International Journal of Software Engineering and Computer Systems (IJSECS)
ISSN: 2289-8522, Volume 4 Issue 1, pp. 48-60, February 2018

https://doi.org/10.15282/ijsecs.4.1.2018.4.0037

AN APPROACH TO INCREASE THE EFFECTIVENESS OF TLC VERIFICATION WITH RESPECT TO THE CONCURRENT STRUCTURE OF TLA+ SPECIFICATION

Vadym Viktorovych Shkarupylo[1]*, Igor Tomičić[2], Kostiantyn Mykolaiovych Kasian[1] and Jamil Abedalrahim Jamil Alsayaydeh[3]

[1]Computer Systems and Networks Dept., Zaporizhzhya National Technical University,
69063, Zhukovsky st., 64, Zaporizhzhya, Ukraine,
[2]Faculty of Organization and Informatics, University of Zagreb,
42000, Pavlinska, 2, Varazdin, Croatia,
[3]Faculty of Engineering Technology, Technical University of Malaysia Malacca
76100, Durian Tunggal, Melaka, Malaysia

E-mail: vadshkar@yandex.ua*, igor.tomicic@foi.hr, konst_k@yahoo.com,
jamil@utem.edu.my

ABSTRACT

Modern approaches to distributed software systems engineering are tightly bounded with formal methods usage. The effective way of certain method application can leverage significant outcome, in terms of corresponding time costs reduction for instance. To this end the TLC model checker has been considered – with respect to TLA+ specifications with concurrent structure. The concurrency itself has been implemented as interleaving. Two different approaches to TLC model checking have been used. The first approach is based on model checking via breadth-first state space search (BFS), the second one – via depth-first search (DFS). The main result of a paper is the new approach to increasing the effectiveness of TLC verification with respect to the concurrent structure of TLA+ specification. To analytically represent synthesized TLA+ specifications with concurrent structure, the Kripke structure has been taken. To assess the measures of state space explosion problem, taking place during the experimentation, the appropriate estimations have been proposed. These estimations have been proved during the case study. The composite web service usage scenario has been considered as a case study. The results, obtained during the experimentation, can be used to increase the effectiveness of automated TLC verification with respect to the concurrent structure of TLA+ specification.

Keywords: TLC, TLA+, Model Checking, BFS, DFS.

INTRODUCTION

These days formal methods (model checkers in particular) are intensively used in different spheres of engineering. One of those are the distributed software systems. Designing certain software system, it's vital to periodically check, whether current solution actually meets the specified requirements. Here the another question arises – whether we can conduct such checking in an automated manner. This peculiarity is especially important in context of iterative development. To this end, to check in an automated manner whether our current design solution meets the specified

requirements, the formal verification is going to be considered from the model checking viewpoint.

Our attention has been put on the TLC (TLA Checker) model checker (Lamport, 2002). This method has already proved its applicability, having been successfully used to find the design flaws in Amazon Web Services for instance (Newcombe, Rath, Zhang, Munteanu, Brooker & Deardeuff, 2015). Moreover, this method is aimed at mathematically rigorous and easy reconfigurable TLA+ specifications (Temporal Logic of Actions) automated verification. Named formalism provides the abilities to create specifications with the desired level of abstraction, grounding on an "action" concept – the specification of relation between some current and subsequent states of system's formal model. This level can be subsequently increased/decreased during the designing, which leverages the flexibility with respect to the altering nature of requirements. In our work the specification is being considered as a composition of parallel and sequential constructs.

To proceed further, let's consider the Software Development Life Cycle (SDLC) with respect to iterative model (Larman, 2004). In accordance with this model, each iteration is a sequence of the following stages (phases): requirements analysis, designing, implementation and testing. We put our focus on the second phase – the designing. During this process a bunch of various models and specifications with different levels of abstraction are being created. It's vital here to operate with correct and unambiguous specifications – to lessen the expenses, taking place within the forthcoming phases – implementation and testing in particular.

The altering nature of different modern business scenarios prompts the necessity to deploy scalable and easy reconfigurable solutions. To this end the principles of Service Oriented Architecture (SOA), e.g., reusability, interoperability, composability, can be utilized (Martini & Paganelli, 2016). The key concept here is the Composite Web Service (CWS). In our work the CWS is considered as a system. The components of such system are some other services (atomic or composite). We contemplate the scenario, when system functioning is provided by way of orchestration – centralized coordination (invocation) of system's components.

The exhaustive overview of SOA- and cloud-related SDLCs has already been conducted (Tran & Feuerlicht, 2016). It can be seen that the number of distinguished SDLC-phases and the phases set itself vary significantly. For instance, SOA-related SDLC can be considered as follows: requirements analysis, services discovery, negotiation, composition and consumption. With respect to this model our focus is put on a composition phase – to check the consistency of system's components.

To sum up, the main insight of our work is that modern SDLC of distributed software system is almost unimaginable without formal verification being conducted. The corresponding verification-related time costs influence the overall SDLC-related expenses, more or less. This influence is even more essential, when iterative SDLC takes place. To diminish this effect – to lessen the verification-related time costs – the model checking method has to be used properly, in particular. To this end the TLC method is being considered in our work from the efficiency viewpoint. We are willing to find out whether the effectiveness of TLC method usage actually depends on a structure of certain TLA+ specification to be verified in an automated manner by way of model checking. And if it is, whether we can influence this effectiveness (in terms of corresponding time costs reduction) by using the right approach to TLC-driven model checking – by way of breadth-first state space search (BFS) or by way of depth-first search (DFS). The preliminary work has already been done: the investigation of TLC

with respect to TLA+ specification with solely sequential structure has been conducted (Shkarupylo, Tomičić & Kasian, 2016). Both approaches (the BFS- and the DFS-driven one) have been used. It has been found out that DFS-driven approach is about two times more effective, comparing to BFS-driven alternative, but the following drawbacks also take place: the necessity to manually specify the depth of state space search, which directly harms the model checking process in terms of automation. Taking into consideration all the aforesaid, we are willing to find out whether the effectiveness of TLC method usage actually depends on a structure of TLA+ specification.

PROBLEM STATEMENT

Given a set of TLA+ specifications with concurrent structures.

Let's consider the effectiveness of TLC method usage from the verification-related time costs viewpoint. There are two approaches to TLC method usage distinguished: model checking by way of BFS and by way of DFS.

Let's consider CWS as a system (Shkarupylo, 2016). Let's assume that CWS is represented with WS-BPEL-description (Web Services Business Process Execution Language), which is the implementation of centralized orchestration model (Mecheraoui, Belala & Saïdouni, 2016). According to this model, the components of CWS are depicted with <invoke> tags. Each <invoke> tag is a certain web service to be invoked.

To analytically represent TLA+ specification, the Kripke structure on a set of atomic prepositions AP has been used (Clarke, Grumberg & Peled, 2001):

$$\langle S, \{s_0\}, R, L \rangle, \quad (1)$$

where S – finite set of states, $s_0 \in S$ – initial state, $R \subseteq S^2$ – set of transitions, $L : S \to 2^{AP}$ – states labeling function.

The S set is as follows:

$$S = V \times D, \quad (2)$$

where $V = \{v_i | i = 1,2,...,n\}$ – set of state variables: $n \in N$ is a number of CWS's components – the amount of <invoke> tags in WS-BPEL-description; $D = \{0,1\}$ – set of values.

To unify the experimentation process, the analytical model of automatically synthesized TLA+ specification has to be proposed. This model should provide the scalability to get specifications for CWSs with different numbers of components.

To assess the effectiveness of TLC method usage with respect to the concurrent structure of TLA+ specification, the estimations of model checking tasks to be solved have to be proposed. These estimations have to be proved experimentally.

The experimentation has to be conducted by using two approaches to TLC model checking – the BFS- and the DFS-driven one. Corresponding results have to be compared in terms of effectiveness.

To generalize the conclusion on TLC method usage effectiveness, the obtained results (with respect to the concurrent structures of TLA+ specifications) have to be compared to the ones, obtained previously (with respect to solely sequential structures of specifications).

APPROACH DESCRIPTION

Let's suppose that there are two groups of TLA+ specifications – the specifications with purely sequential structures (first group) and the ones with concurrent structures (second group).

To synthesize the first-group specifications the <sequence> and <invoke> tags have been used as a basis. The <sequence> tag – as the representative of WS-BPEL Structured Activities group. The <invoke> tag – as the element of WS-BPEL Basic Activities group. To synthesize the second-group specifications, the <flow> tags from Structured Activities group have also been considered.

Let's consider the V set (Eq.(2)): $\forall v_i \in V$ there are only two possible Boolean values allowed – false (0) or true (1). Now let's talk in terms of atomic prepositions – to interpret the elements of AP set (Eq.(1)). For instance, the $(v_i = 0) \in AP$ atomic preposition means that some i-th component of CWS hasn't yet been invoked. On the contrary, if $(v_i = 1) \in AP$ takes place, then it has already been invoked.

Let's consider the first-group specification. The precondition for $(i+1)$-th component invocation can be represented with $(v_i = 1) \in AP$ atomic preposition. This means that i-th component has already been invoked. In this case $|S| = n+1$: $L(s_0) = \{(v_1 = 0), (v_2 = 0), ..., (v_n = 0)\}$, $L(s_1) = L(R(s_0)) = \{(v_1 = 1), (v_2 = 0), ..., (v_n = 0)\}$, ... $L(s_n) = L(R(s_{n-1})) = \{(v_1 = 1), (v_2 = 1), ..., (v_n = 1)\}$, where $s_n \in S$ is the final state to be reached during the automated TLC verification. The more detailed view on TLA+ specifications with solely sequential structure has already been provided (Shkarupylo et al., 2016).

Now let's take a look at TLA+ specifications with concurrent structures. Let's represent the concurrency as interleaving. Let's suppose that the structure of any TLA+ specification is binary tree-like. This decision has been made to obtain scalable and demonstrative solutions. The vertices of such structures are pre- and post-conditions for components invocations. The invocation itself is considered in given scope as an activity (action). In TLA context the activities are the atomic building blocks of specification. To specify certain activity the precondition of its implementation should be given and specified first.

Let's represent the activity as an implication, coupled with X (neXt) temporal operator: $\neg(v_i = 0) \vee X(v_i = 1)$ (Shkarupylo, Polska & Kudermetov, 2015). This means that truth of certain $(v_i = 0) \in AP$ atomic preposition implies the forthcoming truth of the appropriate $(v_i = 1) \in AP$ atomic preposition. That should be interpreted as follows: if certain component is intended to be invoked (in accordance with specification), it has to be invoked. This formalization is built on the following basis: $L(s) \Delta L(s') = \{(v_i = 0), (v_i = 1)\}$, where $s \in S$ is some current state: $L(s)$ is the representation of the precondition for i-th component invocation, $s' = R(s) \in S$ is a subsequent state – the basis for the post-condition, represented with $L(s')$ label as a result of i-th component invocation.

Let's synthesize the TLA+ specifications with concurrent binary tree-like structures. The binary tree template will be slightly modified to more naturally represent the real-world scenarios: all the terminal vertices will be joined to a single one – the

final point, representing the final activity of gatherer-component invocation, modeling the intermediate computational results obtaining.

At least $|V| = 2^2$ state variables are required to synthesize such specification: $V = \{v_1, v_2, v_3, v_4\}$, where $v_1 \in V$ is the basis for the 1-st components invocation activity; this component should initiate the computational process by distributing the tasks to be processed concurrently by a pair of components, represented with $v_2, v_3 \in V$ state variables; $v_4 \in V$ – the representation of a component, intended to be a gatherer (Figure 1).

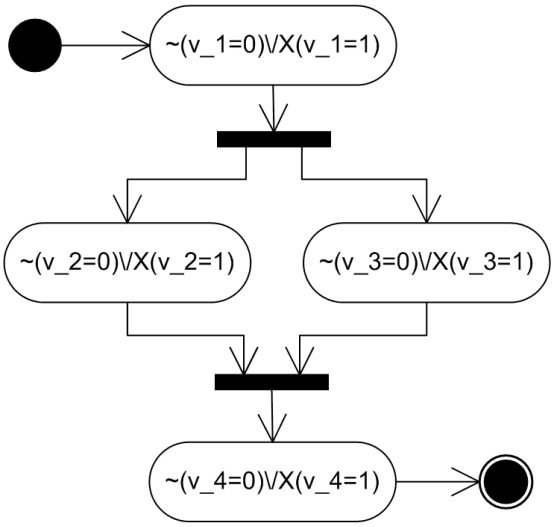

Figure 1. The activity diagram of concurrency

In Figure 1 there are three layers of activities that should be distinguished. The 1-st one, represented with single activity $\neg(v_1 = 0) \vee X(v_1 = 1)$; the 2-nd one – with a pair of activities to be implemented concurrently: $\neg(v_2 = 0) \vee X(v_2 = 1)$ and $\neg(v_3 = 0) \vee X(v_3 = 1)$; the 3-rd one – with single activity $\neg(v_4 = 0) \vee X(v_4 = 1)$. For this particular example let's assume the scenario of distributed π value calculation (Shkarupylo, 2016):

$$\pi = 48 \cdot arctg(1/18) + 32 \cdot arctg(1/57) - 20 \cdot arctg(1/239). \qquad (3)$$

In accordance with Eq.(3), $2/3$ of all computations can be conducted by a component, represented with $v_2 \in V$ state variable, and the rest of computations $(1/3)$ can be conducted by a component, represented with $v_3 \in V$ state variable (or vice versa). The $v_1 \in V$ state variable represents some initiator of computational process. It can be a distinct web service (atomic or composite) that actually initiates the computational process – distributes the whole task for instance. It can also be the representation of WS-BPEL-engine itself – the entity, which initiates and coordinates the computational process in a centralized manner, to provide the consistent system functioning. Finally, the $v_4 \in V$ variable is a representation of some gatherer-component – the atomic or composite web service, which gathers the intermediate

computational results, obtained from the components, represented with $v_2, v_3 \in V$ state variables.

Let's consider the 1-st and the 3-rd layers as the boundary ones (Figure 1). Paired with intermediate (2-nd) layer, each of them form fork- and join-constructs, respectively. The 1-st layer represents the initiation of some computational process and the 3-rd one – the finalization of it. So, in our case to form the minimum basis of concurrency inspired TLA+ specification at least 4 state variables are required. That provides us with minimum set of pair of boundary layers and at least one intermediate layer, representing the concurrency. Such concurrency is going to be modeled as interleaving. To form the additional intermediate layer another 4 state variables should be added, then 8, and so on. The main idea here is that the activities within certain intermediate layer should be implemented concurrently, and this particular layer-oriented concurrency should be modeled as interleaving. Such approach stipulates the state explosion problem within the intermediate layers. To sum up, the main idea here is that each TLA+ specification with concurrent structure should possess strictly a pair of boundary layers and at least one intermediate layer. Then, during the experimentation, the number of intermediate layers should be increased step-by-step, until the limitation of random access memory is faced.

In case of two or more intermediate layers, the boundaries between such adjacent layers then should be considered as barrier-functions (MPI_Barrier for instance, when some computational process is waiting for all another processes to continue/resume the computation). That means that no activity from adjacent subsequent intermediate layer can be executed until all the activities from some current intermediate layer are committed. In general this statement is also fair when some representative of a pair of adjacent layers is the boundary one.

The representatives of intermediate layer(s) jointly cause the exponential state space growth. Herein the number of state variables, involved in concurrent activities, is $(n-2)$ – not taking into consideration the elements from boundary layers.

Let's represent the layers with $H = H^* \cup H' = \{h_0, h_1, ..., h_k, ..., h_{\log_2 n - 1}, h_{\log_2 n}\}$ set, where $H^* = \{h_0, h_{\log_2 n}\} \subset H$ – boundary layers subset, where $h_0 \in H^*$ – boundary fork-layer, $h_{\log_2 n} \in H^*$ – boundary join-layer; $H' = \{h_1, h_2, ..., h_{\log_2 n - 1}\} \subset H$ – intermediate layers subset.

To synthesize the specifications, the n values are going to be taken from the sequence $2^2, 2^3, ..., 2^7$. The 2^7 value has been chosen as an upper limit in accordance with the results, obtained earlier (Shkarupylo et. al., 2016). In accordance with these results, the limitation of random access memory volume has been faced while using the DFS-driven approach to automated TLC verification with respect to the sequential structures of TLA+ specifications.

Let's estimate the total number of components to be invoked concurrently:

$$f_1(n) = \sum_{k=1}^{\log_2 n - 1} 2^k, \qquad (4)$$

where $f_1(n)$ – the function to obtain the total number of state variables from the intermediate layer(s);

2^k – the number of state variables from $h_k \in H'$ intermediate layer.

In our case $f_1(n)=\sum_{k=1}^{(\log_2 4)-1}2^k=2$. This means that both state variables ($v_2\in V$ and $v_3\in V$) are the representatives from a single intermediate layer $h_1\in H'$ (Figure 1): $H=H^*\cup H'=\{h_0,h_2\}\cup\{h_1\}$.

It should be emphasized that $\forall n\in\{2^2,2^3,...,2^7\}$ there is always a pair of state variables ($v_1\in V$ and $v_n\in V$) from the boundary layers: $v_1\in V$ and $v_n\in V$ – from fork- and join-boundary layers, respectively. This means that the total number of state variables can be obtained with the following expression: $f_1(n)+|H^*|=|V|$, where $|H^*|=2=const$ is the total number of state variables from both boundary layers.

As a correctness check for $n=2^2$ $f_1(n)=2$ (Eq.(4)): $n=|V|=f_1(n)+2=2+2$.

For $n=2^3$ $f_1(n)=2+4=6$: $n=|V|=f_1(n)+2=(2+4)+2=6+2$. And so forth.

To estimate (count) the number of states to be reached during the successful TLC-verification, the following function has been proposed:

$$f_2(n)=|S|=\sum_{k=1}^{\log_2 n-1}\left(2^{2^k}-1\right)+\alpha, \qquad (5)$$

where $\left(2^{2^k}-1\right)$ is the number of states to be reached due to the concurrent invocation of the components, represented with state variables from $h_k\in H'$ intermediate layer; (-1) – to take into consideration the adjacency of some current and subsequent intermediate layers, when the resulting state for $h_k\in H'$ is the initial state for $h_{k+1}\in H'$; $\alpha=3=const$ – the number of states, associated with boundary layers – induced by invocation of state variables from the boundary layers: $s_0\in S$, $s^*\in S$ and $s^{**}=R(s^*)\in S$, where $s^*\in S$ and $s^{**}\in S$ are the pre-final and final states, respectively: $L(s^*)\in S$ and $L(s^{**})\in S$ are the pre- and post-conditions for the final $\neg(v_n=0)\vee X(v_n=1)$ activity occurrence (Figure 2).

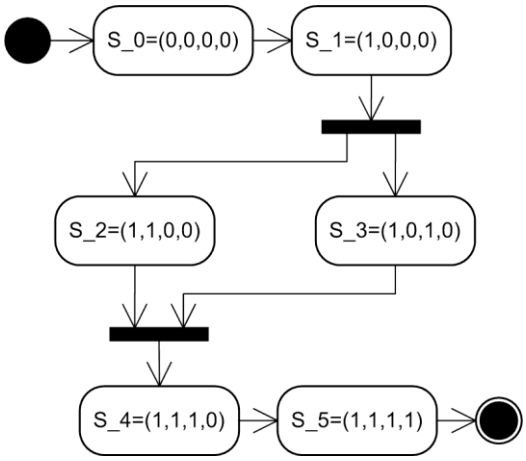

Figure 2. The state diagram for $n=4$

For instance, for $n = 2^2, 2^3, 2^4$ $f_2(n) = 6, 21, 276$, respectively (Eq.(5)). It will allow to check the results of automated TLC-verification of synthesized TLA+ specifications with binary tree-like structure (Figure 1).

To estimate (count) the number of transitions, needed to visit all the states of S set (Eq.(1)), the following function has been proposed:

$$f_3(n) = |R| = \sum_{k=1}^{\log_2 n - 1} \left(2^{2^k + k - 1}\right) + \beta, \qquad (6)$$

where $\beta = 2 = const$ is the number of transitions (activities in particular), taking their ground solely on the boundary layers $h_0, h_{\log_2 n} \in H^*$. For instance, considering the boundary fork-layer $h_0 \in H^*$, the $(s_0, s_1) \in R$ transition takes place: $L(s_0) \Delta L(s_1) = \{(v_1 = 0), (v_1 = 1)\}$ is the basis for the initial activity $\neg(v_1 = 0) \vee X(v_1 = 1)$. For this activity the precondition $(v_1 = 0) \wedge ... \wedge (v_4 = 0)$ is formed from the elements of $L(s_0)$ set as a conjunction; the post-condition $(v_1 = 1) \wedge (v_2 = 0) \wedge .(v_3 = 0) \wedge (v_4 = 0)$ – from the elements of $L(s_1)$ (Figure 2). Considering the boundary join-layer $h_{\log_2 n} \in H^*$, the $(s_4, s_5) \in R$ transition takes place: $L(s_4) \Delta L(s_5) = \{(v_4 = 0), (v_4 = 1)\}$ is the basis for the final activity $\neg(v_4 = 0) \vee X(v_4 = 1)$, where $(v_1 = 1) \wedge (v_2 = 1) \wedge .(v_3 = 1) \wedge (v_4 = 0)$ is a precondition and $(v_1 = 1) \wedge (v_2 = 1) \wedge .(v_3 = 1) \wedge (v_4 = 1)$ – a post-condition – the representation of the final state $s_5 \in S$, to be reached during the automated TLC verification.

For instance, for $n = 2^2, 2^3, 2^4$ $f_3(n) = 6, 38, 1062$, respectively (Eq.(6)).

The $f_3(n)$ function can be used to estimate the complexity of model checking task to be solved and to actually check the successfulness of the solution obtained.

THE CASE STUDY

To conduct the case study, the 2.05 version of TLC has been used.

The platform used: Central Processing Unit – AMD K10, 3 GHz; Random Access Memory (RAM) – 2 GB, DDR3; Operating System – MS Windows 7; Java Runtime Environment version – 1.7.

The numbers of states, found during the verification, are given in Table 1.

Table 1. The BFS-to-DFS model checking tasks measures comparison

States	n values			
	2^2		2^3	
	BFS	DFS	BFS	DFS
Gen	13	25	1009	3601
Found	6		21	
Depth	5		9	

In Table 1 Gen is the total number of states, generated by TLC to verify the TLA+ specification, synthesized in accordance with proposed analytical model. Found – the numbers of states, found and checked during the verification. These values have been obtained experimentally – from the TLC log-files. They can also be obtained analytically – with Eq.(5).

Depth – the depth of state space search – the number of states (including $s_0 \in S$) to be visited, starting from the initial $s_0 \in S$ state and finishing in the final $s^{**} \in S$ state. Taking into consideration the interleaving nature of our concurrency, the Depth is also the length of each path from the initial to the final state. Due to the initial state presence and interleaving, these paths are of the same length for a given n (Figure 2):

$$f_4(n) = n + 1, \tag{7}$$

where $f_4(n)$ is a function to get the length of each path from the initial state to the final one for a given $n = |V|$.

To calculate the total number of such paths, the following expression can be used:

$$f_5(n) = \prod_{k=1}^{\log_2 n - 1} (2^k)!, \tag{8}$$

where $f_5(n)$ is a function to get the total number of paths to be walked through during the automated TLC-verification of certain TLA+ specification, synthesized with accordance to the proposed analytical model. Thus, using Eq.(7) and Eq.(8), the complexity assessment of model checking task to be solved can be represented as multiplication of $f_4(n)$ and $f_5(n)$ functions:

$$f_6(n) = f_4(n) \cdot f_5(n) = (n+1) \cdot \prod_{k=1}^{\log_2 n - 1} (2^k)!, \tag{9}$$

The calculated estimations for $n = 2^2, 2^3, 2^4$, obtained with Eq.(8), are given in Table 2.

Table 2. The estimations of model checking tasks complexities

n	$f_4(n)$	$f_5(n)$	$f_6(n) = f_4(n) \cdot f_5(n)$
2^2	5	2	10
2^3	9	48	432
2^4	17	1935360	32901120

It can be concluded, according to the Table 2, that $f_6(n)$ value for $n = 2^4$ induces some doubts about the random access memory sufficiency to successfully resolve the model checking task (Eq.(9)).

The $f_4(n)$ and $f_5(n)$ values for $n = 2^2, 2^3$, given in Table 2, have been proved with TLC logging data.

To discover the ways for TLC-verification effectiveness increasing, the time costs for BFS- and DFS-driven verifications have been measured (Table 3). In Table 3 the averages of 10^2 measures are given. All the specifications, synthesized for $n = 2^2, 2^3$, have been successfully checked. The numbers of states, visited during the model checking, and the appropriate depths of state space search have been equal to the estimations, given in Table 1.

Table 3. The time costs for BFS- and DFS-driven model checking

n	TLA+ specifications					
	sequential			concurrent		
	$\bar{t}_{BFS}$	$\bar{t}_{DFS}$	$\bar{t}_{BFS} / \bar{t}_{DFS}$	$\bar{t}_{BFS}$	$\bar{t}_{DFS}$	$\bar{t}_{BFS} / \bar{t}_{DFS}$
2^2	0,934	0,420	2,230	0,926	0,425	2,179
2^3	0,952	0,450	2,130	1,094	0,623	1,756
2^4	1,029	0,540	1,910	-	-	-

In Table 3 there are two sections – the "sequential" section and the "concurrent" one.

In the first section the results of previous experimentations are given (Shkarupylo et. al., 2016). These results have been obtained for TLA+ specifications with purely sequential structure. The ratio between $\bar{t}_{BFS}$ and $\bar{t}_{DFS}$ values has been chosen as effectiveness criterion. It can be seen from the Table 3 that BFS-driven approach is about two times more time consuming, comparing to the DFS-driven one (considering the sequential specifications). Thus, in this scenario the DFS-driven approach is about two times more effective, comparing to the BFS-driven alternative. The first one, though, is paired with the significant limitations – the depth of state space search has to be assigned manually. This peculiarity significantly restricts the DFS-driven approach practical usage in terms of automation.

Considering the "concurrent" section of Table 3, we can observe a bit different picture. These results have been obtained during the current experimentation. It can be seen from this section that DFS-related time costs are growing significantly quicker, but the whole picture is pretty similar (considering the cases for $n = 2^2, 2^3$). Thus, the general recommendations in terms of effectiveness are going to be similar to the ones with respect to the aforesaid "sequential" section: the DFS-driven approach is significantly more effective, comparing to BFS-driven one. Nevertheless, the advantage of the first over the later one is not as significant, when talking about the verification of TLA+ specifications with purely sequential structure ("sequential" section). Moreover, the aforesaid drawbacks are still here: the necessity to manually assign the depth of state space search and also the significantly bigger number of accompanying states, generated during the verification (Table 1). It can be assumed that the later peculiarity imposes some extra-requirements to the RAM volume. Nevertheless, both considered approaches (the BFS- and the DFS-driven one) appeared to be inapplicable to conduct the verification of TLA+ specification with a concurrent structure for $n = 2^4$, taking into consideration the estimations, given in Table 2, and also the limitation of RAM volume.

It should be noted, however, that the sizes of TLA+ specifications, synthesized with respect to the proposed analytical model, were the following: for $n = 2^2$ – about 1 KB; for $n = 2^3$ – about 7 KB; for $n = 2^4$ – about 210 MB.

While changing the volume of JVM-accessible (Java Virtual Machine) memory buffer from 256 to 1280 MB, the corresponding BFS- and DFS-related time costs were almost identical (Figure 3). The model checking task hasn't been solved though.

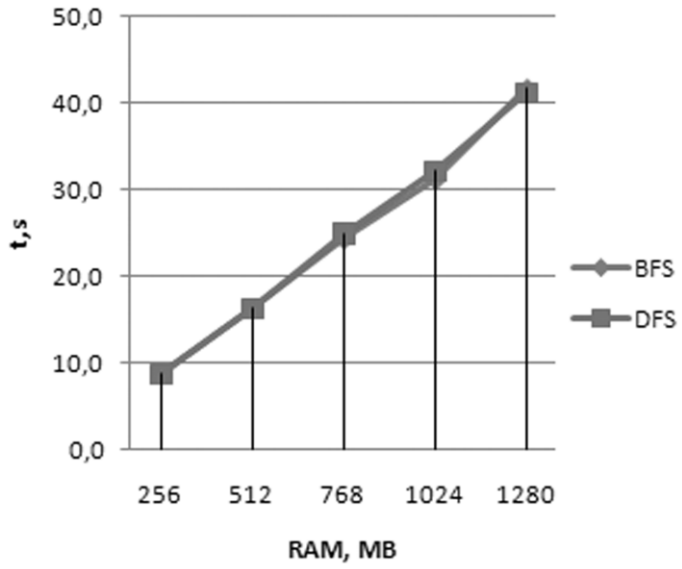

Figure 3. The verification time costs for $n = 2^4$

In Figure 3 the almost linear dependency of verification time costs from the available JVM-buffer size can be seen. The whole pictures for both considered approaches are nearly the same. Both of them appeared to be inapplicable to resolve the model checking task for $n = 2^4$, taking into consideration the limitation of memory space.

CONCLUSION

In this paper the novel approach to increasing the effectiveness of TLC verification with respect to the concurrent structure of TLA+ specification has been proposed. The following results have been obtained:

1. The layered binary tree-like concurrent structure of TLA+ specification has been proposed. The concurrency has been represented as interleaving. This allowed to synthesize unified and structured specifications for the purpose of experimentation.

2. The estimations of TLC model checking tasks have been proposed and proved by way of experimentation. It has been determined that DFS-driven approach to TLC verification of TLA+ specifications with concurrent structures generates significantly larger number of states, comparing to BFS-driven one.

3. The case study has been conducted. It has been determined that the overall tendency of time costs growing for TLA+ specifications with concurrent structure is similar to the one, previously found out by us for specifications with purely sequential structure: the DFS-driven approach is about two times more effective (in terms of corresponding time costs), comparing to the BFS-driven one.

4. The limitation of random access memory has been faced. It has been determined that both approaches – the DFS- and the BFS-driven one – are inapplicable to solve the verification task with respect to TLA+ specifications with $n = 2^4$ state variables, synthesized in accordance with proposed analytical model, taking into consideration the 1280 MB limitation of JVM buffer size.

The further research is aimed at proposed analytical model sophistication, by increasing the set of state variables values to make synthesized formal specifications more relevant.

ACKNOWLEDGEMENT

With special thanks to Ravil Kudermetov (Head of Computer Systems and Networks Department) for the inspiration and all-around support.

REFERENCES

Clarke, E. M., Grumberg, O. & Peled, D. (2001). *Model Checking*. Massachusetts : MIT Press.

Lamport, L. (2002). *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Boston : Addison-Wesley.

Larman, C. (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. New Jersey : Prentice Hall.

Martini, B. & Paganelli, F. (2016). A Service-Oriented Approach for Dynamic Chaining of Virtual Network Functions over Multi-Provider Software-Defined Networks. *Future Internet*, 8(2), 21 p. doi: 10.3390/fi8020024

Mecheraoui, K., Belala, N. & Saïdouni, D. E. (2016) Towards a Comprehensive Formal Model for Business Processes. 22nd International Conference on Information and Software Technologies, 174-186. doi: 10.1007/978-3-319-46254-7_14

Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M. & Deardeuff, M. (2015). How Amazon Web Services Uses Formal Methods. *Communications of the ACM*, 58(4), 66-73. doi: 10.1145/2699417

Shkarupylo, V. V., Tomičić, I. & Kasian, K. M. (2016). The investigation of TLC model checker properties. *Journal of Information and Organizational Sciences*, 40(1), 145-152.

Shkarupylo, V. (2016). A Technique of DEVS-Driven Validation. XIIIth International Conference on Modern Problems of Radio Engineering, Telecommunications, and Computer Science, 495-497. doi: 10.1109/TCSET.2016.7452097

Shkarupylo V. (2016). A Simulation-driven Approach for Composite Web Services Validation. 27th International Central European Conference on Information and Intelligent Systems, 227-231.

Shkarupylo, V. V., Polska O. V. & Kudermetov R. K. (2015). DEVS-oriented technique for composite web services validity checking. *Radio Electronics, Computer Science, Control*, 4, 79-86. doi: 10.15588/1607-3274-2015-4-12

Tran, H. T. & Feuerlicht, G. (2016). Service Development Life Cycle for Hybrid Cloud Environments. *Journal of Software*, 11(7), 704-711. doi: 10.17706/jsw.11.7.704-711

APPENDIX A: THE EXAMPLE OF TLA+ SPECIFICATION

```
VARIABLES v1, v2, v3, v4
Invariant ==    /\ v1 \in BOOLEAN
                   /\ v2 \in BOOLEAN
                   /\ v3 \in BOOLEAN
                   /\ v4 \in BOOLEAN
Init == v1=FALSE /\ v2=FALSE /\ v3=FALSE /\ v4=FALSE
S_1 ==  /\ v1=TRUE /\ v2=FALSE /\ v3=FALSE /\ v4=FALSE
S_2 ==  /\ v1=TRUE /\ v2=TRUE /\ v3=FALSE /\ v4=FALSE
S_3 ==  /\ v1=TRUE /\ v2=FALSE /\ v3=TRUE /\ v4=FALSE
S_4 ==  /\ v1=TRUE /\ v2=TRUE /\ v3=TRUE /\ v4=FALSE
R_0 == v1' = IF Init THEN ~v1 ELSE v1
R_1 == v2' = IF S_1 THEN ~v2 ELSE v2
R_2 == v3' = IF S_1 THEN ~v3 ELSE v3
R_3 == v3' = IF S_2 THEN ~v3 ELSE v3
R_4 == v2' = IF S_3 THEN ~v2 ELSE v2
R_5 == v4' = IF S_4 THEN ~v4 ELSE v4
Next == /\ R_0
 /\      \/ (/\ R_1/\ R_3)
         \/ (/\ R_2/\ R_4)
         /\ R_5
Spec == Init/\[][Next]_<<v1,v2,v3,v4>>
```