Ranged Program Analysis: A Parallel Divide-and-Conquer Approach for Software Verification

Jan Haltermann[1], Marie-Christine Jakobs[2], Cedric Richter[3], Heike Wehrheim[4]

Abstract: Scaling software verification is a challenging problem, as finding a suitable method to distribute the work among multiple verifiers is no trivial question. We propose *ranged program analysis*, a divide-and-conquer approach for software verification, which uses ranges to divide the task of program verification. Thereby, each range can be analyzed by a different verification tool, allowing us to analyze program ranges in parallel and to combine the strengths of different verifiers to solve a task. Our evaluation shows that the effectiveness of path-based analyses like symbolic execution can be increased and that a combination of different analyses can solve tasks none of them can solve standalone.

Keywords: Software Verification; Ranged Program Analysis; Parallel Configurable Program Analysis; Program Instrumentation

There has been enormous progress in automatic software verification in recent years and today various verification tools exist. Still, none of them is superior to all others. Hence cooperatively combining them is one way to enhance the verification performance. In case the verification task should be divided among multiple analyses, a strategy to split the program is needed. We present *ranged program analysis*, a technique for parallelizing software verification, that generalizes the idea of ranged symbolic execution [SK12].

The key idea of ranged program analysis is to divide a program into multiple, disjoint program parts (so-called *program ranges*), which can be analyzed in parallel. Using an ordering on program paths, a range can be represented as interval $[\pi_1, \pi_2]$ for two paths π_1 and π_2. Using the smallest path $\pi_\perp$ and the largest path $\pi^\top$, a program can be divided into numerous ranges, such that each program path is included in at least one range. The concept of ranged program analysis, as illustrated in Figure 1, is based on this idea: The verification task is given to a Splitter, which generates the required number of program ranges. Four splitters are designed and evaluated in [Ha23a], based on either limiting the number of loop unrollings per range or using randomly generated paths. Each generated range is given to a Ranged Analysis, which computes a partial result for the given range. The Joiner collects all partial results computed and combines them into a final result for the full task.

[1] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany jan.haltermann@uol.de
[2] LMU Munich, Munich, Germany m.jakobs@lmu.de
[3] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germanycedric.richter@uol.de
[4] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germanyheike.wehrheim@uol.de

doi:10.18420/sw2024_52

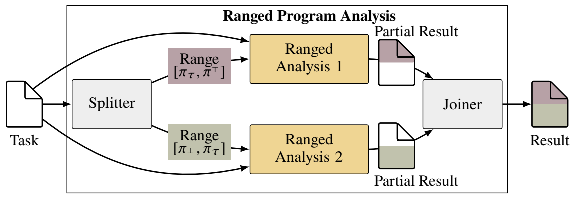

Fig. 1: Conceptual overview of ranged program analysis for two ranged analyses

In [Ha23a], we realize ranged analysis for off-the-shelf analyses that are defined within the configurable program analysis (CPA) framework. We formally define a *range reduction* analysis, that is executed in parallel with the CPA-based program analysis and ensures that only paths within the range are explored and thus analyzed. To generalize ranged program analysis for arbitrary off-the-shelf analyses, we develop a method based on program instrumentation in [Ha23b]. By default, program analyses cannot process a range as additional input. As we aim for black-box cooperation of analyses, the range must be encoded directly into the program given as input to the analysis. By adding additional statements to the program, we obtain a so-called *range program*, restricting the paths that need to be analyzed to those contained in the range.

We implement splitter, joiner and program instrumentation as standalone components and realize ranged program analysis by making use of CoVeriTeam. Our evaluation shows that ranged program analysis can enhance the overall effectiveness of path-based analyses like symbolic execution. Moreover, it enables each of the evaluated analysis combinations to solve tasks that the individual analyzer cannot solve without ranged program analysis. We show that restricting a CPA-based analysis either using range reduction or using program instrumentation leads to a comparable performance. Finally, ranged program analysis can be utilized to increase the efficiency with respect to violation detection in complex tasks.

Data Availability: Our implementation of ranged program analysis is open source and available online. Our artifacts (each evaluated as available and reusable) containing the implementation and all experimental data are archived at Zenodo (https://doi.org/10.5281/zenodo.8398989 for [Ha23a] and https://doi.org/10.5281/zenodo.8096028 for [Ha23b]).

Bibliography

[Ha23a] Haltermann, Jan; Jakobs, Marie-Christine; Richter, Cedric; Wehrheim, Heike: Parallel Program Analysis via Range Splitting. In: Proc. FASE. volume 13991 of LNCS. Springer, pp. 195–219, 2023.

[Ha23b] Haltermann, Jan; Jakobs, Marie-Christine; Richter, Cedric; Wehrheim, Heike: Ranged Program Analysis via Instrumentation. In: Proc. SEFM. volume 14323 of LNCS. Springer, pp. 145–164, 2023.

[SK12] Siddiqui, Junaid Haroon; Khurshid, Sarfraz: Scaling symbolic execution using ranged analysis. In: Proc. OOPSLA. ACM, pp. 523–536, 2012.