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
s.v.grigoriev@spbu.ru

ABSTRACT

Verification of concurrent systems is important and nontrivial problem. One of directions in this area is modeling of sequential subsystems with push-down automata (PDA) and investigating its communication. PDA is equal to context-free languages and "communication" may be expressed as shuffle of them. In this paper we consider the problem of concurrent programs' model checking from the side of context-free languages shuffle: in order to check correctness of system we should check emptiness of intersection of shuffled context-free languages (which describe behavior of the system) with regular language (which describe set of "bad" behaviors). Even in simple case, when regular language is finite, it leads to NP-complete problem and we show how it can be solved by using SAT-solvers. Our reduction is very native and use classical parsing techniques, such as Shared Packed Parse Forest and Generalized LL parsing algorithm, and some ideas from Context-Free Language reachability framework. We do not propose solution for arbitrary regular language (existence of which looks an open problem) but we show a some possible directions of research and hope that ever for restricted case proposed solution may be useful.

CCS CONCEPTS

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

KEYWORDS

Model checking, static analysis, concurrency, shuffle, formal languages, language intersection, context-free languages

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, 3 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Concurrent systems are widely spread and its verification is a nontrivial and important problem. There are a lot of papers that describe concurrent programs behavior via Push Down Systems or Context-Free languages [2–4, 10], and our interest is around a *shuffle* of Context-Free Languages (CFL) [1]. This languages describe the interleaving of CFLs (or PDA) and look perfect to describe the interleaved behavior of concurrent programs.

First of all we introduce the notion of *shuffle* operation ($\odot$), that can be defined for sequences as follows:

- $\varepsilon \odot u = u \odot \varepsilon = u$, for every sequence $u \in \Sigma^*$;
- $\alpha_1 u_1 \odot \alpha_2 u_2 = \{\alpha_1 w | w \in (u_1 \odot \alpha_2 u_2)\} \cup \{\alpha_2 w | w \in (\alpha_1 u 1 \odot u_2)\}, \forall \alpha_1, \alpha_2 \in \Sigma$ and $\forall u_1, u_2 \in \Sigma^*$.

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

Shuffle can be extended to languages as

$$L_1 \odot L_2 = \bigcup_{u_1 \in L_1, u_2 \in L_2} u_1 \odot u_2.$$

We can describe required aspects of behavior of functions (or methods, or subsystems) $f_1, f_2 \ldots f_n$ from our system $\mathcal{S}$ that run concurrently as shuffle of context-free languages $L_{f_1}, L_{f_2} \ldots L_{f_n}$ generated for each of them. As a result, language $\mathcal{L} = L_{f_1} \odot L_{f_2} \odot \ldots \odot L_{f_n}$ describes all possible executions of our system. If we want to check a correctness of $\mathcal{S}$, then we should check whether $\mathcal{L}$ contains any "bad execution". Let suppose that the set of bad executions can be described by some regular language R_1. Now we should inspect an intersection $\mathcal{L} \cap R_1$ — its emptiness means that $\mathcal{S}$ can not demonstrate bad behavior.

The idea described above is used in the paper [11]. As far as shuffled context-free languages are not closed under intersection with the regular one [1] and the problem of defining either string is in the shuffle of CFL is NP-Complete, 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.

While NP-completeness may looks like death warrant, there are SAT-solvers which deal with NP problems very successfully. In this paper we show how to reduce emptiness checking of shuffled CFL and finite regular language intersection to SAT. Our reduction is very native and use some classical parsing techniques. Generalization for arbitrary regular language is a topic for future research.

2 LANGUAGES SHUFFLE TO SAT

First, we assume that R_1 is finite regular language. This is possible in assumption that the error can usually be detected in the first iterations of the loops, so at the first step we can approximate general regular language by fixed unrolling of loops. This assumption is used in bounded model checking [**?**].

Then we appeal to the intuition of shuffle operation. If the string J is in the language $B \odot C$ then there is a split of J on strings $J_B = b_1 b_2 \ldots b_k \in B$ and $J_C = c_1 c_2 \ldots c_k \in C$ such that $b_1 c_1 b_2 c_2 \ldots b_k c_k = J$

FTfJP'18, July 2018, Amsterdam, Netherlands

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

$(b_i, c_i \in (\Sigma^* \cup \varepsilon))$. Both J_B and J_C are contained in a language of lines J with all possible omissions of terminals. An example of such language for string "abc" is $Y = \{a, b, c, ab, ac, bc, abc, \varepsilon\}$. We can narrow the shuffled languages A and B to $A \cap Y$ and $B \cap Y$ since this languages still contain the strings needed to parse J. This reasoning for J can be extended do the automaton R_1 since it contains multiple strings that we want to check for being in shuffle of $L_{f_1} ... L_{f_n}$. Thus we consider $L'_{f_i} = L_{f_i} \cap R'_1, i \in 1..n$ — the finite context-free narrowing of languages L_i. The desired language of strings with omissions is described by an automaton R'_1 — an transitive-closure of R_1 with ε-transitions.

Described language narrowing is based on intersection of context-free and regular languages, and this problem can be solved with use of an algorithm described in paper [5]. This approach is based on Generalised LL (GLL) [7] and utilizes the Binarized Shared Packed Parse Forest (SPPF) [6, 9]. Binarized SPPF compresses derivation trees optimally reusing common nodes and subtrees, thus utilizing it for parsing forest representation grants worst-case cubic space complexity [7].

or

Binarized Shared Packed Parse Forest (SPPF) [6, 9] compresses derivation trees optimally reusing common nodes and subtrees. Version of GLL [7] which utilizes this structure for parsing forest representation achieves worst-case cubic space complexity [8].

Binarized SPPF can be represented as a graph in which each node has one of four types described below. We denote the start and the end positions of substring as i and j respectively, and we call tuple (i, j) an *extension* of a node.

- **Terminal node** with label (i, T, j).
- **Nonterminal node** with label (i, N, j). This node denotes that there is at least one derivation for substring $\alpha = \omega[i..j-1]$ such that $N \Rightarrow^*_G \alpha, \alpha = \omega[i..j-1]$. All derivation trees for the given substring and nonterminal can be extracted from SPPF by left-to-right top-down graph traversal started from respective node.
- **Intermediate node**: a special kind of node used for binarization of SPPF. These nodes are labeled with (i, t, j), where t is a grammar slot.
- **Packed node** with label $(N \to \alpha, k)$. Subgraph with "root" in such node is one possible derivation from nonterminal N in case when the parent is a nonterminal node labeled with $(\diamond (i, N, j))$.

An example of SPPF is presented in figure **??**. We remove redundant intermediate and packed nodes from the SPPF to simplify it and to decrease the size of the structure.

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:

- $\bigcap_{i \in 1..n} W_i = \emptyset$, i.e. each transition of R_2 contained not more then in one of the sets W_i.
- $\bigcup_{i \in 1..n} W_i \in R_2$, 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. [5] 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 CONCLUSION

We propose the way to reduce emptiness checking of intersection of shuffled CF languages with finite regular one to SAT. We show that result formula has a special structure (huge XOR subformula) which require to use XOR-SAT-solvers. We hope that our restriction on regular language is week enough to solve real tasks. To prove it it is necessary to evaluate our approach on real project.

Main question for future research is decidability of emptiness of shuffled CFL and regular language intersection. It is known that shuffled CFL is not closed under intersection with regular languages [1], but decidability of intersection emptiness is looks an open question. If it will be shown that it is undecidable in general case, then it is interesting to find subclasses for which this problem is decidable.

ACKNOWLEDGMENTS

The research was supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

REFERENCES

[1] Martin Berglund, Henrik Björklund, and Johanna Högberg. 2011. Recognizing shuffled languages. In *International Conference on Language and Automata Theory and Applications*. Springer, 142–154.

[2] Ahmed Bouajjani, Javier Esparza, and Tayssir Touili. 2003. A generic approach to the static analysis of concurrent programs with procedures. *International Journal of Foundations of Computer Science* 14, 04 (2003), 551–582.

[3] Sagar Chaki, Edmund Clarke, Nicholas Kidd, Thomas Reps, and Tayssir Touili. 2006. Verifying concurrent message-passing C programs with recursive calls. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 334–349.

[4] Graeme Gange, Jorge A Navas, Peter Schachte, Harald Søndergaard, and Peter J Stuckey. 2015. A tool for intersecting context-free grammars and its applications. In *NASA Formal Methods Symposium*. Springer, 422–428.

[5] 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

[6] Joan Gerard Rekers. 1992. *Parser generation for interactive environments*. Ph.D. Dissertation. Universiteit van Amsterdam.

[7] Elizabeth Scott and Adrian Johnstone. 2010. GLL parsing. *Electronic Notes in Theoretical Computer Science* 253, 7 (2010), 177–189.

[8] Elizabeth Scott and Adrian Johnstone. 2013. GLL parse-tree generation. *Science of Computer Programming* 78, 10 (2013), 1828–1844.

[9] Elizabeth Scott, Adrian Johnstone, and Rob Economopoulos. 2007. BRNGLR: a cubic Tomita-style GLR parsing algorithm. *Acta informatica* 44, 6 (2007), 427–461.
[10] Fu Song and Tayssir Touili. 2015. Model checking dynamic pushdown networks. *Formal Aspects of Computing* 27, 2 (2015), 397–421.
[11] Jari Stenman. 2011. Approximating the Shuffle of Context-free Languages to Find Bugs in Concurrent Recursive Programs.