Formal Proof of CEP2ASP - Operator Mapping

Anonymous Author(s)

For our mapping approach proposed in *Bridging the Gap: Complex Event Processing on Stream Processing Engines*, we first derive formal definitions for Simple Event Algebra [7] operators by modifying well-defined operators of the active database representative Snoop [3]. 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. [4, 5]. In the remainder, we elaborate on the correctness of our mapping approach, i.e., detecting all pattern matches using our mapping. 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 [3]:

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 $false$ [3].

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

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 [2, 6]. 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 = $\{e_1,...,e_n\}$, where for each event e_i it is true that $e_i.ts \in [ts_b,ts_e)$. We modify Eq. 1 as follows:

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

Modification 2: To fulfill the closure properties of SEA, we further need to modify the output of the function $[P]_{ts_b}^{ts_e}$ (Eq. 2). In particular, the function either returns the set of events e_i satisfying all constraints or an empty set and no Boolean value.

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

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 [1]. 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 [1, 9]. We prove this theorem in Sec. 2. For the correctness of our modifications, i.e., detecting all complex events $ce(e_1,..,e_n)$, we also need to ensure that no complex event 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 Sec 1.

1 PROOF OF INTER-WINDOW SEMANTICS

In order to prove the inter-window semantics of our operator mapping, 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 $ts_{e_k} - ts_{b_k} = W$. Furthermore, the window operator contains the semantics of how subsequent substreams S_{k+m} are created. For our mapping, 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: $S_{k+m} = [S]_{ts_{e_{k+m}}}^{ts_{b_{k+m}}}$, 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, a complex event $ce(e_1,...,e_n)$ is a pattern match, i.e., the composition of all participating events e_i. For simplification, we consider that the events $e_i \in ce$ are in temporal order and $e_1.ts$ is the smallest and $e_n.ts$ the largest timestamp in ce ($ts \in \mathbb{N}$). Thus, ce is a valid match of p if $e_n.ts - e_1.ts < W$. Furthermore, ce is detected, if $e_i,e_n \in S_k$ and thus, $e_1.ts,e_n.ts \in [ts_{b_k},ts_{e_k})$.

Theorem 1.1. *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* $S_k = [S]_{ts_{e_k}}^{ts_{b_k}}$ *such that* $e_i,e_n \in S_k$.

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. □

2 PROOF OF INTRA-WINDOW SEMANTICS

In this section, we provide formal proof for the intra-window semantics of the conjunction operator, which 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:

$$(T_1 \wedge T_2) = \{(e_{i,T_1}, e_{j,T_2}) \mid e_{i,T_1} \in T_1 \ \wedge \ e_{j,T_1} \in T_2\} \tag{4}$$

Furthermore, by definition (e_{i,T_1}, e_{j,T_2}) 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. [4] as follows:

$$R_1 \times R_2 = \{(r_{n,R_1}, r_{m,R_2}) \mid r_{n,R_1} \in R_1 \ \wedge \ r_{m,R_2} \in R_2\} \tag{5}$$

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

Theorem 2.1. *Let us consider a pure Cartesian product query q specified by the relations R_1 and R_2. Additionally, let us consider a pure 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 [8].*

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 $e_{l,T_k} \in S$ corresponds to a tuple $r_{h,R_k} \in R_k$. 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 $\forall e_{l,T_k} \in S(l \in [ts_b, ts_e))$ and $W = ts_e - ts_b$.

p ⊂ q: Let (e_{i,T_1}, e_{j,T_2}) be a complex event ce detected by p. Then, by definition, $e_{i,T_1} \in T_1$, $e_{j,T_2} \in T_2$, $i,j \in [ts_b, ts_e)$ and $max(i,j) - min(i,j) < W$. As T_1 and T_2 correspond to the relations R_1 and R_2, $\exists r_{n,R_1} \in R_1$ that resembles e_{i,T_1} and $\exists r_{m,R_2} \in R_2$ that resembles e_{j,T_2}. 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 $r_{h,R_k} \in R_k$ resembles the timestamp of its corresponding event e_{l,T_k}. It follows that for each pair, e_{l,T_k} and t_{h,R_k}, 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. □

q ⊂ p: Let (r_{n,R_1}, r_{m,R_2}) be an output tuple t of q. Then, by definition, $r_{n,R_1} \in R_1$ and $r_{m,R_2} \in R_2$ and $0 \leq n,m < |R_i|$. As T_1 and T_2 correspond to the relations R_1 and R_2, $\exists e_{i,T_1} \in T_1$ that resembles r_{n,R_1} and $\exists e_{j,T_2} \in T_2$ that resembles r_{m,R_2}. Let us define the time window as $W = |R_k|$ and $r(T_i) = \frac{|R_k|}{W}$. Thus, $|[T_k]_{ts_b}^{ts_e}| = W \cdot r(T_i) = |R_k|$. Furthermore, every timestamp $l \in [ts_b, ts_e)$ of an event e_{l,T_k} resembles to the index h of its corresponding $r_{h,R_k} \in R_k$. It follows that for each corresponding pair e_{l,T_k} and t_{h,R_k}, 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. □

REFERENCES

[1] 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.
[2] Iliano Cervesato and Angelo Montanari. 2000. A Calculus of Macro-Events: Progress Report. IEEE Computer Society.
[3] Sharma Chakravarthy, V. Krishnaprasad, Eman Anwar, et al. 1994. Composite Events for Active Databases: Semantics, Contexts and Detection. In *VLDB'94*.
[4] E. F. Codd. 1972. Relational Completeness of Data Base Sublanguages. *Research Report / RJ / IBM / San Jose, California* (1972).
[5] Edgar Frank Codd. 1983. A relational model of data for large shared data banks. *Commun. ACM* (1983).
[6] Antony Galton and Juan Carlos Augusto. 2002. Two Approaches to Event Definition. Springer.
[7] Nikos Giatrakos, Elias Alevizos, et al. 2020. Complex event recognition in the Big Data era: a survey. *VLDB J.* (2020).
[8] Mauro Negri, Giuseppe Pelagatti, et al. 1991. Formal Semantics of SQL Queries. *ACM Trans. Database Syst.* (1991).
[9] Shuhao Zhang, Yancan Mao, et al. 2021. Parallelizing Intra-Window Join on Multicores: An Experimental Study. In *SIGMOD*. ACM.