Sequenced Event Set Pattern Matching

Bruno Cadonna
Free University of Bozen-Bolzano, Italy
Faculty of Computer Science
cadonna@inf.unibz.it

Johann Gamper
Free University of Bozen-Bolzano, Italy
Faculty of Computer Science
gamper@inf.unibz.it

Michael H. Böhlen
University of Zurich, Switzerland
Department of Informatics
boehlen@ifi.uzh.ch

ABSTRACT

Event pattern matching is a query technique where a sequence of input events is matched against a complex pattern that specifies constraints on extent, order, values, and quantification of matching events. The increasing importance of such query techniques is underpinned by a significant amount of research work, the availability of commercial products, and by a recent proposal to extend SQL for event pattern matching. The proposed SQL extension includes an operator PERMUTE, which allows to express patterns that match any permutation of a set of events. No implementation of this operator is known to the authors.

In this paper, we study the sequenced event set pattern matching problem, which is the problem of matching a sequence of input events against a complex pattern that specifies a sequence of sets of events rather than a sequence of single events. Similar to the PERMUTE operator, events that match with a set specified in the pattern can occur in any permutation, whereas events that match with different sets have to be strictly consecutive, following the order of the sets in the pattern specification. We formally define the problem of sequenced event set pattern matching, propose an automaton-based evaluation algorithm, and provide a detailed analysis of its runtime complexity. An empirical evaluation with real-world data shows that our algorithm outperforms a brute force approach that uses existing techniques to solve the sequenced event set pattern matching problem, and it validates the results from our complexity analysis.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithm and Problems—*Pattern matching*; H.2.4 [**Database Management Systems**]: Systems—*Query Processing*

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

General Terms

Algorithms

Keywords

Event pattern matching, Automaton, Set, Permutation

1. INTRODUCTION

Event pattern matching is a query technique where a sequence of input events is matched against a complex pattern that specifies constraints on extent, order, values, and/or quantification of matching events. Due to the wide applicability of event pattern matching in different application domains such as financial services [3, 11, 23], click stream analysis [23], RFID-based tracking and monitoring [3, 13], RSS feed monitoring [11], and health services [19], a significant amount of research has recently been conducted in this field.

The increasing importance of event pattern matching in practical applications is underpinned by the availability of commercial products [10, 25] and by a recent SQL change proposal to extend SQL for pattern matching over sequences of tuples [27]. In the latter, a PERMUTE operator is proposed to retrieve sequences of input tuples that match any *permutation* of a set of variables specified in the pattern. Multiple PERMUTE operators in series permit to match *sequences of sets of tuples*. Only a subset of the pattern matching operators in the SQL change proposal have been implemented [13, 23], and no implementation of the PERMUTE operator is known.

In this paper, we study the problem of *sequenced event set (SES) pattern matching*, which is the problem of matching a sequence of input events against a complex pattern that specifies a sequence of sets of events (SES pattern) rather than a sequence of single events. While the order of the input events that match with the same set is irrelevant (that is, any permutation of the input events is matched), the order of the input events that match with distinct sets must correspond to the order of the sets in the pattern. Furthermore, SES patterns allow the specification of a maximal time interval within which all events that match with a pattern must occur.

Example 1. As a running example throughout the paper, we consider the analysis of chemotherapy data. A chemotherapy is a treatment for patients suffering from cancer and consists of a sequence of events, such as the administration of medications and laboratory examinations according to a well-defined protocol. Figure 1 shows a sample

Event

	ID	*L*	*V*	*U*	*T*
e_1	1	C	1672.5	mg	9 am 3 Jul
e_2	1	B	0	WHO-Tox	10 am 3 Jul
e_3	1	D	84	mgl	11 am 3 Jul
e_4	1	P	111.5	mg	9 am 4 Jul
e_5	2	B	0	WHO-Tox	9 am 5 Jul
e_6	2	P	88	mg	10 am 5 Jul
e_7	2	D	84	mgl	11 am 5 Jul
e_8	2	C	1320	mg	9 am 6 Jul
e_9	1	P	111.5	mg	10 am 6 Jul
e_{10}	2	P	88	mg	11 am 6 Jul
e_{11}	2	P	88	mg	9 am 7 Jul
e_{12}	1	B	1	WHO-Tox	9 am 12 Jul
e_{13}	2	B	1	WHO-Tox	9 am 13 Jul
e_{14}	2	B	0	WHO-Tox	9 am 14 Jul

Figure 1: Events of Chemotherapy Treatments.

relation, `Event`, that records such events. The attributes represent patient ID (*ID*), event type (*L*), value (*V*) with measurement unit (*U*), and occurrence time (*T*) of an event, respectively. For instance, event e_1 represents the administration of 1672.5 mg of Ciclofosfamide to patient 1 at 9 am on 3 July. To study the effect of medications on the blood count, physicians might issue the following query:

> Q1: *For each patient, find the sets of events that match one administration of Ciclofosfamide (C), one or more administrations of Prednisone (P), and one administration of Doxorubicina (D) in any order, followed by a single blood count (B) measurement, and all events occur within eleven days?*

Several subsets of events in relation `Event` match the pattern in Query Q1, e.g., $\{e_1, e_3, e_4, e_{12}\}$, $\{e_1, e_3, e_4, e_{12}\}$, and $\{e_1, e_3, e_4, e_9, e_{12}\}$ for patient 1. In this paper, we are interested in those results which contain (1) the earliest possible matching events and (2) the maximal number of matching events. The first restriction corresponds to the skip-till-next-match event selection strategy [3], and the second one to the MAXIMAL mode with greedy quantifier [27]. Thus, the intended results for Query Q1 are $\{e_1, e_3, e_4, e_9, e_{12}\}$ for patient 1, and $\{e_6, e_7, e_8, e_{10}, e_{11}, e_{13}\}$ for patient 2. Notice that the blood count measurements e_2 and e_5 are ignored because they are carried out during (and not after) the administrations of C, P, and D.

The need to (partially) ignore the order of events and to consider all permutations of a set of events occurs if variations in the order of the input events exist naturally in the application domain but should be ignored for data analysis tasks. For instance, the administration of 'C' and 'P' occurs in a different order for patient 1 and 2, yet both patients should be considered for Query Q1.

Existing solutions for event pattern matching cannot solve the SES pattern matching problem. They either can only express sequences of single events or have limitations to express the full set of SES patterns.

In this paper, we propose a solution for SES pattern matching. The technical contributions can be summarized as follows:

- We introduce and formally define SES pattern matching, which allows to express patterns that consist of sequences of sets of events and match all permutations of individual sets, while maintaining the order among sets.
- We propose an automaton-based algorithm to evaluate SES pattern matching.
- We conduct a detailed complexity analysis of the evaluation algorithm, which considers different kinds of non-determinism and provides upper bounds for the runtime complexity.
- We report the results of an empirical evaluation using real-world data. The study validates the complexity analysis and shows that our solution clearly outperforms a brute force approach that is based on existing techniques.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we introduce and define SES pattern matching. In Section 4 we present an automaton-based evaluation algorithm and analyze its complexity. Section 5 reports the results of an empirical evaluation study. Section 6 concludes the paper and points to future work.

2. RELATED WORK

The SQL change proposal [27] describes an extension to SQL for pattern matching in sequences of tuples. The SQL extension is motivated by various use cases for event processing, such as security, financial, fraud detection, and RFID applications, where a tuple represents an event. The SQL extension defines the PERMUTE operator and its EXPAND FACTORS variant, which allow to express a set of events. Multiple PERMUTE EXPAND FACTORS operators in succession permit to express SES patterns that do not contain any Kleene plus quantifier. The SQL extension focuses on the specification of the language and provides no implementation of the various operators.

The DejaVu system [13] aims to implement pattern matching in a RDBMS (MySQL) to find matches of patterns in live and archived data streams. DejaVu implements a subset of the SQL extension [27] using finite state automata; the PERMUTE operator is not included. Hence, SES patterns cannot be expressed in a simple way. Multiple patterns are required, one for each possible match. The number of possible matches grows exponentially with the cardinality of the sets of events specified in the SES pattern (cf. Section 5.2). Clearly, such a solution becomes quickly cumbersome and inefficient for all but very small sets of events.

SQL-TS [23] extends SQL to process complex sequential patterns in database systems. For an efficient evaluation of sequential pattern queries, the Knuth-Morris-Pratt string matching algorithm is adapted for event pattern matching. SQL-TS does not implement the PERMUTE operator, and SES patterns can only be expressed with multiple patterns similar to DejaVu [13], thus suffering from the same drawbacks.

ZStream [20] is a cost-based query processor for matching sequential patterns enriched with the operators sequence, conjunction, disjunction, negation, and Kleene closure. ZStream uses tree-based query plans, where each of the operators is represented as a variant of the join operator, together with a cost model to find the most efficient query

plan. Similar to a SES pattern, the conjunction operator allows to express that two events occur within a specified time window, and their order does not matter. However, when a conjunction is combined with a Kleene plus quantifier, all occurrences of the quantified event must be consecutive without any other matching event in between.

The pattern specification language SASE+ [18, 26] uses the nondeterministic finite state automaton NFA^b [3] to retrieve matches from a stream of events. SASE+ and NFA^b support the Kleene plus quantifier and four different matching semantics. NFA^b adopts various optimization techniques to reduce space and runtime requirements for the evaluation of pattern queries. The specification of SES patterns is similar to DejaVu [13] and suffers from the same drawbacks.

Constraint-aware complex event processing (C-CEP) [14] focuses on a query unsatisfiability checking technique that detects at runtime optimal points for terminating the evaluation of partial query matches that will never be satisfied. The implemented C-CEP system employs an automaton that allows to specify only SES patterns with equality relationships between events. SES patterns with more general relationships between events are not addressed.

The publish/subscribe system Cayuga [11, 12] reads an incoming stream of events (publications) and selects event sequences according to queries that express user's interests (subscriptions). Cayuga uses an algebra that is inspired by regular expressions to express queries over event streams. For the evaluation, a query is translated into a corresponding nondeterministic finite state automaton. Various indexing techniques are employed to optimize the query evaluation. Cayuga is unable to express all SES patterns because it can only specify relationships between consecutive events and not between arbitrary events in a set of events.

Other publish/subscribe systems [4, 15] have limited expressiveness, since they are unable to specify a temporal order and other relationships among (attributes of) events.

SEQ [24] is a sequence database with an SQL-like query language, called SEQUIN. SEQUIN allows to select event sequences with a combination of selection, join, and aggregation operations. With the growing length of the retrieved event sequences, the queries become more and more intricate due to the increasing number of join and select operations that are required to specify such sequences. SEQUIN is unable to express SES patterns with a Kleene plus quantifier since the number of join operations must be known at query time.

Aurora [2, 7], Borealis [1], STREAM [22, 5], and TelegraphCQ [9] are data stream management systems that mainly focus on efficient resource management and load balancing. Their query languages apply a combination of selection, join, and aggregation operations. As with SEQUIN, such queries become more and more complicated with the growing length of the desired event sequence and the specification of the sets of events, and they are unable to express SES patterns with Kleene plus quantifiers.

Event Analyzer [19] is a data warehouse component to analyze event sequences. Its pattern specification language is able to express patterns that match simultaneous events. Simultaneous events can be interpreted as a special case of SES patterns because no total order can be imposed on the events, and hence, their order should be ignored in a query. However, the pattern specification language is not able to express more general SES patterns.

A theoretical study about the CEDR pattern specification language [6] focuses mainly on the management of stream imperfections. CEDR does not include any operator to match sets of events.

Active databases react to events that originate from operations within the system such as data manipulations. Event languages of active databases allow to express compositions of events to which the system should react. Various event languages have been proposed [8, 16, 17, 21, 28]. They are unable to express general SES patterns because they are either very limited or unable to express relationships between attributes of events.

3. PROBLEM DEFINITION

In this section we introduce and formally define sequenced event set pattern matching.

3.1 Event Model

An *event* is represented as a tuple with schema $\mathbf{E} = (A_1, \ldots, A_l, T)$, where $A_1, \ldots, A_l$ are non-temporal attributes and T is a temporal attribute that represents the occurrence time of the event. For T, we assume a discrete and ordered time domain, $\mathbb{T}$, such as calendar days and hours as in our running example.

An *event relation*, E, is a set of events, and we assume the timestamp attribute, T, defines a total order among events. Relation `Event` in Figure 1 is an example of an event relation.

3.2 SES Pattern Matching

An *event set pattern* is a set of event variables, $V_i = \{v_1, \ldots, v_n\}$, $n \geq 1$. By default, an event variable, $v \in V_i$, is bound to one input event; we call it *singleton variable*. An event variable that is followed by a Kleene plus quantifier, v^+, is a *group variable* and can be bound to one or more events. We use lowercase letters for event variables, optionally followed by a Kleene plus quantifier as superscript, e.g., v, v^+.

Definition 1. (*SES Pattern*) A *sequenced event set pattern*, P, is defined as a triple

$$P = (\langle V_1, \ldots, V_m \rangle, \Theta, \tau),$$

where

- $\langle V_1, \ldots, V_m \rangle$, $m \geq 1$, is a sequence of event set patterns and $V_i \cap V_j = \emptyset$ for $1 \leq i, j \leq m, i \neq j$,
- $\Theta = \{\theta_1, \ldots, \theta_n\}$, $n \geq 0$, is a set of conditions over the event variables in the event set patterns, and
- τ is a duration.

$V_1, \ldots, V_m$ are sets of event variables. Throughout the paper, we use V to refer to the set of all event variables in a SES pattern, i.e., $V = V_1 \cup \cdots \cup V_m$. Θ is a set of conditions over the event variables that express constraints on the values of individual event attributes, which must be satisfied by the matching input events. A condition, $\theta \in \Theta$, has the form $v.A\ \phi\ v'.A'$ or $v.A\ \phi\ C$, where v and v' are (singleton or group) event variables, A and A' are event attributes, C is a constant, and $\phi \in \{=, <, \leq, >, \geq\}$ is a comparison operator. τ is the maximal allowed duration between the (chronologically) first and the last event that match P.

Example 2. Consider Query Q1, which can be formulated as a SES pattern

$$P = (\langle\{c, p^+, d\}, \{b\}\rangle, \Theta, 264),$$

which is composed of two event set patterns. The first event set pattern, $V_1 = \{c, p^+, d\}$, contains two singleton variables (c and d), each of which can be bound to one event, and one group variable (p^+), which can be bound to one or more events. The second event set pattern, $V_2 = \{b\}$, consists of one singleton variable. The conditions over the event variables are given as

$$\begin{aligned}\Theta = \{&\theta_1 \equiv c.L = \text{'C'},\ \ \theta_2 \equiv d.L = \text{'D'},\\ &\theta_3 \equiv p^+.L = \text{'P'},\ \ \theta_4 \equiv b.L = \text{'B'},\\ &\theta_5 \equiv c.ID = p^+.ID,\ \ \theta_6 \equiv c.ID = d.ID,\\ &\theta_7 \equiv d.ID = b.ID\}\end{aligned}$$

and constrain the events that can be matched with the event variables: c matches one administration of Ciclofosfamide (θ_1), p^+ matches one or more administrations of Prednisone (θ_3), d matches one administration of Doxorubicina (θ_2), and b matches one blood count measurement (θ_4); the conditions θ_5, θ_6, and θ_7 require that all matched events refer to the same patient. The last parameter of the pattern specifies a maximal duration of 264 hours (eleven days) between the (chronologically) first event that matches with P and the last event.

To formalize the matching of a SES pattern P and an event relation E, we adopt the concept of a substitution. A *substitution*, $\gamma = \{v_1/e_1, \ldots, v_n/e_n\}$, is a finite set of pairs of event variables and events. A pair v/e is a *binding* for variable v. A substitution contains exactly one binding for each singleton variable in P, but one or more bindings for each group variable in P. All event variables and events in a substitution are contained in P. All events in a substitution are contained in E and are distinct.

Let $\gamma = \{v_1/e_1, \ldots, v_n/e_n\}$ be a substitution, and let θ be a condition of a SES pattern. $\theta\gamma$ denotes the *instantiation* of θ by γ, and it is obtained from θ by simultaneously replacing all occurrences of event variables v_i by the corresponding events e_i. A substitution, γ, that contains multiple bindings for a group variable can be *decomposed* in a set of substitutions $\{\gamma_1, \ldots, \gamma_m\}$, where each γ_i contains one binding for each event variable in γ and there exists a γ_i for each combination of bindings with distinct event variables in γ. The instantiation $\theta\gamma$ gives a set of conditions $\{\theta\gamma_1, \ldots, \theta\gamma_m\}$. For the conditions in a SES pattern, $\Theta = \{\theta_1, \ldots, \theta_n\}$, and a substitution γ, we have $\Theta\gamma = \theta_1\gamma \cup \cdots \cup \theta_n\gamma$.

Example 3. Let $\Theta = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7\}$ be the conditions from the SES pattern in Example 2 and $\gamma = \{c/\mathtt{e}_1, d/\mathtt{e}_3, p^+/\mathtt{e}_4, p^+/\mathtt{e}_9, b/\mathtt{e}_{12}\}$ be a substitution, which contains two bindings for the group variable p^+. Then γ can be decomposed into two substitutions, $\{\gamma_1, \gamma_2\}$, where $\gamma_1 = \{c/\mathtt{e}_1, d/\mathtt{e}_3, p^+/\mathtt{e}_4, b/\mathtt{e}_{12}\}$ and $\gamma_2 = \{c/\mathtt{e}_1, d/\mathtt{e}_3, p^+/\mathtt{e}_9, b/\mathtt{e}_{12}\}$. The instantiation of Θ with γ gives $\Theta\gamma = \{\theta_1\gamma_1, \ldots, \theta_7\gamma_1, \theta_1\gamma_2, \ldots, \theta_7\gamma_2\} = \{\mathtt{e}_1.L = \text{'C'}, \mathtt{e}_3.L = \text{'D'}, \mathtt{e}_4.L = \text{'P'}, \mathtt{e}_{12}.L = \text{'B'}, \mathtt{e}_1.ID = \mathtt{e}_4.ID, \mathtt{e}_1.ID = \mathtt{e}_3.ID, \mathtt{e}_3.ID = \mathtt{e}_{12}.ID, \mathtt{e}_1.L = \text{'C'}, \ldots, \mathtt{e}_9.L = \text{'P'}, \ldots\}$.

To define a matching substitution, we use the following auxiliary function. Let $\min_T(\gamma)$ be a function that returns the binding in substitution γ with the earliest occurrence time, i.e., $\min_T(\gamma) = v/e$ such that $v/e \in \gamma \wedge \nexists v'/e' \in \gamma(e'.T < e.T)$.

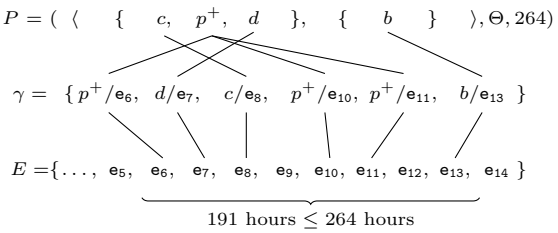


Figure 2: Match of Pattern and Events.

Definition 2. (*Matching Substitution*) Let E be an event relation and $P = (\langle V_1, \ldots, V_m\rangle, \Theta, \tau)$ be a SES pattern. A substitution $\gamma = \{v_1/e_1, \ldots, v_n/e_n\}$ is a *matching substitution* iff the following five conditions hold:

$$\forall\theta_i\gamma_j \in \Theta\gamma\ (\ \theta_i\gamma_j \text{ is satisfied }) \tag{1}$$

$$\forall v/e, v'/e' \in \gamma\ (\ v \in V_i \wedge v' \in V_{i+1} \rightarrow e.T < e'.T\ ) \tag{2}$$

$$\forall v/e, v'/e' \in \gamma\ (\ |e.T - e'.T| \leq \tau\ ) \tag{3}$$

Let Γ be the set of all substitutions (of event variables in P by events in E) that satisfy conditions 1–3.

$$\begin{aligned}&\forall v/e, v'/e' \in \gamma\ (\ \nexists\gamma' \in \Gamma, v''/e'' \in \gamma'\ (\\ &\quad v'' = v' \wedge e.T < e''.T < e'.T \wedge v''/e'' \notin \gamma\ )\ )\end{aligned} \tag{4}$$

$$\forall\gamma' \in \Gamma\ (\ \min_T(\gamma) = \min_T(\gamma') \rightarrow \gamma \not\subset \gamma'\ ) \tag{5}$$

Condition 1 ensures that all events in a matching substitution satisfy all conditions in Θ. Condition 2 ensures that all events that match with a variable in set V_i must occur strictly before all events that match with a variable in set V_{i+1}. Notice that no order is imposed on the events that match with variables from the same event set pattern, hence any permutation is matched. Condition 3 constrains all events in a matching substitution to occur within a time interval τ.

Condition 4 states that after an event e is matched, all events following e in the event relation are ignored until the next event matching with P is encountered. This corresponds to skip-till-next-match event selection strategy [3]. Condition 5 states that a matching substitution is not contained in any other substitution that satisfies conditions 1–3 and starts with the same event. This corresponds to MAXIMAL mode with greedy quantifier [27].

Example 4. Figure 2 illustrates a matching substitution for patient 2. All conditions in $\Theta\gamma$ are satisfied: $\mathtt{e}_8$ is a 'C' event (i.e., $\mathtt{e}_8.L = \text{'C'}$), $\mathtt{e}_6$, $\mathtt{e}_{10}$, and $\mathtt{e}_{11}$ are 'P' events, $\mathtt{e}_7$ is a 'D' event, $\mathtt{e}_{13}$ is a 'B' event, and all events refer to the same patient. The events $\mathtt{e}_6, \mathtt{e}_7, \mathtt{e}_8, \mathtt{e}_{10}, \mathtt{e}_{11}$ that match the first event set pattern occur strictly before $\mathtt{e}_{13}$ that matches the second event set pattern. The time span between the first ($\mathtt{e}_6$) and the last ($\mathtt{e}_{13}$) matching events is less than 264 hours. Substitution γ contains the earliest events possible. In contrast, substitution $\{p^+/\mathtt{e}_6, d/\mathtt{e}_7, c/\mathtt{e}_8, p^+/\mathtt{e}_{10}, p^+/\mathtt{e}_{11}, b/\mathtt{e}_{14}\}$ satisfies conditions 1–3, but violates condition 4 because it contains $\mathtt{e}_{14}$ instead of the earlier event $\mathtt{e}_{13}$. Similar, substitution $\{p^+/\mathtt{e}_6, d/\mathtt{e}_7, c/\mathtt{e}_8, p^+/\mathtt{e}_{10}, b/\mathtt{e}_{13}\}$ satisfies conditions 1–3, but violates the maximality condition 5 because

event $\mathtt{e}_{11}$ is not included, though it matches event variable p^+.

4. AUTOMATON-BASED EVALUATION

In this section we present an automaton-based algorithm for the evaluation of SES pattern queries. The algorithm first translates a query pattern into a SES automaton, which is then executed on the input relation.

4.1 Definition of SES Automaton

A SES automaton is a nondeterministic finite state automaton enriched with a match buffer, β, which during the execution of the automaton collects bindings for the event variables in the SES pattern.

Definition 3. (*SES Automaton*) Let P be a SES pattern and V be the set of all event variables in P. A *SES automaton*, N, is a five-tuple

$$N = (Q, \Delta, q_s, q_f, \tau)$$

where

- $Q = \{q_1, \ldots, q_n\}$, $q_i \subseteq V$, is a finite set of states,
- $\Delta = \{\delta_1, \ldots, \delta_m\}$ is a finite set of transitions of the form $\delta = (q, v, \Theta_\delta)$,
- $q_s \in Q$ is the start state,
- $q_f \in Q$ is the accepting state, and
- τ is a duration.

Each state in Q is defined as a subset of the event variables, i.e., $q_i \subseteq V$. A transition, $\delta = (q, v, \Theta_\delta)$, leads from a source state, $q \in Q$, to a target state, $q \cup \{v\} \in Q$, if the transition condition, Θ_δ, is satisfied. Set Θ_δ contains conditions that constrain the events bound to event variable v with respect to a constant or with respect to other events bound to event variables in q. The start state, q_s, marks the state in which the execution of an automaton begins. The accepting state, q_f, marks the acceptance of the bindings in buffer β as a matching substitution. Duration τ is the maximal time interval that can be spanned by the events in β. The match buffer, β, collects variable bindings during the execution of the automaton. Whenever a transition, $\delta \in \Delta$, is taken, a binding is added to β. The execution of an automaton expires if the time interval spanned by the earliest event in β and the current input event is larger than τ. If then the automaton is in the accepting state, q_f, β contains bindings that determine a matching substitution.

A SES automaton can be graphically represented as a graph. Nodes represent states, and edges represent transitions. An edge is labeled with the event variable that is bound by the transition and the corresponding transition condition. The start state is marked with an incoming arrow, the accepting state is doubly circled.

Example 5. Figure 3 shows the SES automaton for the SES pattern $P = (\langle\{b\}\rangle, \Theta, 264)$ with $\Theta = \{b.L = \text{'B'}\}$, that is the event set pattern V_2 (in isolation) of our running example. The corresponding automaton is $(\{\emptyset, \{b\}\}, \Delta, \emptyset, \{b\})$. It has two states, the start state $\emptyset$ and the accepting state $\{b\}$, respectively. To facilitate reading, in the graphical illustration we denote states by the concatenation of the corresponding event variables, e.g., the node labeled with b represents state $\{b\}$. There is a single transition, $\Delta = \{(\emptyset, \{b\}, \{b.L = \text{'B'}\})\}$ with a condition that constrains variable b to match a blood count measurement. Notice that no other conditions are involved, since V_2 is considered in isolation.

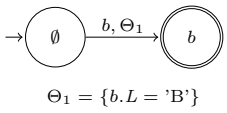


Figure 3: SES Automaton for $P = (\langle\{b\}\rangle, \Theta, 264)$.

4.2 Construction of SES Automaton

The construction of a SES automaton for a pattern, P, is a two-step process: (1) each individual event set pattern, $V_i \in P$, is transformed into a SES automaton and (2) the individual automata from step 1 are concatenated according to the order of the event set patterns in P.

4.2.1 Translation of a Single Event Set Pattern

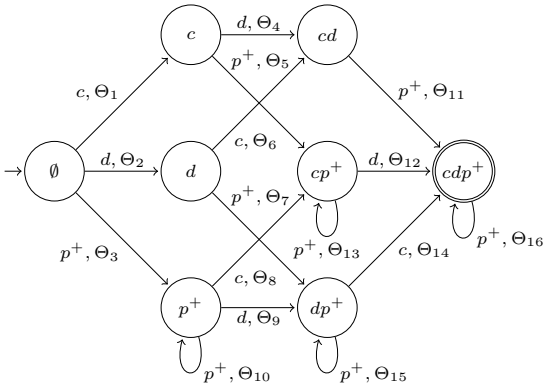
Let $P = (\langle V_1, \ldots, V_m\rangle, \Theta, \tau)$ be a SES pattern, and consider a single event set pattern, $V_i \in P$. For each subset of V_i, the corresponding SES automaton contains a state, i.e., $Q = \{q \mid q \in \mathcal{P}(V_i)\}$. For each state, $q \in Q$, and (singleton and group) event variable, $v \in V_i \setminus q$, a transition $\delta = (q, v, \Theta_\delta)$ is built with condition $\Theta_\delta = \{\theta \mid \theta \in \Theta \wedge (\theta \equiv v.A\ \phi\ C \vee (\theta \equiv v.A\ \phi\ v'.A' \wedge v' \in V_1 \cup \cdots \cup V_{i-1} \cup q \cup \{v\}))\}$. That is, Θ_δ is defined as the set of all conditions from Θ that constrain events bound to event variable v (which is bound by δ) with respect to a constant or with respect to events bound to event variables from preceding event set patterns in P, and the current state. For each state, q, and group variable, $v^+ \in q$, a transition $\delta = (q, v^+, \Theta_\delta)$ is created, which loops at state q since $q \cup \{v^+\} = q$ for $v^+ \in q$; Θ_δ is constructed as before.

Example 6. Figure 4 shows the SES automata N_1 and N_2 for the event set patterns V_1 and V_2, respectively. The automaton for the first pattern is $N_1 = (Q_1, \Delta_1, \emptyset, \{c, d, p^+\}, 264)$, where $Q_1 = \{\emptyset, \{c\}, \{d\}, \{p^+\}, \{c, d\}, \{c, p^+\}, \{d, p^+\}, \{c, d, p^+\}\}$. The transitions Δ_1 and the corresponding transition conditions can be inferred from Figure 4(a). The automaton for the second event set pattern is $N_2 = (Q_2, \Delta_2, \emptyset, \{b\}, 264)$, where $Q_2 = \{\emptyset, \{b\}\}$. The transitions Δ_2 and the corresponding transition conditions can be inferred from Figure 4(b).

4.2.2 Concatenation of SES Automata

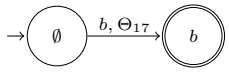
The second step is to concatenate the individual automata $N_1, \ldots, N_m$ from step 1. For two consecutive automata N_i, N_{i+1}, this means essentially to merge the accepting state of N_i with the start state of N_{i+1} plus some renaming and additional constraints which need to be added.

More formally, let $N_1 = (Q_1, \Delta_1, q_{s_1}, q_{f_1}, \tau)$ and $N_2 = (Q_2, \Delta_2, q_{s_2}, q_{f_2}, \tau)$ be the automata for the event set patterns V_1 and V_2, respectively. The concatenation of N_1 and N_2 yields an SES automaton $N = (Q_1 \cup Q_2^*, \Delta_1 \cup \Delta_2^*, q_{s_1}, q_{f_2} \cup V_1)$, which is constructed as follows. The states in Q_2 are renamed to include the event set pattern V_1, i.e., $Q_2^* = \{q \cup V_1 \mid q \in Q_2\}$. The states in the transitions



$\Theta_1 = \{c.L = \text{'C'}\}$
$\Theta_2 = \{d.L = \text{'D'}\}$
$\Theta_3 = \{p^+.L = \text{'P'}\}$
$\Theta_4 = \{d.L = \text{'D'}, c.ID = d.ID\}$
$\Theta_5 = \{p^+.L = \text{'P'}, c.ID = p^+.ID\}$
$\Theta_6 = \{c.L = \text{'C'}, c.ID = d.ID\}$
$\Theta_7 = \{p^+.L = \text{'P'}\}$
$\Theta_8 = \{c.L = \text{'C'}, c.ID = p^+.ID\}$
$\Theta_9 = \{d.L = \text{'D}, c.ID = d.ID\}$
$\Theta_{10} = \{p^+.L = \text{'P'}\}$
$\Theta_{11} = \{p^+.L = \text{'P'}, c.ID = p^+.ID\}$
$\Theta_{12} = \{d.L = \text{'D'}, c.ID = d.ID\}$
$\Theta_{13} = \{p^+.L = \text{'P'}, c.ID = p^+.ID\}$
$\Theta_{14} = \{c.L = \text{'C'}, c.ID = d.ID, c.ID = p^+.ID\}$
$\Theta_{15} = \{p^+.L = \text{'P'}\}$
$\Theta_{16} = \{p^+.L = \text{'P'}, c.ID = p^+.ID\}$

(a) N_1 for $V_1 = \{c, p^+, d\}$



$\Theta_{17} = \{b.L = \text{'B'}, d.ID = b.ID\}$

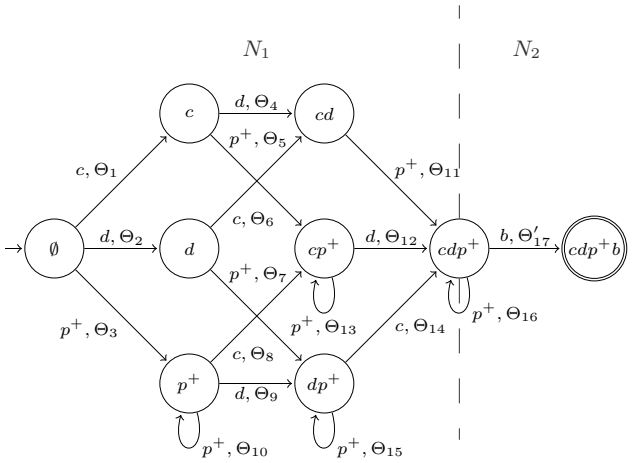
(b) N_2 for $V_2 = \{b\}$

Figure 4: Automata for Event Set Patterns.

are renamed in an analogous way, i.e., for each transition $(q, v, \Theta_\delta) \in \Delta_2$ we have a transition $(q \cup V_1, v, \Theta_\delta)$ in Δ_2^*. Additionally, for each transition $\delta(q_{s_2}, v, \Theta_\delta) \in \Delta_2$, the condition of the corresponding transition in Δ_2^* is extended as follows: $\Theta_\delta = \Theta_\delta \cup \{v'.T < v.T \mid v' \in V_1\}$. These time constraints enforce all events that are bound to V_2 to occur later than the events that are bound to V_1. The start state of the concatenated automaton is q_{s_1} (the start state of N_1), and the accepting state is $q_{f_2} \cup \{V_1\}$ (the accepting state of N_2 after appropriate renaming).

The concatenation of a sequence of automata, $N_1, \ldots, N_m$, is done in the same order as the corresponding event set patterns, $V_1, \ldots, V_m$, are specified in the SES pattern. N_1 and N_2 are concatenated into an intermediate automaton $N' = N_1 N_2$, which is then concatenated with N_3 to give $N'' = (N_1 N_2) N_3$, and so on.

Example 7. The SES automaton in Figure 5 is the result of concatenating N_1 and N_2 from Figure 4, and it corresponds to the SES pattern in our running example. The states of N are given as $Q = \{\emptyset, \{c\}, \{d\}, \{p^+\}, \{c, d\}, \{c, p^+\}, \{d, p^+\}, \{c, d, p^+\}, \{c, d, p^+, b\}\}$, where the accepting state $\{c, d, p^+, b\}$ originates from state $\{b\}$ in N_2 extended by $V_1 = \{c, d, p^+\}$, and the state $\{c, d, p^+\}$ originates from the "merging" of the accepting state of N_1 and the start state of N_2. The conditions, Θ_i, are essentially identical to the conditions in Figure 4, except Θ'_{17} which extends Θ_{17} with time constraints to ensure the correct chronological order between events that match with different event set patterns.



$\Theta'_{17} = \{b.L = \text{'B'}, d.ID = b.ID, c.T < b.T, d.T < b.T, p^+.T < b.T\}$

Figure 5: Automaton for $P = (\langle\{c, p^+, d\}, \{b\}\rangle, \Theta, 264)$.

4.3 Execution of SES Automaton

To describe the execution of a SES automaton, we first define the concept of an automaton instance.

Definition 4. (*Automaton Instance*) An *automaton instance*, $\tilde{N}$, describes a SES automaton N during execution and is a pair

$$\tilde{N} = (q_c, \beta)$$

where $q_c \in Q$ is the state $\tilde{N}$ is currently in and β is the corresponding match buffer.

Algorithm 1 shows function `SESExec` for the execution of a SES automaton. It has two input parameters: a SES automaton, N, which represents a SES pattern, P, and an event relation, E. The function returns all matching substitutions of P in E. The algorithm iterates over all input events, $e \in E$. Each automaton instance $\tilde{N} = (q_c, \beta) \in \Omega$ consumes e. If the duration of the maximal time interval spanned by e and any event in β exceeds τ, $\tilde{N}$ expires. If the expired $\tilde{N}$ is in the accepting state, β is a matching substitution, which is added to the result set R, and $\tilde{N}$ is removed from Ω. Otherwise, if $\tilde{N}$ does not expire, it consumes e by calling function `ConsumeEvent`, which returns a set of automaton instances derived from $\tilde{N}$. Finally, after all automaton instances in Ω are processed, the updated and newly created automaton instances Ω' replace Ω.

Algorithm 2 shows function `ConsumeEvent`, which takes as input parameters an automaton, N, an automaton instance, $\tilde{N}$, and an event, e, and returns a set consisting of

Algorithm 1: `SESExec`(N, E)

Input: SES automaton $N = (Q, \Delta, q_s, q_f, \tau)$, event relation E
Output: set of matching substitutions

```
1  R ← ∅;
2  Ω ← ∅;
3  foreach e ∈ E do
4      Ω ← Ω ∪ {(q_s, ∅)};
5      Ω' ← ∅;
6      foreach Ñ = (q_c, β) ∈ Ω do
7          if max. time span between e and any event in β > τ
           then
8              if q_c = q_f then
9                  R ← R ∪ {β};
10             end
11         else
12             Ω' ← Ω' ∪ ConsumeEvent(N, Ñ, e);
13         end
14     end
15     Ω ← Ω';
16 end
17 return R;
```

automaton instances derived from $\tilde{N}$. The algorithm iterates over all outgoing transitions, $\delta \in \Delta$, from the current state, q_c. In each iteration, e is bound to event variables v in the transition and stored in β' together with the current buffer. Next, Θ_δ is instantiated by the bindings in β' and evaluated. If $\Theta_\delta \beta'$ is satisfied, an automaton instance $(q \cup \{v\}, \beta')$ is created and added to the result set Ω. If the transition conditions, Θ_δ, of several transitions are satisfied, nondeterminism arises, and several new automata branch from $\tilde{N}$. If none of the transitions fires ($\Omega = \emptyset$) the event e is ignored, and if the original automaton instance $\tilde{N}$ is in the start state an empty set is returned, otherwise a set containing only $\tilde{N}$ is returned.

Algorithm 2: `ConsumeEvent`$(N, \tilde{N}, e)$

Input: SES automaton $N = (Q, \Delta, q_s, q_f, \tau)$, automaton instance $\tilde{N} = (q_c, \beta)$, and event e
Output: set of automaton instances

```
1  Ω ← ∅;
2  foreach δ = (q, v, Θ_δ) ∈ Δ such that q = q_c do
3      β' ← β ∪ {v/e};
4      if Θ_δ β' is satisfied then
5          Ω ← Ω ∪ {(q ∪ {v}, β')};
6      end
7  end
8  if Ω = ∅ ∧ q_c ≠ q_s then
9      Ω ← {Ñ};
10 end
11 return Ω;
```

Example 8. Figure 6 shows seven selected steps of the execution algorithm with relation `Event` and the SES automaton N from our running example. The steps refer to the automaton instance, $\tilde{N}$, that produces a matching substitution for patient 1. Each step shows the transition of the automaton instance ($\{\}$ if $\tilde{N}$ does not take any transition), the current state and match buffer of $\tilde{N}$, and the transition graph. The black node represents the current state of $\tilde{N}$ before taking the transition, thick edges represent the transitions that $\tilde{N}$ takes, and gray nodes represent states which $\tilde{N}$ traversed before. For example, in Figure 6(e) the automaton instance is in state $\{c, d\}$, and input event $\mathtt{e}_4$ triggers transition $(\{c, d\}, p^+, \Theta_{11})$. The transition moves the automaton instance to state $\{c, d, p^+\}$ and adds the binding $p^+/\mathtt{e}_4$ to the match buffer β.

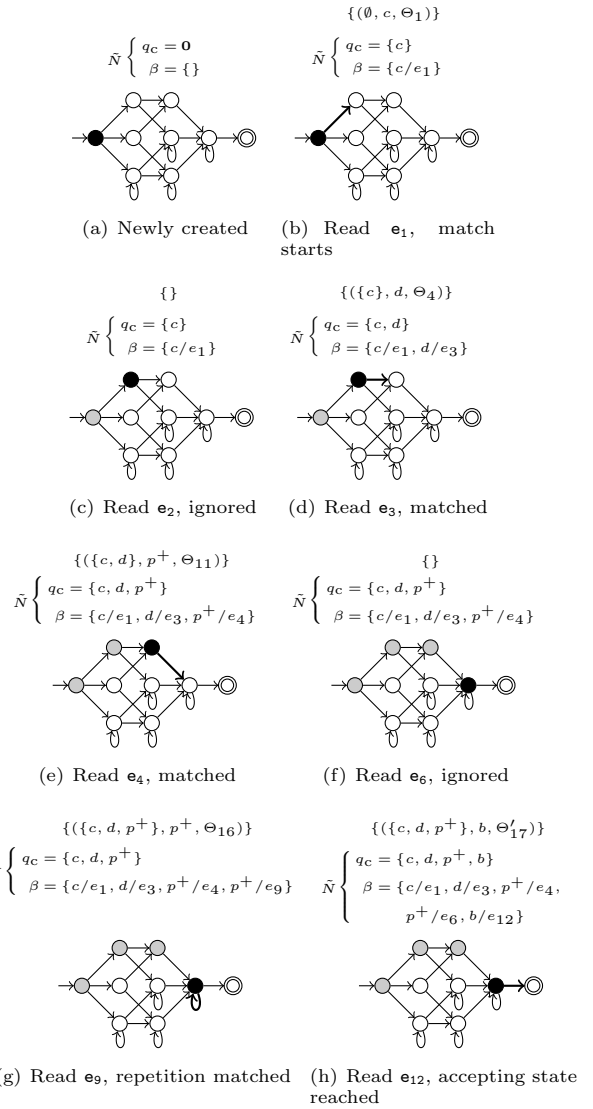


Figure 6: Execution of the SES Automaton N for Pattern $P = (\langle\{c, p^+, d\}, \{b\}\rangle, \Theta, 264)$.

4.4 Complexity Analysis

In this section, we analyze the complexity of `SESExec`, i.e., the execution of a SES automaton, N, for a pattern, P. The complexity of `SESExec` predominantly depends on the number of simultaneous active automaton instances in Ω, which is determined by the number of automaton instances created in the start state of N (line 4 in Algorithm 1) and the number of automaton instances created as a consequence of nondeterminism (line 5 in Algorithm 2). An active automaton instance is aborted if the actual input event and any event in the match buffer span an interval which exceeds duration τ. Therefore, Ω contains only automaton instances that are created while reading events from a time window of width τ.

We begin with the definition of a few concepts, which will be helpful for the complexity analysis.

Definition 5. (*Window Size W*) Let $P = (\langle V_1, \ldots, V_m \rangle, \Theta, \tau)$ be a SES pattern and E be an event relation. The *window size*, W, is defined as the maximal number of events in a time window of size τ sliding over E event-by-event.

The definition of window size W is similar to the size of the partition window [3].

Example 9. In our running example, we have $\tau = 264$ hours, which yields a window size $W = 14$ (from $\mathtt{e}_1$ to $\mathtt{e}_{14}$).

Definition 6. (*Mutually Exclusive Event Variables*) Let $P = (\langle V_1, \ldots, V_m \rangle, \Theta, \tau)$ be a SES pattern and e be an event. Two event variables $v, v' \in V$ are *mutually exclusive* iff

$$\exists v.A\,\phi\,C, v'.A\,\phi'\,C' \in \Theta\ (\ v \neq v' \wedge \nexists e\ (\ (e.A\,\phi\,C) \text{ and } (e.A\,\phi'\,C') \text{ are satisfied }))$$

Example 10. In our running example, all event variables are pairwise mutually exclusive, since Θ contains the conditions $c.L =$ 'C', $d.L =$ 'D', $p^+.L =$ 'P', and $b.L =$ 'B', that are all of the form $v.A\,\phi\,C$ and there does not exist an event that satisfies any two conditions.

LEMMA 1. *Let $P = (\langle V_1, \ldots, V_m \rangle, \Theta, \tau)$ be a SES pattern and N be the corresponding SES automaton. If all event variables in P are pairwise mutually exclusive, nondeterminism cannot occur in any state during the execution of N.*

PROOF. If all event variables in a SES pattern P are pairwise mutually exclusive, Θ contains a condition of the form $v.A\,\phi\,C$ for each event variable v, that cannot be satisfied by the same event. In the corresponding SES automaton N, each two transitions, $\delta = (q, v, \Theta_\delta)$ and $\delta' = (q, v', \Theta'_\delta)$, that leave a state q have in their sets of conditions, Θ_δ and Θ'_δ, conditions of the form $v.A\,\phi\,C$ and $v'.A\,\phi\,C'$, that cannot be satisfied by the same event. Therefore, the set of conditions Θ_δ and Θ'_δ cannot be satisfied contemporarily for the same input event, which excludes nondeterminism. □

In the following analysis we assume a SES automaton, N, for a SES pattern, $P = (\langle V_1 \rangle, \Theta, \tau)$, with a single event set pattern, V_1. Furthermore, we assume that only one automaton instance is started in the start state of N. We distinguish three cases of different patterns and provide for each pattern an upper bound for the cardinality of automaton instances, $|\Omega|$.

Case 1. The event variables in the SES pattern P are pairwise mutually exclusive.

THEOREM 1. *If the variables $v \in V_1$ are pairwise mutually exclusive, the upper bound of $|\Omega|$ is $\mathcal{O}(1)$.*

PROOF. According to Lemma 1, if the event variables in P are pairwise mutually exclusive, nondeterminism cannot occur in N. Consequently, the number of automaton instances in Ω stays constant which leads to a constant upper bound $\mathcal{O}(1)$. □

Figure 7 shows the SES automaton translated from our assumed SES pattern P with $|V_1| = 3$. It illustrates that in case 1 only one automaton instance traverses the automaton. The path from the start state to the accepting state was chosen arbitrarily; any other path would be valid as well.

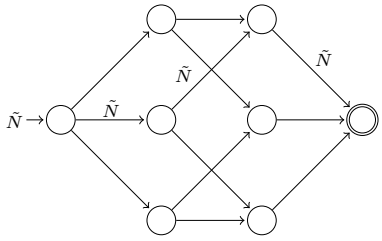


Figure 7: Case 1

Case 2. The event variables in the SES pattern P are not pairwise mutually exclusive, and V_1 does not contain any group variable.

THEOREM 2. *If the variables $v \in V_1$ are not pairwise mutually exclusive and V_1 does not contain any group variable, the upper bound of $|\Omega|$ is $\mathcal{O}(|V_1|!)$.*

PROOF. Since variables $v \in V_1$ are not pairwise mutually exclusive, Lemma 1 does not apply and nondeterminism might occur during the execution of N. In the worst case, each automaton instance that reaches a state $q \in Q$ branches to a number of automaton instances equal to the number of transitions that leave q. If t is the number of transitions that leave q, $t-1$ new automaton instances are created, whereas one transition is taken by the original automaton instance. Thus, there exists an automaton instance for each path from the start state to the accepting state. The number of paths in a SES automaton translated from a SES pattern with one event set pattern V_1 is $|V_1|!$. Thus, the upper bound of $|\Omega|$ is $\mathcal{O}(|V_1|!)$. □

Figure 8 shows the SES automaton translated from our assumed SES pattern P with $|V_1| = 3$. It illustrates how automaton instances branch and which path each one takes to reach the accepting state. Each path in N is used by one automaton instance $\tilde{N}_i$.

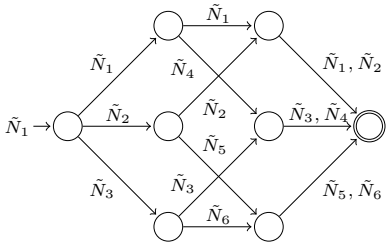


Figure 8: Case 2

Case 3. The event variables $v \in V_1$ are not pairwise mutually exclusive, and V_1 contains group variables.

THEOREM 3. *If the variables $v \in V_1$ are not pairwise mutually exclusive and V_1 contains k group variables, the upper bound of $|\Omega|$ is*

$$\begin{cases} \mathcal{O}((|V_1|-1)! \cdot W^{|V_1|}) & k = 1 \\ \mathcal{O}(k \cdot (|V_1|-1)! \cdot k^{W \cdot |V_1|}) & k > 1 \end{cases}$$

PROOF. Since the variables $v \in V_1$ are not pairwise mutually exclusive, Lemma 1 does not apply and nondeterminism

might occur during the execution of N. Further, with group variables in V_1, N has transitions that loop at a state, i.e., source state q and target state $q \cup \{v\}$ of the transition are the same.

As in case 2, each automaton instance that reaches a state q might branch to a number of automaton instances equal to the number of transitions that leave q. However, if there is a transition that loops at q, the automaton instance that takes this transition returns immediately to q and might cause another branching of automaton instances at q when the next input event is consumed.

Let $r_{q\text{out}}(W_q)$ be a function that returns the number of automaton instances that originate from one automaton instance and that leave q through one transition that does not loop at q. Parameter W_q is the number of events consumed by an automaton instance at q. Function $r_{q\text{out}}(W_q)$ differs depending on the number of transitions that loop at q:

- $r_{q\text{out}}(W_q) = 1$, for a state without any transition that loops,
- $r_{q\text{out}}(W_q) = W_q$, for a state with $k = 1$ transition that loop, and
- $r_{q\text{out}}(W_q) = k^{W_q}$, for a state with $k > 1$ transitions that loop.

On a path from the start state to the accepting state, the maximal number of automaton instances that are created from one automaton instance and that leave a state is $|\Omega|_{\text{path}_i} = \prod_{\text{q} \in path_i} r_{q\text{out}}(W_q)$. The maximal number of automaton instances created on all paths is then $|\Omega|_{\text{out}} = \sum_{i=1}^{|V_1|!} |\Omega|_{\text{path}_i}$.

The start state, $\emptyset$, is the first state of each path through N. Its function $r_{\emptyset\text{out}}(W_\emptyset) = 1$ because $\emptyset$ does not contain any group variable, and hence no transition loops at state $\emptyset$. The paths which match a group variable in their first transition contain the most states with transitions that loop. The number of such paths is $k \cdot (|V_1| - 1)!$ because k path start with a group variable, and after a group variable is matched in the first transition, there are $(|V_1| - 1)!$ paths to the accepting state. The upper bound of $|\Omega|$, with $W_q \leq W$, and k group variables is therefore

$$\begin{cases} \mathcal{O}((|V_1| - 1)! \cdot W^{|V_1|}) & k = 1 \\ \mathcal{O}(k \cdot (|V_1| - 1)! \cdot k^{W \cdot |V_1|}) & k > 1 \end{cases}$$

□

Figure 9 shows the SES automaton translated from our assumed SES pattern P with $|V_1| = 3$ and one group variable. The label on the transitions are the results of the functions $r_{q\text{out}}$, i.e., the number of automaton instances that leave state q. Thick edges emphasize paths which contain the largest number of states with looping transitions.

For a SES pattern P with n event set patterns, the upper bound of $|\Omega|$ is

$$\mathcal{O}(W \cdot (|\Omega|_{\max})^n)$$

where $|\Omega|_{\max}$ is the worst upper bound among the event set patterns in P.

4.5 Filtering Events

Events in an event relation E, which do not satisfy any of the conditions in Θ in a SES pattern, do not cause any transition. Though, they are consumed by each single automaton instance in Ω, and the automaton remains in its current state.

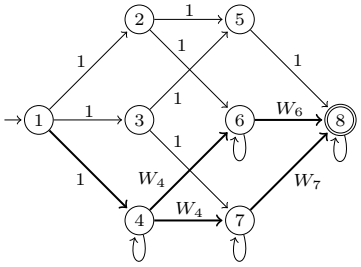


Figure 9: Case 3

To reduce the number of iterations over automaton instances $\tilde{N} \in \Omega$, we only allow iterations over automaton instances if the input event satisfies at least one of the conditions of the form $v.A\ \phi\ C$ in set Θ. All other events are filtered out immediately after they are read. In Algorithm 1, this additional filter is inserted after line 3. Notice, that this optimization does not reduce the number of automaton instances but only the number of iterations over the set of automaton instances Ω.

5. EXPERIMENTS

In this section, we report the results of an empirical evaluation using real-world data.

The experiments have three purposes. The first purpose is to compare the SES automaton algorithm to a brute force approach that matches sequences of sets of events by using a set of automata, each of which matches a sequence of single events. The second purpose is to evaluate the results of the complexity analysis in Section 4.4. The third purpose is to show the effect of filtering events as described in Section 4.5 on the runtime of the SES automaton algorithm.

The three hypotheses corresponding to our purposes are the following. First, the SES automaton algorithm is more efficient in terms of the maximal number of simultaneous automaton instances than the brute force approach. Second, the number of simultaneous automaton instances is upper-bounded according to the theorems in Section 4.4. Third, filtering events is an effective strategy to decrease the runtime of the SES automaton algorithm.

5.1 Setup and Data

For the experiments, we implemented the automaton-based evaluation algorithm in C. The relation with the input events is stored in an Oracle database, Enterprise Edition 11.1, which is accessed over the OCI API. The experiments were performed on a PC with four AMD Opteron 285 processors with 2.6 GHz and 16 GB memory, on which a 64-bit Linux 2.6.32 is installed.

The event relation used in our experiments is a real-world data set with chemotherapy events from the Department of Haematology at the Hospital Meran-Merano. To obtain event relations with different characteristics, we generated from the original data set the following five data sets with corresponding window size W:

- D_1 is the original data set with $W = 1322$;
- D_2 contains each event twice and has $W = 2644$;

- D_3 contains each event three times and has $W = 3966$;
- D_4 contains each event four times and has $W = 5288$;
- D_5 contains each event five times and has $W = 6610$.

5.2 Brute Force Algorithm

We compare our SES automaton that uses one automaton to match sequences of sets of events with a brute force algorithm that uses a set of automata, each of which matches a sequence of single events. The brute force algorithm generates all possible sequences (orderings) of event variables of an event set pattern, creates a SES automaton for each of these sequences, and executes all automata over the event relation.

Let P be a SES pattern with event set patterns $\langle V_1, \ldots, V_n \rangle$, where all event set patterns contain only singleton variables. A sequence of all event variables in P is a concatenation of one permutation of each event set pattern V_i. The number of all possible sequences of event variables is $|V_1|! \cdot |V_2|! \cdots |V_n|!$. The brute force algorithm creates for each of these sequences an automaton and executes all automata in parallel, i.e., iterates for each input event over these automata. By specifying each event variable as an event set pattern with exactly one singleton variable, we can use SES automata. The brute force algorithm essentially corresponds to straightforward extensions of the automata in [13, 3, 11]. We do not consider any optimizations of the automata described in these papers, rather our aim is to compare the efficiency of the plain automata to solve SES pattern matching. The optimization of the SES automaton and its comparison to related approaches remains future work.

Example 11. Consider a slight modification of the SES pattern in our running example, where all event variables are singleton variables, i.e., $(\langle \{c, p, d\}, \{b\} \rangle, \Theta, 264)$. The possible sequences of event variables are given as follows:

$$P_1 = \langle c, d, p, b \rangle \quad P_2 = \langle c, p, c, b \rangle \quad P_3 = \langle d, c, p, b \rangle$$
$$P_4 = \langle d, p, c, b \rangle \quad P_5 = \langle p, c, d, b \rangle \quad P_6 = \langle p, d, c, b \rangle$$

Figure 10 shows the corresponding SES automaton that is created by our evaluation algorithm and the set of SES automata that are created by the brute force algorithm.

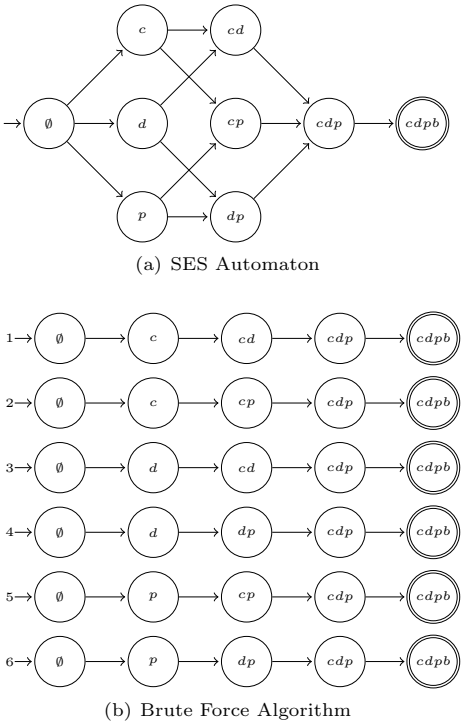
If an event set pattern V_i in P contains group variables, the number of different event sequences, and hence the number of automata created by the brute force algorithm, considerably increases.

5.3 Experiment 1

The purpose of experiment 1 is to compare the SES automaton algorithm to the brute force (BF) algorithm. Our hypothesis is that the SES automaton algorithm produces less simultaneous automaton instances than the brute force algorithm, and the difference is increasing with the size of the event set pattern.

We use the following two SES patterns with identical event set patterns, but different conditions:

- $P_1 = (\langle V_1 = \{c, d, p, v, r, l\}, V_2 = \{b\} \rangle, \Theta_1, 264)$
- $P_2 = (\langle V_1 = \{c, d, p, v, r, l\}, V_2 = \{b\} \rangle, \Theta_2, 264)$



(a) SES Automaton

(b) Brute Force Algorithm

Figure 10: SES Automaton and Set of Automata Created by the Brute Force Algorithm.

Θ_1 specifies that each event variable in V_1 matches a distinct medication administration event, thus all event variables are pairwise mutually exclusive. Θ_2 specifies that all event variables in V_1 match the same type of medication administration events, thus all event variables are not pairwise mutually exclusive. Data set D_1 is used as event relation.

We vary the number of event variables in V_1 from two to six in steps of one, i.e., $\{c, d\}, \{c, d, p\}, \ldots, \{c, d, p, v, r, l\}$. The measured parameter is the maximal number of automaton instances that are simultaneously active during the execution, i.e., $|\Omega|$ in Algorithm 1.

Figure 11 shows the results of this experiment. With pattern P_1, the SES automaton algorithm produces significantly less simultaneous automaton instances than the brute force algorithm. The reason is that when the SES automaton algorithm creates one automaton instance for an event that matches an event variable in V_1, the brute force algorithm creates $(|V_1| - 1)!$ automaton instances because $(|V_1| - 1)!$ automata start with the same event variable (see Figure 10(b)). Since all event variables are pairwise mutually exclusive, nondeterminism does not occur in both algorithms, hence automaton instances never branch during the execution. This can also be seen in Table 1, which shows the ratio of the maximal number of automaton instances that are produced by the two algorithms.

With P_2, the SES automaton algorithm creates between 9% and 20% less automaton instances than the brute force algorithm. The SES automaton algorithm creates $|V_1|$ automaton instances when it consumes an event that matches all event variables in V_1 and then if other events that match all event variables in V_1 are encountered the number increases to maximal $|V_1|!$ (see Theorem 2) due to

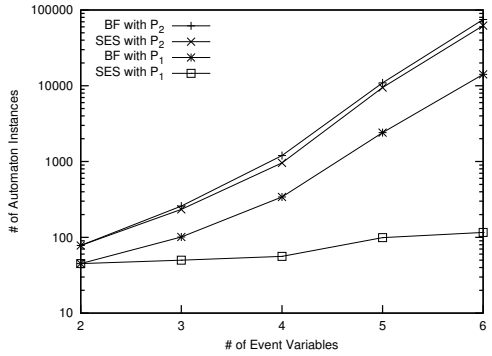


Figure 11: Experiment 1

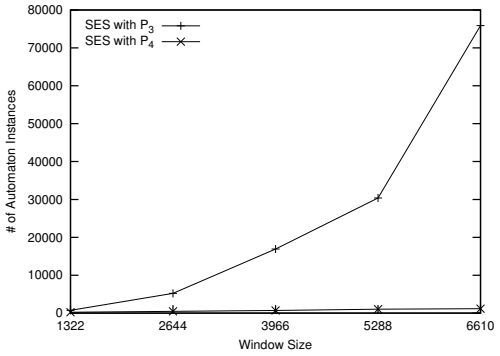


Figure 12: Experiment 2

nondeterminism. The brute force algorithm, instead, creates $|V_1|!$ automaton instances at once and keeps this number of instances until they expire, independently of other events that match all event variables in V_1. To summarize, the SES automaton algorithm creates automaton instances when they are needed and the brute force algorithm creates possibly needed automaton instances in advance duplicating execution steps.

$\|V_1\|$	$\|\Omega\|_{BF}$	$\|\Omega\|_{SES}$	$\frac{(\Omega)_{BF}}{\Omega)_{SES}}$	$(\|V_1\|-1)!$
2	45	45	1	1
3	101	50	2	2
4	341	56	6.1	6
5	2414	99	24.4	24
6	14150	116	122	120

Table 1: Ratio of Numbers of Automaton Instances

5.4 Experiment 2

The purpose of experiment 2 is to validate Theorem 2 and Theorem 3. Our hypothesis is that the maximal number of automaton instances that are simultaneously active is upper-bounded as stated in Theorem 2 and Theorem 3.

We use the following two SES patterns:

- $P_3 = (\langle V_1 = \{c, d, p^+\}, V_2 = \{b\}\rangle, \Theta, 264)$
- $P_4 = (\langle V_1 = \{c, d, p\}, V_2 = \{b\}\rangle, \Theta, 264)$

Θ is identical for both patterns and constrains all event variables in V_1 to match the same type of medication administration events. Hence, the event variables are not pairwise mutually exclusive.

We vary the window size, W, by using the data sets D_1 to D_5. The measured parameter is the maximal number of simultaneous automaton instances during the execution.

The graph in Figure 12 shows the number of simultaneous automaton instances depending on the window size, W. The results show for P_3 a polynomial trend of the number of automaton instances with an increasing W, which validates Theorem 3. For P_4, the results show a linear trend of the number of automaton instances with increasing W, which validates Theorem 2.

5.5 Experiment 3

The purpose of experiment 3 is to show the effect of filtering events as described in Section 4.5 on the runtime. Our hypothesis is that filtering events decreases the runtime of the SES automaton algorithm.

We use the following two SES patterns:

- $P_5 = (\langle V_1 = \{c, d, p^+\}, V_2 = \{b\}\rangle, \Theta_1, 264)$
- $P_6 = (\langle V_1 = \{c, d, p^+\}, V_2 = \{b\}\rangle, \Theta_2, 264)$

The set of conditions, Θ_1, specifies that each event variable in V_1 matches distinct medication administration events, hence all event variables are pairwise mutually exclusive. The set Θ_2 specifies that all event variables in V_1 match the same type of medication administration events, and all event variables are not pairwise mutually exclusive.

We vary window size W similar to the previous experiment in Section 5.4. The measured parameter is the execution time of SES automaton algorithm.

The graph in Figure 13 shows the execution time depending on the window size, W. Filtering events reduces the execution time by an order of magnitude for our test data set, independently whether the event variables are pairwise mutually exclusive or not.

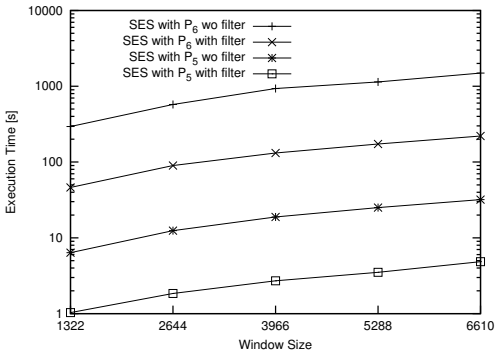


Figure 13: Experiment 3

6. CONCLUSION

In this paper we introduced and formally defined sequenced event set pattern matching, which is the problem of matching a sequence of input events against a pattern that allows to specify a sequence of sets of events. The order of events that match with the same set in the pattern is irrelevant, i.e., any permutation of the events is matched, whereas

events that match distinct sets are strictly consecutive. We proposed an automaton-based evaluation algorithm and provided a detailed complexity analysis for different types of patterns. We conducted an experimental evaluation study using real-world data. The results of the experiments validate the complexity analysis and show that our algorithm clearly outperforms a brute force approach, which is based on existing techniques. The study shows also that event filtering is effective to reduce the number of simultaneous automaton instances.

Future work is possible in various directions, including the following ones: investigate in detail the expressiveness of SES automata, enhance SES automata to support a broader class of SES patterns, and study space and runtime optimizations [3] for our algorithm, including indexing techniques for automaton instances [11].

7. ACKNOWLEDGEMENTS

This work has been done within the framework of the MEDAN project, which is funded by the Hospital of Meran-Merano. We wish to thank Peter Huber, Manfred Mitterer, and Gilbert Spizzo for their continuous support in providing us real-world data and useful domain knowledge.

8. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the Borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[3] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, 2008.

[4] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *PODC*, pages 53–61, 1999.

[5] A. Arasu, S. Babu, and J. Widom. CQL: A language for continuous queries over streams and relations. In *DBPL*, pages 1–19, 2003.

[6] R. S. Barga, J. Goldstein, M. H. Ali, and M. Hong. Consistent streaming through time: A vision for event stream processing. In *CIDR*, pages 363–374, 2007.

[7] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik. Monitoring streams: a new class of data management applications. In *VLDB*, pages 215–226, 2002.

[8] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite events for active databases: Semantics, contexts and detection. In *VLDB*, pages 606–617, 1994.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] Coral8. http://www.aleri.com/developers/documents/coral8.

[11] A. J. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. M. White. Towards expressive publish/subscribe systems. In *EDBT*, pages 627–644, 2006.

[12] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422, 2007.

[13] N. Dindar, B. Güç, P. Lau, A. Ozal, M. Soner, and N. Tatbul. Dejavu: declarative pattern matching over live and archived streams of events. In *SIGMOD*, pages 1023–1026, 2009.

[14] L. Ding, S. Chen, E. A. Rundensteiner, J. Tatemura, W.-P. Hsiung, and K. S. Candan. Runtime semantic query optimization for event stream processing. In *ICDE*, pages 676–685, 2008.

[15] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD*, pages 115–126, 2001.

[16] S. Gatziu and K. R. Dittrich. Events in an active object-oriented database system. In *Rules in Database Systems*, pages 23–39, 1993.

[17] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite event specification in active databases: Model & implementation. In *VLDB*, pages 327–338, 1992.

[18] D. Gyllstrom, J. Agrawal, Y. Diao, and N. Immerman. On supporting Kleene closure over event streams. In *ICDE*, pages 1391–1393, 2008.

[19] L. Harada and Y. Hotta. Order checking in a CPOE using event analyzer. In *CIKM*, pages 549–555, 2005.

[20] Y. Mei and S. Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD*, pages 193–206, 2009.

[21] R. Meo, G. Psaila, and S. Ceri. Composite events in Chimera. In *EDBT*, pages 56–76, 1996.

[22] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. In *CIDR*, 2003.

[23] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. Expressing and optimizing sequence queries in database systems. *ACM Trans. Database Syst.*, 29(2):282–318, 2004.

[24] P. Seshadri, M. Livny, and R. Ramakrishnan. The design and implementation of a sequence database system. In *VLDB*, pages 99–110, 1996.

[25] StreamBase. http://www.streambase.com.

[26] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006.

[27] F. Zemke, A. Witkowski, M. Cherniak, and L. Colby. Pattern matching in sequences of rows. Technical report, 2007.

[28] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. In *ICDE*, page 392, 1999.