# 《Casper: An Efficient Approach to Call Trace Collection》Review

## Paper Info

《Casper: An Efficient Approach to Call Trace Collection》 POPL 2016

Rongxin Wu, Xiao Xiao, Shing-Chi Cheung, Hongyu Zhang, Charles Zhang

## Major Contributions

This work focuses on an interesting and meaningful problem: call trace collection. In many related fields, including anomalous program behaviours detection, debugging, performance diagnosis and program comprehension.

The conventional approach to collect call traces incurs large space and time overhead, so it can not be applied to the real world code projects. In this paper, the authors propose a new approach aimed at reducing the recording overhead by instrumenting only a small amount of call sites while keeping the capability of recovering the full trace. This is the core insight of this work.

The main contributions of this paper include:

- They propose an LL(1) grammar based framework to study the inference based call trace collection problem. They prove that this problem is equivalent to the vertex cover problem, hence it is a NP-hard problem.
- They design an algorithm to construct the suboptimal solution of the call site instrument, and implement a tool Casper that can collect full call trace in complex function-call enviroments.

The paper has both theoretically study value and practical use value. The way to solve this problem is elegant and the proof is rigorous. However, the observation and the insight are natural, and it is easy to link this problem with the property of LL(1) grammar. The application of this work is quite broad, including performance diagnosis, debug assistance and so on.

## Main Work

The author give a framework to reduce the time and space overhead in call trace collection. They observe that some locations in the full call trace can be omitted, and they can infer the full call trace just based on the partial or logged call traces. This is a smart observation. In this way, the total space can be compressed. They can compress the set of full call traces into the set of partial call traces, and then decompress the later into the former.

The author prove that the grammar to parse call trace model is LL(1) grammar. This is important. Then they find a bijection between the grammar parsing full call trace model and the grammar parsing partial or logged call trace model. This provide a theoretically base of their algorithm, in which they construct the instrument of I supporting the convertion between these two grammars.

They prove that the construction of the optimal instrument is NP-hard, and try to find a suboptimal solution. The details of algorithm involves some program struct and will not be discussed here. They implement the tool Casper and perform some experiments.

To sum up, this work has the rigorous mathematical proofs, but also practical design of algorithms and the tool. The results of the tool demonstrate the perfect performance of the tool and the efficient of their approach. The approach in this paper is meaningful and useful in the future research on other field including debug assistance, performance diagnosis and so on. Because of page limitation, the details of the proofs and implementation are not summarized in this review.

## Some Criticism and Future Work

Call trace collection is a powerful technique in many program analysis problems, but it suffers high space and time overhead. This work makes an important step towards reducing the cost of collecting call traces. The paper is the work of Rongxin Wu, my senior fellow apprentice in Tsinghua University. Although it is well-written and has a meaningful impact on program analysis, it has several limitations. Some important applications are also worth of deeper research.

- In this paper, the author construct an LL(1) grammar to parse call trace model in order to reduce the time and space overhead in the call trace collection. However, more time and space overhead can be reduced if LL(k) grammar is used instead of LL(1) grammar. Because LL(k) grammar can omit more unnecesseary information while keeping the call trace recovery available, it must perform better than LL(1) grammar.
- This work can apply to performance diagnosis. Based on the full call trace collected, we can analysis the program locations in each trace. If the program locations in a specific code block or function occurs in many call traces, we can focus on such a code block or function and try to optimize it. This can help the developers improve the entire performance of the project.