# State-Machine Qualia and it's effects on Context-Free Language Reachability

HENRI MALAHIEUDE, University of Colorado Boulder, USA

Since Thomas Reps' paper, Program Analysis via Graph Reachability, much work has been devoted to optimizing the reachability solver, and refining the graph construction process. However, there has been no study yet, to our knowledge, of the types of analysis for which Context-Free Language Reachability (CFLR) cannot be deployed in the traditional techniques of a Dyck-Language and the Pushdown Automata Reachability Solver. For example, dominance of one basic block on the next is not an analysis for which Dyck-Reachability has been employed for, nor has there been work for why that is. This work argues that Dominance, and other analyses that require reasoning over multiple paths, represent a class of questions that cannot be solved with CFLR. Furthermore, the Automata required to solve the problem can bring insight into the analysis of interest including a best approach to optimizing an algorithm.

## 1 INTRODUCTION

Context-Free Language Reachability, introduced by Thomas Reps [9], is an approach to program analysis that leverages graph theory and context-free languages to achieve complex inter-procedural analysis. In Context-Free Language Reachability (CFLR), each edge in a graph is annotated with a terminal from a Context-Free Language. As a graph is traversed each terminal is concatenated together into a final string, such that only paths which can both be reached by the structure of the graph and produce an accepted word in the language are considered "reachable". Using a simple Dyck-Language of balanced parentheses, inter-procedural analysis is no longer ambiguous: only paths calling function x, and therefore have $(_x$ in the string, may follow the return edge $)_x$.

However, this paper argues that CFLR is limited in the analyses that it can do. Some analyses require reasoning over many paths, but CFLR's current formulation only considers *the existence* of a single path between two vertices. Other analyses do not have a clear "beginning" and "end" to represent with the open and closed parenthesis, which has been the traditional language for many CFLR analyses. An analysis of these limitations, we argue, will help create a new approach for both generating a language for an analysis of interest and studying currently existing analyses.

## 2 OVERVIEW

This paper is organized as follows:

- Section 3 introduces the example which exhibits both limitations for which CFLR does not yet solve for.
- Section 4 proposes an extension to CFLR to solve both limitations.
- Section 5 studies what this extension means for analyses using CFLR through the lens of automata theory.
- Section 6 discusses related works.
- Section 7 summarizes the arguments laid out in this work.

## 3 MOTIVATING EXAMPLE

Context-Free Language Reachability has six varieties [9]:

- All-Pairs, there exists a path between all pairs of nodes.
- Single-Source, there exists a path between a selected source and all other nodes.
- Single-Target, there exists a path between all other nodes to a selected node.
- Single-Source&Target, there exists a path between two specific nodes.

Author's address: Henri Malahieude, henri.malahieude@colorado.edu, University of Colorado Boulder, USA.

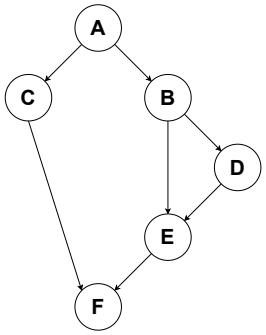


Fig. 1. A Control-Flow Graph for which traditional Context-Free Language Reachability cannot solve Dominance for. Basic Block A dominates E & F, but B only dominates E. C does not dominate anything.

- Multi-Source, there exists a path between multiple sources and all other nodes.
- Multi-Target, there exists a path between all other nodes to multiple targets.

The first limitation of Context-Free Language Reachability (CFLR) is that some analyses require reasoning over multiple paths. It is not just that we must know whether or not a path exists, but the analysis wants to know about the shared properties of these paths. In all varieties listed above, the only property of interest is existence. There is no reasoning about paths themselves.

The second limitation is that the traditional language for CFLR, the Dyck-Language of k numbers of balanced parentheses, only works for analyses that have clear starts and ends. Analyses which are attempting to solve for those points, their input graph could not be annotated with a Dyck-Language without first solving for those start and end points beforehand.

Dominance Analysis on Control-Flow Graphs exhibits both of these limitations. In Figure 1, it is difficult to use parentheses to symbolize dominance. Assume that we could know where to place the symbols to denote the beginning of the dominance and the end of dominance for A. For example, on the edges from A we can place $(_a$ and on the edges into F we can place $)_a$. If their positions are already known, it follows that dominance is already calculated for A since we know the region at which it dominates and does not. Annotating the graph for a separate solver to traverse it would then be redundant work. This is a contradiction that means that we cannot use a balanced language for this analysis.

Applying transformations on the input graph may be the next possible step, but in what way? The more processing steps done to the input graph before the final solver just bypasses the CFLR limitation by not using CFLR.

## 4 EXTENDED CONTEXT-FREE LANGUAGE REACHABILITY

The six previously discussed variations of CFLR in Section 3 are quite powerful for demand-based analysis. By only computing for the existence of a path on nodes of interest, refined answers can be produced with less time than a holistic approach.

For Dominance however they remain insufficient, as the analysis needs to compute all possible paths between two nodes and then find similarities in those paths. To solve the presented issue, a new type of Context-Free Language Reachability (CFLR) analysis is defined as follows. An *All-Paths* variation on Context-Free Language Reachability is a solver divided into two parts, the reachable paths engine, and the paths analyzer.

Applying the All-Paths, and Single-Source selecting node A, variation to Dominance Analysis, we can provide an almost trivial language for CFLR. In Figure 2, each edge is labeled with its source

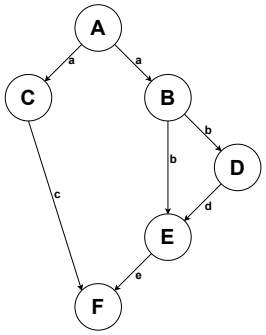


Fig. 2. The CFG from the previous example, where each Basic Block is annotated with the symbol that corresponds to its source Block.

node's name, and all words are accepted for our example Control-Flow Graph. Thus the word is now representing memory as well.

For our example CFG in Fig. 2 all possible words between A and F are:

- ac
- abe
- abde

Our final pass on the paths would be a meet operator which produces all terminals of which are the same between the words. Therefore A dominates F, and we have an algorithm using the CFLR technique to calculate the strict dominance of each node. Dominance is then the simple union of the final calculation with the target node.

## 5 AUTOMATA AND ANALYSIS

### 5.1 Runtime Analysis of the Single-Source All-Paths Dominator Algorithm

The CFLR strict dominance algorithm described above can calculate a target node's dominators in the worst-case sceniario of $O(e \cdot 2 \log v)$, where e is the amount of edges in the graph and v is the total amount of vertices/nodes/basic-blocks. Each path is at worst the total amount of edges within the graph, must then be computed, and finally compared against each other to find similarity. Since a Control-Flow Graph limits outward edges from a vertice to two, the graph may at worst be a Binary-Search Tree. The height of a binary search tree is equivalent to $\log v$ and therefore may also have at most $\log v$ paths to follow. It is also interesting to note that this runtime is the same as the Lengauer-Tarjan Dominance Algorithm [7] whose runtime efficiency is also $O(e \cdot \log v)$.

Note as well that there is a dynamic programming optimization possible for calculating dominators for nodes outside of the target node if the paths pass through them. Since the paths from A to F also include all paths from A to E, calculating the strict dominators of E only considers all words that include its corresponding terminal (e), abe and abde. Then obtain the subtring from the start of the word until before the terminal, ab and abd, and finally apply the final pass on the paths to produce the strict dominators A and B. This refines our algorithm to a Single-Source&Target All-Paths variation of CFLR, from the Entry block to the Exit block.

### 5.2 Connections to Automata

We can relate our algorithm to an automata for more complete reasoning about the analysis.

The runtime for a Regular Expression is at worst $O(w)$ where w is the amount of characters in the word. Considering this, we can construct a regex for our Dominance Path Solver, $L = a((b(d^?e^?)^?)|c)^?$. The path solver for the Control-Flow Graph is equivalent to a Finite State-Machine which represents the regex L. Since we require more than just a single path, we must repeat the regex over all $\log v$ words/paths. The connection between our path solver and Finite State-Mahcines, provides more reasoning on the runtime complexity, as well as describes the space complexity as also being $O(e \cdot \log v)$, where e is equivalent to w.

To convert the CFLR Strict Dominance Algorithm for inter-procedural analysis, a Dyck-Language can be used to augment the regex for calls and returns. Augmenting the regex with balanced parentheses transforms our required path solver into a Pushdown Automata, since Dyck-Languages cannot be solved by Finite State-Machines as they are not Regular. While this does increase the runtime complexity, the approach was trivial. Take an existing analysis and refine the language.

Creating an analysis can also be informed by the memory requirements of the solution. If memory is bounded/finite, a Finite State-Machine solution may be the most applicable. Otherwise, unbounded memory and the existence of infinite length-words will require at minimum a Pushdown Automata, as in the case of inter-procedural analysis. Working "backwards" from the space requirements of the analysis can provide a framework for constructing CFLR algorithms.

## 6 RELATED WORK

### 6.1 Automatas and the Cubic Bottleneck

The Cubic Bottleneck for program analysis has been a well-studied issue, and was one of the topics discussed in Rep's original CFLR paper [9]. Melski and Reps [8], and Kodumal and Aiken [4], study this issue from an conversion perspective between two analysis techniques, Set-Constraints and CFLR. Heintze and McAllester [3] approach the topic from a similar angle as this paper, using automata theory. They argue that almost all analysis questions can be converted into 2-way Non-deterministic Pushdown Automata (2NPDA), and since the late 1960s there has not been a sub-cubic algorithm for computing 2NPDAs.

Visibly Pushdown Languages [2], proposed by Alur and Madhusudan, focus on dividing the automata's alphabet ($\Sigma$) into call symbols, return symbols, and internal symbols. Unless there is a call or return symbol, the stack is not used. This allows usefull properties, such as determining whether a language is included in another for two Visibly Pushdown Automatas (VPAs). However, VPAs do not have a method of improving on the cubic bottleneck.

Recursive State-Machines [1], also proposed by Alur et al., encode memory into the path of the automata rather that in a separate stack. States of a Recursive State-Machine can be state-machines as well as states. The improvements to traditional automata are similar to VPAs, however they crucially provide the possibility for sub-cubic analysis.

### 6.2 Context-Free Language Reachability

Various developments in the field of Context-Free Language Reachability have taken place in the last couple of years. Zhang et al. [11] exploit an equivalence property in bidirectional graphs for faster alias analysis. Ding et al. [10] have formulated a method for composing multiple analyses into a single pass, or dividing a single analysis into multiple pieces that can allow for faster exits. Lei et al. [5, 6] propose methods of optimizing the input graph representing the analysis to speed up the solver.

## 7 CONCLUSION

Over the course of this work, it has been described how traditional Context-Free Language Reachability is limited in the types of analyses it can solve. The work covers how Dominators for Control-Flow Analysis cannot be described by a Dyck-Language without pre-computing the answer before the reachability solver, and how changing the input graph for Dominators would be sidestepping the limitations by simply not using Context-Free Language Reachability. Following the limitations, we define a new class of Context-Free Language Reachability, namely the All-Paths type, and use this new class to propose a Strict Dominance Algorithm using the CFLR framework. The Strict Dominance Algorithm is then analyzed using automata theory to reason about why the runtime complexity might be lower than the Cubic Bottleneck commonly associated with other analyses.

## REFERENCES


[1] Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas Reps, and Mihalis Yannakakis. 2005. Analysis of recursive state machines. *ACM Trans. Program. Lang. Syst.* 27, 4 (July 2005), 786–818. https://doi.org/10.1145/1075382.1075387

[2] Rajeev Alur and P. Madhusudan. 2004. Visibly pushdown languages. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing* (Chicago, IL, USA) *(STOC '04)*. Association for Computing Machinery, New York, NY, USA, 202–211. https://doi.org/10.1145/1007352.1007390

[3] N. Heintze and D. McAllester. 1997. On the cubic bottleneck in subtyping and flow analysis. In *Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science*. 342–351. https://doi.org/10.1109/LICS.1997.614960

[4] John Kodumal and Alex Aiken. 2004. The set constraint/CFL reachability connection in practice. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation* (Washington DC, USA) *(PLDI '04)*. Association for Computing Machinery, New York, NY, USA, 207–218. https://doi.org/10.1145/996841.996867

[5] Yuxiang Lei, Camille Bossut, Yulei Sui, and Qirun Zhang. 2024. Context-Free Language Reachability via Skewed Tabulation. *Proc. ACM Program. Lang.* 8, PLDI, Article 221 (June 2024), 24 pages. https://doi.org/10.1145/3656451

[6] Yuxiang Lei, Yulei Sui, Shin Hwei Tan, and Qirun Zhang. 2023. Recursive State Machine Guided Graph Folding for Context-Free Language Reachability. *Proc. ACM Program. Lang.* 7, PLDI, Article 119 (June 2023), 25 pages. https://doi.org/10.1145/3591233

[7] Thomas Lengauer and Robert Endre Tarjan. 1979. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.* 1, 1 (Jan. 1979), 121–141. https://doi.org/10.1145/357062.357071

[8] David Melski and Thomas Reps. 1997. Interconvertbility of set constraints and context-free language reachability. *SIGPLAN Not.* 32, 12 (Dec. 1997), 74–89. https://doi.org/10.1145/258994.259006

[9] Thomas Reps. 1998. Program analysis via graph reachability1An abbreviated version of this paper appeared as an invited paper in the Proceedings of the 1997 International Symposium on Logic Programming [84].1. *Information and Software Technology* 40, 11 (1998), 701–726. https://doi.org/10.1016/S0950-5849(98)00093-7

[10] Ding Shuo and Qirun Zhang. [n. d.]. Mutual Refinements of Context-Free Language Reachability. *International Static Analysis Symposium* 14284 ([n. d.]). https://par.nsf.gov/biblio/10507107

[11] Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. 2013. Fast algorithms for Dyck-CFL-reachability with applications to alias analysis. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/2491956.2462159