004.052.42

TLC MODEL CHECKING AND THE CONCURRENCY IN SPECIFICATION

Shkarupylo Vadym

Zaporizhzhya National Technical University, Computer Systems and Networks Department

Abstract

The TLC Model Checker has been considered. The investigation of corresponding time costs has been conducted. In the case study conducted the Composite Web Service temporal specifications with concurrency have been considered. Two approaches to model checking have been compared – the breadth-first-search- and the depth-first-search-based.

Анотація

Розглянуто метод перевірки на моделі TLC. Проведено дослідження відповідних часових витрат. У проведеному дослідженні розглянуто темпоральні специфікації композитного веб-сервісу з елементами паралелізму. Порівняно два підходи до верифікації (перевірки на моделі) – BFS- та DFS-орієнтований.

Introduction

Nowadays model checkers are ubiquitously utilized in different spheres. One of those is distributed software systems engineering. To confirm that system specification meets the requirements the formal verification is conducted. The common way to do that in an automated manner is to use some model checking method. In this paper the TLC (TLA Checker) method is being considered [1]. This method is proved to be a useful instrument to find flaws in system design [2]. Rigidly such specifications can be represented as compositions of parallel and sequential structures.

Let's consider the iterative model of Software Development Life Cycle (SDLC) [3]. In this model each iteration is represented with a sequence of the following phases: requirements analysis, design, implementation, testing. Let's consider the second phase on which different models are being created to check the design solutions. One (some) of those is (are) formal model(s) – specification(s) – intended to be used for verification purposes. In context of Web Services and Cloud Computing SDLC can be considered in a different way – from consumer's viewpoint [4].

There is no doubt that verification procedure makes its contribution into the overall SDLC-related time costs – especially with iterative SDLC-model in mind. Verification-related time costs can be significantly reduced by tweaking the model checker in accordance with particular specification properties. The corresponding research of TLC model checker has already been conducted – the key accent has been put on the dependency of verification time costs from the number of state variables [5]. Two approaches to TLC usage have been considered – transition system states attendance by way of breadth-first search (BFS) and by way of depth-first search (DFS).

In given work the focus is put on specification structure. The main insight here is to determine the impact of formal specification structure on corresponding automated verification time costs. The description of the case study conducted is given below.

Problem Statement and the Case Study

Let's suppose that we have the specifications (formal models) created with TLA+ formalism (Temporal Logic of Actions, by L. Lamport) [1]. Let's consider a Composite Web Services (CWS) usage scenario [6]. The CWS is represented with WS-BPEL-description (Web Services Business Process Execution Language) which is the implementation of centralized orchestration model [7]. Here the CWS components – another CWSs or atomic web services –

are represented with <invoke> tags – the elements of Basic Activities group which form CWS components set. To form the structure of CWS, represented in WS-BPEL-description, the <sequence> and <flow> tags are used. Each <invoke> tag is represented in TLA+ specification with a state variable [5]. The Kripke structure on a set of atomic prepositions has been chosen as a transition system model [8].

The concurrency has been represented in specifications as interleaving. The concurrency structure has been chosen to be a binary tree-like. That means that after the invocation of some first component the following two components should be invoked concurrently. After that another four components have to be invoked concurrently, and so on – until there is only one component left – to gather the intermediate results. To this end to conduct the case study the numbers of CWS components (the numbers of state variables) $n \in N$ have been chosen from the sequence $n = 2^2, 2^3, 2^4$.

The complexity estimations of model checking tasks to be solved are given in table 1 and table 2.

Table 1 – The BFS-to-DFS states number 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 – number of states generated by TLC during model checking; Found – number of states that satisfy the temporal formula; Depth – the depth of state space search – the number of states to be visited (including the initial one and the final one) to accomplish distinct interleaving scenario. The depths for all possible interleaving scenarios are equal. It should be noticed that the number of generated states while DFS-driven model checking is significantly bigger comparing to BFS-driven approach. This in theory should induce the extra memory demands for DFS-driven approach. Such assumption has already been confirmed though [5].

Table 2 – The estimations of model checking tasks complexities

n	**Paths length (Depth)**	**Paths number**
2^2	5	2
2^3	9	48
2^4	17	1935360

The model checking has been successfully conducted for $n = 2^2, 2^3$. For $n = 2^4$ the limitation of random access memory (RAM) has been faced – due to significant number of paths to be checked (table 2).

The case study has been conducted on the following platform: CPU – AMD K10, 3 GHz; RAM – DDR3, 2 GB; OS – MS Windows 7; JRE v. 1.7; TLC v. 2.05.

For $n = 2^4$ the volume of JVM-accessible memory buffer has been changed from 256 to 1280 MB. The corresponding BFS- and DFS-related time costs were almost identical. Nevertheless, the model checking task hasn't been solved for such n.

The results obtained are given in table 3.

Table 3 – The time costs of BFS- and DFS-driven model checking

n	**time costs, s**		
	$\bar{t}_{BFS}$	$\bar{t}_{DFS}$	$\bar{t}_{BFS} / \bar{t}_{DFS}$
2^2	0,926	0,425	2,179
2^3	1,094	0,623	1,756
2^4	-	-	-

In table 3 $\bar{t}_{BFS}$ ($\bar{t}_{DFS}$) – the average (of 10^2 measures) time costs of BFS- (DFS-) driven model checking. The time costs for $n = 2^2, 2^3$ with a concurrency in mind are similar to the ones obtained earlier for the specifications with purely sequential structure [5]. The BFS-driven approach allows to successfully verify the specification with sequential structure for $n = 2^7$. The DFS-driven one is inapplicable for such state variables number with memory buffer size available. It should be noted, however, that in case of specifications with concurrency the growth of DFS-related time costs is a bit quicker.

Conclusions

To sum up, there is no significant difference between BFS- and DFS-driven approaches to TLC model checking in terms of practical usage with concurrency in mind. However, this is not the case when talking about purely sequential specifications, where less memory-demanding BFS-driven approach allows to successfully verify specifications with more state variables. The whole picture is similar to the one obtained earlier for sequential specifications: BFS-driven approach is less memory demanding but more time consuming; DFS-driven one – vice versa.

References:

1. Lamport L. Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers / L. Lamport. – Boston : Addison-Wesley, 2002. – 364 p. – ISBN 0-321-14306-X.

2. Newcombe C. How Amazon Web Services Uses Formal Methods / C. Newcombe, T. Rath, F. Zhang [et al.] // Communications of the ACM. – 2015. – Vol. 58, No. 4. – P. 66–73. DOI: 10.1145/2699417.

3. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development / C. Larman. – [3rd ed.]. – New Jersey, USA : Prentice Hall, 2004. – 736 p. – ISBN 0-13-148906-2.

4. Tran H. T. Service Development Life Cycle for Hybrid Cloud Environments / H. T. Tran, G. Feuerlicht // Journal of Software. – 2016. – Vol. 11, No. 7. – P. 704–711. DOI: 10.17706/jsw.11.7.704-711.

5. Shkarupylo V. V. The investigation of TLC model checker properties / V. V. Shkarupylo, I. Tomičić, K. M. Kasian // Journal of Information and Organizational Sciences. – 2016. – Vol. 40, No. 1. – P. 145–152.

6. Shkarupylo V. A Technique of DEVS-Driven Validation / Vadym Shkarupylo // Proc. XIIIth Int. Conf. on Modern Problems of Radio Engineering, Telecommunications, and Computer Science, TCSET'2016 (Lviv-Slavske, Ukraine, February 23–26, 2016). – P. 495–497. DOI 10.1109/TCSET.2016.7452097.

7. Web Services Business Process Execution Language Version 2.0 [Electronic resource] : OASIS Standard, April 11, 2007. – Access mode : http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf. – Title from screen.

8. Clarke E.M. Model Checking / E.M. Clarke, O. Grumberg, D. Peled. – Massachusetts : The MIT Press, 1999. – 330 p. – ISBN 9780262032704.