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 ... 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 ... 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 ... 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

Concurrent systems are widely spread and its verification is a non-trivial and important problem. 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 CFLs 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 behaviour of functions $f_1, f_2 ... f_n$ from our system $\mathcal{S}$ that run concurrently as shuffle of context-free languages $L_{f_1}, L_{f_2} ... L_{f_n}$ generated for each of them. As a result, language $\mathcal{L} = L_{f_1} \odot L_{f_2} \odot ... \odot L_{f_n}$ describes all possible executions of our system. If we want to check a correctness of $\mathcal{S}$, then we should check wether $\mathcal{L}$ 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 demonstarte bad dehaviour.

The idea described above is used in the paper [1]. As far as shuffled context-free languages are not closed under intersection with the regular one [**?**] 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-completness 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 languge intersection to SAT. Our reduction is veri native and use some clasical 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 fyxed anrolling of loops. This assumption is used in bounded model checking [**?**].

Then we appeal to the intuition of shuffle operation. If the sequence 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

FTfJP'18, July 2018, Amsterdam, Netherlands

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

finite therefore we can generate a SAT problem instance to check intersection emptiness.

3 SHUFFLE TO SAT

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 $(L'_{f_1} \odot L'_{f_2} \odot \ldots \odot L'_{f_n}) \cap R_2$ is equivalent to providing a sets $W_1 \ldots W_n$ 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. 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 $(t_i^1 \& t_i^2 \& \ldots \& t_i^k)$, where $t_i^1 \ldots t_i^k$ are transitions of the same set.
- ...
- ...

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

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 $(N \rightarrow \alpha, k)$. 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.

4 EXAMPLE

5 CONCLUSION

We propose the way to reduce emptyness 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 reqire to use XOR-SAT-solvers. We hope that our restriction on regular language is week anough to solve real tasks. To prove it it is necassary to evaluate our approach on real project.

Main question for future reserch is desidability of emptyness of shuffled CFL and regular language intersection. It is known that shuffled CFL is not closed under intersection with regular languages, but desidability of intersection emptyness is looks an open question. If it will be shown that it is undesidable in general case, then it is interesting to find subclasses for which this problem is desidable.

ACKNOWLEDGMENTS

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

REFERENCES

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