Temporal feature analysis method describing for automata models of computation systems

M P Sinev$^{1,3}$, M A Mitrokhin$^1$, A I Martyshkin$^2$, I N Doroshenko$^1$, A V Dubravin$^1$, M S Fedyashov$^1$ and A V Gorynina$^1$

1 Penza State University, 40, Krasnaya Street, Penza, 440026, Russian Federation
2 Penza State Technological University, 1a, Baidukova Proezd / 11, Gagarina Street, Penza, 440039, Russian Federation
3 E-mail: mix.sinev@gmail.com

Abstract. The paper reported the temporal feature analysis method describing for automata models of computation systems. The method is based on from source algorithm to modified algorithm expanding makes it possible to take characteristics of work process related to execution time measurement. The paper considers the transition method from source algorithm representation to modification a final state machine with additional states providing registration of temporal features. The paper demonstrates method usage example on computer system authorization algorithm, approves an algorithm complexity for features registration $O(n)$ for $n$-states automata algorithm. The method can be used for a large class algorithm, but is recommended to apply it to an algorithm, separated on procedure thus mean of commands amount should be more than 500 for more time measurement accuracy.

1. Introduction

Computational complexity holds an important place in software development – it is a dependency of time for result computation from input data amount. The characteristic is fundamental in algorithm computational complexity analysis. In practice, frequently requirement is determination specific parameters of algorithm work that influence execution time, not $O$-notation order [1,2]. For example, it may be functions or methods calls count, its execution time, or function-to-function transition amount.

In our time final states machines model has found wide application in the computer science. Its advantages are simplicity and power simultaneously because straightforward automates define regular grammar. An algorithm in automata representation [3] can be modified to observe parameters of the algorithm execution, such as procedure execution time interval or function-to-function transition amount. It allows deep analysis of processes executed inside of algorithm and detect more optimal solution for tasks or debug it [4,5].

Therefore, we can modify an algorithm for automated collecting and following analyzing characteristics if transform it to automata model.

2. Method describing

A finite state machine that described the analyzed algorithm is an oriented graph $G(V \ E)$ where every $V$ (vertex) is incidental for $E$ (edge). It more practice mean, a state is elementary pat of computation process (algorithm step) which consist is:
• Execution part (procedure)
• Set of incident arcs (delegation of control for another part of algorithm)
• Name (procedure identifier)

Notice that set of arcs must correlate with the signals, upon consuming of which finite state machine performed the transition to the next state. We can implement it by associative array; it is a tuple

\[ S = \{K,V,\lambda\} \tag{1} \]

where \( K \) – set of possible keys, \( V \) – set of possible values, \( \lambda \) – mapping function, \( \lambda(k) \rightarrow v, k \in K, v \in V \).

States are objects that the main essence of which consists in delegated method or function executing that is a part of modeled algorithm. These objects contain associated array mapped input signals to the next states. These objects have an associative array that mapped input signals to the next states. In physical systems, a transition might belong some time, and we introduce mapping signal on the delay time (in ticks). Notice that ticks amount – discrete value that can be translated into analog value. The translation is realized by

\[ d(c) = m * c \tag{2} \]

where \( c \) – ticks amount, \( m \) – mean tick length (in target unit).

Use set theory, automata state defined by a tuple of:

\[ P = \{T,D,f,N\} \tag{3} \]
\[ T = \{S,Pn,\lambda sp\} \tag{4} \]
\[ D = \{S,Z,\lambda sz\} \tag{5} \]

\( T \) – tuple transitions, \( S \) – input automata alphabet, \( Pn \) – set of adjacent vertex \( s \in S \), \( \lambda sp \) – mapping function of an input signal to state; \( D \) – delay ticks tuple, \( Z \) – set of integers, \( \lambda sz \) – mapping function signal to delay value; \( f \) – function executed upon state activation; \( N \) – symbolic name (identifier).

For parameters recording, we must extend a tuple described above. We try to extract two parameters, that is why there are tuples:

\[ Pc = \{P,c,\beta\} \tag{6} \]
\[ Pt = \{P,D,\beta\} \tag{7} \]

\( c \) – integer of state visiting, \( D \) – tuple of integers that appended every time on computation in the current state \( Pt \).

Every from tuples contains base tuple \( P \). In this case, \( \beta \) is the function called on active state visiting, and inside this function has the explicit call of target function \( f \) (part of algorithm). \( \beta \) function is required for computing parameters appearing in \( f \) process execution.

So, we can define finite state machine any complexity and record characteristics such as a delay between transitions. Thus, the method of temporal characteristic estimation include the stages:

1. Definition of a set of states \( M \), from which we want to take characteristics.
2. Modify initial automata so, for every \( s \in M \) we build tuple \( Pc \) or \( Pt \), and for every \( s \notin M \), we build \( P \) tuple. The result of the stage is modified automata.
3. Modeling of modified automat work. The result of the stage is a temporal characteristic value set. Notice that for more representative data registration should repeat the described process several times (amount of repetition depends on modeled algorithm task).

3. Example of temporal characteristics definition from algorithm presented in automat view

Consider the final state machine describing one of the most popular algorithms from client-server architecture systems – user authentication/authorization [8,9]. The automat is temporal because inside
executing processes take some time [10,11]. Automat can use two variables: current_time – variable storing actual timestamp, last_time – variable storing last time token update. Access token expires in AT_TIME, refresh token expires in RT_TIME.

To automat states, will bound next functions:

- $S_1$ – get last time of token update
- $S_2$ – get new tokens by user credential
- $S_3$ – update tokens
- $S_4$ – use current tokens
- $S_5$ – non-existent user
- $S_6$ – save tokens pair
- $S_7$ – send a request to a server

Signals used in the automat has the semantic:

- $X_1$ – signal generated if current_time – last_time >= AT_LIFE
- $X_2$ – signal generated if AT_LIFE <= current_time – last_time < RT_LIFE
- $X_3$ – signal generated if current_time – last_time < RT_LIFE
- $X_4$ – signal generated if response code 200 (success)
- $X_5$ – signal generated if response code 400 (error)

SCE:

$$S_1(t + 1) = S_1\overline{x_1} \lor S_1\overline{x_2} \lor S_1\overline{x_3}$$  \hspace{1cm} (8)

$$S_2(t + 1) = S_2x_1 \lor S_3x_5 \lor S_2\overline{x_4} \lor S_2\overline{x_5}$$  \hspace{1cm} (9)

$$S_3(t + 1) = S_3x_2 \lor S_2\overline{x_4} \lor S_3\overline{x_5}$$  \hspace{1cm} (10)

$$S_4(t + 1) = S_4x_3$$  \hspace{1cm} (11)

$$S_5(t + 1) = S_2x_5$$  \hspace{1cm} (12)

$$S_6(t + 1) = S_2x_4 \lor S_3x_5$$  \hspace{1cm} (13)

The authorization algorithm presented in the final state machine view is shown in figure 1.
For this task important knowledge about durations of processes that used internet connection, it’s $S_2, S_3, \text{and} S_7$ According to previously described stages of the method, we should use characteristics determined by $P_t$ tuple for there, for others $P$ tuple. For all states we build tuple $P$, then we fill tuple $T$ and $D$ (symbolic name can be same with the state name in automat). After the stage, the model will have ready; hence, there is it realization possibility by programming language.

We can initiate characteristics collecting when the automat has attained the final state.

The collecting process is a graph objects traversal (tuple elements), from which information is extracted; it depends on tuple type: in case, where tuple type is $P_t$ an extracted information is a collection of integers that characterize a duration in ticks [12] of substituted process work. Else for $P_c$, a target characteristic is the amount of a state visiting. Because modeling of automata's work is a determined graph traversal, we can visit the state several times; on characteristics collecting we keep a separate list – tuple $P_m$ – containing all of the other tuples of $P_c$ and $P_t$. In other cases, we can apply Dijkstra’s algorithm because tree structure is the special case of a graph with edge weights $= 1$, or Prim’s algorithm, where is a possibility for traversal in a tree by getting spanning tree and deleting cycles.

Nevertheless, we can define any substituted object that will be collect another information that will be unlike from proposed. It requires defining a tuple $P_x$ which at least one of the elements of this is $P$ tuple.

The collecting temporal characteristics method from an automat is shown in figure 2.

![Figure 2. Collecting temporal characteristics UML-activity diagram.](image)

In rectangles with rounded corners are held virtual methods executed on tuples $P$. In case, where the state does not require characteristics extracting, an automat executes $f$ function that bound to state (included in $P$ tuple). Else there is occurring of wrapping of a method by a proxy pattern (executing $\beta$ function that delegates execution to $f$). For $P_c$ and $P_t$ own registration methods execute respectively.
4. Complexity estimation
The registration method of temporal characteristics is based on modifying an existing final state machine built for the algorithm. Wherein initial and final models have some structural differences enclosed in the appearance of new states. These states spawn as the result of a registration characteristics requirement (because registration one or the other characteristics is a function computation). The process offers two base characteristics that can be evaluated on a graph traversal – there are amount of a state $S$ visiting and set of ticks amount during an automat was in state $S$.

The visiting amount can be evaluated by injecting a state that will increment a variable storing a count of incoming transitions. Notice, the increment may be prefix or postfix operation for state $S$.

Ticks amount spent on computation bound with a state $S$ requires two states – state measuring global ticks before state $S$ and state evaluating the difference between current ticks counter and stored value.

The method’s structural complexity can be defined by: some subset of transitions (edges) $V \subset T$ of modified automat that at the same time is graph $G(S, T)$ where $S$ – states set, must be constricted. The subset defined by $s_t$ is one from the target states that registration of characteristics will occur; therefore, we build additional set $S_{tm}$ which states are used to characteristic’s computation (in this paper characteristics is so that $\max(|S_{tm}|) = 2$).

Let functions.

$$\lambda(s_t) \rightarrow S_{tm}$$ (14)

define states set $S_{tm}$ for state $s_t$, (function say explicitly about states built for it)

$$\mu(s_t, S_{tm}) \rightarrow P$$ (15)

the set of unordered pairs $(s_t, s_i)$, $s_i \in S_{tm}$, (edges incidental to a target state and an additional states).

Then the result of constriction $V = (\forall s_t \in S, \lambda(s_t) = S_{tm} \neq \emptyset)(\bigcup_s \mu(s_t, S_{tm}))$ is an initial graph. Obviously $|V| < |T|$, therefore, we can get an initial automat for linear time.

5. Method implementation
Presented in a final state machine view algorithms are implemented by programming languages. Characteristics registration by fundamental methods, for example, keeping counter straightforwardly or a system calls execution [13], bring side effects about execution process, it can make a source code [14,15] harder to understand. The problem is solved by defining and realizing an application programming interface (API) [16,17]. API is a container for algorithm’s computation, which provide:

1. Methods consistency
2. Computational context providing
3. Characteristics collecting

API usage process looks like this:

```java
ProcessMachine fsm = ProcessMachine();
Signal x1 = Signal('x1');
Signal x2 = Signal('x2');
Signal x3 = Signal('x3');
Signal x4 = Signal('x4');
Signal x5 = Signal('x5');

fsm.addUnit(Auth.getLastTokenModifyTime, MetricType.none, 's1');
fsm.addUnit(Auth.getAccessToken, MetricType.time, 's2');
fsm.addUnit(Auth.updateToken, MetricType.time, 's3');
fsm.addUnit(Auth.useCurrentToken, MetricType.none, 's4');
fsm.addUnit(Auth.userNotExist, MetricType.none, 's5');
```
fsm.addUnit(Auth.saveToken, MetricType.none, 's6');
fsm.addUnit(Auth.sendRequest, MetricType.time, 's7');

fsm.setMapping('s1', x1, 's2');
fsm.setMapping('s1', x2, 's3');
fsm.setMapping('s1', x3, 's4');
fsm.setMapping('s3', x5, 's2');
fsm.setMapping('s3', x4, 's6');
fsm.setMapping('s2', x4, 's6');
fsm.setMapping('s2', x5, 's5');
fsm.setMapping('s6', Signal.unconditional, 's7');
fsm.setMapping('s4', Signal.unconditional, 's7');

fsm.setStartUnit('s1');
fsm.setEnvironment(<String, dynamic>{'time':DateTime(2020,9,4,9,29)});
fsm.start();

fsm.send(fsm.environment['out']);
fsm.send(fsm.environment['out']);
print(fsm.absorbMetric());

ProcessMachine – main class, it is an automat. The class’s object provides all functionality described above.

It includes the next methods:

- addUnit – addition unit to an automat; arguments are procedure of an algorithm, characteristic type, state name;
- setMapping – creating named edge between states;
- setStartUnit – specifying the initial state;
- setEnvironment – creating an initial context of execution algorithms;
- start – initial procedure execution.

After execution of the method, it must specify the next node of computation in the “out” parameter. For example:

static void getLastTokenModifyTime(Map<String, dynamic> env)
{
  DateTime last = env['time'];
  if(last == null)
  {
    env['out'] = Signal('x1');
  }
  int delta = DateTime.now().difference(last).inSeconds;
  if(delta >= 120)
  {
    env['out'] = Signal('x1');
  } else if (delta <= 120 && delta >= 30)
  {
    env['out'] = Signal('x2');
  } else
  {
    env['out'] = Signal('x3');
  }
}
env['out'] = Signal('x3');
} return;

We can model an automata behavior and a work process characteristics registering.

6. Conclusion
Our result of developing and researching temporal characteristics method analysis confirms that the method can be applied to algorithms presented in automata form. In our opinion, the method can be applied for many algorithms except these where a block size (f procedure) is too small. In this case, the time characteristics may be imprecise. In addition, we should notice, time characteristics may be measured more accurately by using processor-time measuring procedures for process – time interval when is executed on CPU. An example of that function is system call times from sys/time.h for Unix-like operating systems.

The next explorative question is API implementation, the analyzer of algorithms presented in final state machines.

Acknowledgments
The research was conducted at the expense of a grant from the Russian Science Foundation № 21-71-00110, https://rscf.ru/project/21-71-00110/.

References
[1] Pashchenko D V, Trokoz D A, Martyshkin A I, Sinev M P and Svitunov B L 2020 Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture Bulletin of Electrical Engineering and Informatics 9(3) 1238-50
[2] Heiner M, Schwarick M and Wegener J T 2015 Charlie – an extensible Petri net analysis tool In International Conference on Applications and Theory of Petri Nets and Concurrency (Cham: Springer) 200-11
[3] Sakarovitch J 2009 Elements of Automata Theory (Cambridge: Cambridge University Press)
[4] Konnov N N et al. 2020 Instrumental System of Temporal Analysis of Models of Concurrent Computing Systems Constructed Using Theory of Temporary Finite State Automata Moscow Workshop on Electronic and Networking Technologies (MWENT) 1-8
[5] Martyshkin A I et al. 2020 Using queuing theory to describe adaptive mathematical models of computing systems with resource virtualization and its verification using a virtual server with a configuration similar to the configuration of a given model Bulletin of Electrical Engineering and Informatics 9(3) 1106-20
[6] Pashchenco D V et al. 2020 Formalization of Organization of Signaling Protocol While Making Online Calls Using Automata Theory Moscow Workshop on Electronic and Networking Technologies (MWENT) 1-7
[7] Trokoz D A et al. 2021 Formalized description of cyber-physical systems models using temporary non-deterministic automata J. Phys.: Conf. Ser. 1889 022068
[8] Convery S 2007 Network authentication, authorization, and accounting The Internet Protocol Journal 10(2)
[9] Shahzad M and Singh M P 2017 Continuous authentication and authorization for the internet of things IEEE Internet Computing 21(2) 86-90
[10] Gamma E, Helm R, Johnson R and Vlissides J 1995 Elements of Reusable Object-Oriented Software 99 (Massachusetts: Addison-Wesley)
[11] Knuth D E 1968 The Art of Computer Programming 1 Fundamental Algorithms (Addison-Wesley) 187
[12] Wald A 2004 Sequential Analysis (New York: Mineola)
[13] Olifer N and Olifer V 2005 *Computer Networks: Principles, Technologies and Protocols for Network Design* (Wiley Publishing)

[14] Meyer B 1997 *Object-Oriented Software Construction 2* (Englewood Cliffs: Prentice hall)

[15] Martin R C 2000 Design principles and design patterns *Object Mentor* 1(34) 597

[16] Lafore R 1997 *Object-Oriented Programming in C++* (Pearson Education) 1038

[17] Gast H 2015 *How to Use Objects: Code and Concepts* (New York: Addison-Wesley Professional)