UDC 004.777
Preliminary Communication

The Investigation of TLC Model Checker Properties

Vadym Viktorovych Shkarupylo *vadshkar@yandex.ua*
Computer Systems and Networks Department
Zaporizhzhya National Technical University,
Zaporizhzhya, Ukraine

Igor Tomičić *igor.tomicic@foi.hr*
Faculty of Organization and Informatics
University of Zagreb, Varaždin, Croatia

Kostiantyn Mykolaiovych Kasian *konst_k@yahoo.com*
Computer Systems and Networks Department
Zaporizhzhya National Technical University,
Zaporizhzhya, Ukraine

Abstract

This paper presents the investigation and comparison of TLC model checking method (TLA Checker) properties. There are two different approaches to method usage which are considered. The first one consists of a transition system states attendance by breadth-first search (BFS), and the second one by depth-first search (DFS). The Kripke structure has been chosen as a transition system model. A case study has been conducted, where composite web service usage scenario has been considered. Obtained experimental results are aimed at increasing the effectiveness of TLA+ specifications automated verification.

Keywords: Composite Web Service, Model Checking, WS-BPEL, BFS, DFS, TLA+, TLC.

1. Introduction

Model checking methods are being widely used in different spheres of engineering such as distributed software systems, embedded systems, and similar [1]. One of the most topical spheres of such methods usage is checking the correctness of software design solutions to avoid subsequent errors and inconsistencies. A demonstrative example of the practical implementation of this approach can be observed in Amazon Web Services [2].

A numerous modern web-oriented distributed software systems are based on Service-oriented Architecture (SOA) principles: loose coupling, composability, etc. [3]. Such systems are typically involved in the diverse business processes (tickets booking, logistics scenarios, etc.) maintaining and/or implementing. To perceive the advantages of SOA-principles, Google Maps services can be took into consideration [4].

In general, web services are typically considered to be a loosely coupled software components, which can be atomic, or coupled entities. This paper focuses on coupled ones which are better known as composite web services, and which functioning can be implemented in centralized or decentralized manner. Authors of this paper consider centralized orchestration model, described in WS-BPEL-specification [5].

One of the distinctive features of model checking methods consists of the orientation on temporal logic or process calculi, e.g. Linear Temporal Logic (LTL) and the appropriate SPIN (Simple Promela Interpreter) model checker, Computational Tree Logic (CTL), paired with modern NuSMV (New Symbolic Model Verifier) tool, which can be successfully applied to CTL- as well as to LTL-based specifications [6, 7]. Well-known examples of process calculi are Calculus of Communicating Systems (CCS, by R. Milner) and Communicating Sequential Processes (CSP, by C.A.R. Hoare) [8, 9]. The CCS itself is the basis of LOTOS (Language of Temporal Ordering Specification) formalism [10]. Its specifications are typically verified with CADP (Construction and Analysis of Distributed Processes) toolkit [11].

Despite the broad range of available different formalisms and model checking techniques, we were inspired by the success of rigid TLA+ (Temporal Logic of Actions, by L. Lamport) formalism adoption in Amazon Web Services engineering process [12, 2]. This formalism is incorporated with corresponding model checking method, TLA Checker (TLC) [13]. At the same time, in order to leverage more effectiveness from automated TLC-verification, depending on specifications properties, some tweaking may be conducted. To this end, this paper is aimed at increasing the effectiveness of TLC model checker practical usage.

2. Technique Description

We have considered two approaches to TLC method usage: the breadth-first search (BFS) of transition system states and the depth-first search (DFS).

As effectiveness criterion we used the relation between time spent on automated BFS-driven verification, and time spent on automated DFS-driven one.

The Kripke structure has been chosen as a transition system model [14]:

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

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, AP – atomic prepositions set.

Let

$$AP = V \times D, (2)$$

where $V = \{v_i | i = 1,2,...,n\}$ – state variables set: $n = |V|$ is the amount of atomic web services intended to be composed by orchestration-driven coordination; $D = \{0,1,2\}$ – set of state variables values. Each state variable represents corresponding atomic web service.

Let us assume that current state $s \in S$ change in structure (1) is a representation of some atomic web service invocation, denoted in WS-BPEL-description with corresponding <invoke> tag. As a result of such invocation the subsequent state $s' = R(s) \in S$ should be achieved. In this context the elements of AP set should be interpreted as following:

- $(v_i, 0) \in AP$ – the invocation of i^{th} service hasn't yet been conducted;
- $(v_i, 1) \in AP$ – the invocation of i^{th} service has already been accomplished; service functioning started;
- $(v_i, 2) \in AP$ – i^{th} service functioning completed.

Composite web service WS-BPEL-description provides input data. An appropriate content can be parsed on two distinct groups – basic activities and structured activities. We took one representative from each group: the <invoke> tag – from basic activities group – and the <sequence> tag – from structured activities group (Appendix A). Despite the <sequence> tag, which is a sequence invocation template, the corresponding group also includes formalizations of switch/case- and flow-constructs. The additional constructs are being considered by the authors of this paper as a ground for future research.

Our technique to TLA+ specifications' synthesis is based on the following atomic prepositions' interpretation:

- specify the initial state $s_0 \in S = \{s_0, s_1, ..., s_{2\cdot n}\}$ – *Init* statement – as a conjunction from the elements of $L(s_0)$ set, where $L(s_0) = \{(v_1, 0), (v_2, 0), ..., (v_n, 0)\}$;
- specify $2 \cdot n - 1$ transient states from $S \setminus \{s_0, s_{2\cdot n}\}$ set;
- specify $2 \cdot n$ transitions $(s, s') \in R$, where $s, s' \in S$;
- specify the *Next* statement as a disjunction from transitions specifications;
- compose the resulting temporal formula from *Init* and *Next* statements.

The approach to the conduction of the experiment is described hereafter:

- synthesize 10^2 TLA+ specifications for $n = 2^1, 2^2, ..., 2^8$;

- conduct synthesized specifications BFS- and DFS-driven automated TLC-verifications, measuring the corresponding time costs;
- analyze obtained results and formulate conclusions/recommendations.

Let's consider a synthetic example, where composite web service functional property is implemented by way of two atomic web services sequential invocation. As a case study, the π value calculation with John Machin's formula can be contemplated: one atomic service is used for the pair of arctangents calculation (which can itself be a composite entity), and another one for corresponding multiplications and subtraction.

In this case $V = \{v_1, v_2\}$, $n = 2$, $AP = \{(v_1,0),(v_1,1),(v_1,2),(v_2,0),(v_2,1),(v_2,2)\}$, $S = \{s_0, s_1, s_2, s_3, s_4\}$, where $s_0 \in S$ – initial state: $L(s_0) = \{(v_1,0),(v_2,0)\}$; $s_1, s_2, s_3 \in S$ – transient states; $s_4 \in S$ – final state to be reached: $L(s_4) = \{(v_1,2),(v_2,2)\}$.

The appropriate TLA+ specification is given in Appendix B, where *Spec* is the resulting temporal formula to be checked.

TLA+ specifications for $n > 2^1$ are obtained in the same manner. For instance, for $n = 2^8$ it should reach exactly $2 \cdot n + 1 = 513$ states, including initial and final ones.

3. Results Analysis

The experiment was conducted in an automated single-threaded manner (Figure 1, Table 1).

The platform that was used for the purpose of conducting the experiment has the following configuration:

- CPU: AMD K10, 3 GHz;
- RAM: DDR3, 2 GB, dual channel;
- OS: MS Windows 7;
- Java Runtime Environment: version 1.7.

In Figure 1, each reference point (given in Table 1) represents the average of 10^2 measures.

For relatively small numbers of state variables ($n = 2^1, 2^2 \ldots, 2^4$), in terms of corresponding time costs, we have contemplated the preference of DFS-driven approach: its effectiveness gains estimations (t_{BFS} / t_{DFS}) ranged from 1,5 to 2,23. Such approach automation, though, is affected by a necessity to denote the depth of search, which is not the case for BFS-driven verification.

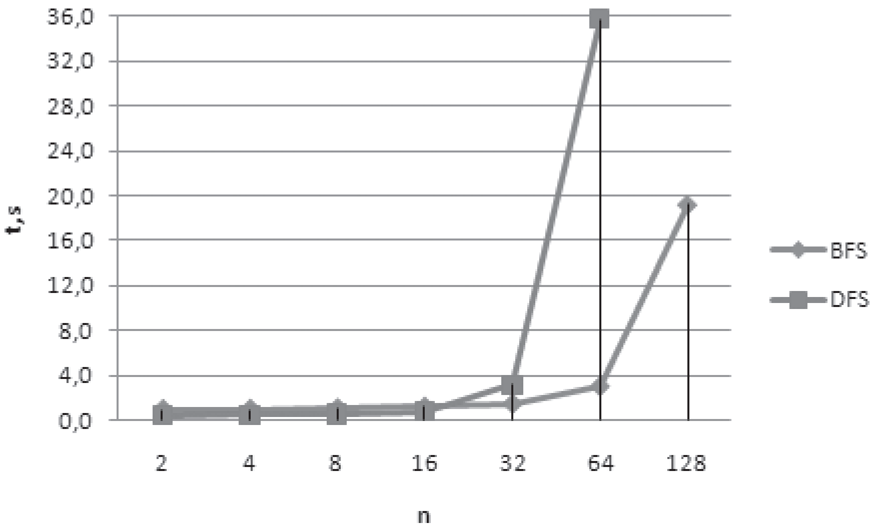

Figure 1. Time costs of BFS- and DFS-driven automated TLC-verifications

n	t_{BFS}, s	t_{DFS}, s	t_{BFS} / t_{DFS}
2	0,934	0,42	2,23
4	0,952	0,45	2,13
8	1,029	0,54	1,91
16	1,154	0,77	1,50
32	1,412	3,17	0,45
64	2,97	35,75	0,08
128	19,21	-	-

Table 1. BFS- and DFS-approaches comparison

In Table 1, t_{BFS} represents time spent on the BFS-driven automated TLC-verification, and t_{DFS} represents time spent on the DFS-driven one.

From Figure 1 we can conclude that for $n > 20$ the BFS-driven approach to TLC-verification is more preferred. For instance, for $n = 64$ this approach is about 12 times faster (more efficient) than the DFS-driven alternative – based on the automatically generated TLA+ specifications with solely sequential structure. We characterized this effectiveness difference as significant enough as to not conduct the DFS-devoted experimentation for $n = 2^7$.

When trying to conduct the BFS-driven verification for $n = 2^8$, the limitation of random access memory volume has been faced.

4. Conclusion and Further Research

In this paper authors have conducted the investigation of TLC model checking method properties. The following conclusions were obtained:

1.Using TLA+ specifications with $2^1,...,2^4$ state variables to increase TLC method effectiveness, it is recommended to adopt the depth-first search approach. The effectiveness increase factor ranges from 2.23 to 1.5, respectively, in comparison to alternative breadth-first search approach. The DFS-based verification also involves the necessity to previously specify the depth of the state space, which can impose certain inconveniences with automation in mind.

2.Using TLA+ specifications with more than 20 state variables to increase TLC method usage effectiveness, the BFS-driven approach is recommended, with no need to specify state space depth.

3.When trying to verify TLA+ specifications with 2^8 state variables, the lack of random access memory quantity (2GB) has been faced.

In order to get a more complex picture about TLC method properties, further research will be aimed at expanding proposed technique to TLA+ specifications synthesis by adopting not only sequential WS-BPEL-constructs, but also switch/case- and flow-structured activities.

Appendix A: WS-BPEL-description template

```
<bpel:sequence>
     <bpel:invoke.../>
     <bpel:invoke.../>
</bpel:sequence>
```

Appendix B: TLA+ specification for WS-BPEL-template

```
EXTENDS Naturals
VARIABLES v1, v2
Invariant == /\ v1 \in (0..2) /\ v2 \in (0..2)
Init == (v1=0) /\ (v2=0)
S1 == (v1=1) /\ (v2=0)
S2 == (v1=2) /\ (v2=0)
S3 == (v1=2) /\ (v2=1)
R_01 == /\ v1' = IF Init THEN v1+1 ELSE v1
     /\ UNCHANGED <<v2>>
R_12 == /\ v1' = IF S1 THEN v1+1 ELSE v1
     /\ UNCHANGED <<v2>>
R_23 == /\ v2' = IF S2 THEN v2+1 ELSE v2
```

```
      /\ UNCHANGED <<v1>>
R_34 == /\ v2' = IF S3 THEN v2+1 ELSE v2
        /\ UNCHANGED <<v1>>
Next == R_01 \/ R_12 \/ R_23 \/ R_34
Spec == Init/\[][Next]_<<v1,v2>>
```

References

[1]S.A. Seshia et al., “Formal Methods for Semi-Autonomous Driving”, In *Proceedings of the 52nd Annual Design Automation Conference*, San Francisco, CA, USA, June 07–11, 2015. DOI: 10.1145/2744769.2747927

[2]C. Newcombe et al., “How Amazon Web Services Uses Formal Methods”, *Communications of the ACM*, 58(4): 66–73, 2015. DOI: 10.1145/2699417

[3]M.P. Papazoglou et al., “Service-Oriented Computing: State of the Art and Research Challenges”, *IEEE Computer*, 40(11): 38–45, 2007.

[4]S. Hu and T. Dai, “Online Map Application Development Using Google Maps API, SQL Database, and ASP.NET”, *International Journal of Information and Communication Technology Research*, 3(3): 102–110, 2013.

[5]M. Fahad et al., “Dynamic Execution of a Business Process via Web Service Selection and Orchestration”, In *Proceedings of the International Conference On Computational Science*, pages 1655–1664, Reykjavik, Iceland, 2015. DOI: 10.1016/j.procs.2015.05.299

[6]A.R. Espada et al., “Using Model Checking to Generate Test Cases for Android Applications”. In *Proc. of 10th Workshop on Model-Based Testing*, pages 7–21, London, UK, 2015. DOI: 10.4204/EPTCS.180.1

[7]M. Norouzi and S. Parsa, “Verification of the Protection Services in Antivirus Systems by Using NuSMV Model Checker”. *International Journal in Foundations of Computer Science & Technology*, 4(5): 57–67, 2014.

[8]C.A.R. Hoare, “Communicating Sequential Processes”. *Communications of the ACM*, 21(8): 666–677, 1978.

[9]R. A Milner, “Calculus of Communicating Systems”. *ECS-LFCS-86-7 Report*, Department of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh, 1986.

[10]J. Schot, “Addressing performance requirements in the FDT-based design of distributed systems”. *Computer Communications*, 15(4): 235–242, 1992. DOI: 10.1016/0140-3664(92)90106-O

[11]H. Garavel et al., “CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes”. *International Journal on Software*

Tools for Technology Transfer, 15(2): 89–107, 2013. DOI: 10.1007/s10009-012-0244-z

[12]M. Cristia, “A TLA+ Encoding of DEVS Models”. In *Proceedings of the International Modeling and Simulation Multiconference*, pages 17–22, Buenos Aires, Argentina, 2007.

[13]L. Lamport, “Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers”. *Addison-Wesley*, Boston, 2002.

[14]E.M. Clarke et al., “Model Checking”. *MIT Press*, Cambridge, Massachusetts, 2001.