Bridging the Gap: Complex Event Processing on Stream Processing Systems

Ariane Ziehn[1], Philipp M. Grulich[1], Steffen Zeuch[1], Volker Markl[1,2]
TU Berlin[1], DFKI GmbH[2], Germany
firstname.lastname@(tu-berlin.de[1], dfki.de[2])

For our operator mapping proposed in *Bridging the Gap: Complex Event Processing on Stream Processing Systems*, we first derive formal definitions for Simple Event Algebra [8] operators by modifying well-defined operators of the active database representative Snoop [4]. Then, we map our derived operator definitions to one (1:1 Mapping) or a combination of relational algebra operators (1:n Mapping), defined by Codd et al. [5, 6]. In the remainder, we elaborate on the correctness of our mapping, i.e., the detecting all pattern matches. As our derived SEA operator definitions are equivalent to the definitions of its relation counterpart, the question of correctness boils down to our modifications applied to the operators of Snoop [4]:

Snoop defines its operators as Boolean functions to detect patterns in a point-in-time manner. The Boolean function P(ts) returns *true* if an (complex) event $e \in T$ occurs at the point in time ts, else $false$ [4].

$$P(ts) = \begin{cases} True, \text{ iff } e \in T \wedge \text{e.ts = ts} \\ False, \text{ else} \end{cases} \quad (1)$$

In order to match the specification of SEA and the stream processing paradigm CEP, we adjust Snoop's general operator semantics with two *operator modifications*:

Modification 1: In order to handle high-frequent and unbounded streams, stream processing uses windows to create finite substreams of length W [3, 7]. We use ts_b and ts_e to define an arbitrary time interval $[ts_b, ts_e)$ so that $W = ts_e - ts_b$. Thus, we adjust the input of P to a set of events $E_{in} = \{e_1,...,e_n\}$, where for each event e_i it is true that $e_i.ts \in [ts_b, ts_e)$. We modify Equation 1 as follows:

$$[P]_{ts_b}^{ts_e} = \begin{cases} True, \text{ iff } \exists e_i \in T \wedge e_i.ts \in [ts_b, ts_e) \\ False, \text{ else} \end{cases} \quad (2)$$

Modification 2: To fulfill the closure properties of SEA, we further need to modify the output of the function $[P]_{ts_b}^{ts_e}$ (Equation 2). In particular, the function either returns the set of events $E_{out} = \{e_1,...,e_m\}$, where each event e_j satisfies all constraints contained in P or an empty set and no Boolean value.

$$[P^*]_{ts_b}^{ts_e} = \begin{cases} \{e | e \in T \wedge e.ts \in [ts_b, ts_e)\} \\ \emptyset, \text{else} \end{cases} \quad (3)$$

Applying both modifications to the operator definitions of Snoop yields the final definitions of our operators. While, on the one hand, we modify in- and output semantics, Modification 1 incorporates the window operators that discretize the stream(s) into a finite substream, i.e., a set of tuples [2]. As a relation is also defined as a set of tuples, the intra-window semantics of our operators, i.e., the operation applied to the set of tuples assigned to a single window, are semantically equivalence to relational algebra operators [2, 14]. We prove this theorem in Section 1. For the correctness of our modifications, i.e., detecting all matches $M = ce(e_1,..,e_n)$, we also need to ensure that no complex event ce is lost by discretizing the stream S. To this end, we analyze and specify the inter-window semantics of sliding windows that define how subsequent windows are created and, thus, how the stream is discretized. Finally, given our window specification, we prove that our operator definitions detect all complex events in Section 2.

EDBT 2024, 25th March-28th March, 2024, Paestum, Italy

ACM ISBN XXXXXXXXXXXX.

1 PROOF OF INTRA-WINDOW SEMANTICS

In this section, we provide formal proof for the intra-window semantics of our mapping inspired by the work of Kolchinsky and Schuster [9]. To this end, we use the conjunction operator as example. However, our proof can be applied to all other operator definitions, respectively.

Conjunction Operator. Let us first recap the formal definition of the conjunction operator. In particular, applying Modification (1) and (2) yields the following definition of the conjunction operator:

$$(T_1 \wedge T_2) = \{(e_i, e_j) \mid e_i \in T_1 \wedge e_j \in T_2\} \quad (4)$$

Furthermore, by definition (e_i, e_j) is a valid output of p if $max(i,j) - min(i,j) < W$, where i,j are the timestamps $ts \in \mathbb{N}$.

The Cartesian product is formally defined by Codd et al. [5] as follows:

$$R_1 \times R_2 = \{(r_n, r_m) \mid r_n \in R_1 \wedge r_m \in R_2\} \quad (5)$$

, where n,m are the indices of the tuples r_n, r_m in R_k $(0 \leq n,m < |R_k|)$, where $|R_k|$ defines the cardinality of R_k.

Theorem 1. *Let us consider an elementary Cartesian product query q specified by the relations R_1 and R_2. Additionally, let us consider an elementary conjunction pattern p specified by the event types T_1 and $T2$ and a time window W. Then, q is semantically equivalent to p, i.e., both requests specify the same output set [10].*

Proof: We will prove this theorem by double inclusion. To this end, let us consider that both types T_1 and T_2 correspond to the relations R_k, respectively. In particular, the schema $T_k(a_1,...,a_n)$ is identical to the schema $R_k(a_1,...,a_n)$. Thus, each event $e \in T_k$ has identical attributes to the tuple $r \in R_k$. Furthermore, each event $e_l \in T_k$ corresponds to a tuple $r_h \in R_k$. In addition, let S be an unbounded stream containing events of type T_1 and T_2. $[S]_{ts_b}^{ts_e}$ is a finite substream of S, where by definition $\forall e_l \in S(l \in [ts_b, ts_e))$ and $W = ts_e - ts_b$.

$\mathbf{p} \subset \mathbf{q}$: Let (e_i, e_j) be a complex event ce detected by p. Then, by definition, $e_i \in T_1$, $e_j \in T_2$, $i,j \in [ts_b, ts_e)$ and $max(i,j) - min(i,j) < W$.

As T_1 and T_2 correspond to the relations R_1 and R_2, $\exists r_n \in R_1$ that resembles e_i and $\exists r_m \in R_2$ that resembles e_j.
Let us assume that $|R_k| = W \cdot r(T_k)$, where $r(T_k)$ is the arrival rate of T_k. Thus, $|R_k| = |[T_k]_{ts_b}^{ts_e}|$, i.e., two sets of the same size, and every index h of a tuple $r_h \in R_k$ resembles the timestamp of its corresponding event e_l. It follows that for each pair, e_l and r_h, it is true that $l = h$ and $l,h \in [ts_b,ts_e)$. Thus, $R_k = [T_k]_{ts_b}^{ts_e}$, and consequently, ce is also contained in the output of q. □

$\mathbf{q} \subset \mathbf{p}$: Let (r_n,r_m) be an output tuple t of q. Then, by definition, $r_n \in R_1$ and $r_m \in R_2$ and $0 \leq n,m < |R_k|$. As T_1 and T_2 correspond to the relations R_1 and R_2, $\exists e_i \in T_1$ that resembles r_n and $\exists e_j \in T_2$ that resembles r_m. Let us define the time window as $W = |R_k|$ and $r(T_k) = \frac{|R_k|}{W}$. Thus, $|[T_k]_{ts_b}^{ts_e}| = W \cdot r(T_k) = |R_k|$. Furthermore, every timestamp $l \in [ts_b,ts_e)$ of an event e_l resembles to the index h of its corresponding $r_h \in R_k$. It follows that for each corresponding pair e_l and t_h, it is true that $l = h$ and $l,h \in [ts_b,ts_e)$. Thus, $[T_k]_{ts_b}^{ts_e} = R_k$, and consequently, the tuple t is also contained in the output of p. □

Extension of Proof. Predicates. While the proof of the conjunction can be respectively applied to the mapping of the disjunction operator, operator definitions of other mappings, i.e., sequence, iteration, and negation, additionally contain a set of predicates C. Predicates specify conditions either on one event type σ_{T_1}, or between attributes of two event types σ_{T_1,T_2}. σ_{T_1,T_2} conditions enable mappings towards join types, i.e., Equi and Theta Joins. To complete our formal proof and make it applicable to all mappings, we extend Theorem 1 as follows:

Theorem 2. *Let us consider an elementary Cartesian product query q specified by the relations R_1 and R_2 and a set of constraints $C_q = \bigwedge \sigma_{s_{R_1,R_2}}$, where $s \in \mathbb{N}$. Additionally, let us consider an elementary conjunction pattern p specified by the event types T_1 and $T2$, a set of constraints $C_p = \bigwedge \sigma_{s_{T_1,T_2}}$, and a time window W. Then, q is semantically equivalent to p, i.e., both requests specify the same output set [10].*

Proof: We will prove this theorem by double inclusion.

$\mathbf{p} \subset \mathbf{q}$: Let (e_i,e_j) be a complex event ce detected by p. Then, by definition, ce satisfies C_p. As shown in the proof of Theorem 2.1., $[T_k]_{ts_b}^{ts_e} = R_k$, hence, ce also satisfies C_q and, thus, is a valid result tuple in q. □

$\mathbf{q} \subset \mathbf{p}$: Let (r_n,r_m) be an output tuple t of q. Then, by definition, t satisfies C_q. As shown in the proof of Theorem 1, $R_k = [T_k]_{ts_b}^{ts_e}$, hence, t also satisfies C_p and, thus, is a valid match of p. □

Furthermore, even tough the mapping of an *NSEQ* pattern (Listing 1) is a ternary operator, it does not differ from the other operator mappings. In particular, it translation is a Theta Join for the sequence and an additional subquery in its WHERE clause specifying the absence condition on the third stream as shown in Listing 2 .

Listing 1: NSEQ pattern.

```
PATTERN SEQ(T1e1, NOT(T2e2),T3e3)
WHERE <predicates>
WITHIN W
```

Listing 2: NSEQ query.

```
SELECT *
FROM Stream T1, Stream T3
WHERE T1.ts < T3.ts ∧ <predicates> ∧
    NOT EXISTS (SELECT *
                FROM Stream T2
                WHERE T1.ts < T2.ts ∧
                      T2.ts < T3.ts)
Window [Range W, s]
```

Leverage Aggregations for Iterations (O3). Our mapping of the iteration towards Theta Joins does not support the unbounded Kleene+ operation, which require one or more repetitions of event occurrences of one type. To overcome this limitation [13], we can leverage ASP aggregations. In particular, we first apply a window aggregation that returns the count n of relevant events of T in the window W. Afterward, we compare n with the user-defined m which is $m = 1$ for the traditional Kleene+ but variations allow $m \in \mathbb{N}^+$. If $n \geq m$, the pattern is fulfilled under the SP Stam and Stnm.

Theorem 3. *Let us consider an unbounded iteration pattern p, specified by the event type T, the minimum number of event occurrences m and a time window W. Additionally, let us consider a query q specified for the relations R which return in case $n \geq m$. Then, q determines that the output of p is not empty, i.e., if p is fulfilled or not for the time window W.*

Proof: By definition, the count aggregation counts the number of rows in R. As shown in the proof of Theorem 1, $R_k = [T_k]_{ts_b}^{ts_e}$, hence, n is also the number of events in W. Then, by comparing n and m, q determines that at least m events occurred in W and thus, p is fulfilled. □

However, we denote O3 as approximate because aggregations return one tuple of the same schema as the input stream per window instead of multiple tuples with the composition of events as the iteration operator. To clarify the differences, let us consider $n = 3$, i.e., e_1,e_2 and e_3, and $m = 1$. Q return a single tuple $e_3(a_1,...,a_n,n)$ containing all attributes of T and the count value n. In contrast, p returns seven tuples as result set, i.e., (e_1),(e_2), (e_3), (e_1,e_2), (e_1,e_2,e_3), (e_1,e_3), and (e_2,e_3), i.e., all possible combinations of events occurring in W. Our approximation of the iteration operator is essential powerful for patterns with threshold filters as optimized standard aggregations of the ASP can be used. Other conditions requires a user defined aggregation function with usually some decrease of performance. However, both cases prevent the combinatorial explosion of partial matches, require only a single pass over the data, and are thus less memory and compute intensive [11] .

2 PROOF OF INTER-WINDOW SEMANTICS

In order to prove the correctness of our proposed inter-window semantics, let us consider a pattern p, which is composed of operators OP and a time window of length W. The semantics of different operators OP, i.e., sequence, conjunction, disjunction, iteration, and negation, are formally defined by our derived definitions and applied to each finite substream $S_k \in S$. In particular, $S_k = [S]_{ts_{e_k}}^{ts_{b_k}}$, where $[ts_{b_k},ts_{b_e})$ describes an arbitrary time interval for which is true that $ts_{e_k} - ts_{b_k} = W$. Finally, a complex event $ce(e_1,...,e_n)$ is a pattern match M, i.e., the composition of all participating events e_j. Thus, ce is a valid match of p if $max(e_j.ts \in ce) - min(e_j.ts \in ce) < W$. Furthermore, ce is detected in $[S]_{ts_{e_k}}^{ts_{b_k}}$, if $e_1,e_n \in S_k$ and, thus, $e_1.ts,e_n.ts \in [ts_{b_k},ts_{e_k})$.

The window operator contains the semantics of how subsequent substreams S_{k+m} are created. We propose two different window types for our mapping, i.e.. time-based sliding windows used for Sliding Window Joins and a content-based window frame [12] for the Interval Join (O1):

Sliding Window Joins. For our default mappings, we use time-based sliding windows, which, in addition to W (a fixed window length), define a slide size s that declares when a subsequent window starts. Thus, sliding windows create the following sequence of potentially overlapping substreams: $S_{k+m} = [S]^{ts_{b_{k+m}}}_{ts_{e_{k+m}}}$, where $ts_{k+m} = ts_k + s*m$, respectively ($m \in \mathbb{N}$). Furthermore, following our mapping directives, our mapping requires sliding per tuple. Thus, $s = 1$ for slide-by-tuple windows or is smaller or equal to the frequency of the stream with the highest arrival rate in p. Finally, for simplification, we consider that the events $e_j \in ce$ are in temporal order and $e_1.ts$ is the smallest and $e_n.ts$ the largest timestamp in ce ($ts \in \mathbb{N}$).

Theorem 4. *Let us consider the pattern p is applied to the stream S and yields a complex event ce. Then, there exists at least one substream $S_k = [S]^{ts_{b_k}}_{ts_{e_k}}$ such that $e_1, e_n \in S_k$ when Slidding Window Joins are applied.*

Proof: As e_1 and e_n are two consecutive events for which it is true that $e_n.ts - e_1.ts < W$, we define $e_n.ts = e_1.ts + q$, where $0 \leq q < W$ and show that for both corner cases, i.e., $q = 0$ and $q = W - 1$ a substream S_k is created by the sliding window operator.

Case 1 $q = 0$: If $e_1 \in S_k \Rightarrow e_1.ts \in [ts_{b_k}, ts_{e_k})$
$\Rightarrow e_n.ts = e_1.ts + 0 = e_1.ts \Rightarrow e_n.ts \in [ts_{b_k}, ts_{e_k})$
$\Rightarrow e_1, e_n \in S_k$ □

Case 2 $q = W - 1$: For case 2, the complex event ce is only detected in S_k, if $e_1.ts = ts_{b_k}$ as otherwise $e_n.ts = e_1.ts + x + W - 1 > ts_{e_k}$. To this end, first, let us consider $e_1 \in S_k \wedge e_1.ts = ts_{b_k}$.

Case 2.1. $q = W - 1 \wedge e_1.ts = ts_{b_k}$: If $e_1.ts = ts_{b_k} \Rightarrow e_1 \in S_k$
$\Rightarrow e_n.ts = e_1.ts + W - 1 = ts_{b_k} + W - 1 = ts_{e_k} - 1 \Rightarrow e_n.ts \in [ts_{b_k}, ts_{e_k})$
$\Rightarrow e_n \in S_k$ □

Second, let us consider that e_1 occurs at $ts_{e_k} - 1$, i.e., e_1 is the last event considered in S_k. It follows, that $e_n.ts \notin S_k$, because $ts_{e_k} + W - 1 > ts_{e_k} - 1$. Thus, we need to ensure that there exists a S_{k+m} in which e_1 and e_n occur, i.e., $\exists S_{k+m}$ so that $e_1.ts = ts_{e_k} - 1 \wedge e_1.ts = ts_{b_{k+m}}$.

Case 2.2. $q = W - 1 \wedge e_1.ts = ts_{e_k} - 1 \wedge e_1.ts = ts_{b_{k+m}}$:
$ts_{e_k} - 1 = ts_{b_{k+m}}$
$\Rightarrow ts_{e_k} - 1 = ts_{b_k} + s*m$
$\Rightarrow ts_{e_k} - ts_{b_k} - 1 = m$
$\Rightarrow W - 1 = m$

Thus, the match ce is detected $W - 1$ windows after e_1 occurs and e_n occurs the first time. □

Interval Joins. For our optimization O1, we use the Interval Join (available in Flink [1]) that creates a content-based frame to compose tuple from two streams. Particularly, this join composes two events $e_1 \in T_1$ and $e_2 \in T_2$ given a key condition and a window condition based of an occurring event $e_1 \in T_1$: $e_2.ts \in (e_1.ts + LB, e_1.ts + UB)$, with lower bound (LB) and upper bound (UB) defined as time measurement [1]. The bounds only depend on the window size W, thus circumventing the setting of a slide size. In particular, for the conjunction, the bounds are defined as follows $(e_1.ts - W, e_1.ts + W)$ as the order of events is irrelevant. All other operators use $(e_1.ts + 0, e_1.ts + W)$.

Theorem 5. *Let us consider an elementary pattern p which is applied to the stream S and yields a complex event ce. Then, there exists at least one substream $S_k = [S]^{ts_{b_k}}_{ts_{e_k}}$ such that $e_1, e_n \in S_k$ when Interval Joins are applied.*

Proof: With the occurrence of e_1 the Interval Join creates a window based on the timestamp $e_1.ts$. We consider two cases, i.e., p is a conjunction pattern, and p is a sequence or iteration pattern.

Case 1 p = conjunction pattern: As the order of events is irrelevant, e_n may occur before or after e_1. However, in order to form a match, it is true that $|e_n.ts - e_1.ts| < W$. To this end, the following time window is created: $(e_1.ts - W, e_1.ts + W)$.
$\Rightarrow S_k = [S]^{e_1.ts - W}_{e_1.ts + W}$
Furthermore, we define $e_n.ts = e_1.ts + q$, where $-W < q < W$.
$\Rightarrow e_n.ts \in (e_1.ts - W, e_1.ts + W)$ and $e_n \in S_k$ □

Case 2 p = sequence (iteration) pattern: If p = sequence pattern the following time window is created: $(e_1.ts + 0, e_1.ts + W)$.
$\Rightarrow S_k = [S]^{e_1.ts}_{e_1.ts + W}$
As e_1 and e_n are two consecutive events for which it is true that $e_n.ts - e_1.ts < W$, we define $e_n.ts = e_1.ts + q$, where $0 < q < W$.
$\Rightarrow e_n.ts \in (e_1.ts, e_1.ts + W)$ and $e_n \in S_k$ □

To this end, the Interval Join is an alternative windowing solution that prevents the creation of duplicates as all relevant $e_2 \in T_2$ are assigned to the unique window of $e_1 \in T_1$ as well as the setting of stream-depend parameters in contrast to Sliding Window Joins.

REFERENCES

[1] 2022. Apache Flink. Accessed Sept. 2023: https://flink.apache.org.
[2] Arvind Arasu, Shivnath Babu, et al. 2003. CQL: A language for continuous queries over streams and relations. In *International Workshop on Database Programming Languages*. Springer.
[3] Iliano Cervesato and Angelo Montanari. 2000. A Calculus of Macro-Events: Progress Report. In *Seventh International Workshop on Temporal Representation and Reasoning, TIME 2000, Nova Scotia, Canada, 2000*. IEEE Computer Society.
[4] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, and S.-K. Kim. 1994. Composite Events for Active Databases: Semantics, Contexts and Detection. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo (Eds.). Morgan Kaufmann.
[5] E. F. Codd. 1972. Relational Completeness of Data Base Sublanguages. *Research Report / RJ / IBM / San Jose, California* (1972).
[6] E. F. Codd. 1983. A Relational Model of Data for Large Shared Data Banks (Reprint). *Commun. ACM* (1983).
[7] Antony Galton and Juan Carlos Augusto. 2002. Two Approaches to Event Definition. In *Database and Expert Systems Applications, 13th International Conference, DEXA 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings (Lecture Notes in Computer Science)*, Abdelkader Hameurlain, Rosine Cicchetti, and Roland Traunmüller (Eds.). Springer.
[8] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* (2020).
[9] Ilya Kolchinsky and Assaf Schuster. 2018. Join query optimization techniques for complex event processing applications. *Proc. VLDB Endow.* (2018).
[10] Mauro Negri, Giuseppe Pelagatti, and Licia Sbattella. 1991. Formal Semantics of SQL Queries. *ACM Trans. Database Syst.* (1991).
[11] Olga Poppe, Chuan Lei, Salah Ahmed, and Elke A. Rundensteiner. 2017. Complete Event Trend Detection in High-Rate Event Streams. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 109–124.
[12] Juliane Verwiebe, Philipp M. Grulich, Jonas Traub, and Volker Markl. 2023. Survey of window types for aggregation in stream processing systems. *VLDB J.* 32, 5 (2023), 985–1011. https://doi.org/10.1007/s00778-022-00778-6
[13] Haopeng Zhang, Yanlei Diao, and Neil Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM.
[14] Shuhao Zhang, Yancan Mao, et al. 2021. Parallelizing Intra-Window Join on Multicores: An Experimental Study. In *SIGMOD*. ACM.