Context-Free Shuffle Languages Parsing via Boolean Satisfiability Problem Solving

Artem Gorokhov
Saint Petersburg State University
Saint Petersburg, Russia
gorohov.art@gmail.com

Semyon Grigorev
Saint Petersburg State University
Saint Petersburg, Russia
semen.grigorev@jetbrains.com

ABSTRACT

In this paper we consider the problem of concurrent programs' model checking from the side of context-free languages shuffle. We argue that this approach ...

CCS CONCEPTS

• **Theory of computation → Grammars and context-free languages**; • **Software and its engineering → Software reliability**;

KEYWORDS

Model checking, static analysis, shuffle, formal languages, language intersection

ACM Reference Format:
Artem Gorokhov and Semyon Grigorev. 2018. Context-Free Shuffle Languages Parsing via Boolean Satisfiability Problem Solving. In *Proceedings of Formal Techniques for Java-like Programs (FTfJP'18).* ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Classic model checking algorithms often use SAT-solvers: the problem of checking of program correctness can be converted to the instance of SAT problem, that describes an intersection of program executions language with regular language of "bad" executions. In single-tread case it is known that the language of program is contained in context-free class, but in interleaved execution scenario things become more complicated. There are a number of papers that describe concurrent programs behavior via Push Down Systems with synchronization actions, but our interest is around a *shuffle* of Context-Free languages(CFL). This languages describe the interleaving of CFL and look perfect to describe the interleaved behavior of concurrent programs.

To explain it we introduce the notion of *shuffle* operation ($\odot$), that can be defined for lines as follows:

- $\varepsilon \odot u = u \odot \varepsilon = u$, for every line $u \in \Sigma^*$;
- $\alpha_1 u_1 \odot \alpha_2 u_2 = \{\alpha_1 w | w \in (u_1 \odot \alpha_2 u_2)\} \cup \{\alpha_2 w | w \in (\alpha_1 u1 \odot u_2)\}, \forall \alpha_1, \alpha_2 \in \Sigma$ and $\forall u_1, u_2 \in \Sigma^*$.

Shuffle extends to languages as $L_1 \odot L_2 = \bigcup\limits_{u_1 \in L_1, u_2 \in L_2} u_1 \odot u_2$.

For example, "ab" $\odot$ "123" $= \{a123b, a1b23, 12ab3, 123ab, etc.\}$.

In paper [2] authors describe an approach to model checking with use of shuffle: context-free grammars of concurrent functions are used. But it is known that the shuffle of context-free languages does not contained in class of CFL and the problem of defining either string is in the shuffle of CFL is NP-Complete. That is why an authors use a context-free approximation of shuffle of CFL and intersect it with error traces, but since the approximation was used this approach didn't found some of known bugs. In this paper we present same approach, but without use of approximation. Instead, we narrow the context-free languages to be shuffled and deal with NP-Completeness with SAT-Solver.

2 LANGUAGES SHUFFLE

Assume functions $f_1, f_2 ... f_n$ that run concurrently and context-free languages $L_{f_1}, L_{f_2} ... L_{f_n}$ generated for each of them. We want to check a correctness of the functions execution. For that we generate regular language R_1 that describes all "bad" executions. For example, if it is needed to check reachability of an *error* operator in program then presented in picture 1 automaton produce the desired language. Now our task is to inspect an intersection $(L_{f_1} \odot L_{f_2} \odot ... \odot L_{f_n}) \cap R_1$ — its emptiness means the program can not reach the *error* state (we note that some conditions preserving the semantics of program should be added to R_1, see the Example section). Since generated languages (both CF and Reg) can potentially generate an infinite number of strings, then representation of this problem in the form of a finite SAT formula is impossible. Therefore, we narrow them to finite.

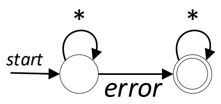

Figure 1: Finite automaton of error traces.

First, we set a restriction on the length of loops in the regular language R_1. This is possible in assumption that the error can usually be detected in the first iterations of the loops. This assumption is used in bounded model checking [?].The result of such bounding is regular language described by finite automaton R_2 without loops that produce finite set of lines.

Then we appeal to the intuition of shuffle operation. If the line J is in the language $B \odot C$ then there is a split of J of J_B and J_C such that $b_1 c_1 b_2 c_2 ... b_k c_k = J$ where $b_i, c_i \in (\Sigma^* \cup \varepsilon)$ and $b_1 b_2 ... b_k = J_B, c_1 c_2 ... c_k = J_C$. Both J_B and J_C are in the language J' — the language of lines J with all possible omissions of terminals. For the finite automaton R_2 the desired language of lines with omissions is described by an automaton R_2' — an epsilon-closure of R_2.

Since the language of R_2' is finite, we can consider the languages $L_{f_i}' = L_{f_i} \cap R_2', i \in 1..n$ — the finite context-free narrowing of languages L_i. Thus, we redefine initial problem of language intersection to $(L_{f_1}' \odot L_{f_2}' \odot ... \odot L_{f_n}') \cap R_2$. All this languages are finite therefore we can generate a SAT problem instance to check intersection emptiness.

An instance of SAT problem is a boolean formula which is checked by solver for satisfiability. Constructing a formula for the problem defined above requires a more deep inspection of the languages' structure. R_2 is a finite automaton that describes multiple paths from initial state to final. Transition labels of this paths define a language os that automaton. In terms of our problem we can consider the terminals of this language as a transitions of the form i_a_j, where i and j are states of R_2; a — transition label. Since the languages L_{f_i}' are $L_{f_i} \cap R_2'$, they describe the sets of transitions in R_2. Thus, the problem of giving the line from the language defined by intersection $(L_{f_1}' \odot L_{f_2}' \odot ... \odot L_{f_n}') \cap R_2$ is equivalent to providing a sets $W_1 ... W_n$ of transitions ($W_i \in L_{f_i}'$) which preserve the following conditions:

- $\bigcap\limits_{i \in 1..n} W_i = \emptyset$, i.e. each transition of R_2 contained not more then in one of the sets W_i.
- $\bigcup\limits_{i \in 1..n} W_i \in R_2$, i.e. union of all transitions is a path in R_2 from initial to final state.

Such problem interpretation intuitively define the rules of the SAT formula generation. The formula consist of following parts, connected with conjunction.

- Define the sets of transitions for each language L_{f_i}' with the alternation of formulas $(t_i^1 \& t_i^2 \& ... \& t_i^k)$, where $t_i^1 ... t_i^k$ are transitions of the same set.
- ...
- ...

3 EXAMPLE

4 CONCLUSION

REFERENCES

[1] Semyon Grigorev and Anastasiya Ragozina. 2017. Context-free Path Querying with Structural Representation of Result. In *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR '17)*. ACM, New York, NY, USA, Article 10, 7 pages. https://doi.org/10.1145/3166094.3166104

[2] Jari Stenman. 2011. Approximating the Shuffle of Context-free Languages to Find Bugs in Concurrent Recursive Programs.