defines the conjunction operator. Moving down the hierarchy introduces specialized operators to express more specific situations. Though any pattern defined using the operators from the hierarchy can be expressed as a combination of the conjunction operator and appropriate temporal constraints, the use of specialized operators in defining event patterns makes the pattern specification an easier and more expressive task.

To verify the usefulness of the operators in the operator hierarchy, the operators have been implemented with reference to the conjunction operator implementation and are found to run better than their alternative versions using the conjunction operator with temporal constraints. Moreover, the work in this paper makes the following contributions:

1) Design of operators to incorporate different meanings for the sequence and the conjunction operators.
2) Design of an operator hierarchy defining the relationships pertaining to time intervals using Allen's relations [6].
3) Experimental evaluation of operators from the operator hierarchy to describe usefulness of the newly defined operator hierarchy.

II. Related Work

Past work on event processing, such as Snoop [3], Ode [7], and SAMOS [8] have collectively defined a powerful set of event operators to specify complex event patterns. However, operators such as the sequence and the repetition operators are not consistently defined in these languages. SEL [9] analyzes these event languages and identifies problems with the semantics of the negation, sequence, and repetition operators. The recent languages ([1], [2]) have adopted operators similar to past work on event processing, but have not considered the semantic inconsistency among the definition of event operators as discussed in [9]. Non-overlapping sequence defined in [10] is considered to be an immediate sequence (i.e., an event A is immediately followed by B with no event in between), while sequence in [4] considers an arbitrary sequence (i.e., without restrictions on intervening events). These inconsistent definitions of the sequence operator as an overlapping and a non-overlapping sequence is relevant when an event is associated with an interval. The work in [5] defines a generic operator with a temporal constraint list to define all of the possible relations among intervals to remove inconsistency in the definition of event operators. However, the expression of event patterns becomes more complex.

III. Event Specification and Basic Concepts

For any two time intervals $i_1 = [t_1, t_2]$ and $i_2 = [t_3, t_4]$, there are thirteen possible relations defined as Allen's interval relations [6]. When an event e has event time $i = [t_s, t_e]$, e is said to have occurred over the interval i, where the event e started occurring at time point t_s and ended at time point t_e.

TABLE I: Situation Monitoring Event Definition

SON()	Stove is brought to ON state.
SOFF()	Stove is brought to OFF state.
LO()	A lid of a kettle is opened.
LC()	A lid of a kettle is closed.
KP()	A kettle is put on the surface.
IPOUR()	An item is put in the kettle.
T(v)	Regular event to provide temperature inside the kettle with the given value.
IP(items)	**TIMES**(IPOUR(), 3) **WITHIN** 2 MINUTES
KL()	**SEQ**(LO(), IP(items), LC()) **WHERE** IP.items.has({"water","milk","tea leaves"})
TP()	**AND**(SON(), KL(), KP(), T(v)) **WHERE** TEMP.v $\geq$ 100

Let us define a scenario to detect an activity defining the situation that the "tea has been prepared". Table I defines several events that are either observed or produced from the external environment (external events), or are a complex combination of other events (internal events). In Table I, the *T(v)* event is an external event that is generated by a thermometer. Other external events are *SON()*, *SOFF()*, *LO()*, *LC()*, *KP()*, and *IPOUR()*. An internal event is a composition of several external or other internal events. The *IP(items)* event, the *KL()*, and the *TP()* defined in Table I are internal events. The *IP(items)* event occurs when the *IPOUR()* event is detected three times within a 2 minutes window. The *KL()* event occurs with the sequential occurrence of the *LO()*, *IP(items)*, *LC()*, where *items* from the *IP(items)* event has water, milk and tea leaves in it. The *TP()* is represented as a pattern defining the occurrences of four events *SON()*, *KL()*, *KP()*, *T(v)*, where the temperature value is $\geq 100°$C.

Discussion in later sections will consider events from the situation monitoring application (Table I) and the occurrences of events shown in Figure 1.

IV. Issues and Semantics of Conjunction and Sequence Operators

This section analyzes the semantics of event operators to identify the issues that must be addressed to define the

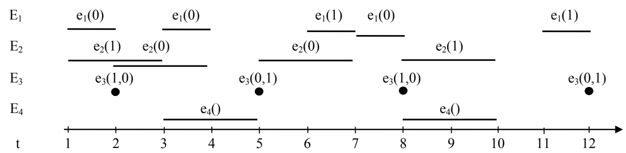


Fig. 1: Example Event Streams

semantics of operators in a clear and consistent way. Work such as that of [3], [4], [1] describes various powerful event constructs that can be categorized into conjunction, disjunction, repetition, negation, and sequence operators. Among different event operators, this work focuses on the semantics of the conjunction and sequence operators with interval-based temporal representation.

Conjunction of events E and F, denoted as $AND(E, F)$, occurs when both E and F occur without temporal ordering restrictions. Detection of $AND(E, F)$ starts when either E (or F) occurs and ends when F (or E) occurs. In case of interval-based semantics, all thirteen possible relations between interval are valid.

Example: The *TP()* event from Table I is a conjunction of five events *SON()*, *KL()*, *KP()*, *T(v)*, and *SOFF()*. Since the constituent events are combined using the **AND** operator, the *TP()* event occurs when all of the constituent events occur.

The complex event defined by a sequence operator has an implicit temporal constraint on events. A sequence of two events E and F, $SEQ(E, F)$, detects the occurrence of an event E followed by the occurrence of the event F. Detection of $SEQ(E, F)$ starts with the detection of an event E and ends with the detection of an event F. Such a requirement of event order by a sequence operator imposes temporal restrictions on event occurrences. When events are considered to be point-based, then $SEQ(E, F)$ is detected if and only if $E.t < F.t$, where t is the time of event occurrence. If events are interval-based, then $SEQ(E, F)$ is detected if and only if $E.[t_1, t_2] < E.[t_3, t_4]$, where $t_1 \leq t_2$ and $t_3 \leq t_4$. With these temporal conditions on event detection, we have two possible conditions: i) $t_2 < t_3$ and ii) $t_2 < t_4$. Though condition ii is included within condition i, the condition i gives the definition of a non-overlapping sequence operator, whereas the condition ii gives the definition of an overlapping sequence operator. In this section, the condition ii is used to define the sequence operation unless otherwise mentioned.

Example: The KL() event is a sequence of *LO()*, *IP(items)*, and *LC()*. The KL() event occurs when all of the constituent events occur with the constraint $LO.[t_s, t_e] < IP.[t_s, t_e] < LC.[t_s, t_e]$.

Using the definition of sequence, $SEQ(E, F)$ says, E must occur before F. As discussed in this subsection, there are two possibilities for the sequence operator defining overlapping and non-overlapping sequences. Past work on event processing considers either an overlapping version of a sequence operator or a non-overlapping version. When an overlapping version of a sequence operator is used, then the sequence operator can be used to detect non-overlapping events, but it requires an explicit temporal condition to specify that the non-overlapping sequence is intended. However, if a non-overlapping sequence is used, it cannot be used to specify an overlapping sequence. One of the solutions to this problem could be the use of the temporal filter on overlapping sequence. However, an application can demand specification of sequences of both kinds and, to make event specification more explicit, a separate operator for non-overlapping sequence may be suitable.

Examples: If only a non-overlapping version is defined, the sequence of events, such as *SEQ(SON(),SOFF())* is intuitively explicit as the stove on event precedes the stove off event. Let us encode the pattern specifying the situation that describes the condition where a kettle loading (*KL()*) process is followed by the detection of an item put (*KP()*) event. In this case, *KL()* can start before the *KP()* event and ends after the *KP()* event. This condition defines the sequence given as *SEQ*(*KP()*, *KL()*), which has the meaning of an overlapping sequence that cannot be encoded using the non-overlapping version.

V. Operator Design

This section addresses the semantic issues discussed in Section IV to design a set of event operators in a way such that each operator has a clear and consistent definition, and the operators are expressible in terms of its intended meaning. While discussing semantics of operators, only the binary operators are considered. One can extend the semantics of binary operators to their n-ary version using the binary semantics. Also, for readability, events are represented using its name only, instead of its schema.

A. Allen's Relations, Sequence, and Conjunction

Allen's 13 relations [6] define all of the possible relations between two intervals when both end points of intervals are fixed. When we have open end point relations, then the disjunction of Allen's relations to capture the desired relation can be complex. Also, the disjunction of Allen's relations or a hierarchical representation of composite relations [11] cannot express pair-wise relations among events due to non-transitivity of some operators such as overlaps [5]. Regardless of difficulty in specifying complex interval patterns, however, Allen's operators are concise constructs for capturing common interval relations. The use of these relations to express event patterns also defines the meaning of the pattern in an expressive manner. For example, the pattern defining the situation that an event E overlaps with an event F, $OVERLAPS(E,F)$ is easier to understand than *AND*(E,F) **WHERE** $E.t_s < F.t_s$ **and** $F.t_s < E.t_e$ **and** $E.t_e < F.t_e$. On the other hand, a pattern expressing the situation such as an event E ends before an event F is easier to understand as *AND*(E,F) **WHERE** $E.t_e < F.t_e$ than the encoding *OR*(*BEFORE*(E,F), $OVERLAPS(E,F)$, $MEETS(E,F)$, $STARTS(E,F)$, $DURING(E,F)$). The paragraphs that follow discuss the use of event operators and the use of temporal constraints in a suitable way to balance between understandability of expression using specific operators and the complexity of specifying the patterns.

Consider the sequence operator (**SEQ**) discussed in Section IV. To be consistent, consider that the semantics of the **SEQ** operator includes overlapping or non-overlapping occurrences of events ordered by end points. Then such a definition will include both interpretations of **SEQ** events from Section IV and this definition of **SEQ** corresponds to the definition of an overlapping sequence from Section IV. When an event pattern seeks only the overlapping sequence or only the non-overlapping sequence, then either a sequence operator with a required temporal restriction can be used to define the restricted sequence, or different operators defined for each condition can be used to specify a restricted sequence. For example, *SEQ*(E,F) **WHERE** $E.t_e < F.t_s$ is the same as Allen's *before* operator. The idea of using temporal constraints with operators or the definition of an equivalent operator defines the hierarchy of sequence operators with respect to Allen's operators. Figure 3 shows the hierarchy of a sequence operator with respect to two different forms of sequence operations along with the relation to Allen's operators. The hierarchy shown in Figure 3 depicts that the sequence operation combines *before*, *meets*, *overlaps*, *starts*, and *during* relations from Allen's relations. Section V-B further analyzes and discusses the sequence hierarchy to define operators.

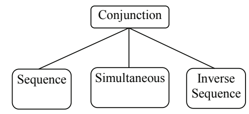


Fig. 2: Conjunction Hierarchy

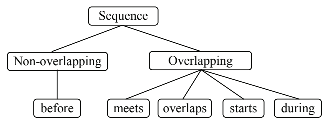


Fig. 3: Sequence Hierarchy

Conjunction (**AND**) is one of the well understood operations in event processing. The use of the conjunction operator does not define temporal restrictions on event occurrences, so the use of temporal constraints with **AND** can define every possible combination of interval relations. This idea of using temporal constraints with the conjunction operator is similar to the work done in [5], where an operator for an interval sequence *iseq*[1] is defined with the temporal constraints to define arbitrary relations on intervals. The sequence operator can be considered as a temporally restricted conjunction operator such that *SEQ*(E,F) = *AND*(E,F) **WHERE** $E.t_e < F.t_e$. Using the relations between the sequence operator and the conjunction operator, the hierarchy shown in Figure 2 can be defined. The conjunction hierarchy shown in Figure 2 defines restrictions of conjunctive combinations of events as sequential combinations, simultaneous combinations, or the inverse of sequential combinations. Section V-B further discusses the conjunction hierarchy to describe the event operators discussed in this work.

B. Operator Hierarchy

Allen's thirteen relations provide a powerful way to express relationships among interval-based events. However, there are

[1]The paper [5] defines it as ISEQ. As this work also defines an operator called **ISEQ** to denote inverse **SEQ**, we use *iseq* to denote an interval sequence operator.

2^{13}-1 (8191) total relations when Allen's relations are combined using disjunctions. When all 8191 relations are treated as operators, then the complexity of pattern specification reduces, although, the large number of event operators to specify an event pattern is undesirable from a language point of view. Further, it is not practical to define all of the operators. To cope with this situation, this section describes an operator hierarchy that defines a small set of event operators as shown in Figure 4.

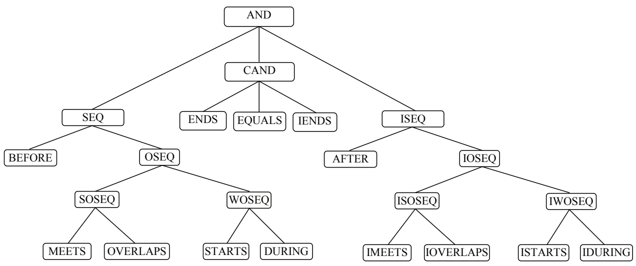


Fig. 4: Operator Hierarchy Defining Conjunction and Sequence

1) Conjunction Operators: Consider the hierarchy shown in Figure 2. The figure shows the trichotomy between two intervals i_1 and i_2 that defines sequence to describe $i_1 < i_2$, simultaneous to describe $i_1 = i_2$, and the inverse sequence to describe $i_1 > i_2$. This trichotomy between intervals considers two events as simultaneous if they end at the same time period. So, the actual relation here is described by a trichotomy between end time-points of two intervals expressed as natural numbers. In other words, if t_{e1} is the end time of the interval i_1 and t_{e2} is the end time of the interval i_2, then $i_1 < i_2$ if and only if $t_{e1} < t_{e2}$, $i_1 = i_2$ if and only if $t_{e1} = t_{e2}$, and $i_1 > i_2$ if and only if $t_{e1} > t_{e2}$. With this idea, the **AND** operator is divided into three different operators **SEQ**, **CAND**, and **ISEQ** as shown in Figure 4. Conceptually, the **AND** operator defines the relation that includes all of Allen's relations, since conjunction has no temporal constraint. The **CAND** operator is meant for concurrent conjunction and, as there are three interval relations specifying the same end time, Figure 4 defines three of Allen's operators, *ends*, *equals*, and *ended by* as **ENDS**, **EQUALS**, and **IENDS**, respectively, as specializations of a concurrent conjunction. Two other operators **SEQ** and **ISEQ** are the inverse of each other and this work does not discuss **ISEQ** in detail as its concepts can be derived from the **SEQ** operator. The hierarchy shown in Figure 4 defines the equivalent event patterns shown in Table II.

Example: In Figure 1, up to time $t = 12$, we can observe that *CAND*(E_2, E_4) is detected as *CAND*$^{[8,10]}(E_2, E_4)$. Also notice that the **CAND** pattern is equivalent to the equivalence 3 in Table II, where *AND*(E_2, E_4) is detected as
AND$^{[1,5]}(E_2, E_4)$, *AND*$^{[2,5]}(E_2, E_4)$,
AND$^{[1,10]}(E_2, E_4)$, *AND*$^{[2,10]}(E_2, E_4)$,
AND$^{[5,10]}(E_2, E_4)$, *AND*$^{[3,7]}(E_2, E_4)$,
AND$^{[3,10]}(E_2, E_4)$, and *AND*$^{[8,10]}(E_2, E_4)$.

Similarly other equivalences can be verified.

TABLE II: Equivalent Event Patterns

1)	*AND*(E, F)	$\equiv$	*OR*(*SEQ*(E, F), *CAND*(E, F), *ISEQ*(E, F))
2)	*SEQ*(E, F)	$\equiv$	*AND*(E, F) **WHERE** $E.t_e < F.t_e$
3)	*CAND*(E, F)	$\equiv$	*AND*(E, F) **WHERE** $E.t_e = F.t_e$
4)	*CAND*(E, F)	$\equiv$	*OR*(*ENDS*(E, F), *EQUALS*(E, F), *IENDS*(E, F))
5)	*ISEQ*(E, F)	$\equiv$	*AND*(E, F) **WHERE** $E.t_e > F.t_e$
6)	*ENDS*(E, F)	$\equiv$	*CAND*(E, F) **WHERE** $E.t_s < F.t_s$
7)	*EQUALS*(E, F)	$\equiv$	*CAND*(E, F) **WHERE** $E.t_s = F.t_s$
8)	*IENDS*(E, F)	$\equiv$	*CAND*(E, F) **WHERE** $E.t_s > F.t_s$

2) Sequence Operators: In Figure 3, there are five Allen's relations that are clustered within the hierarchy of the sequence operator with respect to the definition discussed in the previous sections. The other five Allen's relations correspond to the inverse of sequence that can be described similarly as the sequence hierarchy is described. The remaining three Allen relations correspond to concurrent conjunction as discussed in Subsection V-B1. Figure 4 depicts that the five Allen's relations *before*, *meets*, *overlaps*, *starts*, and *during* are categorized into two different groups defining a sequence that does not overlap (*BEFORE*(E, F) implied by Allen's *before* relations and an overlapping sequence (*OSEQ*(E, F)) implied by the other four relations. The overlapping sequence can be further sub-divided into two groups based upon the relationships between the starting time points of the intervals. For the first division, the sequence of E and F has $E.t_s < F.t_s$ and for the second division $E.t_s \geq F.t_s$. The former sub-division is given the name, strong overlapping sequence (*SOSEQ*(E, F)) where both the start time points and the end time points satisfy the $<$ relation. The later sub-division is understood as a weak overlapping sequence (*WOSEQ*(E, F)), where a start time is strictly not following the $<$ relation. Using the hierarchy shown in Figure 4, the equivalent event patterns for sequence operators can be similarly defined as in Table II, which are omitted due to space constraints. *Example:* In Figure 1, up to time $t = 12$, we can observe that *SOSEQ*(E_1, E_2) is defined as *SOSEQ*$^{[1,4]}(E_1, E_2)$ and *SOSEQ*$^{[7,10]}(E_1, E_2)$ and *WOSEQ*(E_1, E_2) is defined as *WOSEQ*$^{[1,3]}(E_1, E_2)$. With this, *OSEQ*(E_1, E_2) is detected as one of the **SOSEQ** or the **WOSEQ** pattern is detected that verifies the equivalence: *OSEQ*$(E_1, E_2) \equiv$ *SOSEQ*(E_1, E_2) *OR* *WOSEQ*(E_1, E_2).

VI. Experiments and Results

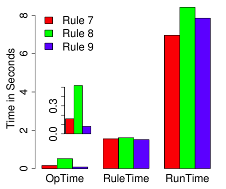
A. Experimental Setup

The experiments were conducted with 12 different pattern groups having equivalent patterns corresponding to each operator from the hierarchy with the implementation of the **AND** operator and the operators from subtrees rooted at **SEQ** and **CAND** in Figure 4. Table III shows examples of two pattern groups, where the first group has three equivalent patterns defined for the **ENDS** operator (Rule 7 - Rule 9) and the second group has five equivalent patterns defined for the **OVERLAPS** operator (Rule 35 - Rule 39). For space reasons, discussion of all the groups with equivalent patterns are omitted from this paper.

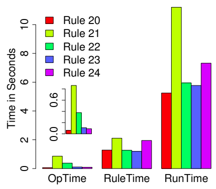
TABLE III: Examples of Equivalent Pattern Groups

No.	Pattern
7	$ENDS(E_4(), E_2())$
8	$AND(E_4(), E_2())$ **WHERE** $E_4.t_e = E_2.t_e \wedge E_4.t_s > E_2.t_s$
9	$CAND(E_4(), E_2())$ **WHERE** $E_4.t_s > E_2.t_s$
35	$OVERLAPS(E_5(), E_2());$
36	$AND(E_5(), E_2())$ **WHERE** $E_5.t_s < E_2.t_s \wedge E_2.t_s < E_5.t_e \wedge E_5.t_e < E_2.t_e$
37	$SEQ(E_5(), E_2())$ **WHERE** $E_5.t_s < E_2.t_s \wedge E_2.t_s < E_5.t_e$
38	$OSEQ(E_5(), E_2())$ **WHERE** $E_5.t_s < E_2.t_s \wedge E_2.t_s < E_5.t_e$
38	$SOSEQ(E_5(), E_2())$ **WHERE** $E_2.t_s < E_5.t_e$

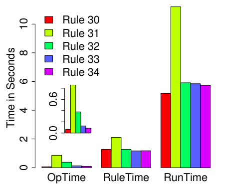
Each pattern was run 10 times for an episode of 3000 time units. For each run, the total time taken by all operators (OpTime), the total time taken by the rule processor for processing a rule after an event to be processed has been identified by event processor (RuleTime), and the total time taken by the event processor (RunTime) were recorded.



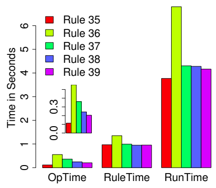
(a) Rule 7 - Rule 9



(b) Rule 20 - Rule 24



(c) Rule 30 - Rule 34



(d) Rule 35 - Rule 39

Fig. 5: Experimental Results of Run-Time Performance of Operators

B. *Experimental Results*

Figure 5 shows the comparisons of run-time for four different pattern groups. In each sub-figure, the first group of bars shows the OpTime, the second group of bars shows the RuleTime, and the third group of bars depicts the RunTime. Notice that each graph in Figure 5 has a graph in an inset to show the magnified form of the OPTime. In all of the graphs and all of the bar groups, the first bar shows the running time for the pattern using the operators designed in this work. Similarly, the second is associated with the equivalent pattern using the **AND** operator with temporal constraints. Other bars represent equivalent patterns, as discussed in Section V, using the parent operator with the temporal constraints, or the use of disjunction of the immediate children operators (See Figure 4).

From the experiments with run-time performance, the following results about the event operators can be observed:

a) 1: The RunTime for patterns defined using the operators from the operator hierarchy is minimum compared to all other alternatives at the higher levels of the hierarchy due to the filtering of incoming events prior to processing them (except for the **ENDS** operator), while an alternative scheme does the post-processing of incoming events.

b) 2: The RuleTime is better than other alternatives for the patterns using the operator set defined in this work, except for some patterns with the graphs shown in Figure 5.

a The graph in the sub-figure 5a shows that the pattern using the **ENDS** operator (Rule 7) takes more time to process the rule than the pattern defined using the **CAND** operator (third bar - sub-figure 5a – Rule 9). This is the direct consequence of processing the event buffer required for the **ENDS** operator and filtering the events after the buffer management. Whereas, Rule 9 detection does not maintain a buffer and events are filtered prior to the detection process.

b The RuleTime for the pattern using the **SOSEQ** operator (sub-figure 5b, first bar – Rule 20) is greater than the pattern using the **OSEQ** operator with temporal constraints (fourth bar – Rule 23). Though Rule 20 spends less time in processing event operators, Rule 20 uses expensive operations such as pattern duplication to manage partial patterns. This makes the RuleTime for Rule 20 greater than Rule 23. In a similar manner, the RuleTime for the patterns using the **MEETS** operator, represented by the first bar (Rule 30) in the sub-figure 5c is higher than the pattern using the **OSEQ** operator (fourth bar – Rule 33) with temporal constraints and the pattern using the **SOSEQ** operator (fifth bar - Rule 34) with temporal constraints. Also, Rule 35 using the **OVERLAPS** operator (first bar – sub-figure 5d) has a RuleTime greater than Rule 38 (fourth bar) using the **OSEQ** operator with temporal constraints and Rule 39 (fifth bar) using the **SOSEQ** operator with constraints.

c) 3: The OpTime is better for the patterns using the operators discussed in this work than other alternative representations for all the groups except for pattern using the **ENDS** operator (Figure 5a- Rule 7). For reasons discussed above, in case of the **ENDS** operator's buffer management and post processing filtering of events, Rule 9 runs faster than Rule 7.

d) 4: Processing with use of the new set of operators always runs faster than the use of the **AND** operator with temporal constraints for all cases of run-time comparisons.

e) 5: When a complex pattern is defined by the temporal constraints among different groups, then it is appropriate to define them using the closest upper level operator with temporal constraints or the disjunction of different operators. This result is seen from the run time comparisons of patterns shown in the graphs represented by the third and beyond bars.

As a conclusion, with the analysis of the run time results discussed above, the set of operators from the operator hierarchy are performing better than using other alternative approaches that use parent operators from the hierarchy with additional temporal constraints or the disjunction of the children operators from the operator hierarchy in terms of total running time.

VII. Conclusions

The work in this paper has identified ambiguities in the definition of event operators in current event processing languages. The conjunction operator and its relationship with the sequence operator is used to define several possible sequential operations using the idea of Allen's interval relations and a relation hierarchy. The definition of the operator hierarchy defines how an event operator should be selected to achieve the required semantics, making the event specification semantically clear. All the operators discussed in this paper were evaluated by comparing the run-time performance. The experimental results showed that the new set of operators performs better than other alternative approaches on run-time.

There are several possible future research directions. The repetition operator is one of the powerful constructs in event pattern specification. Current event processing systems, however, define the repetition operator in an incomplete way. For example, if one specifies five occurrences of an event E, is it that we are expecting sequential repetition over the time (semantics of **SEQ**) or that the repetition does not have any temporal constraints (semantics of **AND**)? Other issues related to the definition of event operators, such as event time computation and event detection have not been addressed in this work and are left as future work.

References

[1] R. S. Barga, J. Goldstein, M. Ali, and M. Hong, "Consistent Streaming Through Time : A Vision for Event Stream Processing," in *3rd Biennial Conference on Innovative Data Systems Reseaarch (CIDR)*, 2007, pp. 363–374.

[2] F. Bry and M. Eckert, "Rule-Based Composite Event Queries: The Language XChangeEQ and Its Semantics," in *Web Reasoning and Rule Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4524, pp. 16–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72982-2_2

[3] S. Chakravarthy and D. Mishra, "Snoop : An Expressive Event Specification Language For Active Databases," *Data Knowl. Eng.*, vol. 14, no. 1, pp. 1–26, 1994.

[4] R. Adaikkalavan and S. Chakravarthy, "SnoopIB: Interval-Based Event Specification and Detection for Active Databases," in *Advances in Databases and Information Systems*. Springer-Verlag Berlin Heidelberg, 2003, pp. 190–204. [Online]. Available: http://www.springerlink.com/content/d3n1vnj0bhp2cdpm

[5] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin, "Complex event pattern detection over streams with interval-based temporal semantics," in *Proceedings of the 5th ACM international conference on Distributed event-based system*, ser. DEBS '11. New York, NY, USA: ACM, 2011, pp. 291–302. [Online]. Available: http://doi.acm.org/10.1145/2002259.2002297

[6] J. F. C. . Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, pp. 832–843, 1983.

[7] N. H. Gehani, H. V. Jagadish, and O. Shmueli, "Composite Event Specification in Active Databases : Model & Implementation," in *18th International Conference on Very Large Data Bases V*, 1992, pp. 327–338.

[8] S. Gatizu and K. R. Dittrich, "Events in an Active Object-Oriented Database System," pp. 1–14, 1993.

[9] D. Zhu and A. Sethi, "Sel, a new event pattern specification language for event correlation," in *Proceedings of Tenth International Conference on Computer Communications and Networks*, 2001, pp. 586–589.

[10] B. Mozafari, K. Zeng, and C. Zaniolo, "High-performance complex event processing over xml streams," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 253–264. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213866

[11] P.-s. Kam and A. W.-C. Fu, "Discovering temporal patterns for interval-based events," in *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, ser. DaWaK 2000. London, UK, UK: Springer-Verlag, 2000, pp. 317–326. [Online]. Available: http://dl.acm.org/citation.cfm?id=646109.679272