UDC 004.042 + 519.713.2

G.M. Zholtkevych, V.V. Dorozhinsky, A.V. Khadikov

V.N. Karazin Kharkiv National University, Kharkiv

REGULAR EVENT PROCESSING AND MACHINE LEARNING CORRECTNESS VERIFICATION

Currently the concepts of Event Driven Architecture and Machine Learning become widely used in the information systems. In the paper the authors propose an approach to combine these two concepts to be able to overcome logical complexity in the design of the information systems based on Event Driven Architecture. The paper considers some machine learning method for the component of the system based on this architecture using rigorous mathematical formulations. In particular, a problem for the method correctness verification is studied. The algorithms and steps for the computer experiments are introduced in details. Finally, further research directions are proposed.

Keywords: *Machine Learning, Event Processing, acceptor, regular language, prefix-free language*

Introduction

In today's Information Technology world, many applications are reactive in the sense that they respond to the detection of events. These applications exist in many domains and are very useful for data processing. Sometimes these applications should be able to react not to a single event but to the sequences of events. In this case a Complex Event Processing (CEP) approach is used [6]. Usually such data processing systems design is based on the Event Driven Architecture (EDA). Mathematical fundamentals to analyze component behaviour of EDA based systems can be found in the papers [1, 9, 10]. The practically important class of event detectors for systems built on EDA has been defined and considered in [2].

While the event types usually are known, in most cases it is impossible to determine all their combinations during system design and development. As a result the processing system is unable to generate a proper reaction on the event combinations that were not taken into account during its design. To overcome this problem the machine learning methods can be used to make a processing system adaptable during runtime. Machine learning explores the study and construction of algorithms that can learn from and make predictions on data [5]. Such algorithms operate by building a model from an example training set of input observations in order to make data-driven predictions or decisions expressed as outputs rather than following strictly static program instructions.

Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms is unfeasible. Also it is sometimes conflated with data mining,[7] where the latter sub-field focuses more on exploratory data analysis and is known as unsupervised learning [3]. In the paper we are considering a learning method for a component of the EDA based system and describing in details correctness verification procedure for the method.

1. Basic Notation and Definitions

In this section the CEP-machine model proposed in [9] is described. We will consider subclass of CEP-machines called a Regular CEP-machine developed in [2]. Below we will use the following notation:

$f : X \xrightarrow{\text{partial}} Y$ denotes that f is a partial mapping from X into Y;

$f(x)\uparrow$ denotes that $f(x)$ is not defined for the member x of X;

$f(x)\downarrow$ denotes that $f(x)$ is defined for the member x of X;

$f(x)\downarrow = y$ denotes that $f(x)\downarrow$ and $y = f(x)$ for the member y of Y;

ε denotes the empty (zero-length) sequence;

X^+ denotes the set of all non-empty finite sequences composed of elements of X;

X^* denotes the set $\{\varepsilon\} \bigcup X^+$;

X^ω denotes the set of all infinite sequences composed of elements of X;

X^∞ denotes the set $X^* \bigcup X^\omega$;

$|x|$ denotes the length of the finite sequence x;

$x[0]$ denotes the first element of a finite or infinite sequence x;

$x[1:]$ denotes the sequence obtained by removing the first element of the sequence x.

Let us start from some basic definitions needed to describe the mathematical model of a CEP-machine:

Definition 1. Define a finite set X that is called an alphabet and its elements are called symbols. Further, we will call a set $L \subseteq X^*$ a language in this context.

The symbols are interpreted as a prime messages that inform about the corresponding elementary events that have happened. Some of these finite symbol sequences carry information about the complex events. Below we will call these sequences as events. In contrast, other finite symbol

sequences do not carry information about any complex event. Below we will call them words.

The sets of Complex Events should respect the certain conditions. The most important one is that any stream of elementary events is uniquely subdivided into a series of complex events by directed viewing the stream from left to right. This condition leads to the following definition.

Definition 2. Let L be a language with the alphabet X then L is prefix- free if for any $u \in L$ and $v \in X^*$ the assumption $uv \in L$ implies the equality $v = \varepsilon$.

Now let us fix that any set of complex events related to the system are prefix-free. Next, let us briefly recall the principal idea of the used model.

Definition 3. (Regular handler). A handler $h : X^+ \xrightarrow{\text{partial}} Y$ is called regular if there exist

— some finite set Z with the marked element $z_0 \in Z$ and

— some mapping $\delta : Z \times X \to Z \bigcup Y$

such that for any $u \in X^+$ and $y \in Y$ the condition $h(u)\downarrow = y$ is fulfilled iff there exist $z_1, ..., z_{|u|-1} \in Z$ such that

$z_{i+1} = \delta(z_i, u[i])$ for $0 \le i < |u| - 1$ and

$y = \delta(z_{|u|-1}, u[|u|-1])$.

In this case we say that the handler h is realized by the triple (Z, z_0, δ).

Definition 4. (Regular CEP-machine Structure). Any Regular CEP- machine is a quintuple $M = (X, Y, H, h_0, \alpha)$ with the following constituents:

— the finite set that is alphabet of atomic messages X;

— the finite set that is alphabet of machine responses Y respectively;

— the finite set of regular handlers H;

— the initial handler $h_0 \in H$;

— the response function $\alpha : Y \to H$.

The Algorithm 1 determines behaviour of any Regular CEP-machine.

Algorithm 1. Operational model of a Regular CEP-machine

1 def run(M,s):

Require: the studied Regular CEP-machine $M = (X, Y, H, h_0, \alpha)$ and some stream of elementary events $s \in X^\omega$.

Ensure : printing of the corresponding response stream

```
2   handler, buff = h0,[]
3   while True:
4       new_event, s = s[0], s[1:]
5       buff.append(new_event)
6       if handler(buff)↑: continue
7       else:
8           response = handler (buff)
9           print(response)  # printing of the
            current response
10          handler, buff = α(response),[]
```

2. Regular CEP-machine Learning Method

In this section we recall a concept of the machine learning method that was proposed in the authors previous paper [11]. The Regular CEP- machine learning problem can be decomposed into series of the Regular Handler learning problems. Since each Regular Handler has only one possible response "accepted", below we will call it "a regular acceptor". Therefore, a Regular Acceptor learning problem can be formulated in the following way.

Problem. Let $E = \{u_1, ..., u_M\} \subset X^+$ be a finite prefix-free set of events and $C = \{v_1, ..., v_N\} \subset X^+$ be a finite set of words such that $E \bigcap C = \varnothing$ then we interpret elements of set E as examples and elements of set C as counterexamples;

we need to find a regular acceptor $h : X^+ \xrightarrow{\text{partial}} \{\text{accepted}\}$ such that the following conditions are fulfilled:

1. $h(u_i)\downarrow = \text{accepted}$ for all $0 \le i < M$;
2. $h(v_i)\uparrow$ for all $0 \le i < N$; and
3. The regular acceptor is minimal. Namely, the corresponding set Z has the least number of elements among all possible.

Now let us discuss the interrelation between regular acceptors and finite state machines. In particular, consider for any regular acceptor $h : X^+ \xrightarrow{\text{partial}} Y$ that realized by the triple (Z, z_0, δ) the machine $(Q = Z \bigcup Y \{q_{trap}\}, X, \bar{\delta} : Q \times X \to Q, q_0 = z_0, Q_{accept} = Y)$ where $q_{trap} \notin Z \bigcup Y$ and $\bar{\delta}(y, x) = q_{trap}$; $\bar{\delta}(q_{trap}, x) = q_{trap}$ for any $x \in X$ and $y \in Y$ then the regular language accepted by this machine coincides with the set of events accepted by the regular acceptor. General automata theory guarantees that the built machine can be uniquely minimized without changing language that is accepted by the machine. Note, that the corresponding minimal machine has only one non-acceptable state q such that $\delta(q, x) = q$ for any $x \in X$. This state is called the trap and denoted as trap. Further, we require $\delta(q, x) = \text{trap}$ for any acceptable state q and any $x \in X$. Now we can describe the pro-

posed learning method. The general view of the method to generate an acceptor is defined by Alg. 2. This method defines the series of redirections for acceptor transitions leading into the trap starting with the minimal acceptor for the set of examples.

Algorithm 2. Specification of the learning method

```
1 def learning_method(E,C):
    Require: the finite prefix-free set of events E
    Ensure: the required acceptor
2         do that:
3          initiate the learning process by applying function init to the set E and denoting the result by acceptor # initialize the set of transitions that cannot be redirected
4         frozen_transitions = set()
5         while halting condition is not fulfilled:
6            do that:
7               modify acceptor by redirecting a transition leading into the trap and minimize the resulting acceptor if the redirected transition is not in frozen_transitions
8            do that:
9               check that acceptor is admissible in the sense that it does not accept any word from C
10           if the checking is successful: continue
11              else:
12                 do that:
13                 roll-back the modification and add the redirected transition into frozen_transitions
```

The algorithm uses two functions: init - that generates initial acceptor using a set of examples E; and modify - that modifies the acceptor so that it "learns" a new event.

These functions are specified separately.

To complete the specification of the learning method the algorithms for the function init (see item 3 of Alg. 2) and for function modify (see item 7 of Alg. 2) need to be described.

To generate the minimal initial acceptor with the init function the following algorithm is used:

1. states of the acceptor are defined recursively as special sets of words;
2. we choose the set E as z_0 and add it to Z;
3. we choose the empty set as trap;
4. if for $x \in X$ in E there is not a word with the first symbol x then assign $\delta(z_0, x) = \text{trap}$ else the set $\{u \in X^* \,|\, xu \in E\}$ add to Z;
5. repeat recursively this consideration for all member of Z until Z is stabilized;
6. denote the set $\{\varepsilon\}$ by accepted.

The acceptor obtained in this manner is assigned as result of the function init.

The function modify is developed using the following algorithm:

1. Define $z = z_0$ set of acceptable events and $w \in X^*$ a new word will be learned by the acceptor.
2. while $w \neq \varepsilon$:
3. add an event w if not exists into z
4. $x, w = w[0], w[1:]$
5. If exists a transition x for the pair $(z, z_x \in Z)$ then $z = z_x$. where $z_x = \{u \in X^+ \,|\, \forall v \in z : v = xu\}$ is a set of suffixes obtained from the set z by removing symbol x.
6. Else if exists a set of suffixes $z_w \in Z$ such that $w \in z_w$ then for the pair (z, z_w) if not in frozen_transitions add a transition marked by x and set $z = z_w$
7. Else add a new state $z_{new} \in Z : z_{new} = \{w\}$ and for the pair (z, z_{new}) add a transition marked by x and set $z = z_{new}$
8. Done

To select a transition for redirection we use the following simple remark: the minimal regular acceptor has at most one state that is an attractor, i.e. any transition that goes out from this state has it as a target. Moreover, if this acceptor accepts a finite language then the existence of the attractor is guaranteed. Further, to minimize the new acceptor we use standard Hopcroft's algorithm [4].

3. Learning Method Correctness Verification

In this section we present in details an experiment to verify validity of the Regular Acceptor learning method. The Alg. 3 specifies the schema of the experiment. The key point in the verification procedure is a test_acceptor random generation. Later this acceptor is used to define the sets of examples E and counterexamples C.

Algorithm 3. The schema of the computer experiment

```
1  for _ in range(given_number_of_experiments):
2  do that:
3  | generate randomly a regular acceptor test_acceptor
4  do that:
5  | generate randomly sets E and C using test_acceptor
6  acceptor' = learning_method( E , C )
7  do that:
8  I compare acceptor' and test_acceptor
```

To generate this test_acceptor at first we generate a syntactic tree of the regular expressions and then convert it to the regular acceptor. To implement the mentioned experimental schema we use language Python with libraries "scipy" and "numpy" [8]. Particularly, all random choices have been provided by the standard function random.choice contained in the library "numpy".

To randomly generate a regular expression the following schema has been used.

Data structures: two data structures are used to generate a regular expression, namely,

1. the following dictionary (in the terms of language Python) is used to hold constant data

```
model_frame = {
      'tokens':  ( # this tuple contains a list of
                   # used tokens
),
'functors':  (# this tuple contains operations
              # and their arities
('star', 1),# Kleene's star
('concatenation', 2)),
('union', 2)
)
}
```

2. The following recursive structure is used to represent the syntactic tree of a regular expression.

```
expression = token
           | (functor, expression, ...)
```

where number of expressions after a functor equals an arity of this functor.

For example, only one expression follows the functor "star".

One can see that an expression is represented by a tree. Each leaf of this tree is marked by a token and each internal node of the tree is marked by a functor. Moreover, the number of children for an internal node equals the arity of the corresponding functor.

Schema of tree random generation: the recursive structure of the syntactic tree that represents some regular expression indicates the way of this tree random generation (see Alg. 4).

Algorithm 4. A tree random generation

```
1  def create_tree(depth = 0):
2 global model_frame
3 do that:
4  | decide whether the tree root is an internal node
or a leaf
5 if the tree root is a leaf:
6      do that:
7       | choose randomly token
8      return token
9 else:
10     do that:
11      | choose randomly functor
12     arity, temp = model_frame[ 'functors' ][
functor ][1], [functor]
13     for _ in range(arity) :
temp.append(create_tree(h+l))
14     return tuple(temp)
```

To make decision whether the root of the current subtree is an internal node or a leaf (item 4 of Alg. 4) we propose to use the following function $p(n)$, that determines the conditional probability to mark the current node as internal if its depth is equal to n if $0 \le n < \mu$ or if $n \ge \mu$^

$$p(n) = \begin{cases} 1-(n/\mu)^2/2, & 0 \le n < \mu; \\ \exp(1-n/\mu)/2, & n \ge \mu. \end{cases}$$

The described generation method provides constructing well-balanced syntactic trees, but it does not control presence of disbalance for marking. Therefore, to correct possible disbalances we propose the next way to perform item 7 and item 11 of Alg. 4: to use a standard function random.choice, which is contained in the package "random" of the library "numpy", for choice an item from the list $[i_1, i_2, ..., i_k]$ under the condition that probability to choose i_j is inversely proportional to number of the realizations for this choice. This correction ensures the "uniformness" of distribution for markings of leaves and internal nodes.

Now, the following procedure called TransformSyntaxTreeToFSM(tree) can be used to convert a syntactic tree to the finite state machine. Next, the obtained finite state machine can be transformed to the regular acceptor by joining all acceptable states.

1. if SyntaxTreeNode.GetType() == leaf then:
   (a) symbol = SyntaxTreeNode.GetValue();
   (b) FSM.AddStartState(q_0);
   (c) FSM.AddTerminalState(q_{symbol});
   (d) FSM.AddArrow(q_0, symbol, q_{symbol});
   (e) Return FSM;
2. else if SyntaxTreeNode.GetType == 'star' then:
   (a) childNode = SyntaxTreeNode.GetChild();
   (b) FSM = TransformSyntaxTreeToFSM(childNode);
   (c) for each terminal arrow
arrow ∈ FSM.GetAllTerminalArrows():
   (d) arrow.replaceEndStateTo(q_0);
   (e) end for each loop;
   (f) Return FSM;
3. else if SyntaxTreeNode.GetType == 'union' then:
   (a) $childNode_0$ = SyntaxTreeNode.GetChild(0);
   (b) $childNode_1$ = SyntaxTreeNode.GetChild(1);
   (c) $subFSM_0$ = TransformSyntaxTreeToFSM($childNode_0$);
   (d) $subFSM_1$ = TransformSyntaxTreeToFSM($childNode_1$);
   (e) FSM = Merge($subFSM_0$, $subFSM_1$);
   (f) Return FSM);
4. else if SyntaxTreeNode.GetType == 'concatenation' then:
   (a) $childNode_0$ = SyntaxTreeNode.GetChild(0);
   (b) $childNode_1$ = SyntaxTreeNode.GetChild(1);

(c) $subFSM_0$ = TransformSyntaxTreeToFSM($childNode_0$) ;

(d) $subFSM_1$ = TransformSyntaxTreeToFSM($childNode_1$) ;

(e) FSM = Concatenate($subFSM_0$, $subFSM_1$) ;

(f) Return FSM ;

Note the Merge() operation merges two finite state machines starting from initial states and removing duplicate arrows. Where as a Concatenate() operation simply concatenates all terminal states of the first finite state machine with the initial state of the second one.

The results of the more than 10,000 experiments have shown that for a randomly generated acceptor with the obtained sets E and C , the presented method of learning was restoring this acceptor using these sets.

Thus, we can assume that the proposed method is precise on regular acceptors. The last assumption can be considered as evidence in favour of the validity of the proposed method of machine learning.

Conclusion

In the paper a Regular CEP-machine learning method and its correctness verification procedure were considered. It was shown that this method can be used during design and development of the systems based on Event Driven Architecture. The advantage of this method is needlessness of all possible event combinations analysis. Instead the system can adapt it self during runtime. The computer experiment based on the correctness validation procedure shows that the method restores tested Regular CEP-machine.

The obtained results make need for further research in the following directions:

- future experimental study of the proposed method in order to clarify the boundaries of its applicability;
- finding rigorous formulations of the method convergence conditions;
- mathematical justification of the method;
- evaluation of the effectiveness of the method

References

1. Dokuchaev, M., Novikov, B., Zholtkevych G.: Partial Actions and Automata. Alg. Discr. Math. 11(2), 51-63 (2011).

2. Dorozhinsky, V.: Regular Complex Event Processing Machines. Systemy obrobky informacii. 8, 82-86 (2015).

3. Friedman, J.: Data Mining and Statistics: What's the connection? Computing Science and Statistics 29 (1): 39 (1998).

4. Hopcroft, J.: An nlogn algorithm for minimizing states in a finite automaton. In: Proc. Internat. Sympos., Technion, Haifa. Theory of machines and computations, pp. 189-196. Academic Press, New York (1971).

5. Kohavi, R., Provost, F.: "Glossary of terms". Machine Learning 30: 271274 (1998).

6. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002).

7. Mannila, H.: Data mining: machine learning, statistics, and databases. Int'l. Conf. Scientific and Statistical Database Management. IEEE Computer Society (1996).

8. Scientific Computing Tools for Python. http://scipy.org/.

9. Zholtkevych, G., Novikov, B., Dorozhinsky, V.: Pre-Automata and Complex Event Processing. In: V. Ermolayev et al. (eds.) ICTERI 2014. CCIS, vol. 469, pp. 100116. Springer International Publishing, Switzerland (2014).

10. Zholtkevych, G.: Realisation of Synchronous and Asynchronous Black Boxes Using Machines. In: V. Yakovyna et al. (eds.) ICTERI 2015. CCIS, vol. 594, pp 124-139. Springer International Publishing, Switzerland (2016).

11. Zholtkevych, G., Dorozhinsky, V., Khadikov, A: Real-Time event processing and machine learning methods. Weapons systems and military equipment. 2, pp. 79-83 (2016).

Надійшла до редколегії 15.06.2016

Рецензент: д-р техн. наук, проф. Г.А. Кучук, Харківський національний університет Повітрянх Сил ім. Івана Кожедуба, Харків.

РЕГУЛЯРНА ОБРОБКА ПОДІЙ ТА ВЕРИФІКАЦІЯ ВІРНОСТІ МЕТОДІВ МАШИННОГО НАВЧАННЯ

Г.М. Жолткевич, В.В. Дорожинський, А.В. Хадіков

Сьогодні такі концепції як архітектура керована подіями та машинне навчання все частіше використовуються підчас розробки інформаційних систем. У статті авторами запропонований підхід до поєднання цих двох концепцій для розв'язання проблеми логічної складності проектування систем, що базуються на архітектурі керованій подіями. Використовуючи чіткі математичні формулювання, у статті розглянуто метод машинного навчання компонентів таких систем. Зокрема досліджується проблема верифікації правильності методу. Окреслені напрямки подальшого дослідження.

Ключові слова: *машинне навчання, обробка подій, акцептор, регулярна мова, безпрефіксна мова.*

РЕГУЛЯРНАЯ ОБРАБОТКА СОБЫТИЙ И ВЕРИФИКАЦИЯ ПРАВИЛЬНОСТИ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

Г.Н. Жолткевич, В.В. Дорожинский, А.В. Хадиков

В настоящее время такие концепции как архитектура управляемая событиями и машинное обучение все чаще используются в информационных системах. В статье авторы предлагают подход объединяющий эти две концепции для решения проблемы логической сложности проектирования систем основанных на архитектуре управляемой событиями. Используя строгие математические формулировки, рассматривается метод машинного обучения компонентов таких систем. В частности, изучается проблема верификации правильности метода. Предлагаются направления для дальнейшего исследования.

Ключевые слова: *машинное обучение, обработка событий, акцептор, регулярный язык, беспрефиксный язык.*