Towards Incremental and Modular Context-Sensitive Analysis

Isabel Garcia-Contreras
IMDEA Software Institute, Pozuelo de Alarcón, Madrid, Spain and
Universidad Politécnica de Madrid (UPM)
isabel.garcia@imdea.org
🆔 https://orcid.org/0000-0001-6098-3895

José F. Morales
IMDEA Software Institute, Pozuelo de Alarcón, Madrid, Spain
josef.morales@imdea.org
🆔 https://orcid.org/0000-0001-9782-8135

Manuel V. Hermenegildo
IMDEA Software Institute, Pozuelo de Alarcón, Madrid, Spain and
Universidad Politécnica de Madrid (UPM)
manuel.hermenegildo@imdea.org
🆔 https://orcid.org/0000-0002-7583-323X

──── **Abstract** ────

This is an extended abstract of [1].

Static program analysis (generally based on computing fixpoints using the technique of abstract interpretation) is widely used for automatically inferring program properties such as correctness, robustness, safety, cost, etc. Performing such analysis interactively during software development allows early detection and reporting of bugs, such as, e.g., assertion violations, back to the programmer. This can be done as the program is being edited by (re-)running the analysis in the background each time a set of changes is made, e.g., when a file is saved, or a commit made in the version control system. However, real-life programs are large, and, typically, have a complex structure combining a good number of modules with other modules in system libraries. Global analysis of such large code bases can be very expensive, and more so if context-sensitivity is supported for precision. This renders triggering a complete reanalysis for each set of changes too costly. A key observation, however, is that in practice each development or transformation iteration is normally formed by relatively small modifications, which in turn are isolated inside a small number of modules. This property can be taken advantage of in order to reduce the cost of re-analysis by reusing

as much information as possible from previous analyses. Such cost reductions have been achieved to date at two different levels, using relatively different techniques:

- *Modular* context-sensitive analyses obtain global information on the whole program by performing local analyses one module at a time. They are typically aimed at reducing memory consumption (working set size) but can also localize the (re)computation of the analysis to the modules affected by changes, achieving some coarse-grained incrementality.
- Context-sensitive (non-modular) *incremental* analyses identify, invalidate, and recompute only those parts of the analysis results that are affected by fine-grain program changes. These analyses have been shown to achieve very high levels of incrementality, at fine levels of granularity (e.g., program line level).

The problem that we address is that while, as mentioned before, large programs are typically highly modular, the context-sensitive, fine-grained incremental analysis techniques presented to date are not easily applicable to the modular setting: The flow of analysis information through the module interfaces requires iterations, since the analysis of a module depends on the analysis of other modules in complex ways, through several paths to different versions of the procedures.

In order to bridge the gap we propose a framework that analyzes separately the modules of a modular program, using context-sensitive fixpoint analysis while achieving both inter-modular (coarse-grain) and intra-modular (fine-grain) incrementality. The proposed analysis algorithm assumes a setting in which we analyze successive "snapshots" of modular programs, i.e., at each analysis iteration, a snapshot of the sources is taken and used to perform the next analysis. Each time an analysis is started, the modules will be analyzed independently and incrementally (possibly several times) until a global fixpoint is reached. The algorithm is designed to work with any partition of the sources. The essential point of the algorithm is that analysis results are represented in a way that allows to partially invalidate the results that are no longer valid, correct, or accurate, while keeping the information that does not need recomputation. The information of source changes is used to invalidate (if necessary), and then decide which parts of the program (modules or predicates) need to be reanalyzed. We solve the problems related to the propagation of the fine-grain change information across module boundaries. We also work out the actions that need to be performed in order to recompute the analysis fixpoint incrementally after multiple additions and deletions across modules in the program. Finally, 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]. Our preliminary results show promising speedups for medium and large programs. The added finer granularity (which allows reusing analysis information both at the intra- and inter-modular levels) 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. An Approach to Incremental and Modular Context-sensitive Analysis of Logic Programs. Technical Report CLIP-2/2018.0, The CLIP Lab, April 2018. URL: `https://arxiv.org/abs/1804.01839`.

2   M.V. Hermenegildo, F. Bueno, M. Carro, P. López, E. Mera, J.F. Morales, and G. Puebla. An Overview of Ciao and its Design Philosophy. *TPLP*, 12(1–2):219–252, 2012.