Pre-automata and Complex Event Processing

Grygoriy Zholtkevych, Boris Novikov, and Volodymyr Dorozhinsky(✉)

Department of Theoretical and Applied Computer Science, V.N. Karazin Kharkiv National University, 4, Svobody Sqr., Kharkiv 61022, Ukraine
g.zholtkevych@karazin.ua, vdorozhinsky@gmail.com

Abstract. In the paper complex event processing systems are considered. The survey of problems and approaches to their solutions associated with this technology is contained in the paper. Based on this survey a mathematical model for complex event processing systems is proposed. Samples of such a model are called a CEP-machines. Authors have demonstrated that this model is closely related with the notion of a pre-automaton introduced in their earlier papers. The proposed model provides rigorous formulations of problems associated with complex event processing. Authors have proved that processing ability of such systems is determined by some compromise between the class of processed event stream and a number of complex events. Further in the paper the problem of CEP-machine synthesis basing on the formal specification has been solved. Authors have established the conditions ensuring possibility of algorithmic realisation for CEP-machines.

Keywords: Pre-automaton · Complex event · Mathematical model · Event processing · Response function · Computable function · Decidable set

1 Introduction

A modern development trend of information and communication technology demonstrates a stable growth of importance to monitor and analyse continuous information streams. These processes require execution "on-the-fly" thereby they should have the ability to respond in a real time mode.

As a rule, a continuous information stream is not continuous in the mathematical sense of this word. A continuous information stream can be considered rather as a sequence of elementary (atomic) events such that each of them carries some information fragment. A finite segment of an event stream is a complex event if this segment has semantic meaning in the context of our interest.

Complex Event Processing (CEP) is a method of tracking and analysing streams of data about things that happen (they are called events) [11, p.3] and about effects caused by them. The goal of complex event processing is to identify

Grygoriy Zholtkevych and Volodymyr Dorozhinsky regret inform about the sudden death of Prof. Boris Novikov.

V. Ermolayev et al. (Eds.): ICTERI 2014, CCIS 469, pp. 100–116, 2014.
DOI: 10.1007/978-3-319-13206-8_5

meaningful events (such as opportunities or threats) and respond to them as quickly as possible [2]. These events may be happening across the various layers of an organization as sales leads, orders or customer service calls. Or, they may be news items [3], text messages, social media posts, stock market feeds, traffic reports, weather reports, or other kinds of data [11].

Important application areas of CEP are the following [7]:

- "Business Activity Monitoring aims at identifying problems and opportunities in early stages by monitoring business process and other critical resources. To this end, it summarizes events into so-called key performance indicators such as, e.g., the average run time of a process";
- "Sensor Networks transmit measured data from the physical world to, e.g., Supervisory Control and Data Acquisition Systems that are used for monitoring of industrial facilities. To minimize measurement and other errors, data of multiple sensors has to be combined frequently. Further, high-level situations (e.g., fire) usually have to be derivative from raw numerical measurements (e.g., temperature, smoke, etc.)";
- "Market Data such as stock or commodity prices can also be considered as events. They have to be analysed in a continuous and timely fashion in order to recognize trends early and to react to them automatically, for example, in algorithmic trading."

The concepts of "timelines" and "flow processing" are critical to explain the need of a new class of systems. Truly, traditional Data Base Management Systems (DBMSs):

- require data to be (persistently) stored and indexed before it could be processed;
- process data only when explicitly asked by the users, i.e., asynchronously with respect to its arrival.

Both aspects contrast with the requirements of modern monitoring systems [4]. To fulfill the requirements two main concepts are used:

- The Data Stream Processing Model [1] represents the information flow processing problem as processing streams of data coming from different sources to generate new data streams as an output, and considers this problem as an extension of traditional data processing, as supported by Data Base Management Systems. Therefore, Data Stream Management Systems (DSMSs) have their roots in DBMSs but introduce essential differences. While traditional DBMSs are designed to work on persistent data, where updates are relatively infrequent, DSMSs are specialized in dealing with transient data that is continuously updated. The same way, while DBMSs run queries just once to return a complete answer, DSMSs execute standing queries, which run continuously and provide updated answers as new data arrives;
- The Complex Event Processing Model [10] considers flowing information items as notifications of events happening in the external world, which have to be filtered and combined to understand what is happening in terms of higher-level events. Therefore, the focus of this model is on detecting occurrences of

particular patterns of (low-level) events that represent the higher-level events whose occurrences has to be notified to the interested parties [4].

The situations (specified as complex events) that need to be detected in the applications mentioned above and the information associated with these situations are distributed over several events. Thus CEP can only derive such situations from a number of correlated (simple) events. To this end many different languages and formalisms for querying events, the so-called Event Query Languages (EQLs), have been developed in the past. Today the following EQL styles are used [7]:

- Composition Operators build complex event queries from simple event queries using composition operators. Complex event queries are expressed by composing single events using different composition operators. Typical operators are conjunction of events (all events must happen, possibly at different times), sequence (all events happen in the specified order), and negation within a sequence (an event does not happen in the time between two other events);
- Data Stream Query Languages are based on the database query language SQL and the following general idea: Data Streams carry events represented as tuples. Each data stream corresponds to exactly one event type. The streams are converted into relations which essentially contain (parts of) the tuples received so far. On these relations a (almost) regular SQL query is evaluated. The result (another relation) is then converted back into a data stream;
- Production Rules are very flexible and well integrated with existing programming languages. However, it entails working on a low abstraction level that is – since it is primarily state and not event oriented – somewhat different from other EQLs. Especially aggregation and negation are therefore hard to express. Production rules are considered to be less efficient then data stream query languages; this is however tied to the flexibility they add in terms of combining queries (in rule conditions) and reactions (in rule actions);
- Timed State Machines are usually used to model the behavior of a stateful system that reacts to events. The system is modelled as a directed graph. The nodes of the graph represent the possible states of the system. Directed edges are labelled with events and temporal conditions on them. The edges specify the transitions between states that occur in reaction to in-coming events. State machines are founded formally on deterministic or non-deterministic finite automata. Since states in a state machine are reached by particular sequence of multiple events occurring over time, they implicitly define complex events;
- Logic Languages express event queries in logic-style formulas. Logic languages offer a natural and convenient way to specify event queries. The main advantage of logic languages is their strong formal foundation, an issue which is neglected by many languages of other styles. Thanks to the separation of different dimensions of event processing, logic languages are highly expressive, extensible and easy to learn and use.

CEP depends on a number of techniques, [8] including: event-pattern detection; event abstraction; event filtering; event aggregation and transformation;

modelling event hierarchies; detecting relationships (such as causality, membership or timing) between events; abstracting event-driven processes.

The presented survey shows that there are a number of technical solutions and software tools for implementing the concept of CEP. But some principal problems for theoretical substantiation of the concept remain open. Among them, e.g. the existence problem for unhandled event streams, which has an important value for system design. The problem of CEP-system synthesis in accordance with its input/reaction specification is the following example of such open problems.

The main objective of this paper is to construct a mathematical model for complex event processing systems to create a theoretical background for formal analysis such systems.

The mathematical model proposed by authors is closely related with the notion of a pre-automaton, which was introduced in [6] and studied further in [12,14–16].

This paper consists of this introduction, seven sections, and conclusion. Section 2 introduces basic notion and notations. Section 3 contains the comparison of architectures for simple event processing and complex event processing.

Section 4 is devoted to constructing a mathematical model of a system for complex event processing. This model is called a CEP-machine. Elementary properties of a CEP-machine are studied in Sect. 5. The CEP-machine synthesis problem is considered in Sect. 6.

Finally, in Sect. 7 the problem of algorithmic realising for a CEP-machine is studied.

2 Basic Notions and Notations

Let X and Y be sets and f be a partial mapping from X into Y then the notation $f : X \dashrightarrow Y$ is used to specify that f is a partial mapping in contrast to the notation $X \to Y$, which is used to specify everywhere defined mappings.

Let $f : X \dashrightarrow Y$ be a partial mapping from a set X into a set Y, x be some element of X then the notation $f(x) \downarrow$ is used to indicate that x belongs to the domain of the mapping f. Moreover, if x belongs to the domain of the mapping f and it is known that $f(x) = y$ then the denotation $f(x) \downarrow y$ is used to express this fact. Similarly, the notation $f(x) \uparrow$ is used to indicate that x does not belong to the domain of the mapping f. As usually, we use the denotation $D(f)$ for the domain of f.

Let Σ be a finite alphabet. As usually, we use the notation Σ^* to refer to the free monoid generated by Σ. The unit of this monoid is denoted by ε. The semigroup of all non-empty words over an alphabet Σ is denoted by Σ^+. Hence, $\Sigma^* = \Sigma^+ \bigcup \{\varepsilon\}$.

For any $u \in \Sigma^+$ we denote by $|u|$ the length of u, i.e. a number of symbols containing in u, and set that $|\varepsilon| = 0$.

Any set L consisting of words over alphabet Σ is called prefix-free if assertions $uv \in L$ and $u \in L$ for $u, v \in \Sigma^*$ imply the equality $v = \varepsilon$. For any $L \subset \Sigma^+$ we denote by $C(L)$ the following set

$$C(L) = \{w \in L \mid w = uv \text{ and } u \in L \text{ imply } v = \varepsilon\}\,.$$

One can easily see that $L \subset \Sigma^+$ is prefix-free if and only if $C(L) = L$.

In addition to words over some alphabet we consider infinite sequences of alphabet symbols. Therefore we use the notation Σ^ω for the set of all one-way infinite sequences of symbols belonging to the alphabet Σ. For $\pi \in \Sigma^\omega$ and $n \in \mathbb{N}$ we use the notation $\pi_{[1..n]}$ to refer to the word that has a length n and coincides with the beginning of the sequence π. We denote also by $\pi_{(n..)}$ the sequence belonging to Σ^ω that is defined by the equality $\pi = \pi_{[1..n]}\pi_{(n..)}$.

3 Event Processing Versus Complex Event Processing

In this section we are going to consider specificity of CEP in contrast to simple event processing.

Definition 1 (Atomic and Complex Events). *We shall say that an event is atomic if it can not be represented as a complex of sub-events that are essential for the domain of our interest.*

In contrast, we shall say that an event is complex if it can be represented as a complex of sub-events that are essential for the domain of our interest.

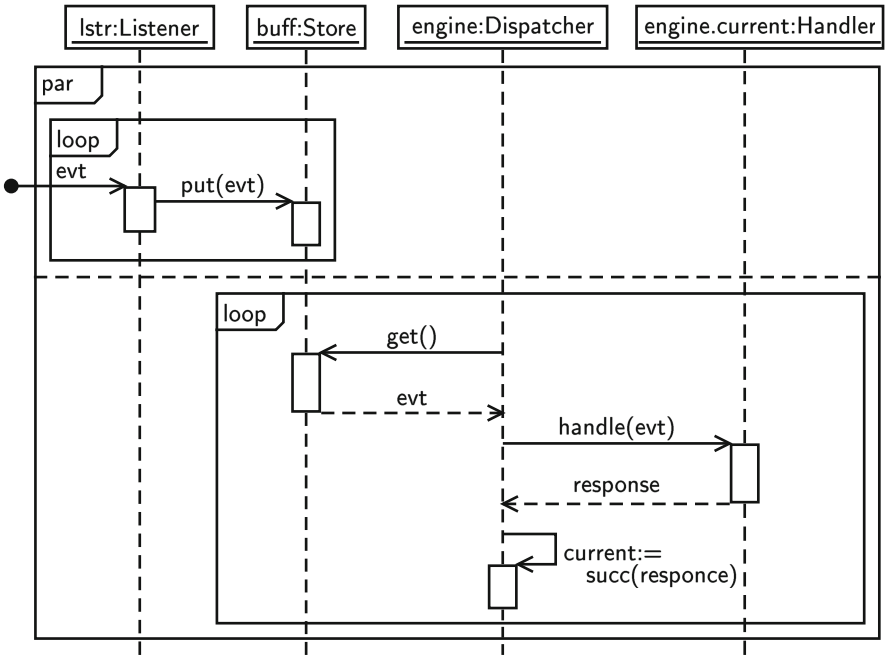

Fig. 1. Simple event processing

Example 1. Language Java distinguishes low-level events and semantic events. A keyboard, a mouse and other input devices produce the low-level events.

In contrast to low-level events semantic events are produced as a result of some sequence of low-level events, e.g. SelectItemEvent is a result of the sequence: GetFocusEvent, DownButtonEvent, and UpButtonEvent.

Definition 2 (Simple Event Processing). *Simple event processing is based on the suggestion that each event is directly related to specific, measurable changes of condition and can be processed standalone.*

In the case of simple event processing each notable happened event initiates immediately the corresponding action.

Simple event processing is commonly used to drive the real-time work-flow thereby reducing lag time and cost [13].

As shown in Fig. 1 event handler processes atomic events as soon as notifications about them being received.

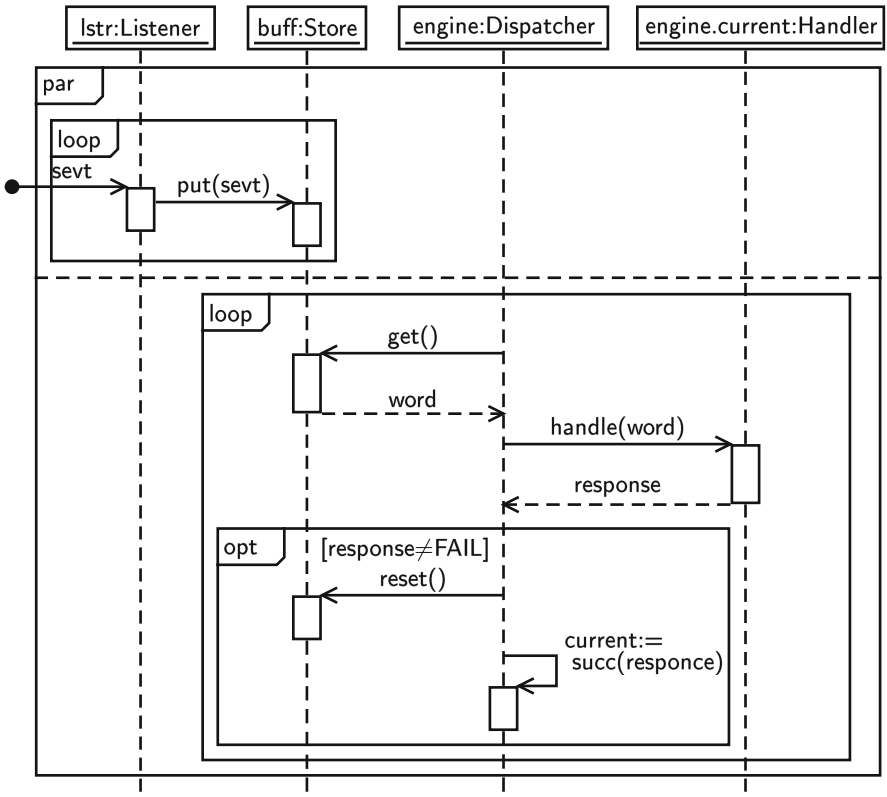

Fig. 2. Complex event processing

Definition 3 (Complex event processing). *Complex event processing is based on the suggestion that each atomic event carries too small volume of information to determine the adequate reaction.*

Complex event processing evaluates a confluence of events, determines an event pattern and then takes the corresponding action.

Complex events may have different types and may occur over a long period of time. The event dependencies may be different also and be causal, temporal, or spatial. CEP requires to use sophisticated event interpreters, event pattern definitions and recognition mechanisms [13].

As shown in Fig. 2 complex event handler is searching through the event buffer to recognise a complex event. And if the complex event was recognised then handler processes it and clears the event buffer. Thus the main difference between Simple Event Processing and Complex Event Processing is that in the first case a processing system reacts on each atomic event immediately while in the second case a processing system is trying to detect and process some meaningful complex event by analysing the buffer of atomic events. Facts mentioned above and results obtained in [12,14–16] show that concept of the Pre-Automata can be widely used in mathematical modelling of Complex Event Processing Systems.

4 Formal Model of Complex Event Processing System

In this section some mathematical model of an abstract machine for complex event processing is built. To do this we choose and fix two finite alphabets Σ and A to describe atomic events and system responses respectively.

Definition 4 (a handler). *A partially defined map $h : \Sigma^+ \dashrightarrow \mathrm{A}$ we shall call a handler if its domain $D(h)$ is a prefix-free set.*

To explain the necessity of the equation $C(D(h)) = D(h)$ note that the requirement to accumulate a sequence of atomic events until it is recognised as a processed sequence, is described by this equation.

Definition 5 (a pattern). *Let $h : \Sigma^+ \dashrightarrow \mathrm{A}$ be a handler and $a \in \mathrm{A}$ be some response then the prefix-free set $P_h(a) = \{w \in D(h) \mid h(w) = a\}$ is called an h, a-pattern.*

A definition of an abstract machine for complex event processing should answer the informal description given by Fig. 2.

Definition 6 (CEP-machine). *An abstract machine for complex event processing (below a CEP-machine) is a quadruple $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ where Σ and A are finite alphabets of events and responses respectively, H is a finite set of handlers, and $\delta : \mathrm{A} \to H$ is a total map, which is called a transition map.*

One can easily see that each handler h in the model corresponds to the method handle() of the class *Handler* and the map δ corresponds to the method succ() of the class Dispatcher (see Fig. 2).

To define a behaviour of a CEP-machine let us consider the set Σ^ω containing all infinite sequences of elements belonging Σ. Such sequences will be called event streams.

Let us define the partial function $T_h : \Sigma^\omega \dashrightarrow \mathbb{N}$ for any handler $h \in H$ by the following conditions

$$\begin{array}{l} T_h(\pi) \downarrow t \quad \text{if } \pi_{[1..t]} \in D(h)\,; \\ T_h(\pi) \uparrow \quad \text{if } (\forall\, t \in \mathbb{N})\ h(\pi_{[1..t]}) \uparrow\ . \end{array} \tag{1}$$

Informally, $T_h(\pi)$ is the first response time of the handler h under processing the event stream π.

Definition 7 (an evolutionary operator of CEP-machine). *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ be a CEP-machine then the partial map $S : H \times \Sigma^\omega \dashrightarrow H \times \Sigma^\omega$ defined by the conditions*

$$\begin{array}{ll} S\langle h, \pi\rangle \downarrow \langle \delta(h(\pi_{[1..t]})), \pi_{(t..)}\rangle & \text{if } T_h(\pi) \downarrow t \\ S\langle h, \pi\rangle \uparrow & \text{if } T_h(\pi) \uparrow \end{array}$$

will be called an evolutionary operator of the CEP-machine.

Now to specify a correct behaviour of a CEP-machine we define its valid scenarios called work-flows.

Definition 8 (a work-flow). *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ be a CEP-machine and $\langle \langle h^{(t)}, \pi^{(t)}\rangle \mid\ t = 0, 1, \dots \rangle$ be a sequence over $H \times \Sigma^\omega$ then it is called a work-flow if the following equality holds*

$$\langle h^{(t+1)}, \pi^{(t+1)}\rangle = S\langle h^{(t)}, \pi^{(t)}\rangle \text{ for all } t \geq 0\,.$$

5 Elementary Properties of CEP-machines

Simple, but very important properties of CEP-machines are a consequence of the existence of some natural topology on the set Σ^ω.

More precisely, for a finite alphabet Σ the family $\mathcal{B} = \{u \cdot \Sigma^\omega \mid u \in \Sigma^+\}$ of subsets over Σ^ω holds characteristic properties of a topological base. Therefore, we consider Σ^ω as a topological space and the corresponding topology is called Tychonoff topology on sequence space. It is well-known that this topological space is a bicompact.

Proposition 1. *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ be a CEP-machine then for any $h \in H$ the function $T_h : \Sigma^\omega \dashrightarrow \mathbb{N}$ defined by (1) is continuous function on the set $\{\pi \in \Sigma^\omega \mid T_h(\pi) \downarrow\}$ under the assumption that the topology on $\mathbb{N}$ is discrete.*

Proof. One can easily see that for any $t \in \mathbb{N}$ the equality

$$T_h^{-1}(t) = \bigcup_{u \in D(h) : |u| = t} u \cdot \Sigma^\omega$$

is true. But the right side of this equality is a union of sets belonging to $\mathcal{B}$, hence $T^{-1}(t)$ is an open set. □

Corollary 1. *The function* $T_h : \Sigma^\omega \dashrightarrow \mathbb{N}$ *is a piecewise constant function.*

Corollary 2. *Let* $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ *be a CEP-machine,* S *be its evolutionary operator, and* $D(S)$ *be domain of* S *then*

$$D(S) = \bigcup_{h \in H} D_h(S)\,,$$

where each $D_h(S) \subset \Sigma^\omega$ *is an open set in Tychonoff topology on* Σ^ω.

Theorem 1. *Let* $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ *be a CEP-machine,* S *be its evolutionary operator, and* $h \in H$ *be some handler then* $S\langle h, \pi\rangle \downarrow$ *for all* $\pi \in \Sigma^\omega$ *if and only if* $D(h)$ *is finite and* $\Sigma^\omega = \bigcup_{u \in D(h)}\ u \cdot \Sigma^\omega$.

Proof. Indeed, suppose that $S\langle h, \pi\rangle \downarrow$ for any event stream π then the statement $T_h(\pi) \downarrow$ is true for all $\pi \in \Sigma^\omega$. It means that

$$\Sigma^\omega = \bigcup_{t=1}^{\infty} T^{-1}(t) = \bigcup_{t=1}^{\infty} \left(\bigcup_{u \in D(h) : |u| = t} u \cdot \Sigma^\omega \right).$$

Further, taking into account Proposition 1 and Weierstrass Boundedness Theorem one can conclude that there exists $M \in \mathbb{N}$ such that $T_h(\pi) \leq M$ for all $\pi \in \Sigma^\omega$. Hence, we obtain that

$$\Sigma^\omega = \bigcup_{t=1}^{M} \left(\bigcup_{u \in D(h) : |u| = t} u \cdot \Sigma^\omega \right).$$

Thus, the set $D_0(h) = \{u \in D(h) \mid 1 \leq |u| \leq M\}$ is finite.

Let $D_0(h) = \{u_1, \ldots, u_m\}$ then

1. $u_i \in D(h)$ for all $i = 1, \ldots, m$ and
2. $\Sigma^\omega = \bigcup_{i=1}^{m}\ u_i \cdot \Sigma^\omega$.

We claim that $u_i \cdot \Sigma^\omega \bigcap u_j \cdot \Sigma^\omega = \emptyset$ for $1 \leq i \neq j \leq m$. Indeed, suppose that there exists some $\pi \in u_i \cdot \Sigma^\omega \bigcap u_j \cdot \Sigma^\omega$ then there are three possibilities:

- $|u_i| = |u_j|$ and in this case we have the equality $u_i = u_j$ that contradicts the assumption $i \neq j$;
- $|u_i| < |u_j|$ and u_i is a prefix of u_j that contradicts the equality $C(D(h)) = D(h)$;
- $|u_i| > |u_j|$ that is impossible too (reasoning is similar to the previous item).

Further, let u be an arbitrary element in $D(h)$ then taking into account properties 1 and 2 of $D_0(h)$ we can conclude the following statement: for each $\pi \in u \cdot \Sigma^\omega$ there exists the unique $1 \leq i(\pi) \leq m$ such that $\pi \in u_{i(\pi)} \cdot \Sigma^\omega$. As above the assumption $|u| \neq |u_{i(\pi)}|$ contradicts the equality $C(D(h)) = D(h)$ and the equality $|u| = |u_{i(\pi)}|$ means $u = u_{i(\pi)}$. Hence, $D(h) = D_0(h)$ and the direct statement of the theorem is proved.

The converse statement of the theorem is evident. □

Remark 1. Theorem 1 is not complicated but it grounds the following important alternative: either a handler is able to provide a response to an infinite number of complex events and, in such a case, it can not provide processing of any event stream, or the handler is able to provide processing of any event stream and, in such a case, its behaviour consists in responding to a finite number of complex events.

6 CEP-machines and Pre-automata

As it noted above, the concept of a pre-automaton is closely related with the concept of a CEP-machine. This section is devoted to explaining of the mentioned relation.

Definition 9. (see [6]). *A triple (X, Σ, μ), where X is a set, Σ is a finite alphabet, and $\mu : X \times \Sigma^* \dashrightarrow X$ is a partial mapping, is called a pre-automaton if the following conditions hold*

1. *the equality $\mu(x, \varepsilon) \downarrow x$ holds for all $x \in X$;*
2. *if the assertions $\mu(x, u) \downarrow$ and $\mu(\mu(x, u), v) \downarrow$ are true for some $x \in X$ and $u, v \in \Sigma^*$ then the assertion $\mu(x, uv) \downarrow \mu(\mu(x, u), v)$ is true too;*
3. *if the assertions $\mu(x, u) \downarrow$ and $\mu(x, uv) \downarrow$ are true for some $x \in X$ and $u, v \in \Sigma^*$ then the assertion $\mu(\mu(x, u), v) \downarrow \mu(x, uv)$ is true too.*

In this context μ is called a transition function.

Now we describe a manner to associate a pre-automaton with a CEP-machine. The origin point for our construction is a CEP-machine $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$. We will find the target pre-automaton as the triple $\widehat{\mathcal{M}} = (H, \Sigma, \mu)$ such that the partial mapping $\mu : H \times \Sigma^* \dashrightarrow H$ satisfies the following condition

$$\begin{array}{l} \text{for any } h \in H,\ w \in D(h),\ \text{and } \pi \in \Sigma^\omega \\ \quad \text{the conjunction of } \mu(h, w) \downarrow \ \text{and } S\langle h, w \cdot \pi\rangle \downarrow \langle \mu(h, w), \pi\rangle \text{ is true} \end{array} \tag{2}$$

where S is the evolutionary operator of the CEP-machine $\mathcal{M}$.

Theorem 2. *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ be a CEP-machine and $\mu : H \times \Sigma^* \dashrightarrow H$ be defined in the following manner*

1. *$\mu(h, \varepsilon) = h$ for any $h \in H$;*
2. *for any $h \in H$, $u \in D(h)$, and $v \in \Sigma^*$ the truth of the assertion $\mu(\delta(h(u)), v) \downarrow$ implies that $\mu(h, uv) \downarrow$ is true and in this case $\mu(h, uv) = \mu(\delta(h(u)), v)$;*
3. *$\mu(h, w) \uparrow$ for all other cases,*

then the triple $\widehat{\mathcal{M}} = (H, \Sigma, \mu)$ is a pre-automaton and the mapping μ satisfies (2).

To prove the Theorem 2 the following lemma is needed.

Lemma 1. *Let μ be as in Theorem 2, $h \in H$, and $w \in \Sigma^+$ be a word such that $\mu(h, w) \downarrow$ then there exists the unique alternating sequence $h_0 \in H$, $u_0 \in \Sigma^+$, $h_1 \in H$, $u_1 \in \Sigma^+$, ..., $u_{n-1} \in \Sigma^+$, $h_n \in H$ such that*

1. *$w = u_0 \dots u_{n-1}$;*
2. *$u_i \in D(h_i)$ for all $i = 0, \dots, n-1$;*
3. *$h_0 = h$ and $h_{i+1} = \mu(h_i, u_i)$ for all $i = 0, \dots, n-1$.*

Proof. To prove the lemma let us consider the following algorithm.

1: let us assign $h_0 \leftarrow h$, $w_0 \leftarrow w$, and $i \leftarrow 0$;
2: if $w_i \in D(h_i)$ then assign $h_{i+1} \leftarrow \mu(h_i, w_i)$, $u_i \leftarrow w_i$, and halt;
3: choose $u_i \in D(h_i)$ and $w_{i+1} \in \Sigma^+$ so as to satisfy the equality $w_i = u_i w_{i+1}$;
4: let assign $h_{i+1} \leftarrow \mu(h_i, u_i)$ and $i \leftarrow i + 1$;
5: go to item 2.

Taking into account that $\mu(h, w) \downarrow$ and the definition of μ one can obtain that for any i either $w_i \in D(h_i)$ and algorithm terminates or w_i can be represented in accordance to item 3. In the last case the definition of μ guaranties that $\mu(h_i, w_i) \downarrow$. Moreover, the inequality $|w_i| < |w_{i-1}|$ holds.

Thus, an algorithm execution is terminated because each its step decreases length of the current word w_i. After such a termination we evidently obtained the required alternating sequence. □

Corollary 3. *Any word $w \in \Sigma^+$ such that $\mu(h, w) \downarrow$ can be uniquely represented in the form $w = uv$ where u is a word satisfying the condition $\mu(h, u) \downarrow$ and $v \in D(\mu(h, u))$.*

Proof (of Theorem 2). Primarily, let us check that μ satisfies conditions 1 – 3 of Definition 9. The validity of condition 1 is evident.

To check condition 2 of the theorem assume that $\mu(h, u) \downarrow$ and $\mu(\mu(h, u), v) \downarrow$. Using Lemma 1 one can construct the alternating sequence

$$h_0 = h, u_0, \dots, u_{m-1}, h_m = \mu(h, u), u_m, \dots, u_{m+n-1}, h_{m+n} = \mu(\mu(h, u), v)$$

such that the sequence $h_0 = h, u_0, \dots, u_{m-1}, h_m = \mu(h, u)$ is the sequence for $\mu(h, u)$ and $h_m = \mu(h, u), u_m, \dots, u_{m+n-1}, h_{m+n} = \mu(\mu(h, u), v)$ is the sequence for $\mu(\mu(h, u), v)$. Thus $u = u_0 \dots u_{m-1}$, $v = u_m \dots u_{m+n-1}$, and $uv = u_0 \dots u_{m-1} u_m \dots u_{m+n-1}$. It is evident that conditions 2 and 3 of Lemma 1 hold therefore the constructing sequence is the sequence for $\mu(uv, h)$. This means that $\mu(h, uv) \downarrow \mu(\mu(h, u), v)$.

To check condition 3 of the theorem is similarly to check condition 2.

The validity of (2) is evident. □

Definition 10 (the dual pre-automaton for a CEP-machine). *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta)$ be a CEP-machine and $\widehat{\mathcal{M}} = (H, \Sigma, \mu)$ be the pre-automaton determined by Theorem 2 then $\widehat{\mathcal{M}}$ will be called the dual pre-automaton for the CEP-machine $\mathcal{M}$.*

We can carry out the converse construction too and associate with any pre-automaton $\mathcal{P}$ a CEP-machine $\mathcal{M}$ so as to satisfy the equality $\mathcal{P} = \widehat{\mathcal{M}}$.

Proposition 2. *Suppose that a pre-machine* $\mathcal{P} = (X, \Sigma, \mu)$ *is given and let us define the quadruple* $\widehat{\mathcal{P}} = (\Sigma, X, H, \delta)$ *such that*

$$H = \{h_x : \Sigma^+ \dashrightarrow X \mid x \in X\} \text{ where}$$

$$D(h_x) = \{w \in \Sigma^+ \mid \mu(x, w) \downarrow \text{ and if } w = uv \wedge \mu(x, u) \downarrow \text{ then } v = \varepsilon\},$$
$$h_x(w) = \mu(x, w) \text{ for } w \in D(h_x);$$

$$\delta(x) = h_x .$$

Then $\widehat{\mathcal{P}}$ *is a CEP-machine and* $\widehat{\widehat{\mathcal{P}}}$ *is isomorphic to* $\mathcal{P}$.

Proof. Let $\widehat{\mu} : H \times \Sigma^* \dashrightarrow H$ be the transition function for the pre-automaton $\widehat{\widehat{\mathcal{P}}}$. Then for $w \in D(h_x)$ we have $\widehat{\mu}(h_x, w) = \delta(h_x(w)) = h_{\mu(x,w)}$. Extending the mapping $\widehat{\mu}(x, w)$ for $w \in \Sigma^*$ in according with the conditions of Theorem 2 we obtain $\widehat{\mu}(h_x, w) \downarrow$ if and only if $\mu(x, w) \downarrow$ and in this case $\widehat{\mu}(h_x, w) = h_{\mu(x,w)}$. Hence, the mapping such that $x \mapsto h_x$ is an isomorphism of pre-automata [6]. □

7 Synthesis of CEP-machines with Specified Behaviour

The real engineering practice requires methods to provide improving of development processes, in particular, for synthesis of CEP-systems. The key notion for this synthesis problem is the notion of "a CEP-machine behaviour". In this section we propose a generalization for the notion "a behaviour", which was defined for automata, and solve the corresponding synthesis problem.

In this context the first problem is to set a method to specify a CEP-machine behaviour. We suggest that to specify the behaviour of the CEP-machine and to specify a mapping associating an input event stream with a sequence of responses of the CEP-machine are the solutions of the same problem. These considerations lead us to the following definitions.

Definition 11 (an initial CEP-machine). *A quintuple* $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta, h_*)$, *where* $h_* \in H$, *is called an initial CEP-machine if the quadruple* $(\Sigma, \mathrm{A}, H, \delta)$ *is a CEP-machine.*

In other words, an initial CEP-machine is a CEP-machine with the marked handler, which is called an initial handler. This handler is chosen as the active handler under initialization the CEP-machine.

Definition 12 (the response function of a CEP-machine). *For an arbitrary initial CEP-machine* $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta, h_*)$ *let us define the partial mapping* $\beta : \Sigma^+ \dashrightarrow \mathrm{A}$ *in the following manner*

1. *the domain of* β *is defined by the equality* $D(\beta) = \{w \in \Sigma^+ \mid \mu(h_*, w) \downarrow\}$ *where* μ *is the transition function of the pre-automaton* $\widehat{\mathcal{M}}$;

2. for $w \in D(\beta)$ using Corollary 3 represent $w = uv$ such that $\mu(h_*, u) \downarrow$ and $v \in D(\mu(h_*, u))$ then define $\beta(w) = \mu(h_*, u)(v)$.

The constructed mapping β is called the response function of the CEP-machine $\mathcal{M}$.

In the following proposition the main property of response functions is established.

Proposition 3. *Let $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta, h_*)$ be an initial CEP-machine and $\beta : \Sigma^+ \dashrightarrow \mathrm{A}$ be its response function then the following condition is true for any $u, v \in D(\beta)$ such that $\beta(u) = \beta(v)$ and any $w \in \Sigma^*$*

$$uw \in D(\beta) \text{ implies } vw \in D(\beta) \text{ and } \beta(uw) = \beta(vw)\,. \tag{3}$$

Proof. Let $u, v \in D(\beta)$ and $\beta(u) = \beta(v)$. Taking into account the last equality and representation $\mu(h_*, u) = \delta(\beta(u))$ one can conclude that $\mu(h_*, u) = \mu(h_*, v)$. If $uw \in D(\beta)$ then $\mu(h_*, uw) \downarrow$ and using condition 3 of Definition 9 we obtain that $\mu(\mu(h_*, u), w) \downarrow$ and, hence,

$$\mu(h_*, uw) = \mu(\mu(h_*, u), w) = \mu(\delta(\beta(u)), w) = \mu(\delta(\beta(v)), w) = \mu(\mu(h_*, v), w)\,.$$

Therefore, $\mu(\mu(h_*, v), w) \downarrow$ and using condition 2 of Definition 9 we obtain that $\mu(h_*, vw) \downarrow$ i.e. $vw \in D(\beta)$.

Further, $\beta(u) = \beta(v)$ implies $\mu(h_*, u) = \mu(h_*, v)$ and we can use the decomposition from Corollary 3 for $h = \mu(h_*, u) = \mu(h_*, v)$ and w. Let $w = w'w''$ be the corresponding decomposition then $\mu(h, w') \downarrow$ and $w'' \in D(\mu(h, w'))$. Definition 9 and the equality $\beta(u) = \beta(v)$ ensure that the statements $\mu(h_*, uw') \downarrow$ and $\mu(h_*, vw') \downarrow$ are true both and the equality $\mu(h_*, uw') = \mu(h_*, vw')$ hold. Therefore, we have

$$\begin{aligned}\beta(uw) = \mu(h_*, uw')(w'') = \mu(\mu(h_*, u), w')(w'') = \mu(h, w')(w'') = \\ \mu(\mu(h_*, v), w')(w'') = \mu(h_*, vw')(w'') = \beta(vw)\,.\end{aligned}$$

Thus, proof is complete. □

The converse statement to the previous proposition is true too.

Theorem 3. (about synthesis of a CEP-machine). *Let Σ and A be finite alphabets and $\beta : \Sigma^+ \dashrightarrow \mathrm{A}$ be a partial mapping satisfying (3) then there exists an initial CEP-machine $\mathcal{M} = (\Sigma, \mathrm{A}, H, \delta, h_* \in H)$ whose response function coincides with β.*

Proof. Let us prove the theorem in two stages. Primarily construct a CEP-machine $\mathcal{M}_\beta$ using the mapping β and then prove that its response function coincides with β.

To realise the first stage of the proof let us consider on the set $D(\beta) \subset \Sigma^*$ the binary relation $\equiv_\beta$ defined by the next way: $u \equiv_\beta v$ means that $\beta(u) = \beta(v)$ is true. It is evident that this relation is an equivalence.

Now let us consider $H = \{h_*\} \bigcup \{h_{[u]_\beta} \mid u \in D(\beta)\}$ where $[u]_\beta$ is the equivalence class of u with respect to equivalence $\equiv_\beta$. Taking into account that $\beta|[u]_\beta$ is a constant mapping for each $u \in D(\beta)$ one can conclude that $|H| \leq |\mathrm{A}| + 1$. Therefore H is a finite set.

Let us define $h_*(w) \downarrow$ if $w \in C(D(\beta))$ and in this case $h_*(w) = \beta(w)$. The definition of $C(D(\beta))$ ensure that h_* is a handler.

Further, for $u \in D(\beta)$ define that $h_{[u]_\beta}(w) \downarrow$ if $uw \in C(D(\beta))$. Condition (3) ensures that this definition does not depend on a choice of $u' \in [u]_\beta$. Moreover, in the considered case the equality $\beta(uw) = \beta(u'w)$ holds. Therefore, the formula $h_{[u]_\beta}(w) = \beta(uw)$ defines $h_{[u]_\beta}$ correctly. To check that $h_{[u]_\beta}$ is a handler let assume that $w'w'' \in D(h_{[u]_\beta})$ and $w' \in D(h_{[u]_\beta})$ then $uw'w'' \in C(D(\beta))$ and $uw' \in C(D(\beta))$ are true both. But the last conjunction means that $w'' = \varepsilon$ by the definition of $C(D(\beta))$. Hence, $h_{[u]_\beta}$ is a handler.

To define $\delta : \mathrm{A} \to H$ note that for any $a \in \beta(D(\beta))$ the equality $\beta(u) = a$ uniquely determines $[u]_\beta$, hence we can define δ on $\beta(D(\beta))$ by the next way $\delta(a) = [u]_\beta$ where $\beta(u) = a$. On the set $\mathrm{A} \setminus \beta(D(\beta))$ we can define the mapping δ in arbitrary way, e.g. setting $\delta|(\mathrm{A} \setminus \beta(D(\beta))) = h_*$.

Thus, we have constructed the CEP-machine $\mathcal{M}_\beta = (\Sigma, \mathrm{A}, H, \delta)$.

To complete proof it is need to check that the response function of the CEP-machine $\mathcal{M}_\beta$ coincides with β. Denote by $\widehat{\beta}$ the response function of the CEP-machine $\mathcal{M}_\beta$ and by μ the transition function of the pre-automaton $\widehat{\mathcal{M}_\beta}$.

By definition $D(\widehat{\beta})$ coincides with the set $\{w \in \Sigma^+ \mid \mu(h_*, w) \downarrow\}$. For such a word w we can apply Lemma 1 and construct the alternating sequence $h_0 = h_*$, u_0, $h_1 = h_{[u_0]_\beta}$, $\ldots$, $h_{n-1} = h_{[u_0 \ldots u_{n-2}]_\beta}$, u_{n-1}, $h_n = h_{[u_{n-1}]_\beta}$ such that $u_0 \ldots u_i \in D(h_i)$ for $i = 0, \ldots, n-1$ and $w = u_0 \ldots u_{n-1}$. Hence, $\widehat{\beta}(w) = \beta(u_0 \ldots u_{n-2} u_{n-1}) = \beta(w)$. □

8 Complex Event Processing and Computability

Above we were considering the general properties of CEP-machines and the theory was being constructed similar to finite automata theory. However, problems associated with CEP-machines are not automatically solvable in contrast to problems associated with finite automata. Therefore computability for CEP-machines is a special problem requiring its studying.

For use the described above approach to solve problems of designing software systems, we restrict the class of handlers, namely, we consider computable handlers only. Such a restriction is a key to provide processing of event streams using a computational system. In the most general form the schema of complex event processing is represented in Fig. 2. We should stress that there is a potential problem in this schema. It is connected with applying the method handle(). This method sequentially applies the corresponding handler to the current buffer contents. But a computable function does not identify data which does not belong to the domain of this function in general case. Therefore an attempt to execute this method can lead to its infinite running. In this section we show that the indicated anomaly can be eliminated.

Theorem 4. (about Decidability of the Halting Problem for Handlers). *Let* Σ *and* A *be finite alphabets,* $h : \Sigma^+ \dashrightarrow \mathrm{A}$ *be a computable handler, and* $\pi \in \Sigma^\omega$ *be an event stream then the predicate* $M(\pi) \cong \exists n\ h(\pi_{[1..n]}) \downarrow$ *is decidable relative to* $g(n) = \pi_{[1..n]}$.

Proof. As it is established in computation theory the domain of a computable function is a semidecidable set. Hence, the predicate $w \in D(h)$ is semidecidable. Such a predicate is represented as $\exists t\ R(w,t)$ for some decidable predicate R (see [5, p.114,Theorem6.4]). As R one can be chosen, for example, the predicate "a program to compute h executes at most t commands and is halted when the input is w".

Let us $\gamma : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be the mapping defined by the formulae

$$\begin{aligned} \gamma_1(n) &= \mu\, k\ \left(n < \frac{k(k+1)}{2}\right) - 1\ , \\ \gamma_2(n) &= n - \gamma_1(n) \end{aligned}$$

where $\mu\, k\ (M(k,n))$ means the minimal value of k that satisfies the condition $M(k,n)$. It is evident that $\gamma = (\gamma_1, \gamma_2)$ is a computable bijection.

Now let us consider the following algorithm

1: let assign $n \leftarrow 1$;
2: let assign $m, t \leftarrow \gamma(n)$;
3: if $R(g(m), t)$ then return $g(m)$ and halt;
4: let assign $n \leftarrow n + 1$;
5: go to item 2.

Taking into account that the domain of h is prefix-free one can easily check the following sentence: either there exists m such that $\pi_{[1..m]} \in D(h)$ and the algorithm finds it or the algorithm is executing infinitely. To complete proof it is sufficient to transform this informal algorithm into the corresponding Turing machine. □

Corollary 4. *If all handlers of a CEP-machine are computable then this machine can be algorithmic realised.*

Proof. Indeed, we can find $\pi_{[1..m]} \in D(h)$ using the algorithm of Theorem 4 if such $\pi_{[1..m]}$ exists and then calculate $h(\pi_{[1..m]})$. □

9 Conclusion

In the paper a study of complex event processing systems has been conducted. This study has based on known examples of software solutions for such systems and results of a generalisation of these examples. Therefore some survey of complex event processing systems has been taken as a basis of the research.

Abstracting of invariants for these examples has led to the formulation of a mathematical model, which has been called a CEP-machine. This model has been

described from two points of view: structural and behavioural. To specify the behavioural part of the model a number of formal notions has been introduced.

Basing on these formal objects the elementary properties of a CEP-machine have been studied. Theorem 1 expresses the principal result of such a study. It consists in that either an event handler is able to provide a response to an infinite number of complex events and, in such a case, it can not provide processing of any event stream, or the handler is able to provide processing of any event stream and, in such a case, its behaviour consists in responding to a finite number of complex events.

Duality between of CEP-machines and pre-automata (see Theorem 2 and Proposition 2) is the another important result obtained in the paper. It provides some technique for studying of a CEP-machine behaviour.

The important problem for applications is the CEP-machine synthesis problem with the given behaviour, which is specified by a response function. Theorem 3 describes and gives grounds for the method to solve this problem.

Finally, Theorem 4 gives a solution for the problem about the algorithmic realisability of a CEP-machine. This theorem has established that the computability condition of machine handlers is sufficient for the algorithmic realisability of the CEP-machine.

Unfortunately, Theorem 3 does not establish any universal properties for the constructed CEP-machine in contrast to the analogous construction for finite state machine. Therefore the question about universality of this construction for CEP-machines is open now.

The following statement "If a response function is computable then the corresponding CEP-machine is algorithmical realisable" is an example of the next open problem.

We should indicate yet another direction of analysing of CEP-machines. The necessity of this direction follows from the property of CEP-machines marked above: if a CEP-machine is non-trivial then there exist event streams that can be processed by this machine. Hence, studying of such anomalies is an important problem. This studying would be carried out in the probabilistic context that would be provide mean estimations for behavioural anomalies of CEP-machines.

References

1. Babcock, B., Datar, M., Motwani, R.: Load shedding for aggregation queries over data streams. In: ICDE'04: Proceedings of the 20th International Conference Data Engineering, pp. 350–361. IEEE CS, Washington (2004)
2. Bates, J.: Secrets Revealed: Trading Tools Uncover Hidden Opportunities. Global Trading, 14 May 2012. http://fixglobal.com/home/secrets-revealed-trading-tools-uncover-hidden-opportunities/
3. Crosman, P.: Aleri, Ravenpack to Feed News into Trading Algos. Wall Street & Technology (2009). http://www.wallstreetandtech.com/data-management/aleri-ravenpack-to-feed-news-into-tradin/217500395
4. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. CSUR **44**(3), 15 (2012). ACM Press

5. Cutland, N.: Computability: an introduction to recursive function theory. Cambridge University Press, London (1980)
6. Dokuchaev, M., Novikov, B., Zholtkevych, G.: Alg. Discr. Math. 11(2), 51–63 (2011)
7. Eckert, Michael, Bry, François, Brodt, Simon, Poppe, Olga, Hausmann, Steffen: A CEP babelfish: languages for complex event processing and querying surveyed. In: Helmer, Sven, Poulovassilis, Alexandra, Xhafa, Fatos (eds.) Reasoning in Event-Based Distributed Systems. SCI, vol. 347, pp. 47–70. Springer, Heidelberg (2011)
8. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications, Stamford (2010)
9. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley, New York (2001)
10. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, Boston (2002)
11. Luckham, D.C.: Event Processing for Business: Organizing the Real-Time Enterprise. John Wiley & Sons Inc., Hoboken (2012)
12. Mikhailova, I., Novikov, B., Zholtkevych, G.: Protoautomata as models of systems with data accumulation. In: Ermolayev, V., et al. (eds.) Proceedings of the 9th International Conference ICTERI 2013, pp. 582–589. CEUR-WS (2013)
13. Michelson, B.: Event-Driven Architecture Overview. Patricia Seybold Group, Boston (2006)
14. Novikov, B., Perepelytsya, I., Zholtkevych, G.: Pre-automata as mathematical models of event flows recognisers. In: Ermolayev, V., et al. (eds.) Proceedings of the 7th International Conference ICTERI 2011, pp. 41–50. CEURS-WS (2011)
15. Perepelytsya, I., Zholtkevych, G.: On some class of mathematical models for static analysis of critical-mission asynchronous systems. Syst. ozbr. ta viysk. tehn. **27**(3), 60–63 (2011)
16. Perepelytsya, I., Zholtkevych, G.: Hierarchic decomposition of pre-machines as models of software system components. Syst. upravl. navig. i zv'iazku. **20**(4), 233–238 (2011)