Model Checking is Possible to Verify Large-scale Vehicle Distributed Application Systems

Haitao Zhang[1], Ayang Tuo[2] and Guoqiang Li[3]
[1,2]School of Information Science and Engineering, Lanzhou University, China, 730000
[3]School of Software, Shanghai Jiao Tong University, China, 200240
htzhang@lzu.edu.cn

***Abstract*—OSEK/VDX is a specification for vehicle-mounted systems. Currently, the specification has been widely adopted by many automotive companies to develop a distributed vehicle application system. However, the ever increasing complexity of the developed distributed application system has created a challenge for exhaustively ensuring its reliability. Model checking as an exhaustive technique has been applied to verify OSEK/VDX distributed application systems to discover subtle errors. Unfortunately, it faces a poor scalability for practical systems because the verification models derived from such systems are highly complex. This paper presents an efficient approach that addresses this problem by reducing the complexity of the verification model such that model checking can easily complete the verification.**

I. Introduction

With the development of automotive industry and electronic technology, more and more complex vehicle-mounted systems are deployed in vehicles. However, how to reuse and transplant the developed systems has become a serious problem for automotive manufacturers, since there is no an uniform development standard in the automotive industry. To finish off this problem, the association of European automotive manufacturers develops and promulgates a standard named OSEK/VDX [1] in 1994. The standard presently has gained widespread usage by automotive manufacturers automotive manufacturers such as BMW, Audi and Volkswagen.

As shown in Fig. 1, an OSEK/VDX vehicle-mounted system generally runs on several processors and consists of three primary components: the operating system (OS), multi-tasking application and communication protocol. The OS located at different processors is in charge of dispatching tasks within the application to execute on a processor, especially a *deterministic* scheduler (static priority scheduling policy) is adopted by the OSEK/VDX OS to dispatch tasks. The application is to realize the concrete functions, which often interacts with each other via a communication protocol such as the controller area network (CAN) [2]. There are two complex execution characteristics in OSEK/VDX vehicle-mounted systems: (i) tasks within an application concurrently execute on a processor under the scheduling of the OSEK/VDX OS; (ii) applications simultaneously run on the different processors and sometimes communicate with each other. Due to the concurrency of tasks and simultaneity between applications, how to exhaustively verify a developed OSEK/VDX distributed application system has become a challenge for developers with the increasing developmental complexity. To exhaustively verify OSEK/VDX distributed application systems such as safety properties and timing properties, model checking [3][4] has attracted much attention in the automobile industry as a promising technique.

By using existing model checkers to verify OSEK/VDX distributed application systems, a plain checking method [5] has been proposed based on the model checker UPPAAL. Although the plain checking method is in a position to verify OSEK/VDX distributed application systems, it is often not capable of dealing with a large-scale system that holds a number of multi-tasking applications running on the several processors. In order to make model checking more scalable for dealing with large-scale OSEK/VDX distributed application systems, this paper presents an efficient approach that aims to reduce the number of concurrent `processes` by means of a sequentialization technique. A series of experiments have been carried out to evaluate the proposed approach on the realistic OSEK/VDX distributed application systems. The experimental results show that the sequentialization checking approach is capable of efficiently simplifying the verification models of OSEK/VDX distributed application systems and can moreover expand the scalability of model checking in verifying large-scale OSEK/VDX distributed application systems compared with the plain checking method.

II. Sequentialization Checking Approach

A. Sequential Translation

The sequentialization checking approach is characterized by its capability to reduce the number of concurrent processes within the verification model by means of a sequentialization idea in this paper. In the approach, each multi-tasking application within a target OSEK/VDX distributed application system is first translated into an equivalent sequential model, which makes only one concurrent `process` sufficient to simulate the multi-tasking application rather than multiple concurrent `processes`. The key processes of the sequentialization translation for an OSEK/VDX multi-tasking application are as follows. In the first step, tasks within a multi-tasking application (coded in C) is interpreted into corresponding control flow graphs (CFGs) by the aid of C Intermediate Language (CIL) [6]. Based on the CFGs of the tasks, the application is then translated into a sequential model. In the translation process, the application is symbolically executed based on an extended directed graph in order to maintain the equivalence between

978-3-9819263-2-3/DATE19/©2019 EDAA

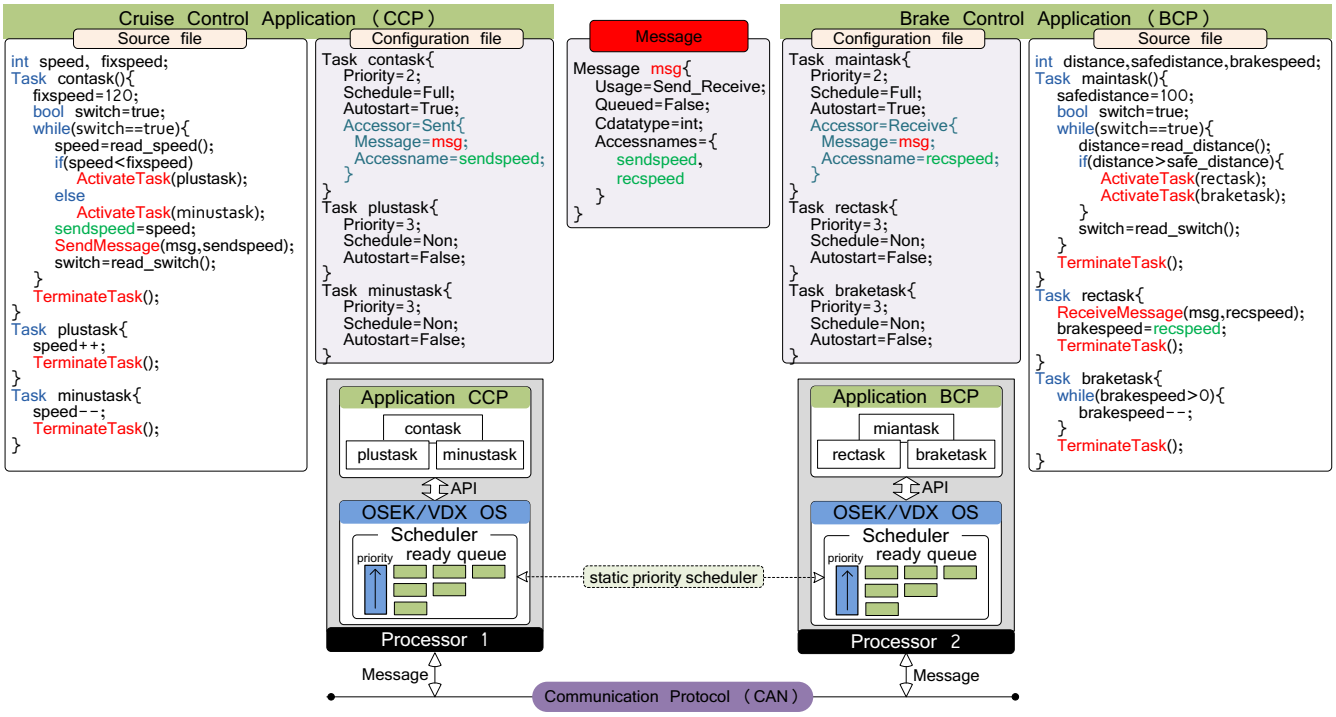

Fig. 1. Running Example: an OSEK/VDX distributed application system which consists of 2 applications (CCP and BCP) and runs on 2 processors.

the given application and the sequential model. In particular, an OSEK/VDX OS model is embedded in the sequential algorithm to explicitly dispatch tasks and respond to the APIs invoked in tasks.

1) Task CFG: C programming language is commonly used to implement an OSEK/VDX program in industry. However, as well know that the C code is often too complex to make a static analysis, e.g., to identify a loop. To sequentialize an OSEK/VDX program coded in C language in automatic way, tasks within the program are first convert into CFGs. The definition of a task CFG is shown in Definition 1. For instance, the corresponding task CFGs for the tasks $contask$, $minustask$ and $plustask$ within the running example have been depicted in Fig. 2.

Definition 1. *The task CFG is a table* $\Upsilon = (\Theta,\ \mathcal{L},\ L_0,\ \Omega)$. *Where,* Θ *is a set of task statements, and the expression of a statement* $\vartheta \in \Theta$ *is as follows:*

$$\vartheta ::= condition \,|\, assignment \,|\, goto \,|\, \texttt{API}$$

$\mathcal{L}=\{L_0, L_1, L_2, L_3 \cdots\}$ *is a set of locations for task statements* Θ. $L_0 \in \mathcal{L}$ *is the start location.* $\Omega \subseteq \mathcal{L} \times \mathcal{L}$ *is a set of directed edges used to label task statements* Θ.

Definition 2. *The extended directed graph* G *is a triple* $G = (V,\ v_0,\ E)$. *Where,* V *is a set of nodes with the start node* $v_0 \in V$. *A node* $v \in V$ *consists of two variables* p *and* D, p *is an array with the size of task number used to record the current statement locations of tasks, and* D *is a set of data structures used to store the scheduling data.* $E \subseteq V \times V$ *is a set of directed edges used to map the statements of tasks.*

2) Sequentialization: There are two problems that should be addressed in order to obtain an equivalent sequential model through sequentializing an OSEK/VDX program: one is how to explicitly realize the scheduling behaviours of OSEK/VDX OS, and the other is how to compute the sequential model. As to solve these two problems, an OSEK/VDX OS model is embedded in the sequential translation module to explicitly perform the scheduling behaviours. The embedded OS model [5] is a component of the sequential translation module, consisting of several data structures and functions. The data structures are in charge of recording the scheduling data such as states of tasks. The functions are responsible for determining running task and updating the data of data structures according to the invoked APIs. In addition, an extended directed graph shown in Definition 2 is employed to implement the computation of sequential model.

In the sequential translation process, the extended directed graph is used to execute an OSEK/VDX program in a symbolic way, and the sequential translation module calls the embedded OS model to dispatch tasks and respond to the invoked APIs when meeting an API in the sequentialization process, i.e., the context switch of tasks in OSEK/VDX programs occurs at the invoked points of APIs. Based on the extended directed graph and the embedded OSEK/VDX OS model, the key processes of the sequential translation are formalized in our previous work [7].

B. Verification Model

By using the sequential translation, the applications CCP and BCP included in the running example can be translated

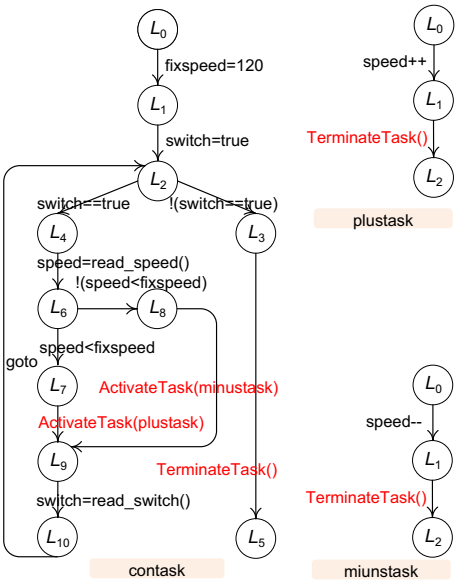

Fig. 2. Task CFGs for the Application CCP.

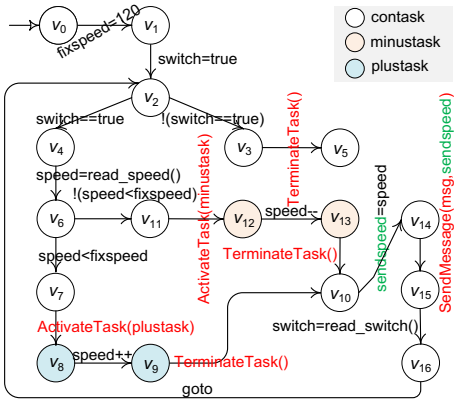

Fig. 3. Sequential model for the application CCP.

into two sequential models. For example, as shown in Fig. 3, the application CCP can be translated into the corresponding sequential model. Based on the sequential models of applications CCP and BCP, a verification model for the running example can be constructed. The constructed verification model includes three concurrent `processes`: one is for CAN, and the other two `processes` are for the sequential models of the applications CCP and BCP. After that, a model checker such as Spin can be employed to carry out verification, e.g., to verify whether a declared variable satisfies its specification.

III. Experiment and Evaluation

A. Experiments and Benchmarks

Based on the implemented tool, a series of experiments are conducted to evaluate the efficiency and scalability of the sequentialization checking approach. In the experiments, all OSEK/VDX applications within an experimental distributed system are first translated into sequential models, and then the model checker Spin is employed to carry out verification. The task number and processor number are the primary factors to influence the verification performance. The developed OSEK/VDX distributed application systems with different numbers of tasks and processors are selected as benchmarks for the experiments. In addition, the selected benchmarks also reflect all execution situations by additionally taking into account the non-preemptive scheduling behaviors, full-preemptive scheduling behaviors, mix-preemptive scheduling behaviors, synchronous behaviors, and accessing shared resource behaviors. In the benchmarks, each task holds more than 100 states, and the communication protocol CAN is used to transmit messages.

All the experiments are conducted on the Intel Core(TM)i7-3770 CPU with 8GB RAM. In the experiments, the time limit and memory limit are set to 2000s and 2GB, respectively. The plain checking method is considered as a comparison object. To investigate the verification scalability of the sequentialization checking approach and plain checking method, the properties are not given in some benchmarks in order to allow the model checker Spin to explore all of the possible states of the target benchmarks. In contrast, a satisfiable or an unsatisfiable property specified as an assertion is inserted into some benchmarks to evaluate whether the sequentialization checking approach is an efficient technique to discover errors. All of the experimental results are listed in TABLE I. In the table, the bold numbers indicate the experiments with given properties compared with the experiments without given properties. `#n` is the number of processors, `#t` is the number of tasks in each processor, and `#s` is the number of explored states by the model checker Spin. "`MB`" and "`Time`" are the memory consumption and time consumption measured in MByte and seconds, respectively.

B. Discussion

The experimental results indicate that the plain checking method fails to verify the benchmarks that contain several tasks running on a number of processors. This is because, the verification model in the plain checking method requires a large number of concurrent `processes` to simulate the target benchmark. In contrast with the plain checking method, the sequentialization checking approach can successfully verify these benchmarks with lower costs in terms of time and memory as well as with less states, except for the benchmark shown in line 20. This is because, in the sequentialization checking approach, multi-tasking applications within the target benchmark are translated into sequential models. As a result, the verification model only uses one concurrent `process` to simulate a multi-tasking application rather than multiple `processes`. Based on this advantage, it is obvious that the task number of an application will not increase the number of concurrent `processes` of the verification model, which is only affected by the number of processors.

IV. Related Work

Currently, there have been many methods that apply model checking techniques to verify the OSEK/VDX standard based

TABLE I
COMPARISON: PLAIN CHECKING METHOD AND SEQUENTIALIZATION CHECKING APPROACH

			Plain Checking Method (Spin)			Sequentialization Checking Approach				
						Sequential Translation		Spin		
Benchmark	#n	#t	#s	Time	MB	Time	MB	#s	Time	MB
Non-preemption Apps	2	1	13578	84	89	0	0	1204	51	60
	2	3	**101221**	**2.21**	**290.2**	18	39	**1794**	**45**	**52**
	2	4	301245	1109	1278	20	47	2890	57	65
Full-preemption Apps	2	5	351680	1534	1821	49	51	4571	59	69
	2	6	382514	1832	2016	64	57	6524	64	72
	2	7	-	T.O.	M.O.	71	68	**8543**	**61**	**69**
Mix-preemption Apps	2	9	-	T.O.	M.O.	101	84	17698	98	101
	2	10	-	T.O.	M.O.	132	94	43211	112	124
	4	2	**304572**	**1423**	**1512**	6	9	**85359**	**143**	**167**
Synchronization Apps	8	2	-	T.O.	M.O.	25	18	158932	499	531
	10	2	-	T.O.	M.O.	32	18	198574	578	712
	12	2	-	T.O.	M.O.	35	18	**219934**	**702**	**921**
Shared-resource Apps	16	4	-	T.O.	M.O.	54	32	325670	1517	1799
	18	4	-	T.O.	M.O.	71	32	364531	1735	1982
	20	4	-	T.O.	M.O.	85	32	**381009**	**1798**	**1990**

software systems. For the developed OSEK/VDX applications, a Spin-based checking method is proposed for checking the safety properties [8]. However, the method is unable to deal with large-scale applications that hold about fifteen tasks because of the large number of concurrent `processes` included in the verification model. To avoid the states of OSEK/VDX OS to be checked in the verification stage, a new technique called execution path generator (EPG) [9] is proposed based on the satisfiability modulo theories (SMT) based bounded model checking. Even so, the method is not efficient to check the applications that hold a number of branches, as the method will spend much time exploring all of the execution paths in order to construct the corresponding transition system (verification model). Compared with the EPG technique, the sequentialization checking approach in this paper uses an extended directed graph to execute the OSEK/VDX applications symbolically rather than exploring the execution paths.

There have been several existing sequentialization techniques that can translate a concurrent program into a sequential model. For example, Cimatti et al. proposed a method [10] to translate the SystemC concurrent programs into sequential programs. Invrso et al. also presented a method [11] to sequentialize the multi-threaded software that conforms to the POSIX standard. However, these existing methods focus on the *non-deterministic* scheduler based concurrent programs, which are not appropriate to sequentialize the *deterministic* scheduler based OSEK/VDX programs. In these existing methods, random numbers are appended in the target programs in order to simulate the *non-deterministic* scheduling behaviors in the sequential translation. Compared with the existing methods, this paper focuses on the *deterministic* scheduler based concurrent programs, and an extended directed graph is employed to compute sequential model, which are different from these existing methods.

V. CONCLUSION

This paper presented a sequentialization checking approach that can make model checking more scalable in the verification of OSEK/VDX distrusted application systems. The approach has been evaluated based on a series of experiments. The experimental results show that the sequentialization checking approach is an efficient technique to verify large-scale OSEK/VDX distrusted application systems.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation of China (Grant Nos. 61602224, 61872232, 61732013) and the Fundamental Research Funds for the Central Universities (Grant No. lzujbky-2018-130).

REFERENCES

[1] J. Lemieux, *Programming in the OSEK/VDX Environment*. Suite 200 Lawrence, KS 66046, USA: CMP, 2001.

[2] K. Etschberger, R. Hofmann, J. Stolberg, "Controller Area Network: Basics, Protocols, Chips and Applications," *International Conference on Networked Sensing Systems*, vol. 8104, no. 1, pp. 37–40, 2001.

[3] E. M. Clarke, O. Grumberg, and D. E. Long, "Model Checking and Abstraction," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 5, pp. 1512–1542, 1994.

[4] E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model Checking: Algorithmic Verification and Debugging," *Communications of the Acm*, vol. 152, no. 11, pp. 74–84, 2009.

[5] L. Waszniowski, J. Krakora, Z. Hanzalek, "Case study on distributed and fault tolerant system modeling based on timed automata," *Journal of Systems and Software*, vol. 82, no. 10, pp. 1678–1694, 2009.

[6] George C. N., Scott M., Shree P. R., and Westley W., "CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs," in *11th International Conference on Compiler Construction*, 2002, pp. 213–228.

[7] H. Zhang, Z. Cheng, G. Li, S. Liu, "autoC: an efficient translator for model checking deterministic scheduler based OSEK/VDX applications," *Science China Information Science*, vol. 61, no. 052102, 2018.

[8] H. Zhang, G. Li, Z. Cheng, J. Xue, "Verifying OSEK/VDX automotive applications: A Spin-based model checking approach," *Software Testing Verification & Reliability*, vol. 28, no. 3, p. e1662, 2018.

[9] H. Zhang, G. Li, D. Sun, Y. Lu, C. Hsu, "Verifying cooperative software: A SMT-based bounded model checking approach for deterministic scheduler," *Journal of Systems Architecture*, vol. 81, pp. 7–16, 2017.

[10] D. Campana, A. Cimatti, I. Narasamdya, M. Roveri, "An Analytic Evaluation of SystemC Encodings in Promela," in *18th SPIN workshop*, 2011, pp. 90–107.

[11] O. Inverso, E. Tomasco, B. Fischer, et al, "Lazy-CSeq: A Lazy Sequentialization Tool for C," in *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2014, pp. 398–401.