BMCLua: Verification of Lua Programs in Digital TV Interactive Applications

Francisco A. P. Januario, Lucas C. Cordeiro,
and Vicente F. de Lucena Jr.
Federal University of Amazonas – UFAM
Email: {franciscojanuario,lucascordeiro,vicente}@ufam.edu.br

Eddie B. de Lima Filho
Science, Technology and Innovation Center
for the Industrial Pole of Manaus – CT-PIM
Email: eddie@ctpim.org.br

***Abstract*—The present paper describes a novel scheme for checking for potential defects in Lua programs, by using Bounded Model Checking (BMC). Such an approach, called BMCLua, translates a Lua program into an ANSI-C one, which is then verified by the Efficient SMT-Based Bounded Model Checker (ESBMC). BMCLua is able to check for safety properties related to array bounds, division by zero, and user-specified assertions, in Lua programs. This paper marks the first application of BMC to Lua programs. The experimental results show that the performance of BMCLua is similar to that of ESBMC, in about 70% of the benchmarks used for evaluation.**

Keywords*—*Digital TV, Model Checking, Lua.

I. Introduction

The advent of digital TV allowed viewers to interact with television programming. The Brazilian Digital TV System (*Sistema Brasileiro de Televisão Digital* - SBTVD), which is based on the Integrated Services Digital Broadcasting (ISDB), enabled the development of interactive applications, with the use of programming languages such as the Nested Context Language (NCL) [1], which is declarative, and the Lua script language [2], which is imperative. As other programming languages, errors that are not detected in Lua programs may cause problems, during application execution in digital TV receivers (e.g., the application may simply freeze).

This paper describes a novel approach for checking possible violations of safety properties in Lua programs, which is based on the Efficient SMT-Based Bounded Model Checker (ESBMC) [3]. Such an approach is implemented in a tool called Bounded Model Checking of Lua programs, which is called BMCLua. Lua is a powerful and light script language, which was designed to extend applications written in other programming languages, mainly for interactive applications and games. In this work, the ANSI-C programming language is used as modelling language for the translator module of the BMCLua tool, since ESBMC supports C/C++ programs. To the best of our knowledge, this is the first work that applies BMC to the Lua programming language.

II. The Lua Programming Language

Lua is a programming language that is typically used for developing games and digital TV applications [4]. In practice, however, it is an extension language that can be used together with other programming languages (e.g., NCL) [1]. It is worth noticing that Lua is quite fast and easy to code, which is very important for the development of digital TV interactive applications, in real-time. The Lua syntax is small, clean, and straightforward. Additionally, it extends features of digital TV applications via the use of the NCLua library (e.g., in order to respond to remote control keys). Some programming errors that are common in Lua programs include incorrect implicit conversion of variable types, returning null from functions with multiple values, and arithmetic overflow. In NCLua scripts, potential programming errors can occur during event handling or graphic-object drawing.

The Lua programming language does not define the data type of a variable declaration, i.e., it is not a strongly-typed language. In Lua, a variable can accept values of different types, hindering the translation of assignment statements. The "table" data type incurs additional complexity in its translation, since it is used to create other structures (e.g., arrays and structs that are typically used in C). Moreover, functions are particularly difficult to translate, because they are considered value types and are used as objects or elements of a table.

III. ESBMC

ESBMC is a Context-Bounded Model Checker based on Satisfiability Modulo Theories (SMT), which is used for ANSI-C/C++ programs [3]. ESBMC verifies sequential and multi-threaded programs and checks for properties related to arithmetic overflow, division by zero, out-of-bounds index, pointer safety, deadlocks, and data races. In ESBMC, the verification process is completely automated and does not require the user to annotate programs with pre/post-conditions.

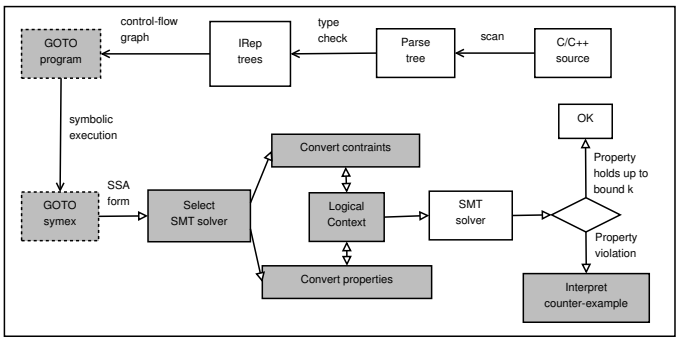

Fig. 1. Overview of the ESBMC architecture.

Fig. 1 shows the ESBMC architecture. As depicted, ESBMC converts an ANSI-C/C++ program into a *GOTO-program*, which simplifies statement representations (e.g., replacement of *while* by *if* and *goto* statements). Then, the *GOTO-program* is executed symbolically by the *GOTO-symex*,

which generates a Single Static Assignment (SSA) form that is later converted into a first-order logic formula; the latter is finally checked by an SMT solver. If a property violation is found, then a counterexample is provided by ESBMC, which assigns values to the program variables to reproduce the error.

IV. ANTLR

The ANother Tool for Language Recognition (ANTLR) [5] is a syntax and lexical analyzer generator for programming languages, which automates the development of languages recognizers (e.g., translators and interpreters). From a grammar that is built from the syntax of a given target language, ANTLR generates classes in Java programming language, which are the lexical analyzer (i.e., the *lexer*) and the syntax analyzer (i.e., the *parser*).

For a translator built by ANTLR, the lexer generates a symbol stream (i.e., *tokens*) from a sequence of input characters. The *parser*, in turn, checks the syntax of the input characters, by analyzing an Abstract Symbol Tree (AST) structure, which is generated by ANTLR. Additionally, from the *parser*, the output can be easily converted to an ANSI-C program or to the syntax of a domain-specific language.

V. BMCLua

The BMCLua is an efficient model checker for programs written in Lua, which was developed using the Java programming language. The basic functions of BMCLua are related to the translation of Lua into ANSI-C code and to the verification of the resulting ANSI-C code, using the ESBMC tool.

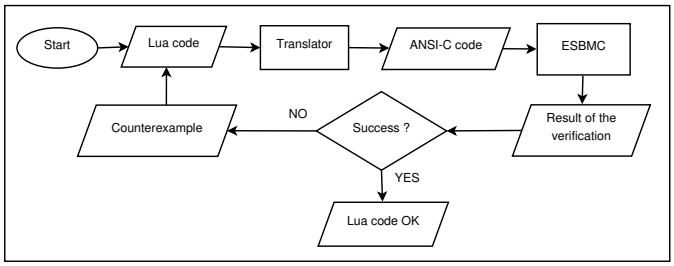

Fig. 2. Overview of the BMCLua architecture.

As shown on the top of Figure 2, BMCLua translates Lua code into ANSI-C code, which is then verified by ESBMC. Here, the counterexample informs the code line where an error occurred, as well as the property violation that was detected. The BMCLua translator is built using the generated classes from the ANTLR tool (i.e., the "parser" and the "lexer").

VI. Experimental Evaluation

The experiments performed with the BMCLua consist of using standard benchmarks from the related literature, in order to check its performance and correctness. The benchmarks include *Bellman-Ford*, *Prim*, *BublleSort*, *SelectionSort*, and *Factorial*. For each benchmark, loop and array limits are defined and verified via assertions.

Table I shows the experimental results, in seconds. The acronym **E** identifies the total number of elements of the *array*, **L** is the total number of lua code lines, **B** shows the limit of performed *loop* iterations, **P** means the total number of checked properties, **TL** is the total verification time, which includes translation and verification, and **TE** is the total verification time used for checking the respective ANSI-C code, in the ESBMC tool, which is used for comparison purposes with **TL**.

TABLE I. Experimental results of the BMCLua tool

Benchmark	E	L	B	P	TL	TE
Bellman-Ford	10	43	11	1	< 1	< 1
	15	43	16	1	< 1	< 1
	20	43	21	1	< 1	< 1
Prim	10	43	11	1	< 1	< 1
	15	43	16	1	< 1	< 1
	20	43	21	1	< 1	< 1
BubbleSort	50	30	51	1	6	5
	70	30	71	1	12	10
	140	30	141	1	68	52
	200	30	201	1	225	163
SelectionSort	50	33	51	1	3	2
	70	33	71	1	6	4
	140	33	141	1	42	25
	200	33	201	1	177	89
Factorial	50	17	51	1	< 1	< 1
	100	17	101	1	< 1	< 1
	150	17	151	1	< 1	< 1
	200	17	201	1	< 1	< 1
	400	17	401	1	2	1

The experimental results show the notable performance of the BCMLua verification time. In particular, the verification time reported in the TL and TE columns are comparable to each other. Additionally, the verification time in the TL column, which is higher than the verification time in the TE one, occurs due to the increase of code lines when translating Lua into ANSI-C code. In most cases, however, the translation time is typically less than one second. In all experiments, BMCLua did not report any false-positive or false-negative result, proving its correctness to verify Lua programs.

VII. Conclusions

The experimental results show the efficiency and correctness of BMCLua to verify Lua programs. In particular, BMCLua is able to detect, in all benchmarks, properties related to division by zero and user-specified assertions without reporting any false-negative result. On average, the verification time of BMCLua is comparable to that of ESBMC; indeed, only 21% of the benchmarks present a verification time that is higher than the ESBMC verification time. Nevertheless, it is worth noticing that improvements in the BMCLua translator could further reduce such a difference. As a future work, BMCLua will be integrated into the Eclipse tool, which will then allow Lua program verification during development. BMCLua will also be integrated into the Ginga *middleware*, in order to check for Lua programs in interactive applications.

Acknowledgements. This research was supported by Samsung, CNPq, and FAPEAM grants.

References

[1] ABNT (Brazilian Association of Technical Standards), *NBR 15606-2:2007: Digital terrestrial television – Data coding and transmission specification for digital broadcasting*. Rio de Janeiro: ABNT, 2007.

[2] J. Kurt and B. Aaron, *Beginning Lua Programming*. Indianapolis: Wiley Publishing, 2007, p. 644.

[3] L. Cordeiro and et al., *SMT-Based Bounded Model Checking for Embedded ANSI-C Software*. In TSE, v. 38, n. 4, pp. 957–974, 2012.

[4] R. Brandão and et al., *Extended Features for the Ginga-NCL Environment: Introducing the LuaTV API*, In ICCCN, pp. 1–6, 2010.

[5] T. Parr, *The Definitive ANTLR Reference - Building Domain-Specific Languages*. North Carolina: The Pragmatic Bookshelf, 2007.