Code. Analyze. Repeat. Incremental and Modular Static Program Analysis*

Poster Abstract†

Isabel Garcia-Contreras
IMDEA Software Institute and
Technical University of Madrid
isabel.garcia@imdea.org

Jose F. Morales
IMDEA Software Institute
josef.morales@imdea.org

Manuel V. Hermenegildo
IMDEA Software Institute and
Technical University of Madrid
manuel.hermenegildo@imdea.org

KEYWORDS

Program Analysis, Abstract Interpretation, Fixpoint Algorithms, Incremental Analysis, Modular Analysis, Static analysis

ACM Reference Format:
Isabel Garcia-Contreras, Jose F. Morales, and Manuel V. Hermenegildo. 2018. Code. Analyze. Repeat. Incremental and Modular Static Program Analysis: Poster Abstract. In *Proceedings of ACM Celebration of Women in Computing (womENcourage'18),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, 1 page.

1 INTRODUCTION

Static program analysis is widely used for automatically inferring program properties such as correctness, robustness, safety, cost, etc. In particular, abstract interpretation is a theory used to reason about the (possibly infinite) executions of a program in a finite way, without actually executing the program. Performing such analysis during software development helps in bug detection and reporting, but given the size and complex structure of real-life programs, combining system libraries and third party software, such analysis can be expensive. Triggering a complete reanalysis for each set of changes is often too costly. However, in practice development iterations normally involve small modifications, isolated inside a small number of files or components. This can be taken advantage of to reduce the cost of re-analysis by reusing previous information.

2 STATE OF THE ART

In the context of abstract interpretation cost reductions have been achieved to date at two levels, using different techniques: ***Modular* context-sensitive analysis** [4] obtains global information on the whole program by performing local analyses of its components (modules). It is aimed at reducing memory consumption but can localize the (re)computation of the analysis to the modules affected by changes, achieving some incrementality. **Fine grain context-sensitive *incremental* analysis** [3] identifies, invalidates, and recomputes only those parts of the analysis results that are affected by program changes. These analyses have been shown to achieve very high levels of incrementality, with finer granularity (e.g., program line level), but do not take advantage of the module structure. The combination of these two techniques is not straightforward.

*Research partially funded by Spanish MINECO grant TIN2015-67522-C3-1-R *TRACES*, FPU grant 16/04811, and the Madrid M141047003 *N-GREENS* program.
†The full version is available as [1].

womENcourage'18, October 2018, Belgrade, Serbia
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

3 INCREMENTAL AND MODULAR ANALYSIS ALGORITHM

We propose a framework that analyzes separately modules or partitions of programs, using context-sensitive fixpoint analysis while achieving both inter-modular (coarse-grain) and intra-modular (fine-grain) incrementality. The essential point of the algorithm is that analysis results are represented in a way that makes it possible to partially invalidate the results that are no longer valid, correct, or accurate, while keeping the information that does not need recomputation. The information on source changes is used to decide which parts of the program (modules or procedures) need to be reanalyzed. We solve the problems related to the propagation of the fine-grain change information across module boundaries. We work out the actions needed to recompute the analysis incrementally after multiple additions and deletions across modules in the program. We prove that the analysis result is always correct and it is the best (most accurate) over-approximation of the actual behavior of the program. We have implemented the proposed approach within the Ciao/CiaoPP system [2] and provide experimental data.

4 RESULTS

Our preliminary results from the implementation show promising speedups for medium and large programs. The added finer granularity of the proposed incremental fixpoint algorithm reduces significantly the cost with respect to modular analysis alone The advantages of fine-grain incremental analysis –making the cost ideally proportional to the size of the changes– thus seem to carry over with our algorithm to the modular analysis case. Furthermore, the fine-grained propagation of analysis information of our algorithm improves performance with respect to traditional modular analysis even when analyzing from scratch.

REFERENCES

[1] I. Garcia-Contreras, J. F. Morales, and M. V. Hermenegildo. 2018. *An Approach to Incremental and Modular Context-sensitive Analysis of Logic Programs.* Technical Report CLIP-2/2018.0. The CLIP Lab. https://arxiv.org/abs/1804.01839
[2] M.V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla. 2012. An Overview of Ciao and its Design Philosophy. *TPLP* 12, 1–2 (2012), 219–252.
[3] M. V. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. 2000. Incremental Analysis of Constraint Logic Programs. *ACM TOPLAS* 22, 2 (March 2000), 187–223.
[4] G. Puebla, J. Correas, M. V. Hermenegildo, F. Bueno, M. García de la Banda, K. Marriott, and P. J. Stuckey. 2004. A Generic Framework for Context-Sensitive Analysis of Modular Programs. In *Program Development in Computational Logic.* Number 3049 in LNCS. Springer-Verlag, 234–261.